formulas Documentation

Release 1.3.4

Vincenzo Arcidiacono

Mar 11, 2026

TABLE OF CONTENTS

1 What is formulas? 3

2 Installation 5
- 2.1 Install extras . . . 5
- 2.2 Development version . . . 5

3 CLI Quickstart 7
- 3.1 What is formulas? . . . 7
- 3.2 Installation . . . 8
  - 3.2.1 Install extras . . . 8
  - 3.2.2 Development version . . . 8
- 3.3 CLI Quickstart . . . 8
- 3.4 Basic Examples . . . 9
  - 3.4.1 Parsing formula . . . 9
  - 3.4.2 Excel workbook . . . 11
    - 3.4.2.1 JSON export/import . . . 15
  - 3.4.3 Custom functions . . . 16
- 3.5 Advanced Examples . . . 16
  - 3.5.1 Minimal Flask integration . . . 16
  - 3.5.2 Batch automation . . . 16
  - 3.5.3 ETL transformer . . . 17
- 3.6 Excel Function Coverage . . . 18
- 3.7 Next moves . . . 18
- 3.8 Contributing to formulas . . . 18
  - 3.8.1 Clone the repository . . . 18
  - 3.8.2 How to implement a new function . . . 19
  - 3.8.3 How to open a pull request . . . 19
- 3.9 Donate . . . 20
- 3.10 Supported by . . . 20
- 3.11 API Reference . . . 20
  - 3.11.1 parser . . . 20
    - 3.11.1.1 Parser . . . 21
  - 3.11.2 builder . . . 22
    - 3.11.2.1 AstBuilder . . . 22
  - 3.11.3 errors . . . 23
    - 3.11.3.1 AnchorRangeName . . . 23
    - 3.11.3.2 BaseError . . . 23
    - 3.11.3.3 BroadcastError . . . 23
    - 3.11.3.4 FormulaError . . . 23
    - 3.11.3.5 FoundError . . . 23

3.11.3.6 FunctionError . . . 23
3.11.3.7 InvalidRangeError . . . 23
3.11.3.8 InvalidRangeName . . . 23
3.11.3.9 ParenthesesError . . . 24
3.11.3.10 RangeValueError . . . 24
3.11.3.11 TokenError . . . 24
3.11.4 tokens . . . 24
3.11.4.1 function . . . 24
3.11.4.2 operand . . . 29
3.11.4.3 operator . . . 47
3.11.4.4 parenthesis . . . 53
3.11.4.5 Token . . . 54
3.11.5 functions . . . 56
3.11.5.1 comp . . . 56
3.11.5.2 date . . . 56
3.11.5.3 eng . . . 59
3.11.5.4 financial . . . 61
3.11.5.5 google . . . 67
3.11.5.6 info . . . 67
3.11.5.7 logic . . . 146
3.11.5.8 look . . . 147
3.11.5.9 math . . . 151
3.11.5.10 operators . . . 154
3.11.5.11 stat . . . 154
3.11.5.12 text . . . 163
3.11.5.13 args2list . . . 167
3.11.5.14 args2vals . . . 167
3.11.5.15 clean_values . . . 167
3.11.5.16 convert2float . . . 167
3.11.5.17 convert_nan . . . 168
3.11.5.18 convert_noshp . . . 168
3.11.5.19 flatten . . . 168
3.11.5.20 get_error . . . 168
3.11.5.21 get_functions . . . 168
3.11.5.22 get_shape . . . 168
3.11.5.23 is_complex . . . 168
3.11.5.24 is_not_empty . . . 168
3.11.5.25 is_number . . . 168
3.11.5.26 not_implemented . . . 168
3.11.5.27 parse_ranges . . . 168
3.11.5.28 raise_errors . . . 168
3.11.5.29 replace_empty . . . 168
3.11.5.30 return_2d_func . . . 169
3.11.5.31 str2complex . . . 169
3.11.5.32 text2num . . . 169
3.11.5.33 to_number . . . 169
3.11.5.34 wrap_func . . . 169
3.11.5.35 wrap_impure_func . . . 169
3.11.5.36 wrap_ranges_func . . . 169
3.11.5.37 wrap_ufunc . . . 169
3.11.5.38 xfilter . . . 169
3.11.5.39 xfilters . . . 169
3.11.5.40 Array . . . 169
3.11.6 ranges . . . 208

3.11.6.1 Ranges . . . 208
3.11.7 cell . . . 210
3.11.7.1 format_output . . . 210
3.11.7.2 wrap_cell_func . . . 210
3.11.7.3 Cell . . . 210
3.11.7.4 CellWrapper . . . 211
3.11.7.5 InvRangesAssembler . . . 212
3.11.7.6 RangesAssembler . . . 212
3.11.7.7 Ref . . . 213
3.11.8 excel . . . 214
3.11.8.1 cycle . . . 214
3.11.8.2 xlreader . . . 214
3.11.8.3 escape_char . . . 216
3.11.8.4 ExcelModel . . . 217
3.11.8.5 XlCircular . . . 220
3.12 Changelog . . . 229
3.12.1 v1.3.4 (2026-03-11) . . . 229
3.12.1.1 Feat . . . 229
3.12.1.2 Fix . . . 229
3.12.1.3 Other . . . 229
3.12.2 v1.3.3 (2025-11-04) . . . 229
3.12.2.1 Feat . . . 229
3.12.2.2 Fix . . . 229
3.12.3 v1.3.2 (2025-10-21) . . . 229
3.12.3.1 Feat . . . 229
3.12.3.2 Fix . . . 230
3.12.4 v1.3.1 (2025-09-15) . . . 230
3.12.4.1 Feat . . . 230
3.12.4.2 Fix . . . 231
3.12.5 v1.3.0 (2025-08-20) . . . 231
3.12.5.1 Feat . . . 231
3.12.5.2 Fix . . . 231
3.12.5.3 Other . . . 232
3.12.6 v1.2.11 (2025-07-28) . . . 232
3.12.6.1 Fix . . . 232
3.12.7 v1.2.10 (2025-05-21) . . . 232
3.12.7.1 Feat . . . 232
3.12.8 v1.2.9 (2025-04-05) . . . 232
3.12.8.1 Feat . . . 232
3.12.8.2 Fix . . . 232
3.12.9 v1.2.8 (2024-07-16) . . . 233
3.12.9.1 Feat . . . 233
3.12.9.2 Fix . . . 233
3.12.10 v1.2.7 (2023-11-14) . . . 233
3.12.10.1 Feat . . . 233
3.12.10.2 Fix . . . 234
3.12.11 v1.2.6 (2022-12-13) . . . 234
3.12.11.1 Fix . . . 234
3.12.12 v1.2.5 (2022-11-07) . . . 234
3.12.12.1 Fix . . . 234
3.12.13 v1.2.4 (2022-07-02) . . . 235
3.12.13.1 Feat . . . 235
3.12.13.2 Fix . . . 235
3.12.14 v1.2.3 (2022-05-10) . . . 235

3.12.14.1 Feat . . . 235
3.12.14.2 Fix . . . 235
3.12.15 v1.2.2 (2022-01-22) . . . 235
3.12.15.1 Fix . . . 235
3.12.16 v1.2.1 (2022-01-21) . . . 235
3.12.16.1 Feat . . . 235
3.12.16.2 Fix . . . 236
3.12.17 v1.2.0 (2021-12-23) . . . 236
3.12.17.1 Feat . . . 236
3.12.17.2 Fix . . . 236
3.12.18 v1.1.1 (2021-10-13) . . . 236
3.12.18.1 Feat . . . 236
3.12.18.2 Fix . . . 236
3.12.19 v1.1.0 (2021-02-16) . . . 237
3.12.19.1 Feat . . . 237
3.12.19.2 Fix . . . 237
3.12.20 v1.0.0 (2020-03-12) . . . 238
3.12.20.1 Feat . . . 238
3.12.20.2 Fix . . . 238
3.12.21 v0.4.0 (2019-08-31) . . . 238
3.12.21.1 Feat . . . 238
3.12.21.2 Fix . . . 239
3.12.22 v0.3.0 (2019-04-24) . . . 239
3.12.22.1 Feat . . . 239
3.12.22.2 Fix . . . 239
3.12.23 v0.2.0 (2018-12-11) . . . 240
3.12.23.1 Feat . . . 240
3.12.23.2 Fix . . . 240
3.12.24 v0.1.4 (2018-10-19) . . . 240
3.12.24.1 Fix . . . 240
3.12.25 v0.1.3 (2018-10-09) . . . 240
3.12.25.1 Feat . . . 240
3.12.25.2 Fix . . . 240
3.12.25.3 Other . . . 240
3.12.26 v0.1.2 (2018-09-12) . . . 241
3.12.26.1 Feat . . . 241
3.12.26.2 Fix . . . 241
3.12.27 v0.1.1 (2018-09-11) . . . 241
3.12.27.1 Feat . . . 241
3.12.27.2 Fix . . . 241
3.12.28 v0.1.0 (2018-07-20) . . . 241
3.12.28.1 Feat . . . 241
3.12.28.2 Fix . . . 241
3.12.29 v0.0.10 (2018-06-05) . . . 241
3.12.29.1 Feat . . . 241
3.12.29.2 Fix . . . 241
3.12.30 v0.0.9 (2018-05-28) . . . 242
3.12.30.1 Feat . . . 242
3.12.30.2 Fix . . . 242
3.12.31 v0.0.8 (2018-05-23) . . . 242
3.12.31.1 Feat . . . 242
3.12.31.2 Fix . . . 242
3.12.31.3 Other . . . 243
3.12.32 v0.0.7 (2017-07-20) . . . 243

3.12.32.1 Feat 243
3.12.32.2 Fix 243
3.12.32.3 Other 244
3.12.33 v0.0.6 (2017-05-31) 244
3.12.33.1 Fix 244
3.12.34 v0.0.5 (2017-05-04) 244
3.12.34.1 Fix 244
3.12.35 v0.0.4 (2017-02-10) 244
3.12.35.1 Fix 244
3.12.36 v0.0.3 (2017-02-09) 244
3.12.36.1 Fix 244
3.12.37 v0.0.2 (2017-02-08) 244
3.12.37.1 Fix 244

4 Indices and tables **245**

Python Module Index **247**

Index **249**

2026-03-11 18:35:00

https://github.com/vinci1it2000/formulas

https://pypi.org/project/formulas/

http://formulas.readthedocs.io/

https://github.com/vinci1it2000/formulas/wiki/

http://github.com/vinci1it2000/formulas/releases/

https://donorbox.org/formulas

excel, formulas, interpreter, compiler, dispatch

- Vincenzo Arcidiacono <vinci1it2000@gmail.com>

EUPL 1.1+

CHAPTER ONE

WHAT IS FORMULAS?

formulas implements an interpreter for Excel formulas, which parses and compile Excel formulas expressions.

Moreover, it compiles Excel workbooks to python and executes without using the Excel COM server. Hence, **Excel is not needed**.

CHAPTER

TWO

INSTALLATION

To install it use (with root privileges):

```
$ pip install formulas
```

Or download the last git version and use (with root privileges):

```
$ python setup.py install
```

2.1 Install extras

Some additional functionality is enabled installing the following extras:

- excel: enables to compile Excel workbooks to python and execute using: `ExcelModel`.
- plot: enables to plot the formula ast and the Excel model.

To install formulas and all extras, do:

```
$ pip install formulas[all]
```

2.2 Development version

To help with the testing and the development of *formulas*, you can install the development version:

```
$ pip install https://github.com/vinci1it2000/formulas/archive/dev.zip
```

CHAPTER

THREE

CLI QUICKSTART

The *formulas* command-line interface works with spreadsheet models and accepts *.xlsx*, *.ods*, and *.json* inputs.

A typical workflow starts by calculating a workbook. You can override input values directly from the command line and request specific cells to be rendered in the output.

```
$ formulas calc test/test_files/excel.xlsx \
    --overwrite "'[excel.xlsx]'!INPUT_A=3" \
    --overwrite "'[excel.xlsx]DATA'!B3=1" \
    --render "'[excel.xlsx]DATA'!C2=result" \
    --output-format json
```

Spreadsheet models can also be converted into a portable JSON representation. This is useful when the model needs to be versioned, inspected, or executed without the original workbook.

```
$ formulas build test/test_files/excel.xlsx \
    --output-file model.json
```

For validation purposes, a workbook can be tested directly from the CLI. The following command runs the tests and prints a short summary.

```
$ formulas test test/test_files/excel.xlsx --summary
```

Finally, a model can be exposed as a lightweight HTTP API, allowing other applications to execute it remotely. The *serve* command requires the optional web dependencies (*pip install formulas[web]*).

```
$ formulas serve test/test_files/excel.xlsx \
    --host 127.0.0.1 \
    --port 5000
```

Each command provides additional options and examples through the built-in help system:

```
$ formulas COMMAND --help
```

3.1 What is formulas?

formulas implements an interpreter for Excel formulas, which parses and compile Excel formulas expressions.

Moreover, it compiles Excel workbooks to python and executes without using the Excel COM server. Hence, **Excel is not needed**.

3.2 Installation

To install it use (with root privileges):

```
$ pip install formulas
```

Or download the last git version and use (with root privileges):

```
$ python setup.py install
```

3.2.1 Install extras

Some additional functionality is enabled installing the following extras:

- excel: enables to compile Excel workbooks to python and execute using: `ExcelModel`.
- plot: enables to plot the formula ast and the Excel model.

To install formulas and all extras, do:

```
$ pip install formulas[all]
```

3.2.2 Development version

To help with the testing and the development of *formulas*, you can install the development version:

```
$ pip install https://github.com/vinci1it2000/formulas/archive/dev.zip
```

3.3 CLI Quickstart

The *formulas* command-line interface works with spreadsheet models and accepts *.xlsx*, *.ods*, and *.json* inputs.

A typical workflow starts by calculating a workbook. You can override input values directly from the command line and request specific cells to be rendered in the output.

```
$ formulas calc test/test_files/excel.xlsx \
    --overwrite "'[excel.xlsx]'!INPUT_A=3" \
    --overwrite "'[excel.xlsx]DATA'!B3=1" \
    --render "'[excel.xlsx]DATA'!C2=result" \
    --output-format json
```

Spreadsheet models can also be converted into a portable JSON representation. This is useful when the model needs to be versioned, inspected, or executed without the original workbook.

```
$ formulas build test/test_files/excel.xlsx \
    --output-file model.json
```

For validation purposes, a workbook can be tested directly from the CLI. The following command runs the tests and prints a short summary.

```
$ formulas test test/test_files/excel.xlsx --summary
```

Finally, a model can be exposed as a lightweight HTTP API, allowing other applications to execute it remotely. The *serve* command requires the optional web dependencies (*pip install formulas[web]*).

```
$ formulas serve test/test_files/excel.xlsx \
    --host 127.0.0.1 \
    --port 5000
```

Each command provides additional options and examples through the built-in help system:

```
$ formulas COMMAND --help
```

3.4 Basic Examples

The following sections will show how to:

- parse a Excel formulas;
- load, compile, and execute a Excel workbook;
- extract a sub-model from a Excel workbook;
- add a custom function.

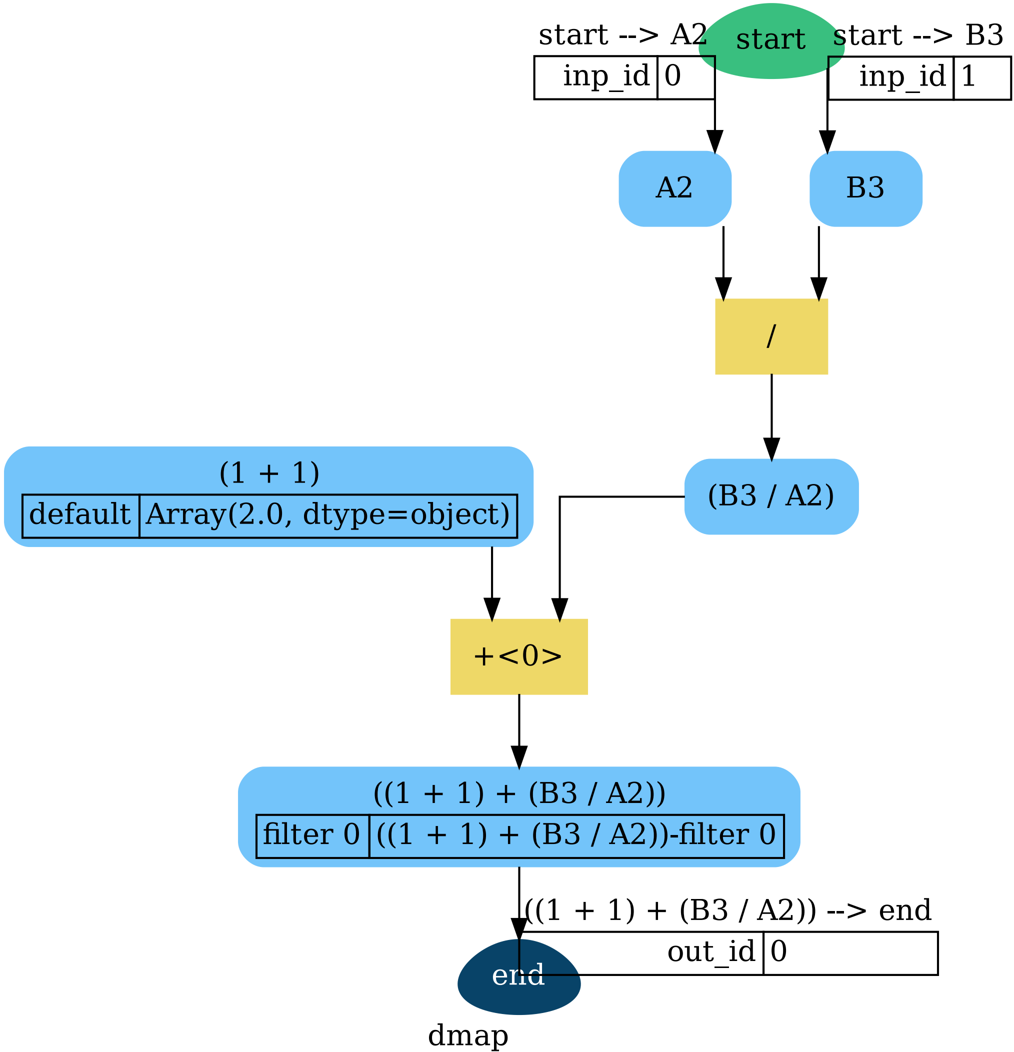
3.4.1 Parsing formula

An example how to parse and execute an Excel formula is the following:

```
>>> import formulas
>>> func = formulas.Parser().ast('=(1 + 1) + B3 / A2')[1].compile()
```

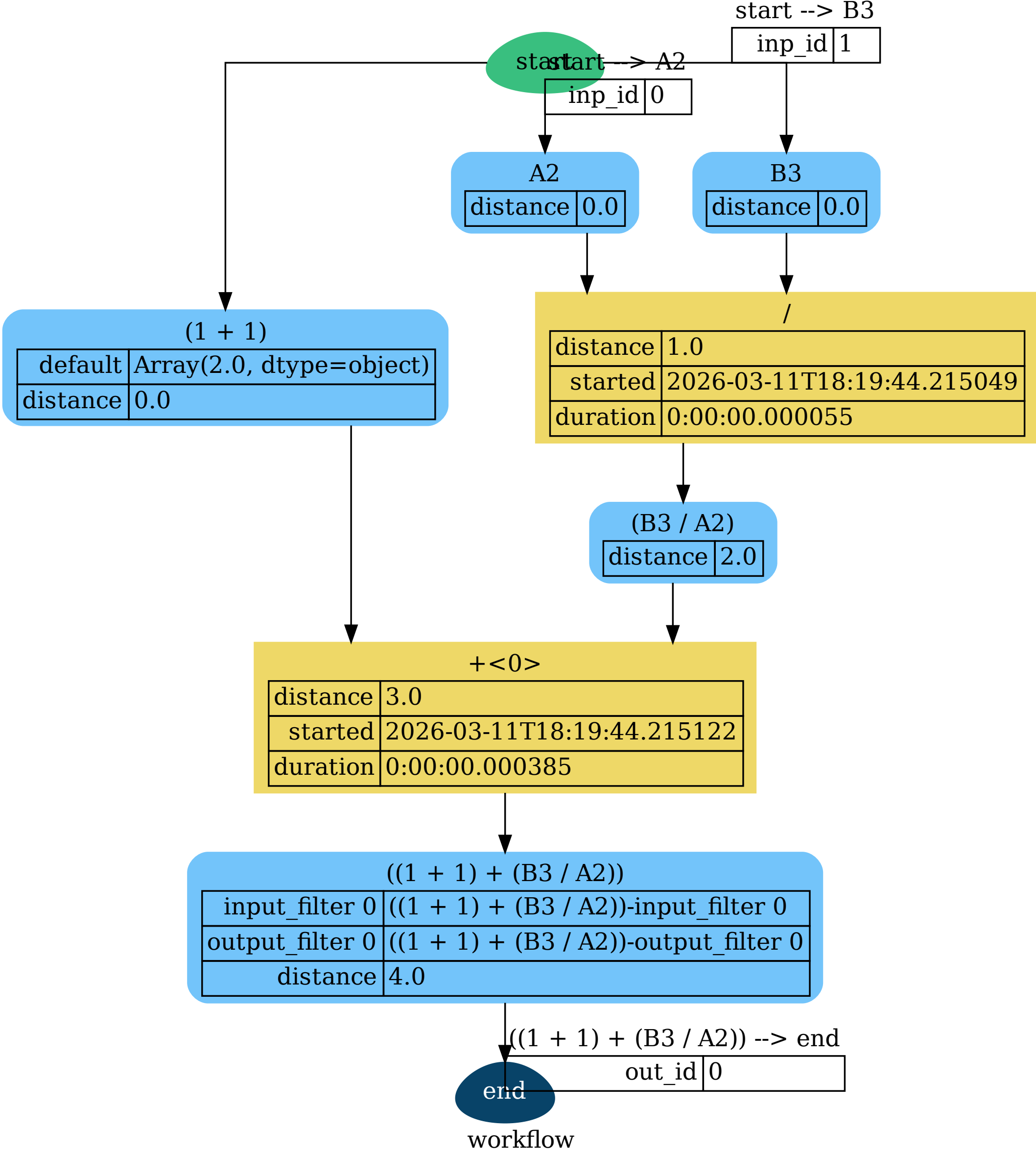
To visualize formula model and get the input order you can do the following:

```
>>> list(func.inputs)
['A2', 'B3']
>>> func.plot(view=False)  # Set view=True to plot in the default browser.
SiteMap({=((1 + 1) + (B3 / A2)): SiteMap({})})
```



Finally to execute the formula and plot the workflow:

```
>>> func(1, 5)
Array(7.0, dtype=object)
>>> func.plot(workflow=True, view=False)  # Set view=True to plot in the default browser.
SiteMap({=((1 + 1) + (B3 / A2)): SiteMap({})})
```



3.4.2 Excel workbook

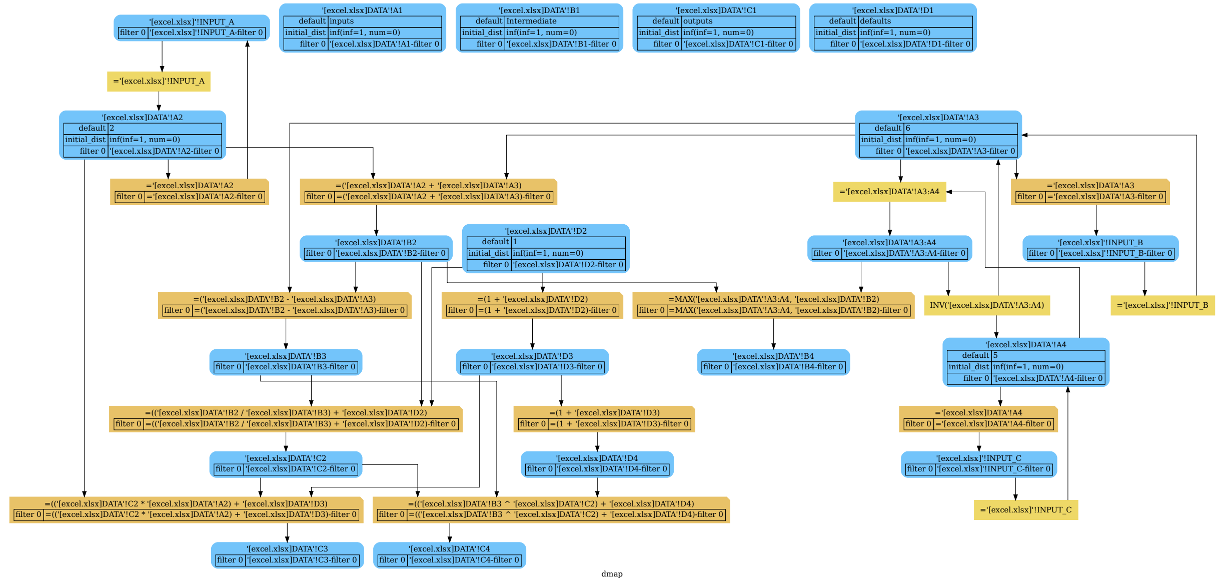
An example how to load, calculate, and write an Excel workbook is the following:

```
>>> import formulas
>>> fpath, dir_output = 'excel.xlsx', 'output'
>>> xl_model = formulas.ExcelModel().loads(fpath).finish()
>>> xl_model.calculate()
Solution(...)
>>> xl_model.write(dirpath=dir_output)
{'EXCEL.XLSX': {Book: <openpyxl.workbook.workbook.Workbook ...>}}
```

> **Tip**
>
> If you have or could have **circular references**, add *circular=True* to *finish* method.

To plot the dependency graph that depict relationships between Excel cells:

```
>>> dsp = xl_model.dsp
>>> dsp.plot(view=False)  # Set view=True to plot in the default browser.
SiteMap({ExcelModel: SiteMap(...)})
```



To overwrite the default inputs that are defined by the excel file or to impose some value to a specific cell:

```
>>> xl_model.calculate(
...     inputs={
...         "'[excel.xlsx]'!INPUT_A": 3,  # To overwrite the default value.
...         "'[excel.xlsx]DATA'!B3": 1  # To impose a value to B3 cell.
...     },
...     outputs=[
...        "'[excel.xlsx]DATA'!C2", "'[excel.xlsx]DATA'!C4"
...     ] # To define the outputs that you want to calculate.
... )
 Solution({"'[excel.xlsx]'!INPUT_A": <Ranges>('[excel.xlsx]DATA'!A2)=[[3]],
           "'[excel.xlsx]DATA'!B3": <Ranges>('[excel.xlsx]DATA'!B3)=[[1]],
           "'[excel.xlsx]DATA'!A2": <Ranges>('[excel.xlsx]DATA'!A2)=[[3]],
           "'[excel.xlsx]DATA'!A3": <Ranges>('[excel.xlsx]DATA'!A3)=[[6]],
           "'[excel.xlsx]DATA'!A4": <Ranges>('[excel.xlsx]DATA'!A4)=[[5]],
           "'[excel.xlsx]DATA'!D2": <Ranges>('[excel.xlsx]DATA'!D2)=[[1]],
           "'[excel.xlsx]'!INPUT_B": <Ranges>('[excel.xlsx]DATA'!A3)=[[6]],
           "'[excel.xlsx]'!INPUT_C": <Ranges>('[excel.xlsx]DATA'!A4)=[[5]],
           "'[excel.xlsx]DATA'!A3:A4": <Ranges>('[excel.xlsx]DATA'!A3:A4)=[[6] [5]],
           "'[excel.xlsx]DATA'!B2": <Ranges>('[excel.xlsx]DATA'!B2)=[[9.0]],
           "'[excel.xlsx]DATA'!D3": <Ranges>('[excel.xlsx]DATA'!D3)=[[2.0]],
           "'[excel.xlsx]DATA'!C2": <Ranges>('[excel.xlsx]DATA'!C2)=[[10.0]],
```

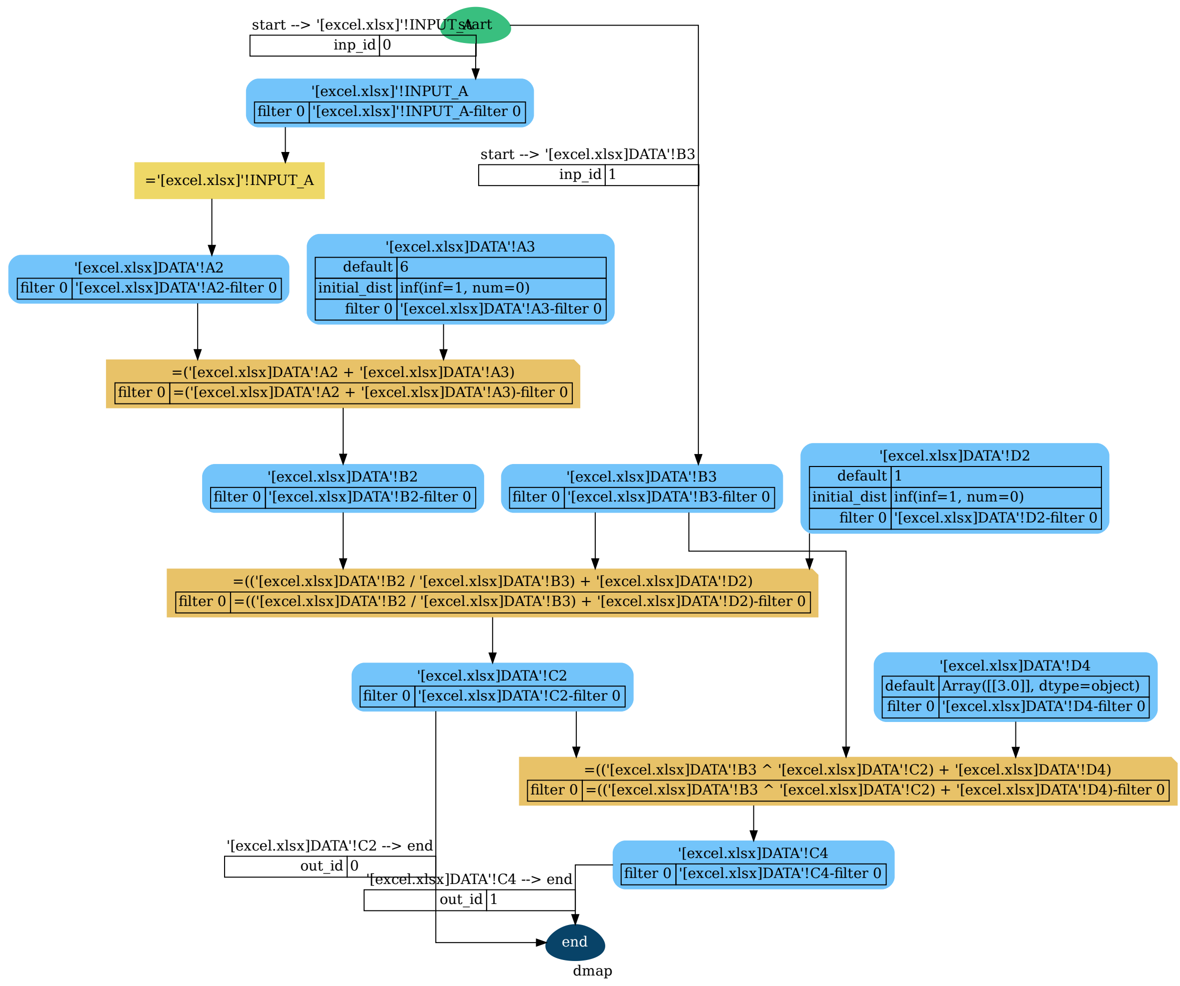
(continues on next page)

(continued from previous page)

```
        "'[excel.xlsx]DATA'!D4": <Ranges>('[excel.xlsx]DATA'!D4)=[[3.0]],
        "'[excel.xlsx]DATA'!C4": <Ranges>('[excel.xlsx]DATA'!C4)=[[4.0]]})
```

To build a single function out of an excel model with fixed inputs and outputs, you can use the *compile* method of the *ExcelModel* that returns a DispatchPipe. This is a function where the inputs and outputs are defined by the data node ids (i.e., cell references).

```
>>> func = xl_model.compile(
...     inputs=[
...         "'[excel.xlsx]'!INPUT_A",  # First argument of the function.
...         "'[excel.xlsx]DATA'!B3"   # Second argument of the function.
...     ], # To define function inputs.
...     outputs=[
...         "'[excel.xlsx]DATA'!C2", "'[excel.xlsx]DATA'!C4"
...     ] # To define function outputs.
... )
>>> func
<schedula.utils.dsp.DispatchPipe object at ...>
>>> [v.value[0, 0] for v in func(3, 1)]  # To retrieve the data.
[10.0, 4.0]
>>> func.plot(view=False)  # Set view=True to plot in the default browser.
SiteMap({ExcelModel: SiteMap(...)})
```



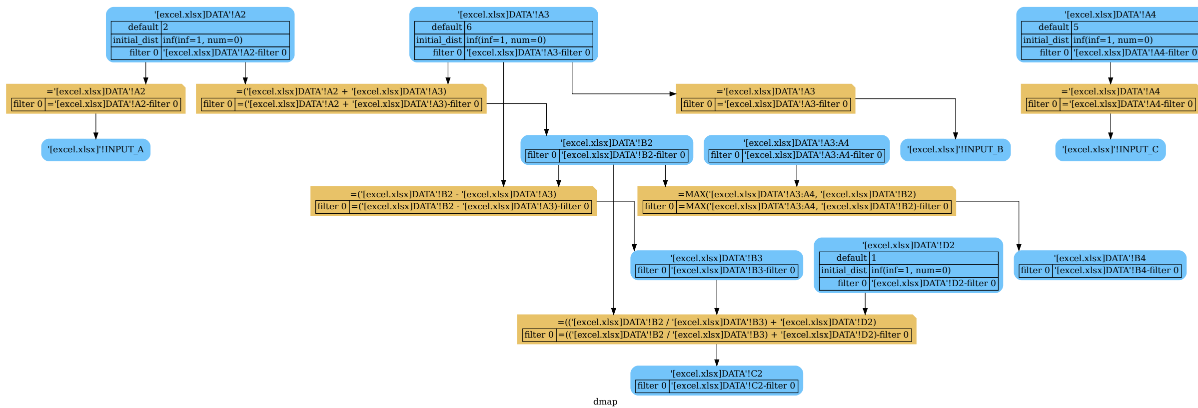
Alternatively, to load a partial excel model from the output cells, you can use the *from_ranges* method of the *ExcelModel*:

```
>>> xl = formulas.ExcelModel().from_ranges(
...     "'[%s]DATA'!C2:D2" % fpath,  # Output range.
...     "'[%s]DATA'!B4" % fpath,  # Output cell.
... )
>>> dsp = xl.dsp
>>> sorted(dsp.data_nodes)
["'[excel.xlsx]'!INPUT_A",
 "'[excel.xlsx]'!INPUT_B",
 "'[excel.xlsx]'!INPUT_C",
 "'[excel.xlsx]DATA'!A2",
 "'[excel.xlsx]DATA'!A3",
 "'[excel.xlsx]DATA'!A3:A4",
 "'[excel.xlsx]DATA'!A4",
 "'[excel.xlsx]DATA'!B2",
 "'[excel.xlsx]DATA'!B3",
 "'[excel.xlsx]DATA'!B4",
 "'[excel.xlsx]DATA'!C2",
```

(continues on next page)

(continued from previous page)

```
"'[excel.xlsx]DATA'!D2"]
```



3.4.2.1 JSON export/import

The *ExcelModel* can be exported/imported to/from a readable JSON format. The reason of this functionality is to have format that can be easily maintained (e.g. using version control programs like *git*). Follows an example on how to export/import to/from JSON an *ExcelModel*:

```
>>> import json
>>> xl_dict = xl_model.to_dict()  # To JSON-able dict.
>>> xl_dict  # Exported format.
{
 "'[excel.xlsx]DATA'!A1": "inputs",
 "'[excel.xlsx]DATA'!B1": "Intermediate",
 "'[excel.xlsx]DATA'!C1": "outputs",
 "'[excel.xlsx]DATA'!D1": "defaults",
 "'[excel.xlsx]DATA'!A2": 2,
 "'[excel.xlsx]DATA'!D2": 1,
 "'[excel.xlsx]DATA'!A3": 6,
 "'[excel.xlsx]DATA'!A4": 5,
 "'[excel.xlsx]DATA'!B2": "=('[excel.xlsx]DATA'!A2 + '[excel.xlsx]DATA'!A3)",
 "'[excel.xlsx]DATA'!C2": "=(('[excel.xlsx]DATA'!B2 / '[excel.xlsx]DATA'!B3) + '[excel.
↪xlsx]DATA'!D2)",
 "'[excel.xlsx]DATA'!B3": "=('[excel.xlsx]DATA'!B2 - '[excel.xlsx]DATA'!A3)",
 "'[excel.xlsx]DATA'!C3": "=(('[excel.xlsx]DATA'!C2 * '[excel.xlsx]DATA'!A2) + '[excel.
↪xlsx]DATA'!D3)",
 "'[excel.xlsx]DATA'!D3": "=(1 + '[excel.xlsx]DATA'!D2)",
 "'[excel.xlsx]DATA'!B4": "=MAX('[excel.xlsx]DATA'!A3:A4, '[excel.xlsx]DATA'!B2)",
 "'[excel.xlsx]DATA'!C4": "=(('[excel.xlsx]DATA'!B3 ^ '[excel.xlsx]DATA'!C2) + '[excel.
↪xlsx]DATA'!D4)",
 "'[excel.xlsx]DATA'!D4": "=(1 + '[excel.xlsx]DATA'!D3)"
}
>>> xl_json = json.dumps(xl_dict, indent=True)  # To JSON.
>>> xl_model = formulas.ExcelModel().from_dict(json.loads(xl_json))  # From JSON.
```

3.4.3 Custom functions

An example how to add a custom function to the formula parser is the following:

```
>>> import formulas
>>> FUNCTIONS = formulas.get_functions()
>>> FUNCTIONS['MYFUNC'] = lambda x, y: 1 + y + x
>>> func = formulas.Parser().ast('=MYFUNC(1, 2)')[1].compile()
>>> func()
4
```

3.5 Advanced Examples

Formulas can also be embedded as a calculation engine inside lightweight applications and automated workflows, without requiring Excel or another spreadsheet GUI.

3.5.1 Minimal Flask integration

This example loads a workbook once, exposes it through a Flask application, and calls the JSON API through a test client.

```
from formulas.app import create_app

app = create_app(files=('test/test_files/excel.xlsx',), circular=False)
client = app.test_client()
response = client.post('/api/calculate', json={
    'inputs': {
        "'[excel.xlsx]'!INPUT_A": 3,
        "'[excel.xlsx]DATA'!B3": 1,
    },
    'renders': ["'[excel.xlsx]DATA'!C2=result"],
})

assert response.status_code == 200
assert response.get_json()['outputs'] == {'result': 10.0}
```

3.5.2 Batch automation

This example creates a temporary batch file and runs *formulas calc* over two scenarios.

```
import json
import subprocess
import sys
import tempfile
from pathlib import Path

with tempfile.TemporaryDirectory() as tmp:
    batch = Path(tmp) / 'batch.json'
    batch.write_text(json.dumps([
        {
            'name': 'base',
            'overwrite': {
                "'[excel.xlsx]'!INPUT_A": 3,
```

(continues on next page)

(continued from previous page)

```
                "'[excel.xlsx]DATA'!B3": 1,
            },
            'renders': ["'[excel.xlsx]DATA'!C2=result"],
        },
        {
            'name': 'stress',
            'overwrite': {
                "'[excel.xlsx]'!INPUT_A": 4,
                "'[excel.xlsx]DATA'!B3": 1,
            },
            'renders': ["'[excel.xlsx]DATA'!C2=result"],
        },
    ], indent=2))

    result = subprocess.run([
        sys.executable, '-m', 'formulas.cli', 'calc',
        'test/test_files/excel.xlsx',
        '--batch', str(batch),
        '--processes', '2',
        '--output-format', 'json',
        '--output-dir', tmp,
    ], capture_output=True, text=True, check=False)

    assert result.returncode == 0, result.stderr
    summary = json.loads(result.stdout)
    assert [item['name'] for item in summary] == ['base', 'stress']
```

3.5.3 ETL transformer

This example treats a workbook as a transformation step over structured input records.

```
import formulas

model = formulas.ExcelModel().loads('test/test_files/excel.xlsx').finish()
func = model.compile(
    inputs=["'[excel.xlsx]'!INPUT_A", "'[excel.xlsx]DATA'!B3"],
    outputs=["'[excel.xlsx]DATA'!C2"],
)
records = [
    {'id': 'row-1', 'input_a': 3, 'b3': 1},
    {'id': 'row-2', 'input_a': 4, 'b3': 1},
]
results = []

for record in records:
    result, = func(record['input_a'], record['b3'])
    results.append({
        'id': record['id'],
        'result': result.value[0, 0],
    })

assert results == [
```

(continues on next page)

(continued from previous page)

```
    {'id': 'row-1', 'result': 10.0},
    {'id': 'row-2', 'result': 11.0},
]
```

3.6 Excel Function Coverage

The current Excel function coverage is tracked in the test workbook *test/test_files/test.xlsx*, sheet *COVERAGE*. The table below summarizes the current implementation status by category.

Category	Implemented	Total	Coverage
AUTOMATION	0	3	0.0%
COMPATIBILITY	40	40	100.0%
CUBE	0	7	0.0%
DATABASE	0	12	0.0%
DATE & TIME	25	25	100.0%
ENGINEERING	54	54	100.0%
FINANCIAL	55	55	100.0%
INFORMATION	16	22	72.7%
LOGICAL	19	19	100.0%
LOOKUP	33	40	82.5%
MATH & TRIG	71	80	88.8%
STATISTICAL	111	111	100.0%
TEXT	44	50	88.0%
WEB	0	3	0.0%
OPERATORS	15	15	100.0%
TOTAL	483	536	90.1%

Overall coverage is currently 483 out of 536 functions (90.1%).

3.7 Next moves

Things yet to do: implement the missing Excel formulas.

3.8 Contributing to formulas

If you want to contribute to **formulas** and make it better, your help is very welcome. The contribution should be sent by a *pull request*. Next sections will explain how to implement and submit a new excel function:

- clone the repository
- implement a new function/functionality
- open a pull request

3.8.1 Clone the repository

The first step to contribute to **formulas** is to clone the repository:

- Create a personal fork of the formulas repository on Github.
- Clone the fork on your local machine. Your remote repo on Github is called `origin`.

- Add the original repository as a remote called `upstream`, to maintain updated your fork.
- If you created your fork a while ago be sure to pull `upstream` changes into your local repository.
- Create a new branch to work on! Branch from `dev`.

3.8.2 How to implement a new function

Before coding, study the Excel function that you want to implement. If there is something similar implemented in **formulas**, try to get inspired by the implemented code (I mean, not reinvent the wheel) and to use `numpy`. Follow the code style of the project, including indentation. Add or change the documentation as needed. Make sure that you have implemented the **full function syntax**, including the array syntax.

Test cases are very important. This library uses a data-driven testing approach. To implement a new function I recommend the test-driven development cycle. Hence, when you implement a new function, you should write new test cases in `test_cell/TestCell.test_output` suite to execute in the *cycle loop*. When you think that the code is ready, add new raw test in `test/test_files/test.xlsx` (please follow the standard used for other functions) and run the `test_excel/TestExcelModel.test_excel_model`. This requires more time but is needed to test the **array syntax** and to check if the Excel documentation respects the reality.

When all test cases are ok (`python setup.py test`), open a pull request.

Do do list:

- Study the excel function syntax and behaviour when used as array formula.
- Check if there is something similar implemented in formulas.
- Implement/fix your feature, comment your code.
- Write/adapt tests and run them!

Tip

Excel functions are categorized by their functionality. If you are implementing a new functionality group, add a new module in `formula/function` and in `formula.function.SUBMODULES` and a new worksheet in `test/test_files/test.xlsx` (please respect the format).

Note

A pull request without new test case will not be taken into consideration.

3.8.3 How to open a pull request

Well done! Your contribution is ready to be submitted:

- Squash your commits into a single commit with git's interactive rebase. Create a new branch if necessary. Always write your commit messages in the present tense. Your commit message should describe what the commit, when applied, does to the code – not what you did to the code.
- Push your branch to your fork on Github (i.e., `git push origin dev`).
- From your fork open a *pull request* in the correct branch. Target the project's `dev` branch!
- Once the *pull request* is approved and merged you can pull the changes from `upstream` to your local repo and delete your extra branch(es).

3.9 Donate

If you want to support the **formulas** development please donate and add your excel function preferences. The selection of the functions to be implemented is done considering the cumulative donation amount per function collected by the campaign.

Note

The cumulative donation amount per function is calculated as the example:

• Function	Donator 1 150€	Donator 2 120€	Donator 3 50€	• TOT	• Implementation order
SUM	50€	40€	25€	125€	1st
SIN	50€		25€	75€	3rd
TAN	50€	40€		90€	2nd
COS		40€		40€	4th

3.10 Supported by

3.11 API Reference

The core of the library is composed from the following modules: It contains a comprehensive list of all modules and classes within formulas.

Modules:

parser	It provides formula parser class.
builder	It provides AstBuilder class.
errors	Defines the formulas exception.
tokens	It provides tokens needed to parse the Excel formulas.
functions	It provides functions implementations to compile the Excel functions.
ranges	It provides Ranges class.
cell	It provides Cell class.
excel	It provides Excel model class.

3.11.1 parser

It provides formula parser class.

Classes

Parser

3.11.1.1 Parser

class Parser(*is_cell=False*)

Methods

__init__
ast
is_formula

__init__

Parser.**__init__**(*is_cell=False*)

ast

Parser.**ast**(*expression*, *context=None*)

is_formula

Parser.**is_formula**(*value*)

__init__(*is_cell=False*)

Attributes

filters
formula_check

filters

```
Parser.filters = [<class 'formulas.tokens.operand.Error'>, <class
'formulas.tokens.operand.String'>, <class 'formulas.tokens.operand.Number'>, <class
'formulas.tokens.function.Lambda'>, <class 'formulas.tokens.operand.Range'>, <class
'formulas.tokens.operator.OperatorToken'>, <class
'formulas.tokens.operator.Separator'>, <class 'formulas.tokens.function.Function'>,
<class 'formulas.tokens.function.Array'>, <class
'formulas.tokens.parenthesis.Parenthesis'>, <class
'formulas.tokens.operator.Intersect'>]
```

formula_check

```
Parser.formula_check = regex.Regex('\n
(?P<array>^\\s*{\\s*=\\s*(?P<name>\\S.*)\\s*}\\s*$)\n |\n
(?P<value>^\\s*=\\s*(?P<name>\\S.*))\n ', flags=regex.S | regex.I | regex.X |
regex.V0)
```

3.11.2 builder

It provides AstBuilder class.

Classes

AstBuilder

3.11.2.1 AstBuilder

class AstBuilder(*dsp=None*, *nodes=None*, *match=None*)

Methods

__init__
append
compile
finish
get_node_id
pop

__init__

AstBuilder.**__init__**(*dsp=None*, *nodes=None*, *match=None*)

append

AstBuilder.**append**(*token*)

compile

AstBuilder.**compile**(*references=None*, *context=None*, ***inputs*)

finish

AstBuilder.**finish**()

get_node_id

AstBuilder.**get_node_id**(*token*)

pop

AstBuilder.**pop**()

__init__(*dsp=None*, *nodes=None*, *match=None*)

compile_class

alias of DispatchPipe

3.11.3 errors

Defines the formulas exception.

Exceptions

AnchorRangeName
BaseError
BroadcastError
FormulaError
FoundError
FunctionError
InvalidRangeError
InvalidRangeName
ParenthesesError
RangeValueError
TokenError

3.11.3.1 AnchorRangeName

exception AnchorRangeName

3.11.3.2 BaseError

exception BaseError(**args*)

3.11.3.3 BroadcastError

exception BroadcastError(**args*)

3.11.3.4 FormulaError

exception FormulaError(**args*)

3.11.3.5 FoundError

exception FoundError(**args*, *err=None*, ***kwargs*)

3.11.3.6 FunctionError

exception FunctionError(**args*)

3.11.3.7 InvalidRangeError

exception InvalidRangeError(**args*)

3.11.3.8 InvalidRangeName

exception InvalidRangeName

3.11.3.9 ParenthesesError

exception ParenthesesError(**args*)

3.11.3.10 RangeValueError

exception RangeValueError(**args*)

3.11.3.11 TokenError

exception TokenError(**args*)

3.11.4 tokens

It provides tokens needed to parse the Excel formulas.

Sub-Modules:

function	It provides Function classes.
operand	It provides Operand classes.
operator	It provides Operator classes.
parenthesis	It provides Parenthesis class.

3.11.4.1 function

It provides Function classes.

Functions

run_function

run_function

run_function(*fun*, **args*, ***kwargs*)

Classes

Array
Function
Lambda
LambdaFunction
LetaFunction

Array

class Array(*s*, *context=None*, *parser=None*)

Methods

__init__
ast
compile
match
process
set_expr
update_input_tokens

__init__

Array.**__init__**(*s*, *context=None*, *parser=None*)

ast

Array.**ast**(*tokens*, *stack*, *builder*, *check_n=<function Array.<lambda>>*)

compile

Array.**compile**()

match

Array.**match**(*s*)

process

Array.**process**(*match*, *context=None*, *parser=None*)

set_expr

Array.**set_expr**(**tokens*)

update_input_tokens

Array.**update_input_tokens**(**tokens*)

__init__(*s*, *context=None*, *parser=None*)

Attributes

name
node_id

name

property Array.**name**

node_id

property Array.**node_id**

Function

class Function(*s*, *context=None*, *parser=None*)

Methods

__init__
ast
compile
match
process
set_expr
update_input_tokens

__init__

Function.**__init__**(*s*, *context=None*, *parser=None*)

ast

Function.**ast**(*tokens*, *stack*, *builder*, *check_n=<function Function.<lambda>>*)

compile

Function.**compile**()

match

Function.**match**(*s*)

process

Function.**process**(*match*, *context=None*, *parser=None*)

set_expr

Function.**set_expr**(**tokens*)

update_input_tokens

Function.**update_input_tokens**(**tokens*)

__init__(*s*, *context=None*, *parser=None*)

Attributes

name
node_id

name

property Function.**name**

node_id

property Function.**node_id**

Lambda

class Lambda(*s*, *context=None*, *parser=None*)

Methods

__init__
ast
compile
match
process
set_expr
update_input_tokens

__init__

Lambda.**__init__**(*s*, *context=None*, *parser=None*)

ast

Lambda.**ast**(*tokens*, *stack*, *builder*, *check_n=<function Function.<lambda>>*)

compile

Lambda.**compile**()

match

Lambda.**match**(*s*)

process

Lambda.**process**(*match*, *context=None*, *parser=None*)

set_expr

Lambda.**set_expr**(**tokens*)

update_input_tokens

Lambda.**update_input_tokens**(**tokens*)

__init__(*s, context=None, parser=None*)

Attributes

name	
node_id	

name

property Lambda.**name**

node_id

property Lambda.**node_id**

LambdaFunction

class LambdaFunction(*_LambdaFunction__func, *args, **kwargs*)

Methods

__init__	

__init__

LambdaFunction.**__init__**()

__init__()

Attributes

args	tuple of arguments to future partial calls
func	function object to use in future partial calls
keywords	dictionary of keyword arguments to future partial calls

args

LambdaFunction.**args**

tuple of arguments to future partial calls

func

`LambdaFunction.func`

function object to use in future partial calls

keywords

`LambdaFunction.keywords`

dictionary of keyword arguments to future partial calls

LetaFunction

`class LetaFunction(_LambdaFunction__func, *args, **kwargs)`

Methods

`__init__`

__init__

`LetaFunction.__init__()`

`__init__()`

Attributes

args	tuple of arguments to future partial calls
func	function object to use in future partial calls
keywords	dictionary of keyword arguments to future partial calls

args

`LetaFunction.args`

tuple of arguments to future partial calls

func

`LetaFunction.func`

function object to use in future partial calls

keywords

`LetaFunction.keywords`

dictionary of keyword arguments to future partial calls

3.11.4.2 operand

It provides Operand classes.

Functions

fast_range2parts
fast_range2parts_v1
fast_range2parts_v2
fast_range2parts_v3
fast_range2parts_v4
fast_range2parts_v5
range2parts

fast_range2parts

fast_range2parts(***kw*)

fast_range2parts_v1

fast_range2parts_v1(*r1*, *c1*, *sheet_id*, *anchor=''*)

fast_range2parts_v2

fast_range2parts_v2(*r1*, *c1*, *r2*, *c2*, *sheet_id*)

fast_range2parts_v3

fast_range2parts_v3(*r1*, *n1*, *sheet_id*, *anchor=''*)

fast_range2parts_v4

fast_range2parts_v4(*r1*, *n1*, *r2*, *n2*, *sheet_id*)

fast_range2parts_v5

fast_range2parts_v5(*ref*, *sheet_id*)

range2parts

range2parts(*outputs*, ***inputs*)

Classes

Empty
Error
Number
Operand
Range
String
XlError

Empty

class **Empty**(*context=None*, *parser=None*)

Methods

__init__
ast
compile
match
process
set_expr
update_input_tokens

__init__

Empty.**__init__**(*context=None*, *parser=None*)

ast

Empty.**ast**(*tokens*, *stack*, *builder*)

compile

static Empty.**compile**()

match

Empty.**match**(*s*)

process

Empty.**process**(*match*, *context=None*, *parser=None*)

set_expr

Empty.**set_expr**(**tokens*)

update_input_tokens

Empty.**update_input_tokens**(**tokens*)

__init__(*context=None*, *parser=None*)

Attributes

name
node_id

name

property Empty.**name**

node_id

property Empty.**node_id**

Error

class Error(*s*, *context=None*, *parser=None*)

Methods

__init__
ast
compile
match
process
set_expr
update_input_tokens

__init__

Error.**__init__**(*s*, *context=None*, *parser=None*)

ast

Error.**ast**(*tokens*, *stack*, *builder*)

compile

Error.**compile**()

match

Error.**match**(*s*)

process

Error.**process**(*match*, *context=None*, *parser=None*)

set_expr

Error.**set_expr**(**tokens*)

update_input_tokens

Error.**update_input_tokens**(**tokens*)

__init__(*s*, *context=None*, *parser=None*)

Attributes

errors
name
node_id

errors

`Error.errors = {'#DIV/0!': #DIV/0!, '#GETTING_DATA': #GETTING_DATA, '#N/A': #N/A, '#NAME?': #NAME?, '#NULL!': #NULL!, '#NUM!': #NUM!, '#REF!': #REF!, '#VALUE!': #VALUE!}`

name

`property Error.name`

node_id

`property Error.node_id`

Number

`class Number(s, context=None, parser=None)`

Methods

__init__
ast
compile
match
process
set_expr
update_input_tokens

__init__

`Number.__init__(s, context=None, parser=None)`

ast

`Number.ast(tokens, stack, builder)`

compile

`Number.compile()`

match

Number.**match**(*s*)

process

Number.**process**(*match*, *context=None*, *parser=None*)

set_expr

Number.**set_expr**(**tokens*)

update_input_tokens

Number.**update_input_tokens**(**tokens*)

__init__(*s*, *context=None*, *parser=None*)

Attributes

name
node_id

name

property Number.**name**

node_id

property Number.**node_id**

Operand

class Operand(*s*, *context=None*, *parser=None*)

Methods

__init__
ast
match
process
set_expr
update_input_tokens

__init__

Operand.**__init__**(*s*, *context=None*, *parser=None*)

ast

Operand.**ast**(*tokens*, *stack*, *builder*)

match

Operand.**match**(*s*)

process

Operand.**process**(*match*, *context=None*, *parser=None*)

set_expr

Operand.**set_expr**(**tokens*)

update_input_tokens

Operand.**update_input_tokens**(**tokens*)

__init__(*s*, *context=None*, *parser=None*)

Attributes

name
node_id

name

property Operand.**name**

node_id

property Operand.**node_id**

Range

class Range(*s*, *context=None*, *parser=None*)

Methods

__init__
ast
compile
match
process
set_expr
update_input_tokens

__init__

Range.**__init__**(*s*, *context=None*, *parser=None*)

ast

Range.**ast**(*tokens*, *stack*, *builder*)

compile

Range.**compile**()

match

Range.**match**(*s*)

process

Range.**process**(*match*, *context=None*, *parser=None*)

set_expr

Range.**set_expr**(**tokens*)

update_input_tokens

Range.**update_input_tokens**(**tokens*)

__init__(*s*, *context=None*, *parser=None*)

Attributes

name
node_id

name

property Range.**name**

node_id

property Range.**node_id**

String

class String(*s*, *context=None*, *parser=None*)

Methods

__init__
ast
compile
match
process
set_expr
update_input_tokens

__init__

String.**__init__**(*s*, *context=None*, *parser=None*)

ast

String.**ast**(*tokens*, *stack*, *builder*)

compile

String.**compile**()

match

String.**match**(*s*)

process

String.**process**(*match*, *context=None*, *parser=None*)

set_expr

String.**set_expr**(**tokens*)

update_input_tokens

String.**update_input_tokens**(**tokens*)

__init__(*s*, *context=None*, *parser=None*)

Attributes

name
node_id

name

property String.**name**

node_id

`property String.node_id`

XlError

`class XlError(*args)`

Methods

__init__	
capitalize	Return a capitalized version of the string.
casefold	Return a version of the string suitable for caseless comparisons.
center	Return a centered string of length width.
count	Return the number of non-overlapping occurrences of substring sub in string S[start:end].
encode	Encode the string using the codec registered for encoding.
endswith	Return True if S ends with the specified suffix, False otherwise.
expandtabs	Return a copy where all tab characters are expanded using spaces.
find	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
format	Return a formatted version of S, using substitutions from args and kwargs.
format_map	Return a formatted version of S, using substitutions from mapping.
index	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
isalnum	Return True if the string is an alpha-numeric string, False otherwise.
isalpha	Return True if the string is an alphabetic string, False otherwise.
isascii	Return True if all characters in the string are ASCII, False otherwise.
isdecimal	Return True if the string is a decimal string, False otherwise.
isdigit	Return True if the string is a digit string, False otherwise.
isidentifier	Return True if the string is a valid Python identifier, False otherwise.
islower	Return True if the string is a lowercase string, False otherwise.
isnumeric	Return True if the string is a numeric string, False otherwise.
isprintable	Return True if the string is printable, False otherwise.
isspace	Return True if the string is a whitespace string, False otherwise.
istitle	Return True if the string is a title-cased string, False otherwise.

continues on next page

Table 35 – continued from previous page

isupper	Return True if the string is an uppercase string, False otherwise.
join	Concatenate any number of strings.
ljust	Return a left-justified string of length width.
lower	Return a copy of the string converted to lowercase.
lstrip	Return a copy of the string with leading whitespace removed.
maketrans	Return a translation table usable for str.translate().
partition	Partition the string into three parts using the given separator.
removeprefix	Return a str with the given prefix string removed if present.
removesuffix	Return a str with the given suffix string removed if present.
replace	Return a copy with all occurrences of substring old replaced by new.
rfind	Return the highest index in S where substring sub is found, such that sub is contained within S[start:end].
rindex	Return the highest index in S where substring sub is found, such that sub is contained within S[start:end].
rjust	Return a right-justified string of length width.
rpartition	Partition the string into three parts using the given separator.
rsplit	Return a list of the substrings in the string, using sep as the separator string.
rstrip	Return a copy of the string with trailing whitespace removed.
split	Return a list of the substrings in the string, using sep as the separator string.
splitlines	Return a list of the lines in the string, breaking at line boundaries.
startswith	Return True if S starts with the specified prefix, False otherwise.
strip	Return a copy of the string with leading and trailing whitespace removed.
swapcase	Convert uppercase characters to lowercase and lowercase characters to uppercase.
title	Return a version of the string where each word is titlecased.
translate	Replace each character in the string using the given translation table.
upper	Return a copy of the string converted to uppercase.
zfill	Pad a numeric string with zeros on the left, to fill a field of the given width.

__init__

`XlError.__init__(*args)`

capitalize

XlError.**capitalize**()

Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower case.

casefold

XlError.**casefold**()

Return a version of the string suitable for caseless comparisons.

center

XlError.**center**(*width*, *fillchar=' '*, */ (Positional-only parameter separator (PEP 570))*)

Return a centered string of length width.

Padding is done using the specified fill character (default is a space).

count

XlError.**count**(*sub*[, *start*[, *end*]]) → int

Return the number of non-overlapping occurrences of substring sub in string S[start:end]. Optional arguments start and end are interpreted as in slice notation.

encode

XlError.**encode**(*encoding='utf-8'*, *errors='strict'*)

Encode the string using the codec registered for encoding.

encoding
The encoding in which to encode the string.

errors
The error handling scheme to use for encoding errors. The default is 'strict' meaning that encoding errors raise a UnicodeEncodeError. Other possible values are 'ignore', 'replace' and 'xmlcharrefreplace' as well as any other name registered with codecs.register_error that can handle UnicodeEncodeErrors.

endswith

XlError.**endswith**(*suffix*[, *start*[, *end*]]) → bool

Return True if S ends with the specified suffix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. suffix can also be a tuple of strings to try.

expandtabs

XlError.**expandtabs**(*tabsize=8*)

Return a copy where all tab characters are expanded using spaces.

If tabsize is not given, a tab size of 8 characters is assumed.

find

XlError.**find**(*sub*[, *start*[, *end*]]) → int

Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

format

XlError.**format**(**args*, ***kwargs*) → str

Return a formatted version of S, using substitutions from args and kwargs. The substitutions are identified by braces ('{' and '}').

format_map

XlError.**format_map**(*mapping*) → str

Return a formatted version of S, using substitutions from mapping. The substitutions are identified by braces ('{' and '}').

index

XlError.**index**(*sub*[, *start*[, *end*]]) → int

Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Raises ValueError when the substring is not found.

isalnum

XlError.**isalnum**()

Return True if the string is an alpha-numeric string, False otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string.

isalpha

XlError.**isalpha**()

Return True if the string is an alphabetic string, False otherwise.

A string is alphabetic if all characters in the string are alphabetic and there is at least one character in the string.

isascii

XlError.**isascii**()

Return True if all characters in the string are ASCII, False otherwise.

ASCII characters have code points in the range U+0000-U+007F. Empty string is ASCII too.

isdecimal

`XlError.isdecimal()`

Return True if the string is a decimal string, False otherwise.

A string is a decimal string if all characters in the string are decimal and there is at least one character in the string.

isdigit

`XlError.isdigit()`

Return True if the string is a digit string, False otherwise.

A string is a digit string if all characters in the string are digits and there is at least one character in the string.

isidentifier

`XlError.isidentifier()`

Return True if the string is a valid Python identifier, False otherwise.

Call keyword.iskeyword(s) to test whether string s is a reserved identifier, such as "def" or "class".

islower

`XlError.islower()`

Return True if the string is a lowercase string, False otherwise.

A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

isnumeric

`XlError.isnumeric()`

Return True if the string is a numeric string, False otherwise.

A string is numeric if all characters in the string are numeric and there is at least one character in the string.

isprintable

`XlError.isprintable()`

Return True if the string is printable, False otherwise.

A string is printable if all of its characters are considered printable in repr() or if it is empty.

isspace

`XlError.isspace()`

Return True if the string is a whitespace string, False otherwise.

A string is whitespace if all characters in the string are whitespace and there is at least one character in the string.

istitle

`XlError.istitle()`

Return True if the string is a title-cased string, False otherwise.

In a title-cased string, upper- and title-case characters may only follow uncased characters and lowercase characters only cased ones.

isupper

`XlError.isupper()`

Return True if the string is an uppercase string, False otherwise.

A string is uppercase if all cased characters in the string are uppercase and there is at least one cased character in the string.

join

`XlError.join(iterable, /)`

Concatenate any number of strings.

The string whose method is called is inserted in between each given string. The result is returned as a new string.

Example: '.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'

ljust

`XlError.ljust(width, fillchar=' ', /)`

Return a left-justified string of length width.

Padding is done using the specified fill character (default is a space).

lower

`XlError.lower()`

Return a copy of the string converted to lowercase.

lstrip

`XlError.lstrip(chars=None, /)`

Return a copy of the string with leading whitespace removed.

If chars is given and not None, remove characters in chars instead.

maketrans

`static XlError.maketrans()`

Return a translation table usable for str.translate().

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in x will be mapped to the character at the same position in y. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

partition

XlError.**partition**(*sep*, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original string and two empty strings.

removeprefix

XlError.**removeprefix**(*prefix*, /)

Return a str with the given prefix string removed if present.

If the string starts with the prefix string, return string[len(prefix):]. Otherwise, return a copy of the original string.

removesuffix

XlError.**removesuffix**(*suffix*, /)

Return a str with the given suffix string removed if present.

If the string ends with the suffix string and that suffix is not empty, return string[:-len(suffix)]. Otherwise, return a copy of the original string.

replace

XlError.**replace**(*old*, *new*, *count=-1*, /)

Return a copy with all occurrences of substring old replaced by new.

count
Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument count is given, only the first count occurrences are replaced.

rfind

XlError.**rfind**(*sub*[, *start*[, *end*]]) → int

Return the highest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

rindex

XlError.**rindex**(*sub*[, *start*[, *end*]]) → int

Return the highest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Raises ValueError when the substring is not found.

rjust

`XlError.rjust(width, fillchar=' ', /)`

Return a right-justified string of length width.

Padding is done using the specified fill character (default is a space).

rpartition

`XlError.rpartition(sep, /)`

Partition the string into three parts using the given separator.

This will search for the separator in the string, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty strings and the original string.

rsplit

`XlError.rsplit(sep=None, maxsplit=-1)`

Return a list of the substrings in the string, using sep as the separator string.

sep
The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including n r t f and spaces) and will discard empty strings from the result.

maxsplit
Maximum number of splits. -1 (the default value) means no limit.

Splitting starts at the end of the string and works to the front.

rstrip

`XlError.rstrip(chars=None, /)`

Return a copy of the string with trailing whitespace removed.

If chars is given and not None, remove characters in chars instead.

split

`XlError.split(sep=None, maxsplit=-1)`

Return a list of the substrings in the string, using sep as the separator string.

sep
The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including n r t f and spaces) and will discard empty strings from the result.

maxsplit
Maximum number of splits. -1 (the default value) means no limit.

Splitting starts at the front of the string and works to the end.

Note, str.split() is mainly useful for data that has been intentionally delimited. With natural text that includes punctuation, consider using the regular expression module.

splitlines

`XlError.splitlines(keepends=False)`

Return a list of the lines in the string, breaking at line boundaries.

Line breaks are not included in the resulting list unless keepends is given and true.

startswith

`XlError.startswith(prefix[, start[, end]]) → bool`

Return True if S starts with the specified prefix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. prefix can also be a tuple of strings to try.

strip

`XlError.strip(chars=None, /)`

Return a copy of the string with leading and trailing whitespace removed.

If chars is given and not None, remove characters in chars instead.

swapcase

`XlError.swapcase()`

Convert uppercase characters to lowercase and lowercase characters to uppercase.

title

`XlError.title()`

Return a version of the string where each word is titlecased.

More specifically, words start with uppercased characters and all remaining cased characters have lower case.

translate

`XlError.translate(table, /)`

Replace each character in the string using the given translation table.

table
Translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, strings, or None.

The table must implement lookup/indexing via __getitem__, for instance a dictionary or list. If this operation raises LookupError, the character is left untouched. Characters mapped to None are deleted.

upper

`XlError.upper()`

Return a copy of the string converted to uppercase.

zfill

XlError.**zfill**(*width*, */*)

Pad a numeric string with zeros on the left, to fill a field of the given width.

The string is never truncated.

__init__(**args*)

3.11.4.3 operator

It provides Operator classes.

Classes

Intersect
Operator
OperatorToken
Separator

Intersect

class Intersect(*s*, *context=None*, *parser=None*)

Methods

__init__
ast
compile
match
process
set_expr
update_input_tokens
update_name

__init__

Intersect.**__init__**(*s*, *context=None*, *parser=None*)

ast

Intersect.**ast**(*tokens*, *stack*, *builder*)

compile

Intersect.**compile**()

match

Intersect.**match**(*s*)

process

Intersect.**process**(*match*, *context=None*, *parser=None*)

set_expr

Intersect.**set_expr**(**tokens*)

update_input_tokens

Intersect.**update_input_tokens**(**tokens*)

update_name

Intersect.**update_name**(*tokens*, *stack*)

__init__(*s*, *context=None*, *parser=None*)

Attributes

get_n_args
name
node_id
pred

get_n_args

property Intersect.**get_n_args**

name

property Intersect.**name**

node_id

property Intersect.**node_id**

pred

property Intersect.**pred**

Operator

class **Operator**(*s*, *context=None*, *parser=None*)

Methods

__init__
ast

continues on next page

Table 39 – continued from previous page

compile
match
process
set_expr
update_input_tokens
update_name

__init__

Operator.**__init__**(*s*, *context=None*, *parser=None*)

ast

Operator.**ast**(*tokens*, *stack*, *builder*)

compile

Operator.**compile**()

match

Operator.**match**(*s*)

process

Operator.**process**(*match*, *context=None*, *parser=None*)

set_expr

Operator.**set_expr**(**tokens*)

update_input_tokens

Operator.**update_input_tokens**(**tokens*)

update_name

Operator.**update_name**(*tokens*, *stack*)

__init__(*s*, *context=None*, *parser=None*)

Attributes

get_n_args
name
node_id
pred

get_n_args

property Operator.**get_n_args**

name

property Operator.**name**

node_id

property Operator.**node_id**

pred

property Operator.**pred**

OperatorToken

class OperatorToken(*s*, *context=None*, *parser=None*)

Methods

__init__
ast
compile
match
process
set_expr
update_input_tokens
update_name

__init__

OperatorToken.**__init__**(*s*, *context=None*, *parser=None*)

ast

OperatorToken.**ast**(*tokens*, *stack*, *builder*)

compile

OperatorToken.**compile**()

match

OperatorToken.**match**(*s*)

process

OperatorToken.**process**(*match*, *context=None*, *parser=None*)

set_expr

OperatorToken.**set_expr**(**tokens*)

update_input_tokens

OperatorToken.**update_input_tokens**(**tokens*)

update_name

OperatorToken.**update_name**(*tokens*, *stack*)

__init__(*s*, *context=None*, *parser=None*)

Attributes

get_n_args
name
node_id
pred

get_n_args

property OperatorToken.**get_n_args**

name

property OperatorToken.**name**

node_id

property OperatorToken.**node_id**

pred

property OperatorToken.**pred**

Separator

class Separator(*s*, *context=None*, *parser=None*)

Methods

__init__
ast

continues on next page

Table 43 – continued from previous page

compile
match
process
set_expr
update_input_tokens
update_name

__init__

Separator.**__init__**(*s*, *context=None*, *parser=None*)

ast

Separator.**ast**(*tokens*, *stack*, *builder*)

compile

Separator.**compile**()

match

Separator.**match**(*s*)

process

Separator.**process**(*match*, *context=None*, *parser=None*)

set_expr

Separator.**set_expr**(**tokens*)

update_input_tokens

Separator.**update_input_tokens**(**tokens*)

update_name

Separator.**update_name**(*tokens*, *stack*)

__init__(*s*, *context=None*, *parser=None*)

Attributes

get_n_args
name
node_id
pred

get_n_args

property Separator.**get_n_args**

name

property Separator.**name**

node_id

property Separator.**node_id**

pred

property Separator.**pred**

3.11.4.4 parenthesis

It provides Parenthesis class.

Classes

Parenthesis

Parenthesis

class Parenthesis(*s*, *context=None*, *parser=None*)

Methods

__init__
ast
match
process
set_expr
update_input_tokens

__init__

Parenthesis.**__init__**(*s*, *context=None*, *parser=None*)

ast

Parenthesis.**ast**(*tokens*, *stack*, *builder*)

match

Parenthesis.**match**(*s*)

process

Parenthesis.**process**(*match*, *context=None*, *parser=None*)

set_expr

Parenthesis.**set_expr**(**tokens*)

update_input_tokens

Parenthesis.**update_input_tokens**(**tokens*)

__init__(*s*, *context=None*, *parser=None*)

Attributes

n_args
name
node_id
opens

n_args

Parenthesis.**n_args** = **0**

name

property Parenthesis.**name**

node_id

property Parenthesis.**node_id**

opens

Parenthesis.**opens** = **{')': '('}**

Classes

Token

3.11.4.5 Token

class Token(*s*, *context=None*, *parser=None*)

Methods

__init__

continues on next page

Table 49 – continued from previous page

ast
match
process
set_expr
update_input_tokens

__init__

Token.**__init__**(*s*, *context=None*, *parser=None*)

ast

Token.**ast**(*tokens*, *stack*, *builder*)

match

Token.**match**(*s*)

process

Token.**process**(*match*, *context=None*, *parser=None*)

set_expr

Token.**set_expr**(**tokens*)

update_input_tokens

Token.**update_input_tokens**(**tokens*)

__init__(*s*, *context=None*, *parser=None*)

Attributes

name
node_id

name

property Token.**name**

node_id

property Token.**node_id**

3.11.5 functions

It provides functions implementations to compile the Excel functions.

Sub-Modules:

comp	Python equivalents of compatibility Excel functions.
date	Python equivalents of financial Excel functions.
eng	Python equivalents of engineering Excel functions.
financial	Python equivalents of financial Excel functions.
google	Python equivalents of google Excel functions.
info	Python equivalents of information Excel functions.
logic	Python equivalents of logical Excel functions.
look	Python equivalents of lookup and reference Excel functions.
math	Python equivalents of math and trigonometry Excel functions.
operators	Python equivalents of Excel operators.
stat	Python equivalents of statistical Excel functions.
text	Python equivalents of text Excel functions.

3.11.5.1 comp

Python equivalents of compatibility Excel functions.

Functions

xtdist

xtdist

xtdist(*x*, *deg_freedom*, *tails*)

3.11.5.2 date

Python equivalents of financial Excel functions.

Functions

args_xnetworkdays_intl
args_xworkday_intl
day_count
xdate
xdatedif
xdatevalue
xday
xdays
xdays360
xedate
xeomonth
xisoweeknum

continues on next page

Table 53 – continued from previous page

xnetworkdays_intl
xnow
xsecond
xtime
xtimevalue
xtoday
xweekday
xweeknum
xworkday_intl
xyearfrac
year_days

args_xnetworkdays_intl

args_xnetworkdays_intl(*start_date*, *end_date*, *weekend=1*, *holidays=()*, *true_err=False*)

args_xworkday_intl

args_xworkday_intl(*start_date*, *days*, *weekend=1*, *holidays=()*, *true_err=False*)

day_count

day_count(*start_date*, *end_date*, *basis*, *exact=False*)

xdate

xdate(*year*, *month*, *day*)

xdatedif

xdatedif(*start_date*, *end_date*, *unit*)

xdatevalue

xdatevalue(*date_text*)

xday

xday(*serial_number*, *n=2*)

xdays

xdays(*end_date*, *start_date*)

xdays360

xdays360(*start_date*, *end_date*, *method=False*)

xedate

xedate(*start_date*, *months*)

xeomonth

xeomonth(*start_date*, *months*)

xisoweeknum

xisoweeknum(*serial_number*)

xnetworkdays_intl

xnetworkdays_intl(*start_date*, *end_date*, *weekend=1*, *holidays=()*)

xnow

xnow()

xsecond

xsecond(*serial_number*, *n=2*)

xtime

xtime(*hour*, *minute*, *second*)

xtimevalue

xtimevalue(*time_text*)

xtoday

xtoday()

xweekday

xweekday(*serial_number*, *n=1*)

xweeknum

xweeknum(*serial_number*, *n=1*)

xworkday_intl

xworkday_intl(*start_date*, *days*, *weekend=1*, *holidays=()*)

xyearfrac

xyearfrac(*start_date*, *end_date*, *basis=0*)

year_days

year_days(*start_date*, *end_date*, *basis*)

3.11.5.3 eng

Python equivalents of engineering Excel functions.

Functions

hex2dec2bin2oct	
xbesseli	
xbesselj	
xbesselk	
xbessely	
xbitand	
xbitlshift	Excel-like BITLSHIFT (shift >= 0).
xbitor	Excel-like BITOR.
xbitrshift	Excel-like BITRSHIFT (shift >= 0).
xbitxor	Excel-like BITXOR.
xcomplex	
xconvert	
xdelta	
xerf	
xerf_precise	
xgestep	
ximargument	
ximpower	

hex2dec2bin2oct

hex2dec2bin2oct(*function_id*, *memo*)

xbesseli

xbesseli(*x*, *n*)

xbesselj

xbesselj(*x*, *n*)

xbesselk

xbesselk(*x*, *n*)

xbessely

xbessely(*x*, *n*)

xbitand

xbitand(*x*, *y*)

xbitlshift

xbitlshift(*x*, *shift*)

Excel-like BITLSHIFT (shift >= 0).

xbitor

xbitor(*x*, *y*)

Excel-like BITOR.

xbitrshift

xbitrshift(*x*, *shift*)

Excel-like BITRSHIFT (shift >= 0).

xbitxor

xbitxor(*x*, *y*)

Excel-like BITXOR.

xcomplex

xcomplex(*real_num*, *i_num*, *suffix='i'*)

xconvert

xconvert(*number*, *from_unit*, *to_unit*)

xdelta

xdelta(*x*, *y=0*)

xerf

xerf(*lower*, *upper=None*)

xerf_precise

xerf_precise(*x*, *func=<built-in function erf>*)

xgestep

xgestep(*x*, *step=0*)

ximargument

ximargument(*x*)

ximpower

ximpower(*z*, *power*)

3.11.5.4 financial

Python equivalents of financial Excel functions.

Functions

args_parser_disc	
args_parser_fvschedule	
args_parser_intrate	
args_parser_received	
mirr_args_parser	
parse_basis	
parse_date	
total_depr	Cumulative (total) depreciation up to 'period' (can be fractional), using double-declining (or general 'factor') balance with optional straight-line switchover when SLN gives a higher amount.
xaccrint	
xaccrintm	
xamordegrc	
xamorlinc	
xcoupdaybs	
xcoupdays	
xcoupdaysnc	
xcoupncd	
xcoupnum	
xcouppcd	
xcumipmt	
xdate2date	
xdb	
xddb	
xdisc	
xdollarde	
xdollarfr	
xduration	
xeffect	
xfvschedule	
xintrate	
xirr	
xispmt	

continues on next page

Table 55 – continued from previous page

xmirr
xnominal
xnper
xnpv
xoddfprice
xoddfyield
xoddlprice
xoddlyield
xpduration
xppmt
xprice
xpricedisc
xpricemat
xrate
xreceived
xrri
xsln
xsyd
xtbilleq
xtbillprice
xtbillyield
xvdb
xxirr
xxnpv
xyield
xyielddisc
xyieldmat

args_parser_disc

args_parser_disc(*settlement*, *maturity*, *pr*, *redemption*, *basis=0*)

args_parser_fvschedule

args_parser_fvschedule(*principal*, *schedule*)

args_parser_intrate

args_parser_intrate(*settlement*, *maturity*, *investment*, *redemption*, *basis=0*)

args_parser_received

args_parser_received(*settlement*, *maturity*, *investment*, *discount*, *basis=0*)

mirr_args_parser

mirr_args_parser(*values*, *finance_rate*, *reinvest_rate*)

parse_basis

parse_basis(*basis*, *func=<class 'int'>*)

parse_date

parse_date(*date*)

total_depr

total_depr(*cost: float*, *salvage: float*, *life: float*, *period: float*, *factor: float*, *straight_line: bool*) → float

Cumulative (total) depreciation up to 'period' (can be fractional), using double-declining (or general 'factor') balance with optional straight-line switchover when SLN gives a higher amount.

Mirrors the provided JS logic:

- period can be fractional (e.g., 3.4); only first and last partials are prorated.
- When straight_line is True, switch from DDB to SLN when SLN > DDB on that step.
- Caps total depreciation so book value doesn't go below salvage.

xaccrint

xaccrint(*issue*, *first_interest*, *settlement*, *rate*, *par*, *frequency*, *basis=0*, *calc_method=1*)

xaccrintm

xaccrintm(*issue*, *settlement*, *rate*, *par*, *basis=0*)

xamordegrc

xamordegrc(*cost*, *date_purchased*, *first_period*, *salvage*, *period*, *rate*, *basis=0*)

xamorlinc

xamorlinc(*cost*, *date_purchased*, *first_period*, *salvage*, *period*, *rate*, *basis=0*)

xcoupdaybs

xcoupdaybs(*settlement*, *maturity*, *frequency*, *basis=0*, ** (Keyword-only parameters separator (PEP 3102))*, *coupons=()*)

xcoupdays

xcoupdays(*settlement*, *maturity*, *frequency*, *basis=0*, ***, *coupons=()*)

xcoupdaysnc

xcoupdaysnc(*settlement*, *maturity*, *frequency*, *basis=0*, ***, *coupons=()*)

xcoupncd

xcoupncd(*settlement, maturity, frequency, basis=0, *, coupons=()*)

xcoupnum

xcoupnum(*settlement, maturity, frequency, basis=0, *, coupons=()*)

xcouppcd

xcouppcd(*settlement, maturity, frequency, basis=0, *, coupons=()*)

xcumipmt

xcumipmt(*rate, nper, pv, start_period, end_period, type, *, func=functools.partial(<function _npf>, 'ipmt')*)

xdate2date

xdate2date(**date*)

xdb

xdb(*cost, salvage, life, period, month=12*)

xddb

xddb(*cost, salvage, life, period, factor=2*)

xdisc

xdisc(*num, dates, basis=0*)

xdollarde

xdollarde(*fractional, denominator*)

xdollarfr

xdollarfr(*decimal_dollar, fraction*)

xduration

xduration(*settlement, maturity, coupon, yld, frequency, basis=0, *, face=100.0, modified=False*)

xeffect

xeffect(*nominal_rate, npery*)

xfvschedule

xfvschedule(*principal*, *prod*)

xintrate

xintrate(*num*, *dates*, *basis=0*)

xirr

xirr(*values*, *guess=0.1*)

xispmt

xispmt(*rate*, *per*, *nper*, *pv*)

xmirr

xmirr(*values*, *finance_rate*, *reinvest_rate*)

xnominal

xnominal(*effect_rate*, *npery*)

xnper

xnper(*rate*, *pmt*, *pv*, *fv=0*, *type=0*)

xnpv

xnpv(*rate*, *values*, *dates=None*)

xoddfprice

xoddfprice(*settlement*, *maturity*, *issue*, *first_coupon*, *rate*, *yld*, *redemption*, *frequency*, *basis=0*)

xoddfyield

xoddfyield(*settlement*, *maturity*, *issue*, *first_coupon*, *rate*, *pr*, *redemption*, *frequency*, *basis=0*)

xoddlprice

xoddlprice(*settlement*, *maturity*, *last_interest*, *rate*, *yld*, *redemption*, *frequency*, *basis=0*)

xoddlyield

xoddlyield(*settlement*, *maturity*, *last_interest*, *rate*, *pr*, *redemption*, *frequency*, *basis=0*)

xpduration

xpduration(*rate*, *pv*, *fv*)

xppmt

xppmt(*rate*, *per*, *nper*, *pv*, *fv=0*, *type=0*)

xprice

xprice(*settlement*, *maturity*, *rate*, *yld*, *redemption*, *frequency*, *basis=0*)

xpricedisc

xpricedisc(*settlement*, *maturity*, *discount*, *redemption*, *basis=0*)

xpricemat

xpricemat(*settlement*, *maturity*, *issue*, *rate*, *yld*, *basis=0*)

xrate

xrate(*nper*, *pmt*, *pv*, *fv=0*, *type=0*, *guess=0.1*)

xreceived

xreceived(*investment*, *discount*, *dates*, *basis=0*)

xrri

xrri(*rate*, *pv*, *fv*)

xsln

xsln(*cost*, *salvage*, *life*)

xsyd

xsyd(*cost*, *salvage*, *life*, *per*)

xtbilleq

xtbilleq(*settlement*, *maturity*, *discount*)

xtbillprice

xtbillprice(*settlement*, *maturity*, *discount*)

xtbillyield

xtbillyield(*settlement*, *maturity*, *pr*)

xvdb

xvdb(*cost*, *salvage*, *life*, *start_period*, *end_period*, *factor=2*, *no_switch=0*)

xxirr

xxirr(*values*, *dates*, *x=0.1*)

xxnpv

xxnpv(*rate*, *values*, *dates*)

xyield

xyield(*settlement*, *maturity*, *rate*, *pr*, *redemption*, *frequency*, *basis=0*)

xyielddisc

xyielddisc(*settlement*, *maturity*, *price*, *redemption*, *basis=0*)

xyieldmat

xyieldmat(*settlement*, *maturity*, *issue*, *rate*, *pr*, *basis=0*)

3.11.5.5 google

Python equivalents of google Excel functions.

Functions

xdummy

xdummy

xdummy(**args*)

3.11.5.6 info

Python equivalents of information Excel functions.

Functions

iserr
iserror
isna
isref

continues on next page

Table 57 – continued from previous page

xerrortype
xiseven_odd
xisformula
xn
xna
xtype

iserr

iserr(*val*)

iserror

iserror(*val*, *check=<function <lambda>>*, *array=<class 'formulas.functions.info.TrueArray'>*)

isna

isna(*value*)

isref

isref(*val*)

xerrortype

xerrortype(*val*)

xiseven_odd

xiseven_odd(*number*, *odd=False*)

xisformula

xisformula(*dsp=None*, *ref=None*)

xn

xn(*val*)

xna

xna()

xtype

xtype(*val*)

Classes

FalseArray
TrueArray

FalseArray

class FalseArray(*shape*, *dtype=None*, *buffer=None*, *offset=0*, *strides=None*, *order=None*)

Methods

__init__	
`all`	Returns True if all elements evaluate to True.
`any`	Returns True if any of the elements of *a* evaluate to True.
`argmax`	Return indices of the maximum values along the given axis.
`argmin`	Return indices of the minimum values along the given axis.
`argpartition`	Returns the indices that would partition this array.
`argsort`	Returns the indices that would sort this array.
`astype`	Copy of the array, cast to a specified type.
`byteswap`	Swap the bytes of the array elements
`choose`	Use an index array to construct a new array from a set of choices.
`clip`	Return an array whose values are limited to `[min, max]`.
`collapse`	
`compress`	Return selected slices of this array along given axis.
`conj`	Complex-conjugate all elements.
`conjugate`	Return the complex conjugate, element-wise.
`copy`	Return a copy of the array.
`cumprod`	Return the cumulative product of the elements along the given axis.
`cumsum`	Return the cumulative sum of the elements along the given axis.
`diagonal`	Return specified diagonals.
`dot`	Refer to `numpy.dot()` for full documentation.
`dump`	Dump a pickle of the array to the specified file.
`dumps`	Returns the pickle of the array as a string.
`fill`	Fill the array with a scalar value.
`flatten`	Return a copy of the array collapsed into one dimension.
`getfield`	Returns a field of the given array as a certain type.
`item`	Copy an element of an array to a standard Python scalar and return it.
`max`	Return the maximum along a given axis.
`mean`	Returns the average of the array elements along given axis.
`min`	Return the minimum along a given axis.
`nonzero`	Return the indices of the elements that are non-zero.

continues on next page

Table 59 – continued from previous page

partition	Partially sorts the elements in the array in such a way that the value of the element in k-th position is in the position it would be in a sorted array.
prod	Return the product of the array elements over the given axis
put	Set `a.flat[n] = values[n]` for all `n` in indices.
ravel	Return a flattened array.
repeat	Repeat elements of an array.
reshape	Returns an array containing the same data with a new shape.
resize	Change shape and size of array in-place.
round	Return *a* with each element rounded to the given number of decimals.
searchsorted	Find indices where elements of *v* should be inserted in *a* to maintain order.
setfield	Put a value into a specified place in a field defined by a data-type.
setflags	Set array flags WRITEABLE, ALIGNED, WRITE-BACKIFCOPY, respectively.
sort	Sort an array in-place.
squeeze	Remove axes of length one from *a*.
std	Returns the standard deviation of the array elements along given axis.
sum	Return the sum of the array elements over the given axis.
swapaxes	Return a view of the array with *axis1* and *axis2* interchanged.
take	Return an array formed from the elements of *a* at the given indices.
to_device	For Array API compatibility.
tobytes	Construct Python bytes containing the raw data bytes in the array.
tofile	Write array to a file as text or binary (default).
tolist	Return the array as an `a.ndim`-levels deep nested list of Python scalars.
trace	Return the sum along diagonals of the array.
transpose	Returns a view of the array with axes transposed.
var	Returns the variance of the array elements, along given axis.
view	New view of array with the same data.

__init__

FalseArray.**__init__**()

all

FalseArray.**all**(*axis=None*, *out=None*, ***, *keepdims=<no value>*, *where=<no value>*)

Returns True if all elements evaluate to True.

Refer to *numpy.all* for full documentation.

See Also

numpy.all : equivalent function

any

`FalseArray.any`(*axis=None*, *out=None*, *, *keepdims=<no value>*, *where=<no value>*)

Returns True if any of the elements of *a* evaluate to True.

Refer to *numpy.any* for full documentation.

See Also

numpy.any : equivalent function

argmax

`FalseArray.argmax`(*axis=None*, *out=None*, *, *keepdims=False*)

Return indices of the maximum values along the given axis.

Refer to *numpy.argmax* for full documentation.

See Also

numpy.argmax : equivalent function

argmin

`FalseArray.argmin`(*axis=None*, *out=None*, *, *keepdims=False*)

Return indices of the minimum values along the given axis.

Refer to *numpy.argmin* for detailed documentation.

See Also

numpy.argmin : equivalent function

argpartition

`FalseArray.argpartition`(*kth*, *axis=-1*, *kind='introselect'*, *order=None*)

Returns the indices that would partition this array.

Refer to *numpy.argpartition* for full documentation.

See Also

numpy.argpartition : equivalent function

argsort

`FalseArray.argsort`(*axis=-1*, *kind=None*, *order=None*, *, *stable=None*)

Returns the indices that would sort this array.

Refer to *numpy.argsort* for full documentation.

See Also

numpy.argsort : equivalent function

astype

`FalseArray.astype(dtype, order='K', casting='unsafe', subok=True, copy=True)`

Copy of the array, cast to a specified type.

Parameters

dtype
[str or dtype] Typecode or data-type to which the array is cast.

order
[{'C', 'F', 'A', 'K'}, optional] Controls the memory layout order of the result. 'C' means C order, 'F' means Fortran order, 'A' means 'F' order if all the arrays are Fortran contiguous, 'C' order otherwise, and 'K' means as close to the order the array elements appear in memory as possible. Default is 'K'.

casting
[{'no', 'equiv', 'safe', 'same_kind', 'same_value', 'unsafe'}, optional] Controls what kind of data casting may occur. Defaults to 'unsafe' for backwards compatibility.

- 'no' means the data types should not be cast at all.
- 'equiv' means only byte-order changes are allowed.
- 'safe' means only casts which can preserve values are allowed.
- 'same_kind' means only safe casts or casts within a kind, like float64 to float32, are allowed.
- 'unsafe' means any data conversions may be done.
- 'same_value' means any data conversions may be done, but the values must not change, including rounding of floats or overflow of ints

Added in version 2.4: Support for `'same_value'` was added.

subok
[bool, optional] If True, then sub-classes will be passed-through (default), otherwise the returned array will be forced to be a base-class array.

copy
[bool, optional] By default, astype always returns a newly allocated array. If this is set to false, and the *dtype*, *order*, and *subok* requirements are satisfied, the input array is returned instead of a copy.

Returns

arr_t
[ndarray] Unless *copy* is False and the other conditions for returning the input array are satisfied (see description for *copy* input parameter), *arr_t* is a new array of the same shape as the input array, with dtype, order given by *dtype*, *order*.

Raises

ComplexWarning
When casting from complex to float or int. To avoid this, one should use `a.real.astype(t)`.

ValueError
When casting using `'same_value'` and the values change or would overflow

Examples

```
>>> import numpy as np
>>> x = np.array([1, 2, 2.5])
>>> x
array([1. ,  2. ,  2.5])
```

```
>>> x.astype(int)
array([1, 2, 2])
```

```
>>> x.astype(int, casting="same_value")
Traceback (most recent call last):
...
ValueError: could not cast 'same_value' double to long
```

```
>>> x[:2].astype(int, casting="same_value")
array([1, 2])
```

byteswap

FalseArray.**byteswap**(*inplace=False*)

Swap the bytes of the array elements

Toggle between low-endian and big-endian data representation by returning a byteswapped array, optionally swapped in-place. Arrays of byte-strings are not swapped. The real and imaginary parts of a complex number are swapped individually.

Parameters

inplace
[bool, optional] If `True`, swap bytes in-place, default is `False`.

Returns

out
[ndarray] The byteswapped array. If *inplace* is `True`, this is a view to self.

Examples

```
>>> import numpy as np
>>> A = np.array([1, 256, 8755], dtype=np.int16)
>>> list(map(hex, A))
['0x1', '0x100', '0x2233']
>>> A.byteswap(inplace=True)
array([  256,     1, 13090], dtype=int16)
>>> list(map(hex, A))
['0x100', '0x1', '0x3322']
```

Arrays of byte-strings are not swapped

```
>>> A = np.array([b'ceg', b'fac'])
>>> A.byteswap()
array([b'ceg', b'fac'], dtype='|S3')
```

`A.view(A.dtype.newbyteorder()).byteswap()` produces an array with the same values but different representation in memory

```
>>> A = np.array([1, 2, 3],dtype=np.int64)
>>> A.view(np.uint8)
array([1, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0,
       0, 0], dtype=uint8)
>>> A.view(A.dtype.newbyteorder()).byteswap(inplace=True)
array([1, 2, 3], dtype='>i8')
>>> A.view(np.uint8)
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0,
       0, 3], dtype=uint8)
```

choose

`FalseArray.`**`choose`**`(`*choices*, *out=None*, *mode='raise'*`)`

Use an index array to construct a new array from a set of choices.

Refer to *numpy.choose* for full documentation.

See Also

numpy.choose : equivalent function

clip

`FalseArray.`**`clip`**`(`*min=<no value>*, *max=<no value>*, *out=None*, ***kwargs*`)`

Return an array whose values are limited to `[min, max]`. One of max or min must be given.

Refer to *numpy.clip* for full documentation.

See Also

numpy.clip : equivalent function

collapse

`FalseArray.`**`collapse`**`(`*shape*`)`

compress

`FalseArray.`**`compress`**`(`*condition*, *axis=None*, *out=None*`)`

Return selected slices of this array along given axis.

Refer to *numpy.compress* for full documentation.

See Also

numpy.compress : equivalent function

conj

`FalseArray.conj()`

Complex-conjugate all elements.

Refer to *numpy.conjugate* for full documentation.

See Also

numpy.conjugate : equivalent function

conjugate

`FalseArray.conjugate()`

Return the complex conjugate, element-wise.

Refer to *numpy.conjugate* for full documentation.

See Also

numpy.conjugate : equivalent function

copy

`FalseArray.copy(order='C')`

Return a copy of the array.

Parameters

order

[{'C', 'F', 'A', 'K'}, optional] Controls the memory layout of the copy. 'C' means C-order, 'F' means F-order, 'A' means 'F' if *a* is Fortran contiguous, 'C' otherwise. 'K' means match the layout of *a* as closely as possible. (Note that this function and `numpy.copy()` are very similar but have different default values for their order= arguments, and this function always passes sub-classes through.)

See also

numpy.copy : Similar function with different default behavior numpy.copyto

Notes

This function is the preferred method for creating an array copy. The function `numpy.copy()` is similar, but it defaults to using order 'K', and will not pass sub-classes through by default.

Examples

```
>>> import numpy as np
>>> x = np.array([[1,2,3],[4,5,6]], order='F')
```

```
>>> y = x.copy()
```

```
>>> x.fill(0)
```

```
>>> x
array([[0, 0, 0],
       [0, 0, 0]])
```

```
>>> y
array([[1, 2, 3],
       [4, 5, 6]])
```

```
>>> y.flags['C_CONTIGUOUS']
True
```

For arrays containing Python objects (e.g. dtype=object), the copy is a shallow one. The new array will contain the same object which may lead to surprises if that object can be modified (is mutable):

```
>>> a = np.array([1, 'm', [2, 3, 4]], dtype=object)
>>> b = a.copy()
>>> b[2][0] = 10
>>> a
array([1, 'm', list([10, 3, 4])], dtype=object)
```

To ensure all elements within an `object` array are copied, use *copy.deepcopy*:

```
>>> import copy
>>> a = np.array([1, 'm', [2, 3, 4]], dtype=object)
>>> c = copy.deepcopy(a)
>>> c[2][0] = 10
>>> c
array([1, 'm', list([10, 3, 4])], dtype=object)
>>> a
array([1, 'm', list([2, 3, 4])], dtype=object)
```

cumprod

`FalseArray.`**`cumprod`**(*axis=None*, *dtype=None*, *out=None*)

Return the cumulative product of the elements along the given axis.

Refer to *numpy.cumprod* for full documentation.

See Also

numpy.cumprod : equivalent function

cumsum

`FalseArray.`**`cumsum`**(*axis=None*, *dtype=None*, *out=None*)

Return the cumulative sum of the elements along the given axis.

Refer to *numpy.cumsum* for full documentation.

See Also

numpy.cumsum : equivalent function

diagonal

`FalseArray.diagonal(offset=0, axis1=0, axis2=1)`

Return specified diagonals. In NumPy 1.9 the returned array is a read-only view instead of a copy as in previous NumPy versions. In a future version the read-only restriction will be removed.

Refer to `numpy.diagonal()` for full documentation.

See Also

numpy.diagonal : equivalent function

dot

`FalseArray.dot(other, /, out=None)`

Refer to `numpy.dot()` for full documentation.

See Also

numpy.dot : equivalent function

dump

`FalseArray.dump(file)`

Dump a pickle of the array to the specified file. The array can be read back with pickle.load or numpy.load.

Parameters

file
[str or Path] A string naming the dump file.

dumps

`FalseArray.dumps()`

Returns the pickle of the array as a string. `pickle.loads` will convert the string back to an array.

Parameters

None

fill

`FalseArray.fill(value)`

Fill the array with a scalar value.

Parameters

value
[scalar] All elements of *a* will be assigned this value.

Examples

```
>>> import numpy as np
>>> a = np.array([1, 2])
>>> a.fill(0)
>>> a
array([0, 0])
>>> a = np.empty(2)
>>> a.fill(1)
>>> a
array([1.,  1.])
```

Fill expects a scalar value and always behaves the same as assigning to a single array element. The following is a rare example where this distinction is important:

```
>>> a = np.array([None, None], dtype=object)
>>> a[0] = np.array(3)
>>> a
array([array(3), None], dtype=object)
>>> a.fill(np.array(3))
>>> a
array([array(3), array(3)], dtype=object)
```

Where other forms of assignments will unpack the array being assigned:

```
>>> a[...] = np.array(3)
>>> a
array([3, 3], dtype=object)
```

flatten

`FalseArray.flatten(order='C')`

Return a copy of the array collapsed into one dimension.

Parameters

order

[{‘C’, ‘F’, ‘A’, ‘K’}, optional] ‘C’ means to flatten in row-major (C-style) order. ‘F’ means to flatten in column-major (Fortran- style) order. ‘A’ means to flatten in column-major order if *a* is Fortran *contiguous* in memory, row-major order otherwise. ‘K’ means to flatten *a* in the order the elements occur in memory. The default is ‘C’.

Returns

y

[ndarray] A copy of the input array, flattened to one dimension.

See Also

ravel : Return a flattened array. flat : A 1-D flat iterator over the array.

Examples

```
>>> import numpy as np
>>> a = np.array([[1,2], [3,4]])
>>> a.flatten()
array([1, 2, 3, 4])
>>> a.flatten('F')
array([1, 3, 2, 4])
```

getfield

FalseArray.**getfield**(*dtype*, *offset=0*)

Returns a field of the given array as a certain type.

A field is a view of the array data with a given data-type. The values in the view are determined by the given type and the offset into the current array in bytes. The offset needs to be such that the view dtype fits in the array dtype; for example an array of dtype complex128 has 16-byte elements. If taking a view with a 32-bit integer (4 bytes), the offset needs to be between 0 and 12 bytes.

Parameters

dtype
[str or dtype] The data type of the view. The dtype size of the view can not be larger than that of the array itself.

offset
[int] Number of bytes to skip before beginning the element view.

Examples

```
>>> import numpy as np
>>> x = np.diag([1.+1.j]*2)
>>> x[1, 1] = 2 + 4.j
>>> x
array([[1.+1.j,  0.+0.j],
       [0.+0.j,  2.+4.j]])
>>> x.getfield(np.float64)
array([[1.,  0.],
       [0.,  2.]])
```

By choosing an offset of 8 bytes we can select the complex part of the array for our view:

```
>>> x.getfield(np.float64, offset=8)
array([[1.,  0.],
       [0.,  4.]])
```

item

FalseArray.**item**(**args*)

Copy an element of an array to a standard Python scalar and return it.

Parameters

*args : Arguments (variable number and type)

- none: in this case, the method only works for arrays with one element (*a.size == 1*), which element is copied into a standard Python scalar object and returned.
- int_type: this argument is interpreted as a flat index into the array, specifying which element to copy and return.
- tuple of int_types: functions as does a single int_type argument, except that the argument is interpreted as an nd-index into the array.

Returns

z

[Standard Python scalar object] A copy of the specified element of the array as a suitable Python scalar

Notes

When the data type of *a* is longdouble or clongdouble, item() returns a scalar array object because there is no available Python scalar that would not lose information. Void arrays return a buffer object for item(), unless fields are defined, in which case a tuple is returned.

item is very similar to a[args], except, instead of an array scalar, a standard Python scalar is returned. This can be useful for speeding up access to elements of the array and doing arithmetic on elements of the array using Python's optimized math.

Examples

```
>>> import numpy as np
>>> np.random.seed(123)
>>> x = np.random.randint(9, size=(3, 3))
>>> x
array([[2, 2, 6],
       [1, 3, 6],
       [1, 0, 1]])
>>> x.item(3)
1
>>> x.item(7)
0
>>> x.item((0, 1))
2
>>> x.item((2, 2))
1
```

For an array with object dtype, elements are returned as-is.

```
>>> a = np.array([np.int64(1)], dtype=object)
>>> a.item() #return np.int64
np.int64(1)
```

max

`FalseArray.max`(*axis=None*, *out=None*, *, *keepdims=<no value>*, *initial=<no value>*, *where=<no value>*)

Return the maximum along a given axis.

Refer to *numpy.amax* for full documentation.

See Also

numpy.amax : equivalent function

mean

`FalseArray.mean`(*axis=None*, *dtype=None*, *out=None*, *, *keepdims=<no value>*, *where=<no value>*)

Returns the average of the array elements along given axis.

Refer to *numpy.mean* for full documentation.

See Also

numpy.mean : equivalent function

min

`FalseArray.min`(*axis=None*, *out=None*, *, *keepdims=<no value>*, *initial=<no value>*, *where=<no value>*)

Return the minimum along a given axis.

Refer to *numpy.amin* for full documentation.

See Also

numpy.amin : equivalent function

nonzero

`FalseArray.nonzero()`

Return the indices of the elements that are non-zero.

Refer to *numpy.nonzero* for full documentation.

See Also

numpy.nonzero : equivalent function

partition

`FalseArray.partition`(*kth*, *axis=-1*, *kind='introselect'*, *order=None*)

Partially sorts the elements in the array in such a way that the value of the element in k-th position is in the position it would be in a sorted array. In the output array, all elements smaller than the k-th element are located to the left of this element and all equal or greater are located to its right. The ordering of the elements in the two partitions on the either side of the k-th element in the output array is undefined.

Parameters

kth
[int or sequence of ints] Element index to partition by. The kth element value will be in its final sorted position and all smaller elements will be moved before it and all equal or greater elements behind it. The order of all elements in the partitions is undefined. If provided with a sequence of kth it will partition all elements indexed by kth of them into their sorted position at once.

Deprecated since version 1.22.0: Passing booleans as index is deprecated.

axis
[int, optional] Axis along which to sort. Default is -1, which means sort along the last axis.

kind
[{'introselect'}, optional] Selection algorithm. Default is 'introselect'.

order
[str or list of str, optional] When *a* is an array with fields defined, this argument specifies which fields to compare first, second, etc. A single field can be specified as a string, and not all fields need to be specified, but unspecified fields will still be used, in the order in which they come up in the dtype, to break ties.

See Also

numpy.partition : Return a partitioned copy of an array. argpartition : Indirect partition. sort : Full sort.

Notes

See `np.partition` for notes on the different algorithms.

Examples

```
>>> import numpy as np
>>> a = np.array([3, 4, 2, 1])
>>> a.partition(3)
>>> a
array([2, 1, 3, 4]) # may vary
```

```
>>> a.partition((1, 3))
>>> a
array([1, 2, 3, 4])
```

prod

`FalseArray.prod`(*axis=None*, *dtype=None*, *out=None*, ***, *keepdims=<no value>*, *initial=<no value>*, *where=<no value>*)

Return the product of the array elements over the given axis

Refer to *numpy.prod* for full documentation.

See Also

numpy.prod : equivalent function

put

`FalseArray.put(indices, values, mode='raise')`

Set `a.flat[n] = values[n]` for all `n` in indices.

Refer to *numpy.put* for full documentation.

See Also

numpy.put : equivalent function

ravel

`FalseArray.ravel(order='C')`

Return a flattened array.

Refer to *numpy.ravel* for full documentation.

See Also

numpy.ravel : equivalent function ndarray.flat : a flat iterator on the array.

repeat

`FalseArray.repeat(repeats, axis=None)`

Repeat elements of an array.

Refer to *numpy.repeat* for full documentation.

See Also

numpy.repeat : equivalent function

reshape

`FalseArray.reshape(shape, /, *, order='C', copy=None)`

`FalseArray.reshape(*shape, order='C', copy=None)`

Returns an array containing the same data with a new shape.

Refer to *numpy.reshape* for full documentation.

See Also

numpy.reshape : equivalent function

Notes

Unlike the free function *numpy.reshape*, this method on *ndarray* allows the elements of the shape parameter to be passed in as separate arguments. For example, `a.reshape(4, 2)` is equivalent to `a.reshape((4, 2))`.

resize

`FalseArray.resize(new_shape, /, *, refcheck=True)`

`FalseArray.resize(*new_shape, refcheck=True)`

Change shape and size of array in-place.

Parameters

new_shape
[tuple of ints, or *n* ints] Shape of resized array.

refcheck
[bool, optional] If False, reference count will not be checked. Default is True.

Returns

None

Raises

ValueError
If *a* does not own its own data or references or views to it exist, and the data memory must be changed. PyPy only: will always raise if the data memory must be changed, since there is no reliable way to determine if references or views to it exist.

SystemError
If the *order* keyword argument is specified. This behaviour is a bug in NumPy.

See Also

resize : Return a new array with the specified shape.

Notes

This reallocates space for the data area if necessary.

Only contiguous arrays (data elements consecutive in memory) can be resized.

The purpose of the reference count check is to make sure you do not use this array as a buffer for another Python object and then reallocate the memory. However, reference counts can increase in other ways so if you are sure that you have not shared the memory for this array with another Python object, then you may safely set *refcheck* to False.

Examples

Shrinking an array: array is flattened (in the order that the data are stored in memory), resized, and reshaped:

```
>>> import numpy as np
```

```
>>> a = np.array([[0, 1], [2, 3]], order='C')
>>> a.resize((2, 1))
>>> a
array([[0],
       [1]])
```

```
>>> a = np.array([[0, 1], [2, 3]], order='F')
>>> a.resize((2, 1))
>>> a
array([[0],
       [2]])
```

Enlarging an array: as above, but missing entries are filled with zeros:

```
>>> b = np.array([[0, 1], [2, 3]])
>>> b.resize(2, 3) # new_shape parameter doesn't have to be a tuple
>>> b
array([[0, 1, 2],
       [3, 0, 0]])
```

Referencing an array prevents resizing…

```
>>> c = a
>>> a.resize((1, 1))
Traceback (most recent call last):
...
ValueError: cannot resize an array that references or is referenced ...
```

Unless *refcheck* is False:

```
>>> a.resize((1, 1), refcheck=False)
>>> a
array([[0]])
>>> c
array([[0]])
```

round

FalseArray.**round**(*decimals=0*, *out=None*)

Return *a* with each element rounded to the given number of decimals.

Refer to *numpy.around* for full documentation.

See Also

numpy.around : equivalent function

searchsorted

FalseArray.**searchsorted**(*v*, *side='left'*, *sorter=None*)

Find indices where elements of *v* should be inserted in *a* to maintain order.

For full documentation, see *numpy.searchsorted*.

See Also

numpy.searchsorted : equivalent function

setfield

`FalseArray.setfield(val, dtype, offset=0)`

Put a value into a specified place in a field defined by a data-type.

Place *val* into *a*'s field defined by *dtype* and beginning *offset* bytes into the field.

Parameters

val
[object] Value to be placed in field.

dtype
[dtype object] Data-type of the field in which to place *val*.

offset
[int, optional] The number of bytes into the field at which to place *val*.

Returns

None

See Also

getfield

Examples

```
>>> import numpy as np
>>> x = np.eye(3)
>>> x.getfield(np.float64)
array([[1.,  0.,  0.],
       [0.,  1.,  0.],
       [0.,  0.,  1.]])
>>> x.setfield(3, np.int32)
>>> x.getfield(np.int32)
array([[3, 3, 3],
       [3, 3, 3],
       [3, 3, 3]], dtype=int32)
>>> x
array([[1.0e+000, 1.5e-323, 1.5e-323],
       [1.5e-323, 1.0e+000, 1.5e-323],
       [1.5e-323, 1.5e-323, 1.0e+000]])
>>> x.setfield(np.eye(3), np.int32)
>>> x
array([[1.,  0.,  0.],
       [0.,  1.,  0.],
       [0.,  0.,  1.]])
```

setflags

`FalseArray.setflags(write=None, align=None, uic=None)`

Set array flags WRITEABLE, ALIGNED, WRITEBACKIFCOPY, respectively.

These Boolean-valued flags affect how numpy interprets the memory area used by *a* (see Notes below). The ALIGNED flag can only be set to True if the data is actually aligned according to the type. The WRITEBACKIFCOPY flag can never be set to True. The flag WRITEABLE can only be set to True if the array owns its own memory, or the ultimate owner of the memory exposes a writeable buffer interface, or is a string. (The exception for string is made so that unpickling can be done without copying memory.)

Parameters

write
[bool, optional] Describes whether or not *a* can be written to.

align
[bool, optional] Describes whether or not *a* is aligned properly for its type.

uic
[bool, optional] Describes whether or not *a* is a copy of another "base" array.

Notes

Array flags provide information about how the memory area used for the array is to be interpreted. There are 7 Boolean flags in use, only three of which can be changed by the user: WRITEBACKIFCOPY, WRITEABLE, and ALIGNED.

WRITEABLE (W) the data area can be written to;

ALIGNED (A) the data and strides are aligned appropriately for the hardware (as determined by the compiler);

WRITEBACKIFCOPY (X) this array is a copy of some other array (referenced by .base). When the C-API function PyArray_ResolveWritebackIfCopy is called, the base array will be updated with the contents of this array.

All flags can be accessed using the single (upper case) letter as well as the full name.

Examples

```
>>> import numpy as np
>>> y = np.array([[3, 1, 7],
...                [2, 0, 0],
...                [8, 5, 9]])
>>> y
array([[3, 1, 7],
       [2, 0, 0],
       [8, 5, 9]])
>>> y.flags
  C_CONTIGUOUS : True
  F_CONTIGUOUS : False
  OWNDATA : True
  WRITEABLE : True
  ALIGNED : True
  WRITEBACKIFCOPY : False
>>> y.setflags(write=0, align=0)
>>> y.flags
  C_CONTIGUOUS : True
  F_CONTIGUOUS : False
  OWNDATA : True
```

(continues on next page)

(continued from previous page)

```
  WRITEABLE : False
  ALIGNED : False
  WRITEBACKIFCOPY : False
>>> y.setflags(uic=1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: cannot set WRITEBACKIFCOPY flag to True
```

sort

FalseArray.**sort**(*axis=-1*, *kind=None*, *order=None*, ***, *stable=None*)

Sort an array in-place. Refer to *numpy.sort* for full documentation.

Parameters

axis
[int, optional] Axis along which to sort. Default is -1, which means sort along the last axis.

kind
[{'quicksort', 'mergesort', 'heapsort', 'stable'}, optional] Sorting algorithm. The default is 'quicksort'. Note that both 'stable' and 'mergesort' use timsort under the covers and, in general, the actual implementation will vary with datatype. The 'mergesort' option is retained for backwards compatibility.

order
[str or list of str, optional] When *a* is an array with fields defined, this argument specifies which fields to compare first, second, etc. A single field can be specified as a string, and not all fields need be specified, but unspecified fields will still be used, in the order in which they come up in the dtype, to break ties.

stable
[bool, optional] Sort stability. If `True`, the returned array will maintain the relative order of `a` values which compare as equal. If `False` or `None`, this is not guaranteed. Internally, this option selects `kind='stable'`. Default: `None`.

Added in version 2.0.0.

See Also

numpy.sort : Return a sorted copy of an array. numpy.argsort : Indirect sort. numpy.lexsort : Indirect stable sort on multiple keys. numpy.searchsorted : Find elements in sorted array. numpy.partition: Partial sort.

Notes

See *numpy.sort* for notes on the different sorting algorithms.

Examples

```
>>> import numpy as np
>>> a = np.array([[1,4], [3,1]])
>>> a.sort(axis=1)
>>> a
```

(continues on next page)

(continued from previous page)

```
array([[1, 4],
       [1, 3]])
>>> a.sort(axis=0)
>>> a
array([[1, 3],
       [1, 4]])
```

Use the *order* keyword to specify a field to use when sorting a structured array:

```
>>> a = np.array([('a', 2), ('c', 1)], dtype=[('x', 'S1'), ('y', int)])
>>> a.sort(order='y')
>>> a
array([(b'c', 1), (b'a', 2)],
      dtype=[('x', 'S1'), ('y', '<i8')])
```

squeeze

FalseArray.**squeeze**(*axis=None*)

Remove axes of length one from *a*.

Refer to *numpy.squeeze* for full documentation.

See Also

numpy.squeeze : equivalent function

std

FalseArray.**std**(*axis=None*, *dtype=None*, *out=None*, *ddof=0*, ***, *keepdims=<no value>*, *where=<no value>*, *mean=<no value>*)

Returns the standard deviation of the array elements along given axis.

Refer to *numpy.std* for full documentation.

See Also

numpy.std : equivalent function

sum

FalseArray.**sum**(*axis=None*, *dtype=None*, *out=None*, ***, *keepdims=<no value>*, *initial=<no value>*, *where=<no value>*)

Return the sum of the array elements over the given axis.

Refer to *numpy.sum* for full documentation.

See Also

numpy.sum : equivalent function

swapaxes

`FalseArray.swapaxes(axis1, axis2, /)`

Return a view of the array with *axis1* and *axis2* interchanged.

Refer to *numpy.swapaxes* for full documentation.

See Also

numpy.swapaxes : equivalent function

take

`FalseArray.take(indices, axis=None, out=None, mode='raise')`

Return an array formed from the elements of *a* at the given indices.

Refer to *numpy.take* for full documentation.

See Also

numpy.take : equivalent function

to_device

`FalseArray.to_device(device, /, *, stream=None)`

For Array API compatibility. Since NumPy only supports CPU arrays, this method is a no-op that returns the same array.

Parameters

device
[“cpu”] Must be `"cpu"`.

stream
[None, optional] Currently unsupported.

Returns

out
[Self] Returns the same array.

tobytes

`FalseArray.tobytes(order='C')`

Construct Python bytes containing the raw data bytes in the array.

Constructs Python bytes showing a copy of the raw contents of data memory. The bytes object is produced in C-order by default. This behavior is controlled by the `order` parameter.

Parameters

order
[{‘C’, ‘F’, ‘A’}, optional] Controls the memory layout of the bytes object. ‘C’ means C-order, ‘F’ means F-order, ‘A’ (short for *Any*) means ‘F’ if *a* is Fortran contiguous, ‘C’ otherwise. Default is ‘C’.

Returns

s
[bytes] Python bytes exhibiting a copy of *a*'s raw data.

See also

frombuffer
Inverse of this operation, construct a 1-dimensional array from Python bytes.

Examples

```
>>> import numpy as np
>>> x = np.array([[0, 1], [2, 3]], dtype='<u2')
>>> x.tobytes()
b'\x00\x00\x01\x00\x02\x00\x03\x00'
>>> x.tobytes('C') == x.tobytes()
True
>>> x.tobytes('F')
b'\x00\x00\x02\x00\x01\x00\x03\x00'
```

tofile

`FalseArray.tofile`(*fid*, */*, *sep=''*, *format='%s'*)

Write array to a file as text or binary (default).

Data is always written in 'C' order, independent of the order of *a*. The data produced by this method can be recovered using the function fromfile().

Parameters

fid
[file or str or Path] An open file object, or a string containing a filename.

sep
[str] Separator between array items for text output. If "" (empty), a binary file is written, equivalent to `file.write(a.tobytes())`.

format
[str] Format string for text file output. Each entry in the array is formatted to text by first converting it to the closest Python type, and then using "format" % item.

Notes

This is a convenience function for quick storage of array data. Information on endianness and precision is lost, so this method is not a good choice for files intended to archive data or transport data between machines with different endianness. Some of these problems can be overcome by outputting the data as text files, at the expense of speed and file size.

When fid is a file object, array contents are directly written to the file, bypassing the file object's `write` method. As a result, tofile cannot be used with files objects supporting compression (e.g., GzipFile) or file-like objects that do not support `fileno()` (e.g., BytesIO).

tolist

FalseArray.**tolist**()

Return the array as an `a.ndim`-levels deep nested list of Python scalars.

Return a copy of the array data as a (nested) Python list. Data items are converted to the nearest compatible builtin Python type, via the *~numpy.ndarray.item* method.

If `a.ndim` is 0, then since the depth of the nested list is 0, it will not be a list at all, but a simple Python scalar.

Parameters

none

Returns

y

[object, or list of object, or list of list of object, or . . .] The possibly nested list of array elements.

Notes

The array may be recreated via `a = np.array(a.tolist())`, although this may sometimes lose precision.

Examples

For a 1D array, `a.tolist()` is almost the same as `list(a)`, except that `tolist` changes numpy scalars to Python scalars:

```
>>> import numpy as np
>>> a = np.uint32([1, 2])
>>> a_list = list(a)
>>> a_list
[np.uint32(1), np.uint32(2)]
>>> type(a_list[0])
<class 'numpy.uint32'>
>>> a_tolist = a.tolist()
>>> a_tolist
[1, 2]
>>> type(a_tolist[0])
<class 'int'>
```

Additionally, for a 2D array, `tolist` applies recursively:

```
>>> a = np.array([[1, 2], [3, 4]])
>>> list(a)
[array([1, 2]), array([3, 4])]
>>> a.tolist()
[[1, 2], [3, 4]]
```

The base case for this recursion is a 0D array:

```
>>> a = np.array(1)
>>> list(a)
```

(continues on next page)

(continued from previous page)

```
Traceback (most recent call last):
  ...
TypeError: iteration over a 0-d array
>>> a.tolist()
1
```

trace

FalseArray.**trace**(*offset=0*, *axis1=0*, *axis2=1*, *dtype=None*, *out=None*)

Return the sum along diagonals of the array.

Refer to *numpy.trace* for full documentation.

See Also

numpy.trace : equivalent function

transpose

FalseArray.**transpose**(**axes*)

Returns a view of the array with axes transposed.

Refer to *numpy.transpose* for full documentation.

Parameters

axes : None, tuple of ints, or *n* ints

- None or no argument: reverses the order of the axes.
- tuple of ints: *i* in the *j*-th place in the tuple means that the array's *i*-th axis becomes the transposed array's *j*-th axis.
- *n* ints: same as an n-tuple of the same ints (this form is intended simply as a "convenience" alternative to the tuple form).

Returns

p

[ndarray] View of the array with its axes suitably permuted.

See Also

transpose : Equivalent function. ndarray.T : Array property returning the array transposed. ndarray.reshape : Give a new shape to an array without changing its data.

Examples

```
>>> import numpy as np
>>> a = np.array([[1, 2], [3, 4]])
>>> a
array([[1, 2],
       [3, 4]])
```

(continues on next page)

(continued from previous page)

```
>>> a.transpose()
array([[1, 3],
       [2, 4]])
>>> a.transpose((1, 0))
array([[1, 3],
       [2, 4]])
>>> a.transpose(1, 0)
array([[1, 3],
       [2, 4]])
```

```
>>> a = np.array([1, 2, 3, 4])
>>> a
array([1, 2, 3, 4])
>>> a.transpose()
array([1, 2, 3, 4])
```

var

FalseArray.**var**(*axis=None*, *dtype=None*, *out=None*, *ddof=0*, ***, *keepdims=<no value>*, *where=<no value>*, *mean=<no value>*)

Returns the variance of the array elements, along given axis.

Refer to *numpy.var* for full documentation.

See Also

numpy.var : equivalent function

view

FalseArray.**view**(*[dtype][, type]*)

New view of array with the same data.

> **Note**
>
> Passing None for `dtype` is different from omitting the parameter, since the former invokes `dtype(None)` which is an alias for `dtype('float64')`.

Parameters

dtype

[data-type or ndarray sub-class, optional] Data-type descriptor of the returned view, e.g., float32 or int16. Omitting it results in the view having the same data-type as *a*. This argument can also be specified as an ndarray sub-class, which then specifies the type of the returned object (this is equivalent to setting the `type` parameter).

type

[Python type, optional] Type of the returned view, e.g., ndarray or matrix. Again, omission of the parameter results in type preservation.

Notes

`a.view()` is used two different ways:

`a.view(some_dtype)` or `a.view(dtype=some_dtype)` constructs a view of the array's memory with a different data-type. This can cause a reinterpretation of the bytes of memory.

`a.view(ndarray_subclass)` or `a.view(type=ndarray_subclass)` just returns an instance of *ndarray_subclass* that looks at the same array (same shape, dtype, etc.) This does not cause a reinterpretation of the memory.

For `a.view(some_dtype)`, if `some_dtype` has a different number of bytes per entry than the previous dtype (for example, converting a regular array to a structured array), then the last axis of `a` must be contiguous. This axis will be resized in the result.

Changed in version 1.23.0: Only the last axis needs to be contiguous. Previously, the entire array had to be C-contiguous.

Examples

```
>>> import numpy as np
>>> x = np.array([(-1, 2)], dtype=[('a', np.int8), ('b', np.int8)])
```

Viewing array data using a different type and dtype:

```
>>> nonneg = np.dtype([("a", np.uint8), ("b", np.uint8)])
>>> y = x.view(dtype=nonneg, type=np.recarray)
>>> x["a"]
array([-1], dtype=int8)
>>> y.a
array([255], dtype=uint8)
```

Creating a view on a structured array so it can be used in calculations

```
>>> x = np.array([(1, 2),(3,4)], dtype=[('a', np.int8), ('b', np.int8)])
>>> xv = x.view(dtype=np.int8).reshape(-1,2)
>>> xv
array([[1, 2],
       [3, 4]], dtype=int8)
>>> xv.mean(0)
array([2.,  3.])
```

Making changes to the view changes the underlying array

```
>>> xv[0,1] = 20
>>> x
array([(1, 20), (3,  4)], dtype=[('a', 'i1'), ('b', 'i1')])
```

Using a view to convert an array to a recarray:

```
>>> z = x.view(np.recarray)
>>> z.a
array([1, 3], dtype=int8)
```

Views share data:

```
>>> x[0] = (9, 10)
>>> z[0]
np.record((9, 10), dtype=[('a', 'i1'), ('b', 'i1')])
```

Views that change the dtype size (bytes per entry) should normally be avoided on arrays defined by slices, transposes, fortran-ordering, etc.:

```
>>> x = np.array([[1, 2, 3], [4, 5, 6]], dtype=np.int16)
>>> y = x[:, ::2]
>>> y
array([[1, 3],
       [4, 6]], dtype=int16)
>>> y.view(dtype=[('width', np.int16), ('length', np.int16)])
Traceback (most recent call last):
    ...
ValueError: To change to a dtype of a different size, the last axis must be␣
↪contiguous
>>> z = y.copy()
>>> z.view(dtype=[('width', np.int16), ('length', np.int16)])
array([[(1, 3)],
       [(4, 6)]], dtype=[('width', '<i2'), ('length', '<i2')])
```

However, views that change dtype are totally fine for arrays with a contiguous last axis, even if the rest of the axes are not C-contiguous:

```
>>> x = np.arange(2 * 3 * 4, dtype=np.int8).reshape(2, 3, 4)
>>> x.transpose(1, 0, 2).view(np.int16)
array([[[ 256,  770],
        [3340, 3854]],

       [[1284, 1798],
        [4368, 4882]],

       [[2312, 2826],
        [5396, 5910]]], dtype=int16)
```

__init__()

Attributes

T	View of the transposed array.
base	Base object if memory is from some other object.
ctypes	An object to simplify the interaction of the array with the ctypes module.
data	Python buffer object pointing to the start of the array's data.
device	
dtype	Data-type of the array's elements.
flags	Information about the memory layout of the array.
flat	A 1-D iterator over the array.
imag	The imaginary part of the array.
itemsize	Length of one array element in bytes.

continues on next page

Table 60 – continued from previous page

mT	View of the matrix transposed array.
nbytes	Total bytes consumed by the elements of the array.
ndim	Number of array dimensions.
real	The real part of the array.
shape	Tuple of array dimensions.
size	Number of elements in the array.
strides	Tuple of bytes to step in each dimension when traversing an array.

T

FalseArray.**T**

View of the transposed array.

Same as `self.transpose()`.

Examples

```
>>> import numpy as np
>>> a = np.array([[1, 2], [3, 4]])
>>> a
array([[1, 2],
       [3, 4]])
>>> a.T
array([[1, 3],
       [2, 4]])
```

```
>>> a = np.array([1, 2, 3, 4])
>>> a
array([1, 2, 3, 4])
>>> a.T
array([1, 2, 3, 4])
```

See Also

transpose

base

FalseArray.**base**

Base object if memory is from some other object.

Examples

The base of an array that owns its memory is None:

```
>>> import numpy as np
>>> x = np.array([1,2,3,4])
>>> x.base is None
True
```

Slicing creates a view, whose memory is shared with x:

```
>>> y = x[2:]
>>> y.base is x
True
```

ctypes

`FalseArray.ctypes`

An object to simplify the interaction of the array with the ctypes module.

This attribute creates an object that makes it easier to use arrays when calling shared libraries with the ctypes module. The returned object has, among others, data, shape, and strides attributes (see Notes below) which themselves return ctypes objects that can be used as arguments to a shared library.

Parameters

None

Returns

c
[Python object] Possessing attributes data, shape, strides, etc.

See Also

numpy.ctypeslib

Notes

Below are the public attributes of this object which were documented in "Guide to NumPy" (we have omitted undocumented public attributes, as well as documented private attributes):

`_ctypes.data`

A pointer to the memory area of the array as a Python integer. This memory area may contain data that is not aligned, or not in correct byte-order. The memory area may not even be writeable. The array flags and data-type of this array should be respected when passing this attribute to arbitrary C-code to avoid trouble that can include Python crashing. User Beware! The value of this attribute is exactly the same as: `self._array_interface_['data'][0]`.

Note that unlike `data_as`, a reference won't be kept to the array: code like `ctypes.c_void_p((a + b).ctypes.data)` will result in a pointer to a deallocated array, and should be spelt `(a + b).ctypes.data_as(ctypes.c_void_p)`

`_ctypes.shape`

(c_intp*self.ndim): A ctypes array of length self.ndim where the basetype is the C-integer corresponding to `dtype('p')` on this platform (see *~numpy.ctypeslib.c_intp*). This base-type could be *ctypes.c_int*, *ctypes.c_long*, or *ctypes.c_longlong* depending on the platform. The ctypes array contains the shape of the underlying array.

`_ctypes.strides`

(c_intp*self.ndim): A ctypes array of length self.ndim where the basetype is the same as for the shape attribute. This ctypes array contains the strides information from the underlying array. This strides information is important for showing how many bytes must be jumped to get to the next element in the array.

_ctypes.**data_as**(*obj*)

Return the data pointer cast to a particular c-types object. For example, calling `self._as_parameter_` is equivalent to `self.data_as(ctypes.c_void_p)`. Perhaps you want to use the data as a pointer to a ctypes array of floating-point data: `self.data_as(ctypes.POINTER(ctypes.c_double))`.

The returned pointer will keep a reference to the array.

_ctypes.**shape_as**(*obj*)

Return the shape tuple as an array of some other c-types type. For example: `self.shape_as(ctypes.c_short)`.

_ctypes.**strides_as**(*obj*)

Return the strides tuple as an array of some other c-types type. For example: `self.strides_as(ctypes.c_longlong)`.

If the ctypes module is not available, then the ctypes attribute of array objects still returns something useful, but ctypes objects are not returned and errors may be raised instead. In particular, the object will still have the `as_parameter` attribute which will return an integer equal to the data attribute.

Examples

```
>>> import numpy as np
>>> import ctypes
>>> x = np.array([[0, 1], [2, 3]], dtype=np.int32)
>>> x
array([[0, 1],
       [2, 3]], dtype=int32)
>>> x.ctypes.data
31962608 # may vary
>>> x.ctypes.data_as(ctypes.POINTER(ctypes.c_uint32))
<__main__.LP_c_uint object at 0x7ff2fc1fc200> # may vary
>>> x.ctypes.data_as(ctypes.POINTER(ctypes.c_uint32)).contents
c_uint(0)
>>> x.ctypes.data_as(ctypes.POINTER(ctypes.c_uint64)).contents
c_ulong(4294967296)
>>> x.ctypes.shape
<numpy._core._internal.c_long_Array_2 object at 0x7ff2fc1fce60> # may vary
>>> x.ctypes.strides
<numpy._core._internal.c_long_Array_2 object at 0x7ff2fc1ff320> # may vary
```

data

FalseArray.**data**

Python buffer object pointing to the start of the array's data.

device

FalseArray.**device**

dtype

`FalseArray.dtype`

Data-type of the array's elements.

Warning

Setting `arr.dtype` is discouraged and may be deprecated in the future. Setting will replace the `dtype` without modifying the memory (see also *ndarray.view* and *ndarray.astype*).

Parameters

None

Returns

d : numpy dtype object

See Also

ndarray.astype : Cast the values contained in the array to a new data-type. ndarray.view : Create a view of the same data but a different data-type. numpy.dtype

Examples

```
>>> import numpy as np
>>> x = np.arange(4).reshape((2, 2))
>>> x
array([[0, 1],
       [2, 3]])
>>> x.dtype
dtype('int64')   # may vary (OS, bitness)
>>> isinstance(x.dtype, np.dtype)
True
```

flags

`FalseArray.flags`

Information about the memory layout of the array.

Attributes

C_CONTIGUOUS (C)
The data is in a single, C-style contiguous segment.

F_CONTIGUOUS (F)
The data is in a single, Fortran-style contiguous segment.

OWNDATA (O)
The array owns the memory it uses or borrows it from another object.

WRITEABLE (W)
The data area can be written to. Setting this to False locks the data, making it read-only. A view

(slice, etc.) inherits WRITEABLE from its base array at creation time, but a view of a writeable array may be subsequently locked while the base array remains writeable. (The opposite is not true, in that a view of a locked array may not be made writeable. However, currently, locking a base object does not lock any views that already reference it, so under that circumstance it is possible to alter the contents of a locked array via a previously created writeable view onto it.) Attempting to change a non-writeable array raises a RuntimeError exception.

ALIGNED (A)
The data and all elements are aligned appropriately for the hardware.

WRITEBACKIFCOPY (X)
This array is a copy of some other array. The C-API function PyArray_ResolveWritebackIfCopy must be called before deallocating to the base array will be updated with the contents of this array.

FNC
F_CONTIGUOUS and not C_CONTIGUOUS.

FORC
F_CONTIGUOUS or C_CONTIGUOUS (one-segment test).

BEHAVED (B)
ALIGNED and WRITEABLE.

CARRAY (CA)
BEHAVED and C_CONTIGUOUS.

FARRAY (FA)
BEHAVED and F_CONTIGUOUS and not C_CONTIGUOUS.

Notes

The *flags* object can be accessed dictionary-like (as in `a.flags['WRITEABLE']`), or by using lowercased attribute names (as in `a.flags.writeable`). Short flag names are only supported in dictionary access.

Only the WRITEBACKIFCOPY, WRITEABLE, and ALIGNED flags can be changed by the user, via direct assignment to the attribute or dictionary entry, or by calling *ndarray.setflags*.

The array flags cannot be set arbitrarily:

- WRITEBACKIFCOPY can only be set `False`.
- ALIGNED can only be set `True` if the data is truly aligned.
- WRITEABLE can only be set `True` if the array owns its own memory or the ultimate owner of the memory exposes a writeable buffer interface or is a string.

Arrays can be both C-style and Fortran-style contiguous simultaneously. This is clear for 1-dimensional arrays, but can also be true for higher dimensional arrays.

Even for contiguous arrays a stride for a given dimension `arr.strides[dim]` may be *arbitrary* if `arr.shape[dim] == 1` or the array has no elements. It does *not* generally hold that `self.strides[-1] == self.itemsize` for C-style contiguous arrays or `self.strides[0] == self.itemsize` for Fortran-style contiguous arrays is true.

flat

`FalseArray.flat`

A 1-D iterator over the array.

This is a *numpy.flatiter* instance, which acts similarly to, but is not a subclass of, Python's built-in iterator object.

See Also

flatten : Return a copy of the array collapsed into one dimension.

flatiter

Examples

```
>>> import numpy as np
>>> x = np.arange(1, 7).reshape(2, 3)
>>> x
array([[1, 2, 3],
       [4, 5, 6]])
>>> x.flat[3]
4
>>> x.T
array([[1, 4],
       [2, 5],
       [3, 6]])
>>> x.T.flat[3]
5
>>> type(x.flat)
<class 'numpy.flatiter'>
```

An assignment example:

```
>>> x.flat = 3; x
array([[3, 3, 3],
       [3, 3, 3]])
>>> x.flat[[1,4]] = 1; x
array([[3, 1, 3],
       [3, 1, 3]])
```

imag

`FalseArray.imag`

The imaginary part of the array.

Examples

```
>>> import numpy as np
>>> x = np.sqrt([1+0j, 0+1j])
>>> x.imag
array([ 0.        ,  0.70710678])
>>> x.imag.dtype
dtype('float64')
```

itemsize

`FalseArray.itemsize`

Length of one array element in bytes.

Examples

```
>>> import numpy as np
>>> x = np.array([1,2,3], dtype=np.float64)
>>> x.itemsize
8
>>> x = np.array([1,2,3], dtype=np.complex128)
>>> x.itemsize
16
```

mT

`FalseArray.mT`

View of the matrix transposed array.

The matrix transpose is the transpose of the last two dimensions, even if the array is of higher dimension.

Added in version 2.0.

Raises

ValueError
If the array is of dimension less than 2.

Examples

```
>>> import numpy as np
>>> a = np.array([[1, 2], [3, 4]])
>>> a
array([[1, 2],
       [3, 4]])
>>> a.mT
array([[1, 3],
       [2, 4]])
```

```
>>> a = np.arange(8).reshape((2, 2, 2))
>>> a
array([[[0, 1],
        [2, 3]],

       [[4, 5],
        [6, 7]]])
>>> a.mT
array([[[0, 2],
        [1, 3]],

       [[4, 6],
        [5, 7]]])
```

nbytes

`FalseArray.nbytes`

Total bytes consumed by the elements of the array.

Notes

Does not include memory consumed by non-element attributes of the array object.

See Also

sys.getsizeof
Memory consumed by the object itself without parents in case view. This does include memory consumed by non-element attributes.

Examples

```
>>> import numpy as np
>>> x = np.zeros((3,5,2), dtype=np.complex128)
>>> x.nbytes
480
>>> np.prod(x.shape) * x.itemsize
480
```

ndim

`FalseArray.ndim`

Number of array dimensions.

Examples

```
>>> import numpy as np
>>> x = np.array([1, 2, 3])
>>> x.ndim
1
>>> y = np.zeros((2, 3, 4))
>>> y.ndim
3
```

real

`FalseArray.real`

The real part of the array.

Examples

```
>>> import numpy as np
>>> x = np.sqrt([1+0j, 0+1j])
>>> x.real
array([ 1.        ,  0.70710678])
>>> x.real.dtype
dtype('float64')
```

See Also

numpy.real : equivalent function

shape

`FalseArray.shape`

Tuple of array dimensions.

The shape property is usually used to get the current shape of an array, but may also be used to reshape the array in-place by assigning a tuple of array dimensions to it. As with *numpy.reshape*, one of the new shape dimensions can be -1, in which case its value is inferred from the size of the array and the remaining dimensions. Reshaping an array in-place will fail if a copy is required.

> **Warning**
>
> Setting `arr.shape` is discouraged and may be deprecated in the future. Using *ndarray.reshape* is the preferred approach.

Examples

```
>>> import numpy as np
>>> x = np.array([1, 2, 3, 4])
>>> x.shape
(4,)
>>> y = np.zeros((2, 3, 4))
>>> y.shape
(2, 3, 4)
>>> y.shape = (3, 8)
>>> y
array([[ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.]])
>>> y.shape = (3, 6)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: cannot reshape array of size 24 into shape (3,6)
>>> np.zeros((4,2))[::2].shape = (-1,)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: Incompatible shape for in-place modification. Use
`.reshape()` to make a copy with the desired shape.
```

See Also

numpy.shape : Equivalent getter function. numpy.reshape : Function similar to setting `shape`. ndarray.reshape : Method similar to setting `shape`.

size

`FalseArray.size`

Number of elements in the array.

Equal to `np.prod(a.shape)`, i.e., the product of the array's dimensions.

Notes

a.size returns a standard arbitrary precision Python integer. This may not be the case with other methods of obtaining the same value (like the suggested `np.prod(a.shape)`, which returns an instance of `np.int_`), and may be relevant if the value is used further in calculations that may overflow a fixed size integer type.

Examples

```
>>> import numpy as np
>>> x = np.zeros((3, 5, 2), dtype=np.complex128)
>>> x.size
30
>>> np.prod(x.shape)
30
```

strides

`FalseArray.strides`

Tuple of bytes to step in each dimension when traversing an array.

The byte offset of element `(i[0], i[1], ..., i[n])` in an array *a* is:

```
offset = sum(np.array(i) * a.strides)
```

A more detailed explanation of strides can be found in arrays.ndarray.

> **Warning**
>
> Setting `arr.strides` is discouraged and may be deprecated in the future. *numpy.lib.stride_tricks.as_strided* should be preferred to create a new view of the same data in a safer way.

Notes

Imagine an array of 32-bit integers (each 4 bytes):

```
x = np.array([[0, 1, 2, 3, 4],
              [5, 6, 7, 8, 9]], dtype=np.int32)
```

This array is stored in memory as 40 bytes, one after the other (known as a contiguous block of memory). The strides of an array tell us how many bytes we have to skip in memory to move to the next position along a certain axis. For example, we have to skip 4 bytes (1 value) to move to the next column, but 20 bytes (5 values) to get to the same position in the next row. As such, the strides for the array *x* will be `(20, 4)`.

See Also

numpy.lib.stride_tricks.as_strided

Examples

```
>>> import numpy as np
>>> y = np.reshape(np.arange(2 * 3 * 4, dtype=np.int32), (2, 3, 4))
>>> y
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]],
       [[12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23]]], dtype=np.int32)
>>> y.strides
(48, 16, 4)
>>> y[1, 1, 1]
np.int32(17)
>>> offset = sum(y.strides * np.array((1, 1, 1)))
>>> offset // y.itemsize
np.int64(17)
```

```
>>> x = np.reshape(np.arange(5*6*7*8, dtype=np.int32), (5, 6, 7, 8))
>>> x = x.transpose(2, 3, 1, 0)
>>> x.strides
(32, 4, 224, 1344)
>>> i = np.array([3, 5, 2, 2], dtype=np.int32)
>>> offset = sum(i * x.strides)
>>> x[3, 5, 2, 2]
np.int32(813)
>>> offset // x.itemsize
np.int64(813)
```

TrueArray

`class TrueArray`(*shape*, *dtype=None*, *buffer=None*, *offset=0*, *strides=None*, *order=None*)

Methods

__init__	
all	Returns True if all elements evaluate to True.
any	Returns True if any of the elements of *a* evaluate to True.
argmax	Return indices of the maximum values along the given axis.
argmin	Return indices of the minimum values along the given axis.
argpartition	Returns the indices that would partition this array.
argsort	Returns the indices that would sort this array.
astype	Copy of the array, cast to a specified type.

continues on next page

Table 61 – continued from previous page

byteswap	Swap the bytes of the array elements
choose	Use an index array to construct a new array from a set of choices.
clip	Return an array whose values are limited to `[min, max]`.
collapse	
compress	Return selected slices of this array along given axis.
conj	Complex-conjugate all elements.
conjugate	Return the complex conjugate, element-wise.
copy	Return a copy of the array.
cumprod	Return the cumulative product of the elements along the given axis.
cumsum	Return the cumulative sum of the elements along the given axis.
diagonal	Return specified diagonals.
dot	Refer to `numpy.dot()` for full documentation.
dump	Dump a pickle of the array to the specified file.
dumps	Returns the pickle of the array as a string.
fill	Fill the array with a scalar value.
flatten	Return a copy of the array collapsed into one dimension.
getfield	Returns a field of the given array as a certain type.
item	Copy an element of an array to a standard Python scalar and return it.
max	Return the maximum along a given axis.
mean	Returns the average of the array elements along given axis.
min	Return the minimum along a given axis.
nonzero	Return the indices of the elements that are non-zero.
partition	Partially sorts the elements in the array in such a way that the value of the element in k-th position is in the position it would be in a sorted array.
prod	Return the product of the array elements over the given axis
put	Set `a.flat[n] = values[n]` for all `n` in indices.
ravel	Return a flattened array.
repeat	Repeat elements of an array.
reshape	Returns an array containing the same data with a new shape.
resize	Change shape and size of array in-place.
round	Return *a* with each element rounded to the given number of decimals.
searchsorted	Find indices where elements of *v* should be inserted in *a* to maintain order.
setfield	Put a value into a specified place in a field defined by a data-type.
setflags	Set array flags WRITEABLE, ALIGNED, WRITEBACKIFCOPY, respectively.
sort	Sort an array in-place.
squeeze	Remove axes of length one from *a*.
std	Returns the standard deviation of the array elements along given axis.

continues on next page

Table 61 – continued from previous page

`sum`	Return the sum of the array elements over the given axis.
`swapaxes`	Return a view of the array with *axis1* and *axis2* interchanged.
`take`	Return an array formed from the elements of *a* at the given indices.
`to_device`	For Array API compatibility.
`tobytes`	Construct Python bytes containing the raw data bytes in the array.
`tofile`	Write array to a file as text or binary (default).
`tolist`	Return the array as an `a.ndim`-levels deep nested list of Python scalars.
`trace`	Return the sum along diagonals of the array.
`transpose`	Returns a view of the array with axes transposed.
`var`	Returns the variance of the array elements, along given axis.
`view`	New view of array with the same data.

__init__

`TrueArray.__init__()`

all

`TrueArray.all(`*axis=None*, *out=None*, *, *keepdims=<no value>*, *where=<no value>*`)`

Returns True if all elements evaluate to True.

Refer to *numpy.all* for full documentation.

See Also

numpy.all : equivalent function

any

`TrueArray.any(`*axis=None*, *out=None*, *, *keepdims=<no value>*, *where=<no value>*`)`

Returns True if any of the elements of *a* evaluate to True.

Refer to *numpy.any* for full documentation.

See Also

numpy.any : equivalent function

argmax

`TrueArray.argmax(`*axis=None*, *out=None*, *, *keepdims=False*`)`

Return indices of the maximum values along the given axis.

Refer to *numpy.argmax* for full documentation.

See Also

numpy.argmax : equivalent function

argmin

`TrueArray.argmin`(*axis=None*, *out=None*, *, *keepdims=False*)

Return indices of the minimum values along the given axis.

Refer to *numpy.argmin* for detailed documentation.

See Also

numpy.argmin : equivalent function

argpartition

`TrueArray.argpartition`(*kth*, *axis=-1*, *kind='introselect'*, *order=None*)

Returns the indices that would partition this array.

Refer to *numpy.argpartition* for full documentation.

See Also

numpy.argpartition : equivalent function

argsort

`TrueArray.argsort`(*axis=-1*, *kind=None*, *order=None*, *, *stable=None*)

Returns the indices that would sort this array.

Refer to *numpy.argsort* for full documentation.

See Also

numpy.argsort : equivalent function

astype

`TrueArray.astype`(*dtype*, *order='K'*, *casting='unsafe'*, *subok=True*, *copy=True*)

Copy of the array, cast to a specified type.

Parameters

dtype
: [str or dtype] Typecode or data-type to which the array is cast.

order
: [{'C', 'F', 'A', 'K'}, optional] Controls the memory layout order of the result. 'C' means C order, 'F' means Fortran order, 'A' means 'F' order if all the arrays are Fortran contiguous, 'C' order otherwise, and 'K' means as close to the order the array elements appear in memory as possible. Default is 'K'.

casting
: [{'no', 'equiv', 'safe', 'same_kind', 'same_value', 'unsafe'}, optional] Controls what kind of data casting may occur. Defaults to 'unsafe' for backwards compatibility.

- 'no' means the data types should not be cast at all.
- 'equiv' means only byte-order changes are allowed.
- 'safe' means only casts which can preserve values are allowed.
- 'same_kind' means only safe casts or casts within a kind, like float64 to float32, are allowed.
- 'unsafe' means any data conversions may be done.
- 'same_value' means any data conversions may be done, but the values must not change, including rounding of floats or overflow of ints

Added in version 2.4: Support for `'same_value'` was added.

subok
[bool, optional] If True, then sub-classes will be passed-through (default), otherwise the returned array will be forced to be a base-class array.

copy
[bool, optional] By default, astype always returns a newly allocated array. If this is set to false, and the *dtype*, *order*, and *subok* requirements are satisfied, the input array is returned instead of a copy.

Returns

arr_t
[ndarray] Unless *copy* is False and the other conditions for returning the input array are satisfied (see description for *copy* input parameter), *arr_t* is a new array of the same shape as the input array, with dtype, order given by *dtype*, *order*.

Raises

ComplexWarning
When casting from complex to float or int. To avoid this, one should use `a.real.astype(t)`.

ValueError
When casting using `'same_value'` and the values change or would overflow

Examples

```
>>> import numpy as np
>>> x = np.array([1, 2, 2.5])
>>> x
array([1. ,  2. ,  2.5])
```

```
>>> x.astype(int)
array([1, 2, 2])
```

```
>>> x.astype(int, casting="same_value")
Traceback (most recent call last):
...
ValueError: could not cast 'same_value' double to long
```

```
>>> x[:2].astype(int, casting="same_value")
array([1, 2])
```

byteswap

`TrueArray.byteswap(inplace=False)`

Swap the bytes of the array elements

Toggle between low-endian and big-endian data representation by returning a byteswapped array, optionally swapped in-place. Arrays of byte-strings are not swapped. The real and imaginary parts of a complex number are swapped individually.

Parameters

inplace
[bool, optional] If `True`, swap bytes in-place, default is `False`.

Returns

out
[ndarray] The byteswapped array. If *inplace* is `True`, this is a view to self.

Examples

```
>>> import numpy as np
>>> A = np.array([1, 256, 8755], dtype=np.int16)
>>> list(map(hex, A))
['0x1', '0x100', '0x2233']
>>> A.byteswap(inplace=True)
array([  256,     1, 13090], dtype=int16)
>>> list(map(hex, A))
['0x100', '0x1', '0x3322']
```

Arrays of byte-strings are not swapped

```
>>> A = np.array([b'ceg', b'fac'])
>>> A.byteswap()
array([b'ceg', b'fac'], dtype='|S3')
```

`A.view(A.dtype.newbyteorder()).byteswap()` produces an array with the same values but different representation in memory

```
>>> A = np.array([1, 2, 3],dtype=np.int64)
>>> A.view(np.uint8)
array([1, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0,
       0, 0], dtype=uint8)
>>> A.view(A.dtype.newbyteorder()).byteswap(inplace=True)
array([1, 2, 3], dtype='>i8')
>>> A.view(np.uint8)
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0,
       0, 3], dtype=uint8)
```

choose

`TrueArray.choose(choices, out=None, mode='raise')`

Use an index array to construct a new array from a set of choices.

Refer to *numpy.choose* for full documentation.

See Also

numpy.choose : equivalent function

clip

TrueArray.**clip**(*min=<no value>*, *max=<no value>*, *out=None*, ***kwargs*)

Return an array whose values are limited to `[min, max]`. One of max or min must be given.

Refer to *numpy.clip* for full documentation.

See Also

numpy.clip : equivalent function

collapse

TrueArray.**collapse**(*shape*)

compress

TrueArray.**compress**(*condition*, *axis=None*, *out=None*)

Return selected slices of this array along given axis.

Refer to *numpy.compress* for full documentation.

See Also

numpy.compress : equivalent function

conj

TrueArray.**conj**()

Complex-conjugate all elements.

Refer to *numpy.conjugate* for full documentation.

See Also

numpy.conjugate : equivalent function

conjugate

TrueArray.**conjugate**()

Return the complex conjugate, element-wise.

Refer to *numpy.conjugate* for full documentation.

See Also

numpy.conjugate : equivalent function

copy

TrueArray.**copy**(*order='C'*)

Return a copy of the array.

Parameters

order
[{'C', 'F', 'A', 'K'}, optional] Controls the memory layout of the copy. 'C' means C-order, 'F' means F-order, 'A' means 'F' if *a* is Fortran contiguous, 'C' otherwise. 'K' means match the layout of *a* as closely as possible. (Note that this function and `numpy.copy()` are very similar but have different default values for their order= arguments, and this function always passes sub-classes through.)

See also

numpy.copy : Similar function with different default behavior numpy.copyto

Notes

This function is the preferred method for creating an array copy. The function `numpy.copy()` is similar, but it defaults to using order 'K', and will not pass sub-classes through by default.

Examples

```
>>> import numpy as np
>>> x = np.array([[1,2,3],[4,5,6]], order='F')
```

```
>>> y = x.copy()
```

```
>>> x.fill(0)
```

```
>>> x
array([[0, 0, 0],
       [0, 0, 0]])
```

```
>>> y
array([[1, 2, 3],
       [4, 5, 6]])
```

```
>>> y.flags['C_CONTIGUOUS']
True
```

For arrays containing Python objects (e.g. dtype=object), the copy is a shallow one. The new array will contain the same object which may lead to surprises if that object can be modified (is mutable):

```
>>> a = np.array([1, 'm', [2, 3, 4]], dtype=object)
>>> b = a.copy()
>>> b[2][0] = 10
>>> a
array([1, 'm', list([10, 3, 4])], dtype=object)
```

To ensure all elements within an `object` array are copied, use *copy.deepcopy*:

```
>>> import copy
>>> a = np.array([1, 'm', [2, 3, 4]], dtype=object)
>>> c = copy.deepcopy(a)
>>> c[2][0] = 10
>>> c
array([1, 'm', list([10, 3, 4])], dtype=object)
>>> a
array([1, 'm', list([2, 3, 4])], dtype=object)
```

cumprod

`TrueArray.cumprod`(*axis=None*, *dtype=None*, *out=None*)

Return the cumulative product of the elements along the given axis.

Refer to *numpy.cumprod* for full documentation.

See Also

numpy.cumprod : equivalent function

cumsum

`TrueArray.cumsum`(*axis=None*, *dtype=None*, *out=None*)

Return the cumulative sum of the elements along the given axis.

Refer to *numpy.cumsum* for full documentation.

See Also

numpy.cumsum : equivalent function

diagonal

`TrueArray.diagonal`(*offset=0*, *axis1=0*, *axis2=1*)

Return specified diagonals. In NumPy 1.9 the returned array is a read-only view instead of a copy as in previous NumPy versions. In a future version the read-only restriction will be removed.

Refer to `numpy.diagonal()` for full documentation.

See Also

numpy.diagonal : equivalent function

dot

`TrueArray.dot`(*other*, */*, *out=None*)

Refer to `numpy.dot()` for full documentation.

See Also

numpy.dot : equivalent function

dump

TrueArray.**dump**(*file*)

Dump a pickle of the array to the specified file. The array can be read back with pickle.load or numpy.load.

Parameters

file
[str or Path] A string naming the dump file.

dumps

TrueArray.**dumps**()

Returns the pickle of the array as a string. `pickle.loads` will convert the string back to an array.

Parameters

None

fill

TrueArray.**fill**(*value*)

Fill the array with a scalar value.

Parameters

value
[scalar] All elements of *a* will be assigned this value.

Examples

```
>>> import numpy as np
>>> a = np.array([1, 2])
>>> a.fill(0)
>>> a
array([0, 0])
>>> a = np.empty(2)
>>> a.fill(1)
>>> a
array([1.,  1.])
```

Fill expects a scalar value and always behaves the same as assigning to a single array element. The following is a rare example where this distinction is important:

```
>>> a = np.array([None, None], dtype=object)
>>> a[0] = np.array(3)
>>> a
array([array(3), None], dtype=object)
>>> a.fill(np.array(3))
>>> a
array([array(3), array(3)], dtype=object)
```

Where other forms of assignments will unpack the array being assigned:

```
>>> a[...] = np.array(3)
>>> a
array([3, 3], dtype=object)
```

flatten

TrueArray.**flatten**(*order='C'*)

Return a copy of the array collapsed into one dimension.

Parameters

order

[{'C', 'F', 'A', 'K'}, optional] 'C' means to flatten in row-major (C-style) order. 'F' means to flatten in column-major (Fortran- style) order. 'A' means to flatten in column-major order if *a* is Fortran *contiguous* in memory, row-major order otherwise. 'K' means to flatten *a* in the order the elements occur in memory. The default is 'C'.

Returns

y

[ndarray] A copy of the input array, flattened to one dimension.

See Also

ravel : Return a flattened array. flat : A 1-D flat iterator over the array.

Examples

```
>>> import numpy as np
>>> a = np.array([[1,2], [3,4]])
>>> a.flatten()
array([1, 2, 3, 4])
>>> a.flatten('F')
array([1, 3, 2, 4])
```

getfield

TrueArray.**getfield**(*dtype*, *offset=0*)

Returns a field of the given array as a certain type.

A field is a view of the array data with a given data-type. The values in the view are determined by the given type and the offset into the current array in bytes. The offset needs to be such that the view dtype fits in the array dtype; for example an array of dtype complex128 has 16-byte elements. If taking a view with a 32-bit integer (4 bytes), the offset needs to be between 0 and 12 bytes.

Parameters

dtype

[str or dtype] The data type of the view. The dtype size of the view can not be larger than that of the array itself.

offset

[int] Number of bytes to skip before beginning the element view.

Examples

```
>>> import numpy as np
>>> x = np.diag([1.+1.j]*2)
>>> x[1, 1] = 2 + 4.j
>>> x
array([[1.+1.j,  0.+0.j],
       [0.+0.j,  2.+4.j]])
>>> x.getfield(np.float64)
array([[1.,  0.],
       [0.,  2.]])
```

By choosing an offset of 8 bytes we can select the complex part of the array for our view:

```
>>> x.getfield(np.float64, offset=8)
array([[1.,  0.],
       [0.,  4.]])
```

item

TrueArray.**item**(**args*)

Copy an element of an array to a standard Python scalar and return it.

Parameters

*args : Arguments (variable number and type)

- none: in this case, the method only works for arrays with one element (*a.size == 1*), which element is copied into a standard Python scalar object and returned.
- int_type: this argument is interpreted as a flat index into the array, specifying which element to copy and return.
- tuple of int_types: functions as does a single int_type argument, except that the argument is interpreted as an nd-index into the array.

Returns

z

[Standard Python scalar object] A copy of the specified element of the array as a suitable Python scalar

Notes

When the data type of *a* is longdouble or clongdouble, item() returns a scalar array object because there is no available Python scalar that would not lose information. Void arrays return a buffer object for item(), unless fields are defined, in which case a tuple is returned.

item is very similar to a[args], except, instead of an array scalar, a standard Python scalar is returned. This can be useful for speeding up access to elements of the array and doing arithmetic on elements of the array using Python's optimized math.

Examples

```
>>> import numpy as np
>>> np.random.seed(123)
>>> x = np.random.randint(9, size=(3, 3))
>>> x
array([[2, 2, 6],
       [1, 3, 6],
       [1, 0, 1]])
>>> x.item(3)
1
>>> x.item(7)
0
>>> x.item((0, 1))
2
>>> x.item((2, 2))
1
```

For an array with object dtype, elements are returned as-is.

```
>>> a = np.array([np.int64(1)], dtype=object)
>>> a.item() #return np.int64
np.int64(1)
```

max

`TrueArray.max`(*axis=None*, *out=None*, *, *keepdims=<no value>*, *initial=<no value>*, *where=<no value>*)

Return the maximum along a given axis.

Refer to *numpy.amax* for full documentation.

See Also

numpy.amax : equivalent function

mean

`TrueArray.mean`(*axis=None*, *dtype=None*, *out=None*, *, *keepdims=<no value>*, *where=<no value>*)

Returns the average of the array elements along given axis.

Refer to *numpy.mean* for full documentation.

See Also

numpy.mean : equivalent function

min

`TrueArray.min`(*axis=None*, *out=None*, *, *keepdims=<no value>*, *initial=<no value>*, *where=<no value>*)

Return the minimum along a given axis.

Refer to *numpy.amin* for full documentation.

See Also

numpy.amin : equivalent function

nonzero

`TrueArray.nonzero()`

Return the indices of the elements that are non-zero.

Refer to *numpy.nonzero* for full documentation.

See Also

numpy.nonzero : equivalent function

partition

`TrueArray.partition`(*kth*, *axis=-1*, *kind='introselect'*, *order=None*)

Partially sorts the elements in the array in such a way that the value of the element in k-th position is in the position it would be in a sorted array. In the output array, all elements smaller than the k-th element are located to the left of this element and all equal or greater are located to its right. The ordering of the elements in the two partitions on the either side of the k-th element in the output array is undefined.

Parameters

kth

[int or sequence of ints] Element index to partition by. The kth element value will be in its final sorted position and all smaller elements will be moved before it and all equal or greater elements behind it. The order of all elements in the partitions is undefined. If provided with a sequence of kth it will partition all elements indexed by kth of them into their sorted position at once.

Deprecated since version 1.22.0: Passing booleans as index is deprecated.

axis

[int, optional] Axis along which to sort. Default is -1, which means sort along the last axis.

kind

[{'introselect'}, optional] Selection algorithm. Default is 'introselect'.

order

[str or list of str, optional] When *a* is an array with fields defined, this argument specifies which fields to compare first, second, etc. A single field can be specified as a string, and not all fields need to be specified, but unspecified fields will still be used, in the order in which they come up in the dtype, to break ties.

See Also

numpy.partition : Return a partitioned copy of an array. argpartition : Indirect partition. sort : Full sort.

Notes

See `np.partition` for notes on the different algorithms.

Examples

```
>>> import numpy as np
>>> a = np.array([3, 4, 2, 1])
>>> a.partition(3)
>>> a
array([2, 1, 3, 4]) # may vary
```

```
>>> a.partition((1, 3))
>>> a
array([1, 2, 3, 4])
```

prod

`TrueArray.prod`(*axis=None*, *dtype=None*, *out=None*, ***, *keepdims=<no value>*, *initial=<no value>*, *where=<no value>*)

Return the product of the array elements over the given axis

Refer to *numpy.prod* for full documentation.

See Also

numpy.prod : equivalent function

put

`TrueArray.put`(*indices*, *values*, *mode='raise'*)

Set `a.flat[n] = values[n]` for all `n` in indices.

Refer to *numpy.put* for full documentation.

See Also

numpy.put : equivalent function

ravel

`TrueArray.ravel`(*order='C'*)

Return a flattened array.

Refer to *numpy.ravel* for full documentation.

See Also

numpy.ravel : equivalent function ndarray.flat : a flat iterator on the array.

repeat

`TrueArray.repeat`(*repeats*, *axis=None*)

Repeat elements of an array.

Refer to *numpy.repeat* for full documentation.

See Also

numpy.repeat : equivalent function

reshape

`TrueArray.reshape(`*shape*, /, *, *order='C'*, *copy=None*`)`

`TrueArray.reshape(`**shape*, *order='C'*, *copy=None*`)`

Returns an array containing the same data with a new shape.

Refer to *numpy.reshape* for full documentation.

See Also

numpy.reshape : equivalent function

Notes

Unlike the free function *numpy.reshape*, this method on *ndarray* allows the elements of the shape parameter to be passed in as separate arguments. For example, `a.reshape(4, 2)` is equivalent to `a.reshape((4, 2))`.

resize

`TrueArray.resize(`*new_shape*, /, *, *refcheck=True*`)`

`TrueArray.resize(`**new_shape*, *refcheck=True*`)`

Change shape and size of array in-place.

Parameters

new_shape
[tuple of ints, or *n* ints] Shape of resized array.

refcheck
[bool, optional] If False, reference count will not be checked. Default is True.

Returns

None

Raises

ValueError
If *a* does not own its own data or references or views to it exist, and the data memory must be changed. PyPy only: will always raise if the data memory must be changed, since there is no reliable way to determine if references or views to it exist.

SystemError
If the *order* keyword argument is specified. This behaviour is a bug in NumPy.

See Also

resize : Return a new array with the specified shape.

Notes

This reallocates space for the data area if necessary.

Only contiguous arrays (data elements consecutive in memory) can be resized.

The purpose of the reference count check is to make sure you do not use this array as a buffer for another Python object and then reallocate the memory. However, reference counts can increase in other ways so if you are sure that you have not shared the memory for this array with another Python object, then you may safely set *refcheck* to False.

Examples

Shrinking an array: array is flattened (in the order that the data are stored in memory), resized, and reshaped:

```
>>> import numpy as np
```

```
>>> a = np.array([[0, 1], [2, 3]], order='C')
>>> a.resize((2, 1))
>>> a
array([[0],
       [1]])
```

```
>>> a = np.array([[0, 1], [2, 3]], order='F')
>>> a.resize((2, 1))
>>> a
array([[0],
       [2]])
```

Enlarging an array: as above, but missing entries are filled with zeros:

```
>>> b = np.array([[0, 1], [2, 3]])
>>> b.resize(2, 3) # new_shape parameter doesn't have to be a tuple
>>> b
array([[0, 1, 2],
       [3, 0, 0]])
```

Referencing an array prevents resizing...

```
>>> c = a
>>> a.resize((1, 1))
Traceback (most recent call last):
...
ValueError: cannot resize an array that references or is referenced ...
```

Unless *refcheck* is False:

```
>>> a.resize((1, 1), refcheck=False)
>>> a
array([[0]])
>>> c
array([[0]])
```

round

`TrueArray.round(decimals=0, out=None)`

Return *a* with each element rounded to the given number of decimals.

Refer to *numpy.around* for full documentation.

See Also

numpy.around : equivalent function

searchsorted

`TrueArray.searchsorted(v, side='left', sorter=None)`

Find indices where elements of *v* should be inserted in *a* to maintain order.

For full documentation, see *numpy.searchsorted*.

See Also

numpy.searchsorted : equivalent function

setfield

`TrueArray.setfield(val, dtype, offset=0)`

Put a value into a specified place in a field defined by a data-type.

Place *val* into *a*'s field defined by *dtype* and beginning *offset* bytes into the field.

Parameters

val
: [object] Value to be placed in field.

dtype
: [dtype object] Data-type of the field in which to place *val*.

offset
: [int, optional] The number of bytes into the field at which to place *val*.

Returns

None

See Also

getfield

Examples

```
>>> import numpy as np
>>> x = np.eye(3)
>>> x.getfield(np.float64)
array([[1.,  0.,  0.],
       [0.,  1.,  0.],
```

(continues on next page)

(continued from previous page)

```
       [0.,  0.,  1.]])
>>> x.setfield(3, np.int32)
>>> x.getfield(np.int32)
array([[3, 3, 3],
       [3, 3, 3],
       [3, 3, 3]], dtype=int32)
>>> x
array([[1.0e+000, 1.5e-323, 1.5e-323],
       [1.5e-323, 1.0e+000, 1.5e-323],
       [1.5e-323, 1.5e-323, 1.0e+000]])
>>> x.setfield(np.eye(3), np.int32)
>>> x
array([[1.,  0.,  0.],
       [0.,  1.,  0.],
       [0.,  0.,  1.]])
```

setflags

`TrueArray.setflags(`*write=None*, *align=None*, *uic=None*`)`

Set array flags WRITEABLE, ALIGNED, WRITEBACKIFCOPY, respectively.

These Boolean-valued flags affect how numpy interprets the memory area used by *a* (see Notes below). The ALIGNED flag can only be set to True if the data is actually aligned according to the type. The WRITEBACKIFCOPY flag can never be set to True. The flag WRITEABLE can only be set to True if the array owns its own memory, or the ultimate owner of the memory exposes a writeable buffer interface, or is a string. (The exception for string is made so that unpickling can be done without copying memory.)

Parameters

write
: [bool, optional] Describes whether or not *a* can be written to.

align
: [bool, optional] Describes whether or not *a* is aligned properly for its type.

uic
: [bool, optional] Describes whether or not *a* is a copy of another "base" array.

Notes

Array flags provide information about how the memory area used for the array is to be interpreted. There are 7 Boolean flags in use, only three of which can be changed by the user: WRITEBACKIFCOPY, WRITEABLE, and ALIGNED.

WRITEABLE (W) the data area can be written to;

ALIGNED (A) the data and strides are aligned appropriately for the hardware (as determined by the compiler);

WRITEBACKIFCOPY (X) this array is a copy of some other array (referenced by .base). When the C-API function PyArray_ResolveWritebackIfCopy is called, the base array will be updated with the contents of this array.

All flags can be accessed using the single (upper case) letter as well as the full name.

Examples

```
>>> import numpy as np
>>> y = np.array([[3, 1, 7],
...               [2, 0, 0],
...               [8, 5, 9]])
>>> y
array([[3, 1, 7],
       [2, 0, 0],
       [8, 5, 9]])
>>> y.flags
  C_CONTIGUOUS : True
  F_CONTIGUOUS : False
  OWNDATA : True
  WRITEABLE : True
  ALIGNED : True
  WRITEBACKIFCOPY : False
>>> y.setflags(write=0, align=0)
>>> y.flags
  C_CONTIGUOUS : True
  F_CONTIGUOUS : False
  OWNDATA : True
  WRITEABLE : False
  ALIGNED : False
  WRITEBACKIFCOPY : False
>>> y.setflags(uic=1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: cannot set WRITEBACKIFCOPY flag to True
```

sort

`TrueArray.`**`sort`**(*axis=-1*, *kind=None*, *order=None*, ***, *stable=None*)

Sort an array in-place. Refer to *numpy.sort* for full documentation.

Parameters

axis
: [int, optional] Axis along which to sort. Default is -1, which means sort along the last axis.

kind
: [{'quicksort', 'mergesort', 'heapsort', 'stable'}, optional] Sorting algorithm. The default is 'quicksort'. Note that both 'stable' and 'mergesort' use timsort under the covers and, in general, the actual implementation will vary with datatype. The 'mergesort' option is retained for backwards compatibility.

order
: [str or list of str, optional] When *a* is an array with fields defined, this argument specifies which fields to compare first, second, etc. A single field can be specified as a string, and not all fields need be specified, but unspecified fields will still be used, in the order in which they come up in the dtype, to break ties.

stable
: [bool, optional] Sort stability. If `True`, the returned array will maintain the relative order of `a` values

which compare as equal. If `False` or `None`, this is not guaranteed. Internally, this option selects `kind='stable'`. Default: `None`.

Added in version 2.0.0.

See Also

numpy.sort : Return a sorted copy of an array. numpy.argsort : Indirect sort. numpy.lexsort : Indirect stable sort on multiple keys. numpy.searchsorted : Find elements in sorted array. numpy.partition: Partial sort.

Notes

See *numpy.sort* for notes on the different sorting algorithms.

Examples

```
>>> import numpy as np
>>> a = np.array([[1,4], [3,1]])
>>> a.sort(axis=1)
>>> a
array([[1, 4],
       [1, 3]])
>>> a.sort(axis=0)
>>> a
array([[1, 3],
       [1, 4]])
```

Use the *order* keyword to specify a field to use when sorting a structured array:

```
>>> a = np.array([('a', 2), ('c', 1)], dtype=[('x', 'S1'), ('y', int)])
>>> a.sort(order='y')
>>> a
array([(b'c', 1), (b'a', 2)],
      dtype=[('x', 'S1'), ('y', '<i8')])
```

squeeze

`TrueArray.`**`squeeze`**(*axis=None*)

Remove axes of length one from *a*.

Refer to *numpy.squeeze* for full documentation.

See Also

numpy.squeeze : equivalent function

std

`TrueArray.`**`std`**(*axis=None*, *dtype=None*, *out=None*, *ddof=0*, ***, *keepdims=<no value>*, *where=<no value>*, *mean=<no value>*)

Returns the standard deviation of the array elements along given axis.

Refer to *numpy.std* for full documentation.

See Also

numpy.std : equivalent function

sum

`TrueArray.sum`(*axis=None*, *dtype=None*, *out=None*, *, *keepdims=<no value>*, *initial=<no value>*, *where=<no value>*)

Return the sum of the array elements over the given axis.

Refer to *numpy.sum* for full documentation.

See Also

numpy.sum : equivalent function

swapaxes

`TrueArray.swapaxes`(*axis1*, *axis2*, /)

Return a view of the array with *axis1* and *axis2* interchanged.

Refer to *numpy.swapaxes* for full documentation.

See Also

numpy.swapaxes : equivalent function

take

`TrueArray.take`(*indices*, *axis=None*, *out=None*, *mode='raise'*)

Return an array formed from the elements of *a* at the given indices.

Refer to *numpy.take* for full documentation.

See Also

numpy.take : equivalent function

to_device

`TrueArray.to_device`(*device*, /, *, *stream=None*)

For Array API compatibility. Since NumPy only supports CPU arrays, this method is a no-op that returns the same array.

Parameters

device
[“cpu”] Must be `"cpu"`.

stream
[None, optional] Currently unsupported.

Returns

out
[Self] Returns the same array.

tobytes

`TrueArray.tobytes(order='C')`

Construct Python bytes containing the raw data bytes in the array.

Constructs Python bytes showing a copy of the raw contents of data memory. The bytes object is produced in C-order by default. This behavior is controlled by the `order` parameter.

Parameters

order
[{'C', 'F', 'A'}, optional] Controls the memory layout of the bytes object. 'C' means C-order, 'F' means F-order, 'A' (short for *Any*) means 'F' if *a* is Fortran contiguous, 'C' otherwise. Default is 'C'.

Returns

s
[bytes] Python bytes exhibiting a copy of *a*'s raw data.

See also

frombuffer
Inverse of this operation, construct a 1-dimensional array from Python bytes.

Examples

```
>>> import numpy as np
>>> x = np.array([[0, 1], [2, 3]], dtype='<u2')
>>> x.tobytes()
b'\x00\x00\x01\x00\x02\x00\x03\x00'
>>> x.tobytes('C') == x.tobytes()
True
>>> x.tobytes('F')
b'\x00\x00\x02\x00\x01\x00\x03\x00'
```

tofile

`TrueArray.tofile(fid, /, sep='', format='%s')`

Write array to a file as text or binary (default).

Data is always written in 'C' order, independent of the order of *a*. The data produced by this method can be recovered using the function fromfile().

Parameters

fid
[file or str or Path] An open file object, or a string containing a filename.

sep
[str] Separator between array items for text output. If "" (empty), a binary file is written, equivalent to `file.write(a.tobytes())`.

format
[str] Format string for text file output. Each entry in the array is formatted to text by first converting it to the closest Python type, and then using "format" % item.

Notes

This is a convenience function for quick storage of array data. Information on endianness and precision is lost, so this method is not a good choice for files intended to archive data or transport data between machines with different endianness. Some of these problems can be overcome by outputting the data as text files, at the expense of speed and file size.

When fid is a file object, array contents are directly written to the file, bypassing the file object's `write` method. As a result, tofile cannot be used with files objects supporting compression (e.g., GzipFile) or file-like objects that do not support `fileno()` (e.g., BytesIO).

tolist

`TrueArray.tolist()`

Return the array as an `a.ndim`-levels deep nested list of Python scalars.

Return a copy of the array data as a (nested) Python list. Data items are converted to the nearest compatible builtin Python type, via the *~numpy.ndarray.item* method.

If `a.ndim` is 0, then since the depth of the nested list is 0, it will not be a list at all, but a simple Python scalar.

Parameters

none

Returns

y
[object, or list of object, or list of list of object, or . . .] The possibly nested list of array elements.

Notes

The array may be recreated via `a = np.array(a.tolist())`, although this may sometimes lose precision.

Examples

For a 1D array, `a.tolist()` is almost the same as `list(a)`, except that `tolist` changes numpy scalars to Python scalars:

```
>>> import numpy as np
>>> a = np.uint32([1, 2])
>>> a_list = list(a)
>>> a_list
[np.uint32(1), np.uint32(2)]
>>> type(a_list[0])
```

(continues on next page)

(continued from previous page)

```
<class 'numpy.uint32'>
>>> a_tolist = a.tolist()
>>> a_tolist
[1, 2]
>>> type(a_tolist[0])
<class 'int'>
```

Additionally, for a 2D array, `tolist` applies recursively:

```
>>> a = np.array([[1, 2], [3, 4]])
>>> list(a)
[array([1, 2]), array([3, 4])]
>>> a.tolist()
[[1, 2], [3, 4]]
```

The base case for this recursion is a 0D array:

```
>>> a = np.array(1)
>>> list(a)
Traceback (most recent call last):
  ...
TypeError: iteration over a 0-d array
>>> a.tolist()
1
```

trace

`TrueArray.`**`trace`**(*offset=0*, *axis1=0*, *axis2=1*, *dtype=None*, *out=None*)

Return the sum along diagonals of the array.

Refer to *numpy.trace* for full documentation.

See Also

numpy.trace : equivalent function

transpose

`TrueArray.`**`transpose`**(**axes*)

Returns a view of the array with axes transposed.

Refer to *numpy.transpose* for full documentation.

Parameters

axes : None, tuple of ints, or *n* ints

- None or no argument: reverses the order of the axes.
- tuple of ints: *i* in the *j*-th place in the tuple means that the array's *i*-th axis becomes the transposed array's *j*-th axis.
- *n* ints: same as an n-tuple of the same ints (this form is intended simply as a "convenience" alternative to the tuple form).

Returns

p
[ndarray] View of the array with its axes suitably permuted.

See Also

transpose : Equivalent function. ndarray.T : Array property returning the array transposed. ndarray.reshape : Give a new shape to an array without changing its data.

Examples

```
>>> import numpy as np
>>> a = np.array([[1, 2], [3, 4]])
>>> a
array([[1, 2],
       [3, 4]])
>>> a.transpose()
array([[1, 3],
       [2, 4]])
>>> a.transpose((1, 0))
array([[1, 3],
       [2, 4]])
>>> a.transpose(1, 0)
array([[1, 3],
       [2, 4]])
```

```
>>> a = np.array([1, 2, 3, 4])
>>> a
array([1, 2, 3, 4])
>>> a.transpose()
array([1, 2, 3, 4])
```

var

`TrueArray.var`(*axis=None*, *dtype=None*, *out=None*, *ddof=0*, ***, *keepdims=<no value>*, *where=<no value>*, *mean=<no value>*)

Returns the variance of the array elements, along given axis.

Refer to *numpy.var* for full documentation.

See Also

numpy.var : equivalent function

view

`TrueArray.view`(*[dtype][, type]*)

New view of array with the same data.

Note

Passing None for `dtype` is different from omitting the parameter, since the former invokes `dtype(None)` which is an alias for `dtype('float64')`.

Parameters

dtype
: [data-type or ndarray sub-class, optional] Data-type descriptor of the returned view, e.g., float32 or int16. Omitting it results in the view having the same data-type as *a*. This argument can also be specified as an ndarray sub-class, which then specifies the type of the returned object (this is equivalent to setting the `type` parameter).

type
: [Python type, optional] Type of the returned view, e.g., ndarray or matrix. Again, omission of the parameter results in type preservation.

Notes

`a.view()` is used two different ways:

`a.view(some_dtype)` or `a.view(dtype=some_dtype)` constructs a view of the array's memory with a different data-type. This can cause a reinterpretation of the bytes of memory.

`a.view(ndarray_subclass)` or `a.view(type=ndarray_subclass)` just returns an instance of *ndarray_subclass* that looks at the same array (same shape, dtype, etc.) This does not cause a reinterpretation of the memory.

For `a.view(some_dtype)`, if `some_dtype` has a different number of bytes per entry than the previous dtype (for example, converting a regular array to a structured array), then the last axis of `a` must be contiguous. This axis will be resized in the result.

Changed in version 1.23.0: Only the last axis needs to be contiguous. Previously, the entire array had to be C-contiguous.

Examples

```
>>> import numpy as np
>>> x = np.array([(-1, 2)], dtype=[('a', np.int8), ('b', np.int8)])
```

Viewing array data using a different type and dtype:

```
>>> nonneg = np.dtype([("a", np.uint8), ("b", np.uint8)])
>>> y = x.view(dtype=nonneg, type=np.recarray)
>>> x["a"]
array([-1], dtype=int8)
>>> y.a
array([255], dtype=uint8)
```

Creating a view on a structured array so it can be used in calculations

```
>>> x = np.array([(1, 2),(3,4)], dtype=[('a', np.int8), ('b', np.int8)])
>>> xv = x.view(dtype=np.int8).reshape(-1,2)
>>> xv
array([[1, 2],
```

(continues on next page)

(continued from previous page)

```
       [3, 4]], dtype=int8)
>>> xv.mean(0)
array([2.,  3.])
```

Making changes to the view changes the underlying array

```
>>> xv[0,1] = 20
>>> x
array([(1, 20), (3,  4)], dtype=[('a', 'i1'), ('b', 'i1')])
```

Using a view to convert an array to a recarray:

```
>>> z = x.view(np.recarray)
>>> z.a
array([1, 3], dtype=int8)
```

Views share data:

```
>>> x[0] = (9, 10)
>>> z[0]
np.record((9, 10), dtype=[('a', 'i1'), ('b', 'i1')])
```

Views that change the dtype size (bytes per entry) should normally be avoided on arrays defined by slices, transposes, fortran-ordering, etc.:

```
>>> x = np.array([[1, 2, 3], [4, 5, 6]], dtype=np.int16)
>>> y = x[:, ::2]
>>> y
array([[1, 3],
       [4, 6]], dtype=int16)
>>> y.view(dtype=[('width', np.int16), ('length', np.int16)])
Traceback (most recent call last):
    ...
ValueError: To change to a dtype of a different size, the last axis must be␣
↪contiguous
>>> z = y.copy()
>>> z.view(dtype=[('width', np.int16), ('length', np.int16)])
array([[(1, 3)],
       [(4, 6)]], dtype=[('width', '<i2'), ('length', '<i2')])
```

However, views that change dtype are totally fine for arrays with a contiguous last axis, even if the rest of the axes are not C-contiguous:

```
>>> x = np.arange(2 * 3 * 4, dtype=np.int8).reshape(2, 3, 4)
>>> x.transpose(1, 0, 2).view(np.int16)
array([[[ 256,  770],
        [3340, 3854]],

       [[1284, 1798],
        [4368, 4882]],

       [[2312, 2826],
        [5396, 5910]]], dtype=int16)
```

`__init__()`

Attributes

`T`	View of the transposed array.
`base`	Base object if memory is from some other object.
`ctypes`	An object to simplify the interaction of the array with the ctypes module.
`data`	Python buffer object pointing to the start of the array's data.
`device`	
`dtype`	Data-type of the array's elements.
`flags`	Information about the memory layout of the array.
`flat`	A 1-D iterator over the array.
`imag`	The imaginary part of the array.
`itemsize`	Length of one array element in bytes.
`mT`	View of the matrix transposed array.
`nbytes`	Total bytes consumed by the elements of the array.
`ndim`	Number of array dimensions.
`real`	The real part of the array.
`shape`	Tuple of array dimensions.
`size`	Number of elements in the array.
`strides`	Tuple of bytes to step in each dimension when traversing an array.

T

`TrueArray.T`

View of the transposed array.

Same as `self.transpose()`.

Examples

```
>>> import numpy as np
>>> a = np.array([[1, 2], [3, 4]])
>>> a
array([[1, 2],
       [3, 4]])
>>> a.T
array([[1, 3],
       [2, 4]])
```

```
>>> a = np.array([1, 2, 3, 4])
>>> a
array([1, 2, 3, 4])
>>> a.T
array([1, 2, 3, 4])
```

See Also

transpose

base

TrueArray.**base**

Base object if memory is from some other object.

Examples

The base of an array that owns its memory is None:

```
>>> import numpy as np
>>> x = np.array([1,2,3,4])
>>> x.base is None
True
```

Slicing creates a view, whose memory is shared with x:

```
>>> y = x[2:]
>>> y.base is x
True
```

ctypes

TrueArray.**ctypes**

An object to simplify the interaction of the array with the ctypes module.

This attribute creates an object that makes it easier to use arrays when calling shared libraries with the ctypes module. The returned object has, among others, data, shape, and strides attributes (see Notes below) which themselves return ctypes objects that can be used as arguments to a shared library.

Parameters

None

Returns

c

[Python object] Possessing attributes data, shape, strides, etc.

See Also

numpy.ctypeslib

Notes

Below are the public attributes of this object which were documented in "Guide to NumPy" (we have omitted undocumented public attributes, as well as documented private attributes):

_ctypes.**data**

A pointer to the memory area of the array as a Python integer. This memory area may contain data that is not aligned, or not in correct byte-order. The memory area may not even be writeable. The

array flags and data-type of this array should be respected when passing this attribute to arbitrary C-code to avoid trouble that can include Python crashing. User Beware! The value of this attribute is exactly the same as: `self._array_interface_['data'][0]`.

Note that unlike `data_as`, a reference won't be kept to the array: code like `ctypes.c_void_p((a + b).ctypes.data)` will result in a pointer to a deallocated array, and should be spelt `(a + b).ctypes.data_as(ctypes.c_void_p)`

`_ctypes.`**`shape`**

(c_intp*self.ndim): A ctypes array of length self.ndim where the basetype is the C-integer corresponding to `dtype('p')` on this platform (see *~numpy.ctypeslib.c_intp*). This base-type could be *ctypes.c_int*, *ctypes.c_long*, or *ctypes.c_longlong* depending on the platform. The ctypes array contains the shape of the underlying array.

`_ctypes.`**`strides`**

(c_intp*self.ndim): A ctypes array of length self.ndim where the basetype is the same as for the shape attribute. This ctypes array contains the strides information from the underlying array. This strides information is important for showing how many bytes must be jumped to get to the next element in the array.

`_ctypes.`**`data_as`**`(`*obj*`)`

Return the data pointer cast to a particular c-types object. For example, calling `self._as_parameter_` is equivalent to `self.data_as(ctypes.c_void_p)`. Perhaps you want to use the data as a pointer to a ctypes array of floating-point data: `self.data_as(ctypes.POINTER(ctypes.c_double))`.

The returned pointer will keep a reference to the array.

`_ctypes.`**`shape_as`**`(`*obj*`)`

Return the shape tuple as an array of some other c-types type. For example: `self.shape_as(ctypes.c_short)`.

`_ctypes.`**`strides_as`**`(`*obj*`)`

Return the strides tuple as an array of some other c-types type. For example: `self.strides_as(ctypes.c_longlong)`.

If the ctypes module is not available, then the ctypes attribute of array objects still returns something useful, but ctypes objects are not returned and errors may be raised instead. In particular, the object will still have the `as_parameter` attribute which will return an integer equal to the data attribute.

Examples

```
>>> import numpy as np
>>> import ctypes
>>> x = np.array([[0, 1], [2, 3]], dtype=np.int32)
>>> x
array([[0, 1],
       [2, 3]], dtype=int32)
>>> x.ctypes.data
31962608 # may vary
>>> x.ctypes.data_as(ctypes.POINTER(ctypes.c_uint32))
<__main__.LP_c_uint object at 0x7ff2fc1fc200> # may vary
>>> x.ctypes.data_as(ctypes.POINTER(ctypes.c_uint32)).contents
c_uint(0)
>>> x.ctypes.data_as(ctypes.POINTER(ctypes.c_uint64)).contents
c_ulong(4294967296)
```

(continues on next page)

(continued from previous page)

```
>>> x.ctypes.shape
<numpy._core._internal.c_long_Array_2 object at 0x7ff2fc1fce60> # may vary
>>> x.ctypes.strides
<numpy._core._internal.c_long_Array_2 object at 0x7ff2fc1ff320> # may vary
```

data

`TrueArray.data`

Python buffer object pointing to the start of the array's data.

device

`TrueArray.device`

dtype

`TrueArray.dtype`

Data-type of the array's elements.

Warning

Setting `arr.dtype` is discouraged and may be deprecated in the future. Setting will replace the `dtype` without modifying the memory (see also *ndarray.view* and *ndarray.astype*).

Parameters

None

Returns

d : numpy dtype object

See Also

ndarray.astype : Cast the values contained in the array to a new data-type. ndarray.view : Create a view of the same data but a different data-type. numpy.dtype

Examples

```
>>> import numpy as np
>>> x = np.arange(4).reshape((2, 2))
>>> x
array([[0, 1],
       [2, 3]])
>>> x.dtype
dtype('int64')   # may vary (OS, bitness)
>>> isinstance(x.dtype, np.dtype)
True
```

flags

`TrueArray.flags`

Information about the memory layout of the array.

Attributes

C_CONTIGUOUS (C)
The data is in a single, C-style contiguous segment.

F_CONTIGUOUS (F)
The data is in a single, Fortran-style contiguous segment.

OWNDATA (O)
The array owns the memory it uses or borrows it from another object.

WRITEABLE (W)
The data area can be written to. Setting this to False locks the data, making it read-only. A view (slice, etc.) inherits WRITEABLE from its base array at creation time, but a view of a writeable array may be subsequently locked while the base array remains writeable. (The opposite is not true, in that a view of a locked array may not be made writeable. However, currently, locking a base object does not lock any views that already reference it, so under that circumstance it is possible to alter the contents of a locked array via a previously created writeable view onto it.) Attempting to change a non-writeable array raises a RuntimeError exception.

ALIGNED (A)
The data and all elements are aligned appropriately for the hardware.

WRITEBACKIFCOPY (X)
This array is a copy of some other array. The C-API function PyArray_ResolveWritebackIfCopy must be called before deallocating to the base array will be updated with the contents of this array.

FNC
F_CONTIGUOUS and not C_CONTIGUOUS.

FORC
F_CONTIGUOUS or C_CONTIGUOUS (one-segment test).

BEHAVED (B)
ALIGNED and WRITEABLE.

CARRAY (CA)
BEHAVED and C_CONTIGUOUS.

FARRAY (FA)
BEHAVED and F_CONTIGUOUS and not C_CONTIGUOUS.

Notes

The *flags* object can be accessed dictionary-like (as in `a.flags['WRITEABLE']`), or by using lowercased attribute names (as in `a.flags.writeable`). Short flag names are only supported in dictionary access.

Only the WRITEBACKIFCOPY, WRITEABLE, and ALIGNED flags can be changed by the user, via direct assignment to the attribute or dictionary entry, or by calling *ndarray.setflags*.

The array flags cannot be set arbitrarily:

- WRITEBACKIFCOPY can only be set `False`.
- ALIGNED can only be set `True` if the data is truly aligned.

- WRITEABLE can only be set `True` if the array owns its own memory or the ultimate owner of the memory exposes a writeable buffer interface or is a string.

Arrays can be both C-style and Fortran-style contiguous simultaneously. This is clear for 1-dimensional arrays, but can also be true for higher dimensional arrays.

Even for contiguous arrays a stride for a given dimension `arr.strides[dim]` may be *arbitrary* if `arr.shape[dim] == 1` or the array has no elements. It does *not* generally hold that `self.strides[-1] == self.itemsize` for C-style contiguous arrays or `self.strides[0] == self.itemsize` for Fortran-style contiguous arrays is true.

flat

`TrueArray.flat`

A 1-D iterator over the array.

This is a *numpy.flatiter* instance, which acts similarly to, but is not a subclass of, Python's built-in iterator object.

See Also

flatten : Return a copy of the array collapsed into one dimension.

flatiter

Examples

```
>>> import numpy as np
>>> x = np.arange(1, 7).reshape(2, 3)
>>> x
array([[1, 2, 3],
       [4, 5, 6]])
>>> x.flat[3]
4
>>> x.T
array([[1, 4],
       [2, 5],
       [3, 6]])
>>> x.T.flat[3]
5
>>> type(x.flat)
<class 'numpy.flatiter'>
```

An assignment example:

```
>>> x.flat = 3; x
array([[3, 3, 3],
       [3, 3, 3]])
>>> x.flat[[1,4]] = 1; x
array([[3, 1, 3],
       [3, 1, 3]])
```

imag

TrueArray.**imag**

The imaginary part of the array.

Examples

```
>>> import numpy as np
>>> x = np.sqrt([1+0j, 0+1j])
>>> x.imag
array([ 0.        ,  0.70710678])
>>> x.imag.dtype
dtype('float64')
```

itemsize

TrueArray.**itemsize**

Length of one array element in bytes.

Examples

```
>>> import numpy as np
>>> x = np.array([1,2,3], dtype=np.float64)
>>> x.itemsize
8
>>> x = np.array([1,2,3], dtype=np.complex128)
>>> x.itemsize
16
```

mT

TrueArray.**mT**

View of the matrix transposed array.

The matrix transpose is the transpose of the last two dimensions, even if the array is of higher dimension.

Added in version 2.0.

Raises

ValueError

If the array is of dimension less than 2.

Examples

```
>>> import numpy as np
>>> a = np.array([[1, 2], [3, 4]])
>>> a
array([[1, 2],
       [3, 4]])
>>> a.mT
array([[1, 3],
       [2, 4]])
```

```
>>> a = np.arange(8).reshape((2, 2, 2))
>>> a
array([[[0, 1],
        [2, 3]],

       [[4, 5],
        [6, 7]]])
>>> a.mT
array([[[0, 2],
        [1, 3]],

       [[4, 6],
        [5, 7]]])
```

nbytes

TrueArray.**nbytes**

Total bytes consumed by the elements of the array.

Notes

Does not include memory consumed by non-element attributes of the array object.

See Also

sys.getsizeof
Memory consumed by the object itself without parents in case view. This does include memory consumed by non-element attributes.

Examples

```
>>> import numpy as np
>>> x = np.zeros((3,5,2), dtype=np.complex128)
>>> x.nbytes
480
>>> np.prod(x.shape) * x.itemsize
480
```

ndim

TrueArray.**ndim**

Number of array dimensions.

Examples

```
>>> import numpy as np
>>> x = np.array([1, 2, 3])
>>> x.ndim
1
>>> y = np.zeros((2, 3, 4))
```

(continues on next page)

(continued from previous page)

```
>>> y.ndim
3
```

real

`TrueArray.real`

The real part of the array.

Examples

```
>>> import numpy as np
>>> x = np.sqrt([1+0j, 0+1j])
>>> x.real
array([ 1.        ,  0.70710678])
>>> x.real.dtype
dtype('float64')
```

See Also

numpy.real : equivalent function

shape

`TrueArray.shape`

Tuple of array dimensions.

The shape property is usually used to get the current shape of an array, but may also be used to reshape the array in-place by assigning a tuple of array dimensions to it. As with *numpy.reshape*, one of the new shape dimensions can be -1, in which case its value is inferred from the size of the array and the remaining dimensions. Reshaping an array in-place will fail if a copy is required.

> **Warning**
>
> Setting `arr.shape` is discouraged and may be deprecated in the future. Using *ndarray.reshape* is the preferred approach.

Examples

```
>>> import numpy as np
>>> x = np.array([1, 2, 3, 4])
>>> x.shape
(4,)
>>> y = np.zeros((2, 3, 4))
>>> y.shape
(2, 3, 4)
>>> y.shape = (3, 8)
>>> y
array([[ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
```

(continues on next page)

(continued from previous page)

```
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.]])
>>> y.shape = (3, 6)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: cannot reshape array of size 24 into shape (3,6)
>>> np.zeros((4,2))[::2].shape = (-1,)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: Incompatible shape for in-place modification. Use
`.reshape()` to make a copy with the desired shape.
```

See Also

numpy.shape : Equivalent getter function. numpy.reshape : Function similar to setting `shape`. ndarray.reshape : Method similar to setting `shape`.

size

`TrueArray.size`

Number of elements in the array.

Equal to `np.prod(a.shape)`, i.e., the product of the array's dimensions.

Notes

a.size returns a standard arbitrary precision Python integer. This may not be the case with other methods of obtaining the same value (like the suggested `np.prod(a.shape)`, which returns an instance of `np.int_`), and may be relevant if the value is used further in calculations that may overflow a fixed size integer type.

Examples

```
>>> import numpy as np
>>> x = np.zeros((3, 5, 2), dtype=np.complex128)
>>> x.size
30
>>> np.prod(x.shape)
30
```

strides

`TrueArray.strides`

Tuple of bytes to step in each dimension when traversing an array.

The byte offset of element `(i[0], i[1], ..., i[n])` in an array *a* is:

```
offset = sum(np.array(i) * a.strides)
```

A more detailed explanation of strides can be found in arrays.ndarray.

Warning

Setting `arr.strides` is discouraged and may be deprecated in the future. *numpy.lib.stride_tricks.as_strided* should be preferred to create a new view of the same data in a safer way.

Notes

Imagine an array of 32-bit integers (each 4 bytes):

```
x = np.array([[0, 1, 2, 3, 4],
              [5, 6, 7, 8, 9]], dtype=np.int32)
```

This array is stored in memory as 40 bytes, one after the other (known as a contiguous block of memory). The strides of an array tell us how many bytes we have to skip in memory to move to the next position along a certain axis. For example, we have to skip 4 bytes (1 value) to move to the next column, but 20 bytes (5 values) to get to the same position in the next row. As such, the strides for the array *x* will be `(20, 4)`.

See Also

numpy.lib.stride_tricks.as_strided

Examples

```
>>> import numpy as np
>>> y = np.reshape(np.arange(2 * 3 * 4, dtype=np.int32), (2, 3, 4))
>>> y
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]],
       [[12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23]]], dtype=np.int32)
>>> y.strides
(48, 16, 4)
>>> y[1, 1, 1]
np.int32(17)
>>> offset = sum(y.strides * np.array((1, 1, 1)))
>>> offset // y.itemsize
np.int64(17)
```

```
>>> x = np.reshape(np.arange(5*6*7*8, dtype=np.int32), (5, 6, 7, 8))
>>> x = x.transpose(2, 3, 1, 0)
>>> x.strides
(32, 4, 224, 1344)
>>> i = np.array([3, 5, 2, 2], dtype=np.int32)
>>> offset = sum(i * x.strides)
>>> x[3, 5, 2, 2]
np.int32(813)
>>> offset // x.itemsize
np.int64(813)
```

3.11.5.7 logic

Python equivalents of logical Excel functions.

Functions

solve_cycle
xand
xbycol
xif
xiferror
xifna
xifs
xmakearray
xmap
xscan
xswitch

solve_cycle

solve_cycle(**args*)

xand

xand(*logical*, **logicals*, *func=<built-in method reduce of numpy.ufunc object>*)

xbycol

xbycol(*array*, *func*, *axis=0*)

xif

xif(*condition*, *x*, *y=False*)

xiferror

xiferror(*val*, *val_if_error*)

xifna

xifna(*val*, *val_if_error*)

xifs

xifs(**cond_vals*)

xmakearray

xmakearray(*rows*, *cols*, *func*)

xmap

xmap(*array*, *func*, **arrays*)

xscan

xscan(*initial_value*, *array*, *func*)

xswitch

xswitch(*val*, **args*)

3.11.5.8 look

Python equivalents of lookup and reference Excel functions.

Functions

args_parser_hlookup	
args_parser_lookup_array	
args_parser_match_array	
args_parser_typed_array	
args_parser_xlookup_array	
args_parser_xmatch_array	
input_parser_xlookup	
input_parser_xmatch	
map_multiindex_take_last_if_tuple	For each label in a (Multi)Index, if that label is a tuple, replace it with its second element.
return_trimrange_func	
return_unique_func	
to_python	
xaddress	
xareas	
xchoosecols	
xcolumn	
xcolumns	
xdrop	
xexpand	
xfilter	
xgroupby	
xhstack	
xindex	
xlookup	
xmatch	
xrow	
xsingle	
xsort	
xsortby	
xtake	
xtocol	
xtranspose	
xtrimrange	

continues on next page

Table 64 – continued from previous page

xunique
xvstack
xwrapcols
xxlookup
xxmatch

args_parser_hlookup

args_parser_hlookup(*val*, *vec*, *index*, *match_type=1*, *transpose=False*)

args_parser_lookup_array

args_parser_lookup_array(*lookup_val*, *lookup_vec*, *result_vec=None*, *match_type=1*)

args_parser_match_array

args_parser_match_array(*val*, *arr*, *match_type=1*, *lower=<numpy.vectorize object>*)

args_parser_typed_array

args_parser_typed_array(*arr*)

args_parser_xlookup_array

args_parser_xlookup_array(*lookup_val*, *lookup_vec*, *return_array*, *if_not_found=#N/A*, *match_mode=0*, *search_mode=1*)

args_parser_xmatch_array

args_parser_xmatch_array(*val*, *arr*, *match_mode=0*, *search_mode=1*)

input_parser_xlookup

input_parser_xlookup(*val_types*, *val*, *val_raw*, *index*, *arr_types*, *lookup_array*, *lookup_array_raw*, *match_mode*, *search_mode*, *if_not_found*, *return_array*)

input_parser_xmatch

input_parser_xmatch(*val_types*, *val*, *val_raw*, *index*, *arr_types*, *lookup_array*, *lookup_array_raw*, *match_mode*, *search_mode*)

map_multiindex_take_last_if_tuple

map_multiindex_take_last_if_tuple(*mi*)

For each label in a (Multi)Index, if that label is a tuple, replace it with its second element. Works for both Index and MultiIndex.

return_trimrange_func

return_trimrange_func(*res*, **args*)

return_unique_func

return_unique_func(*res*, **args*)

to_python

to_python(*obj*)

xaddress

xaddress(*row_num*, *column_num*, *abs_num=1*, *a1=True*, *sheet_text=None*)

xareas

xareas(*ref*)

xchoosecols

xchoosecols(*arr*, *col*, **cols*, *axis=1*)

xcolumn

xcolumn(*cell=None*, *ref=None*)

xcolumns

xcolumns(*ref*, *, *axis=1*)

xdrop

xdrop(*array*, *rows*, *columns=0*)

xexpand

xexpand(*array*, *rows*, *columns=None*, *pad_with=#N/A*)

xfilter

xfilter(*array*, *condition*, *if_empty=#VALUE!*)

xgroupby

xgroupby(*row_fields*, *values*, *aggfunc*, *field_headers=None*, *row_total_depth=1*, *row_sort_order=1*, *filter_array=None*, *field_relationship=0*)

xhstack

xhstack(*array*, **arrays*)

xindex

xindex(*array*, *row_num*, *col_num=None*, *area_num=1*)

xlookup

xlookup(*lookup_value_type*, *lookup_value*, *lookup_value_raw*, *lookup_array_index*, *lookup_array_type*, *lookup_array*, *lookup_array_raw*, *match_type=1*, *result_vec=None*)

xmatch

xmatch(*lookup_value_type*, *lookup_value*, *lookup_value_raw*, *lookup_array_index*, *lookup_array_type*, *lookup_array*, *lookup_array_raw*, *match_type=1*)

xrow

xrow(*cell=None*, *ref=None*)

xsingle

xsingle(*cell*, *rng*)

xsort

xsort(*array*, *sort_index=1*, *sort_order=1*, *by_col=0*)

xsortby

xsortby(*ref*, *by_array*, *sort_order1=1*, **args*)

xtake

xtake(*array*, *rows*, *columns=None*)

xtocol

xtocol(*array*, *ignore=0*, *scan_by_column=0*, *axis=1*)

xtranspose

xtranspose(*array*)

xtrimrange

xtrimrange(*array*, *trim_rows*, *trim_cols*)

xunique

xunique(*array*, *by_col=0*, *exactly_once=0*)

xvstack

xvstack(*array*, **arrays*)

xwrapcols

xwrapcols(*array*, *wrap_count*, *pad_with*, *cols=True*)

xxlookup

xxlookup(*lookup_value_type*, *lookup_value*, *lookup_value_raw*, *lookup_array_index*, *lookup_array_type*, *lookup_array*, *lookup_array_raw*, *match_mode=0*, *search_mode=1*, *if_not_found=#N/A*, *return_array=None*)

xxmatch

xxmatch(*lookup_value_type*, *lookup_value*, *lookup_value_raw*, *lookup_array_index*, *lookup_array_type*, *lookup_array*, *lookup_array_raw*, *match_type=1*, *search_mode=1*)

3.11.5.9 math

Python equivalents of math and trigonometry Excel functions.

Functions

`return_func`
`round_up`
`sumx2my2`
`xarabic`
`xarctan2`
`xceiling`
`xceiling_math`
`xcot`
`xdecimal`
`xeven`
`xfact`
`xfactdouble`
`xgcd`
`xlcm`
`xmdeterm`
`xmmult`
`xmod`
`xmround`
`xmultinominal`
`xmunit`
`xodd`
`xpercentof`

continues on next page

Table 65 – continued from previous page

xpower
xrandbetween
xroman
xround
xsrqtpi
xsum
xsumproduct
xtrunc

return_func

return_func(*res*, *inp*)

round_up

round_up(*x*)

sumx2my2

sumx2my2(*array_x*, *array_y*, *func=<function <lambda>>*)

xarabic

xarabic(*text*)

xarctan2

xarctan2(*x*, *y*)

xceiling

xceiling(*num*, *sig*, *ceil=<built-in function ceil>*, *dfl=0*)

xceiling_math

xceiling_math(*num*, *sig=None*, *mode=0*, *ceil=<built-in function ceil>*)

xcot

xcot(*x*, *func=<ufunc 'tan'>*)

xdecimal

xdecimal(*text*, *radix*)

xeven

xeven(*x*)

xfact

xfact(*number*, *fact=<built-in function factorial>*, *limit=0*)

xfactdouble

xfactdouble(*number*)

xgcd

xgcd(**args*)

xlcm

xlcm(**args*)

xmdeterm

xmdeterm(*x*, *func=<function det>*)

xmmult

xmmult(*x*, *y*)

xmod

xmod(*x*, *y*)

xmround

xmround(**args*)

xmultinominal

xmultinominal(**args*)

xmunit

xmunit(*x*)

xodd

xodd(*x*)

xpercentof

xpercentof(*data_subset*, *data_all*)

xpower

xpower(*number*, *power*)

xrandbetween

xrandbetween(*bottom*, *top*)

xroman

xroman(*num*, *form=0*)

xround

xround(*x*, *d*, *func=<function round_up>*)

xsrqtpi

xsrqtpi(*number*)

xsum

xsum(**args*, *func=<function sum>*)

xsumproduct

xsumproduct(**args*)

xtrunc

xtrunc(*x*, *d=0*, *func=<built-in function trunc>*)

3.11.5.10 operators

Python equivalents of Excel operators.

Functions

logic_input_parser

logic_input_parser

logic_input_parser(*x*, *y*)

3.11.5.11 stat

Python equivalents of statistical Excel functions.

Functions

xbetadist	
xbetainv	
xbinomdist	
xbinomdistrange	
xbinominv	
xchisqdist	
xchisqdistrt	
xchisqinv	
xchisqinvrt	
xchisqtest	
xconfidence_norm	CONFIDENCE.NORM(alpha, standard_dev, size) - alpha in (0,1) - standard_dev > 0 - size > 0 (integer) Returns the margin of error for a population mean when is known.
xconfidence_t	CONFIDENCE.T(alpha, standard_dev, size) Returns the margin of error: t_{1-/2, n-1} * sd / sqrt(n)
xcorrel	
xcovariance_p	
xcovariance_s	
xexpon_dist	EXPON.DIST(x, lambda, cumulative)
xfdist	
xfdistrt	
xfinv	
xfinvrt	
xfisher	FISHER(number) -> 0.5 * ln((1+x)/(1-x)) Excel domain: -1 < x < 1 (else #NUM!)
xfisherinv	FISHERINV(y) -> inverse Fisher transform = tanh(y) - Domain: any real y (returns value in (-1, 1)).
xforecast	
xforecast_ets	
xforecast_ets_confint	
xforecast_ets_seasonality	
xforecast_ets_stat	
xfrequency	FREQUENCY(data_array, bins_array) -> list of counts
xftest	
xfunc	
xgamma	GAMMA(number) Excel domain: - allowed: any real except non-positive integers - error: number in { ..., -2, -1, 0 } -> #NUM!
xgamma_dist	
xgamma_inv	GAMMA.INV(probability, alpha, beta) - 0 < probability < 1 - alpha > 0, beta > 0
xgammaln	GAMMALN(x) and GAMMALN.PRECISE(x) Excel domain: x > 0 (else #NUM!)
xgauss	GAUSS(z) = (z) - 0.5 (is the CDF of N(0,1); result in (-0.5, 0.5))
xgrowth	
xhypergeom_dist	EXPON.DIST(x, lambda, cumulative)
xintercept	
xlinest	

continues on next page

Table 67 – continued from previous page

xlogest	
xlognormdist	
xlognorminv	
xnegbinomdist	
xnormdist	
xnorminv	
xpearson	
xpercentile	
xpercentrank	PERCENTRANK.EXC(array, x, [significance]) - Returns rank of x in array as a percentage in (0,1) (excludes endpoints).
xpermut	
xpermutationa	
xphi	
xpoisson_dist	
xprob	PROB(x_range, prob_range, lower_limit, [upper_limit])
xquartile	
xrank	
xrsq	RSQ(known_y's, known_x's) -> r^2
xslope	
xsort	
xstandardize	
xstdev	
xsteyx	
xt_dist	
xt_dist2t	
xt_distrt	
xt_inv	
xt_inv2t	
xt_test	
xtrend	
xtrimmean	TRIMMEAN(array, percent)
xweibulldist	
xz_test	Z.TEST(array, x, [sigma]) -> one-tailed p-value

xbetadist

xbetadist(*x*, *_alpha*, *_beta*, *cumulative=True*, *lb=0*, *ub=1*)

xbetainv

xbetainv(*x*, *_alpha*, *_beta*, *lb=0*, *ub=1*)

xbinomdist

xbinomdist(*number_s*, *trials*, *probability_s*, *cumulative=True*)

xbinomdistrange

xbinomdistrange(*trials*, *probability_s*, *number_s*, *number_s2=None*)

xbinominv

xbinominv(*trials*, *probability_s*, *alpha*)

xchisqdist

xchisqdist(*x*, *deg_freedom*, *cumulative=True*)

xchisqdistrt

xchisqdistrt(*x*, *deg_freedom*)

xchisqinv

xchisqinv(*probability*, *deg_freedom*)

xchisqinvrt

xchisqinvrt(*probability*, *deg_freedom*)

xchisqtest

xchisqtest(*actual_range*, *expected_range*)

xconfidence_norm

xconfidence_norm(*alpha*, *standard_dev*, *size*)

CONFIDENCE.NORM(alpha, standard_dev, size) - alpha in (0,1) - standard_dev > 0 - size > 0 (integer) Returns the margin of error for a population mean when is known.

xconfidence_t

xconfidence_t(*alpha*, *standard_dev*, *size*)

CONFIDENCE.T(alpha, standard_dev, size) Returns the margin of error: t_{1-/2, n-1} * sd / sqrt(n)

xcorrel

xcorrel(*arr1*, *arr2*)

xcovariance_p

xcovariance_p(*arr1*, *arr2*)

xcovariance_s

xcovariance_s(*arr1*, *arr2*)

xexpon_dist

xexpon_dist(*x*, *rate*, *cumulative*)

EXPON.DIST(x, lambda, cumulative)

- cumulative TRUE -> 1 - exp(-lambda * x)
- cumulative FALSE -> lambda * exp(-lambda * x)
- **Excel errors:**
  - x < 0 or lambda <= 0 -> #NUM!
  - cumulative not TRUE/FALSE -> #VALUE!

xfdist

xfdist(*x*, *deg_freedom1*, *deg_freedom2*, *cumulative=True*)

xfdistrt

xfdistrt(*x*, *deg_freedom1*, *deg_freedom2*)

xfinv

xfinv(*probability*, *deg_freedom1*, *deg_freedom2*)

xfinvrt

xfinvrt(*probability*, *deg_freedom1*, *deg_freedom2*)

xfisher

xfisher(*number*)

FISHER(number) -> 0.5 * ln((1+x)/(1-x)) Excel domain: -1 < x < 1 (else #NUM!)

xfisherinv

xfisherinv(*y*)

FISHERINV(y) -> inverse Fisher transform = tanh(y) - Domain: any real y (returns value in (-1, 1)).

xforecast

xforecast(*x*, *a=None*, *b=None*)

xforecast_ets

xforecast_ets(*target_date*, *values*, *timeline*, *seasonality=1*, *data_completion=1*, *aggregation=0*)

xforecast_ets_confint

xforecast_ets_confint(*target_date*, *values*, *timeline*, *confidence_level=0.95*, *seasonality=1*, *data_completion=1*, *aggregation=0*)

xforecast_ets_seasonality

xforecast_ets_seasonality(*values*, *timeline*, *data_completion=1*, *aggregation=0*)

xforecast_ets_stat

xforecast_ets_stat(*values*, *timeline*, *statistic_type*, *seasonality=1*, *data_completion=1*, *aggregation=0*)

xfrequency

xfrequency(*data_array*, *bins_array*)

FREQUENCY(data_array, bins_array) -> list of counts
counts[i] = # of data <= bins_sorted[i] and > previous bin counts[-1] = # of data > last bin

xftest

xftest(*array1*, *array2*)

xfunc

xfunc(**args*, *func=<built-in function max>*, *check=<function is_number>*, *convert=None*, *default=0*, *_raise=True*, *parse_args=None*)

xgamma

xgamma(*number*)

GAMMA(number) Excel domain:

- allowed: any real except non-positive integers
- error: number in { ..., -2, -1, 0 } -> #NUM!

xgamma_dist

xgamma_dist(*x*, *alpha*, *beta*, *cumulative=True*)

xgamma_inv

xgamma_inv(*probability*, *alpha*, *beta*)

GAMMA.INV(probability, alpha, beta) - 0 < probability < 1 - alpha > 0, beta > 0

xgammaln

xgammaln(*x*)

GAMMALN(x) and GAMMALN.PRECISE(x) Excel domain: x > 0 (else #NUM!)

xgauss

xgauss(*x*)

GAUSS(z) = (z) - 0.5 (is the CDF of N(0,1); result in (-0.5, 0.5))

xgrowth

xgrowth(*known_y*, *known_x=None*, *new_x=None*, *const=True*)

xhypergeom_dist

xhypergeom_dist(*sample_s*, *number_sample*, *population_s*, *number_pop*, *cumulative*)

EXPON.DIST(x, lambda, cumulative)

- cumulative TRUE -> 1 - exp(-lambda * x)
- cumulative FALSE -> lambda * exp(-lambda * x)
- **Excel errors:**
  - x < 0 or lambda <= 0 -> #NUM!
  - cumulative not TRUE/FALSE -> #VALUE!

xintercept

xintercept(*yp*, *xp*)

xlinest

xlinest(*known_y*, *known_x=None*, *const=True*, *_stats=False*)

xlogest

xlogest(*known_y*, *known_x=None*, *const=True*, *_stats=False*)

xlognormdist

xlognormdist(*z*, *mu*, *sigma*, *cumulative=True*)

xlognorminv

xlognorminv(*z*, *mu=0*, *sigma=1*)

xnegbinomdist

xnegbinomdist(*number_f*, *number_s*, *probability_s*, *cumulative=True*)

xnormdist

xnormdist(*z*, *mu*, *sigma*, *cumulative=True*)

xnorminv

xnorminv(*z*, *mu=0*, *sigma=1*)

xpearson

xpearson(*array1*, *array2*)

xpercentile

xpercentile(*v*, *p*, *exclusive=False*)

xpercentrank

xpercentrank(*exc*, *vals*, *x*, *significance=3*)

PERCENTRANK.EXC(array, x, [significance]) - Returns rank of x in array as a percentage in (0,1) (excludes endpoints). - If x is outside [min(array), max(array)] -> #N/A - Rounds to 'significance' decimal places (default 3).

xpermut

xpermut(*number*, *number_chosen*)

xpermutationa

xpermutationa(*number*, *number_chosen*)

xphi

xphi(*y*)

xpoisson_dist

xpoisson_dist(*x*, *mean*, *cumulative*)

xprob

xprob(*x_range*, *prob_range*, *lower_limit*, *upper_limit=None*)

PROB(x_range, prob_range, lower_limit, [upper_limit])
Excel-like behavior:

- Pairwise: ignore pairs where either x or p is text/boolean.
- **Errors:**
  - length mismatch (before filtering) -> #N/A (via _parse_yxp)
  - no valid numeric pairs -> #N/A
  - any probability < 0 or > 1 -> #NUM!
  - sum(probabilities) != 1 (±tol) -> #NUM!
- If upper_limit omitted -> P(X == lower_limit).
- If upper_limit provided -> P(lower_limit <= X <= upper_limit).
- If lower_limit > upper_limit -> 0.

xquartile

xquartile(*v*, *q*, *exclusive=False*)

xrank

xrank(*method*, *number*, *ref*, *order=0*)

xrsq

xrsq(*known_ys*, *known_xs*)

RSQ(known_y's, known_x's) -> r^2

xslope

xslope(*yp*, *xp*)

xsort

xsort(*values*, *k*, *large=True*)

xstandardize

xstandardize(*x*, *mean*, *standard_dev*)

xstdev

xstdev(*args*, *ddof=1*, *func=<function std>*)

xsteyx

xsteyx(*known_ys*, *known_xs*)

xt_dist

xt_dist(*x*, *deg_freedom*, *cumulative=True*)

xt_dist2t

xt_dist2t(*x*, *deg_freedom*)

xt_distrt

xt_distrt(*x*, *deg_freedom*)

xt_inv

xt_inv(*probability*, *deg_freedom*)

xt_inv2t

xt_inv2t(*probability*, *deg_freedom*)

xt_test

xt_test(*array1*, *array2*, *tails*, *ttype*)

xtrend

xtrend(*known_y*, *known_x=None*, *new_x=None*, *const=True*)

xtrimmean

xtrimmean(*array*, *percent*)

TRIMMEAN(array, percent)

- percent in [0,1); trims floor(percent*n/2) from each tail
- ignores text/booleans/blanks
- **errors:**
  - no numeric data -> #NUM!
  - percent < 0 or >= 1 -> #NUM!
  - trims away all data (2k >= n) -> #NUM!

xweibulldist

xweibulldist(*x*, *alfa*, *beta*, *cumulative=True*)

xz_test

xz_test(*array*, *x*, *sigma=None*)

Z.TEST(array, x, [sigma]) -> one-tailed p-value

- If sigma is omitted/empty, use sample stdev (ddof=1).
- Returns #N/A if no numeric data (or n<2 when sigma omitted).
- Returns #DIV/0! if sigma <= 0 (or sample stdev is 0).

3.11.5.12 text

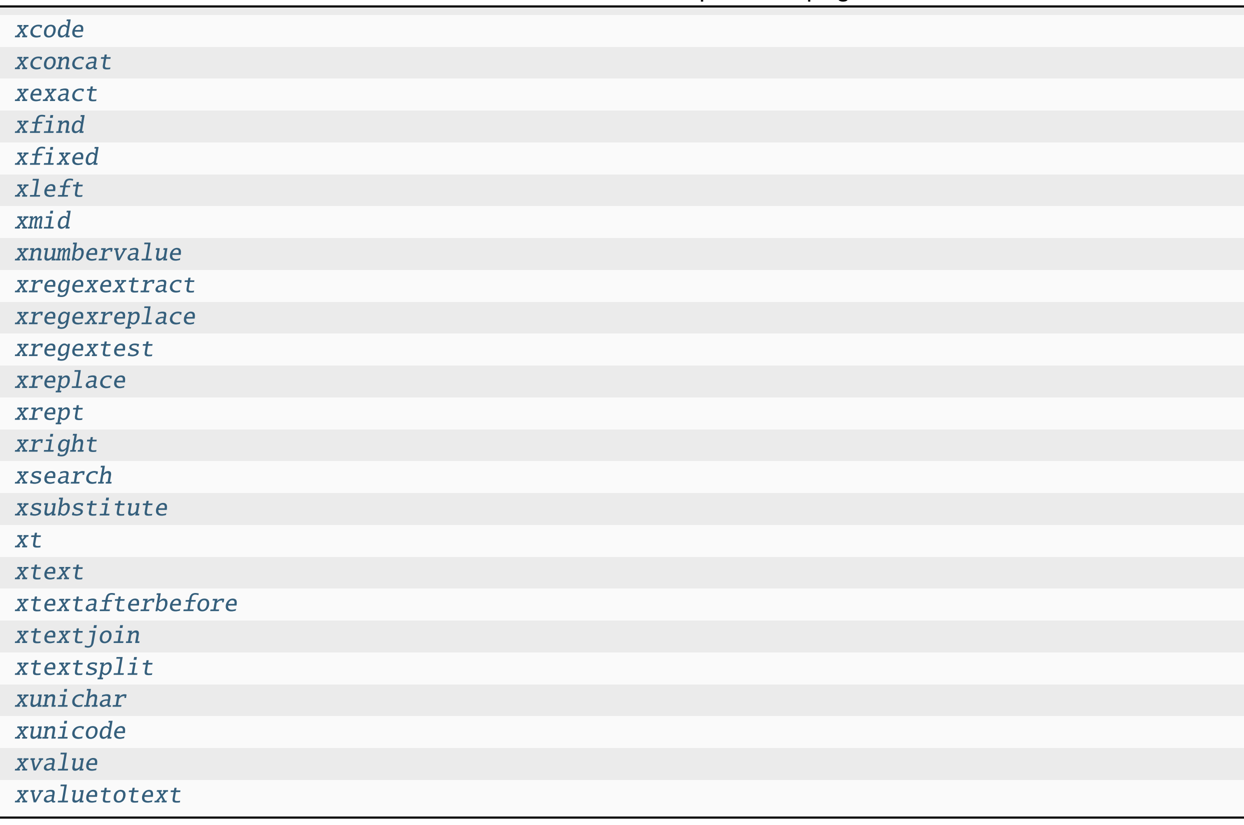
Python equivalents of text Excel functions.

Functions

xarraytotext	
xasc	Excel-like ASC: convert full-width ASCII & Katakana to half-width.
xbahttext	
xchar	
xclean	

continues on next page

Table 68 – continued from previous page



xcode
xconcat
xexact
xfind
xfixed
xleft
xmid
xnumbervalue
xregexextract
xregexreplace
xregextest
xreplace
xrept
xright
xsearch
xsubstitute
xt
xtext
xtextafterbefore
xtextjoin
xtextsplit
xunichar
xunicode
xvalue
xvaluetotext

xarraytotext

xarraytotext(*array*, *format_type=0*)

xasc

xasc(*value*)

Excel-like ASC: convert full-width ASCII & Katakana to half-width. Leaves hiragana/kanji unchanged.

xbahttext

xbahttext(*value*)

xchar

xchar(*number*)

xclean

xclean(*value*)

xcode

xcode(*character*)

xconcat

xconcat(*text*, **args*)

xexact

xexact(*text1*, *text2*)

xfind

xfind(*find_text*, *within_text*, *start_num=1*)

xfixed

xfixed(*number*, *decimals=2*, *no_commas=False*)

xleft

xleft(*from_str*, *num_chars=1*)

xmid

xmid(*from_str*, *start_num*, *num_chars*)

xnumbervalue

xnumbervalue(*text*, *decimal_sep=None*, *group_sep=None*)

xregexextract

xregexextract(*text*, *pattern*, *return_mode=0*, *case_sensitivity=0*)

xregexreplace

xregexreplace(*text*, *pattern*, *replacement*, *occurrence=0*, *case_sensitivity=0*)

xregextest

xregextest(*text*, *pattern*, *case_sensitivity=0*)

xreplace

xreplace(*old_text*, *start_num*, *num_chars*, *new_text*)

xrept

xrept(*text*, *number_times*)

xright

xright(*from_str*, *num_chars=1*)

xsearch

xsearch(*find_text*, *within_text*, *start_num=1*)

xsubstitute

xsubstitute(*text*, *old_text*, *new_text*, *instance_num=None*)

xt

xt(*value*)

xtext

xtext(*value*, *format_code*)

xtextafterbefore

xtextafterbefore(*after*, *text*, *delimiter*, *instance_num=1*, *match_mode=0*, *match_end=0*, *if_not_found=#N/A*)

xtextjoin

xtextjoin(*delimiter*, *ignore_empty*, *text*, **args*)

xtextsplit

xtextsplit(*text*, *col_delimiter*, *row_delimiter=None*, *ignore_empty=False*, *match_mode=0*, *pad_with=#N/A*)

xunichar

xunichar(*number*)

xunicode

xunicode(*character*)

xvalue

xvalue(*value*)

xvaluetotext

xvaluetotext(*text*, *format_type=0*)

Functions

args2list	
args2vals	
clean_values	
convert2float	
convert_nan	
convert_noshp	
flatten	
get_error	
get_functions	
get_shape	
is_complex	
is_not_empty	
is_number	
not_implemented	
parse_ranges	
raise_errors	
replace_empty	
return_2d_func	
str2complex	
text2num	
to_number	
wrap_func	
wrap_impure_func	
wrap_ranges_func	
wrap_ufunc	Helps call a numpy universal function (ufunc).
xfilter	
xfilters	

3.11.5.13 args2list

args2list(*max_shape*, *shapes*, **args*)

3.11.5.14 args2vals

args2vals(*args*)

3.11.5.15 clean_values

clean_values(*values*)

3.11.5.16 convert2float

convert2float(**a*)

3.11.5.17 convert_nan

convert_nan(*value*, *default=#NUM!*)

3.11.5.18 convert_noshp

convert_noshp(*value*)

3.11.5.19 flatten

flatten(*v*, *check=<function is_number>*, *drop_empty=False*)

3.11.5.20 get_error

get_error(**vals*)

3.11.5.21 get_functions

get_functions()

3.11.5.22 get_shape

get_shape(*r=1*, *c=1*)

3.11.5.23 is_complex

is_complex(*value*)

3.11.5.24 is_not_empty

is_not_empty(*v*)

3.11.5.25 is_number

is_number(*number*, *xl_return=True*, *bool_return=False*)

3.11.5.26 not_implemented

not_implemented(**args*, ***kwargs*)

3.11.5.27 parse_ranges

parse_ranges(**args*, ***kw*)

3.11.5.28 raise_errors

raise_errors(**args*)

3.11.5.29 replace_empty

replace_empty(*x*, *empty=0*)

3.11.5.30 return_2d_func

return_2d_func(*res*, **args*)

3.11.5.31 str2complex

str2complex(*string*)

3.11.5.32 text2num

text2num(**args*, ***kwargs*)

3.11.5.33 to_number

to_number(**args*, ***kwargs*)

3.11.5.34 wrap_func

wrap_func(*func*, *ranges=False*)

3.11.5.35 wrap_impure_func

wrap_impure_func(*func*)

3.11.5.36 wrap_ranges_func

wrap_ranges_func(*func*, *n_out=1*)

3.11.5.37 wrap_ufunc

wrap_ufunc(*func*, *input_parser=<function <lambda>>*, *check_error=<function get_error>*, *args_parser=<function <lambda>>*, *otype=<class 'formulas.functions.Array'>*, *ranges=False*, *return_func=<function <lambda>>*, *check_nan=True*, ***kw*)

Helps call a numpy universal function (ufunc).

3.11.5.38 xfilter

xfilter(*accumulator*, *test_range*, *condition*, *operating_range=None*)

3.11.5.39 xfilters

xfilters(*accumulator*, *operating_range*, *test_range*, *condition*, **args*)

Classes

Array

3.11.5.40 Array

class Array(*shape*, *dtype=None*, *buffer=None*, *offset=0*, *strides=None*, *order=None*)

Methods

__init__	
`all`	Returns True if all elements evaluate to True.
`any`	Returns True if any of the elements of *a* evaluate to True.
`argmax`	Return indices of the maximum values along the given axis.
`argmin`	Return indices of the minimum values along the given axis.
`argpartition`	Returns the indices that would partition this array.
`argsort`	Returns the indices that would sort this array.
`astype`	Copy of the array, cast to a specified type.
`byteswap`	Swap the bytes of the array elements
`choose`	Use an index array to construct a new array from a set of choices.
`clip`	Return an array whose values are limited to `[min, max]`.
`collapse`	
`compress`	Return selected slices of this array along given axis.
`conj`	Complex-conjugate all elements.
`conjugate`	Return the complex conjugate, element-wise.
`copy`	Return a copy of the array.
`cumprod`	Return the cumulative product of the elements along the given axis.
`cumsum`	Return the cumulative sum of the elements along the given axis.
`diagonal`	Return specified diagonals.
`dot`	Refer to `numpy.dot()` for full documentation.
`dump`	Dump a pickle of the array to the specified file.
`dumps`	Returns the pickle of the array as a string.
`fill`	Fill the array with a scalar value.
`flatten`	Return a copy of the array collapsed into one dimension.
`getfield`	Returns a field of the given array as a certain type.
`item`	Copy an element of an array to a standard Python scalar and return it.
`max`	Return the maximum along a given axis.
`mean`	Returns the average of the array elements along given axis.
`min`	Return the minimum along a given axis.
`nonzero`	Return the indices of the elements that are non-zero.
`partition`	Partially sorts the elements in the array in such a way that the value of the element in k-th position is in the position it would be in a sorted array.
`prod`	Return the product of the array elements over the given axis
`put`	Set `a.flat[n] = values[n]` for all `n` in indices.
`ravel`	Return a flattened array.
`repeat`	Repeat elements of an array.
reshape	Returns an array containing the same data with a new shape.

continues on next page

Table 71 – continued from previous page

`resize`	Change shape and size of array in-place.
`round`	Return *a* with each element rounded to the given number of decimals.
`searchsorted`	Find indices where elements of *v* should be inserted in *a* to maintain order.
`setfield`	Put a value into a specified place in a field defined by a data-type.
`setflags`	Set array flags WRITEABLE, ALIGNED, WRITE-BACKIFCOPY, respectively.
`sort`	Sort an array in-place.
`squeeze`	Remove axes of length one from *a*.
`std`	Returns the standard deviation of the array elements along given axis.
`sum`	Return the sum of the array elements over the given axis.
`swapaxes`	Return a view of the array with *axis1* and *axis2* interchanged.
`take`	Return an array formed from the elements of *a* at the given indices.
`to_device`	For Array API compatibility.
`tobytes`	Construct Python bytes containing the raw data bytes in the array.
`tofile`	Write array to a file as text or binary (default).
`tolist`	Return the array as an `a.ndim`-levels deep nested list of Python scalars.
`trace`	Return the sum along diagonals of the array.
`transpose`	Returns a view of the array with axes transposed.
`var`	Returns the variance of the array elements, along given axis.
`view`	New view of array with the same data.

__init__

`Array.__init__()`

all

`Array.all`(*axis=None*, *out=None*, *, *keepdims=<no value>*, *where=<no value>*)

Returns True if all elements evaluate to True.

Refer to *numpy.all* for full documentation.

See Also

numpy.all : equivalent function

any

`Array.any`(*axis=None*, *out=None*, *, *keepdims=<no value>*, *where=<no value>*)

Returns True if any of the elements of *a* evaluate to True.

Refer to *numpy.any* for full documentation.

See Also

numpy.any : equivalent function

argmax

`Array.argmax(axis=None, out=None, *, keepdims=False)`

Return indices of the maximum values along the given axis.

Refer to *numpy.argmax* for full documentation.

See Also

numpy.argmax : equivalent function

argmin

`Array.argmin(axis=None, out=None, *, keepdims=False)`

Return indices of the minimum values along the given axis.

Refer to *numpy.argmin* for detailed documentation.

See Also

numpy.argmin : equivalent function

argpartition

`Array.argpartition(kth, axis=-1, kind='introselect', order=None)`

Returns the indices that would partition this array.

Refer to *numpy.argpartition* for full documentation.

See Also

numpy.argpartition : equivalent function

argsort

`Array.argsort(axis=-1, kind=None, order=None, *, stable=None)`

Returns the indices that would sort this array.

Refer to *numpy.argsort* for full documentation.

See Also

numpy.argsort : equivalent function

astype

`Array.astype(dtype, order='K', casting='unsafe', subok=True, copy=True)`

Copy of the array, cast to a specified type.

Parameters

dtype
[str or dtype] Typecode or data-type to which the array is cast.

order
[{'C', 'F', 'A', 'K'}, optional] Controls the memory layout order of the result. 'C' means C order, 'F' means Fortran order, 'A' means 'F' order if all the arrays are Fortran contiguous, 'C' order otherwise, and 'K' means as close to the order the array elements appear in memory as possible. Default is 'K'.

casting
[{'no', 'equiv', 'safe', 'same_kind', 'same_value', 'unsafe'}, optional] Controls what kind of data casting may occur. Defaults to 'unsafe' for backwards compatibility.

- 'no' means the data types should not be cast at all.
- 'equiv' means only byte-order changes are allowed.
- 'safe' means only casts which can preserve values are allowed.
- 'same_kind' means only safe casts or casts within a kind, like float64 to float32, are allowed.
- 'unsafe' means any data conversions may be done.
- 'same_value' means any data conversions may be done, but the values must not change, including rounding of floats or overflow of ints

Added in version 2.4: Support for `'same_value'` was added.

subok
[bool, optional] If True, then sub-classes will be passed-through (default), otherwise the returned array will be forced to be a base-class array.

copy
[bool, optional] By default, astype always returns a newly allocated array. If this is set to false, and the *dtype*, *order*, and *subok* requirements are satisfied, the input array is returned instead of a copy.

Returns

arr_t
[ndarray] Unless *copy* is False and the other conditions for returning the input array are satisfied (see description for *copy* input parameter), *arr_t* is a new array of the same shape as the input array, with dtype, order given by *dtype*, *order*.

Raises

ComplexWarning
When casting from complex to float or int. To avoid this, one should use `a.real.astype(t)`.

ValueError
When casting using `'same_value'` and the values change or would overflow

Examples

```
>>> import numpy as np
>>> x = np.array([1, 2, 2.5])
>>> x
array([1. ,  2. ,  2.5])
```

```
>>> x.astype(int)
array([1, 2, 2])
```

```
>>> x.astype(int, casting="same_value")
Traceback (most recent call last):
...
ValueError: could not cast 'same_value' double to long
```

```
>>> x[:2].astype(int, casting="same_value")
array([1, 2])
```

byteswap

`Array.byteswap(inplace=False)`

Swap the bytes of the array elements

Toggle between low-endian and big-endian data representation by returning a byteswapped array, optionally swapped in-place. Arrays of byte-strings are not swapped. The real and imaginary parts of a complex number are swapped individually.

Parameters

inplace
[bool, optional] If `True`, swap bytes in-place, default is `False`.

Returns

out
[ndarray] The byteswapped array. If *inplace* is `True`, this is a view to self.

Examples

```
>>> import numpy as np
>>> A = np.array([1, 256, 8755], dtype=np.int16)
>>> list(map(hex, A))
['0x1', '0x100', '0x2233']
>>> A.byteswap(inplace=True)
array([  256,     1, 13090], dtype=int16)
>>> list(map(hex, A))
['0x100', '0x1', '0x3322']
```

Arrays of byte-strings are not swapped

```
>>> A = np.array([b'ceg', b'fac'])
>>> A.byteswap()
array([b'ceg', b'fac'], dtype='|S3')
```

`A.view(A.dtype.newbyteorder()).byteswap()` produces an array with the same values but different representation in memory

```
>>> A = np.array([1, 2, 3],dtype=np.int64)
>>> A.view(np.uint8)
```

(continues on next page)

(continued from previous page)

```
array([1, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0,
       0, 0], dtype=uint8)
>>> A.view(A.dtype.newbyteorder()).byteswap(inplace=True)
array([1, 2, 3], dtype='>i8')
>>> A.view(np.uint8)
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0,
       0, 3], dtype=uint8)
```

choose

`Array.choose`(*choices*, *out=None*, *mode='raise'*)

Use an index array to construct a new array from a set of choices.

Refer to *numpy.choose* for full documentation.

See Also

numpy.choose : equivalent function

clip

`Array.clip`(*min=<no value>*, *max=<no value>*, *out=None*, ***kwargs*)

Return an array whose values are limited to `[min, max]`. One of max or min must be given.

Refer to *numpy.clip* for full documentation.

See Also

numpy.clip : equivalent function

collapse

`Array.collapse`(*shape*)

compress

`Array.compress`(*condition*, *axis=None*, *out=None*)

Return selected slices of this array along given axis.

Refer to *numpy.compress* for full documentation.

See Also

numpy.compress : equivalent function

conj

`Array.conj`()

Complex-conjugate all elements.

Refer to *numpy.conjugate* for full documentation.

See Also

numpy.conjugate : equivalent function

conjugate

`Array.conjugate()`

Return the complex conjugate, element-wise.

Refer to *numpy.conjugate* for full documentation.

See Also

numpy.conjugate : equivalent function

copy

`Array.copy(order='C')`

Return a copy of the array.

Parameters

order
[{'C', 'F', 'A', 'K'}, optional] Controls the memory layout of the copy. 'C' means C-order, 'F' means F-order, 'A' means 'F' if *a* is Fortran contiguous, 'C' otherwise. 'K' means match the layout of *a* as closely as possible. (Note that this function and `numpy.copy()` are very similar but have different default values for their order= arguments, and this function always passes sub-classes through.)

See also

numpy.copy : Similar function with different default behavior numpy.copyto

Notes

This function is the preferred method for creating an array copy. The function `numpy.copy()` is similar, but it defaults to using order 'K', and will not pass sub-classes through by default.

Examples

```
>>> import numpy as np
>>> x = np.array([[1,2,3],[4,5,6]], order='F')
```

```
>>> y = x.copy()
```

```
>>> x.fill(0)
```

```
>>> x
array([[0, 0, 0],
       [0, 0, 0]])
```

```
>>> y
array([[1, 2, 3],
       [4, 5, 6]])
```

```
>>> y.flags['C_CONTIGUOUS']
True
```

For arrays containing Python objects (e.g. dtype=object), the copy is a shallow one. The new array will contain the same object which may lead to surprises if that object can be modified (is mutable):

```
>>> a = np.array([1, 'm', [2, 3, 4]], dtype=object)
>>> b = a.copy()
>>> b[2][0] = 10
>>> a
array([1, 'm', list([10, 3, 4])], dtype=object)
```

To ensure all elements within an `object` array are copied, use *copy.deepcopy*:

```
>>> import copy
>>> a = np.array([1, 'm', [2, 3, 4]], dtype=object)
>>> c = copy.deepcopy(a)
>>> c[2][0] = 10
>>> c
array([1, 'm', list([10, 3, 4])], dtype=object)
>>> a
array([1, 'm', list([2, 3, 4])], dtype=object)
```

cumprod

`Array.`**`cumprod`**(*axis=None*, *dtype=None*, *out=None*)

Return the cumulative product of the elements along the given axis.

Refer to *numpy.cumprod* for full documentation.

See Also

numpy.cumprod : equivalent function

cumsum

`Array.`**`cumsum`**(*axis=None*, *dtype=None*, *out=None*)

Return the cumulative sum of the elements along the given axis.

Refer to *numpy.cumsum* for full documentation.

See Also

numpy.cumsum : equivalent function

diagonal

`Array.`**`diagonal`**(*offset=0*, *axis1=0*, *axis2=1*)

Return specified diagonals. In NumPy 1.9 the returned array is a read-only view instead of a copy as in previous NumPy versions. In a future version the read-only restriction will be removed.

Refer to `numpy.diagonal()` for full documentation.

See Also

numpy.diagonal : equivalent function

dot

`Array.dot(other, /, out=None)`

Refer to `numpy.dot()` for full documentation.

See Also

numpy.dot : equivalent function

dump

`Array.dump(file)`

Dump a pickle of the array to the specified file. The array can be read back with pickle.load or numpy.load.

Parameters

file
[str or Path] A string naming the dump file.

dumps

`Array.dumps()`

Returns the pickle of the array as a string. `pickle.loads` will convert the string back to an array.

Parameters

None

fill

`Array.fill(value)`

Fill the array with a scalar value.

Parameters

value
[scalar] All elements of *a* will be assigned this value.

Examples

```
>>> import numpy as np
>>> a = np.array([1, 2])
>>> a.fill(0)
>>> a
array([0, 0])
>>> a = np.empty(2)
>>> a.fill(1)
>>> a
array([1.,  1.])
```

Fill expects a scalar value and always behaves the same as assigning to a single array element. The following is a rare example where this distinction is important:

```
>>> a = np.array([None, None], dtype=object)
>>> a[0] = np.array(3)
>>> a
array([array(3), None], dtype=object)
>>> a.fill(np.array(3))
>>> a
array([array(3), array(3)], dtype=object)
```

Where other forms of assignments will unpack the array being assigned:

```
>>> a[...] = np.array(3)
>>> a
array([3, 3], dtype=object)
```

flatten

`Array.flatten(order='C')`

Return a copy of the array collapsed into one dimension.

Parameters

order

[{'C', 'F', 'A', 'K'}, optional] 'C' means to flatten in row-major (C-style) order. 'F' means to flatten in column-major (Fortran- style) order. 'A' means to flatten in column-major order if *a* is Fortran *contiguous* in memory, row-major order otherwise. 'K' means to flatten *a* in the order the elements occur in memory. The default is 'C'.

Returns

y

[ndarray] A copy of the input array, flattened to one dimension.

See Also

ravel : Return a flattened array. flat : A 1-D flat iterator over the array.

Examples

```
>>> import numpy as np
>>> a = np.array([[1,2], [3,4]])
>>> a.flatten()
array([1, 2, 3, 4])
>>> a.flatten('F')
array([1, 3, 2, 4])
```

getfield

`Array.getfield(dtype, offset=0)`

Returns a field of the given array as a certain type.

A field is a view of the array data with a given data-type. The values in the view are determined by the given type and the offset into the current array in bytes. The offset needs to be such that the view dtype fits in the array dtype; for example an array of dtype complex128 has 16-byte elements. If taking a view with a 32-bit integer (4 bytes), the offset needs to be between 0 and 12 bytes.

Parameters

dtype
: [str or dtype] The data type of the view. The dtype size of the view can not be larger than that of the array itself.

offset
: [int] Number of bytes to skip before beginning the element view.

Examples

```
>>> import numpy as np
>>> x = np.diag([1.+1.j]*2)
>>> x[1, 1] = 2 + 4.j
>>> x
array([[1.+1.j,  0.+0.j],
       [0.+0.j,  2.+4.j]])
>>> x.getfield(np.float64)
array([[1.,  0.],
       [0.,  2.]])
```

By choosing an offset of 8 bytes we can select the complex part of the array for our view:

```
>>> x.getfield(np.float64, offset=8)
array([[1.,  0.],
       [0.,  4.]])
```

item

`Array.item(*args)`

Copy an element of an array to a standard Python scalar and return it.

Parameters

*args : Arguments (variable number and type)

- none: in this case, the method only works for arrays with one element (*a.size == 1*), which element is copied into a standard Python scalar object and returned.
- int_type: this argument is interpreted as a flat index into the array, specifying which element to copy and return.
- tuple of int_types: functions as does a single int_type argument, except that the argument is interpreted as an nd-index into the array.

Returns

z
: [Standard Python scalar object] A copy of the specified element of the array as a suitable Python scalar

Notes

When the data type of *a* is longdouble or clongdouble, item() returns a scalar array object because there is no available Python scalar that would not lose information. Void arrays return a buffer object for item(), unless fields are defined, in which case a tuple is returned.

item is very similar to a[args], except, instead of an array scalar, a standard Python scalar is returned. This can be useful for speeding up access to elements of the array and doing arithmetic on elements of the array using Python's optimized math.

Examples

```
>>> import numpy as np
>>> np.random.seed(123)
>>> x = np.random.randint(9, size=(3, 3))
>>> x
array([[2, 2, 6],
       [1, 3, 6],
       [1, 0, 1]])
>>> x.item(3)
1
>>> x.item(7)
0
>>> x.item((0, 1))
2
>>> x.item((2, 2))
1
```

For an array with object dtype, elements are returned as-is.

```
>>> a = np.array([np.int64(1)], dtype=object)
>>> a.item() #return np.int64
np.int64(1)
```

max

`Array.max`(*axis=None*, *out=None*, *, *keepdims=<no value>*, *initial=<no value>*, *where=<no value>*)

Return the maximum along a given axis.

Refer to *numpy.amax* for full documentation.

See Also

numpy.amax : equivalent function

mean

`Array.mean`(*axis=None*, *dtype=None*, *out=None*, *, *keepdims=<no value>*, *where=<no value>*)

Returns the average of the array elements along given axis.

Refer to *numpy.mean* for full documentation.

See Also

numpy.mean : equivalent function

min

`Array.min(axis=None, out=None, *, keepdims=<no value>, initial=<no value>, where=<no value>)`

Return the minimum along a given axis.

Refer to *numpy.amin* for full documentation.

See Also

numpy.amin : equivalent function

nonzero

`Array.nonzero()`

Return the indices of the elements that are non-zero.

Refer to *numpy.nonzero* for full documentation.

See Also

numpy.nonzero : equivalent function

partition

`Array.partition(kth, axis=-1, kind='introselect', order=None)`

Partially sorts the elements in the array in such a way that the value of the element in k-th position is in the position it would be in a sorted array. In the output array, all elements smaller than the k-th element are located to the left of this element and all equal or greater are located to its right. The ordering of the elements in the two partitions on the either side of the k-th element in the output array is undefined.

Parameters

kth
[int or sequence of ints] Element index to partition by. The kth element value will be in its final sorted position and all smaller elements will be moved before it and all equal or greater elements behind it. The order of all elements in the partitions is undefined. If provided with a sequence of kth it will partition all elements indexed by kth of them into their sorted position at once.

Deprecated since version 1.22.0: Passing booleans as index is deprecated.

axis
[int, optional] Axis along which to sort. Default is -1, which means sort along the last axis.

kind
[{'introselect'}, optional] Selection algorithm. Default is 'introselect'.

order
[str or list of str, optional] When *a* is an array with fields defined, this argument specifies which fields to compare first, second, etc. A single field can be specified as a string, and not all fields need to be specified, but unspecified fields will still be used, in the order in which they come up in the dtype, to break ties.

See Also

numpy.partition : Return a partitioned copy of an array. argpartition : Indirect partition. sort : Full sort.

Notes

See `np.partition` for notes on the different algorithms.

Examples

```
>>> import numpy as np
>>> a = np.array([3, 4, 2, 1])
>>> a.partition(3)
>>> a
array([2, 1, 3, 4]) # may vary
```

```
>>> a.partition((1, 3))
>>> a
array([1, 2, 3, 4])
```

prod

`Array.prod`(*axis=None*, *dtype=None*, *out=None*, ***, *keepdims=<no value>*, *initial=<no value>*, *where=<no value>*)

Return the product of the array elements over the given axis

Refer to *numpy.prod* for full documentation.

See Also

numpy.prod : equivalent function

put

`Array.put`(*indices*, *values*, *mode='raise'*)

Set `a.flat[n] = values[n]` for all `n` in indices.

Refer to *numpy.put* for full documentation.

See Also

numpy.put : equivalent function

ravel

`Array.ravel`(*order='C'*)

Return a flattened array.

Refer to *numpy.ravel* for full documentation.

See Also

numpy.ravel : equivalent function ndarray.flat : a flat iterator on the array.

repeat

`Array.repeat`(*repeats*, *axis=None*)

Repeat elements of an array.

Refer to *numpy.repeat* for full documentation.

See Also

numpy.repeat : equivalent function

reshape

`Array.reshape`(*shape*, /, *, *order='C'*, *copy=None*)

`Array.reshape`(**shape*, *order='C'*, *copy=None*)

Returns an array containing the same data with a new shape.

Refer to *numpy.reshape* for full documentation.

See Also

numpy.reshape : equivalent function

Notes

Unlike the free function *numpy.reshape*, this method on *ndarray* allows the elements of the shape parameter to be passed in as separate arguments. For example, `a.reshape(4, 2)` is equivalent to `a.reshape((4, 2))`.

resize

`Array.resize`(*new_shape*, /, *, *refcheck=True*)

`Array.resize`(**new_shape*, *refcheck=True*)

Change shape and size of array in-place.

Parameters

new_shape
[tuple of ints, or *n* ints] Shape of resized array.

refcheck
[bool, optional] If False, reference count will not be checked. Default is True.

Returns

None

Raises

ValueError
If *a* does not own its own data or references or views to it exist, and the data memory must be changed. PyPy only: will always raise if the data memory must be changed, since there is no reliable way to determine if references or views to it exist.

SystemError
If the *order* keyword argument is specified. This behaviour is a bug in NumPy.

See Also

resize : Return a new array with the specified shape.

Notes

This reallocates space for the data area if necessary.

Only contiguous arrays (data elements consecutive in memory) can be resized.

The purpose of the reference count check is to make sure you do not use this array as a buffer for another Python object and then reallocate the memory. However, reference counts can increase in other ways so if you are sure that you have not shared the memory for this array with another Python object, then you may safely set *refcheck* to False.

Examples

Shrinking an array: array is flattened (in the order that the data are stored in memory), resized, and reshaped:

```
>>> import numpy as np
```

```
>>> a = np.array([[0, 1], [2, 3]], order='C')
>>> a.resize((2, 1))
>>> a
array([[0],
       [1]])
```

```
>>> a = np.array([[0, 1], [2, 3]], order='F')
>>> a.resize((2, 1))
>>> a
array([[0],
       [2]])
```

Enlarging an array: as above, but missing entries are filled with zeros:

```
>>> b = np.array([[0, 1], [2, 3]])
>>> b.resize(2, 3) # new_shape parameter doesn't have to be a tuple
>>> b
array([[0, 1, 2],
       [3, 0, 0]])
```

Referencing an array prevents resizing…

```
>>> c = a
>>> a.resize((1, 1))
Traceback (most recent call last):
...
ValueError: cannot resize an array that references or is referenced ...
```

Unless *refcheck* is False:

```
>>> a.resize((1, 1), refcheck=False)
>>> a
array([[0]])
>>> c
array([[0]])
```

round

`Array.round(decimals=0, out=None)`

Return *a* with each element rounded to the given number of decimals.

Refer to *numpy.around* for full documentation.

See Also

numpy.around : equivalent function

searchsorted

`Array.searchsorted(v, side='left', sorter=None)`

Find indices where elements of *v* should be inserted in *a* to maintain order.

For full documentation, see *numpy.searchsorted*.

See Also

numpy.searchsorted : equivalent function

setfield

`Array.setfield(val, dtype, offset=0)`

Put a value into a specified place in a field defined by a data-type.

Place *val* into *a*'s field defined by *dtype* and beginning *offset* bytes into the field.

Parameters

val
[object] Value to be placed in field.

dtype
[dtype object] Data-type of the field in which to place *val*.

offset
[int, optional] The number of bytes into the field at which to place *val*.

Returns

None

See Also

getfield

Examples

```
>>> import numpy as np
>>> x = np.eye(3)
>>> x.getfield(np.float64)
array([[1.,  0.,  0.],
       [0.,  1.,  0.],
       [0.,  0.,  1.]])
>>> x.setfield(3, np.int32)
>>> x.getfield(np.int32)
array([[3, 3, 3],
       [3, 3, 3],
       [3, 3, 3]], dtype=int32)
>>> x
array([[1.0e+000, 1.5e-323, 1.5e-323],
       [1.5e-323, 1.0e+000, 1.5e-323],
       [1.5e-323, 1.5e-323, 1.0e+000]])
>>> x.setfield(np.eye(3), np.int32)
>>> x
array([[1.,  0.,  0.],
       [0.,  1.,  0.],
       [0.,  0.,  1.]])
```

setflags

`Array.setflags`(*write=None*, *align=None*, *uic=None*)

Set array flags WRITEABLE, ALIGNED, WRITEBACKIFCOPY, respectively.

These Boolean-valued flags affect how numpy interprets the memory area used by *a* (see Notes below). The ALIGNED flag can only be set to True if the data is actually aligned according to the type. The WRITEBACKIFCOPY flag can never be set to True. The flag WRITEABLE can only be set to True if the array owns its own memory, or the ultimate owner of the memory exposes a writeable buffer interface, or is a string. (The exception for string is made so that unpickling can be done without copying memory.)

Parameters

write
: [bool, optional] Describes whether or not *a* can be written to.

align
: [bool, optional] Describes whether or not *a* is aligned properly for its type.

uic
: [bool, optional] Describes whether or not *a* is a copy of another "base" array.

Notes

Array flags provide information about how the memory area used for the array is to be interpreted. There are 7 Boolean flags in use, only three of which can be changed by the user: WRITEBACKIFCOPY, WRITEABLE, and ALIGNED.

WRITEABLE (W) the data area can be written to;

ALIGNED (A) the data and strides are aligned appropriately for the hardware (as determined by the compiler);

WRITEBACKIFCOPY (X) this array is a copy of some other array (referenced by .base). When the C-API function PyArray_ResolveWritebackIfCopy is called, the base array will be updated with the contents of this array.

All flags can be accessed using the single (upper case) letter as well as the full name.

Examples

```
>>> import numpy as np
>>> y = np.array([[3, 1, 7],
...               [2, 0, 0],
...               [8, 5, 9]])
>>> y
array([[3, 1, 7],
       [2, 0, 0],
       [8, 5, 9]])
>>> y.flags
  C_CONTIGUOUS : True
  F_CONTIGUOUS : False
  OWNDATA : True
  WRITEABLE : True
  ALIGNED : True
  WRITEBACKIFCOPY : False
>>> y.setflags(write=0, align=0)
>>> y.flags
  C_CONTIGUOUS : True
  F_CONTIGUOUS : False
  OWNDATA : True
  WRITEABLE : False
  ALIGNED : False
  WRITEBACKIFCOPY : False
>>> y.setflags(uic=1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: cannot set WRITEBACKIFCOPY flag to True
```

sort

`Array.sort`(*axis=-1*, *kind=None*, *order=None*, ***, *stable=None*)

Sort an array in-place. Refer to *numpy.sort* for full documentation.

Parameters

axis
[int, optional] Axis along which to sort. Default is -1, which means sort along the last axis.

kind
[{'quicksort', 'mergesort', 'heapsort', 'stable'}, optional] Sorting algorithm. The default is 'quicksort'. Note that both 'stable' and 'mergesort' use timsort under the covers and, in general, the actual implementation will vary with datatype. The 'mergesort' option is retained for backwards compatibility.

order
[str or list of str, optional] When *a* is an array with fields defined, this argument specifies which fields to compare first, second, etc. A single field can be specified as a string, and not all fields need be specified, but unspecified fields will still be used, in the order in which they come up in the dtype, to break ties.

stable
[bool, optional] Sort stability. If `True`, the returned array will maintain the relative order of `a` values which compare as equal. If `False` or `None`, this is not guaranteed. Internally, this option selects `kind='stable'`. Default: `None`.

Added in version 2.0.0.

See Also

numpy.sort : Return a sorted copy of an array. numpy.argsort : Indirect sort. numpy.lexsort : Indirect stable sort on multiple keys. numpy.searchsorted : Find elements in sorted array. numpy.partition: Partial sort.

Notes

See *numpy.sort* for notes on the different sorting algorithms.

Examples

```
>>> import numpy as np
>>> a = np.array([[1,4], [3,1]])
>>> a.sort(axis=1)
>>> a
array([[1, 4],
       [1, 3]])
>>> a.sort(axis=0)
>>> a
array([[1, 3],
       [1, 4]])
```

Use the *order* keyword to specify a field to use when sorting a structured array:

```
>>> a = np.array([('a', 2), ('c', 1)], dtype=[('x', 'S1'), ('y', int)])
>>> a.sort(order='y')
>>> a
array([(b'c', 1), (b'a', 2)],
      dtype=[('x', 'S1'), ('y', '<i8')])
```

squeeze

`Array.squeeze(axis=None)`

Remove axes of length one from *a*.

Refer to *numpy.squeeze* for full documentation.

See Also

numpy.squeeze : equivalent function

std

`Array.std(axis=None, dtype=None, out=None, ddof=0, *, keepdims=<no value>, where=<no value>, mean=<no value>)`

Returns the standard deviation of the array elements along given axis.

Refer to *numpy.std* for full documentation.

See Also

numpy.std : equivalent function

sum

`Array.sum(axis=None, dtype=None, out=None, *, keepdims=<no value>, initial=<no value>, where=<no value>)`

Return the sum of the array elements over the given axis.

Refer to *numpy.sum* for full documentation.

See Also

numpy.sum : equivalent function

swapaxes

`Array.swapaxes(axis1, axis2, /)`

Return a view of the array with *axis1* and *axis2* interchanged.

Refer to *numpy.swapaxes* for full documentation.

See Also

numpy.swapaxes : equivalent function

take

`Array.take(indices, axis=None, out=None, mode='raise')`

Return an array formed from the elements of *a* at the given indices.

Refer to *numpy.take* for full documentation.

See Also

numpy.take : equivalent function

to_device

`Array.to_device(device, /, *, stream=None)`

For Array API compatibility. Since NumPy only supports CPU arrays, this method is a no-op that returns the same array.

Parameters

device
[“cpu”] Must be `"cpu"`.

stream
[None, optional] Currently unsupported.

Returns

out
[Self] Returns the same array.

tobytes

`Array.tobytes(order='C')`

Construct Python bytes containing the raw data bytes in the array.

Constructs Python bytes showing a copy of the raw contents of data memory. The bytes object is produced in C-order by default. This behavior is controlled by the `order` parameter.

Parameters

order
[{‘C’, ‘F’, ‘A’}, optional] Controls the memory layout of the bytes object. ‘C’ means C-order, ‘F’ means F-order, ‘A’ (short for *Any*) means ‘F’ if *a* is Fortran contiguous, ‘C’ otherwise. Default is ‘C’.

Returns

s
[bytes] Python bytes exhibiting a copy of *a*’s raw data.

See also

frombuffer
Inverse of this operation, construct a 1-dimensional array from Python bytes.

Examples

```
>>> import numpy as np
>>> x = np.array([[0, 1], [2, 3]], dtype='<u2')
>>> x.tobytes()
b'\x00\x00\x01\x00\x02\x00\x03\x00'
```

(continues on next page)

(continued from previous page)

```
>>> x.tobytes('C') == x.tobytes()
True
>>> x.tobytes('F')
b'\x00\x00\x02\x00\x01\x00\x03\x00'
```

tofile

`Array.tofile(fid, /, sep='', format='%s')`

Write array to a file as text or binary (default).

Data is always written in 'C' order, independent of the order of *a*. The data produced by this method can be recovered using the function fromfile().

Parameters

fid
[file or str or Path] An open file object, or a string containing a filename.

sep
[str] Separator between array items for text output. If "" (empty), a binary file is written, equivalent to `file.write(a.tobytes())`.

format
[str] Format string for text file output. Each entry in the array is formatted to text by first converting it to the closest Python type, and then using "format" % item.

Notes

This is a convenience function for quick storage of array data. Information on endianness and precision is lost, so this method is not a good choice for files intended to archive data or transport data between machines with different endianness. Some of these problems can be overcome by outputting the data as text files, at the expense of speed and file size.

When fid is a file object, array contents are directly written to the file, bypassing the file object's `write` method. As a result, tofile cannot be used with files objects supporting compression (e.g., GzipFile) or file-like objects that do not support `fileno()` (e.g., BytesIO).

tolist

`Array.tolist()`

Return the array as an `a.ndim`-levels deep nested list of Python scalars.

Return a copy of the array data as a (nested) Python list. Data items are converted to the nearest compatible builtin Python type, via the *~numpy.ndarray.item* method.

If `a.ndim` is 0, then since the depth of the nested list is 0, it will not be a list at all, but a simple Python scalar.

Parameters

none

Returns

y
[object, or list of object, or list of list of object, or …] The possibly nested list of array elements.

Notes

The array may be recreated via `a = np.array(a.tolist())`, although this may sometimes lose precision.

Examples

For a 1D array, `a.tolist()` is almost the same as `list(a)`, except that `tolist` changes numpy scalars to Python scalars:

```
>>> import numpy as np
>>> a = np.uint32([1, 2])
>>> a_list = list(a)
>>> a_list
[np.uint32(1), np.uint32(2)]
>>> type(a_list[0])
<class 'numpy.uint32'>
>>> a_tolist = a.tolist()
>>> a_tolist
[1, 2]
>>> type(a_tolist[0])
<class 'int'>
```

Additionally, for a 2D array, `tolist` applies recursively:

```
>>> a = np.array([[1, 2], [3, 4]])
>>> list(a)
[array([1, 2]), array([3, 4])]
>>> a.tolist()
[[1, 2], [3, 4]]
```

The base case for this recursion is a 0D array:

```
>>> a = np.array(1)
>>> list(a)
Traceback (most recent call last):
  ...
TypeError: iteration over a 0-d array
>>> a.tolist()
1
```

trace

`Array.trace`(*offset=0*, *axis1=0*, *axis2=1*, *dtype=None*, *out=None*)

Return the sum along diagonals of the array.

Refer to *numpy.trace* for full documentation.

See Also

numpy.trace : equivalent function

transpose

`Array.transpose(*axes)`

Returns a view of the array with axes transposed.

Refer to *numpy.transpose* for full documentation.

Parameters

axes : None, tuple of ints, or *n* ints

- None or no argument: reverses the order of the axes.
- tuple of ints: *i* in the *j*-th place in the tuple means that the array's *i*-th axis becomes the transposed array's *j*-th axis.
- *n* ints: same as an n-tuple of the same ints (this form is intended simply as a "convenience" alternative to the tuple form).

Returns

p

[ndarray] View of the array with its axes suitably permuted.

See Also

transpose : Equivalent function. ndarray.T : Array property returning the array transposed. ndarray.reshape : Give a new shape to an array without changing its data.

Examples

```
>>> import numpy as np
>>> a = np.array([[1, 2], [3, 4]])
>>> a
array([[1, 2],
       [3, 4]])
>>> a.transpose()
array([[1, 3],
       [2, 4]])
>>> a.transpose((1, 0))
array([[1, 3],
       [2, 4]])
>>> a.transpose(1, 0)
array([[1, 3],
       [2, 4]])
```

```
>>> a = np.array([1, 2, 3, 4])
>>> a
array([1, 2, 3, 4])
>>> a.transpose()
array([1, 2, 3, 4])
```

var

`Array.var`(*axis=None*, *dtype=None*, *out=None*, *ddof=0*, ***, *keepdims=<no value>*, *where=<no value>*, *mean=<no value>*)

Returns the variance of the array elements, along given axis.

Refer to *numpy.var* for full documentation.

See Also

numpy.var : equivalent function

view

`Array.view`(*[dtype][, type]*)

New view of array with the same data.

Note

Passing None for `dtype` is different from omitting the parameter, since the former invokes `dtype(None)` which is an alias for `dtype('float64')`.

Parameters

dtype
[data-type or ndarray sub-class, optional] Data-type descriptor of the returned view, e.g., float32 or int16. Omitting it results in the view having the same data-type as *a*. This argument can also be specified as an ndarray sub-class, which then specifies the type of the returned object (this is equivalent to setting the `type` parameter).

type
[Python type, optional] Type of the returned view, e.g., ndarray or matrix. Again, omission of the parameter results in type preservation.

Notes

`a.view()` is used two different ways:

`a.view(some_dtype)` or `a.view(dtype=some_dtype)` constructs a view of the array's memory with a different data-type. This can cause a reinterpretation of the bytes of memory.

`a.view(ndarray_subclass)` or `a.view(type=ndarray_subclass)` just returns an instance of *ndarray_subclass* that looks at the same array (same shape, dtype, etc.) This does not cause a reinterpretation of the memory.

For `a.view(some_dtype)`, if `some_dtype` has a different number of bytes per entry than the previous dtype (for example, converting a regular array to a structured array), then the last axis of `a` must be contiguous. This axis will be resized in the result.

Changed in version 1.23.0: Only the last axis needs to be contiguous. Previously, the entire array had to be C-contiguous.

Examples

```
>>> import numpy as np
>>> x = np.array([(-1, 2)], dtype=[('a', np.int8), ('b', np.int8)])
```

Viewing array data using a different type and dtype:

```
>>> nonneg = np.dtype([("a", np.uint8), ("b", np.uint8)])
>>> y = x.view(dtype=nonneg, type=np.recarray)
>>> x["a"]
array([-1], dtype=int8)
>>> y.a
array([255], dtype=uint8)
```

Creating a view on a structured array so it can be used in calculations

```
>>> x = np.array([(1, 2),(3,4)], dtype=[('a', np.int8), ('b', np.int8)])
>>> xv = x.view(dtype=np.int8).reshape(-1,2)
>>> xv
array([[1, 2],
       [3, 4]], dtype=int8)
>>> xv.mean(0)
array([2.,  3.])
```

Making changes to the view changes the underlying array

```
>>> xv[0,1] = 20
>>> x
array([(1, 20), (3,  4)], dtype=[('a', 'i1'), ('b', 'i1')])
```

Using a view to convert an array to a recarray:

```
>>> z = x.view(np.recarray)
>>> z.a
array([1, 3], dtype=int8)
```

Views share data:

```
>>> x[0] = (9, 10)
>>> z[0]
np.record((9, 10), dtype=[('a', 'i1'), ('b', 'i1')])
```

Views that change the dtype size (bytes per entry) should normally be avoided on arrays defined by slices, transposes, fortran-ordering, etc.:

```
>>> x = np.array([[1, 2, 3], [4, 5, 6]], dtype=np.int16)
>>> y = x[:, ::2]
>>> y
array([[1, 3],
       [4, 6]], dtype=int16)
>>> y.view(dtype=[('width', np.int16), ('length', np.int16)])
Traceback (most recent call last):
    ...
ValueError: To change to a dtype of a different size, the last axis must be␣
```

(continues on next page)

(continued from previous page)

```
↪contiguous
>>> z = y.copy()
>>> z.view(dtype=[('width', np.int16), ('length', np.int16)])
array([[(1, 3)],
       [(4, 6)]], dtype=[('width', '<i2'), ('length', '<i2')])
```

However, views that change dtype are totally fine for arrays with a contiguous last axis, even if the rest of the axes are not C-contiguous:

```
>>> x = np.arange(2 * 3 * 4, dtype=np.int8).reshape(2, 3, 4)
>>> x.transpose(1, 0, 2).view(np.int16)
array([[[ 256,  770],
        [3340, 3854]],

       [[1284, 1798],
        [4368, 4882]],

       [[2312, 2826],
        [5396, 5910]]], dtype=int16)
```

`__init__()`

Attributes

T	View of the transposed array.
base	Base object if memory is from some other object.
ctypes	An object to simplify the interaction of the array with the ctypes module.
data	Python buffer object pointing to the start of the array's data.
device	
dtype	Data-type of the array's elements.
flags	Information about the memory layout of the array.
flat	A 1-D iterator over the array.
imag	The imaginary part of the array.
itemsize	Length of one array element in bytes.
mT	View of the matrix transposed array.
nbytes	Total bytes consumed by the elements of the array.
ndim	Number of array dimensions.
real	The real part of the array.
shape	Tuple of array dimensions.
size	Number of elements in the array.
strides	Tuple of bytes to step in each dimension when traversing an array.

T

`Array.T`

View of the transposed array.

Same as `self.transpose()`.

Examples

```
>>> import numpy as np
>>> a = np.array([[1, 2], [3, 4]])
>>> a
array([[1, 2],
       [3, 4]])
>>> a.T
array([[1, 3],
       [2, 4]])
```

```
>>> a = np.array([1, 2, 3, 4])
>>> a
array([1, 2, 3, 4])
>>> a.T
array([1, 2, 3, 4])
```

See Also

transpose

base

`Array.base`

Base object if memory is from some other object.

Examples

The base of an array that owns its memory is None:

```
>>> import numpy as np
>>> x = np.array([1,2,3,4])
>>> x.base is None
True
```

Slicing creates a view, whose memory is shared with x:

```
>>> y = x[2:]
>>> y.base is x
True
```

ctypes

`Array.ctypes`

An object to simplify the interaction of the array with the ctypes module.

This attribute creates an object that makes it easier to use arrays when calling shared libraries with the ctypes module. The returned object has, among others, data, shape, and strides attributes (see Notes below) which themselves return ctypes objects that can be used as arguments to a shared library.

Parameters

None

Returns

c
: [Python object] Possessing attributes data, shape, strides, etc.

See Also

numpy.ctypeslib

Notes

Below are the public attributes of this object which were documented in "Guide to NumPy" (we have omitted undocumented public attributes, as well as documented private attributes):

`_ctypes.`**`data`**

A pointer to the memory area of the array as a Python integer. This memory area may contain data that is not aligned, or not in correct byte-order. The memory area may not even be writeable. The array flags and data-type of this array should be respected when passing this attribute to arbitrary C-code to avoid trouble that can include Python crashing. User Beware! The value of this attribute is exactly the same as: `self._array_interface_['data'][0]`.

Note that unlike `data_as`, a reference won't be kept to the array: code like `ctypes.c_void_p((a + b).ctypes.data)` will result in a pointer to a deallocated array, and should be spelt `(a + b).ctypes.data_as(ctypes.c_void_p)`

`_ctypes.`**`shape`**

(c_intp*self.ndim): A ctypes array of length self.ndim where the basetype is the C-integer corresponding to `dtype('p')` on this platform (see *~numpy.ctypeslib.c_intp*). This base-type could be *ctypes.c_int*, *ctypes.c_long*, or *ctypes.c_longlong* depending on the platform. The ctypes array contains the shape of the underlying array.

`_ctypes.`**`strides`**

(c_intp*self.ndim): A ctypes array of length self.ndim where the basetype is the same as for the shape attribute. This ctypes array contains the strides information from the underlying array. This strides information is important for showing how many bytes must be jumped to get to the next element in the array.

`_ctypes.`**`data_as`**`(`*obj*`)`

Return the data pointer cast to a particular c-types object. For example, calling `self._as_parameter_` is equivalent to `self.data_as(ctypes.c_void_p)`. Perhaps you want to use the data as a pointer to a ctypes array of floating-point data: `self.data_as(ctypes.POINTER(ctypes.c_double))`.

The returned pointer will keep a reference to the array.

`_ctypes.`**`shape_as`**`(`*obj*`)`

Return the shape tuple as an array of some other c-types type. For example: `self.shape_as(ctypes.c_short)`.

`_ctypes.`**`strides_as`**`(`*obj*`)`

Return the strides tuple as an array of some other c-types type. For example: `self.strides_as(ctypes.c_longlong)`.

If the ctypes module is not available, then the ctypes attribute of array objects still returns something useful, but ctypes objects are not returned and errors may be raised instead. In particular, the object will still have the `as_parameter` attribute which will return an integer equal to the data attribute.

Examples

```
>>> import numpy as np
>>> import ctypes
>>> x = np.array([[0, 1], [2, 3]], dtype=np.int32)
>>> x
array([[0, 1],
       [2, 3]], dtype=int32)
>>> x.ctypes.data
31962608 # may vary
>>> x.ctypes.data_as(ctypes.POINTER(ctypes.c_uint32))
<__main__.LP_c_uint object at 0x7ff2fc1fc200> # may vary
>>> x.ctypes.data_as(ctypes.POINTER(ctypes.c_uint32)).contents
c_uint(0)
>>> x.ctypes.data_as(ctypes.POINTER(ctypes.c_uint64)).contents
c_ulong(4294967296)
>>> x.ctypes.shape
<numpy._core._internal.c_long_Array_2 object at 0x7ff2fc1fce60> # may vary
>>> x.ctypes.strides
<numpy._core._internal.c_long_Array_2 object at 0x7ff2fc1ff320> # may vary
```

data

`Array.data`

Python buffer object pointing to the start of the array's data.

device

`Array.device`

dtype

`Array.dtype`

Data-type of the array's elements.

Warning

Setting `arr.dtype` is discouraged and may be deprecated in the future. Setting will replace the `dtype` without modifying the memory (see also *ndarray.view* and *ndarray.astype*).

Parameters

None

Returns

d : numpy dtype object

See Also

ndarray.astype : Cast the values contained in the array to a new data-type. ndarray.view : Create a view of the same data but a different data-type. numpy.dtype

Examples

```
>>> import numpy as np
>>> x = np.arange(4).reshape((2, 2))
>>> x
array([[0, 1],
       [2, 3]])
>>> x.dtype
dtype('int64')   # may vary (OS, bitness)
>>> isinstance(x.dtype, np.dtype)
True
```

flags

`Array.flags`

Information about the memory layout of the array.

Attributes

C_CONTIGUOUS (C)
The data is in a single, C-style contiguous segment.

F_CONTIGUOUS (F)
The data is in a single, Fortran-style contiguous segment.

OWNDATA (O)
The array owns the memory it uses or borrows it from another object.

WRITEABLE (W)
The data area can be written to. Setting this to False locks the data, making it read-only. A view (slice, etc.) inherits WRITEABLE from its base array at creation time, but a view of a writeable array may be subsequently locked while the base array remains writeable. (The opposite is not true, in that a view of a locked array may not be made writeable. However, currently, locking a base object does not lock any views that already reference it, so under that circumstance it is possible to alter the contents of a locked array via a previously created writeable view onto it.) Attempting to change a non-writeable array raises a RuntimeError exception.

ALIGNED (A)
The data and all elements are aligned appropriately for the hardware.

WRITEBACKIFCOPY (X)
This array is a copy of some other array. The C-API function PyArray_ResolveWritebackIfCopy must be called before deallocating to the base array will be updated with the contents of this array.

FNC
F_CONTIGUOUS and not C_CONTIGUOUS.

FORC
F_CONTIGUOUS or C_CONTIGUOUS (one-segment test).

BEHAVED (B)
ALIGNED and WRITEABLE.

CARRAY (CA)
BEHAVED and C_CONTIGUOUS.

FARRAY (FA)
BEHAVED and F_CONTIGUOUS and not C_CONTIGUOUS.

Notes

The *flags* object can be accessed dictionary-like (as in `a.flags['WRITEABLE']`), or by using lowercased attribute names (as in `a.flags.writeable`). Short flag names are only supported in dictionary access.

Only the WRITEBACKIFCOPY, WRITEABLE, and ALIGNED flags can be changed by the user, via direct assignment to the attribute or dictionary entry, or by calling *ndarray.setflags*.

The array flags cannot be set arbitrarily:

- WRITEBACKIFCOPY can only be set `False`.
- ALIGNED can only be set `True` if the data is truly aligned.
- WRITEABLE can only be set `True` if the array owns its own memory or the ultimate owner of the memory exposes a writeable buffer interface or is a string.

Arrays can be both C-style and Fortran-style contiguous simultaneously. This is clear for 1-dimensional arrays, but can also be true for higher dimensional arrays.

Even for contiguous arrays a stride for a given dimension `arr.strides[dim]` may be *arbitrary* if `arr.shape[dim] == 1` or the array has no elements. It does *not* generally hold that `self.strides[-1] == self.itemsize` for C-style contiguous arrays or `self.strides[0] == self.itemsize` for Fortran-style contiguous arrays is true.

flat

`Array.flat`

A 1-D iterator over the array.

This is a *numpy.flatiter* instance, which acts similarly to, but is not a subclass of, Python's built-in iterator object.

See Also

flatten : Return a copy of the array collapsed into one dimension.

flatiter

Examples

```
>>> import numpy as np
>>> x = np.arange(1, 7).reshape(2, 3)
>>> x
array([[1, 2, 3],
       [4, 5, 6]])
>>> x.flat[3]
```

(continues on next page)

(continued from previous page)

```
4
>>> x.T
array([[1, 4],
       [2, 5],
       [3, 6]])
>>> x.T.flat[3]
5
>>> type(x.flat)
<class 'numpy.flatiter'>
```

An assignment example:

```
>>> x.flat = 3; x
array([[3, 3, 3],
       [3, 3, 3]])
>>> x.flat[[1,4]] = 1; x
array([[3, 1, 3],
       [3, 1, 3]])
```

imag

`Array.imag`

The imaginary part of the array.

Examples

```
>>> import numpy as np
>>> x = np.sqrt([1+0j, 0+1j])
>>> x.imag
array([ 0.        ,  0.70710678])
>>> x.imag.dtype
dtype('float64')
```

itemsize

`Array.itemsize`

Length of one array element in bytes.

Examples

```
>>> import numpy as np
>>> x = np.array([1,2,3], dtype=np.float64)
>>> x.itemsize
8
>>> x = np.array([1,2,3], dtype=np.complex128)
>>> x.itemsize
16
```

mT

Array.**mT**

View of the matrix transposed array.

The matrix transpose is the transpose of the last two dimensions, even if the array is of higher dimension.

Added in version 2.0.

Raises

ValueError
If the array is of dimension less than 2.

Examples

```
>>> import numpy as np
>>> a = np.array([[1, 2], [3, 4]])
>>> a
array([[1, 2],
       [3, 4]])
>>> a.mT
array([[1, 3],
       [2, 4]])
```

```
>>> a = np.arange(8).reshape((2, 2, 2))
>>> a
array([[[0, 1],
        [2, 3]],

       [[4, 5],
        [6, 7]]])
>>> a.mT
array([[[0, 2],
        [1, 3]],

       [[4, 6],
        [5, 7]]])
```

nbytes

Array.**nbytes**

Total bytes consumed by the elements of the array.

Notes

Does not include memory consumed by non-element attributes of the array object.

See Also

sys.getsizeof
Memory consumed by the object itself without parents in case view. This does include memory consumed by non-element attributes.

Examples

```
>>> import numpy as np
>>> x = np.zeros((3,5,2), dtype=np.complex128)
>>> x.nbytes
480
>>> np.prod(x.shape) * x.itemsize
480
```

ndim

`Array.ndim`

Number of array dimensions.

Examples

```
>>> import numpy as np
>>> x = np.array([1, 2, 3])
>>> x.ndim
1
>>> y = np.zeros((2, 3, 4))
>>> y.ndim
3
```

real

`Array.real`

The real part of the array.

Examples

```
>>> import numpy as np
>>> x = np.sqrt([1+0j, 0+1j])
>>> x.real
array([ 1.        ,  0.70710678])
>>> x.real.dtype
dtype('float64')
```

See Also

numpy.real : equivalent function

shape

`Array.shape`

Tuple of array dimensions.

The shape property is usually used to get the current shape of an array, but may also be used to reshape the array in-place by assigning a tuple of array dimensions to it. As with *numpy.reshape*, one of the new shape dimensions can be -1, in which case its value is inferred from the size of the array and the remaining dimensions. Reshaping an array in-place will fail if a copy is required.

Warning

Setting `arr.shape` is discouraged and may be deprecated in the future. Using *ndarray.reshape* is the preferred approach.

Examples

```
>>> import numpy as np
>>> x = np.array([1, 2, 3, 4])
>>> x.shape
(4,)
>>> y = np.zeros((2, 3, 4))
>>> y.shape
(2, 3, 4)
>>> y.shape = (3, 8)
>>> y
array([[ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.]])
>>> y.shape = (3, 6)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: cannot reshape array of size 24 into shape (3,6)
>>> np.zeros((4,2))[::2].shape = (-1,)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: Incompatible shape for in-place modification. Use
`.reshape()` to make a copy with the desired shape.
```

See Also

numpy.shape : Equivalent getter function. numpy.reshape : Function similar to setting `shape`. ndarray.reshape : Method similar to setting `shape`.

size

`Array.size`

Number of elements in the array.

Equal to `np.prod(a.shape)`, i.e., the product of the array's dimensions.

Notes

a.size returns a standard arbitrary precision Python integer. This may not be the case with other methods of obtaining the same value (like the suggested `np.prod(a.shape)`, which returns an instance of `np.int_`), and may be relevant if the value is used further in calculations that may overflow a fixed size integer type.

Examples

```
>>> import numpy as np
>>> x = np.zeros((3, 5, 2), dtype=np.complex128)
```

(continues on next page)

(continued from previous page)

```
>>> x.size
30
>>> np.prod(x.shape)
30
```

strides

`Array.strides`

Tuple of bytes to step in each dimension when traversing an array.

The byte offset of element `(i[0], i[1], ..., i[n])` in an array *a* is:

```
offset = sum(np.array(i) * a.strides)
```

A more detailed explanation of strides can be found in arrays.ndarray.

> **Warning**
>
> Setting `arr.strides` is discouraged and may be deprecated in the future. *numpy.lib.stride_tricks.as_strided* should be preferred to create a new view of the same data in a safer way.

Notes

Imagine an array of 32-bit integers (each 4 bytes):

```
x = np.array([[0, 1, 2, 3, 4],
              [5, 6, 7, 8, 9]], dtype=np.int32)
```

This array is stored in memory as 40 bytes, one after the other (known as a contiguous block of memory). The strides of an array tell us how many bytes we have to skip in memory to move to the next position along a certain axis. For example, we have to skip 4 bytes (1 value) to move to the next column, but 20 bytes (5 values) to get to the same position in the next row. As such, the strides for the array *x* will be `(20, 4)`.

See Also

numpy.lib.stride_tricks.as_strided

Examples

```
>>> import numpy as np
>>> y = np.reshape(np.arange(2 * 3 * 4, dtype=np.int32), (2, 3, 4))
>>> y
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]],
       [[12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23]]], dtype=np.int32)
>>> y.strides
```

(continues on next page)

(continued from previous page)

```
(48, 16, 4)
>>> y[1, 1, 1]
np.int32(17)
>>> offset = sum(y.strides * np.array((1, 1, 1)))
>>> offset // y.itemsize
np.int64(17)
```

```
>>> x = np.reshape(np.arange(5*6*7*8, dtype=np.int32), (5, 6, 7, 8))
>>> x = x.transpose(2, 3, 1, 0)
>>> x.strides
(32, 4, 224, 1344)
>>> i = np.array([3, 5, 2, 2], dtype=np.int32)
>>> offset = sum(i * x.strides)
>>> x[3, 5, 2, 2]
np.int32(813)
>>> offset // x.itemsize
np.int64(813)
```

reshape(*shape*, /, *, *order='C'*, *copy=None*)

reshape(**shape*, *order='C'*, *copy=None*)

Returns an array containing the same data with a new shape.

Refer to *numpy.reshape* for full documentation.

See Also

numpy.reshape : equivalent function

Notes

Unlike the free function *numpy.reshape*, this method on *ndarray* allows the elements of the shape parameter to be passed in as separate arguments. For example, `a.reshape(4, 2)` is equivalent to `a.reshape((4, 2))`.

3.11.6 ranges

It provides Ranges class.

Classes

Ranges

3.11.6.1 Ranges

class Ranges(*ranges=()*, *values=None*)

Methods

__init__

continues on next page

Table 74 – continued from previous page

format_range
get_range
intersect
push
pushes
set_value
simplify

__init__

Ranges.**__init__**(*ranges=()*, *values=None*)

format_range

static Ranges.**format_range**(**args*, ***kwargs*)

get_range

static Ranges.**get_range**(*ref*, *context=None*, *raise_anchor=True*)

intersect

Ranges.**intersect**(*other*)

push

Ranges.**push**(*ref*, *value=empty*, *context=None*, *raise_anchor=True*)

pushes

Ranges.**pushes**(*refs*, *values=()*, *context=None*)

set_value

Ranges.**set_value**(*rng*, *value=empty*)

simplify

Ranges.**simplify**()

__init__(*ranges=()*, *values=None*)

Attributes

ranges
values
is_set
value

ranges

`Ranges.ranges`

values

`Ranges.values`

is_set

`property Ranges.is_set`

value

`property Ranges.value`

3.11.7 cell

It provides Cell class.

Functions

format_output
wrap_cell_func

3.11.7.1 format_output

format_output(*rng*, *value*)

3.11.7.2 wrap_cell_func

wrap_cell_func(*func*, *parse_args=<function <lambda>>*, *parse_kwargs=<function <lambda>>*)

Classes

Cell
CellWrapper
InvRangesAssembler
RangesAssembler
Ref

3.11.7.3 Cell

class Cell(*reference*, *value*, *context=None*, *check_formula=True*, *replace_missing_ref=True*, *raise_anchor=False*)

continues on next page

Table 78 – continued from previous page

Methods

__init__
add
compile
update_inputs

__init__

Cell.**__init__**(*reference*, *value*, *context=None*, *check_formula=True*, *replace_missing_ref=True*, *raise_anchor=False*)

add

Cell.**add**(*dsp*, *context=None*)

compile

Cell.**compile**(*references=None*, *context=None*)

update_inputs

Cell.**update_inputs**(*references=None*)

__init__(*reference*, *value*, *context=None*, *check_formula=True*, *replace_missing_ref=True*, *raise_anchor=False*)

Attributes

parser

parser

Cell.**parser = <formulas.parser.Parser object>**

3.11.7.4 CellWrapper

class **CellWrapper**(*func*, *parse_args*, *parse_kwargs*)

Methods

__init__
check_cycles

__init__

CellWrapper.**__init__**(*func*, *parse_args*, *parse_kwargs*)

check_cycles

CellWrapper.**check_cycles**(*cycle*)

__init__(*func*, *parse_args*, *parse_kwargs*)

3.11.7.5 InvRangesAssembler

class **InvRangesAssembler**(*assembler*)

Methods

__init__
add
push

__init__

InvRangesAssembler.**__init__**(*assembler*)

add

InvRangesAssembler.**add**(*dsp*)

push

InvRangesAssembler.**push**(*indices*, *output=None*)

__init__(*assembler*)

Attributes

output

output

property InvRangesAssembler.**output**

3.11.7.6 RangesAssembler

class **RangesAssembler**(*ref*, *context=None*, *compact=1*)

continues on next page

Table 83 – continued from previous page

Methods

__init__
add
push

__init__

RangesAssembler.**__init__**(*ref*, *context=None*, *compact=1*)

add

RangesAssembler.**add**(*dsp*)

push

RangesAssembler.**push**(*indices*, *output=None*)

__init__(*ref*, *context=None*, *compact=1*)

Attributes

output

output

property RangesAssembler.**output**

3.11.7.7 Ref

class Ref(*reference*, *value*, *context=None*, *check_formula=True*)

Methods

__init__
add
compile
update_inputs

__init__

Ref.**__init__**(*reference*, *value*, *context=None*, *check_formula=True*)

add

Ref.**add**(*dsp*, *context=None*)

compile

Ref.**compile**(*references=None*, *context=None*)

update_inputs

Ref.**update_inputs**(*references=None*)

__init__(*reference*, *value*, *context=None*, *check_formula=True*)

Attributes

parser

parser

Ref.**parser = <formulas.parser.Parser object>**

3.11.8 excel

It provides Excel model class.

Sub-Modules:

cycle	A dependency-free version of networkx's implementation of *simple_cycles*.
xlreader	It provides a custom Excel Reader class.

3.11.8.1 cycle

A dependency-free version of networkx's implementation of *simple_cycles*.

Functions

simple_cycles

simple_cycles

simple_cycles(*graph*, *copy=True*, *skip_nodes=()*)

3.11.8.2 xlreader

It provides a custom Excel Reader class.

Functions

load_workbook	
read_string_table	Read in all shared strings in the table
replace_hex	

load_workbook

load_workbook(*filename*, *_raw_data=False*, ***kw*)

read_string_table

read_string_table(*_raw_data*, *xml_source*)

Read in all shared strings in the table

replace_hex

replace_hex(*match*)

Classes

XlReader

XlReader

class XlReader(**args*, *raw_date=True*, *_raw_data=False*, ***kwargs*)

Methods

__init__
read
read_chartsheet
read_custom
read_manifest
read_properties
read_strings
read_theme
read_workbook
read_worksheets

__init__

XlReader.**__init__**(**args*, *raw_date=True*, *_raw_data=False*, ***kwargs*)

read

XlReader.**read**()

read_chartsheet

XlReader.**read_chartsheet**(*sheet*, *rel*)

read_custom

XlReader.**read_custom**()

read_manifest

XlReader.**read_manifest**()

read_properties

XlReader.**read_properties**()

read_strings

XlReader.**read_strings**()

read_theme

XlReader.**read_theme**()

read_workbook

XlReader.**read_workbook**()

read_worksheets

XlReader.**read_worksheets**()

__init__(**args*, *raw_date=True*, *_raw_data=False*, ***kwargs*)

Functions

escape_char

3.11.8.3 escape_char

escape_char(*m*)

Classes

ExcelModel
XlCircular

3.11.8.4 ExcelModel

class ExcelModel

Methods

__init__
add_anchor
add_book
add_cell
add_references
add_sheet
anchors
assemble
calculate
compare
compile
compile_cell
complete
external_links
finish
formula_ranges
formula_references
from_dict
from_ranges
inverse_references
load
loads
push
pushes
solve_circular
to_dict
write

__init__

ExcelModel.**__init__**()

add_anchor

ExcelModel.**add_anchor**(*rng*, *data_nodes=None*, *context=None*, *set_ref=True*)

add_book

ExcelModel.**add_book**(*book=None*, *context=None*, *data_only=False*)

add_cell

ExcelModel.**add_cell**(*cell*, *context*, *formula_ranges*)

add_references

ExcelModel.**add_references**(*book*, *context=None*)

add_sheet

ExcelModel.**add_sheet**(*worksheet*, *context*)

anchors

ExcelModel.**anchors**(*stack=None*)

assemble

ExcelModel.**assemble**(*compact=1*)

calculate

ExcelModel.**calculate**(**args*, ***kwargs*)

compare

ExcelModel.**compare**(**fpaths*, *target=None*, *actual=None*, *solution=None*, *books=None*, *dirpath=None*, *tolerance=0*, *absolute_tolerance=1e-06*, *dot_notation=False*, *formatted=True*, ***kwargs*)

compile

ExcelModel.**compile**(*inputs*, *outputs*)

compile_cell

ExcelModel.**compile_cell**(*cell*, *context*, *references*, *formula_references*)

complete

ExcelModel.**complete**(*stack=None*)

external_links

ExcelModel.**external_links**(*ctx*)

finish

ExcelModel.**finish**(*complete=True*, *circular=False*, *assemble=True*, *anchors=True*)

formula_ranges

ExcelModel.**formula_ranges**(*ctx*)

formula_references

ExcelModel.**formula_references**(*ctx*)

from_dict

ExcelModel.**from_dict**(*adict*, *context=None*, *assemble=True*, *ref=True*)

from_ranges

ExcelModel.**from_ranges**(**ranges*)

inverse_references

ExcelModel.**inverse_references**()

load

ExcelModel.**load**(*filename*)

loads

ExcelModel.**loads**(**file_names*)

push

ExcelModel.**push**(*worksheet*, *context*)

pushes

ExcelModel.**pushes**(**worksheets*, *context=None*)

solve_circular

ExcelModel.**solve_circular**()

to_dict

ExcelModel.**to_dict**()

write

ExcelModel.**write**(*books=None*, *solution=None*, *dirpath=None*)

__init__()

Attributes

references

references

property ExcelModel.**references**

compile_class

alias of DispatchPipe

3.11.8.5 XlCircular

class XlCircular(**args*)

Methods

__init__	
capitalize	Return a capitalized version of the string.
casefold	Return a version of the string suitable for caseless comparisons.
center	Return a centered string of length width.
count	Return the number of non-overlapping occurrences of substring sub in string S[start:end].
encode	Encode the string using the codec registered for encoding.
endswith	Return True if S ends with the specified suffix, False otherwise.
expandtabs	Return a copy where all tab characters are expanded using spaces.
find	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
format	Return a formatted version of S, using substitutions from args and kwargs.
format_map	Return a formatted version of S, using substitutions from mapping.
index	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
isalnum	Return True if the string is an alpha-numeric string, False otherwise.
isalpha	Return True if the string is an alphabetic string, False otherwise.
isascii	Return True if all characters in the string are ASCII, False otherwise.
isdecimal	Return True if the string is a decimal string, False otherwise.
isdigit	Return True if the string is a digit string, False otherwise.
isidentifier	Return True if the string is a valid Python identifier, False otherwise.
islower	Return True if the string is a lowercase string, False otherwise.
isnumeric	Return True if the string is a numeric string, False otherwise.
isprintable	Return True if the string is printable, False otherwise.
isspace	Return True if the string is a whitespace string, False otherwise.

continues on next page

Table 96 – continued from previous page

`istitle`	Return True if the string is a title-cased string, False otherwise.
`isupper`	Return True if the string is an uppercase string, False otherwise.
`join`	Concatenate any number of strings.
`ljust`	Return a left-justified string of length width.
`lower`	Return a copy of the string converted to lowercase.
`lstrip`	Return a copy of the string with leading whitespace removed.
`maketrans`	Return a translation table usable for str.translate().
`partition`	Partition the string into three parts using the given separator.
`removeprefix`	Return a str with the given prefix string removed if present.
`removesuffix`	Return a str with the given suffix string removed if present.
`replace`	Return a copy with all occurrences of substring old replaced by new.
`rfind`	Return the highest index in S where substring sub is found, such that sub is contained within S[start:end].
`rindex`	Return the highest index in S where substring sub is found, such that sub is contained within S[start:end].
`rjust`	Return a right-justified string of length width.
`rpartition`	Partition the string into three parts using the given separator.
`rsplit`	Return a list of the substrings in the string, using sep as the separator string.
`rstrip`	Return a copy of the string with trailing whitespace removed.
`split`	Return a list of the substrings in the string, using sep as the separator string.
`splitlines`	Return a list of the lines in the string, breaking at line boundaries.
`startswith`	Return True if S starts with the specified prefix, False otherwise.
`strip`	Return a copy of the string with leading and trailing whitespace removed.
`swapcase`	Convert uppercase characters to lowercase and lowercase characters to uppercase.
`title`	Return a version of the string where each word is titlecased.
`translate`	Replace each character in the string using the given translation table.
`upper`	Return a copy of the string converted to uppercase.
`zfill`	Pad a numeric string with zeros on the left, to fill a field of the given width.

__init__

XlCircular.**__init__**(**args*)

capitalize

XlCircular.**capitalize**()

Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower case.

casefold

XlCircular.**casefold**()

Return a version of the string suitable for caseless comparisons.

center

XlCircular.**center**(*width*, *fillchar=' '*, */*)

Return a centered string of length width.

Padding is done using the specified fill character (default is a space).

count

XlCircular.**count**(*sub*[, *start*[, *end*]]) → int

Return the number of non-overlapping occurrences of substring sub in string S[start:end]. Optional arguments start and end are interpreted as in slice notation.

encode

XlCircular.**encode**(*encoding='utf-8'*, *errors='strict'*)

Encode the string using the codec registered for encoding.

encoding
The encoding in which to encode the string.

errors
The error handling scheme to use for encoding errors. The default is 'strict' meaning that encoding errors raise a UnicodeEncodeError. Other possible values are 'ignore', 'replace' and 'xmlcharrefreplace' as well as any other name registered with codecs.register_error that can handle UnicodeEncodeErrors.

endswith

XlCircular.**endswith**(*suffix*[, *start*[, *end*]]) → bool

Return True if S ends with the specified suffix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. suffix can also be a tuple of strings to try.

expandtabs

XlCircular.**expandtabs**(*tabsize=8*)

Return a copy where all tab characters are expanded using spaces.

If tabsize is not given, a tab size of 8 characters is assumed.

find

XlCircular.**find**(*sub*[, *start*[, *end*]]) → int

Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

format

XlCircular.**format**(**args*, ***kwargs*) → str

Return a formatted version of S, using substitutions from args and kwargs. The substitutions are identified by braces ('{' and '}').

format_map

XlCircular.**format_map**(*mapping*) → str

Return a formatted version of S, using substitutions from mapping. The substitutions are identified by braces ('{' and '}').

index

XlCircular.**index**(*sub*[, *start*[, *end*]]) → int

Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Raises ValueError when the substring is not found.

isalnum

XlCircular.**isalnum**()

Return True if the string is an alpha-numeric string, False otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string.

isalpha

XlCircular.**isalpha**()

Return True if the string is an alphabetic string, False otherwise.

A string is alphabetic if all characters in the string are alphabetic and there is at least one character in the string.

isascii

`XlCircular.isascii()`

Return True if all characters in the string are ASCII, False otherwise.

ASCII characters have code points in the range U+0000-U+007F. Empty string is ASCII too.

isdecimal

`XlCircular.isdecimal()`

Return True if the string is a decimal string, False otherwise.

A string is a decimal string if all characters in the string are decimal and there is at least one character in the string.

isdigit

`XlCircular.isdigit()`

Return True if the string is a digit string, False otherwise.

A string is a digit string if all characters in the string are digits and there is at least one character in the string.

isidentifier

`XlCircular.isidentifier()`

Return True if the string is a valid Python identifier, False otherwise.

Call keyword.iskeyword(s) to test whether string s is a reserved identifier, such as "def" or "class".

islower

`XlCircular.islower()`

Return True if the string is a lowercase string, False otherwise.

A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

isnumeric

`XlCircular.isnumeric()`

Return True if the string is a numeric string, False otherwise.

A string is numeric if all characters in the string are numeric and there is at least one character in the string.

isprintable

`XlCircular.isprintable()`

Return True if the string is printable, False otherwise.

A string is printable if all of its characters are considered printable in repr() or if it is empty.

isspace

`XlCircular.isspace()`

Return True if the string is a whitespace string, False otherwise.

A string is whitespace if all characters in the string are whitespace and there is at least one character in the string.

istitle

`XlCircular.istitle()`

Return True if the string is a title-cased string, False otherwise.

In a title-cased string, upper- and title-case characters may only follow uncased characters and lowercase characters only cased ones.

isupper

`XlCircular.isupper()`

Return True if the string is an uppercase string, False otherwise.

A string is uppercase if all cased characters in the string are uppercase and there is at least one cased character in the string.

join

`XlCircular.join(iterable, /)`

Concatenate any number of strings.

The string whose method is called is inserted in between each given string. The result is returned as a new string.

Example: '.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'

ljust

`XlCircular.ljust(width, fillchar=' ', /)`

Return a left-justified string of length width.

Padding is done using the specified fill character (default is a space).

lower

`XlCircular.lower()`

Return a copy of the string converted to lowercase.

lstrip

`XlCircular.lstrip(chars=None, /)`

Return a copy of the string with leading whitespace removed.

If chars is given and not None, remove characters in chars instead.

maketrans

static XlCircular.**maketrans**()

Return a translation table usable for str.translate().

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in x will be mapped to the character at the same position in y. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

partition

XlCircular.**partition**(*sep*, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original string and two empty strings.

removeprefix

XlCircular.**removeprefix**(*prefix*, /)

Return a str with the given prefix string removed if present.

If the string starts with the prefix string, return string[len(prefix):]. Otherwise, return a copy of the original string.

removesuffix

XlCircular.**removesuffix**(*suffix*, /)

Return a str with the given suffix string removed if present.

If the string ends with the suffix string and that suffix is not empty, return string[:-len(suffix)]. Otherwise, return a copy of the original string.

replace

XlCircular.**replace**(*old*, *new*, *count=-1*, /)

Return a copy with all occurrences of substring old replaced by new.

count
Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument count is given, only the first count occurrences are replaced.

rfind

XlCircular.**rfind**(*sub*[, *start*[, *end*]]) → int

Return the highest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

rindex

XlCircular.**rindex**(*sub*[, *start*[, *end*]]) → int

Return the highest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Raises ValueError when the substring is not found.

rjust

XlCircular.**rjust**(*width*, *fillchar=' '*, */*)

Return a right-justified string of length width.

Padding is done using the specified fill character (default is a space).

rpartition

XlCircular.**rpartition**(*sep*, */*)

Partition the string into three parts using the given separator.

This will search for the separator in the string, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty strings and the original string.

rsplit

XlCircular.**rsplit**(*sep=None*, *maxsplit=-1*)

Return a list of the substrings in the string, using sep as the separator string.

sep
The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including n r t f and spaces) and will discard empty strings from the result.

maxsplit
Maximum number of splits. -1 (the default value) means no limit.

Splitting starts at the end of the string and works to the front.

rstrip

XlCircular.**rstrip**(*chars=None*, */*)

Return a copy of the string with trailing whitespace removed.

If chars is given and not None, remove characters in chars instead.

split

XlCircular.**split**(*sep=None*, *maxsplit=-1*)

Return a list of the substrings in the string, using sep as the separator string.

sep
The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including n r t f and spaces) and will discard empty strings from the result.

maxsplit
Maximum number of splits. -1 (the default value) means no limit.

Splitting starts at the front of the string and works to the end.

Note, str.split() is mainly useful for data that has been intentionally delimited. With natural text that includes punctuation, consider using the regular expression module.

splitlines

`XlCircular.splitlines(keepends=False)`

Return a list of the lines in the string, breaking at line boundaries.

Line breaks are not included in the resulting list unless keepends is given and true.

startswith

`XlCircular.startswith(prefix[, start[, end]]) → bool`

Return True if S starts with the specified prefix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. prefix can also be a tuple of strings to try.

strip

`XlCircular.strip(chars=None, /)`

Return a copy of the string with leading and trailing whitespace removed.

If chars is given and not None, remove characters in chars instead.

swapcase

`XlCircular.swapcase()`

Convert uppercase characters to lowercase and lowercase characters to uppercase.

title

`XlCircular.title()`

Return a version of the string where each word is titlecased.

More specifically, words start with uppercased characters and all remaining cased characters have lower case.

translate

`XlCircular.translate(table, /)`

Replace each character in the string using the given translation table.

table
Translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, strings, or None.

The table must implement lookup/indexing via __getitem__, for instance a dictionary or list. If this operation raises LookupError, the character is left untouched. Characters mapped to None are deleted.

upper

`XlCircular.upper()`

Return a copy of the string converted to uppercase.

zfill

`XlCircular.zfill(width, /)`

Pad a numeric string with zeros on the left, to fill a field of the given width.

The string is never truncated.

`__init__(*args)`

3.12 Changelog

3.12.1 v1.3.4 (2026-03-11)

3.12.1.1 Feat

- (test): Add test for Excel function coverage and integrate README summary.
- (cli): add integration demos and docs.
- (cli): Enhance CLI with expanded commands, detailed help, and comprehensive examples.
- (cli): Add *serve* command with Flask endpoints for health, model info, and calculation.
- (test): Add extensive tests for CLI model calculations and batch processing.
- (cli): Add comprehensive CLI for model building, testing, and calculations.

3.12.1.2 Fix

- (test): Adjust regex in test_cell for improved accuracy.

3.12.1.3 Other

- Chore(docs): Remove CLI documentation and integration examples.

3.12.2 v1.3.3 (2025-11-04)

3.12.2.1 Feat

- (test): Update all test cases.

3.12.2.2 Fix

- (logic): Remove na check for if conditions.
- (look): Correct INDEX Function result.

3.12.3 v1.3.2 (2025-10-21)

3.12.3.1 Feat

- (core): Make parse dynamic arrays.
- (test): Update all test cases.

- (excel): Convert ranges before writing on Excel file.
- (excel): Add log when reading files.
- (math): Add *MULTINOMIAL*.
- (math): Add *SUMX2MY2*, *SUMX2PY2*, *SUMXMY2*.
- (compatibility): Add *BETAINV*, *BINOMDIST*, *CHIDIST*, *CHIINV*, *CHITEST*, *CHIDIST*, *CONFIDENCE*, *COVAR*, *CRITBINOM*, *EXPONDIST*, *FDIST*, *FINV*, *FTEST*, *GAMMADIST*, *GAMMAINV*, *LOGINV*, *MODE*, *PERCENTRANK*, *POISSON*, *RANK*, *TINV*, *TTEST*, *WEIBULL*, *ZTEST*, *BETADIST*, *HYPGEOMDIST*, *LOGNORMDIST*, *NEGBINOMDIST*, *TDIST*.
- (financial): Add *VDB*.
- (financial): Add *ODDFPRICE*, *ODDFYIELD*.
- (financial): Add *ODDLPRICE*, *ODDLYIELD*.
- (date): Add *NETWORKDAYS.INTL*, *NETWORKDAYS*, *WORKDAY.INTL*, *WORKDAY*.
- (date): Add *DAYS360*, *DAYS*.
- (financial): Add *TBILLEQ*, *TBILLPRICE*, *TBILLYIELD*.
- (financial): Add *PRICE*, *PRICEDISC*, *PRICEMAT*, *YIELD*, *YIELDDISC*, *YIELDMAT*, *AMORLINC*, *AMORDEGRC*.
- (financial): Add *ACCRINT* with some limitation on basis 1,2,3.
- (financial): Add *FVSCHEDULE*, *ISPMT*, *SLN*, *SYD*.
- (financial): Add *INTRATE*, *RECEIVED*, *DISC*, *DB*, *DDB*, *DOLLARDE*, *DOLLARFR*.
- (financial): Add *ACCRINTM*, *COUPNUM*, *COUPNCD*, *COUPPCD*, *COUPDAYS*, *COUPDAYBS*, *COUPDAYSNC*, *DURATION*, *MDURATION*, *PDURATION*, *RRI*, *EFFECT*, *NOMINAL*, *CUMPRINC*, *MIRR*.

3.12.3.2 Fix

- (text): Remove unused code.
- (date): Correct *DAYS360* calculation for basis==0.
- (look): Avoid running INDEX during compilation.

3.12.4 v1.3.1 (2025-09-15)

3.12.4.1 Feat

- (test): Update all test cases.
- (logic): Add *MAKEARRAY*, *BYCOL*, *BYROW*, *REDUCE*, *SCAN*, *MAP*.
- (info): Add *ERROR.TYPE*, *ISREF*, *N*, *TYPE*, *ISFORMULA*.
- (look): Add *TRIMRANGE*.
- (look): Add *GROUPBY*, *DROP*, *TAKE*, *EXPAND*, *HSTACK*, *VSTACK*, *WRAPROWS*, *WRAPCOLS*.
- (operand): Add new error *#GETTING_DATA*.
- (logic): Remove unused functions.
- (math): Add *PERCENTOF*.
- (tokens): Add function _xleta type.
- (tokens): Implement parser logic for complex LAMBDA function calls.

- (look): Add *UNIQUE*, *SORT*, *SORTBY*, *PIVOTBY* functions.
- (logic): Add *LAMBDA*, *LET* functions.
- (look): Add *AREAS*, *COLUMNS*, *ROWS*, *CHOOSE*, *CHOOSECOLS*, *CHOOSEROWS*, *TOCOL*, *TOROW*.
- (excel): Change order of compare results.
- (look): Add *XMATCH* and *XLOOKUP*.
- (functions): Add *SKEW.P*, *SKEW*, *MODE.SNGL*, *MODE.MULT*, *KURT*, *HARMEAN*, *GEOMEAN*, *DEVSQ*, *AVEDEV*, *INTERCEPT*, *FORECAST.ETS.STAT*, *FORECAST.ETS.SEASONALITY*, *FORECAST.ETS.CONFINT*, *FORECAST.ETS*, *WEIBULL.DIST*, *LOGNORM.DIST*, *LOGNORM.INV*, *BETA.DIST*, *BETA.INV*, *NEGBINOM.DIST*, *BINOM.DIST*, *BINOM.DIST.RANGE*, *BINOM.INV*, *CHISQ.DIST*, *CHISQ.INV*, *CHISQ.DIST.RT*, *CHISQ.INV.RT*, *CHISQ.TEST*, *CONFIDENCE.NORM*, *CONFIDENCE.T*, *COVARIANCE.P*, *COVARIANCE.S*, *F.DIST*, *F.INV*, *F.DIST.RT*, *F.INV.RT*, *F.TEST*, *T.DIST*, *T.INV*, *T.DIST.2T*, *T.DIST.RT*, *T.INV.2T*, *T.TEST*, *EXPON.DIST*, *POISSON.DIST*, *FISHER*, *FISHERINV*, *PHI*, *GAMMA*, *GAMMA.DIST*, *GAMMA.INV*, *GAMMALN.PRECISE*, *GAMMALN*, *GAUSS*, *HYPGEOM.DIST*, *PEARSON*, *PERCENTRANK.EXC*, *PERCENTRANK.INC*, *PERMUT*, *PERMUTATIONA*, *RANK.EQ*, *RANK.AVG*, *RSQ*, *STEYX*, *STANDARDIZE*, *TRIMMEAN*, *Z.TEST*, *FREQUENCY*, *PROB*, *LINEST*, *LOGEST*, *TREND*, *GROWTH*.

3.12.4.2 Fix

- (look): Implement missing functionalities of *PIVOTBY*.

3.12.5 v1.3.0 (2025-08-20)

3.12.5.1 Feat

- (setup): Add package_data.
- (excel): Improve readability of Excel comparison method.
- (functions): Add *ASC*, *BAHTTEXT*, *CLEAN*, *CHAR*, *UNICHAR*, *UNICODE*, *EXACT*, *FINDB*, *LEFTB*, *LENB*, *MIDB*, *NUMBERVALUE*, *PROPER*, *REGEXEXTRACT*, *REGEXREPLACE*, *REGEXTEST*, *REPLACEB*, *REPT*, *RIGHTB*, *SEARCHB*, *VALUETOTEXT*, *ARRAYTOTEXT*, *FIXED*, *TEXTSPLIT*, *TEXTAFTER*, *TEXTBEFORE*.
- (functions): Add *BESSELJ*, *BESSELI*, *BESSELK*, *BESSELY*, *BITAND*, *BITOR*, *BITXOR*, *BITLSHIFT*, *BITRSHIFT*, *CONVERT*, *ERF*, *ERF.PRECISE*, *ERFC*, *ERFC.PRECISE*, *DELTA*, *GESTEP*, *COMPLEX*, *IMDIV*, *IMSUB*, *IMSUM*, *IMPRODUCT*, *IMABS*, *IMREAL*, *IMAGINARY*, *IMARGUMENT*, *IMCONJUGATE*, *IMCOS*, *IMCOSH*, *IMCOT*, *IMCSC*, *IMCSCH*, *IMEXP*, *IMLN*, *IMLOG10*, *IMLOG2*, *IMSEC*, *IMSECH*, *IMSIN*, *IMSINH*, *IMSQRT*, *IMTAN*, *IMPOWER*.
- (operators): Add support for Implicit intersection operator @.
- (core): Add *.ods* file support.

3.12.5.2 Fix

- (excel): Correct external link of *.ods* files.
- (setup): Add missing requirements.
- (excel): Parse properly dynamic arrays also from JSON format.
- (look): Correct *FILTER* function was changing input values.
- (text): Correct *T* function behaviour.
- (token): Correct compile error for excel # *Errors*.
- (operators): Correct behaviour of = operator *"A"="a"* return now *TRUE* like excel.

- (excel): Correct handling of Excel Illegal Character.

3.12.5.3 Other

- Update test cases to improve coverage.
- Add new test cases for *.ods* file.
- Add new test cases for all new functions.

3.12.6 v1.2.11 (2025-07-28)

3.12.6.1 Fix

- (test): Update test cases.
- (look) #170: Correct *MATCH* and *SUMIFS* behaviour.
- (math): Update *RANDBETWEEN* error handling.
- (look): Correct INDEX function bug beacuse of new numpy 2.4.
- (doc): Correct readme badges.

3.12.7 v1.2.10 (2025-05-21)

3.12.7.1 Feat

- (functions): Add *EOMONTH*, *SUMIFS*, *AVERAGEIFS*, *COUNTIFS*, *MAXIFS*, *MINIFS*.

3.12.8 v1.2.9 (2025-04-05)

3.12.8.1 Feat

- (core): Update test cases.
- (core): Update python versions.
- (excel): Add *InvRangesAssembler*.
- (functions): Add *FILTER*, *TRANSPOSE*, *SUBSTITUTE*, *TEXTJOIN*, *T*.
- (doc): Add JetBrains sponsor.
- (look): Improve performances of *MATCH* and *LOOKUP*.
- (math): Add *MDETERM*, *MINVERSE*, and *MMULT* functions.
- (core): Add ANCHORARRAY functionality.

3.12.8.2 Fix

- (text) #146: Correct TEXT function date formatter logic.
- (functions) #147: Correct array collapse behaviour.
- (text) #149, #158: Add default value of LEFT and RIGHT functions.
- (core) #159: Correct parsing error.
- (doc): Correct doctests.
- (text): Add missing function *_XLFN.CONCATENATE*.
- (doc): Correct documentation issue.

3.12.9 v1.2.8 (2024-07-16)

3.12.9.1 Feat

- (core): Update Copyright.
- (functions) #109, #111, #124, #125: Update test cases.
- (stat) #111: Add *PERCENTILE*, *PERCENTILE.INC*, and *PERCENTILE.EXC* functions.
- (stat) #111: Add *NORM.S.DIST*, *NORM.S.INV*, *NORM.DIST*, *NORM.INV*,`NORMDIST`, *NORMINV*,`NORMSINV` functions.
- (stat) #111: Add *NORMSDIST* function.
- (stat) #124: Correct implementation *QUARTILE* and add *QUARTILE.INC* and *QUARTILE.EXC*.
- (functions) #124: Add *QUARTILE* to stat functions.
- (functions) #125: Add *SUMSQ* to stat functions.
- (tokens) #139: Allow last parameters to be empty in a function call.
- (tokens) #139: Allow first param to be empty.
- (core): Update *.gitignore* settings.
- (text): Add *CODE* function.
- (text): Add *CHAR* function.
- (test): Update coverage python version.

3.12.9.2 Fix

- (test) #111: Correct test case for windows.
- (excel) #109: Correct parser for named range with backslash in name.
- (functions) #125: Move *SUMSQ* function to math.
- (core): Correct repr formatting of ranges for numpy version 2.x.
- (tokens) #145: Correct handling of *#REF!* when compiling functions.
- (text): Correct *CODE* function.
- (text): Add *CODE* text case.
- (excel) #132: Correction on how to handle the empty values used within a formula.
- (excel): Add *#EMPTY* value to save correctly the model as dict.
- (excel) #134, #135: Correct *inverse_references* handling when model defined with *from_dict*.
- (excel): Correct tolerance.
- (setup): Correct setup config file.

3.12.10 v1.2.7 (2023-11-14)

3.12.10.1 Feat

- (builder) #104: Allow custom reference definition.
- (test): Update test cases.
- (operand) #106: Accept number like *.3* to be parsed.

- (text) #113: Add *TEXT* function without fraction formatting.
- (logic): Update logic functions according to new excel logic.
- (text) #113: Add *VALUE* function.
- (math) #121: Improve performances of *SUMPRODUCT*, *PRODUCT*, *SUM*, and *SUMIF*.
- (setup): Update requirements.
- (core): Change development status.
- (core): Add support for python 3.10 and 3.11.
- (functions) #121: Improve handling of EMPTY values.
- (excel): Avoid using *flatten* function in basic routines.
- (doc): Add Read the Docs configuration file.
- (excel): Add tolerance when comparing two excels.
- (excel): Add compare method to verify if formulas is able to replicate excel values.

3.12.10.2 Fix

- (doc): Remove broken badge.
- (excel) #100: Correct reading rounding from excel.
- (math) #100: Correct *TRUNC* defaults.
- (tokens) #113: Correct *sheet_id* definition.
- (functions): Correct dill pickling error.
- (excel): Correct reference parsing when loading from JSON.
- (functions): Use an alternative method of vectorize when more than 32 arguments are provided.
- (look): Correct *MATCH*, *LOOKUP*,`HLOOKUP`, and *VLOOKUP* behaviour when empty values are given.
- (date): Correct *DATEDIF* behaviour when unit is lowercase.
- (test): Use regex for unstable tests due to changes in last digits.
- (doc): Correct documentation bug due to new *sphinx*.
- (excel) #114: Update reading code according to *openpyxl>=3.1*.

3.12.11 v1.2.6 (2022-12-13)

3.12.11.1 Fix

- (setup): Update *schedula* requirement.

3.12.12 v1.2.5 (2022-11-07)

3.12.12.1 Fix

- (parser): Correct missing raise.
- (excel): Skip hidden named ranges.

3.12.13 v1.2.4 (2022-07-02)

3.12.13.1 Feat

- (core): Improve speed performance.
- (cell): Improve speed *RangesAssembler* definition.

3.12.13.2 Fix

- (cell): Correct range assembler defaults when no *sheet_id* is defined.
- (math) #99: Convert args into np.arrays in func *xsumproduct*.
- (look): Correct lookup parser for float and strings.

3.12.14 v1.2.3 (2022-05-10)

3.12.14.1 Feat

- (test): Add more error logs.
- (test): Improve code coverage.
- (builder): Add *compile_class* attribute to *AstBuilder*.
- (info): Add *ISODD*, *ISEVEN*, *ISBLANK*, *ISTEXT*, *ISNONTEXT*, and *ISLOGICAL* functions.

3.12.14.2 Fix

- (excel): Correct file path excel definition.
- (logic): Correct *SWITCH* error handling.
- (actions): Rename workflow name.
- (readme): Correct badge link for dependencies status.
- (excel): Correct *basedir* reference to load files.
- (date): Correct *YEARFRAC* and *DATEDIF* formulation.
- (cell): Enable R1C1 notation for absolute and relative references.
- (cell): Correct RangeAssembler value assignment.

3.12.15 v1.2.2 (2022-01-22)

3.12.15.1 Fix

- (excel): Correct function compilation from excel.

3.12.16 v1.2.1 (2022-01-21)

3.12.16.1 Feat

- (functions): Improve performances caching results.
- (excel): Make replacing missing ref optional in *from_dict* method.
- (excel) #73, #75: Improve performances to parse full ranges.

3.12.16.2 Fix

- (excel): Correct compile function when inputs are computed with a default function.

3.12.17 v1.2.0 (2021-12-23)

3.12.17.1 Feat

- (binder): Refresh environment binder for 2021.
- (look) #87: Add *ADDRESS* function.
- (test): Update test cases.
- (financial) #74, #87: Add *FV*, *PV*, *IPMT*, *PMT*, *PPMT*, *RATE*, *CUMIPMT*, and *NPER* functions.
- (info, logic): Add *ISNA* and *IFNA* functions.
- (date) #87: Add *WEEKDAY*, *WEEKNUM*, *ISOWEEKNUM*, and *DATEDIF* functions.
- (stat, math) #87: Add *SLOPE* and *PRODUCT* functions.
- (stats) #87: Add *CORREL* and *MEDIAN* functions.
- (bin): Add *bin* folder.
- (actions): Add test cases.
- (stats) #80: Add *FORECAST* and *FORECAST.LINEAR* functions.
- (excel) #82: Add inverse of simple references.

3.12.17.2 Fix

- (stat): Correct *LARGE* and *SMALL* error handling.
- (actions): Skip *Setup Graphviz* when not needed.
- (actions): Correct coverall setting.
- (actions): Remove unstable test case.
- (actions): Disable fail fast.
- (date, stat): Correct collapsed return value.
- (function) #78, #79, #91: Correct import error.

3.12.18 v1.1.1 (2021-10-13)

3.12.18.1 Feat

- (excel): Improve performances of *complete* method.
- (setup): Add add python 3.9 in setup.py.
- (functions): Add *SEARCH*, *ISNUMBER*, and *EDATE* functions.
- (travis): Update python version for coveralls.

3.12.18.2 Fix

- (doc): Correct missing documentation link.
- (doc): Correct typo.
- (operator) #70: Correct % operator preceded by space.

3.12.19 v1.1.0 (2021-02-16)

3.12.19.1 Feat

- (look) #57: Add *SINGLE* function.
- (function) #51: Add google Excel functions.
- (logic) #55, #57: Add IFS function.
- (excel) #65: Add documentation and rename method to load models from ranges.
- (excel) #65: Add method to load sub-models from range.
- (doc): Update Copyright.
- (excel): Improve performances.
- (excel) #64: Read model from outputs.
- (core): Update range definition with path file.
- (excel) #64: Add warning for missing reference.
- (excel) #64: Add warning message when book loading fails.
- (readme) #44: Add example to export and import the model to JSON format.
- (readme) #53: Add instructions to install the development version.
- (excel) #44: Add feature to export and import the model to JSON- able dict.
- (stat, comp) #43: Add *STDEV*, *STDEV.S*, *STDEV.P*, *STDEVA*, *STDEVPA*, *VAR*, *VAR.S*, *VAR.P*, *VARA*, and *VARPA* functions.

3.12.19.2 Fix

- (financial): Correct requirements for *irr* function.
- (excel) #48: Correct reference pointing to different workbooks.
- (function) #67: Correct compilation of impure functions (e.g., *rand*, *now*, etc.).
- (look) #66: Correct *check* function did not return value.
- (test): Remove *temp* dir.
- (excel): Correct external link reading.
- (operator) #63: Correct operator parser when starts with spaces.
- (text) #61: Convert float as int when stringify if it is an integer.
- (math) #59: Convert string to number in math operations.
- (functions): Correct *_xfilter* operating range type.
- (parser) #61: Skip *n* in formula expression.
- (operator) #58: Correct operator parser for composed operators.
- (excel): Correct invalid range definition and missing sheet or files.
- (operand) #52: Correct range parser.
- (operand) #50: Correct sheet name parser with space.
- (tokens): Correct closure parenthesis parser.
- (excel): Skip function compilation for string cells.

- (tokens): Correct error parsing when sheet name is defined.

3.12.20 v1.0.0 (2020-03-12)

3.12.20.1 Feat

- (core): Add *CODE_OF_CONDUCT.md*.
- (function) #39: Transform *NotImplementedError* into *#NAME?*.
- (text) #39: Add *CONCAT* and *CONCATENATE* functions.
- (logic) #38: Add TRUE/FALSE functions.
- (excel) #42: Save missing nodes.
- (excel) #42: Update logic for *RangesAssembler*.
- (excel): Improve performance of *finish* method.
- (core): Update build script.
- (core): Add support for python 3.8 and drop python 3.5 and drop *appveyor*.
- (core): Improve memory performance.
- (refact): Update copyright.
- (operand): Add *fast_range2parts_v4* for named ranges.

3.12.20.2 Fix

- (math) #37: Match excel default rounding algorithm of round half up.
- (cell): Correct reference in *push* method.
- (readme): Correct doctest.
- (token): Correct separator parser.
- (excel) #35: Update logic to parse named ranges.
- (operand): Associate *excel_id==0* to current excel.
- (array): Ensure correct deepcopy of *Array* attributes.
- (operand) #39: Correct range parser for named ranges.
- (operand) #41: Correct named ranges parser.

3.12.21 v0.4.0 (2019-08-31)

3.12.21.1 Feat

- (doc): Add binder.
- (setup): Add env *ENABLE_SETUP_LONG_DESCRIPTION*.
- (core): Add useful constants.
- (excel): Add option to write all calculate books inside a folder.
- (stat) #21: Add *COUNTBLANK*, *LARGE*, *SMALL* functions.
- (date) #35: Add *NPV*, *XNPV*, *IRR*, *XIRR* functions.
- (stat) #21: Add *AVERAGEIF*, *COUNT*, *COUNTA*, *COUNTIF* functions.

- (math) #21: Add *SUMIF* function.
- (date) #21, #35, #36: Add *date* functions *DATE*, *DATEVALUE*, *DAY*, *MONTH*, *YEAR*, *TODAY*, *TIME*, *TIMEVALUE*, *SECOND*, *MINUTE*, *HOUR*, *NOW*, *YEARFRAC*.
- (info) #21: Add *NA* function.
- (date) #21, #35, #36: Add *date* functions *DATE*, *DATEVALUE*, *DAY*, *MONTH*, *YEAR*, *TODAY*, *TIME*, *TIMEVALUE*, *SECOND*, *MINUTE*, *HOUR*, *NOW*, *YEARFRAC*.
- (stat) #35: Add *MINA*, *AVERAGEA*, *MAXA* functions.

3.12.21.2 Fix

- (setup): Update tests requirements.
- (setup): Correct setup dependency (*beautifulsoup4*).
- (stat): Correct round indices.
- (setup) #34: Build universal wheels.
- (test): Correct import error.
- (date) #35: Correct behaviour of *LOOKUP* function when dealing with errors.
- (excel) #35: Improve cycle detection.
- (excel,date) #21, #35: Add custom Excel Reader to parse raw datetime.
- (excel) #35: Correct when definedName is relative *#REF!*.

3.12.22 v0.3.0 (2019-04-24)

3.12.22.1 Feat

- (logic) #27: Add *OR*, *XOR*, *AND*, *NOT* functions.
- (look) #27: Add *INDEX* function.
- (look) #24: Improve performances of *look* functions.
- (functions) #26: Add *SWITCH*.
- (functions) #30: Add *GCD* and *LCM*.
- (chore): Improve performances avoiding *combine_dicts*.
- (chore): Improve performances checking intersection.

3.12.22.2 Fix

- (tokens): Correct string nodes ids format adding “.
- (ranges): Correct behaviour union of ranges.
- (import): Enable PyCharm autocomplete.
- (import): Save imports.
- (test): Add repo path to system path.
- (parser): Parse empty args for functions.
- (functions) #30: Correct implementation of *GCD* and *LCM*.
- (ranges) #24: Enable full column and row reference.

- (excel): Correct bugs due to new *openpyxl*.

3.12.23 v0.2.0 (2018-12-11)

3.12.23.1 Feat

- (doc) #23: Enhance *ExcelModel* documentation.

3.12.23.2 Fix

- (core): Add python 3.7 and drop python 3.4.
- (excel): Make *ExcelModel* dillable and pickable.
- (builder): Avoid FormulaError exception during formulas compilation.
- (excel): Correct bug when compiling excel with circular references.

3.12.24 v0.1.4 (2018-10-19)

3.12.24.1 Fix

- (tokens) #20: Improve Number regex.

3.12.25 v0.1.3 (2018-10-09)

3.12.25.1 Feat

- (excel) #16: Solve circular references.
- (setup): Add donate url.

3.12.25.2 Fix

- (functions) #18: Enable *check_error* in *IF* function just for the first argument.
- (functions) #18: Disable *input_parser* in *IF* function to return any type of values.
- (rtd): Define *fpath* from *prj_dir* for rtd.
- (rtd): Add missing requirements *openpyxl* for rtd.
- (setup): Patch to use *sphinxcontrib.restbuilder* in setup *long_description*.

3.12.25.3 Other

- Update documentation.
- Replace *excel* with *Excel*.
- Create PULL_REQUEST_TEMPLATE.md.
- Update issue templates.
- Update copyright.
- (doc): Update author mail.

3.12.26 v0.1.2 (2018-09-12)

3.12.26.1 Feat

- (functions) #14: Add *ROW* and *COLUMN*.
- (cell): Pass cell reference when compiling cell + new function struct with dict to add inputs like CELL.

3.12.26.2 Fix

- (ranges): Replace system max size with excel max row and col.
- (tokens): Correct number regex.

3.12.27 v0.1.1 (2018-09-11)

3.12.27.1 Feat

- (contrib): Add contribution instructions.
- (setup): Add additional project_urls.
- (setup): Update *Development Status* to *4 - Beta*.

3.12.27.2 Fix

- (init) #15: Replace *FUNCTIONS* and *OPERATORS* objs with *get_functions*, *SUBMODULES*.
- (doc): Correct link docs_status.

3.12.28 v0.1.0 (2018-07-20)

3.12.28.1 Feat

- (readme) #6, #7: Add examples.
- (doc): Add changelog.
- (test): Add info of executed test of *test_excel_model*.
- (functions) #11: Add *HEX2OCT*, *HEX2BIN*, *HEX2DEC*, *OCT2HEX*, *OCT2BIN*, *OCT2DEC*, *BIN2HEX*, *BIN2OCT*, *BIN2DEC*, *DEC2HEX*, *DEC2OCT*, and *DEC2BIN* functions.
- (setup) #13: Add extras_require to setup file.

3.12.28.2 Fix

- (excel): Use DispatchPipe to compile a sub model of excel workbook.
- (range) #11: Correct range regex to avoid parsing of function like ranges (e.g., HEX2DEC).

3.12.29 v0.0.10 (2018-06-05)

3.12.29.1 Feat

- (look): Simplify *_get_type_id* function.

3.12.29.2 Fix

- (functions): Correct ImportError for FUNCTIONS.
- (operations): Correct behaviour of the basic operations.

3.12.30 v0.0.9 (2018-05-28)

3.12.30.1 Feat

- (excel): Improve performances pre-calculating the range format.
- (core): Improve performances using *DispatchPipe* instead *SubDispatchPipe* when compiling formulas.
- (function): Improve performances setting *errstate* outside vectorization.
- (core): Improve performances of range2parts function (overall 50% faster).

3.12.30.2 Fix

- (ranges): Minimize conversion str to int and vice versa.
- (functions) #10: Avoid returning shapeless array.

3.12.31 v0.0.8 (2018-05-23)

3.12.31.1 Feat

- (functions): Add *MATCH*, *LOOKUP*, *HLOOKUP*, *VLOOKUP* functions.
- (excel): Add method to compile *ExcelModel*.
- (travis): Run coveralls in python 3.6.
- (functions): Add *FIND*,`LEFT`,`LEN`,`LOWER`,`MID`,`REPLACE`,`RIGHT`,`TRIM`, and`UPPER` functions.
- (functions): Add *IRR* function.
- (formulas): Custom reshape to Array class.
- (functions): Add *ISO.CEILING*, *SQRTPI*, *TRUNC* functions.
- (functions): Add *ROUND*, *ROUNDDOWN*, *ROUNDUP*, *SEC*, *SECH*, *SIGN* functions.
- (functions): Add *DECIMAL*, *EVEN*, *MROUND*, *ODD*, *RAND*, *RANDBETWEEN* functions.
- (functions): Add *FACT* and *FACTDOUBLE* functions.
- (functions): Add *ARABIC* and *ROMAN* functions.
- (functions): Parametrize function *wrap_ufunc*.
- (functions): Split function *raise_errors* adding *get_error* function.
- (ranges): Add custom default and error value for defining ranges Arrays.
- (functions): Add *LOG10* function + fix *LOG*.
- (functions): Add *CSC* and *CSCH* functions.
- (functions): Add *COT* and *COTH* functions.
- (functions): Add *FLOOR*, *FLOOR.MATH*, and *FLOOR.PRECISE* functions.
- (test): Improve log message of test cell.

3.12.31.2 Fix

- (rtd): Update installation file for read the docs.
- (functions): Remove unused functions.
- (formulas): Avoid too broad exception.

- (functions.math): Drop scipy dependency for calculate factorial2.
- (functions.logic): Correct error behaviour of *if* and *iferror* functions + add BroadcastError.
- (functions.info): Correct behaviour of *iserr* function.
- (functions): Correct error behaviour of average function.
- (functions): Correct *iserror* and *iserr* returning a custom Array.
- (functions): Now *xceiling* function returns np.nan instead Error.errors['#NUM!'].
- (functions): Correct *is_number* function, now returns False when number is a bool.
- (test): Ensure same order of workbook comparisons.
- (functions): Correct behaviour of *min max* and *int* function.
- (ranges): Ensure to have a value with correct shape.
- (parser): Change order of parsing to avoid TRUE and FALSE parsed as ranges or errors as strings.
- (function):Remove unused kwargs n_out.
- (parser): Parse error string as formulas.
- (readme): Remove *downloads_count* because it is no longer available.

3.12.31.3 Other

- Refact: Update Copyright + minor pep.
- Excel returns 1-indexed string positions???
- Added common string functions.
- Merge pull request #9 from ecatkins/irr.
- Implemented IRR function using numpy.

3.12.32 v0.0.7 (2017-07-20)

3.12.32.1 Feat

- (appveyor): Add python 3.6.
- (functions) #4: Add *sumproduct* function.

3.12.32.2 Fix

- (install): Force update setuptools>=36.0.1.
- (functions): Correct *iserror iserr* functions.
- (ranges): Replace '#N/A' with '' as empty value when assemble values.
- (functions) #4: Remove check in ufunc when inputs have different size.
- (functions) #4: Correct *power*, *arctan2*, and *mod* error results.
- (functions) #4: Simplify ufunc code.
- (test) #4: Check that all results are in the output.
- (functions) #4: Correct *atan2* argument order.
- (range) #5: Avoid parsing function name as range when it is followed by *(*.

- (operator) #3: Replace *strip* with *replace*.
- (operator) #3: Correct valid operators like ^- or *+.

3.12.32.3 Other

- Made the ufunc wrapper work with multi input functions, e.g., power, mod, and atan2.
- Created a workbook comparison method in TestExcelModel.
- Added MIN and MAX to the test.xlsx.
- Cleaned up the ufunc wrapper and added min and max to the functions list.
- Relaxed equality in TestExcelModel and made some small fixes to functions.py.
- Added a wrapper for numpy ufuncs, mapped some Excel functions to ufuncs and provided tests.

3.12.33 v0.0.6 (2017-05-31)

3.12.33.1 Fix

- (plot): Update schedula to 0.1.12.
- (range): Sheet name without commas has this [^Wd][w.] format.

3.12.34 v0.0.5 (2017-05-04)

3.12.34.1 Fix

- (doc): Update schedula to 0.1.11.

3.12.35 v0.0.4 (2017-02-10)

3.12.35.1 Fix

- (regex): Remove deprecation warnings.

3.12.36 v0.0.3 (2017-02-09)

3.12.36.1 Fix

- (appveyor): Setup of lxml.
- (excel): Remove deprecation warning openpyxl.
- (requirements): Update schedula requirement 0.1.9.

3.12.37 v0.0.2 (2017-02-08)

3.12.37.1 Fix

- (setup): setup fails due to long description.
- (excel): Remove deprecation warning *remove_sheet* –> *remove*.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

f

formulas, 20
formulas.builder, 22
formulas.cell, 210
formulas.errors, 23
formulas.excel, 214
formulas.excel.cycle, 214
formulas.excel.xlreader, 214
formulas.functions, 56
formulas.functions.comp, 56
formulas.functions.date, 56
formulas.functions.eng, 59
formulas.functions.financial, 61
formulas.functions.google, 67
formulas.functions.info, 67
formulas.functions.logic, 146
formulas.functions.look, 147
formulas.functions.math, 151
formulas.functions.operators, 154
formulas.functions.stat, 154
formulas.functions.text, 163
formulas.parser, 20
formulas.ranges, 208
formulas.tokens, 24
formulas.tokens.function, 24
formulas.tokens.operand, 29
formulas.tokens.operator, 47
formulas.tokens.parenthesis, 53

INDEX

Symbols

__init__() (*Array method*), 25, 197
__init__() (*AstBuilder method*), 22
__init__() (*Cell method*), 211
__init__() (*CellWrapper method*), 212
__init__() (*Empty method*), 31
__init__() (*Error method*), 32
__init__() (*ExcelModel method*), 219
__init__() (*FalseArray method*), 96
__init__() (*Function method*), 26
__init__() (*Intersect method*), 48
__init__() (*InvRangesAssembler method*), 212
__init__() (*Lambda method*), 28
__init__() (*LambdaFunction method*), 28
__init__() (*LetaFunction method*), 29
__init__() (*Number method*), 34
__init__() (*Operand method*), 35
__init__() (*Operator method*), 49
__init__() (*OperatorToken method*), 51
__init__() (*Parenthesis method*), 54
__init__() (*Parser method*), 21
__init__() (*Range method*), 36
__init__() (*Ranges method*), 209
__init__() (*RangesAssembler method*), 213
__init__() (*Ref method*), 214
__init__() (*Separator method*), 52
__init__() (*String method*), 37
__init__() (*Token method*), 55
__init__() (*TrueArray method*), 134
__init__() (*XlCircular method*), 229
__init__() (*XlError method*), 47
__init__() (*XlReader method*), 216

A

args2list() (*in module formulas.functions*), 167
args2vals() (*in module formulas.functions*), 167
args_parser_disc() (*in module formulas.functions.financial*), 62
args_parser_fvschedule() (*in module formulas.functions.financial*), 62
args_parser_hlookup() (*in module formulas.functions.look*), 148
args_parser_intrate() (*in module formulas.functions.financial*), 62
args_parser_lookup_array() (*in module formulas.functions.look*), 148
args_parser_match_array() (*in module formulas.functions.look*), 148
args_parser_received() (*in module formulas.functions.financial*), 62
args_parser_typed_array() (*in module formulas.functions.look*), 148
args_parser_xlookup_array() (*in module formulas.functions.look*), 148
args_parser_xmatch_array() (*in module formulas.functions.look*), 148
args_xnetworkdays_intl() (*in module formulas.functions.date*), 57
args_xworkday_intl() (*in module formulas.functions.date*), 57
Array (*class in formulas.functions*), 169
Array (*class in formulas.tokens.function*), 24
AstBuilder (*class in formulas.builder*), 22

C

Cell (*class in formulas.cell*), 210
CellWrapper (*class in formulas.cell*), 211
clean_values() (*in module formulas.functions*), 167
compile_class (*AstBuilder attribute*), 22
compile_class (*ExcelModel attribute*), 220
convert2float() (*in module formulas.functions*), 167
convert_nan() (*in module formulas.functions*), 168
convert_noshp() (*in module formulas.functions*), 168

D

day_count() (*in module formulas.functions.date*), 57

E

Empty (*class in formulas.tokens.operand*), 31
Error (*class in formulas.tokens.operand*), 32
escape_char() (*in module formulas.excel*), 216
ExcelModel (*class in formulas.excel*), 217

F

FalseArray (*class in formulas.functions.info*), 69
fast_range2parts() (*in module formulas.tokens.operand*), 30
fast_range2parts_v1() (*in module formulas.tokens.operand*), 30
fast_range2parts_v2() (*in module formulas.tokens.operand*), 30
fast_range2parts_v3() (*in module formulas.tokens.operand*), 30
fast_range2parts_v4() (*in module formulas.tokens.operand*), 30
fast_range2parts_v5() (*in module formulas.tokens.operand*), 30
flatten() (*in module formulas.functions*), 168
format_output() (*in module formulas.cell*), 210
formulas
    module, 20
formulas.builder
    module, 22
formulas.cell
    module, 210
formulas.errors
    module, 23
formulas.excel
    module, 214
formulas.excel.cycle
    module, 214
formulas.excel.xlreader
    module, 214
formulas.functions
    module, 56
formulas.functions.comp
    module, 56
formulas.functions.date
    module, 56
formulas.functions.eng
    module, 59
formulas.functions.financial
    module, 61
formulas.functions.google
    module, 67
formulas.functions.info
    module, 67
formulas.functions.logic
    module, 146
formulas.functions.look
    module, 147
formulas.functions.math
    module, 151
formulas.functions.operators
    module, 154
formulas.functions.stat
    module, 154
formulas.functions.text
    module, 163
formulas.parser
    module, 20
formulas.ranges
    module, 208
formulas.tokens
    module, 24
formulas.tokens.function
    module, 24
formulas.tokens.operand
    module, 29
formulas.tokens.operator
    module, 47
formulas.tokens.parenthesis
    module, 53
Function (*class in formulas.tokens.function*), 26

G

get_error() (*in module formulas.functions*), 168
get_functions() (*in module formulas.functions*), 168
get_shape() (*in module formulas.functions*), 168

H

hex2dec2bin2oct() (*in module formulas.functions.eng*), 59

I

input_parser_xlookup() (*in module formulas.functions.look*), 148
input_parser_xmatch() (*in module formulas.functions.look*), 148
Intersect (*class in formulas.tokens.operator*), 47
InvRangesAssembler (*class in formulas.cell*), 212
is_complex() (*in module formulas.functions*), 168
is_not_empty() (*in module formulas.functions*), 168
is_number() (*in module formulas.functions*), 168
iserr() (*in module formulas.functions.info*), 68
iserror() (*in module formulas.functions.info*), 68
isna() (*in module formulas.functions.info*), 68
isref() (*in module formulas.functions.info*), 68

L

Lambda (*class in formulas.tokens.function*), 27
LambdaFunction (*class in formulas.tokens.function*), 28
LetaFunction (*class in formulas.tokens.function*), 29
load_workbook() (*in module formulas.excel.xlreader*), 215
logic_input_parser() (*in module formulas.functions.operators*), 154

M

map_multiindex_take_last_if_tuple() (*in module formulas.functions.look*), 148

mirr_args_parser() (*in module formulas.functions.financial*), 62
module
formulas, 20
formulas.builder, 22
formulas.cell, 210
formulas.errors, 23
formulas.excel, 214
formulas.excel.cycle, 214
formulas.excel.xlreader, 214
formulas.functions, 56
formulas.functions.comp, 56
formulas.functions.date, 56
formulas.functions.eng, 59
formulas.functions.financial, 61
formulas.functions.google, 67
formulas.functions.info, 67
formulas.functions.logic, 146
formulas.functions.look, 147
formulas.functions.math, 151
formulas.functions.operators, 154
formulas.functions.stat, 154
formulas.functions.text, 163
formulas.parser, 20
formulas.ranges, 208
formulas.tokens, 24
formulas.tokens.function, 24
formulas.tokens.operand, 29
formulas.tokens.operator, 47
formulas.tokens.parenthesis, 53

N

not_implemented() (*in module formulas.functions*), 168
Number (*class in formulas.tokens.operand*), 33

O

Operand (*class in formulas.tokens.operand*), 34
Operator (*class in formulas.tokens.operator*), 48
OperatorToken (*class in formulas.tokens.operator*), 50

P

Parenthesis (*class in formulas.tokens.parenthesis*), 53
parse_basis() (*in module formulas.functions.financial*), 63
parse_date() (*in module formulas.functions.financial*), 63
parse_ranges() (*in module formulas.functions*), 168
Parser (*class in formulas.parser*), 21

R

raise_errors() (*in module formulas.functions*), 168
Range (*class in formulas.tokens.operand*), 35
range2parts() (*in module formulas.tokens.operand*), 30
Ranges (*class in formulas.ranges*), 208
RangesAssembler (*class in formulas.cell*), 212
read_string_table() (*in module formulas.excel.xlreader*), 215
Ref (*class in formulas.cell*), 213
replace_empty() (*in module formulas.functions*), 168
replace_hex() (*in module formulas.excel.xlreader*), 215
reshape() (*Array method*), 208
return_2d_func() (*in module formulas.functions*), 169
return_func() (*in module formulas.functions.math*), 152
return_trimrange_func() (*in module formulas.functions.look*), 149
return_unique_func() (*in module formulas.functions.look*), 149
round_up() (*in module formulas.functions.math*), 152
run_function() (*in module formulas.tokens.function*), 24

S

Separator (*class in formulas.tokens.operator*), 51
simple_cycles() (*in module formulas.excel.cycle*), 214
solve_cycle() (*in module formulas.functions.logic*), 146
str2complex() (*in module formulas.functions*), 169
String (*class in formulas.tokens.operand*), 36
sumx2my2() (*in module formulas.functions.math*), 152

T

text2num() (*in module formulas.functions*), 169
to_number() (*in module formulas.functions*), 169
to_python() (*in module formulas.functions.look*), 149
Token (*class in formulas.tokens*), 54
total_depr() (*in module formulas.functions.financial*), 63
TrueArray (*class in formulas.functions.info*), 107

W

wrap_cell_func() (*in module formulas.cell*), 210
wrap_func() (*in module formulas.functions*), 169
wrap_impure_func() (*in module formulas.functions*), 169
wrap_ranges_func() (*in module formulas.functions*), 169
wrap_ufunc() (*in module formulas.functions*), 169

X

xaccrint() (*in module formulas.functions.financial*), 63
xaccrintm() (*in module formulas.functions.financial*), 63

xaddress() (*in module formulas.functions.look*), 149
xamordegrc() (*in module formulas.functions.financial*), 63
xamorlinc() (*in module formulas.functions.financial*), 63
xand() (*in module formulas.functions.logic*), 146
xarabic() (*in module formulas.functions.math*), 152
xarctan2() (*in module formulas.functions.math*), 152
xareas() (*in module formulas.functions.look*), 149
xarraytotext() (*in module formulas.functions.text*), 164
xasc() (*in module formulas.functions.text*), 164
xbahttext() (*in module formulas.functions.text*), 164
xbesseli() (*in module formulas.functions.eng*), 59
xbesselj() (*in module formulas.functions.eng*), 59
xbesselk() (*in module formulas.functions.eng*), 59
xbessely() (*in module formulas.functions.eng*), 60
xbetadist() (*in module formulas.functions.stat*), 156
xbetainv() (*in module formulas.functions.stat*), 156
xbinomdist() (*in module formulas.functions.stat*), 156
xbinomdistrange() (*in module formulas.functions.stat*), 157
xbinominv() (*in module formulas.functions.stat*), 157
xbitand() (*in module formulas.functions.eng*), 60
xbitlshift() (*in module formulas.functions.eng*), 60
xbitor() (*in module formulas.functions.eng*), 60
xbitrshift() (*in module formulas.functions.eng*), 60
xbitxor() (*in module formulas.functions.eng*), 60
xbycol() (*in module formulas.functions.logic*), 146
xceiling() (*in module formulas.functions.math*), 152
xceiling_math() (*in module formulas.functions.math*), 152
xchar() (*in module formulas.functions.text*), 164
xchisqdist() (*in module formulas.functions.stat*), 157
xchisqdistrt() (*in module formulas.functions.stat*), 157
xchisqinv() (*in module formulas.functions.stat*), 157
xchisqinvrt() (*in module formulas.functions.stat*), 157
xchisqtest() (*in module formulas.functions.stat*), 157
xchoosecols() (*in module formulas.functions.look*), 149
xclean() (*in module formulas.functions.text*), 164
xcode() (*in module formulas.functions.text*), 165
xcolumn() (*in module formulas.functions.look*), 149
xcolumns() (*in module formulas.functions.look*), 149
xcomplex() (*in module formulas.functions.eng*), 60
xconcat() (*in module formulas.functions.text*), 165
xconfidence_norm() (*in module formulas.functions.stat*), 157
xconfidence_t() (*in module formulas.functions.stat*), 157
xconvert() (*in module formulas.functions.eng*), 60
xcorrel() (*in module formulas.functions.stat*), 157
xcot() (*in module formulas.functions.math*), 152
xcoupdaybs() (*in module formulas.functions.financial*), 63
xcoupdays() (*in module formulas.functions.financial*), 63
xcoupdaysnc() (*in module formulas.functions.financial*), 63
xcoupncd() (*in module formulas.functions.financial*), 64
xcoupnum() (*in module formulas.functions.financial*), 64
xcouppcd() (*in module formulas.functions.financial*), 64
xcovariance_p() (*in module formulas.functions.stat*), 157
xcovariance_s() (*in module formulas.functions.stat*), 157
xcumipmt() (*in module formulas.functions.financial*), 64
xdate() (*in module formulas.functions.date*), 57
xdate2date() (*in module formulas.functions.financial*), 64
xdatedif() (*in module formulas.functions.date*), 57
xdatevalue() (*in module formulas.functions.date*), 57
xday() (*in module formulas.functions.date*), 57
xdays() (*in module formulas.functions.date*), 57
xdays360() (*in module formulas.functions.date*), 57
xdb() (*in module formulas.functions.financial*), 64
xddb() (*in module formulas.functions.financial*), 64
xdecimal() (*in module formulas.functions.math*), 152
xdelta() (*in module formulas.functions.eng*), 60
xdisc() (*in module formulas.functions.financial*), 64
xdollarde() (*in module formulas.functions.financial*), 64
xdollarfr() (*in module formulas.functions.financial*), 64
xdrop() (*in module formulas.functions.look*), 149
xdummy() (*in module formulas.functions.google*), 67
xduration() (*in module formulas.functions.financial*), 64
xedate() (*in module formulas.functions.date*), 58
xeffect() (*in module formulas.functions.financial*), 64
xeomonth() (*in module formulas.functions.date*), 58
xerf() (*in module formulas.functions.eng*), 60
xerf_precise() (*in module formulas.functions.eng*), 60
xerrortype() (*in module formulas.functions.info*), 68
xeven() (*in module formulas.functions.math*), 152
xexact() (*in module formulas.functions.text*), 165
xexpand() (*in module formulas.functions.look*), 149
xexpon_dist() (*in module formulas.functions.stat*), 158
xfact() (*in module formulas.functions.math*), 153
xfactdouble() (*in module formulas.functions.math*), 153
xfdist() (*in module formulas.functions.stat*), 158
xfdistrt() (*in module formulas.functions.stat*), 158
xfilter() (*in module formulas.functions*), 169
xfilter() (*in module formulas.functions.look*), 149
xfilters() (*in module formulas.functions*), 169
xfind() (*in module formulas.functions.text*), 165

xfinv() (*in module formulas.functions.stat*), 158
xfinvrt() (*in module formulas.functions.stat*), 158
xfisher() (*in module formulas.functions.stat*), 158
xfisherinv() (*in module formulas.functions.stat*), 158
xfixed() (*in module formulas.functions.text*), 165
xforecast() (*in module formulas.functions.stat*), 158
xforecast_ets() (*in module formulas.functions.stat*), 158
xforecast_ets_confint() (*in module formulas.functions.stat*), 158
xforecast_ets_seasonality() (*in module formulas.functions.stat*), 159
xforecast_ets_stat() (*in module formulas.functions.stat*), 159
xfrequency() (*in module formulas.functions.stat*), 159
xftest() (*in module formulas.functions.stat*), 159
xfunc() (*in module formulas.functions.stat*), 159
xfvschedule() (*in module formulas.functions.financial*), 65
xgamma() (*in module formulas.functions.stat*), 159
xgamma_dist() (*in module formulas.functions.stat*), 159
xgamma_inv() (*in module formulas.functions.stat*), 159
xgammaln() (*in module formulas.functions.stat*), 159
xgauss() (*in module formulas.functions.stat*), 159
xgcd() (*in module formulas.functions.math*), 153
xgestep() (*in module formulas.functions.eng*), 61
xgroupby() (*in module formulas.functions.look*), 149
xgrowth() (*in module formulas.functions.stat*), 160
xhstack() (*in module formulas.functions.look*), 150
xhypergeom_dist() (*in module formulas.functions.stat*), 160
xif() (*in module formulas.functions.logic*), 146
xiferror() (*in module formulas.functions.logic*), 146
xifna() (*in module formulas.functions.logic*), 146
xifs() (*in module formulas.functions.logic*), 146
ximargument() (*in module formulas.functions.eng*), 61
ximpower() (*in module formulas.functions.eng*), 61
xindex() (*in module formulas.functions.look*), 150
xintercept() (*in module formulas.functions.stat*), 160
xintrate() (*in module formulas.functions.financial*), 65
xirr() (*in module formulas.functions.financial*), 65
xiseven_odd() (*in module formulas.functions.info*), 68
xisformula() (*in module formulas.functions.info*), 68
xisoweeknum() (*in module formulas.functions.date*), 58
xispmt() (*in module formulas.functions.financial*), 65
XlCircular (*class in formulas.excel*), 220
xlcm() (*in module formulas.functions.math*), 153
xleft() (*in module formulas.functions.text*), 165
XlError (*class in formulas.tokens.operand*), 38
xlinest() (*in module formulas.functions.stat*), 160
xlogest() (*in module formulas.functions.stat*), 160
xlognormdist() (*in module formulas.functions.stat*), 160
xlognorminv() (*in module formulas.functions.stat*), 160
xlookup() (*in module formulas.functions.look*), 150
XlReader (*class in formulas.excel.xlreader*), 215
xmakearray() (*in module formulas.functions.logic*), 146
xmap() (*in module formulas.functions.logic*), 147
xmatch() (*in module formulas.functions.look*), 150
xmdeterm() (*in module formulas.functions.math*), 153
xmid() (*in module formulas.functions.text*), 165
xmirr() (*in module formulas.functions.financial*), 65
xmmult() (*in module formulas.functions.math*), 153
xmod() (*in module formulas.functions.math*), 153
xmround() (*in module formulas.functions.math*), 153
xmultinominal() (*in module formulas.functions.math*), 153
xmunit() (*in module formulas.functions.math*), 153
xn() (*in module formulas.functions.info*), 68
xna() (*in module formulas.functions.info*), 68
xnegbinomdist() (*in module formulas.functions.stat*), 160
xnetworkdays_intl() (*in module formulas.functions.date*), 58
xnominal() (*in module formulas.functions.financial*), 65
xnormdist() (*in module formulas.functions.stat*), 160
xnorminv() (*in module formulas.functions.stat*), 160
xnow() (*in module formulas.functions.date*), 58
xnper() (*in module formulas.functions.financial*), 65
xnpv() (*in module formulas.functions.financial*), 65
xnumbervalue() (*in module formulas.functions.text*), 165
xodd() (*in module formulas.functions.math*), 153
xoddfprice() (*in module formulas.functions.financial*), 65
xoddfyield() (*in module formulas.functions.financial*), 65
xoddlprice() (*in module formulas.functions.financial*), 65
xoddlyield() (*in module formulas.functions.financial*), 65
xpduration() (*in module formulas.functions.financial*), 66
xpearson() (*in module formulas.functions.stat*), 161
xpercentile() (*in module formulas.functions.stat*), 161
xpercentof() (*in module formulas.functions.math*), 153
xpercentrank() (*in module formulas.functions.stat*), 161
xpermut() (*in module formulas.functions.stat*), 161
xpermutationa() (*in module formulas.functions.stat*), 161
xphi() (*in module formulas.functions.stat*), 161
xpoisson_dist() (*in module formulas.functions.stat*), 161
xpower() (*in module formulas.functions.math*), 154
xppmt() (*in module formulas.functions.financial*), 66
xprice() (*in module formulas.functions.financial*), 66

xpricedisc() (*in module formulas.functions.financial*), 66
xpricemat() (*in module formulas.functions.financial*), 66
xprob() (*in module formulas.functions.stat*), 161
xquartile() (*in module formulas.functions.stat*), 162
xrandbetween() (*in module formulas.functions.math*), 154
xrank() (*in module formulas.functions.stat*), 162
xrate() (*in module formulas.functions.financial*), 66
xreceived() (*in module formulas.functions.financial*), 66
xregexextract() (*in module formulas.functions.text*), 165
xregexreplace() (*in module formulas.functions.text*), 165
xregextest() (*in module formulas.functions.text*), 165
xreplace() (*in module formulas.functions.text*), 165
xrept() (*in module formulas.functions.text*), 166
xright() (*in module formulas.functions.text*), 166
xroman() (*in module formulas.functions.math*), 154
xround() (*in module formulas.functions.math*), 154
xrow() (*in module formulas.functions.look*), 150
xrri() (*in module formulas.functions.financial*), 66
xrsq() (*in module formulas.functions.stat*), 162
xscan() (*in module formulas.functions.logic*), 147
xsearch() (*in module formulas.functions.text*), 166
xsecond() (*in module formulas.functions.date*), 58
xsingle() (*in module formulas.functions.look*), 150
xsln() (*in module formulas.functions.financial*), 66
xslope() (*in module formulas.functions.stat*), 162
xsort() (*in module formulas.functions.look*), 150
xsort() (*in module formulas.functions.stat*), 162
xsortby() (*in module formulas.functions.look*), 150
xsrqtpi() (*in module formulas.functions.math*), 154
xstandardize() (*in module formulas.functions.stat*), 162
xstdev() (*in module formulas.functions.stat*), 162
xsteyx() (*in module formulas.functions.stat*), 162
xsubstitute() (*in module formulas.functions.text*), 166
xsum() (*in module formulas.functions.math*), 154
xsumproduct() (*in module formulas.functions.math*), 154
xswitch() (*in module formulas.functions.logic*), 147
xsyd() (*in module formulas.functions.financial*), 66
xt() (*in module formulas.functions.text*), 166
xt_dist() (*in module formulas.functions.stat*), 162
xt_dist2t() (*in module formulas.functions.stat*), 162
xt_distrt() (*in module formulas.functions.stat*), 162
xt_inv() (*in module formulas.functions.stat*), 162
xt_inv2t() (*in module formulas.functions.stat*), 163
xt_test() (*in module formulas.functions.stat*), 163
xtake() (*in module formulas.functions.look*), 150
xtbilleq() (*in module formulas.functions.financial*), 66
xtbillprice() (*in module formulas.functions.financial*), 66
xtbillyield() (*in module formulas.functions.financial*), 67
xtdist() (*in module formulas.functions.comp*), 56
xtext() (*in module formulas.functions.text*), 166
xtextafterbefore() (*in module formulas.functions.text*), 166
xtextjoin() (*in module formulas.functions.text*), 166
xtextsplit() (*in module formulas.functions.text*), 166
xtime() (*in module formulas.functions.date*), 58
xtimevalue() (*in module formulas.functions.date*), 58
xtocol() (*in module formulas.functions.look*), 150
xtoday() (*in module formulas.functions.date*), 58
xtranspose() (*in module formulas.functions.look*), 150
xtrend() (*in module formulas.functions.stat*), 163
xtrimmean() (*in module formulas.functions.stat*), 163
xtrimrange() (*in module formulas.functions.look*), 150
xtrunc() (*in module formulas.functions.math*), 154
xtype() (*in module formulas.functions.info*), 68
xunichar() (*in module formulas.functions.text*), 166
xunicode() (*in module formulas.functions.text*), 166
xunique() (*in module formulas.functions.look*), 151
xvalue() (*in module formulas.functions.text*), 166
xvaluetotext() (*in module formulas.functions.text*), 167
xvdb() (*in module formulas.functions.financial*), 67
xvstack() (*in module formulas.functions.look*), 151
xweekday() (*in module formulas.functions.date*), 58
xweeknum() (*in module formulas.functions.date*), 58
xweibulldist() (*in module formulas.functions.stat*), 163
xworkday_intl() (*in module formulas.functions.date*), 58
xwrapcols() (*in module formulas.functions.look*), 151
xxirr() (*in module formulas.functions.financial*), 67
xxlookup() (*in module formulas.functions.look*), 151
xxmatch() (*in module formulas.functions.look*), 151
xxnpv() (*in module formulas.functions.financial*), 67
xyearfrac() (*in module formulas.functions.date*), 59
xyield() (*in module formulas.functions.financial*), 67
xyielddisc() (*in module formulas.functions.financial*), 67
xyieldmat() (*in module formulas.functions.financial*), 67
xz_test() (*in module formulas.functions.stat*), 163

Y

year_days() (*in module formulas.functions.date*), 59