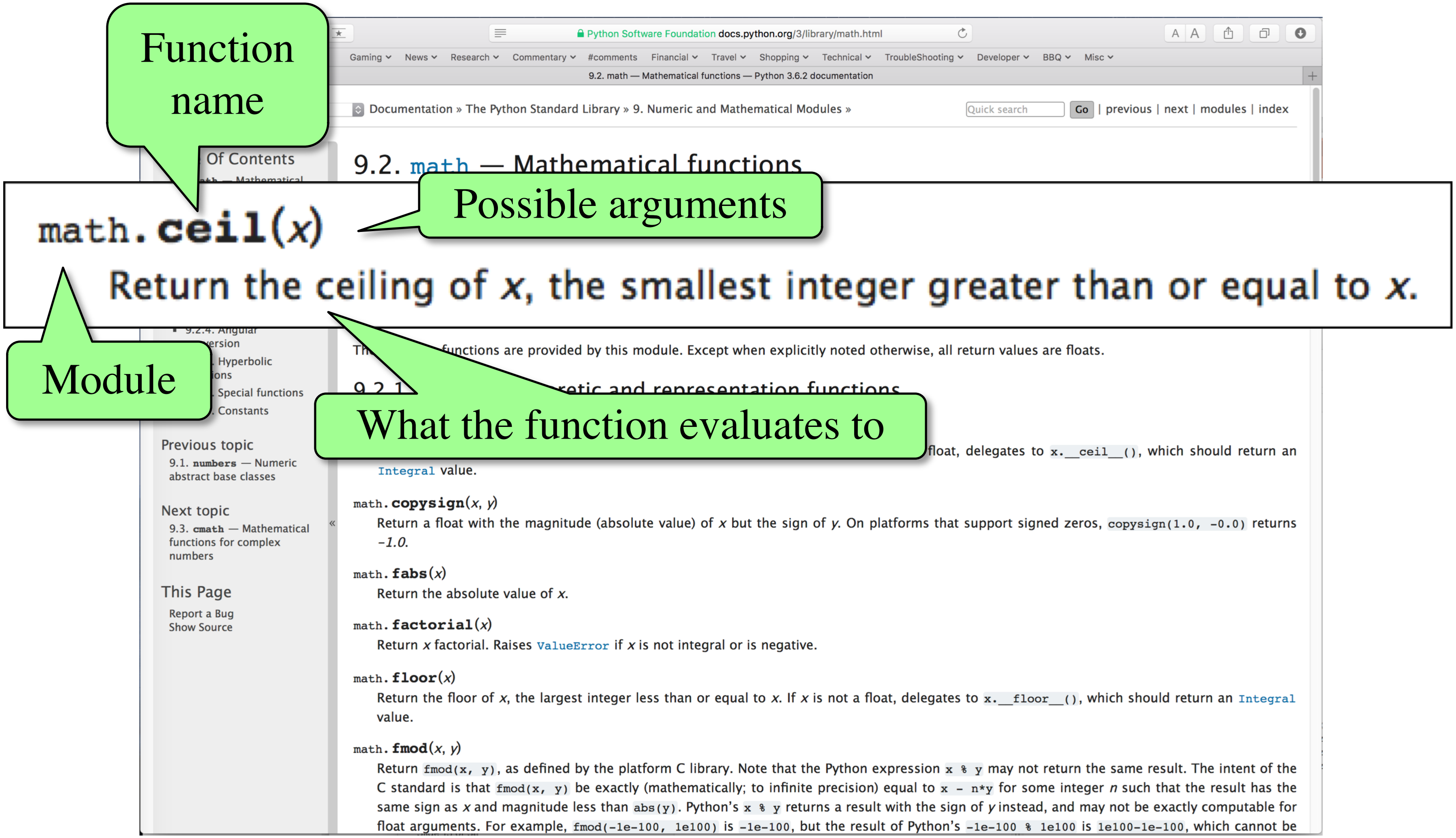
Mini-Lecture 8

Specifications

Recall: The Python API


Python Software Foundation docs.python.org/3/library/math.html
Gaming
News
Research
Commentary
#comments
Financial
Travel
Shopping
Technical
TroubleShooting
Developer
BBQ
Misc
9.2. math — Mathematical functions — Python 3.6.2 documentation
Documentation » The Python Standard Library » 9. Numeric and Mathematical Modules »
Quick search
Go
| previous | next | modules | index
Of Contents
…ath — Mathematical
9.2.4. Angular …ersion
Hyperbolic …ons
Special functions
Constants
Previous topic
9.1. numbers — Numeric abstract base classes
Next topic
9.3. cmath — Mathematical functions for complex numbers
This Page
Report a Bug
Show Source

Function name

Possible arguments

Module

What the function evaluates to

9.2. math — Mathematical functions

math.ceil(*x*)

Return the ceiling of *x*, the smallest integer greater than or equal to *x*.

Th… functions are provided by this module. Except when explicitly noted otherwise, all return values are floats.

9.2.1 …retic and representation functions

…float, delegates to `x.__ceil__()`, which should return an `Integral` value.

math.**copysign**(*x*, *y*)
Return a float with the magnitude (absolute value) of *x* but the sign of *y*. On platforms that support signed zeros, `copysign(1.0, -0.0)` returns *-1.0*.

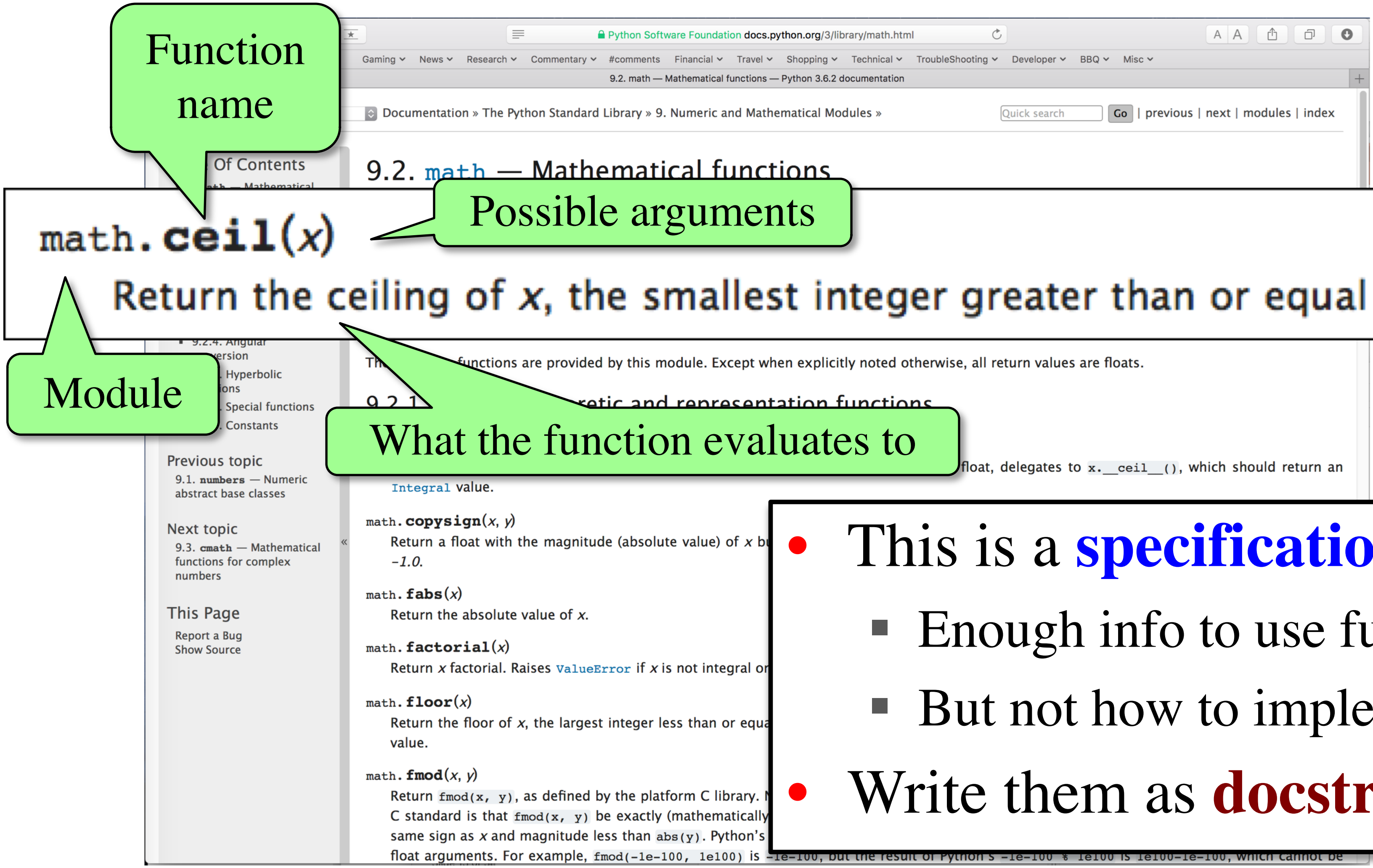
math.**fabs**(*x*)
Return the absolute value of *x*.

math.**factorial**(*x*)
Return *x* factorial. Raises `ValueError` if *x* is not integral or is negative.

math.**floor**(*x*)
Return the floor of *x*, the largest integer less than or equal to *x*. If *x* is not a float, delegates to `x.__floor__()`, which should return an `Integral` value.

math.**fmod**(*x*, *y*)
Return `fmod(x, y)`, as defined by the platform C library. Note that the Python expression `x % y` may not return the same result. The intent of the C standard is that `fmod(x, y)` be exactly (mathematically; to infinite precision) equal to `x - n*y` for some integer *n* such that the result has the same sign as *x* and magnitude less than `abs(y)`. Python's `x % y` returns a result with the sign of *y* instead, and may not be exactly computable for float arguments. For example, `fmod(-1e-100, 1e100)` is `-1e-100`, but the result of Python's `-1e-100 % 1e100` is `1e100-1e-100`, which cannot be

Recall: The Python API


Python Software Foundation docs.python.org/3/library/math.html
Gaming
News
Research
Commentary
#comments
Financial
Travel
Shopping
Technical
TroubleShooting
Developer
BBQ
Misc
9.2. math — Mathematical functions — Python 3.6.2 documentation
Documentation » The Python Standard Library » 9. Numeric and Mathematical Modules »
Quick search
Go
previous | next | modules | index
Of Contents
9.2.4. Angular
ersion
Hyperbolic
ons
Special functions
Constants
Previous topic
9.1. numbers — Numeric abstract base classes
Next topic
9.3. cmath — Mathematical functions for complex numbers
This Page
Report a Bug
Show Source
Function name
Possible arguments
Module
What the function evaluates to

9.2. math — Mathematical functions

math.ceil(*x*)

Return the ceiling of *x*, the smallest integer greater than or equal to *x*.

Th functions are provided by this module. Except when explicitly noted otherwise, all return values are floats.

9.2.1 retic and representation functions

float, delegates to `x.__ceil__()`, which should return an `Integral` value.

math.copysign(*x*, *y*)

Return a float with the magnitude (absolute value) of *x* bu -1.0.

math.fabs(*x*)

Return the absolute value of *x*.

math.factorial(*x*)

Return *x* factorial. Raises `ValueError` if *x* is not integral or

math.floor(*x*)

Return the floor of *x*, the largest integer less than or equa value.

math.fmod(*x*, *y*)

Return `fmod(x, y)`, as defined by the platform C library. N C standard is that `fmod(x, y)` be exactly (mathematically same sign as *x* and magnitude less than `abs(y)`. Python's float arguments. For example, `fmod(-1e-100, 1e100)` is -1e-100, but the result of Python's -1e-100 % 1e100 is 1e100-1e-100, which cannot be

- This is a **specification**
  - Enough info to use func.
  - But not how to implement
- Write them as **docstrings**

Anatomy of a Specification

```
def greet(n):
    """Prints a greeting to the name n

    Greeting has format 'Hello <n>!'
    Followed by conversation starter.

    Parameter n: person to greet
    Precondition: n is a string"""
    print('Hello '+n+'!')
    print('How are you?')
```

One line description, followed by blank line

Anatomy of a Specification

```
def greet(n):
    """Prints a greeting to the name n

    Greeting has format 'Hello <n>!'
    Followed by conversation starter.

    Parameter n: person to greet
    Precondition: n is a string"""
    print('Hello '+n+'!')
    print('How are you?')
```

One line description, followed by blank line

More detail about the function. It may be many paragraphs.

Anatomy of a Specification

```
def greet(n):
    """Prints a greeting to the name n

    Greeting has format 'Hello <n>!'
    Followed by conversation starter.

    Parameter n: person to greet
    Precondition: n is a string"""
    print('Hello '+n+'!')
    print('How are you?')
```

One line description, followed by blank line

More detail about the function. It may be many paragraphs.

Parameter description

Anatomy of a Specification

```
def greet(n):
    """Prints a greeting to the name n

    Greeting has format 'Hello <n>!'
    Followed by conversation starter.

    Parameter n: person to greet
    Precondition: n is a string"""
    print('Hello '+n+'!')
    print('How are you?')
```

One line description, followed by blank line

More detail about the function. It may be many paragraphs.

Parameter description

Precondition specifies assumptions we make about the arguments

Anatomy of a Specification

```
def to_centigrade(x):
    """Returns: x converted to centigrade

    Value returned has type float.

    Parameter x: temp in fahrenheit
    Precondition: x is a float"""
    return 5*(x-32)/9.0
```

One line description, followed by blank line

More detail about the function. It may be many paragraphs.

Parameter description

Precondition specifies assumptions we make about the arguments

Anatomy of a Specification

```
def to_centigrade(x):
    """Returns: x converted to centigrade

    Value returned has type float.

    Parameter x: temp in fahrenheit
    Precondition: x is a float"""
    return 5*(x-32)/9.0
```

“Returns” indicates a fruitful function

More detail about the function. It may be many paragraphs.

Parameter description

Precondition specifies assumptions we make about the arguments

Preconditions

- Precondition is a promise
  - If precondition is true, the function works
  - If precondition is false, no guarantees at all
- Get **software bugs** when
  - Function precondition is not documented properly
  - Function is used in ways that violates precondition

```
>>> to_centigrade(32.0)
0.0
>>> to_centigrade(212)
100.0
```

Preconditions

- Precondition is a promise
  - If precondition is true, the function works
  - If precondition is false, no guarantees at all
- Get **software bugs** when
  - Function precondition is not documented properly
  - Function is used in ways that violates precondition

```
>>> to_centigrade(32.0)
0.0
>>> to_centigrade(212)
100.0
>>> to_centigrade('32')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "temperature.py", line 19 ...
TypeError: unsupported operand type(s)
for -: 'str' and 'int'
```

Precondition violated

String Extraction Example

```
def firstparens(text):
    """Returns: substring in ()
    Uses the first set of parens
    Param text: a string with ()"""

    # Find the open parenthesis

    # Store part AFTER paren

    # Find the close parenthesis

    # Return the result
```

```
>>> s = 'Prof (Walker) White'
>>> firstparens(s)
'Walker’
>>> t = '(A) B (C) D'
>>> firstparens(t)
'A'
```

String Extraction Example

```
def firstparens(text):
    """Returns: substring in ()
    Uses the first set of parens
    Param text: a string with ()"""

    # Find the open parenthesis
    start = introcs.index_str(s,'(')
    # Store part AFTER paren


    # Find the close parenthesis


    # Return the result

```

```
>>> s = 'Prof (Walker) White'
>>> firstparens(s)
'Walker’
>>> t = '(A) B (C) D'
>>> firstparens(t)
'A'
```

String Extraction Example

```
def firstparens(text):
    """Returns: substring in ()
    Uses the first set of parens
    Param text: a string with ()"""

    # Find the open parenthesis
    start = introcs.index_str(s,'(')
    # Store part AFTER paren
    tail = s[start+1:]
    # Find the close parenthesis


    # Return the result
```

```
>>> s = 'Prof (Walker) White'
>>> firstparens(s)
'Walker’
>>> t = '(A) B (C) D'
>>> firstparens(t)
'A'
```

String Extraction Example

```
def firstparens(text):
    """Returns: substring in ()
    Uses the first set of parens
    Param text: a string with ()"""

    # Find the open parenthesis
    start = introcs.index_str(s,'(')
    # Store part AFTER paren
    tail = s[start+1:]
    # Find the close parenthesis
    end = introcs.index_str(tail,')')
    # Return the result
```

```
>>> s = 'Prof (Walker) White'
>>> firstparens(s)
'Walker’
>>> t = '(A) B (C) D'
>>> firstparens(t)
'A'
```

String Extraction Example

```
def firstparens(text):
    """Returns: substring in ()
    Uses the first set of parens
    Param text: a string with ()"""

    # Find the open parenthesis
    start = introcs.index_str(s,'(')
    # Store part AFTER paren
    tail = s[start+1:]
    # Find the close parenthesis
    end = introcs.index_str(tail,')')
    # Return the result
    return tail[:end]
```

```
>>> s = 'Prof (Walker) White'
>>> firstparens(s)
'Walker’
>>> t = '(A) B (C) D'
>>> firstparens(t)
'A'
```

String Extraction Example

```
def second(thelist):
    """Returns: second elt in thelist
    Ex: second('A, B, C') => 'B'
    Param thelist: a list of words
    Precond: thelist has words sep.
    by commas, spaces."""
```

```
>>> second('cat,␣dog, mouse, lion')
'dog'
>>> second('apple,␣pear, banana')
'pear'
```

String Extraction Example

```
def second(thelist):
    """Returns: second elt in thelist
    Ex: second('A, B, C') => 'B'
    Param thelist: a list of words
    Precond: thelist has words sep.
    by commas, spaces."""

    # Find start of second elt

    # Find end of second elt

    # Slice from start to end

    # Return result
```

```
>>> second('cat,␣dog, mouse, lion')
'dog'
>>> second('apple,␣pear, banana')
'pear'
```

String Extraction Example

```
def second(thelist):
    """Returns: second elt in thelist
    Ex: second('A, B, C') => 'B'
    Param thelist: a list of words
    Precond: thelist has words sep.
    by commas, spaces."""

    # Find FIRST comma
    # Find SECOND COMMA
    # Slice from comma to comma
    # Return result
```

```
>>> second('cat, dog, mouse, lion')
'dog'
>>> second('apple, pear, banana')
'pear'
```

String Extraction Example

```
def second(thelist):
    """Returns: second elt in thelist
    Ex: second('A, B, C') => 'B'
    Param thelist: a list of words
    Precond: thelist has words sep.
    by commas, spaces."""

    s = introcs.index_str(thelist,',’)
    e = introcs.index_str(thelist,',',s+1)
    result = thelist[s+1:e]
    return result
```

```
>>> second('cat, dog, mouse, lion')
'dog'
>>> second('apple, pear, banana')
'pear'
```

String Extraction Example

```
def second(thelist):
    """Returns: second elt in thelist
    Ex: second('A, B, C') => 'B'
    Param thelist: a list of words
    Precond: thelist has words sep.
    by commas, spaces."""

    s = introcs.index_str(thelist,',’)
    e = introcs.index_str(thelist,',',s+1)
    result = thelist[s+1:e]
    return result
```

```
>>> second('cat, dog, mouse, lion')
'dog'
>>> second('apple, pear, banana')
'pear'
```

Where is the error?

A: Line 1
B: Line 2
C: Line 3
D: Line 4
E: There is no error

String Extraction Example

```
def second(thelist):
    """Returns: second elt in thelist
    Ex: second('A, B, C') => 'B'
    Param thelist: a list of words
    Precond: thelist has words sep.
    by commas, spaces."""

    s = introcs.index_str(thelist,',')
    e = introcs.index_str(thelist,',',s+1)
    result = thelist[s+1:e]
    return result
```

```
>>> second('cat, dog, mouse, lion')
'dog'
>>> second('apple, pear, banana')
'pear'
```

```
result = thelist[s+2:end]
```

OR

```
result = introcs.strip(result)
```