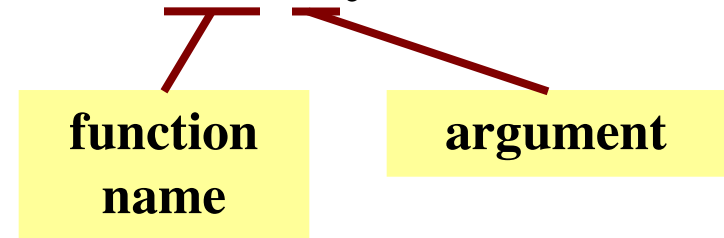


Mini-Lecture 4

Functions

Function Calls

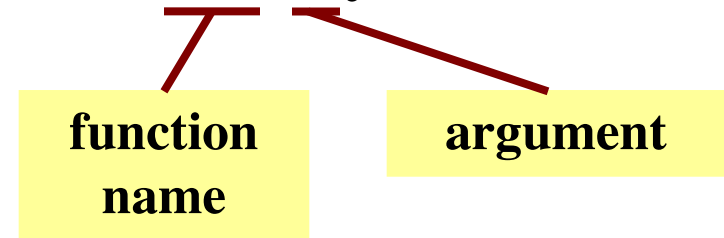
- Python supports expressions with math-like functions
 - A function in an expression is a **function call**
 - Will explain the meaning of this later
- Function expressions have the form **fun(x,y,...)**



- **Examples** (math functions that work in Python):
 - `round(2.34)`
 - `max(a+3,24)`

Function Calls

- Python supports expressions with math-like functions
 - A function in an expression is a **function call**
 - Will explain the meaning of this later
- Function expressions have the form **fun(x,y,...)**



- **Examples** (math functions that work in Python):
 - `round(2.34)`
 - `max(a+3,24)`

Arguments can be any **expression**

Built-In Functions

- You have seen many functions already
 - Type casting functions: `int()`, `float()`, `bool()`
 - Dynamically type an expression: `type()`
 - Help function: `help()`
 - Quit function: `quit()`
- One of the most important function is `print()`
 - `print(exp)` displays value of `exp` on screen
 - Will see later why this is important

Arguments go in (),
but `name()` refers to
function in general

Built-in Functions vs Modules

- The number of built-in functions is small
 - <http://docs.python.org/3/library/functions.html>
- Missing a lot of functions you would expect
 - **Example:** `cos()`, `sqrt()`
- **Module:** file that contains Python code
 - A way for Python to provide optional functions
 - To access a module, the `import` command
 - Access the functions using module as a *prefix*

Example: Module `math`

```
>>> import math
```

```
>>> math.cos(0)
```

```
1.0
```

```
>>> cos(0)
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
NameError: name 'cos' is not defined
```

```
>>> math.pi
```

```
3.141592653589793
```

```
>>> math.cos(math.pi)
```

```
-1.0
```

Example: Module `math`

```
>>> import math
```

To access math functions

```
>>> math.cos(0)
```

```
1.0
```

```
>>> cos(0)
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
NameError: name 'cos' is not defined
```

```
>>> math.pi
```

```
3.141592653589793
```

```
>>> math.cos(math.pi)
```

```
-1.0
```

Example: Module `math`

```
>>> import math
```

To access math functions

```
>>> math.cos(0)
```

```
1.0
```

```
>>> cos(0)
```

Functions require math prefix!

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in <module>
```

```
NameError: name 'cos' is not defined
```

```
>>> math.pi
```

```
3.141592653589793
```

```
>>> math.cos(math.pi)
```

```
-1.0
```


Example: Module `math`

```
>>> import math
```

To access math functions

```
>>> math.cos(0)
```

```
1.0
```

Functions require math prefix!

```
>>> cos(0)
```

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in <module>
```

```
NameError: name 'cos' is not defined
```

```
>>> math.pi
```

Module has variables too!

```
3.141592653589793
```

```
>>> math.cos(math.pi)
```

```
-1.0
```

Example: Module `math`

```
>>> import math
```

To access math functions

```
>>> math.cos(0)
```

```
1.0
```

```
>>> cos(0)
```

Functions require math prefix!

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in <module>
```

```
NameError: name 'cos' is not defined
```

```
>>> math.pi
```

Module has variables too!

```
3.141592653589793
```

```
>>> math.cos(math.pi)
```

```
-1.0
```

Other Modules

- `io`
 - Read/write from files
- `random`
 - Generate random numbers
 - Can pick any distribution
- `string`
 - Useful string functions
- `sys`
 - Information about your OS

Using the from Keyword

```
>>> import math
```

```
>>> math.pi
```

Must prefix with
module name

```
3.141592653589793
```

```
>>> from math import pi
```

```
>>> pi
```

No prefix needed
for variable pi

```
3.141592653589793
```

```
>>> from math import *
```

```
>>> cos(pi)
```

```
-1.0
```

No prefix needed
for anything in math

- Be careful using from!
- Using import is *safer*
 - Modules might conflict (functions w/ same name)
 - What if import both?
- **Example:** cos
 - 2 modules: math, numpy
 - Both have func. cos()
 - Each has tradeoffs

Other Variations

```
>>> import math as m
```

```
>>> m.cos(0)
```

```
1.0
```

```
>>> from math import cos
```

```
>>> cos(0)
```

```
1.0
```

```
>>> sin(0)
```

```
ERROR
```

Reading the Python Documentation

The screenshot shows a web browser displaying the Python documentation for the `math` module. The browser's address bar shows the URL `docs.python.org/3/library/math.html`. The page title is "9.2. math — Mathematical functions — Python 3.6.2 documentation". The left sidebar contains a "Table Of Contents" for the `math` module, listing sections from 9.2.1 to 9.2.7. The main content area is titled "9.2. math — Mathematical functions" and contains the following text:

This module is always available. It provides access to the mathematical functions defined by the C standard.

These functions cannot be used with complex numbers; use the functions of the same name from the `cmath` module if you require support for complex numbers. The distinction between functions which support complex numbers and those which don't is made since most users do not want to learn quite as much mathematics as required to understand complex numbers. Receiving an exception instead of a complex result allows earlier detection of the unexpected complex number used as a parameter, so that the programmer can determine how and why it was generated in the first place.

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

9.2.1. Number-theoretic and representation functions

`math.ceil(x)`
Return the ceiling of `x`, the smallest integer greater than or equal to `x`. If `x` is not a 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 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

A red box highlights the URL `http://docs.python.org/library` in the bottom right corner of the page.

Reading the Python Documentation

The screenshot shows the Python documentation page for the `math` module. The page title is "9.2. `math` — Mathematical functions". The main content includes a description of the module and a list of functions. A callout box highlights the following function:

`math.ceil(x)`
Return the ceiling of `x` as a float, the smallest integer value greater than or equal to `x`.

Other functions visible in the screenshot include:

- `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 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

Reading the Python Documentation

The screenshot shows the Python documentation page for the `math` module. The page title is "9.2. `math` — Mathematical functions". The page content includes a description of the module, a list of functions, and a detailed description of the `math.ceil(x)` function. The callouts point to the following elements:

- Function name:** Points to the function signature `math.ceil(x)`.
- Possible arguments:** Points to the parameter `x` in the function signature.
- Module:** Points to the `math` module name in the function signature.
- What the function evaluates to:** Points to the description of the function: "Return the ceiling of `x` as a float, the smallest integer value greater than or equal to `x`."

The URL <http://docs.python.org/library> is highlighted in a box at the bottom right of the screenshot.