(Optional) Readings

Reading for Next Week

- 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
- Sections 8.1, 8.2, 8.5, 8.8
  - Strings are needed for A1
  - But Chap 8 mixes easy stuff with advanced stuff

8/30/18 Functions & Modules
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 $\text{fun}(x, y, \ldots)$

- **Examples** (math functions that work in Python):
  - $\text{round}(2.34)$
  - $\text{max}(a + 3, 24)$
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 functions is print()
  ▪ print(exp) displays value of 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**

```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

```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

```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`

```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, `math.` prefix is required!

Functions require math prefix!

Module has variables too!

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

```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()
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.

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. No C standard is that fmod(x, y) be exactly (mathematically; to within 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(-1e100, 1e100) is -1e100, but the result of Python's -1e100 % 1e100 is 1e100-1e100, which cannot be
9.2. \texttt{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 \texttt{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.

\begin{verbatim}
math.ceil(x)
\end{verbatim}

Return the ceiling of \textit{x}, the smallest integer greater than or equal to \textit{x}.
Reading the Python Documentation

http://docs.python.org/library

Functions & Modules

http://docs.python.org/library

8/30/18
Interactive Shell vs. Modules

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

- Write in a code editor
  - We use Atom Editor
  - But anything will work
- Load module with `import`
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

8/30/18
Using a Module

Module Contents

""" A simple module.

This file shows how modules work
"""

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

Single line comment
(not executed)

8/30/18
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```

8/30/18

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
"""

# This is a comment

```
x = 1+2
x = 3*x
8/30/18
```

Functions & Modules

Single line comment
(not executed)

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

Commands
Executed on import
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

x = 1+2
x = 3*x
x

Python Shell

>>> 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
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

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

## Python Shell

```python
>>> import module

>>> x

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

>>> module.x

9

>>> help(module)

```

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

Prints docstring and module contents
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

Module you want is in this folder
## Modules vs. Scripts

<table>
<thead>
<tr>
<th>Module</th>
<th>Script</th>
</tr>
</thead>
</table>
| • Provides functions, variables  
  ▪ **Example**: temp.py  
  • import it into Python shell  
    >>> import temp  
    >>> temp.to_fahrenheit(100)  
    212.0  
    >>> | • 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

**module.py**

""" 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)
```
Scripts and Print Statements

module.py

""" 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)

8/30/18

Only difference
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)
When you run a script, only statements are executed.

<table>
<thead>
<tr>
<th>module.py</th>
</tr>
</thead>
<tbody>
<tr>
<td>```python</td>
</tr>
<tr>
<td>x = 9</td>
</tr>
<tr>
<td>print(x)</td>
</tr>
<tr>
<td>```</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>script.py</th>
</tr>
</thead>
<tbody>
<tr>
<td>```python</td>
</tr>
<tr>
<td>x = 9</td>
</tr>
<tr>
<td>print(x)</td>
</tr>
<tr>
<td>```</td>
</tr>
</tbody>
</table>

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

- We see something this time!
- Python did the following:
  - Executed the assignments
  - Executed the last line
    (Prints the contents of x)
User Input

```python
>>> input('Type something')
Type something: abc
'abc'

>>> input('Type something: ')
Type something: abc
'abc'

>>> x = input('Type something: ')
Type something: abc
```

Assign result to variable.

No space after the prompt.

Proper space after prompt.
Making a Script Interactive

A script showing off input.

This file shows how to make a script interactive.

```python
x = input("Give me a something: ")
print("You said: "+x)
```

[wmw2] folder> python script.py
Give me something: Hello
You said: Hello

[wmw2] folder> python script.py
Give me something: Goodbye
You said: Goodbye

Not using the interactive shell
Numeric Input

- input returns a string
  - Even if looks like int
  - It cannot know better
- You must convert values
  - `int()`, `float()`, `bool()`, etc.
  - Error if cannot convert
- One way to program
  - But it is a *bad* way
  - Cannot be automated

```python
>>> x = input('Number: ')
Number: 3
>>> x
'3'
>>> x + 1
TypeError: must be str, not int
>>> x = int(x)
>>> x+1
4
```
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

---

8/30/18
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

---

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

But only define function **ONCE**
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**