Lecture 3

Functions & Modules
Labs this Week

• Lab 1 is due at the **beginning** of your lab
  ▪ If it is not yet by then, you cannot get credit
  ▪ Only exception is for students who added late
    (Those students should talk to me)

• Should spend time **entirely** on Lab 2
  ▪ Getting behind this early is bad
  ▪ We are getting you ready for Assignment 1
**Thursday Reading**

- Chapter 3 in the text
  - But can skip section 3.9
- Browse the Python API
  - Will learn what that is today
  - Do not need to read all of it

**Next Week**

- Sections 8.1, 8.2, 8.5, 8.8
Reminder: AEWs

• Small discussion study group for class
  ▪ Graded on attendance only
  ▪ Can really help if you have no experience
• Enrollment is extremely low right now
  ▪ Each section has room for up to 25 people
  ▪ Recommend it if you have the time
• Recommended to help with exams

8/31/17
Defining 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)`
Built-In Functions

• You have seen many functions already
  ▪ Type casting functions: \texttt{int()}, \texttt{float()}, \texttt{bool()}
  ▪ Dynamically type an expression: \texttt{type()}
  ▪ Help function: \texttt{help()}
  ▪ Quit function: \texttt{quit()}
  
  Arguments go in (), but \texttt{name()} refers to function in general

• One of the most important functions is \texttt{print()}
  ▪ \texttt{print(exp)} displays value of \texttt{exp} on screen
  ▪ Will see later why this is important
Built-in Functions vs Modules

• The number of built-in functions is small
  ▪ [http://docs.python.org/3/library/functions.html](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

```python
>>> 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
>>> 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

>>> 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`

```python
>>> 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
```

To access math functions

Functions require math prefix!

Module has variables too!
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

Other Modules

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

To access math functions
Functions require math prefix!
Module has variables too!
9.2. math — Mathematical functions

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. Not a C standard is that `fmod(x, y)` be exactly (mathematically; to within the limits of floating point precision) equal to `x - x // y * y` for some integer `n` such that the result has the same sign as `x` and magnitude less than `|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(-1e100, 1e100)` is -1e100, but the result of Python’s `-1e-100 % 1e100` is 1e-100-1e-100, which cannot be

---

http://docs.python.org/library
Reading the Python Documentation

9.2. math — Mathematical functions

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.

```python
math.ceil(x)
```

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

http://docs.python.org/library
Reading the Python Documentation

http://docs.python.org/library

Function name

Possible arguments

Module

What the function evaluates to

math.ceil(x)

Return the ceiling of x as a float, the smallest integer value greater than or equal to x.
Interactive Shell vs. Modules

- Launch in command line
- Type each line separately
- Python executes as you type

- **Write in a text editor**
  - We use Komodo Edit
  - But anything will work
- Load module with `import`

---

[Image of command line and text editor with code]

8/29/17 Functions & Modules 17
Using a Module

Module Contents

"""" A simple module.

This file shows how modules work
"""

# This is a comment

x = 1+2
x = 3*x
x

8/29/17
Using a Module

Module Contents

""" A simple module.

This file shows how modules work
"""

# This is a comment

x = 1+2
x = 3*x

x

Single line comment (not executed)
Using a Module

Module Contents

""" A simple module.

This file shows how modules work
"""

# This is a comment

x = 1+2
x = 3*x
x

Single line comment (not executed)

Docstring (note the Triple Quotes)
Acts as a multiple-line comment
Useful for code documentation
Using a Module

Module Contents

""" A simple module.

This file shows how modules work
"""

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

8/29/17
Using a Module

Module Contents

""" A simple module.
This file shows how modules work
"""

# This is a comment
x = 1+2
x = 3*x
x

Docstring (note the Triple Quotes)
Acts as a multiple-line comment
Useful for code documentation

Single line comment
(not executed)

Commands
Executed on import

Not a command.
import ignores this
# Using a Module

## Module Contents

""" A simple module.

This file shows how modules work
"""

# This is a comment

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

## Python Shell

```python
>>> import module

>>> x
```
Using a Module

Module Contents

""
A simple module.

This file shows how modules work
""

# This is a comment

x = 1+2
x = 3*x
x

Python Shell

>>> import module
>>> x
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
Using a Module

Module Contents

""" A simple module.

This file shows how modules work
"""

# This is a comment

x = 1+2
x = 3*x
x

Python Shell

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

>>> module.x
9

"""Module data""" must be prefixed by module name
# Using a Module

## Module Contents

```python
""" A simple module.

This file shows how modules work
"""

# This is a comment

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

```python
"""Module data""" must be prefixed by module name
```

```python
"docstring" and module contents
```

---

## Python Shell

```python
>>> import module
```

```python
>>> x
```

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

```n
>>> module.x
```

```python
9
```

```python
>>> help(module)
```

---

8/29/17 Functions & Modules
Modules Must be in Working Directory!

Module you want is in this folder
Modules Must be in Working Directory!

Have to navigate to folder **BEFORE** running Python.
Using the `from` Keyword

```python
>>> import math
>>> math.pi
3.141592653589793
>>> from math import pi
>>> pi
3.141592653589793
>>> from math import *
>>> cos(pi)
-1.0
```

- Be careful using `from`!
- Using `import` is safer
  - Modules might conflict (functions w/ same name)
  - What if import both?
- **Example**: Turtles
  - Used in Assignment 4
  - 2 modules: turtle, tkturtle
  - Both have func. Turtle()
Modules vs. Scripts

Module

- Provides functions, variables
  - Example: temp.py
- import it into Python shell
  >>> import temp
  >>> temp.to_fahrenheit(100)
  212.0
  >>>

Script

- Behaves like an application
  - Example: helloApp.py
- Run it from command line:
  python helloApp.py

Hello World!
Modules vs. Scripts

Module

- Provides functions, variables
  - **Example**: temp.py
- import it into Python shell
  
  ```
  >>> import temp
  >>> temp.to_fahrenheit(100)
  212.0
  >>>
  ```

Script

- Behaves like an application
  - **Example**: helloApp.py
- Run it from command line:
  
  ```
  python helloApp.py
  Hello World!
  ```

Files look the same. Difference is how you use them.
## Scripts and Print Statements

<table>
<thead>
<tr>
<th>module.py</th>
<th>script.py</th>
</tr>
</thead>
</table>
| """ A simple module. """

This file shows how modules work

```
# This is a comment
x = 1+2
x = 3*x
x
```

| |
|--------|--------|
| script.py | """ A simple script. """

This file shows why we use print

```
# This is a comment
x = 1+2
x = 3*x
print(x)
```

8/29/17
### Scripts and Print Statements

<table>
<thead>
<tr>
<th>module.py</th>
<th>script.py</th>
</tr>
</thead>
</table>
| """" A simple module. """
This file shows how modules work """

# This is a comment
x = 1 + 2
x = 3 * x
x |

<table>
<thead>
<tr>
<th>Only difference</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>script.py</th>
<th>module.py</th>
</tr>
</thead>
</table>
| """" A simple script. """
This file shows why we use print """

# This is a comment
x = 1 + 2
x = 3 * x
print(x) |
Scripts and Print Statements

module.py

- Looks like nothing happens
- Python did the following:
  - Executed the assignments
  - Skipped the last line
    (‘x’ is not a statement)

script.py

- We see something this time!
- Python did the following:
  - Executed the assignments
  - Executed the last line
    (Prints the contents of x)
Scripts and Print Statements

module.py

- Looks like nothing happens
- Python did the following:
  - Executed the assignments
  - Skipped the last line ('x' is not a statement)

script.py

- We see something this time!
- Python did the following:
  - Executed the assignments
  - Executed the last line (Prints the contents of x)
Next Time: Defining Functions

Function Call

- Command to **do** the function
- Can put it anywhere
  - In the Python shell
  - Inside another module

Function Definition

- Command to **do** the function
- Belongs inside a module

8/29/17

Functions & Modules
Next Time: Defining Functions

**Function Call**
- Command to do the function
- Can put it anywhere
  - In the Python shell
  - Inside another module

**Function Definition**
- Command to do the function
- Belongs inside a module

But only define function **ONCE**

Can **call** as many times as you want
Next Time: Defining Functions

Function Call

- Command to **do** the function
- Can put it anywhere
  - In the Python shell
  - Inside another module

```
>>> import plusone
>>> plusone.plus(1)
2
>>> plusone.plus(2)
3
>>> plusone.plus(3)
4
```

**arguments** inside ()

Can **call** as many times as you want

Function Definition

- Command to **do** the function
- Belongs inside a module

```
def plus(n):
    """
    Returns: the value of n+1
    """
    return (n+1)
```

But only define function **ONCE**
Functions and Modules

• Purpose of modules is function definitions
  ▪ Function definitions are written in module file
  ▪ Import the module to call the functions

• Your Python workflow (right now) is

1. Write a function in a module (a .py file)
2. Open up the Terminal/Command Prompt
3. Move to the directory with this file
4. Start Python (type python)
5. Import the module
6. Try out the function