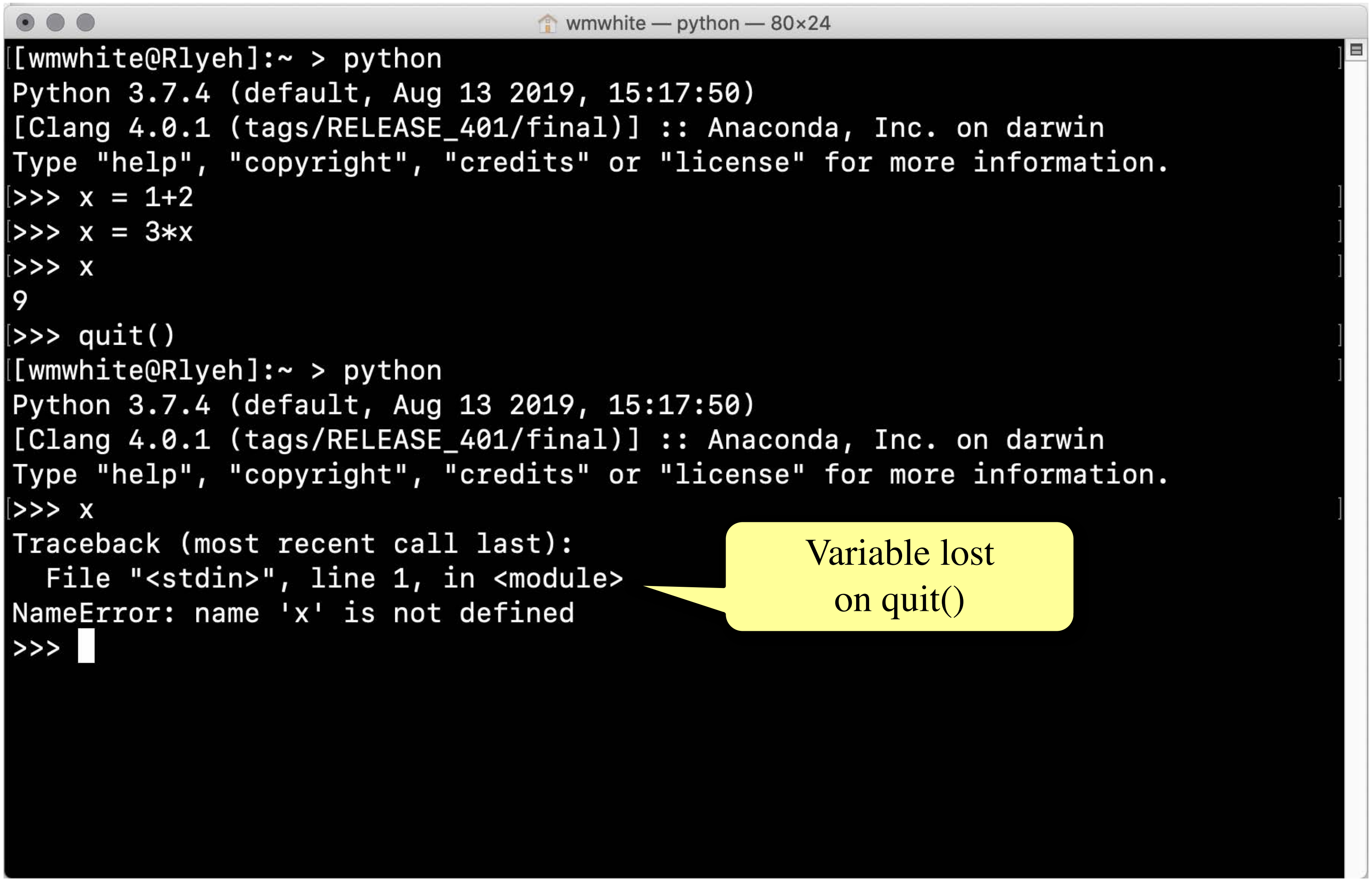
Module 4

Scripts and Modules

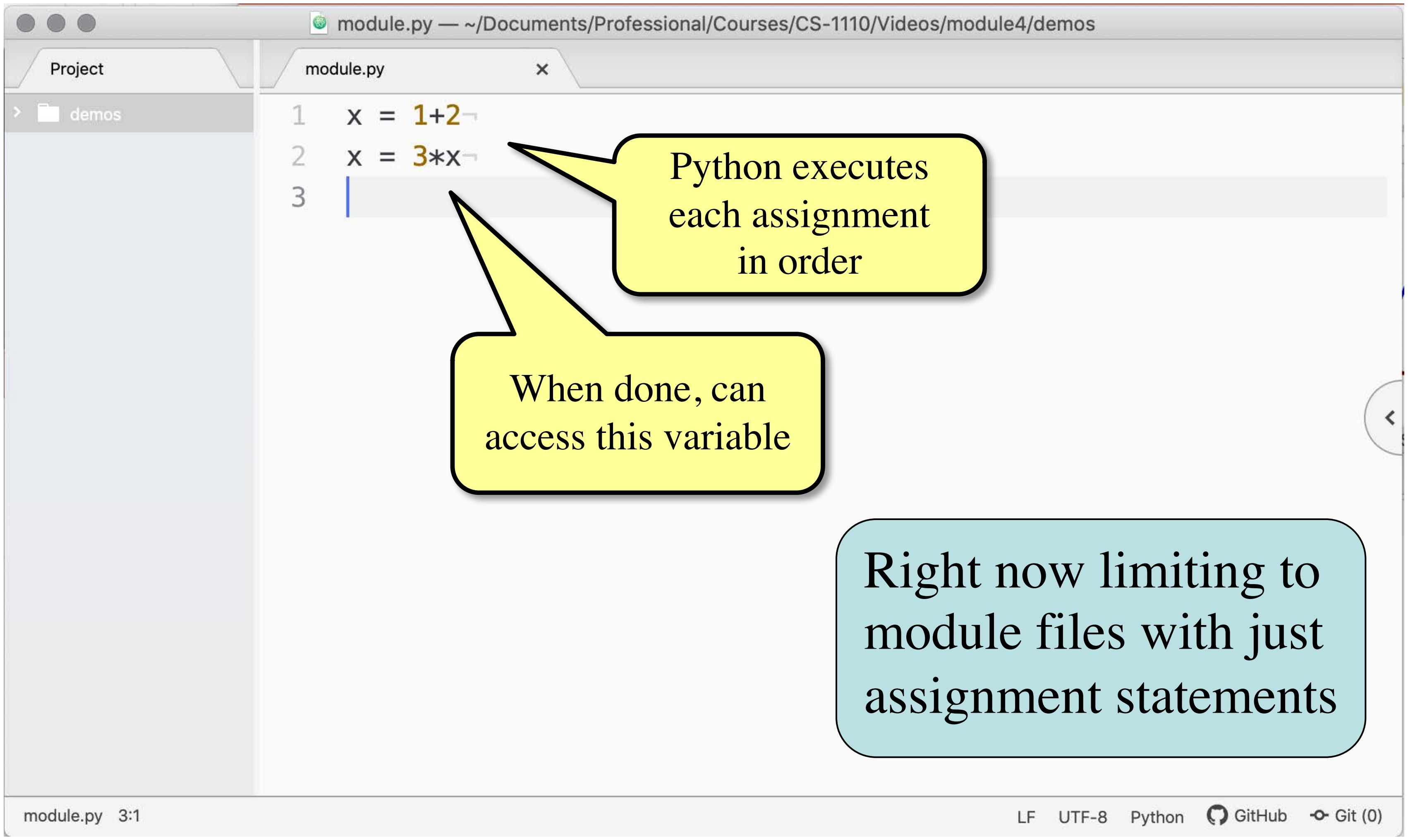
Limitations of the Interactive Shell



```
[wmwhite@Rlyeh]:~ > python
Python 3.7.4 (default, Aug 13 2019, 15:17:50)
[Clang 4.0.1 (tags/RELEASE_401/final)] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> x = 1+2
>>> x = 3*x
>>> x
9
>>> quit()
[wmwhite@Rlyeh]:~ > python
Python 3.7.4 (default, Aug 13 2019, 15:17:50)
[Clang 4.0.1 (tags/RELEASE_401/final)] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> x
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
>>> 
```

Variable lost on quit()

Solution: Use a (Module) File



module.py — ~/Documents/Professional/Courses/CS-1110/Videos/module4/demos
Project
module.py
demos
module.py 3:1
LF UTF-8 Python GitHub Git (0)

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

Python executes each assignment in order

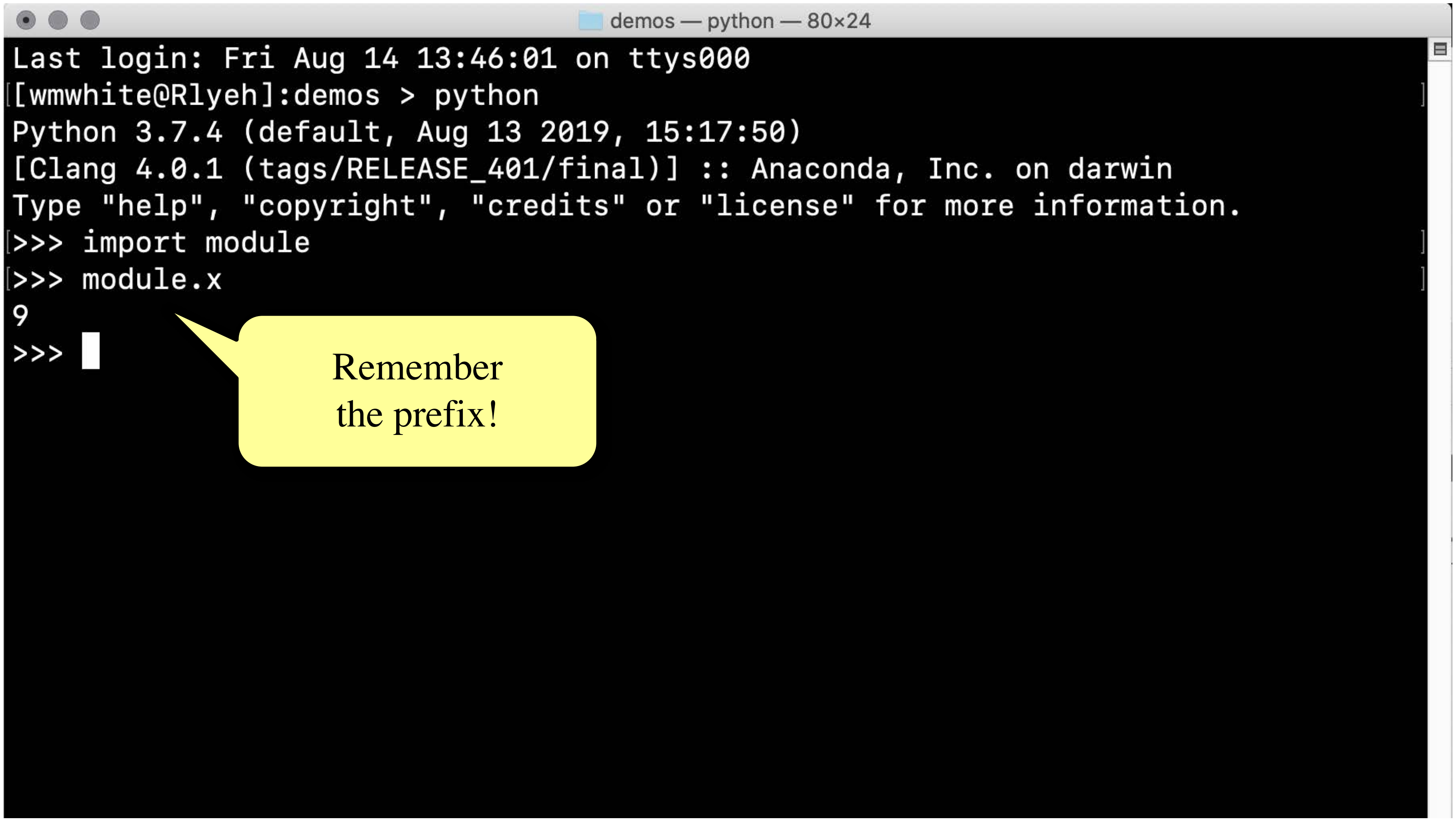
When done, can access this variable

Right now limiting to module files with just assignment statements

How Do We Use This File?

- Remember scripts in Module 0!
  - Navigate to a folder; `python module.py`
  - But does not do anything (it is not a script)
- But you can use it as a module
  - Navigate to folder; `python`; `import module`
  - Notice that we do not put the .py on the end
- Importing a module…
  - Executes all of the statements inside
  - Allows us to access the variables assigned

Putting It All Together



```
Last login: Fri Aug 14 13:46:01 on ttys000
[wmwhite@Rlyeh]:demos > python
Python 3.7.4 (default, Aug 13 2019, 15:17:50)
[Clang 4.0.1 (tags/RELEASE_401/final)] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import module
>>> module.x
9
>>> 
```

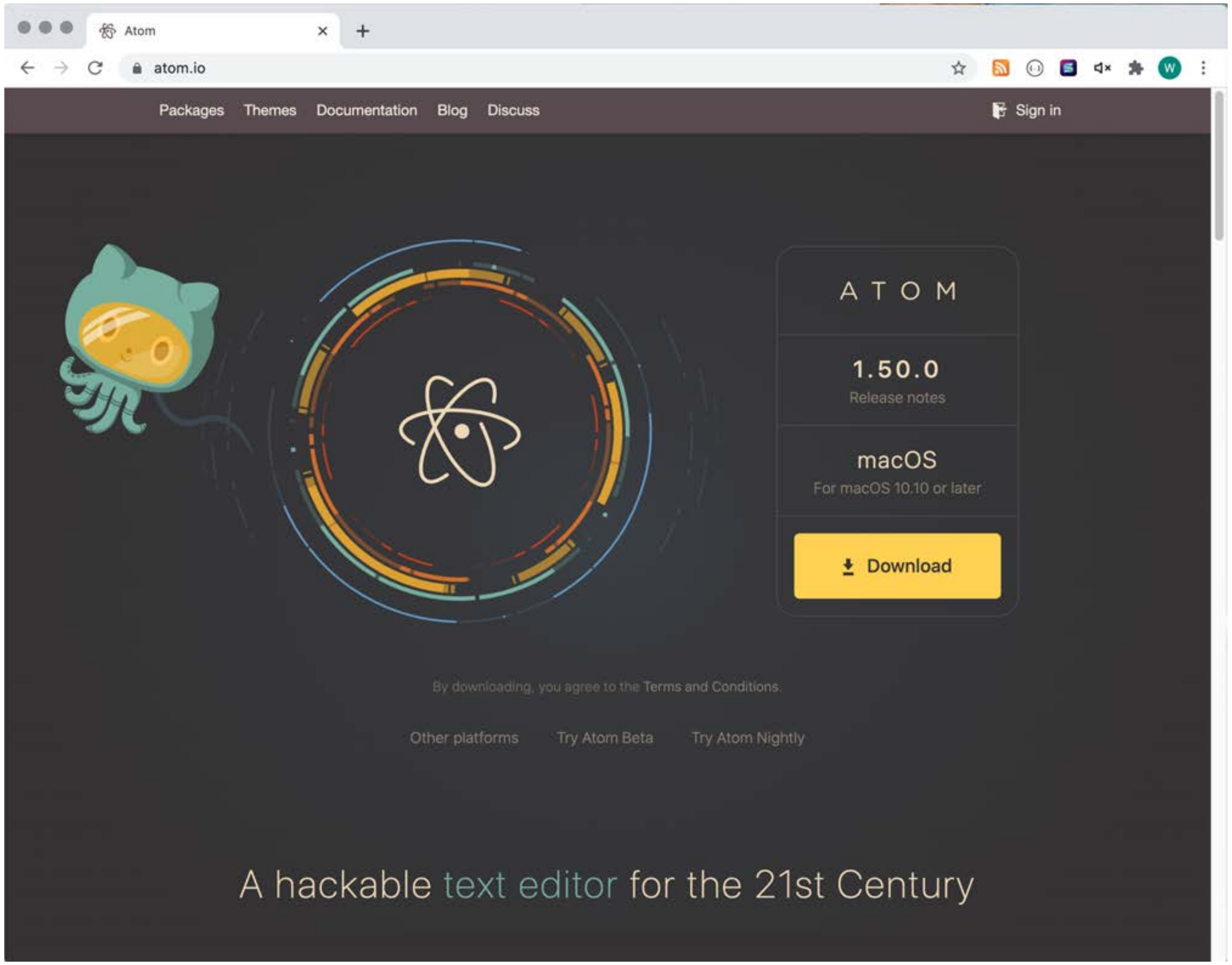
Purpose of Code Editors

- You will need something to edit code files
- How about Microsoft Word?
  - Do not want fonts or formatting
  - Just want to edit plain text
- How about NotePad (W) or TextEdit (M)?
  - Better (and some people use them), but not ideal
- Want something that can help you code
  - Designed to help you look for code mistakes
  - Special purpose program is a **Code Editor**

Using a Code Editor

- Code Editor is a program to edit code
  - Not limited to Python; supports many langs
  - Can do (some) error checking for you
  - Colors text in ways we talk about later
- There are many popular code editors
  - Two most popular: Atom Editor, VS Code
  - We prefer Atom Editor
    - Best python support out of box
    - (Almost) the same on all computers

Atom Editor

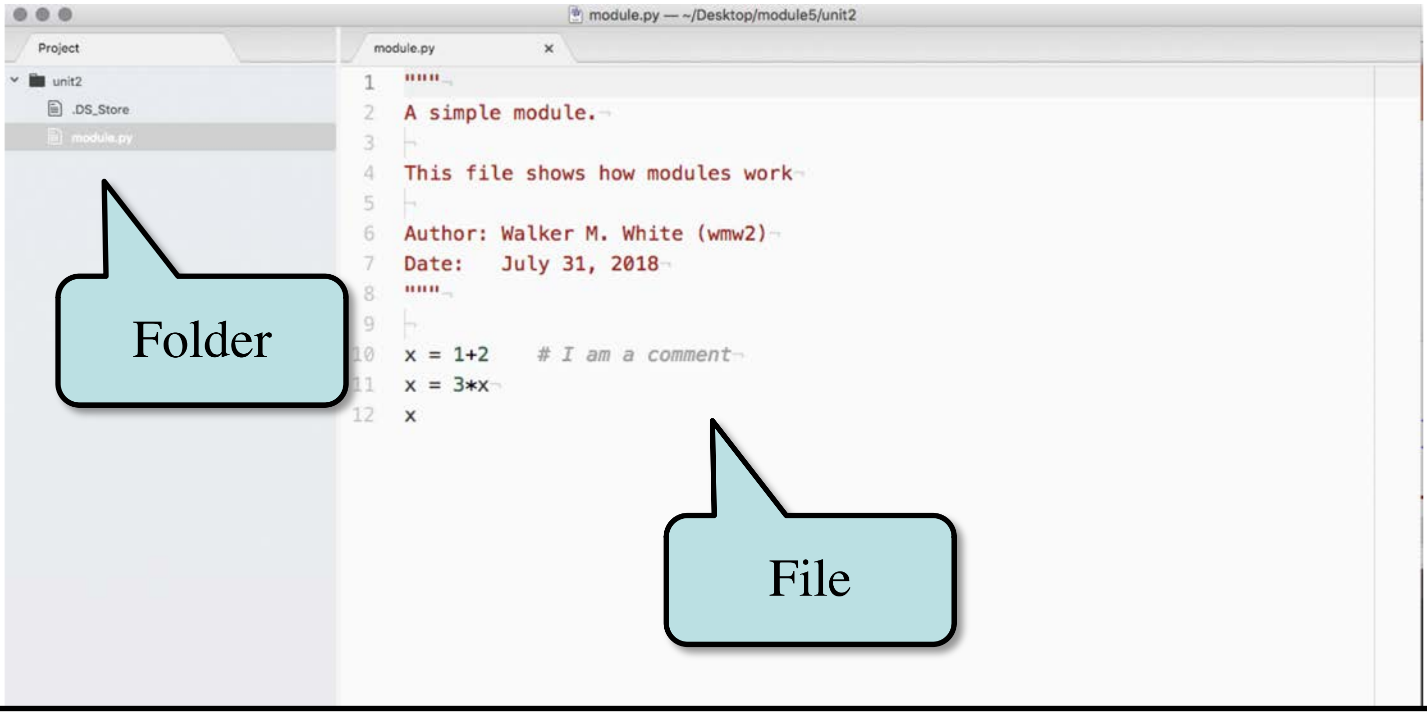


Atom
atom.io
Packages
Themes
Documentation
Blog
Discuss
Sign in
ATOM
1.50.0
Release notes
macOS
For macOS 10.10 or later
Download
By downloading, you agree to the Terms and Conditions.
Other platforms
Try Atom Beta
Try Atom Nightly
A hackable text editor for the 21st Century

Getting Started with Atom

- Double click on Atom Editor
  - You will see a lot of windows
  - Can close the tabs by clicking at the top
- Can open a file in two ways
  - Select Open from the menu on computer
  - Drag and drop on to the application icon
- When you open, folder to the left
  - Lists all of files in folder
  - Can click on any to open

File Organization



```
"""
A simple module.

This file shows how modules work

Author: Walker M. White (wmw2)
Date:   July 31, 2018
"""

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

- This is a natural way to program
  - We organize related Python files in folders
  - Can also open the whole folder, not file

Final Word on Workflow

- Python programmers have two windows open
  - The Code Editor
  - Terminal
  - Often like them side by side
  - Do not recommend different desktops
  - Swiping back and forth can get confusing
- Often will have a third window open
  - The browser or the documentation
  - This one is okay in a different desktop

The Basic Elements

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**

The Basic Elements

Module Contents

```
""" A simple module.

This file shows how modules work
"""

# This is

x = 1+2

x = 3*x

x
```

Python Shell

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

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

Prints **docstring** and module contents

Modules Can Import Modules

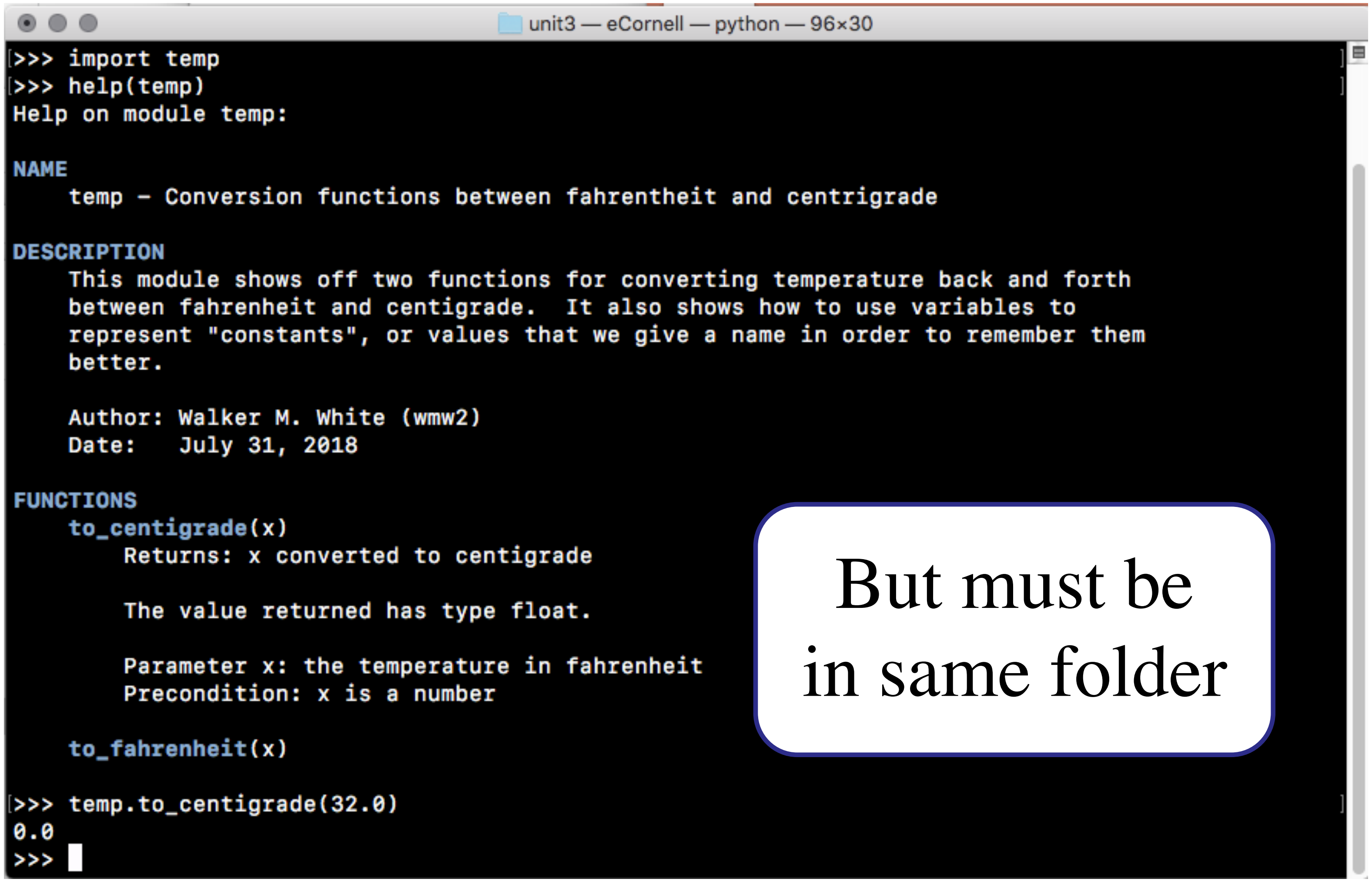
```
"""
A module that imports another module.
"""
# Import a standard python module
import math
x = math.cos(0)

# Import a user-defined module
import temp
y = temp.to_centigrade(32.0)
```

Standard Module

User-Defined Module

Can Use **temp** w/o Understanding It



```
>>> import temp
>>> help(temp)
Help on module temp:

NAME
    temp - Conversion functions between fahrentheit and centrigrade

DESCRIPTION
    This module shows off two functions for converting temperature back and forth
    between fahrenheit and centigrade.  It also shows how to use variables to
    represent "constants", or values that we give a name in order to remember them
    better.

    Author: Walker M. White (wmw2)
    Date:   July 31, 2018

FUNCTIONS
    to_centigrade(x)
        Returns: x converted to centigrade

        The value returned has type float.

        Parameter x: the temperature in fahrenheit
        Precondition: x is a number

    to_fahrenheit(x)

>>> temp.to_centigrade(32.0)
0.0
>>> 
```

But must be in same folder

Recall: Scripts

- Script is a file containing Python code
  - Ends with the suffix .py
  - Run it by typing: `python <script>`
  - Gave you several examples at course start
- But modules contain Python code too!
  - Are they also scripts?
  - What is the difference between them?

Understanding the Difference

Module

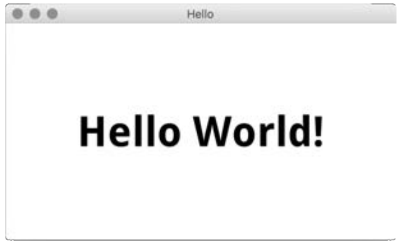
- Provides functions, variables
  - **Example**: temp.py
- import it into Python shell

```
>>> import temp
>>> temp.to_fahrenheit(100)
212.0
>>>
```

Script

- Behaves like an application
  - **Example**: hello_app.py
- Run it from command line:

```
python hello_kivy.py
```



Files are 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
```

Only difference

script.py

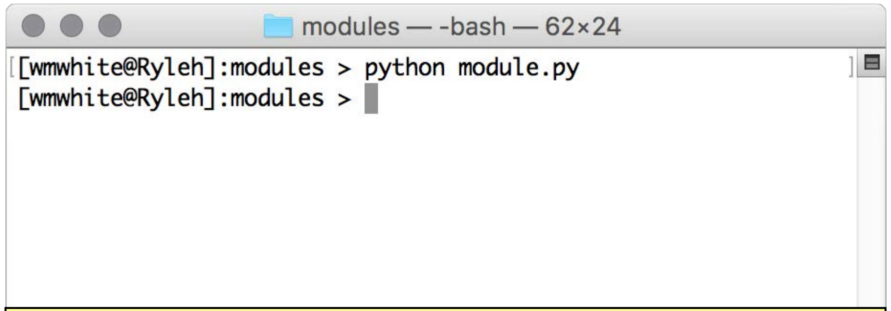
```
""" A simple script.

This file shows why we use print
"""

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

Scripts and Print Statements

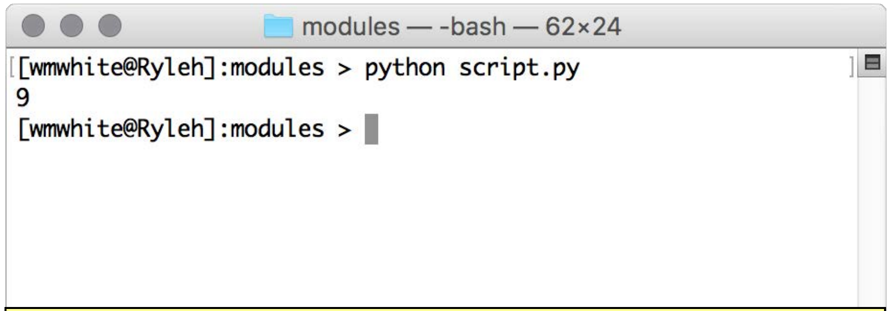
module.py



```
[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

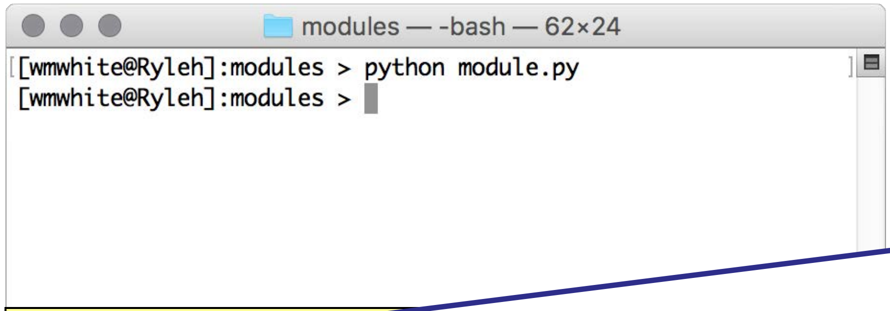


```
[wmwhite@Ryleh]:modules > python script.py
9
[wmwhite@Ryleh]:modules > 
```

- 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

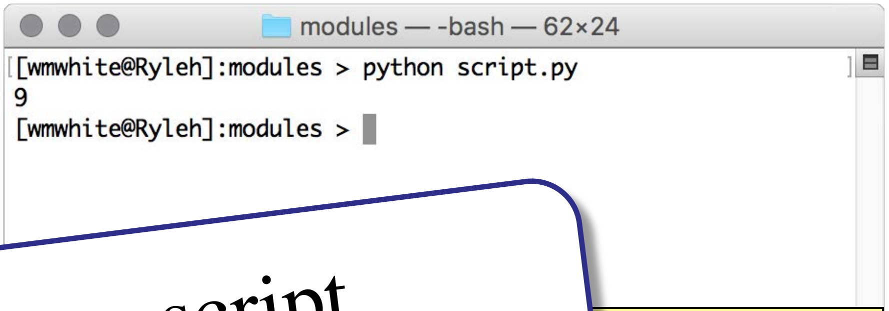
module.py



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

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

script.py



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

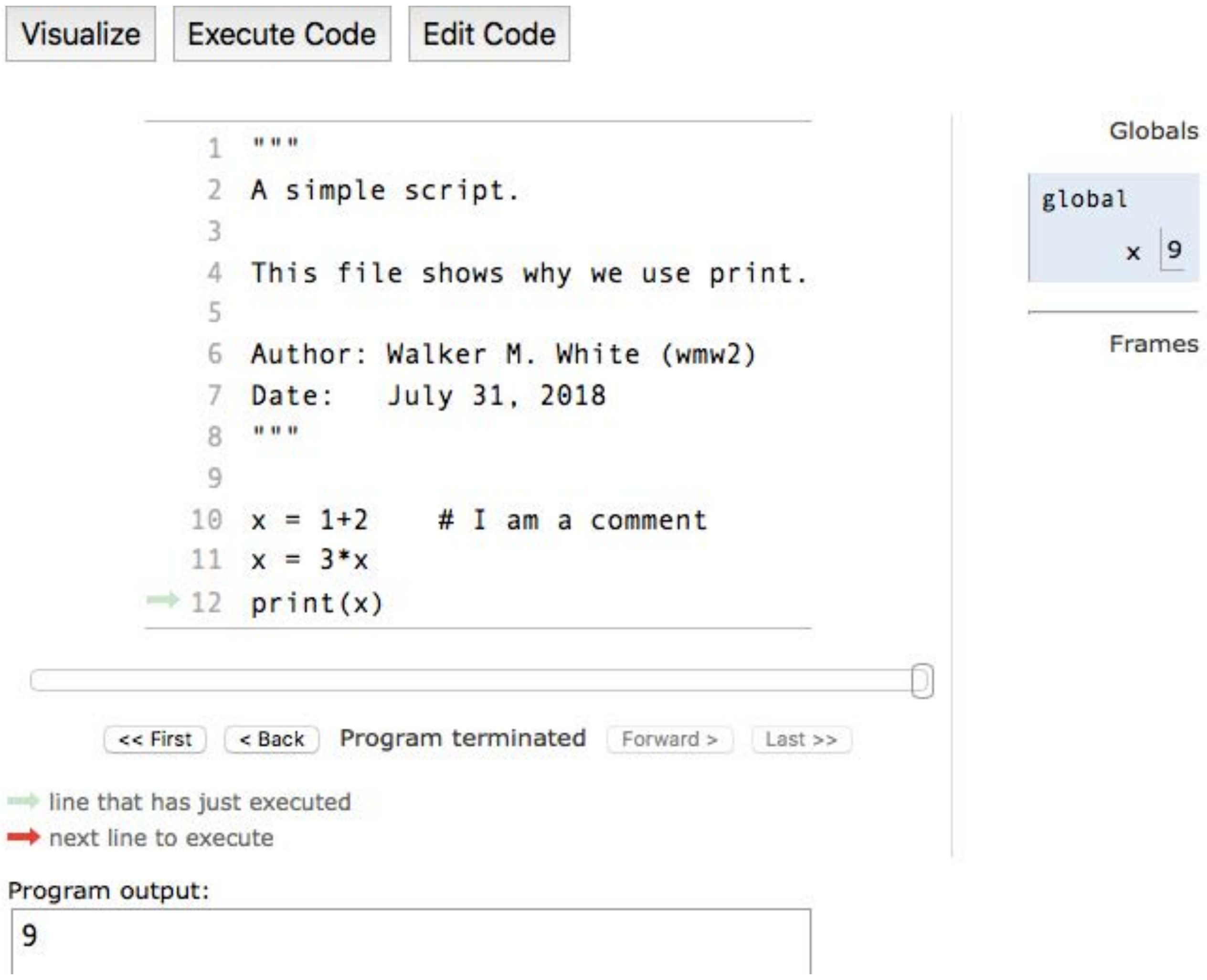
- his time!
- the following:
  - Executed the assignments
  - Executed the last line
    (Prints the contents of `x`)

When you run a script,
only statements are executed

The Problem Working with Scripts

- When scripts run we do not see a lot
  - We see any print statements they make
  - But we cannot see any of the variables
  - Or any of the function calls
- This is can make it hard to find bugs
  - Particularly for the project you are working on
  - If something wrong, cannot see it
- Once again, an argument for **visualization**

Visualizing Scripts: The Python Tutor



```
"""
A simple script.

This file shows why we use print.

Author: Walker M. White (wmw2)
Date:   July 31, 2018
"""

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

global

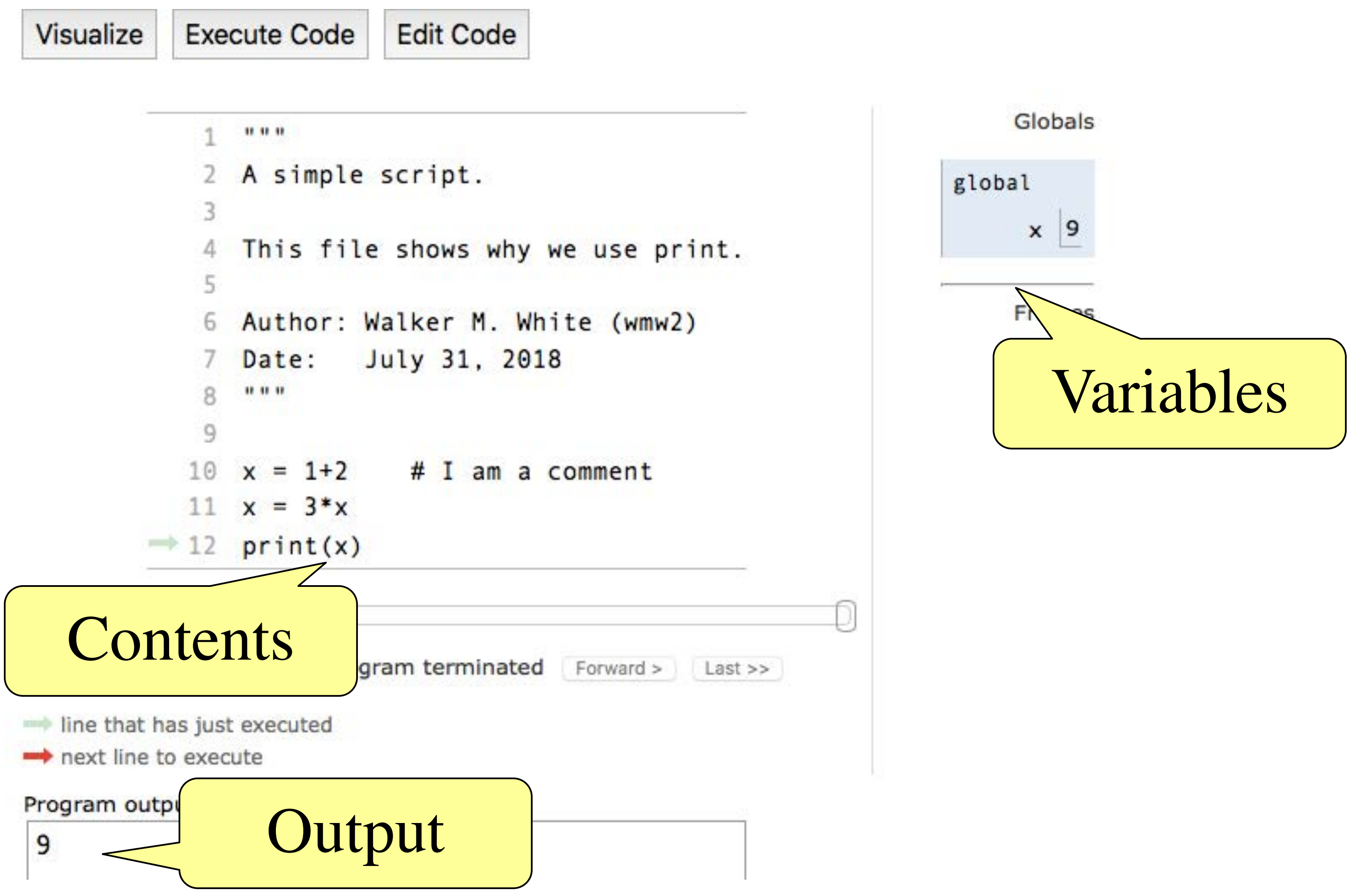
x	9

➡ line that has just executed
➡ next line to execute

Program output:

```
9
```

Visualizing Scripts: The Python Tutor



```
"""
A simple script.

This file shows why we use print.

Author: Walker M. White (wmw2)
Date:   July 31, 2018
"""

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

```
9
```

The Problem Statement

- Right now, our scripts are not very interesting
  - We can introduce randomness, but still limited
- Typical programs interact with the user
  - The user gives input (mouse, typing)
  - Program does something different in response
- **Recall:** we do that with input(msg)

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

Evaluates to what is typed

Numeric Input

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

Convert it to the type you want