Lecture 14:

FUNCTION BASICS AND THE `main()` FUNCTION

CSC111: Introduction to CS through Programming

R. Jordan Crouser

Assistant Professor of Computer Science

Smith College

About Assignment 3

- 100 / 102 assignments submitted on time (!!!)
- Almost everyone included a header and had appropriate variable names (nice work)
- Functionality was great! (esp. ❤️ed your creativity)
- A few notes on **attribution:**

A reminder from the syllabus

- **Attribution**:
  - The names of all collaborating students should be listed at the top of the submission. [Note: include help from TAs here]
  - If you worked alone, please state: "*I did not collaborate with anyone on this assignment.*"
- "**References**"
  - Citations to any resources you used, including page numbers (if a printed resource) or a direct URL (if an online resource).
  - If you did not use any resources in completing the assignment, please state: "*I did not utilize any external resources in completing this assignment.*"
- Approximately 74% of A3 submissions had these

Here's what I want to see

*documentations.py - /Users/jcrouser/Google Drive/Teaching/Course Material/CSC111/CSC111/demos/documentations....

```
#-----------------------------------------------
#       Names: Jordan Crouser & Morganne Crouser
#        Date: 26 September 2018
#    Filename: demo.py
# Description: This is a demonstration of how to
#              propertly attribute help on a
#              CSC111 assignment
#-----------------------------------------------

name = input("Enter your name: ")
formatted_string = "{0:>10}".format(name)
print(formatted_string)

# REFERENCES
# I googled how to use the str.format(...) method
# and found the Python documentation here:
# https://docs.python.org/3/library/stdtypes.html#str.format
```

Ln: 17 Col: 60

Here's what I want to see

*documentations.py - /Users/jcrouser/Google Drive/Teaching/Course Material/CSC111/CSC111/demos/documentations....

```
#-----------------------------------------------
#       Names: Jordan Crouser & Morganne Crouser
#        Date: 26 September 2018
#    Filename: demo.py
# Description: This is a demonstration of how to
#              propertly attribute help on a
#              CSC111 assignment
#-----------------------------------------------

name = input("Enter your name: ")
formatted_string = "{0:>10}".format(name)
print(formatted_string)

# REFERENCES
# I googled how to use the str.format(...) method
# and found the Python documentation here:
# https://docs.python.org/3/library/stdtypes.html#str.format
```

Ln: 17 Col: 60

Here's what I want to see

*documentations.py - /Users/jcrouser/Google Drive/Teaching/Course Material/CSC111/CSC111/demos/documentations....

```
#-----------------------------------------------
#       Names: Jordan Crouser & Morganne Crouser
#        Date: 26 September 2018
#    Filename: demo.py
# Description: This is a demonstration of how to
#              propertly attribute help on a
#              CSC111 assignment
#-----------------------------------------------

name = input("Enter your name: ")
formatted_string = "{0:>10}".format(name)
print(formatted_string)

# REFERENCES
# I googled how to use the str.format(...) method
# and found the Python documentation here:
# https://docs.python.org/3/library/stdtypes.html#str.format
```

Ln: 17 Col: 60

Here's what I want to see

documentations.py - /Users/jcrouser/Google Drive/Teaching/Course Material/CSC111/CSC111/demos/documentations....

```
#-----------------------------------------------
#       Names: Jordan Crouser (I did not collaborate
#              with anyone on this assignment)
#        Date: 26 September 2018
#    Filename: demo.py
# Description: This is a demonstration of how to
#              propertly attribute help on a
#              CSC111 assignment
#-----------------------------------------------

name = input("Enter your name: ")
formatted_string = "{0:>10}".format(name)
print(formatted_string)

# REFERENCES
# I did not utilize any external resources
# in completing this assignment.
```

Ln: 15 Col: 12

Here's what I want to see

documentations.py - /Users/jcrouser/Google Drive/Teaching/Course Material/CSC111/CSC111/demos/documentations....

```
#-----------------------------------------------
#       Names: Jordan Crouser (I did not collaborate
#              with anyone on this assignment)
#        Date: 26 September 2018
#    Filename: demo.py
# Description: This is a demonstration of how to
#              propertly attribute help on a
#              CSC111 assignment
#-----------------------------------------------

name = input("Enter your name: ")
formatted_string = "{0:>10}".format(name)
print(formatted_string)

# REFERENCES
# I did not utilize any external resources
# in completing this assignment.
```

Ln: 15 Col: 12

Outline

- Monday: first lesson in functions
- Lab: wrapping up collections (lists, dictionaries)
- Wednesday: more on functions
- Friday: data structures comparison

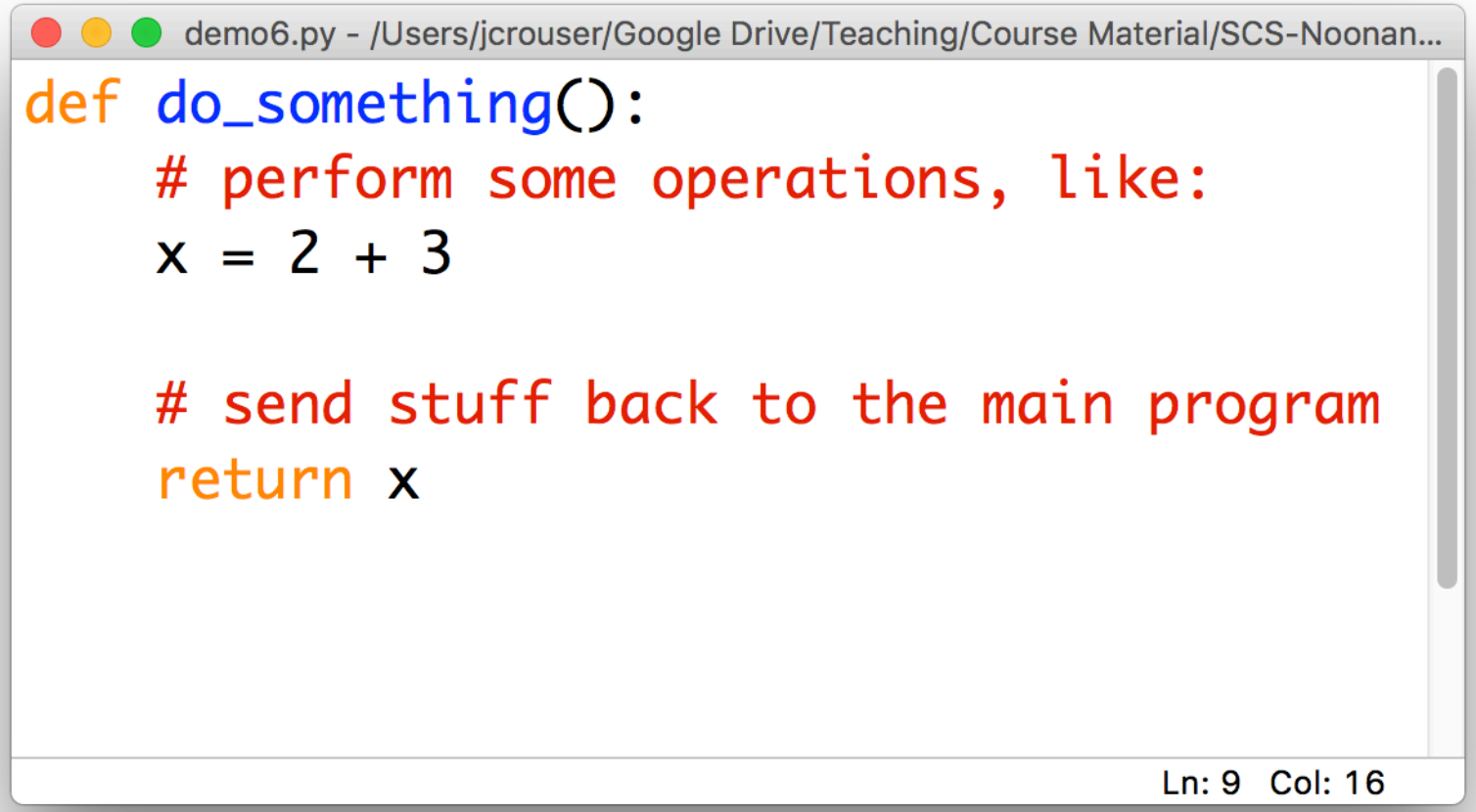
Functions

- **Recall**: a **function** is a procedure / routine that takes in some input and does something with it (just like in math)

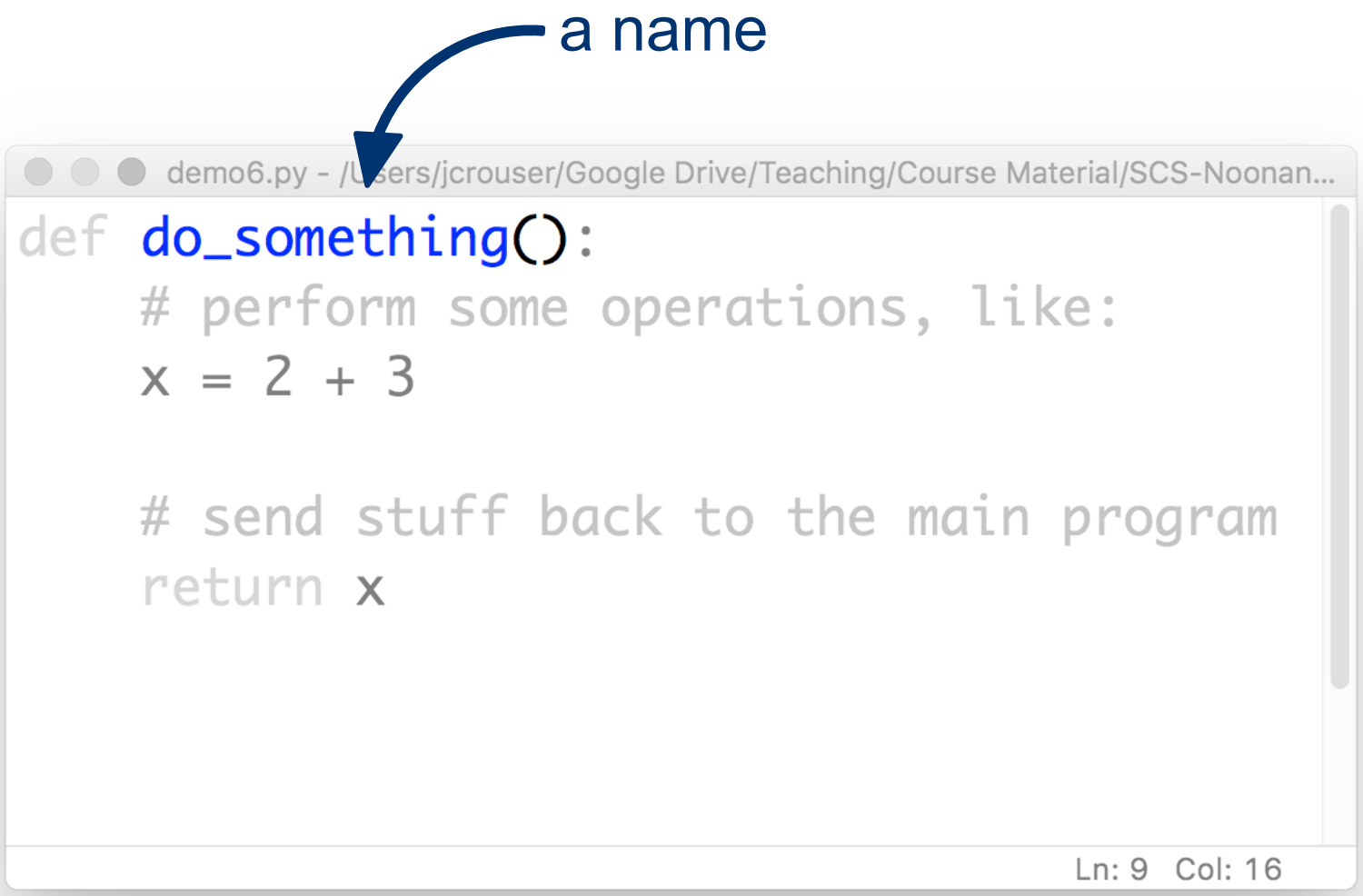
- We've seen lots of built-in functions:
  - `print(…)`
  - `input(…)`
  - `eval(…)`
  - `round(…)`

- Perhaps unsurprisingly, Python lets us write custom functions as well (like yesterday's `main()` function)

Basic components of a function



Basic components of a function



Convention: use **`_underscores_`** or **`camelCase`**

Basic components of a function

which is defined
using the **`def`** keyword

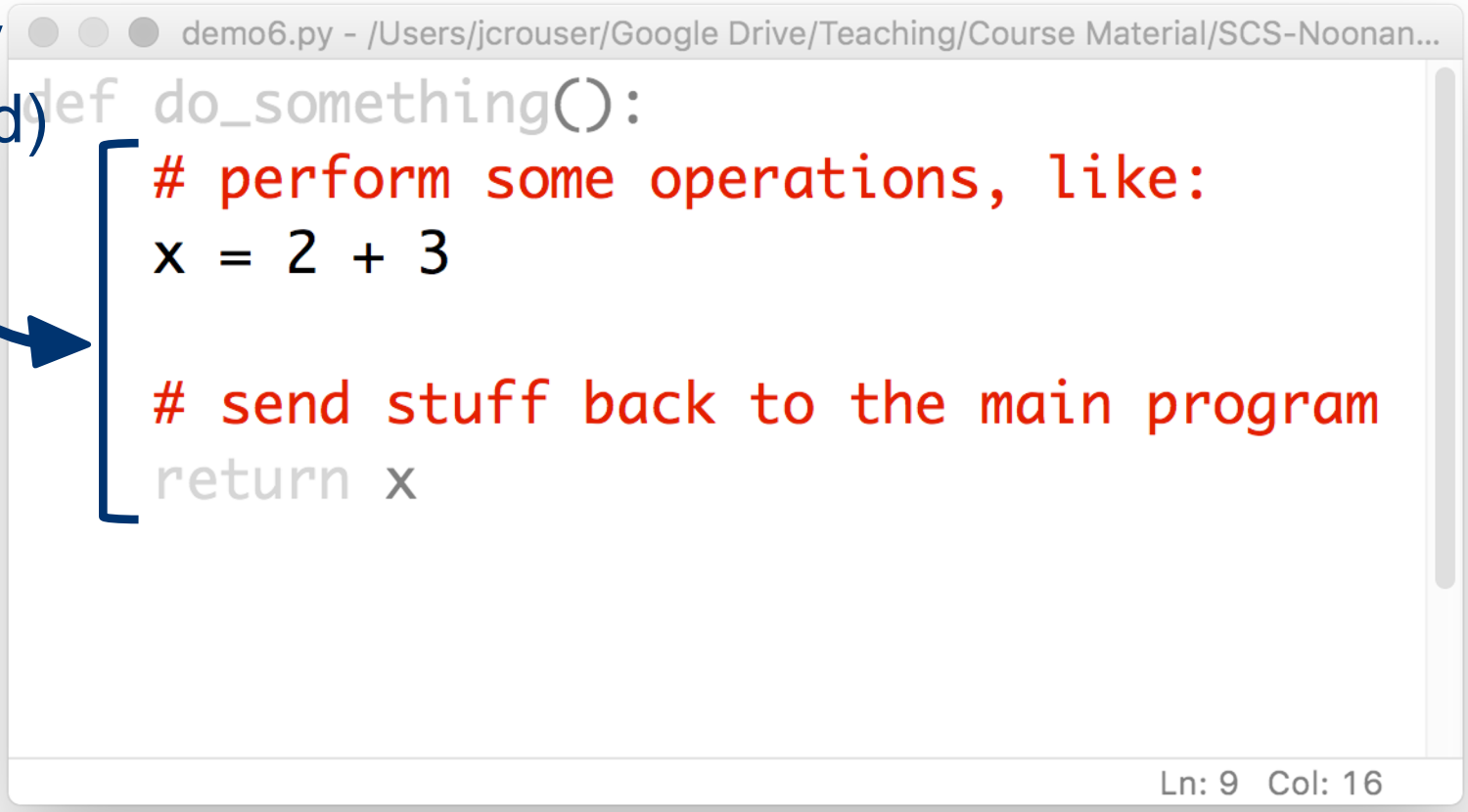
demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan...

```
def do_something():
    # perform some operations, like:
    x = 2 + 3

    # send stuff back to the main program
    return x
```

Ln: 9 Col: 16

Basic components of a function

a **body** (indented)



Basic components of a function

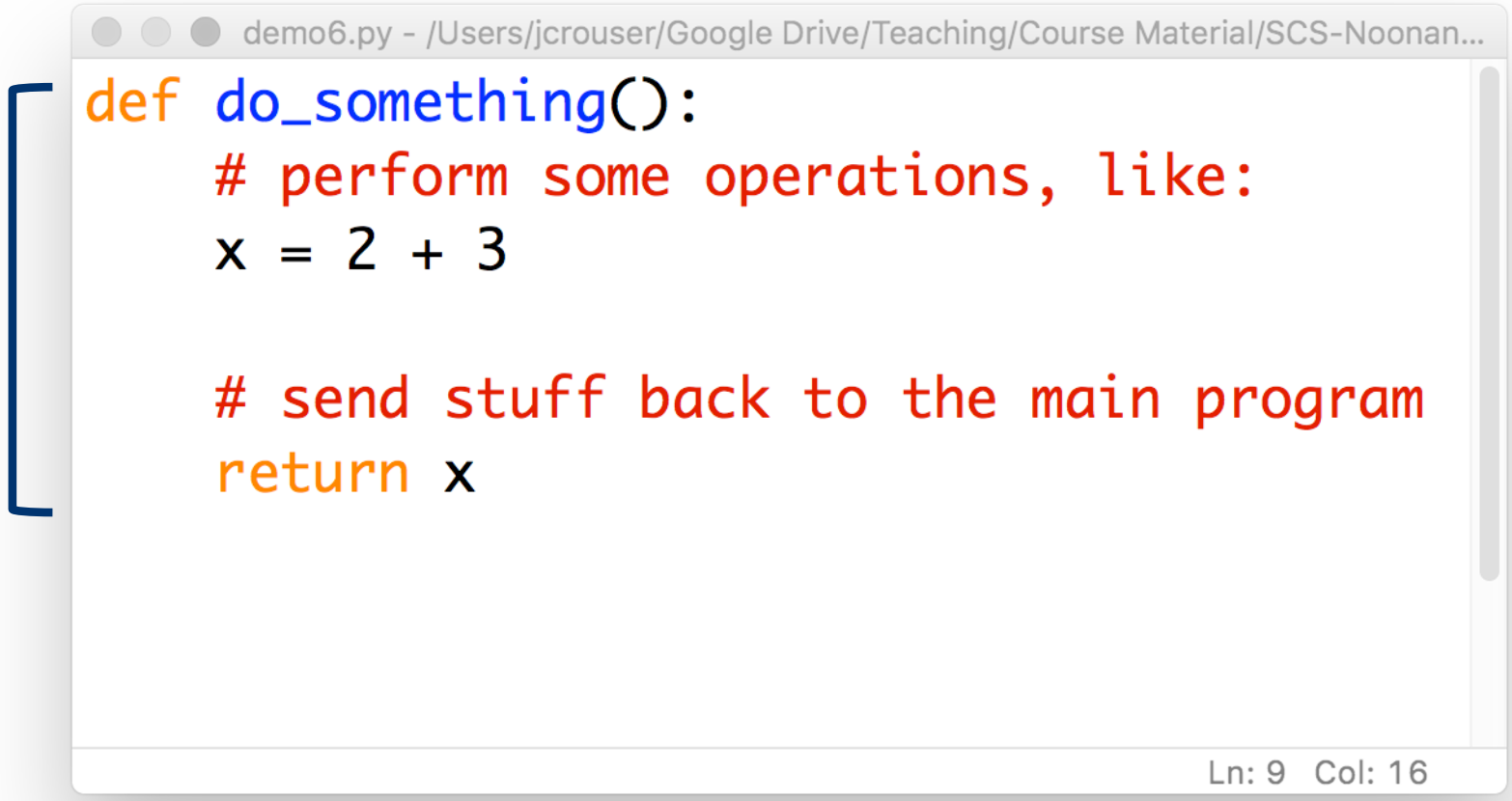
demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan...

```
def do_something():
    # perform some operations, like:
    x = 2 + 3

    # send stuff back to the main program
    return x
```

a **return** (optional)

Ln: 9 Col: 16

A “function definition”



Discussion

What happens if we **run** this program?

demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan...

```
def do_something():
    # perform some operations, like:
    x = 2 + 3

    # send stuff back to the main program
    return x
```

Col: 16

A “function definition” is a **description**

demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan...

```
def do_something():
    # perform some operations, like:
    x = 2 + 3

    # send stuff back to the main program
    return x
```

Ln: 9 Col: 16

(but not a **directive**)

Function calls: “hey, Python! do this”

demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan...

```
def do_something():
    # perform some operations, like:
    x = 2 + 3

    # send stuff back to the main program
    return x

y = do_something()
```

a function **call**

Ln: 9 Col: 16

Function calls: “hey, Python! do this”

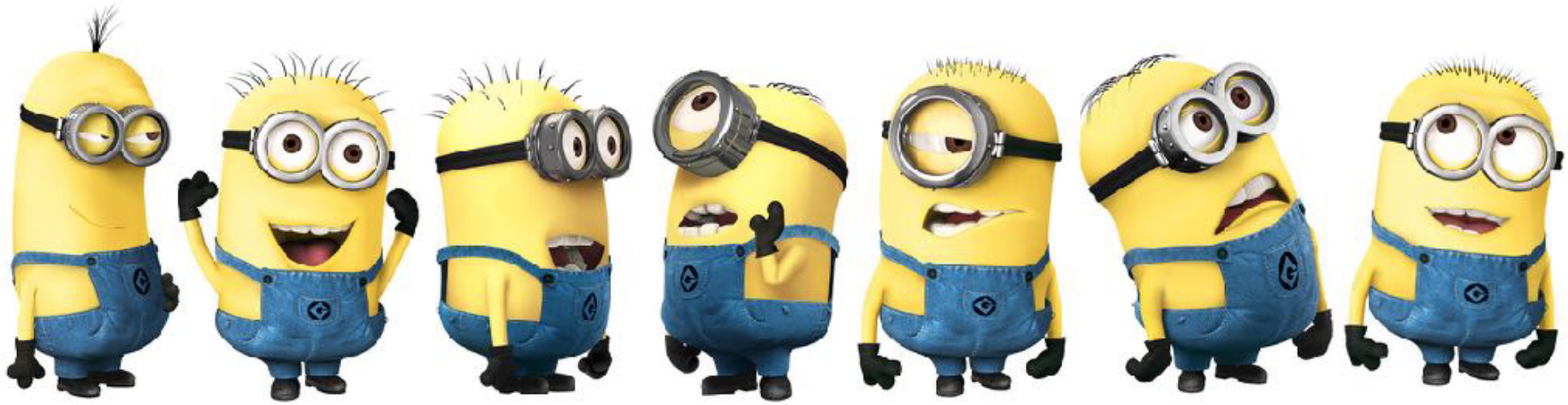
demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan...

```
def do_something():
    # perform some operations, like:
    x = 2 + 3

    # send stuff back to the main program
    return x

y =
```

5

Ln: 9 Col: 16

An analogy



`functions` are your **MINIONS**

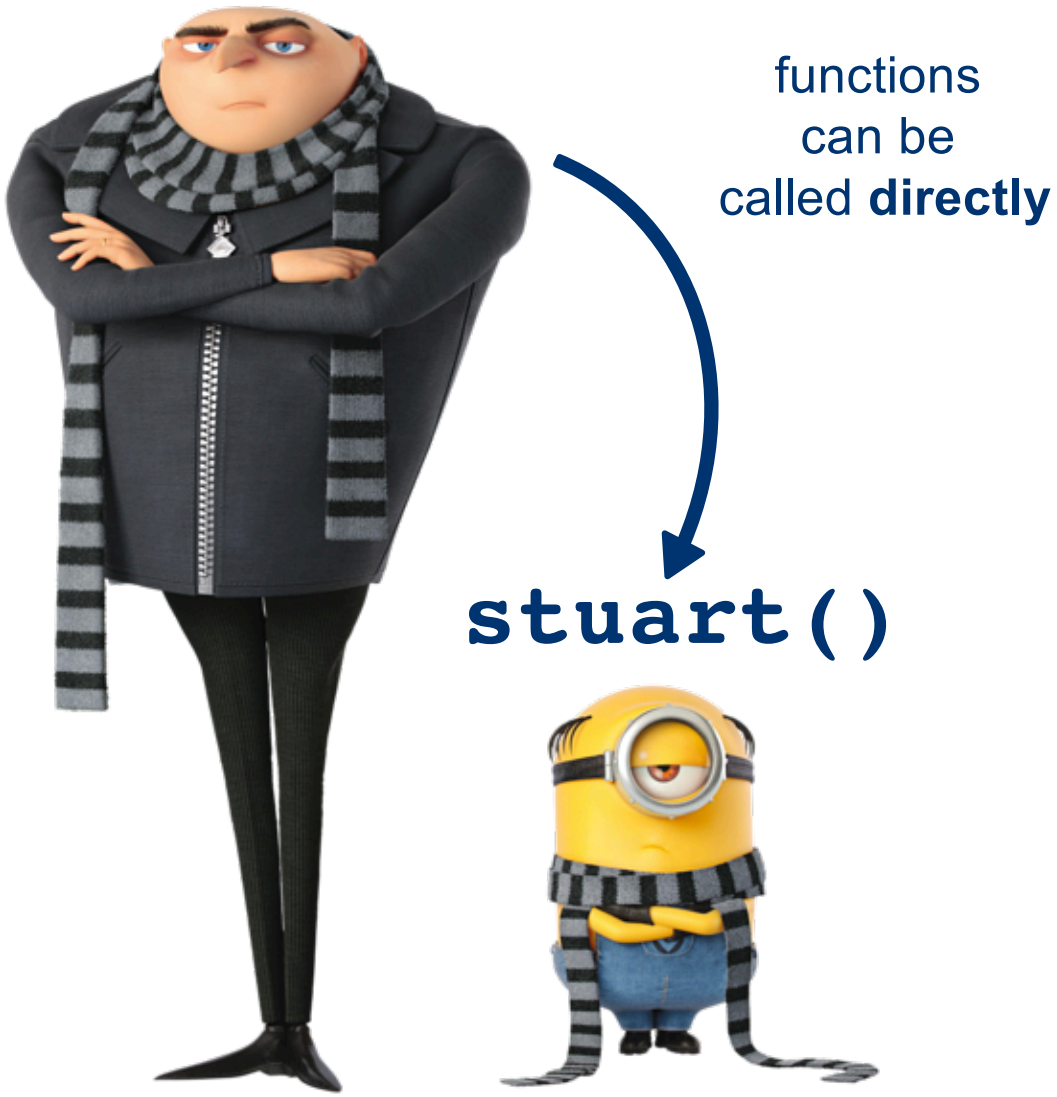
An analogy

`functions` have **NAMES**

An analogy

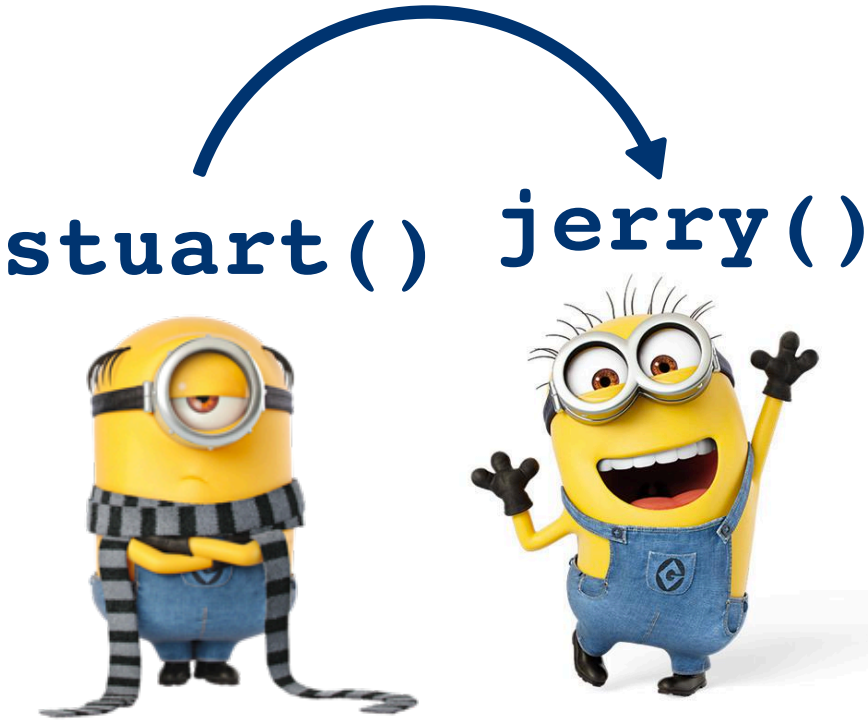
they only work when you **CALL** them

An analogy



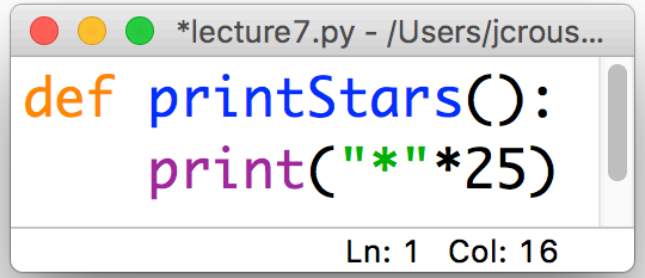
An analogy

functions
can also be
called by
one another



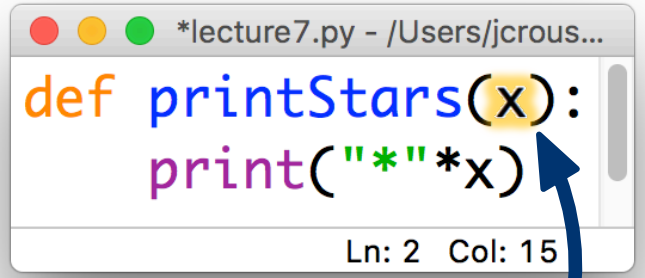
Two kinds of functions

Some functions always **do the same thing**




```
printStars()
printStars()
printStars()
```

Others adjust their behavior based on **what we give them**



"parameter"

```
printStars(5)
printStars(32)
printStars(1527)
```

Quick demo: Happy Birthday

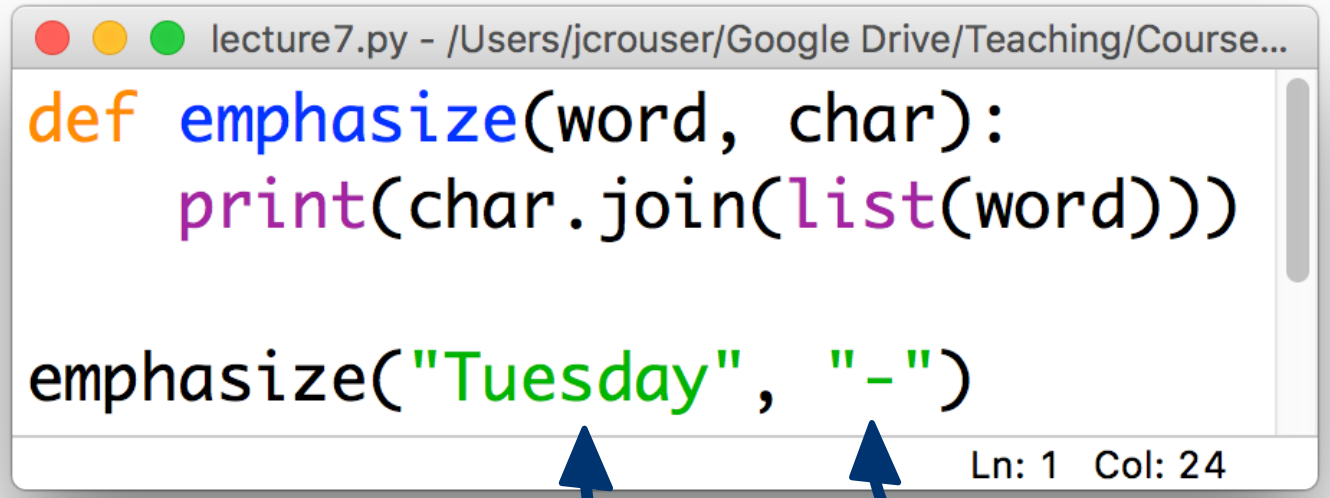
- Write a function called **`happyBirthday(name)`** that takes in a string **`name`** and prints out the lyrics to the song ”Happy Birthday” with the name inserted:

```
Happy birthday to you!
Happy birthday to you!
Happy birthday, dear name
Happy birthday to you!
```

- use **`input(…)`** to get the value of **`name`** from the user , and then call the **`happyBirthday(…)`** function to print a customized happy birthday song

Parameters

- Functions can be defined to take take several parameters:

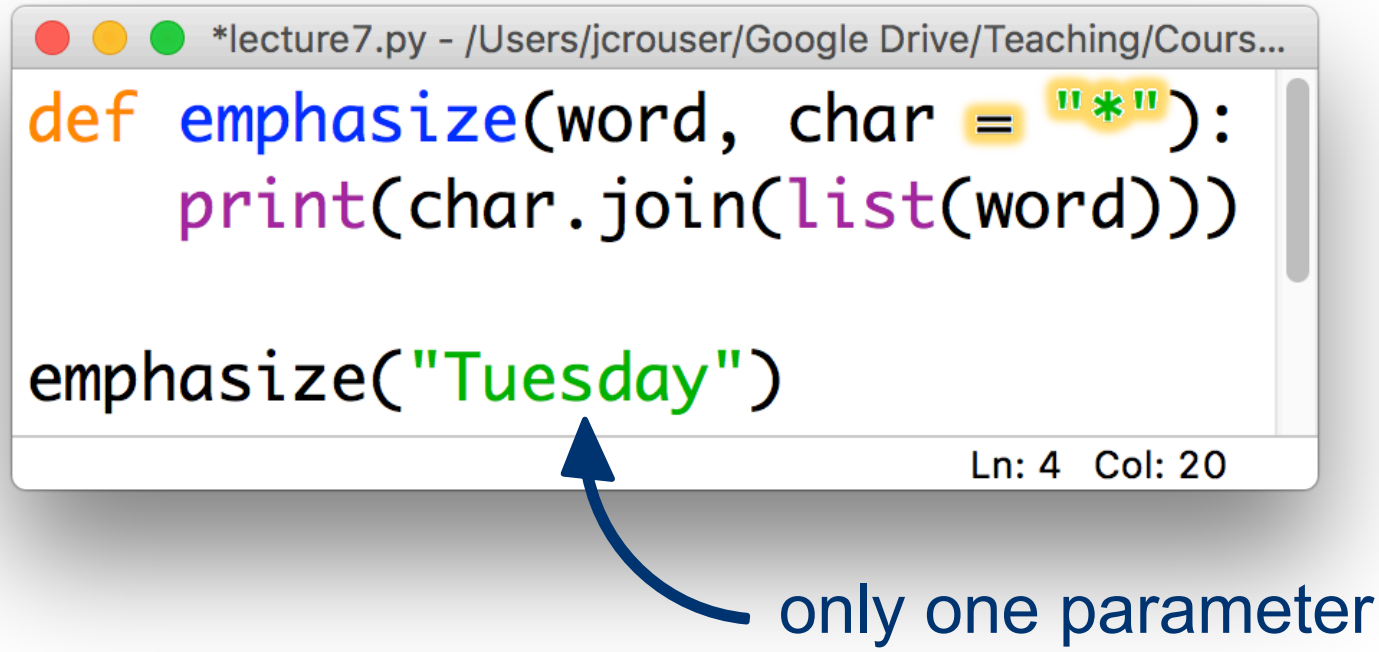


- **Result:**

```
T-u-e-s-d-a-y
```

Default parameters

- We can include a "default" value for some (or all) of them:

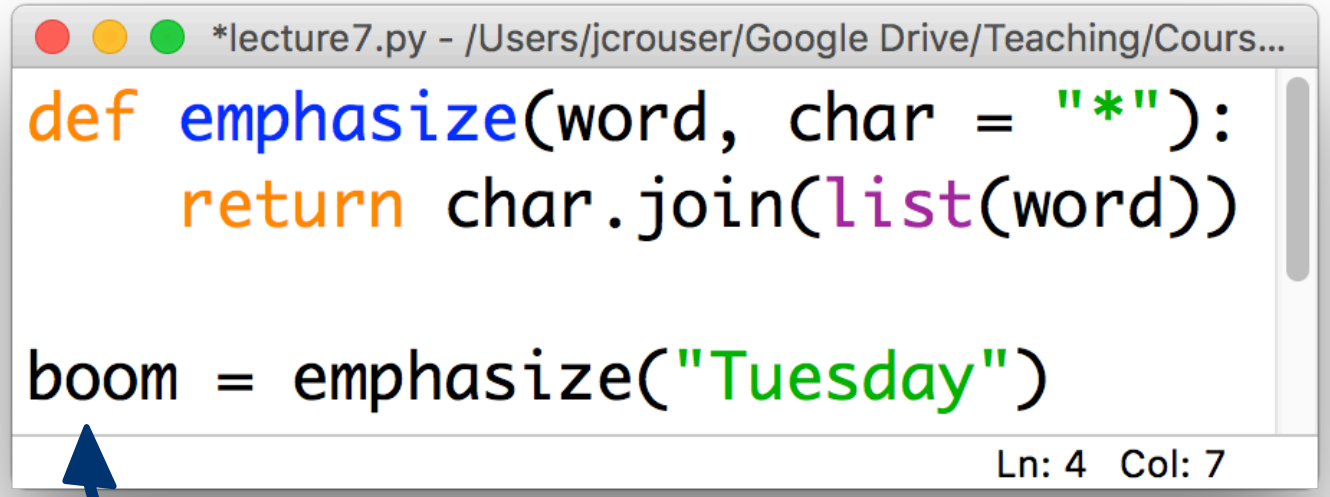


- **Result:**

```
T*u*e*s*d*a*y
```

Returning values

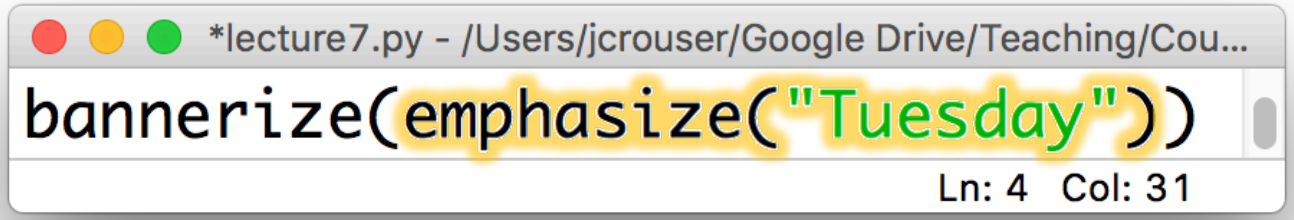
- We may want to **return** the results rather than print them:

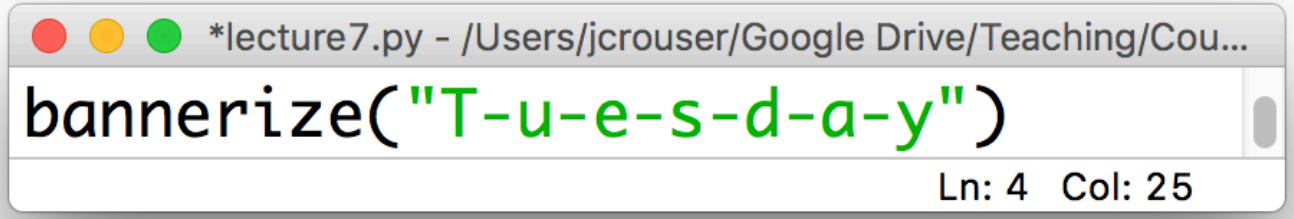


the results of the **`return`** in **`emphasize()`** are stored in **`boom`**

Advanced: chaining functions

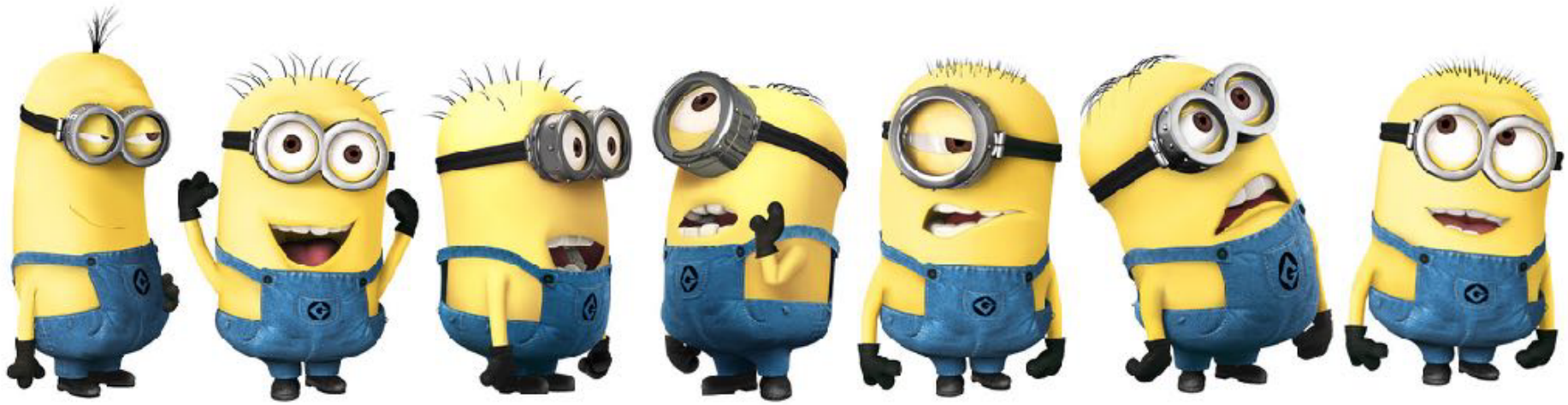
- Return values allow us to call functions **inside** other function calls:





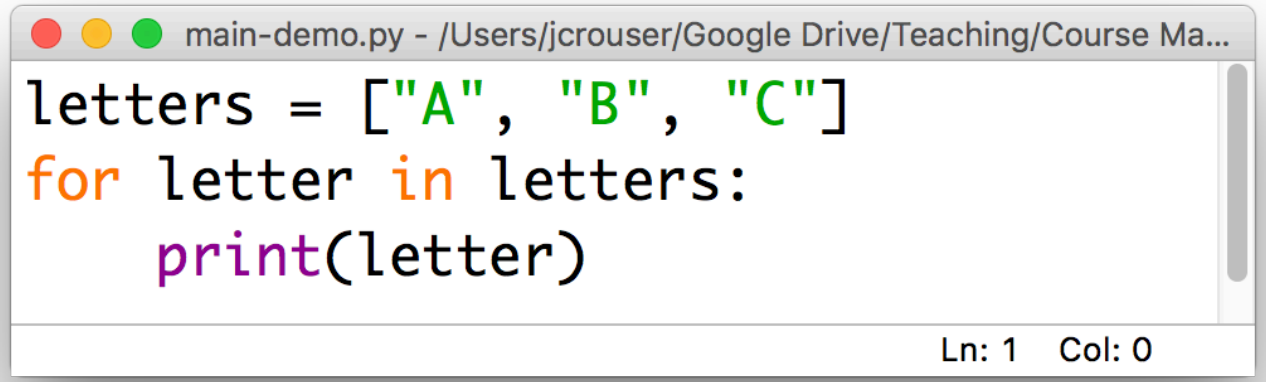
Recap: functions

- If you have to do something **multiple times**, then you probably want a function: this helps to "modularize" code (i.e. organize it for easy reuse)
- **Define** once, **call** as many times as necessary
- Naming convention: use **`camelCase`**
- **Important**: one function = one task



One more thing…

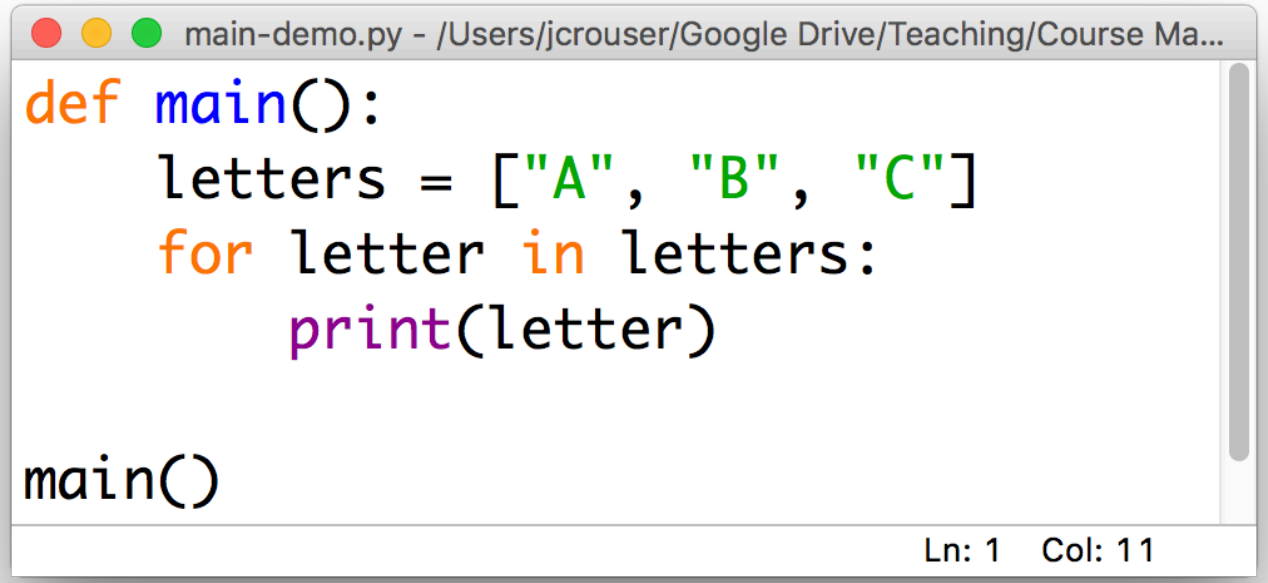
- So far, we've been writing code in files as if we were writing it on the console:



- When we do this, the Python interpreter executes everything from the **top down**

An alternative

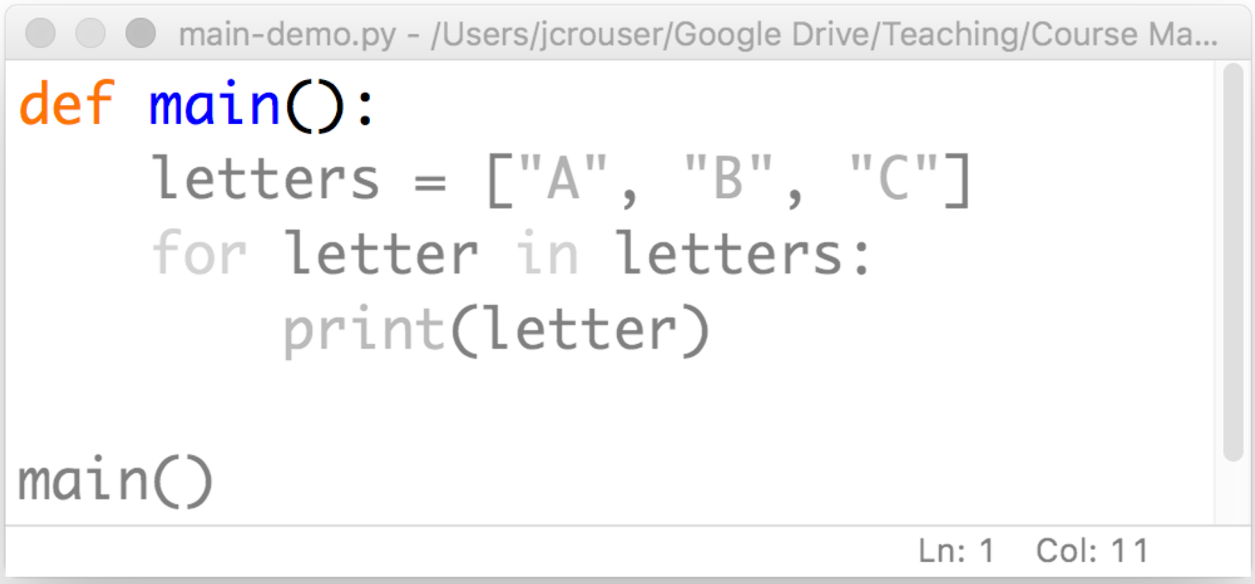
- It is better practice to write the code you want to execute inside a `main()` function, e.g.



- This lets the interpreter ”read ahead” and **then** execute

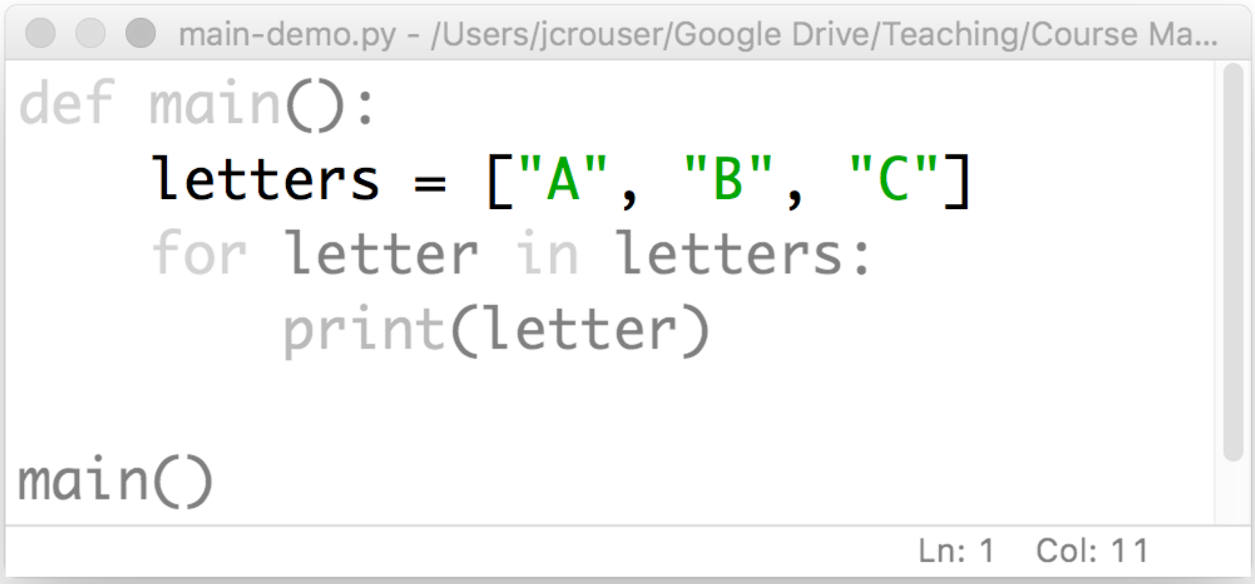
How this works

- **Remember**: the interpreter reads from the top down, which means that it reads the **definition** first



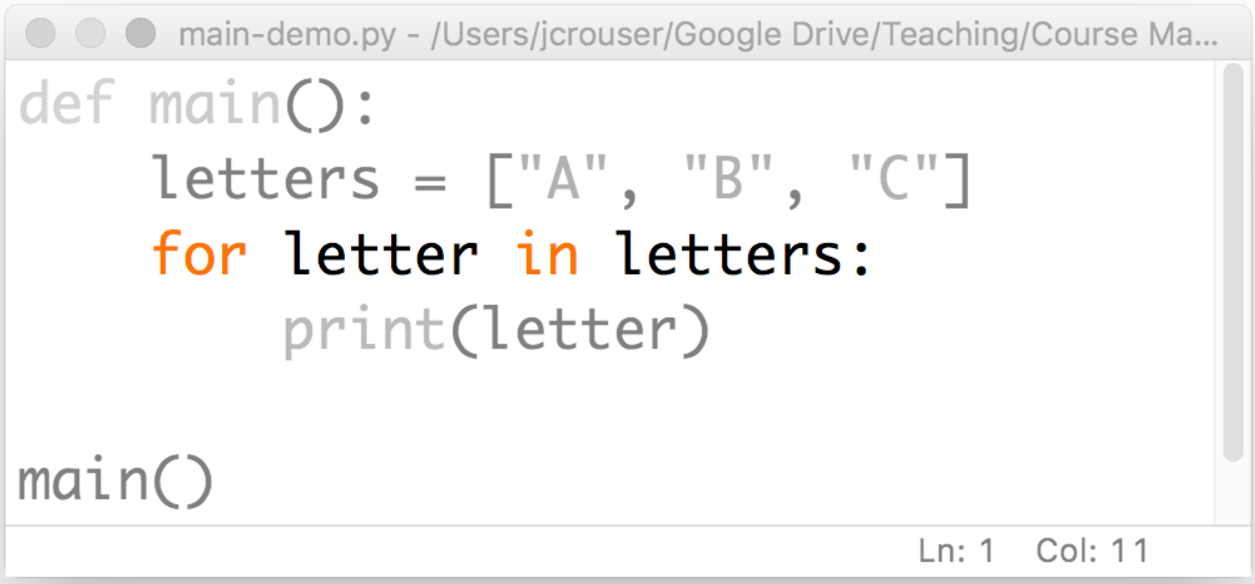
How this works

- Then it reads each line inside the definition, but these don't get **executed** yet



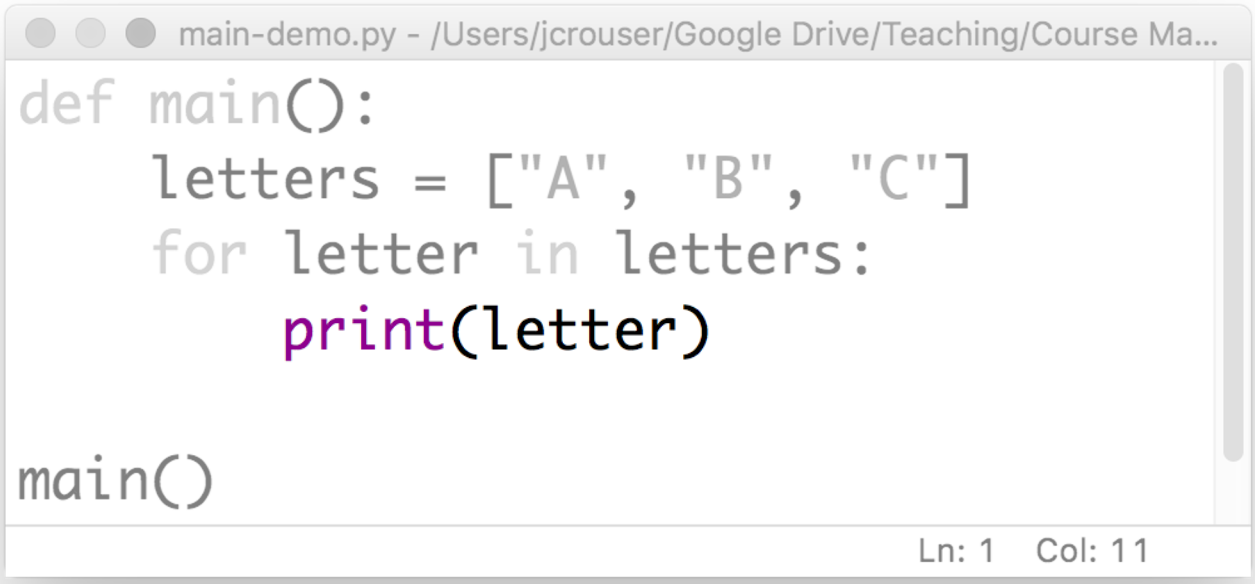
How this works

- Then it reads each line inside the definition, but these don't get **executed** yet



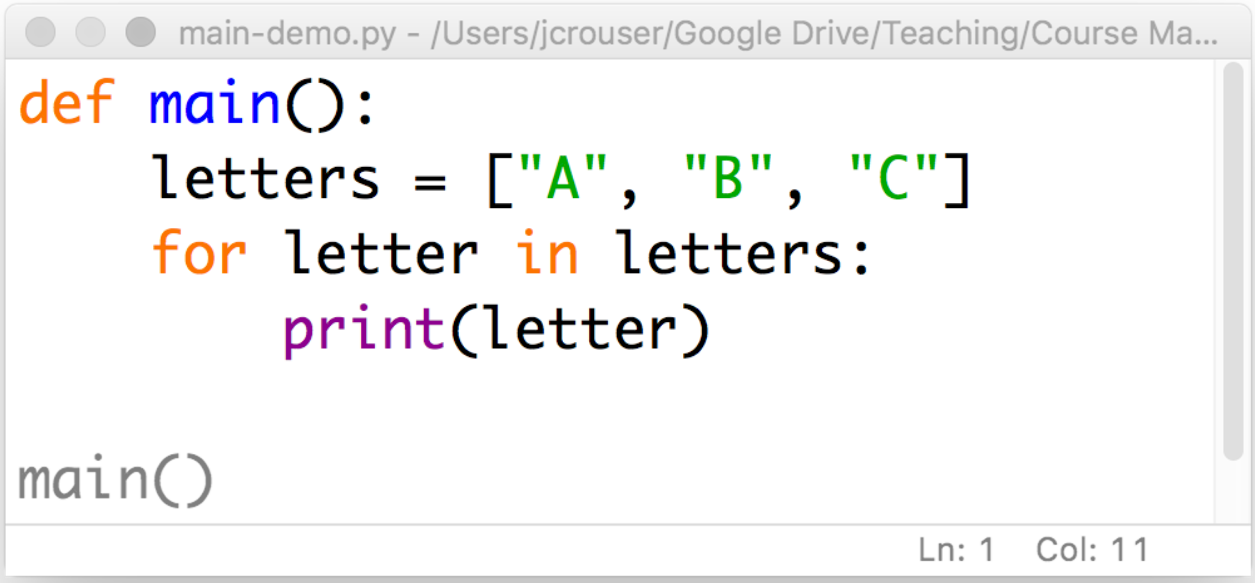
How this works

- Then it reads each line inside the definition, but these don't get **executed** yet



How this works

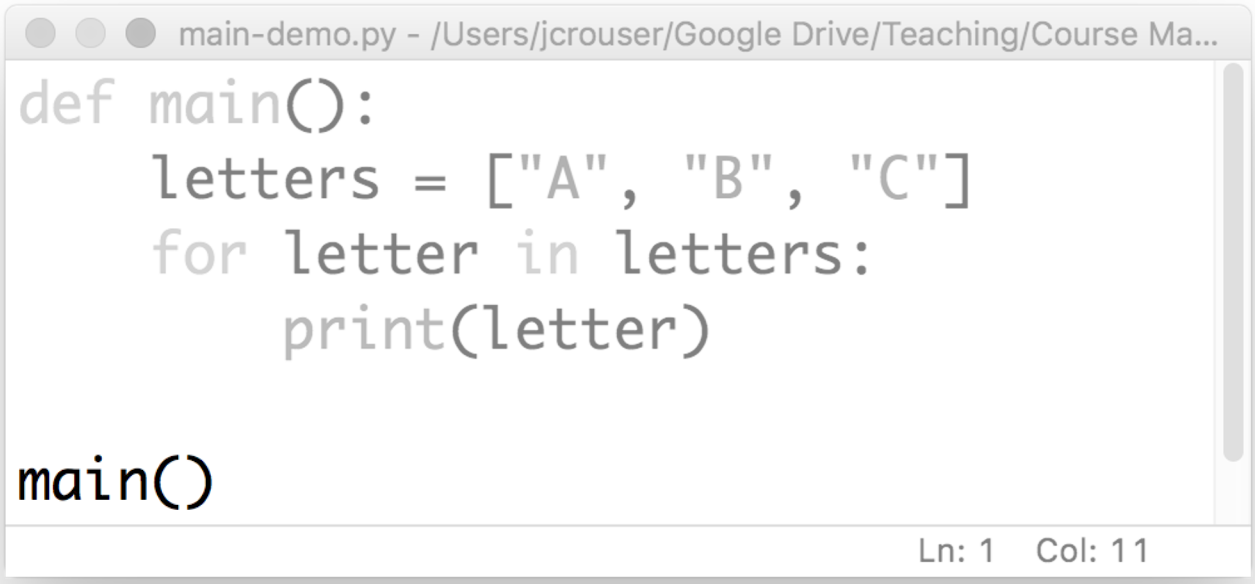
- At this stage, we've given python a "recipe" for what we want it to do when we call **`main()`**



- If we stop here, **nothing will actually happen**

How this works

- The real work happens only when we actually **call** the **`main()`** function



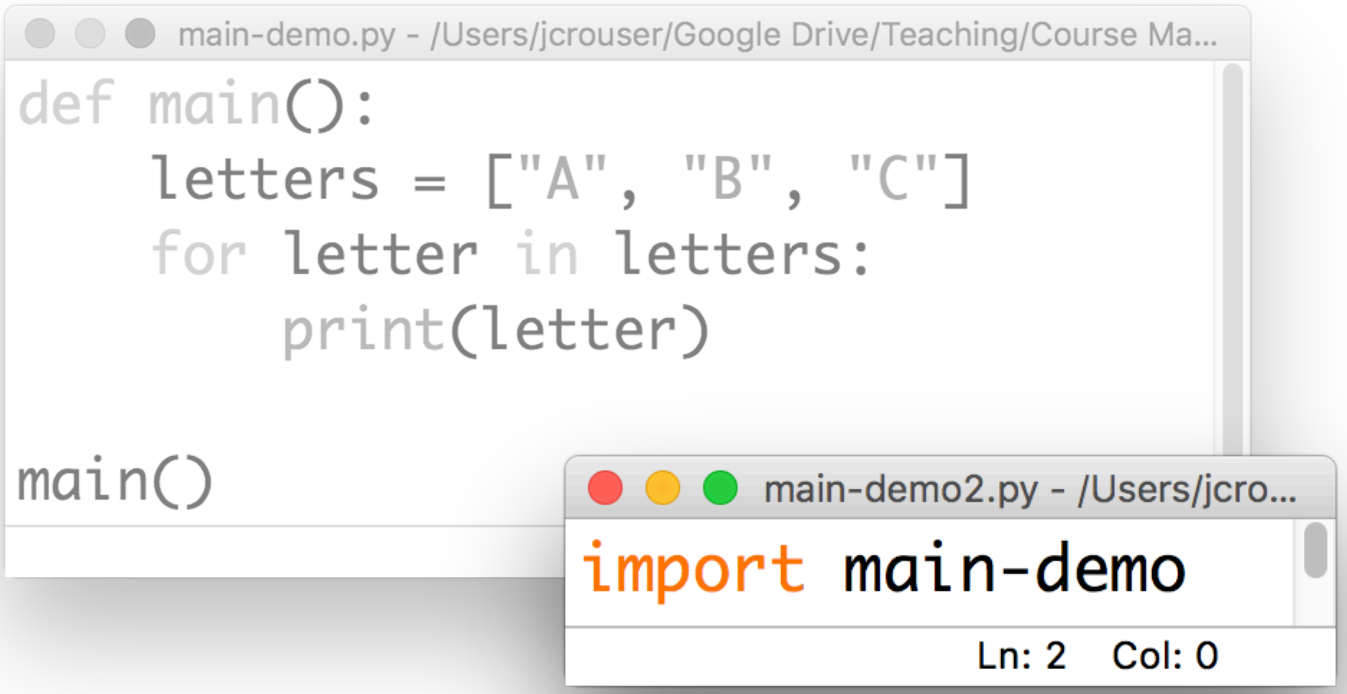
- When we do, python goes to the **`main()`** box and follows the instructions it finds there

Discussion

Why bother?

Just one more thing…

- What happens if someday we want to use the code in this file as **part of another program**?



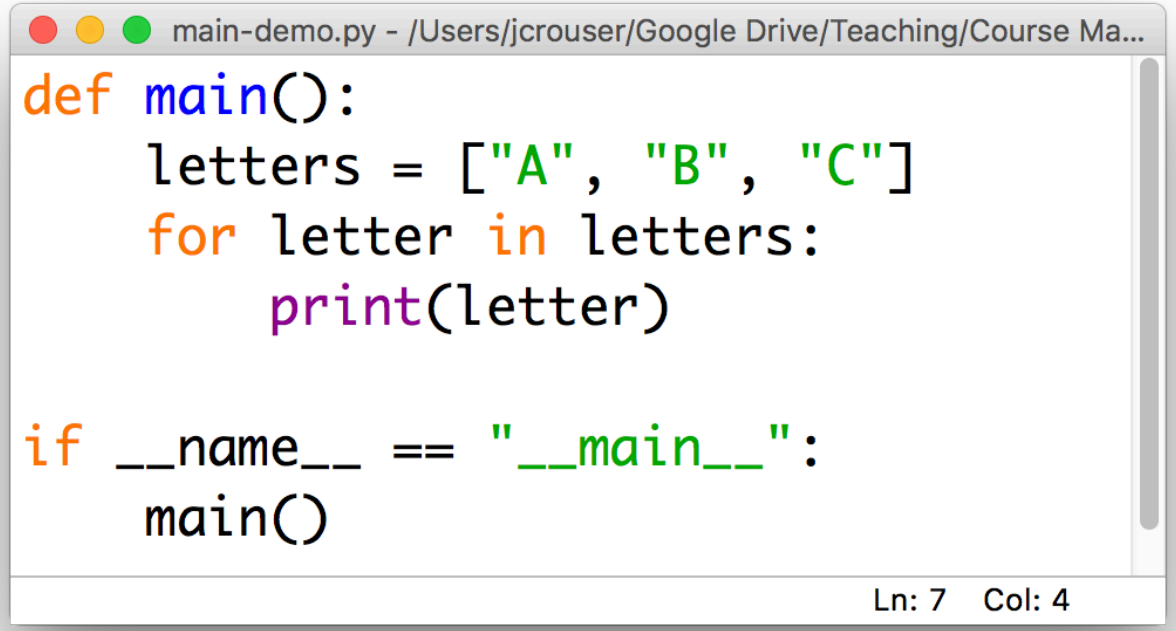
Discussion

- **What we need:** a way to tell python to behave one way when we **run it as a "stand-alone" program**, and a different way when we `import` it

Ideas?

Python convention

- We can use an **`if`** statement to tell python to call the **`main()`** function only if the program is being run directly



Python convention

- This is a little bit **confusing**: we named the function we created to hold our program was called **`main()`**

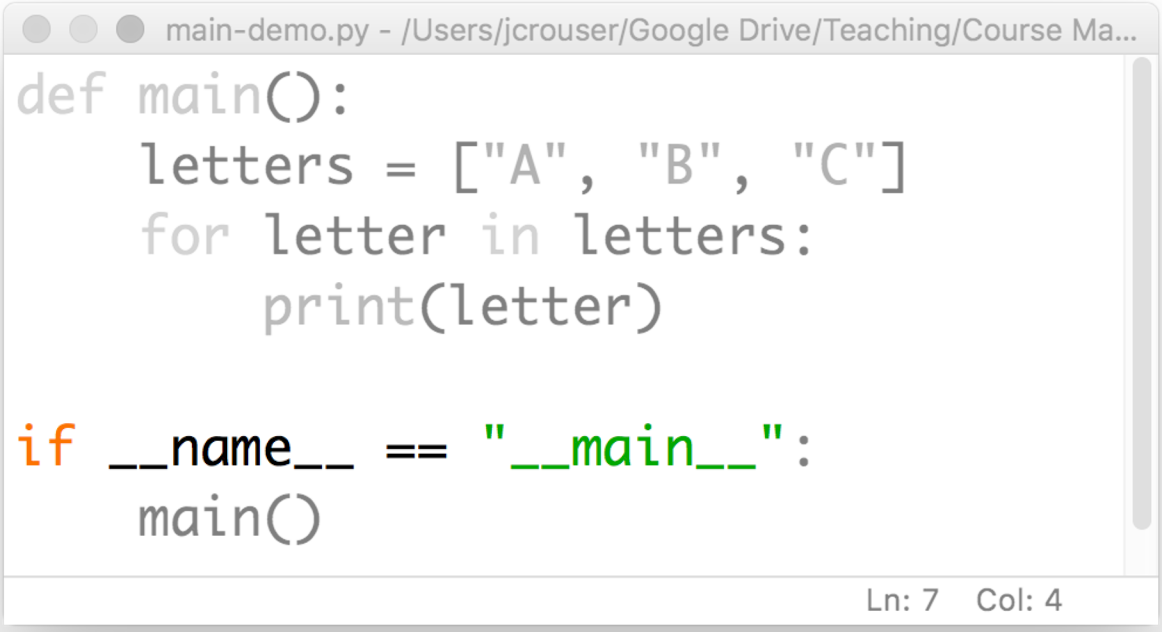
main-demo.py - /Users/jcrouser/Google Drive/Teaching/Course Ma...

```
def main():
    letters = ["A", "B", "C"]
    for letter in letters:
        print(letter)

if __name__ == "__main__":
    main()
```

Ln: 7 Col: 4

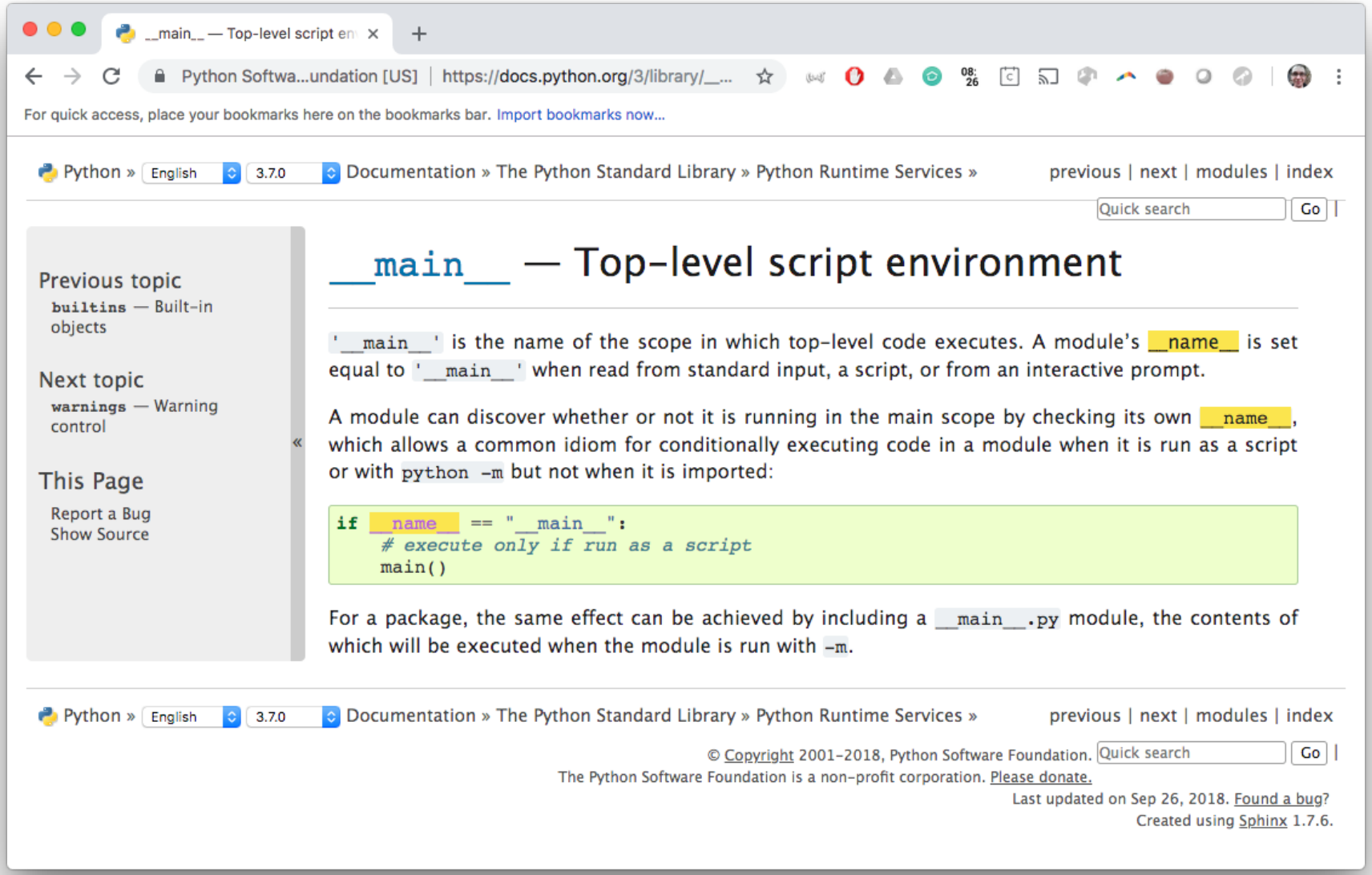
Python convention

- In our **`if` statement**, we're asking whether some variable called **`__name__`** is equal to the string **`"__main__"`**



- (not to mention I don't recall initializing anything called **`__name__`**...)

To the documentation!



15-minute exercise

- Write a program that contains a `main()` function, which contains instructions for printing out the phrase:

```
"This program is being run directly."
```

- Use an `if` statement combined with checking the value of the `__name__` variable to call `main()` only when the program is run directly
- Add an `else` statement so that whenever the program ("module") is `import`ed, it prints out the phrase:

```
"This program has been imported!"
```

Discussion

What did you come up with?

Takeaways

- Programs (“modules”) that are well-organized are **easier to read**, more **versatile**, and potentially **more efficient**
- The first step we’ll take toward organizing our code is to include a `main()` function, which includes the instructions we want our program to run
- To make it easier to `import` code we write now into later modules, we will follow the convention of including:

```
if __name__ == "__main__":
    main()
```

at the end of each module

Helpful tip: have a starter template

starter.py - /Users/jcrouser/Google Drive/Teaching/Course Material/CSC111/CSC111/demos/start...

```
#-----------------------------------------------
#        Names: Jordan Crouser & <PARTNER>
#         Date: <DATE>
#     Filename: starter.py
# Description: This is a demonstration of how to
#              organize your starter code (incl.
#              a main function and scope check)
#-----------------------------------------------

def main():
    # This is where my code will go

if __name__ == "__main__":
    main()

# REFERENCES
```

Ln: 3 Col: 21

Takeaways

What's **one thing** you learned

in today's class?

Outline

- ✓Monday: first lesson in functions
- Lab: wrapping up collections (lists, dictionaries)
- Wednesday: more on functions
- Friday: data structures comparison