Lecture 3: Functions & Modules
(Sections 3.1-3.3, 2.4)
CS 1110
Introduction to Computing Using Python

Function Calls

• Function expressions have the form:
  \[ \text{fun}(x, y, \ldots) \]

• Some math functions built into Python:

```
>>> x = 5
>>> y = 4
>>> bigger = max(x, y)
>>> bigger
5
```

Always-available Built-in Functions

• You have seen many functions already
  * Type casting functions: `int()`, `float()`, `bool()`
  * Get type of a value: `type()`
  * Exit function: `exit()`

```
>>> a = round(3.14159265)
>>> a
3
```

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

Module Variables

• Modules can have variables, too
• Can access them like this:
  `<module name>`, `<variable name>`

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

Announcements/Reminders

• New seat assignments for all students at in-person sections. See CMS “Seat Assignments – week 2”.
• Textbook: We deactivated instant access ($$$) for CS1110; use the free online version from the course homepage
• Do pre-lecture activities (reading/videos) before each lecture
• Zoom: please use the raise hand tool 🙋🏻 to indicate that you want to ask a question. Lower hand afterwards.
Visualizing functions & variables available

- So far just built-ins

```
C:\> python
>>> int()
float()
str()
type()
print()
```

Visualizing functions & variables available

- So far just built-ins
- Now we’ve defined a new variable

```
C:\> python
>>> x = 7
>>> x
7
```

Visualizing functions & variables available

- So far just built-ins
- Now we’ve defined a new variable
- Now we’ve imported a module

```
C:\> python
>>> x = 7
>>> import math
>>> help(<module name>)
```

**module help**

After importing a module, can see what functions and variables are available:

```
>>> help(math)
```

**Reading the Python Documentation**

https://docs.python.org/3.7/library/math.html

**Function name**

<table>
<thead>
<tr>
<th>Possible arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module</td>
</tr>
<tr>
<td>What the function evaluates to</td>
</tr>
</tbody>
</table>

---

---
Other Useful Modules

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

Making your Own Module

Write in a text editor

We recommend Atom…
…but any editor will work

Interactive Shell vs. Modules

**Python Interactive Shell**

- Type `python` at command line
- Type commands after `>>>`
- Python executes statements when `import` is called

**Module**

- Written in text editor
- Loaded through `import`
- Python executes as you type

Section 2.4 in your textbook discusses a few differences

my_module.py

```python
# my_module.py

'''This is a simple module.
It shows how modules work'''

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

What’s in the module

- Single line comment (not executed)
- **Docstring**
  - (note the Triple Quotes)
  - Acts as a multi-line comment
  - Useful for code documentation

- **Commands**
  - Executed on `import`

Using a Module (my_module.py)

**Module Text**

```python
# my_module.py

'''This is a simple module.
It shows how modules work'''

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

**Python Shell**

```python
>>> import my_module
```

```
>>> import my_module

This is a simple module.
It shows how modules work

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

Needs to be the same name as the file without the “.py”

Modules Must be in Working Directory!

Must run `python` from same folder as the module

---

3
On import…

On import, Python does not execute the code that is preceded by a `#` symbol (comments) and the code that is inside triple quotes (docstrings).

```python
# my_module.py

'''This is a simple module.
It shows how modules work'''

x = 1+2
x = 3*x

```

```python
Python executes this.

```

```python
Python does not execute
(because of #)
```

```python
Python does not execute
(because of """)
```

Python executes this.

variable x stays “within” the module

Clicker Question!

After you hit “Return” here what will python print next?

- (A) `>>>
- (B) 0.0
- (C) an error message
- (D) The text of fah2cel.py
- (E) Sorry, no clue.

Using a Module (my_module.py)

```python
# my_module.py

'''This is a simple module.
It shows how modules work'''

x = 1+2
x = 3*x

```

```python
Python executes this.

```

```python
Python executes this.

```

```python
module name

variable to access

```

You Must import

With import

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

Without import

```python
C:\> python
>>> p = math.ceil(3.14159)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'math' is not defined
```

You Must Use the Module Name

```python
>>> import my_module
>>> my_module.x
9
```

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

What does the docstring do?

```python
# my_module.py

'''This is a simple module.
It shows how modules work'''

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

```python
my_module

```

```python
my_module

```
from command

• You can also import like this:
  
  ```python
  from <module> import <function name>
  ```

• Example:

  ```python
  >>> from math import pi
  >>> pi
  3.141592653589793
  ```

  `pi` no longer need the module name

from command

• You can also import * everywhere:

  ```python
  from <module> import *
  ```

• Example:

  ```python
  >>> from math import *
  >>> pi
  3.141592653589793
  >>> ceil(pi)
  4
  ```

  Module functions now behave like built-in functions

Dangers of Importing Everything

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

`e` was overwritten!

Avoiding from Keeps Variables Separate

```python
>>> e = 12345
>>> import math
>>> math.e
2.718281828459045
>>> e
12345
```

Ways of Executing Python Code

1. running the Python Interactive Shell
2. importing a module
3. NEW: running a script

Running a Script

• From the command line, type:

  ```bash
  python <script filename>
  ```

• Example:

  ```bash
  C:\> python my_module.py
  C:\>
  ```

  looks like nothing happened

• Actually, something did happen

  * Python executed all of my_module.py
Running my_module.py as a script

my_module.py

```python
# my_module.py

'''This is a simple module.
It shows how modules work'''

```x = 1+2
x = 3\times

```Command Line

C:\> python module.py

```

Python does not execute (because of """
and """")

Python executes this.

Python executes this.

Creating Evidence that the Script Ran

• New (very useful!) command: `print`  
  
  ```python
  print(<expression>)
  ```
  
• `print` evaluates the `<expression>` and writes the value to the console

Running script.py as a script

```python
# script.py

''' This is a simple script.
It shows why we use print'''

```x = 1+2
x = 3\times

```Command Line

C:\> python script.py

9

C:\> python script.py

```

\text{Only difference}

```

Clicker Question

fah2cel.py

```python
# fah2cel.py

'''Convert 32 degrees Fahrenheit
to degrees Celsius'''

```f= 32.0
c= (f-32)*5/9

Command Line

C:\> python fah2cel.py

```

After you hit "Return" here what will be printed next?

(A) >>>
(B) 0.0
(C) an error message
(D) The text of fah2cel.py
(E) Sorry, no clue.

my_module.py vs. script.py

```python
# my_module.py

''' This is a simple module.
It shows how modules work'''

```x = 1+2
x = 3\times

```script.py

```python
# script.py

''' This is a simple script.
It shows why we use print'''

```x = 1+2
x = 3\times

```print(x)

```

```

Only difference

Syntax:

```

print(<expression>)
```
## Subtle difference about script mode

<table>
<thead>
<tr>
<th>Interactive mode</th>
<th>script.py</th>
</tr>
</thead>
<tbody>
<tr>
<td>C:&gt; python</td>
<td># script.py</td>
</tr>
</tbody>
</table>
| >>> x = 1+2      | ```
| >>> x = 3*x      | """This is a simple script.
| >>> x            | It shows why we use print""
| 9                | x = 1+2
| >>> print(x)     | x = 3*x
| 9                | print(x)
| >>>              | # note: in script mode, you will
|                  | # not get output if you just type x

## Modules vs. Scripts

<table>
<thead>
<tr>
<th>Module</th>
<th>Script</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provides functions, variables</td>
<td>Behaves like an application</td>
</tr>
<tr>
<td>import it into Python shell</td>
<td>Run it from command line</td>
</tr>
<tr>
<td>Within Python shell you have access to the functions and variables of the imported module</td>
<td>After running the app you’re back at the command line (not in Python shell)</td>
</tr>
</tbody>
</table>

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