formulas Documentation

Release 0.0.10

Vincenzo Arcidiacono

Jun 05, 2018

Table of Contents

release 0.0.10

date 2018-06-05 13:00:00

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

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

docs http://formulas.readthedocs.io/

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

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

keywords excel, formulas, interpreter, compiler, dispatch

developers

- Vincenzo Arcidiacono <vincenzo.arcidiacono@ext.jrc.ec.europa.eu>

license EUPL 1.1+

CHAPTER 1

What is formulas?

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

CHAPTER 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
```

What is formulas?

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

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
```

Next moves

Things yet to do implement the missing excel formulas.

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.
formulas	It contains a comprehensive list of all modules and classes within formulas.
ranges	It provides Ranges class.
cell	It provides Cell class.
excel	It provides excel model class.

parser

It provides formula parser class.

Classes

Parser

Parser

class Parser

Methods

ast
is_formula

ast

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

is_formula

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

__init__()
Initialize self. See help(type(self)) for accurate signature.

Attributes

filters
formula_check

filters

Parser.**filters** = [<class 'formulas.tokens.operand.Error'>, <class 'formulas.tokens.operand.String'>, <class 'formul

formula_check

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

builder

It provides AstBuilder class.

Classes

AstBuilder

AstBuilder

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

Methods

__init__	
append	
appendleft	Add an element to the left side of the deque.
clear	Remove all elements from the deque.
compile	
copy	Return a shallow copy of a deque.
count	
extend	Extend the right side of the deque with elements from the iterable
extendleft	Extend the left side of the deque with elements from the iterable
finish	
get_node_id	
index	Raises ValueError if the value is not present.
insert	D.insert(index, object) – insert object before index
pop	Remove and return the rightmost element.
popleft	Remove and return the leftmost element.
remove	D.remove(value) – remove first occurrence of value.
reverse	D.reverse() – reverse *IN PLACE*
rotate	Rotate the deque n steps to the right (default n=1).

__init__

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

append

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

appendleft

AstBuilder.**appendleft**()
Add an element to the left side of the deque.

clear

AstBuilder.**clear**()
Remove all elements from the deque.

compile

AstBuilder.**compile**(*references=None*)

copy

AstBuilder.**copy**()
Return a shallow copy of a deque.

count

AstBuilder.**count**(*value*) → integer – return number of occurrences of value

extend

AstBuilder.**extend**()
Extend the right side of the deque with elements from the iterable

extendleft

AstBuilder.**extendleft**()
Extend the left side of the deque with elements from the iterable

finish

AstBuilder.**finish**()

get_node_id

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

index

AstBuilder.**index**(*value*[, *start*[, *stop*]]) → integer – return first index of value.
Raises ValueError if the value is not present.

insert

AstBuilder.**insert**()
D.insert(index, object) – insert object before index

pop

AstBuilder.**pop**()
Remove and return the rightmost element.

popleft

AstBuilder.**popleft**()
Remove and return the leftmost element.

remove

AstBuilder.**remove**()
D.remove(value) – remove first occurrence of value.

reverse

AstBuilder.**reverse**()
D.reverse() – reverse *IN PLACE*

rotate

AstBuilder.**rotate**()
Rotate the deque n steps to the right (default n=1). If n is negative, rotates left.

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

Attributes

maxlen	maximum size of a deque or None if unbounded

maxlen

AstBuilder.**maxlen**
maximum size of a deque or None if unbounded

errors

Defines the formulas exception.

Exceptions

BaseError
BroadcastError
FormulaError
FoundError
FunctionError
ParenthesesError
RangeValueError
TokenError

BaseError

exception **BaseError**(**args*)

BroadcastError

exception **BroadcastError**(**args*)

FormulaError

exception **FormulaError**(**args*)

FoundError

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

FunctionError

exception **FunctionError**(**args*)

ParenthesesError

exception **ParenthesesError**(**args*)

RangeValueError

exception **RangeValueError**(**args*)

TokenError

exception **TokenError**(**args*)

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.

function

It provides Function classes.

Classes

Array
Function

Array

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

Methods

__init__
ast
compile
match
process
set_expr
update_input_tokens

__init__

`Array.`**`__init__`**(*s*, *context=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*)

set_expr

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

update_input_tokens

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

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

Attributes

name
node_id

name

Array.**name**

node_id

Array.**node_id**

Function

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

Methods

__init__
ast
compile
match
process
set_expr
update_input_tokens

__init__

Function.**__init__**(*s*, *context=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*)

set_expr

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

update_input_tokens

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

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

Attributes

name
node_id

name

Function.**name**

node_id

Function.**node_id**

operand

It provides Operand classes.

Functions

fast_range2parts
fast_range2parts_v1
fast_range2parts_v2
fast_range2parts_v3
range2parts

fast_range2parts

fast_range2parts(***kw*)

fast_range2parts_v1

fast_range2parts_v1(*r1*, *c1*, *excel*, *sheet=''*)

fast_range2parts_v2

fast_range2parts_v2(*r1*, *c1*, *r2*, *c2*, *excel*, *sheet=''*)

fast_range2parts_v3

fast_range2parts_v3(*r1*, *n1*, *r2*, *n2*, *excel*, *sheet=''*)

range2parts

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

Classes

Error
Number
Operand
Range
String
XlError

Error

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

Methods

__init__
ast
compile
match
process
set_expr
update_input_tokens

__init__

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

ast

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

compile

Error.**compile**()

match

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

process

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

set_expr

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

update_input_tokens

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

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

Attributes

errors

Continued on next page

Table 2.18 – continued from previous page

k
name
node_id

errors

Error.**errors** = {'#VALUE!': #VALUE!, '#NAME?': #NAME?, '#N/A': #N/A, '#REF!': #REF!, '#DIV/0!': #DIV/0!, '#

k

Error.**k** = '#N/A'

name

Error.**name**

node_id

Error.**node_id**

Number

class **Number**(*s*, *context=None*)

Methods

__init__
ast
compile
match
process
set_expr
update_input_tokens

__init__

Number.**__init__**(*s*, *context=None*)

ast

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

compile

Number.**compile**()

match

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

process

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

set_expr

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

update_input_tokens

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

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

Attributes

name
node_id

name

Number.**name**

node_id

Number.**node_id**

Operand

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

Methods

__init__
ast

Continued on next page

Table 2.21 – continued from previous page

match
process
set_expr
update_input_tokens

__init__

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

ast

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

match

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

process

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

set_expr

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

update_input_tokens

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

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

Attributes

name
node_id

name

Operand.**name**

node_id

Operand.**node_id**

Range

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

Methods

__init__
ast
compile
match
process
set_expr
update_input_tokens

__init__

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

ast

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

compile

Range.**compile**()

match

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

process

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

set_expr

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

update_input_tokens

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

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

Attributes

name
node_id

name

Range.**name**

node_id

Range.**node_id**

String

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

Methods

__init__
ast
compile
match
process
set_expr
update_input_tokens

__init__

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

ast

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

compile

String.**compile**()

match

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

process

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

set_expr

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

update_input_tokens

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

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

Attributes

name
node_id

name

String.**name**

node_id

String.**node_id**

XlError

class XlError

Methods

capitalize	Return a capitalized version of S, i.e.
casefold	Return a version of S suitable for caseless comparisons.
center	Return S centered in a string of length width.
count	Return the number of non-overlapping occurrences of substring sub in string S[start:end].
encode	Encode S using the codec registered for encoding.
endswith	Return True if S ends with the specified suffix, False otherwise.
expandtabs	Return a copy of S 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].

Continued on next page

Table 2.27 – continued from previous page

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	Like S.find() but raise ValueError when the substring is not found.
isalnum	Return True if all characters in S are alphanumeric and there is at least one character in S, False otherwise.
isalpha	Return True if all characters in S are alphabetic and there is at least one character in S, False otherwise.
isdecimal	Return True if there are only decimal characters in S, False otherwise.
isdigit	Return True if all characters in S are digits and there is at least one character in S, False otherwise.
isidentifier	Return True if S is a valid identifier according to the language definition.
islower	Return True if all cased characters in S are lowercase and there is at least one cased character in S, False otherwise.
isnumeric	Return True if there are only numeric characters in S, False otherwise.
isprintable	Return True if all characters in S are considered printable in repr() or S is empty, False otherwise.
isspace	Return True if all characters in S are whitespace and there is at least one character in S, False otherwise.
istitle	Return True if S is a titlecased string and there is at least one character in S, i.e.
isupper	Return True if all cased characters in S are uppercase and there is at least one cased character in S, False otherwise.
join	Return a string which is the concatenation of the strings in the iterable.
ljust	Return S left-justified in a Unicode string of length width.
lower	Return a copy of the string S converted to lowercase.
lstrip	Return a copy of the string S with leading whitespace removed.
maketrans	Return a translation table usable for str.translate().
partition	Search for the separator sep in S, and return the part before it, the separator itself, and the part after it.
replace	Return a copy of S 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	Like S.rfind() but raise ValueError when the substring is not found.
rjust	Return S right-justified in a string of length width.
rpartition	Search for the separator sep in S, starting at the end of S, and return the part before it, the separator itself, and the part after it.

Continued on next page

Table 2.27 – continued from previous page

rsplit	Return a list of the words in S, using sep as the delimiter string, starting at the end of the string and working to the front.
rstrip	Return a copy of the string S with trailing whitespace removed.
split	Return a list of the words in S, using sep as the delimiter string.
splitlines	Return a list of the lines in S, breaking at line boundaries.
startswith	Return True if S starts with the specified prefix, False otherwise.
strip	Return a copy of the string S with leading and trailing whitespace removed.
swapcase	Return a copy of S with uppercase characters converted to lowercase and vice versa.
title	Return a titlecased version of S, i.e.
translate	Return a copy of the string S in which each character has been mapped through the given translation table.
upper	Return a copy of S converted to uppercase.
zfill	Pad a numeric string S with zeros on the left, to fill a field of the specified width.

capitalize

XlError.**capitalize**() → str
Return a capitalized version of S, i.e. make the first character have upper case and the rest lower case.

casefold

XlError.**casefold**() → str
Return a version of S suitable for caseless comparisons.

center

XlError.**center**(*width*[, *fillchar*]) → str
Return S centered in a 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'*) → bytes
Encode S using the codec registered for encoding. Default encoding is 'utf-8'. errors may be given to

set a different error handling scheme. 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*) → str

Return a copy of S 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

Like S.find() but raise ValueError when the substring is not found.

isalnum

XlError.**isalnum**() → bool

Return True if all characters in S are alphanumeric and there is at least one character in S, False otherwise.

isalpha

XlError.**isalpha**() → bool
Return True if all characters in S are alphabetic and there is at least one character in S, False otherwise.

isdecimal

XlError.**isdecimal**() → bool
Return True if there are only decimal characters in S, False otherwise.

isdigit

XlError.**isdigit**() → bool
Return True if all characters in S are digits and there is at least one character in S, False otherwise.

isidentifier

XlError.**isidentifier**() → bool
Return True if S is a valid identifier according to the language definition.

Use keyword.iskeyword() to test for reserved identifiers such as "def" and "class".

islower

XlError.**islower**() → bool
Return True if all cased characters in S are lowercase and there is at least one cased character in S, False otherwise.

isnumeric

XlError.**isnumeric**() → bool
Return True if there are only numeric characters in S, False otherwise.

isprintable

XlError.**isprintable**() → bool
Return True if all characters in S are considered printable in repr() or S is empty, False otherwise.

isspace

XlError.**isspace**() → bool
Return True if all characters in S are whitespace and there is at least one character in S, False otherwise.

istitle

XlError.**istitle**() → bool

Return True if S is a titlecased string and there is at least one character in S, i.e. upper- and titlecase characters may only follow uncased characters and lowercase characters only cased ones. Return False otherwise.

isupper

XlError.**isupper**() → bool

Return True if all cased characters in S are uppercase and there is at least one cased character in S, False otherwise.

join

XlError.**join**(*iterable*) → str

Return a string which is the concatenation of the strings in the iterable. The separator between elements is S.

ljust

XlError.**ljust**(*width*[, *fillchar*]) → str

Return S left-justified in a Unicode string of length width. Padding is done using the specified fill character (default is a space).

lower

XlError.**lower**() → str

Return a copy of the string S converted to lowercase.

lstrip

XlError.**lstrip**([*chars*]) → str

Return a copy of the string S with leading whitespace removed. If chars is given and not None, remove characters in chars instead.

maketrans

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) -> (head*, *sep*, *tail*)
Search for the separator sep in S, and return the part before it, the separator itself, and the part after it. If the separator is not found, return S and two empty strings.

replace

XlError.**replace**(*old*, *new*[, *count*]) → str
Return a copy of S with all occurrences of substring old replaced by new. 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
Like S.rfind() but raise ValueError when the substring is not found.

rjust

XlError.**rjust**(*width*[, *fillchar*]) → str
Return S right-justified in a string of length width. Padding is done using the specified fill character (default is a space).

rpartition

XlError.**rpartition**(*sep) -> (head*, *sep*, *tail*)
Search for the separator sep in S, starting at the end of S, and return the part before it, the separator itself, and the part after it. If the separator is not found, return two empty strings and S.

rsplit

XlError.**rsplit**(*sep=None*, *maxsplit=-1*) → list of strings
Return a list of the words in S, using sep as the delimiter string, starting at the end of the string and working to the front. If maxsplit is given, at most maxsplit splits are done. If sep is not specified, any whitespace string is a separator.

rstrip

XlError.**rstrip**([*chars*]) → str

Return a copy of the string S with trailing whitespace removed. If chars is given and not None, remove characters in chars instead.

split

XlError.**split**(*sep=None*, *maxsplit=-1*) → list of strings

Return a list of the words in S, using sep as the delimiter string. If maxsplit is given, at most maxsplit splits are done. If sep is not specified or is None, any whitespace string is a separator and empty strings are removed from the result.

splitlines

XlError.**splitlines**([*keepends*]) → list of strings

Return a list of the lines in S, 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*]) → str

Return a copy of the string S with leading and trailing whitespace removed. If chars is given and not None, remove characters in chars instead.

swapcase

XlError.**swapcase**() → str

Return a copy of S with uppercase characters converted to lowercase and vice versa.

title

XlError.**title**() → str

Return a titlecased version of S, i.e. words start with title case characters, all remaining cased characters have lower case.

translate

XlError.**translate**(*table*) → str
Return a copy of the string S in which each character has been mapped through the given translation table. The table must implement lookup/indexing via __getitem__, for instance a dictionary or list, mapping Unicode ordinals to Unicode ordinals, strings, or None. If this operation raises LookupError, the character is left untouched. Characters mapped to None are deleted.

upper

XlError.**upper**() → str
Return a copy of S converted to uppercase.

zfill

XlError.**zfill**(*width*) → str
Pad a numeric string S with zeros on the left, to fill a field of the specified width. The string S is never truncated.

__init__()
Initialize self. See help(type(self)) for accurate signature.

operator

It provides Operator classes.

Classes

Intersect
Operator
OperatorToken
Separator

Intersect

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

Methods

__init__
ast
compile
match
process
set_expr
update_input_tokens

Continued on next page

Table 2.29 – continued from previous page

update_name

__init__

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

ast

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

compile

Intersect.**compile**()

match

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

process

Intersect.**process**(*match*, *context=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*)

Attributes

get_n_args
name
node_id
pred

get_n_args

Intersect.**get_n_args**

name

Intersect.**name**

node_id

Intersect.**node_id**

pred

Intersect.**pred**

Operator

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

Methods

__init__
ast
compile
match
process
set_expr
update_input_tokens
update_name

__init__

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

ast

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

compile

Operator.**compile**()

match

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

process

Operator.**process**(*match*, *context=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*)

Attributes

get_n_args
name
node_id
pred

get_n_args

Operator.**get_n_args**

name

Operator.**name**

node_id

Operator.**node_id**

pred

Operator.**pred**

OperatorToken

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

Methods

__init__
ast
compile
match
process
set_expr
update_input_tokens
update_name

__init__

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

ast

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

compile

OperatorToken.`compile`()

match

OperatorToken.`match`(*s*)

process

OperatorToken.`process`(*match*, *context=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*)

Attributes

get_n_args
name
node_id
pred

get_n_args

OperatorToken.**get_n_args**

name

OperatorToken.**name**

node_id

OperatorToken.**node_id**

pred

OperatorToken.**pred**

Separator

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

Methods

__init__
ast
compile
match
process
set_expr
update_input_tokens
update_name

__init__

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

ast

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

compile

Separator.**compile**()

match

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

process

Separator.**process**(*match*, *context=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*)

Attributes

get_n_args
name
node_id
pred

get_n_args

Separator.**get_n_args**

name

Separator.**name**

node_id

Separator.**node_id**

pred

Separator.**pred**

parenthesis

It provides Parenthesis class.

Classes

Parenthesis

Parenthesis

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

Methods

__init__
ast
match
process
set_expr
update_input_tokens

__init__

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

ast

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

match

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

process

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

set_expr

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

update_input_tokens

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

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

Attributes

n_args
name
node_id
opens

n_args

Parenthesis.**n_args = 0**

name

Parenthesis.**name**

node_id

Parenthesis.**node_id**

opens

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

Classes

Token

Token

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

Methods

__init__
ast
match
process
set_expr
update_input_tokens

__init__

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

ast

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

match

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

process

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

set_expr

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

update_input_tokens

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

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

Attributes

name
node_id

name

Token.**name**

node_id

Token.**node_id**

ranges

It provides Ranges class.

Classes

Ranges

Ranges

class Ranges(*ranges=()*, *values=None*, *is_set=False*, *all_values=True*)

Methods

__init__
format_range
get_range
push
pushes
set_value
simplify

__init__

Ranges.**__init__**(*ranges=()*, *values=None*, *is_set=False*, *all_values=True*)

format_range

static Ranges.**format_range**(**args*, ***kwargs*)

get_range

static Ranges.**get_range**(*format_range*, *ref*, *context=None*)

push

Ranges.**push**(*ref*, *value=empty*, *context=None*)

pushes

Ranges.**pushes**(*refs*, *values=()*, *context=None*)

set_value

Ranges.**set_value**(*rng*, *value=empty*)

simplify

Ranges.**simplify**()

__init__(*ranges=()*, *values=None*, *is_set=False*, *all_values=True*)

Attributes

input_fields
value

input_fields

Ranges.**input_fields** = **('excel', 'sheet', 'n1', 'n2', 'r1', 'r2')**

value

Ranges.**value**

cell

It provides Cell class.

Functions

format_output
wrap_cell_func

format_output

format_output(*rng*, *value*)

wrap_cell_func

wrap_cell_func(*func*, *parse_args=<function <lambda>>*, *parse_kwargs=<function <lambda>>*)

Classes

Cell
RangesAssembler

Cell

class **Cell**(*reference*, *value*, *context=None*)

Methods

__init__
add
compile
update_inputs

__init__

Cell.**__init__**(*reference*, *value*, *context=None*)

add

Cell.**add**(*dsp*, *context=None*)

compile

Cell.**compile**(*references=None*)

update_inputs

Cell.**update_inputs**(*references=None*)

__init__(*reference*, *value*, *context=None*)

Attributes

output

output

Cell.**output**

RangesAssembler

class **RangesAssembler**(*ref*, *context=None*)

Methods

__init__
push

__init__

RangesAssembler.**__init__**(*ref*, *context=None*)

push

RangesAssembler.**push**(*cell*)

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

Attributes

output

output

RangesAssembler.**output**

excel

It provides excel model class.

Classes

ExcelModel

ExcelModel

class **ExcelModel**

Methods

__init__

Continued on next page

Table 2.53 – continued from previous page

add_book
add_cell
add_sheet
compile
complete
finish
load
loads
push
pushes
write

__init__

ExcelModel.**__init__**()

add_book

ExcelModel.**add_book**(*book*, *context=None*, *data_only=False*)

add_cell

ExcelModel.**add_cell**(*cell*, *context*, *references=None*, *formula_references=None*, *formula_ranges=None*, *external_links=None*)

add_sheet

ExcelModel.**add_sheet**(*worksheet*, *context*)

compile

ExcelModel.**compile**(*inputs*, *outputs*)

complete

ExcelModel.**complete**()

finish

ExcelModel.**finish**(*complete=True*)

load

ExcelModel.**load**(*filename*)

loads

ExcelModel.**loads**(**file_names*)

push

ExcelModel.**push**(*worksheet*, *context*)

pushes

ExcelModel.**pushes**(**worksheets*, ***, *context=None*)

write

ExcelModel.**write**(*books=None*, *solution=None*)

__init__()

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

f

Symbols

A

C

E

F

I

N

O

P

R

S

T

W

X