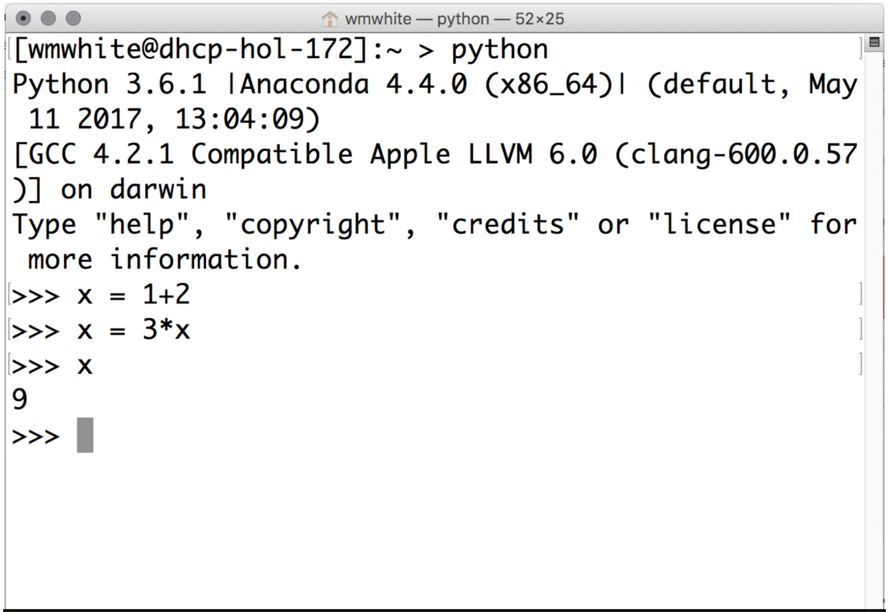
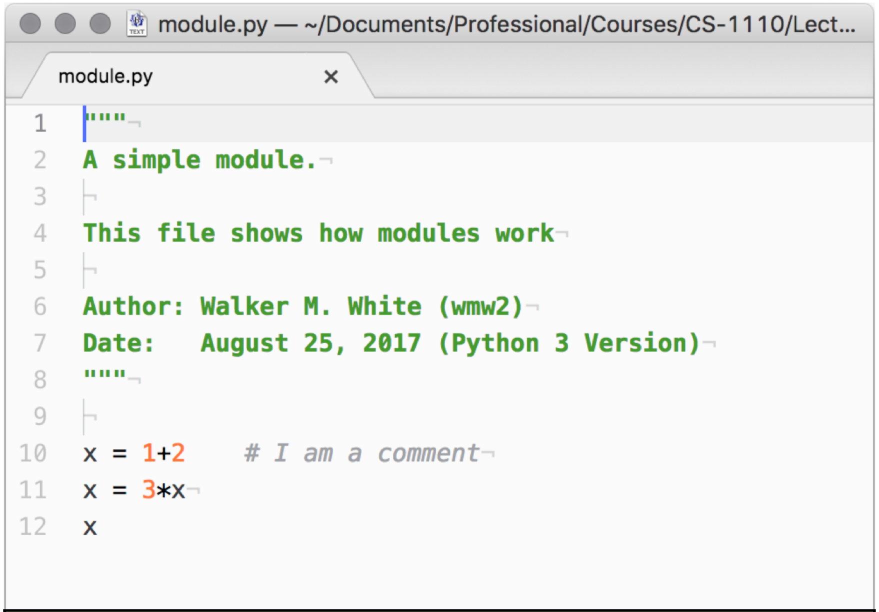
Not-So-Mini-Lecture 6

Modules & Scripts

Interactive Shell vs. Modules



```
[wmwhite@dhcp-hol-172]:~ > python
Python 3.6.1 |Anaconda 4.4.0 (x86_64)| (default, May
 11 2017, 13:04:09)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57
)] on darwin
Type "help", "copyright", "credits" or "license" for
 more information.
>>> x = 1+2
>>> x = 3*x
>>> x
9
>>> 
```



module.py — ~/Documents/Professional/Courses/CS-1110/Lect...
module.py

```
"""
A simple module.

This file shows how modules work

Author: Walker M. White (wmw2)
Date:   August 25, 2017 (Python 3 Version)
"""

x = 1+2     # I am a comment
x = 3*x
x
```

- 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
```

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
```

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

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
```

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

Single line comment
(not executed)

Commands
Executed on import

Using a Module

Module Contents

```
""" A simple module.

This file shows how modules work
"""
```

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

```
# This is a comment
```

Single line comment
(not executed)

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

Commands
Executed on import

```
x
```

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
```

“**Module data**” must be prefixed by module name

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
```

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
```

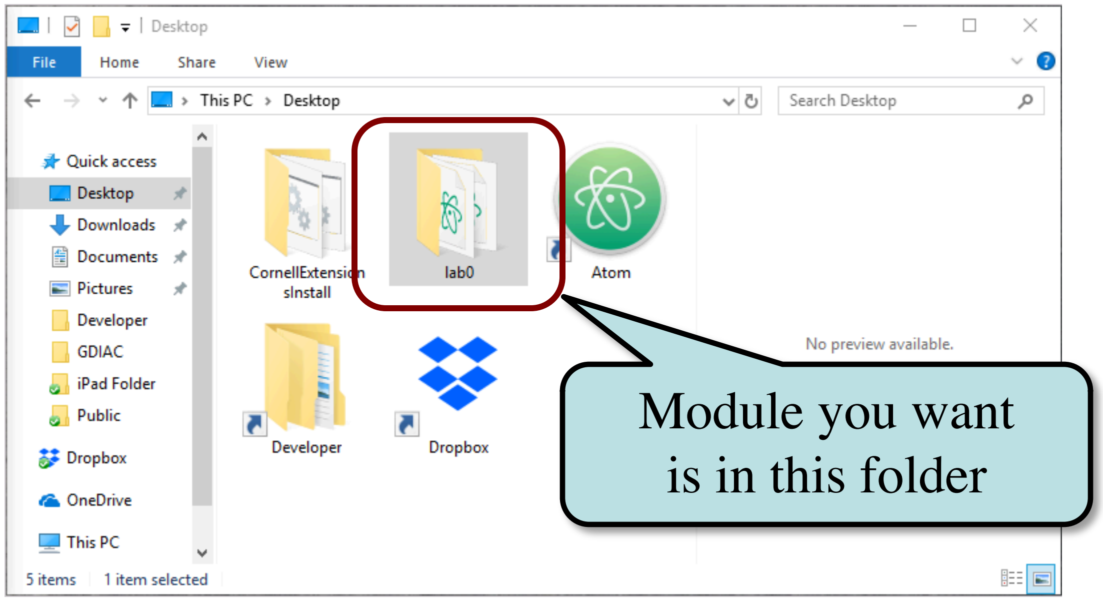
“**Module data**” must be prefixed by module name

Prints **docstring** and module contents

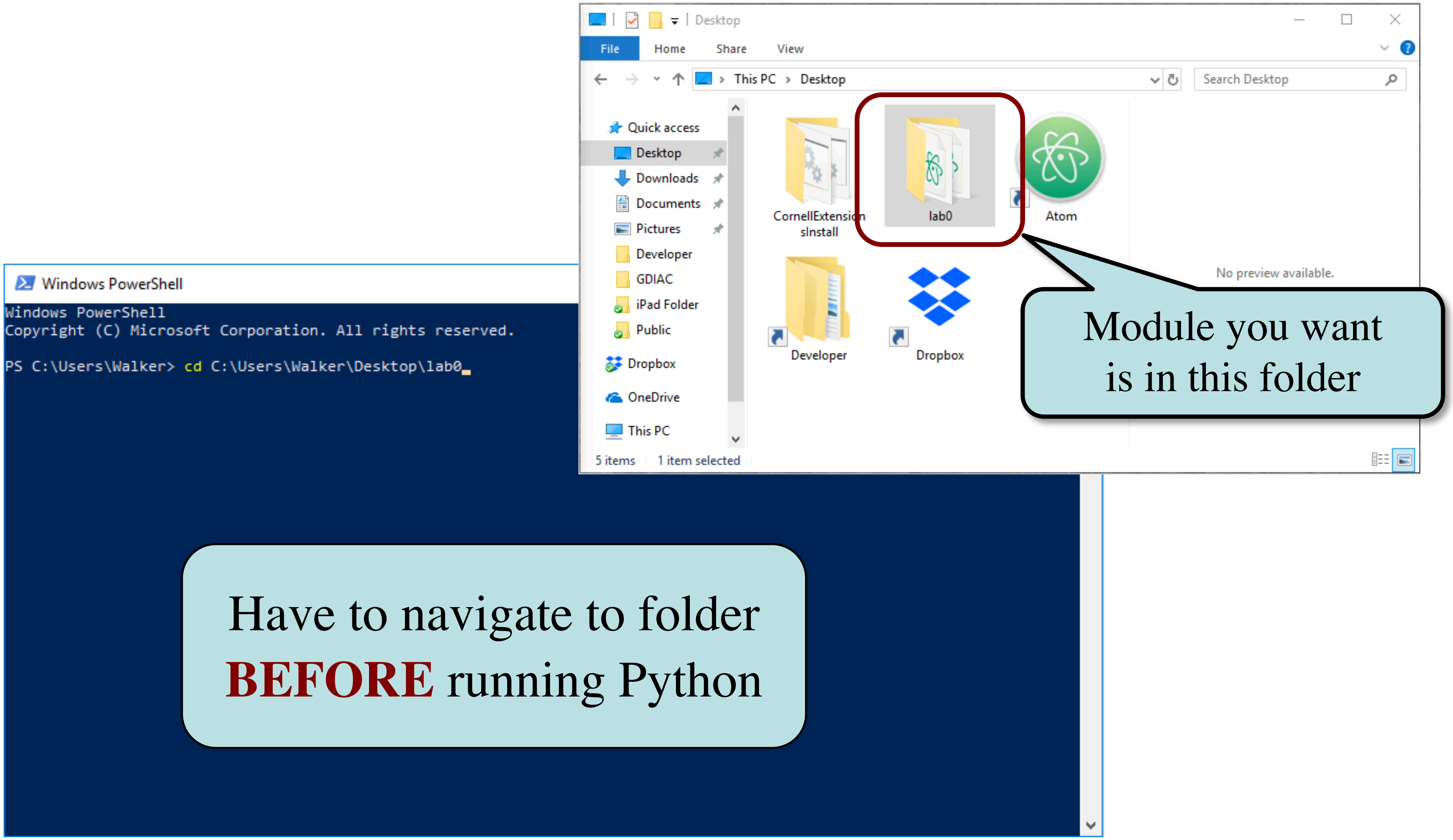
Python Shell

```
>>> import module
>>> x
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
>>> module.x
9
>>> help(module)
```

Modules Must be in Active Directory!



Modules Must be in Active Directory!



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\Walker> cd C:\Users\Walker\Desktop\lab0_
```

Module you want is in this folder

Have to navigate to folder **BEFORE** running Python

Modules vs. Scripts

Module

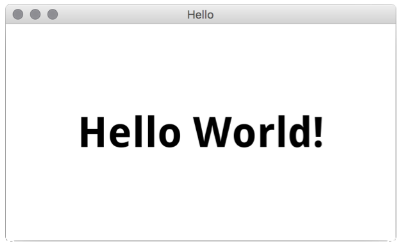
- Provides functions, variables
  - **Example**: temp.py
- import it into Python shell

```
>>> 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

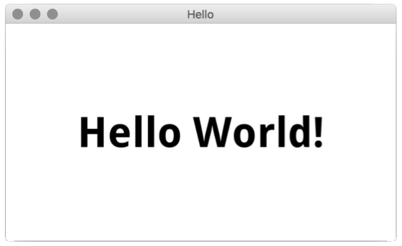
- Provides functions, variables
  - **Example**: temp.py
- import it into Python shell

```
>>> 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

module.py

```python
""" A simple module.

This file shows how modules work
"""

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

script.py

```python
""" 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
```

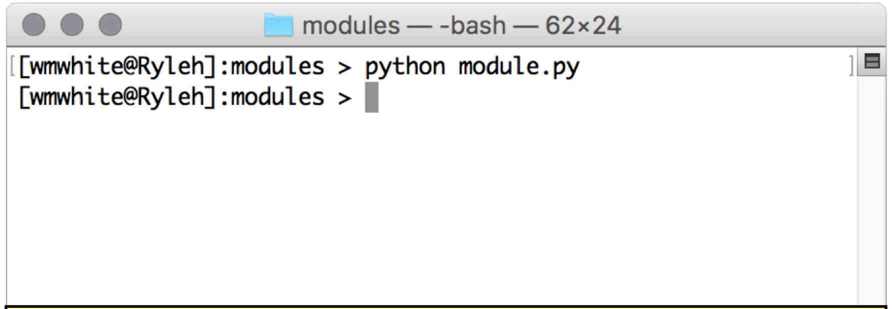
script.py

```
""" A simple script.

This file shows why we use print
"""

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

Only difference

Scripts and Print Statements

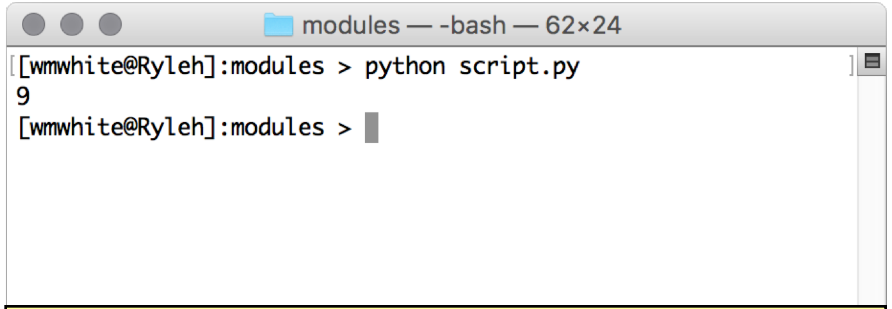
module.py



```
[wmwhite@Ryleh]:modules > python module.py
[wmwhite@Ryleh]:modules > 
```

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

script.py

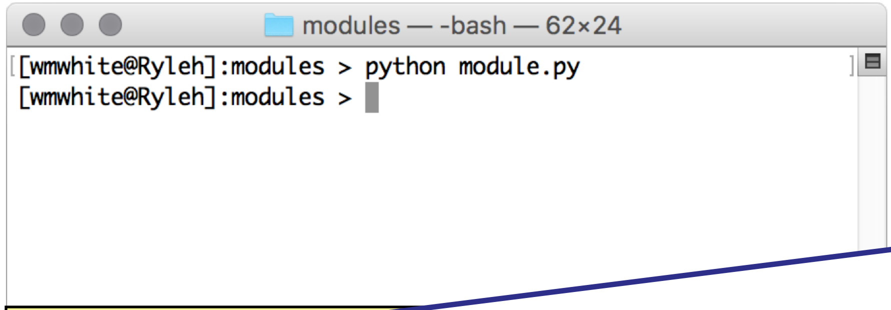


```
[wmwhite@Ryleh]:modules > python script.py
9
[wmwhite@Ryleh]:modules > 
```

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

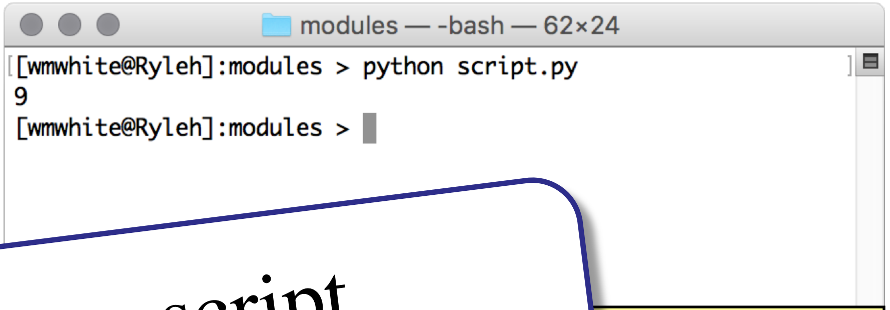
Scripts and Print Statements

module.py



```
[wmwhite@Ryleh]:modules > python module.py
[wmwhite@Ryleh]:modules >
```

script.py



```
[wmwhite@Ryleh]:modules > python script.py
9
[wmwhite@Ryleh]:modules >
```

- Looks l
- Python
  - Execu assignments
  - Skipped the last line
    (‘x’ is not a statement)

his time!

he following:

- Executed the assignments
- Executed the last line
  (Prints the contents of x)

When you run a script,
only statements are executed

User Input

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

No space after the prompt.

Proper space after prompt.

Assign result to variable.

Making a Script Interactive

```python
"""
A script showing off input.

This file shows how to make a script
interactive.
"""

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
[wmw2] folder>
```

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

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

Value is a string.

Must convert to int.