

CS1110 Spring 2018 Announcements

Check course page for announcements!
<http://www.cs.cornell.edu/courses/cs1110/2018sp>

ENGRG 1010. AEW workshops still space

- can enroll through Student Center
- 1-credit S/U course
- 2-hour weekly workshop
- work on related problem sets

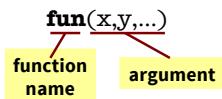
Full? Or need a different time?

<https://tinyurl.com/aew-request>

2

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:



- Some math functions built into Python:
 - `round(2.34)`
 - `max(a+3,24)`

Arguments can be
any expression

4

Modules

- “Libraries” of functions and variables
- To access a module, use the import command:
`import <module name>`
- Can then access functions like this:
`<module name>.<function name>(<arguments>)`
- **Example:**

```
>>> import math  
>>> math.cos(2.0)  
-0.4161468365471424
```

7

Module Variables

- Modules can have variables, too
- Can access them like this:
`<module name>.<variable name>`
- **Example:**

```
>>> import math  
>>> math.pi  
3.141592653589793
```

9

A Closer Reading of the Python Documentation

<https://docs.python.org/3/library/math.html>

A screenshot of the Python documentation for the `math` module. The page shows various mathematical functions. Callouts highlight specific parts of the documentation:

- A callout labeled "Function name" points to the `ceil` function.
- A callout labeled "Possible arguments" points to the parameters of the `ceil` function.
- A callout labeled "Module" points to the `math` module entry in the sidebar.
- A callout labeled "What the function evaluates to" points to the description of what the function returns.

The sidebar on the left lists other modules like `random`, `math`, and `operator`. The bottom right corner of the screenshot has the number "12".

1

Interactive Shell vs. Modules

Python Interactive Shell

```
[iMac:~] leithberg-hsl-172:~ = python  
Python 3.6.1 (Anaconda 4.4.0 (46.64)) (default, May  
31 2017, 13:04:09)  
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-800.0.57  
3) on darwin]  
Type "copyright", "credits" or "license" for  
more information.  
...>>> x = 1+2  
...>>> x = 3*x  
...>>> x
```

- Type python at command line
- Type commands after >>>
- Python executes as you type

Module

```
my_module.py  
1 # my_module.py  
2  
3 """ This is a simple module.  
4 It shows how modules work """  
5  
6 x = 1+2  
7 x = 3*x
```

- Written in text editor
- Loaded through import
- Python executes statements when import is called

Section 2.4 in your textbook discusses a few differences

15

my_module.py

Module Text

```
# my_module.py
```

Single line comment
(not executed)

"""This is a simple module.

It shows how modules work"""

```
x = 1+2  
x = 3*x
```

Commands
Executed on import

Docstring
(note the Triple Quotes)

Acts as a multi-line comment
Useful for *code documentation*

16

You Must import

C:\> python
>>> import my_module
>>> my_module.x
9

C:\> python
>>> my_module.x
Traceback (most recent call last):
 File "<stdin>", line 1, in <module>
NameError: name 'my_module' is not defined

25

Dangers of Importing Everything

```
>>> e = 12345  
>>> from math import *  
>>> e  
2.718281828459045
```

e was overwritten!

30

Modules vs. Scripts

Module

- Provides functions, variables
- import it into Python shell

Script

- Behaves like an application
- Run it from command line

Files look the same. Difference is how you use them.

42

Next Time: Defining Functions

- Today we created a module with a *variable*
- Have not discussed how to make a *function*
- **Example:**

```
>>> import math  
>>> math.cos(2.0) ← we want to  
-0.4161468365471424 make functions  
like this
```

43