

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. ❤️d 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
#                  properly 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
#                  properly 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  
#                  properly 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  
#                  properly 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  
#                  properly 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
```

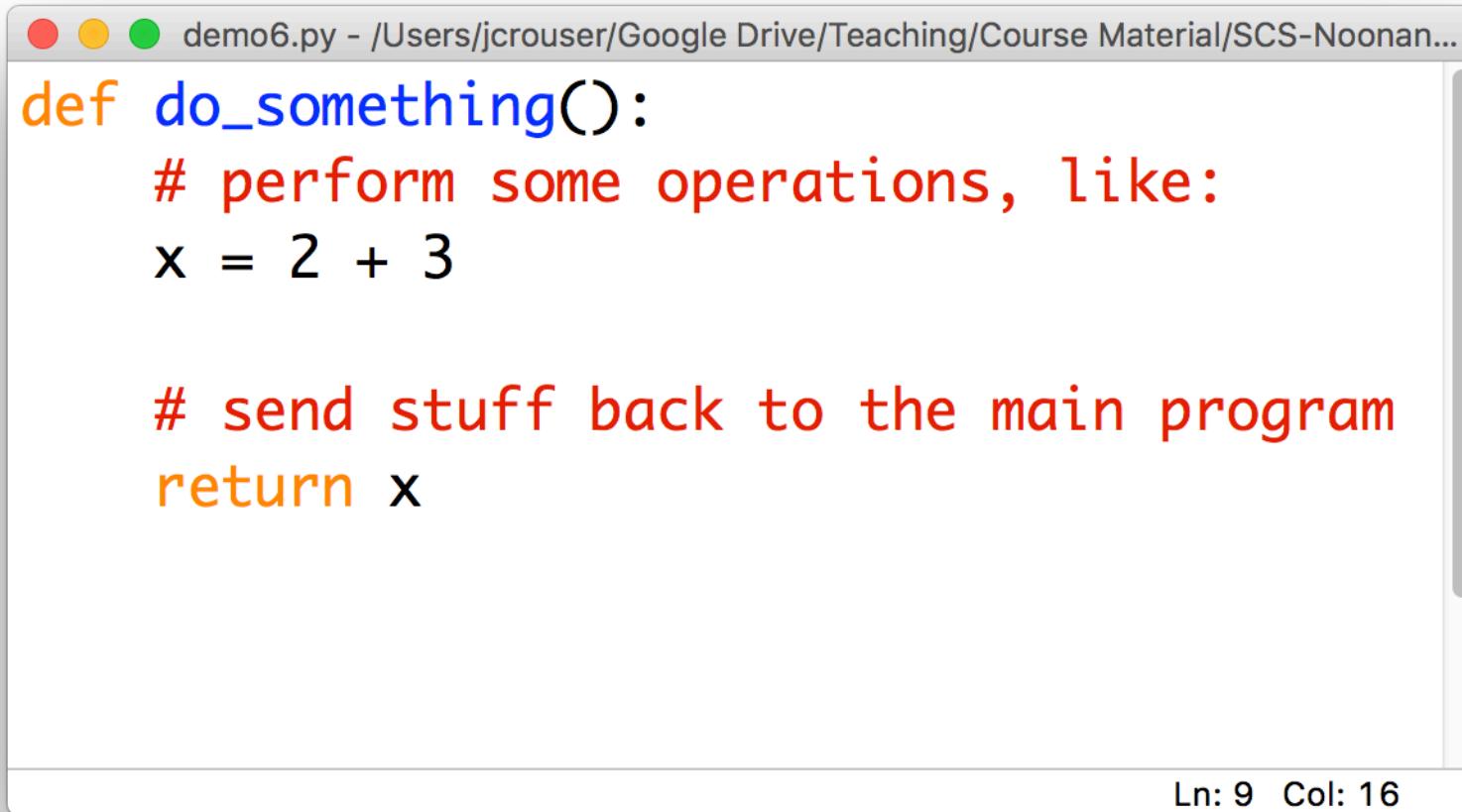
Outline

- 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



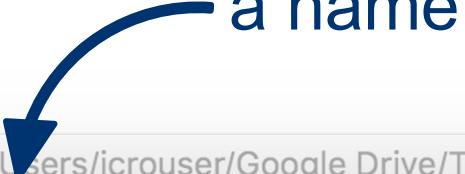
A screenshot of a Mac OS X application window titled "demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan...". The window contains Python code:

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

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

The status bar at the bottom right shows "Ln: 9 Col: 16".

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

Ln: 9 Col: 16
```

Convention: use _underscores_ or **camelCase**

Basic components of a function

which is defined
using the **def** keyword



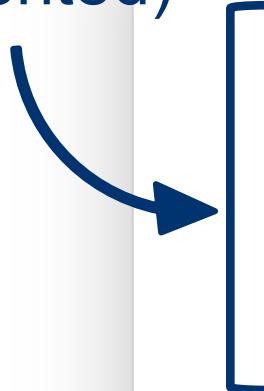
```
demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan...
def do_something():
    # perform some operations, like:
    x = 2 + 3

    # send stuff back to the main program
    return x

Ln: 9 Col: 16
```

Basic components of a function

a **body**
(indented)



```
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

```
● ● ● demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan...
def do_something():
    # perform some operations, like:
    x = 2 + 3

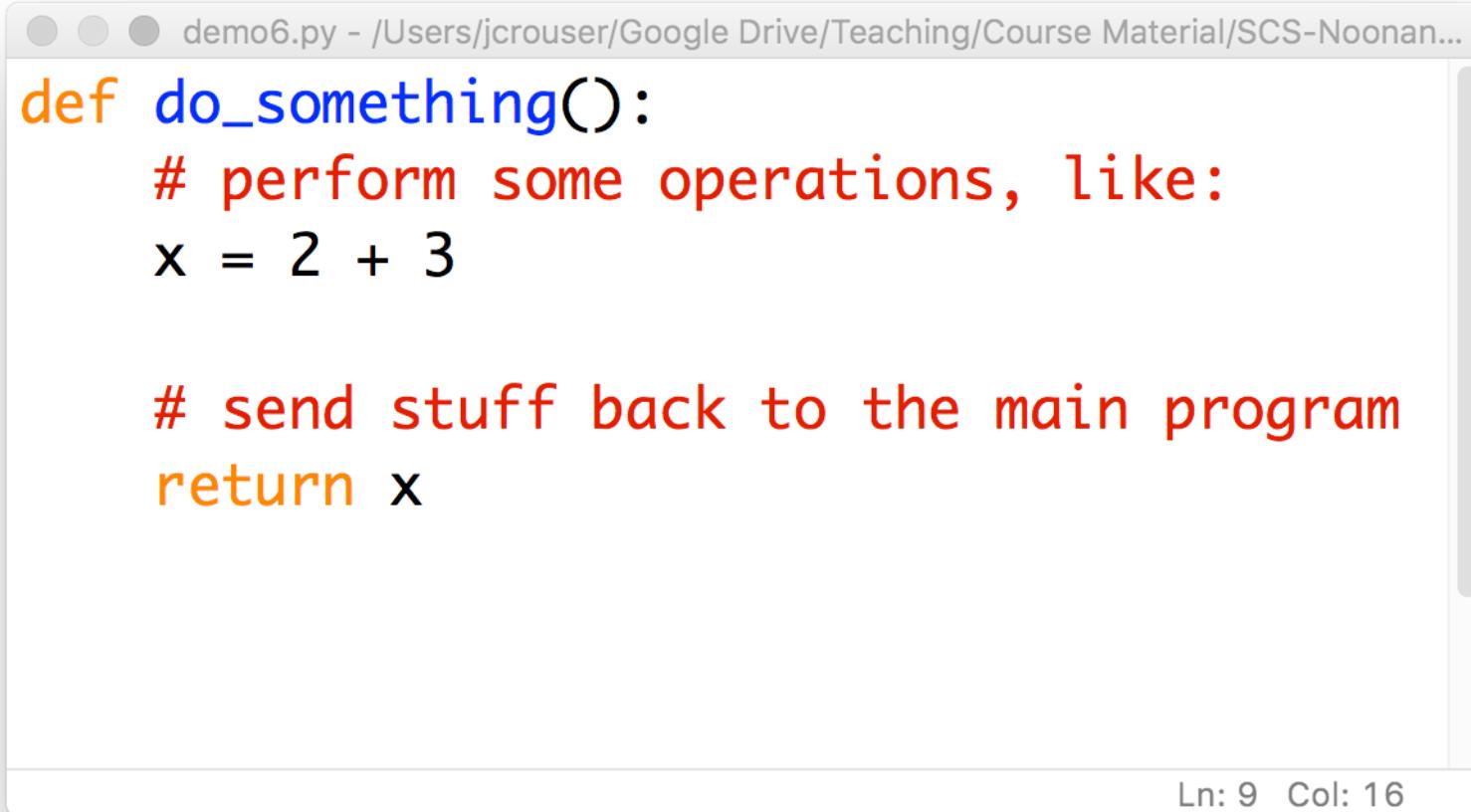
    # send stuff back to the main program
    return x
```

a **return** (optional)



Ln: 9 Col: 16

A “function definition”



The image shows a screenshot of a code editor window titled "demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan...". The code editor displays the following Python function definition:

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

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

A blue bracket on the left side of the slide points to the start of the function definition, specifically the "def" keyword.

Ln: 9 Col: 16

Discussion

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

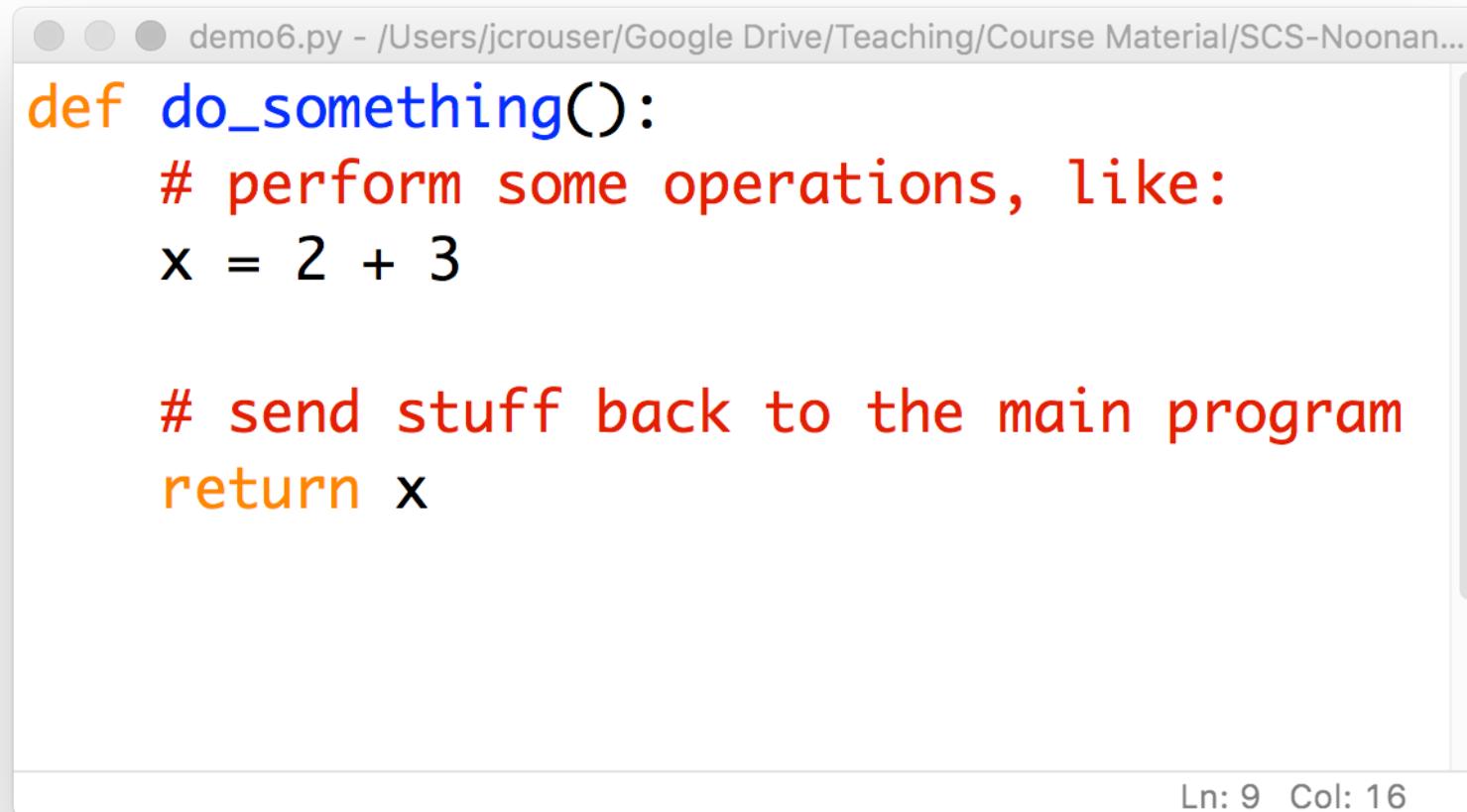
```
demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan...
def do_something():
    # perform some operations, like:
    x = 2 + 3

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

Col: 16



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



The image shows a screenshot of a code editor window titled "demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan...". The code defines a function named "do_something" that performs some operations and returns a value. The code is color-coded: "def" and "do_something" are orange, while the docstring and the assignment statement are blue. The return statement is red. A vertical scrollbar is visible on the right side of the editor window. In the bottom right corner of the window, there is a status bar displaying "Ln: 9 Col: 16".

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

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

(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”



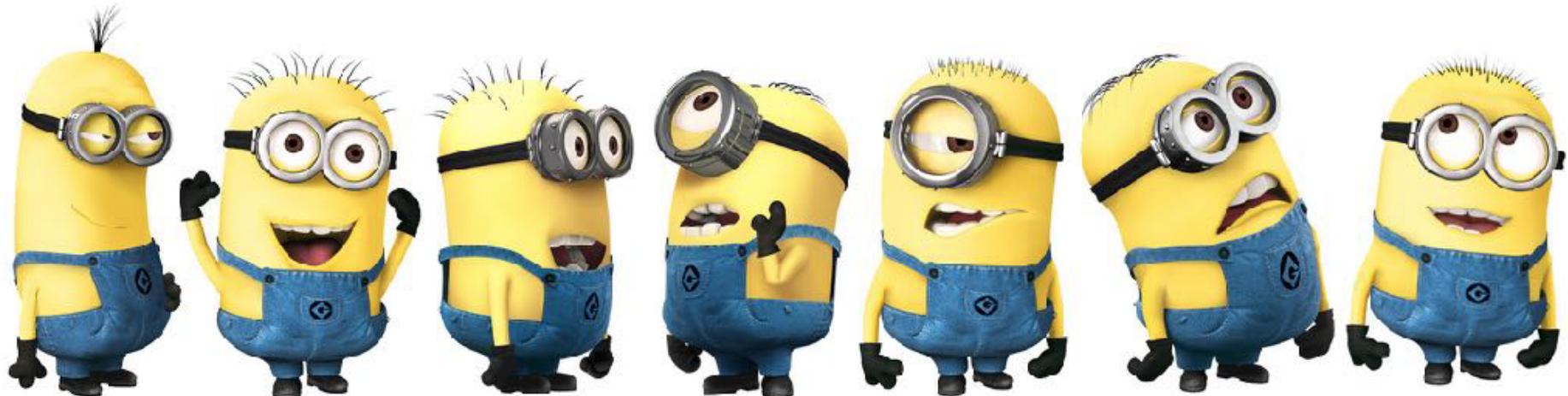
```
demo6.py - /Users/jcrouser/Google Drive/Teaching/Course Material/SCS-Noonan...
def do_something():
    # perform some operations, like:
    x = 2 + 3

    # send stuff back to the main program
    return x 5

y =
```

Ln: 9 Col: 16

An analogy



functions are your **MINIONS**

An analogy



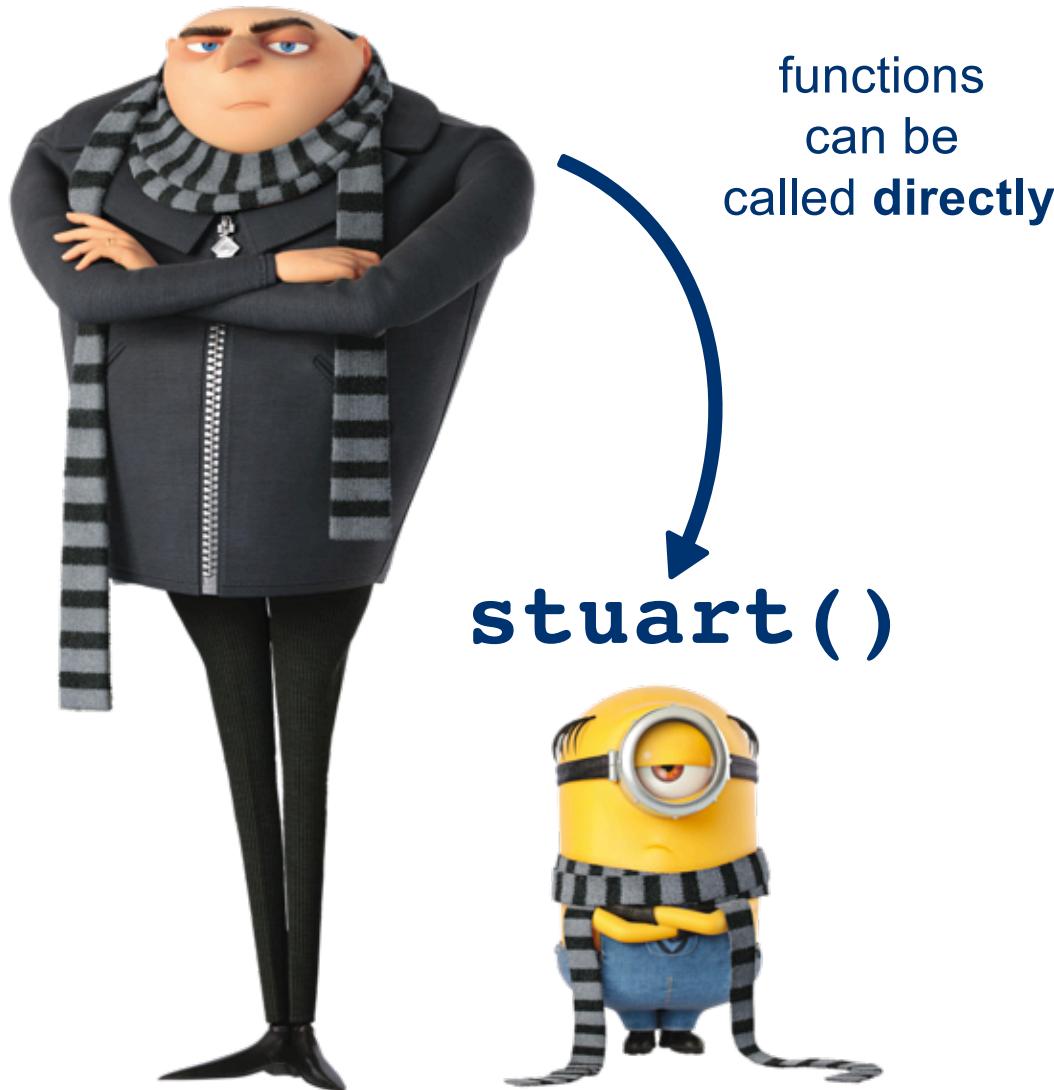
functions have NAMES

An analogy



they only work when you **CALL** them

An analogy



An analogy



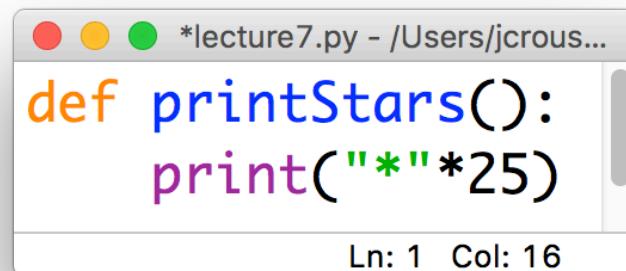
functions
can also be
called by
one another

`stuart()` `jerry()`



Two kinds of functions

Some functions always
do the same thing



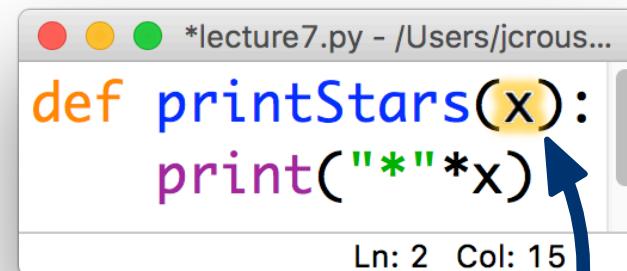
A screenshot of a Mac OS X terminal window titled "*lecture7.py - /Users/jcrou...". It contains the following Python code:

```
def printStars():
    print("*"*25)
```

The status bar at the bottom shows "Ln: 1 Col: 16".

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

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



A screenshot of a Mac OS X terminal window titled "*lecture7.py - /Users/jcrou...". It contains the following Python code:

```
def printStars(x):
    print("*"*x)
```

The status bar at the bottom shows "Ln: 2 Col: 15". A blue arrow points from the word "parameter" below to the variable "x" in the code.

```
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 several parameters:

A screenshot of a Mac OS X window titled "lecture7.py - /Users/jcrouser/GDrive/Teaching/Course...". The code editor displays the following Python code:

```
def emphasize(word, char):
    print(char.join(list(word)))
```

Below the code, a call to the function is shown:

```
emphasize("Tuesday", "-")
```

Two blue arrows originate from the parameter names "word" and "char" in the call and point to their respective arguments in the list. The arrow from "word" points to the string "Tuesday", and the arrow from "char" points to the string "-". A status bar at the bottom right of the code editor shows "Ln: 1 Col: 24".

• Result:

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

Default parameters

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

A screenshot of a Mac OS X desktop showing a terminal window titled "*lecture7.py - /Users/jcrouser/Google Drive/Teaching/Cours...". The window contains the following Python code:

```
def emphasize(word, char = "*"):
    print(char.join(list(word)))
```

Below the code, the function is called with the argument "Tuesday":

```
emphasize("Tuesday")
```

The status bar at the bottom right of the terminal window shows "Ln: 4 Col: 20". A blue arrow points from the text "only one parameter" to the opening parenthesis of the function call "emphasize(".

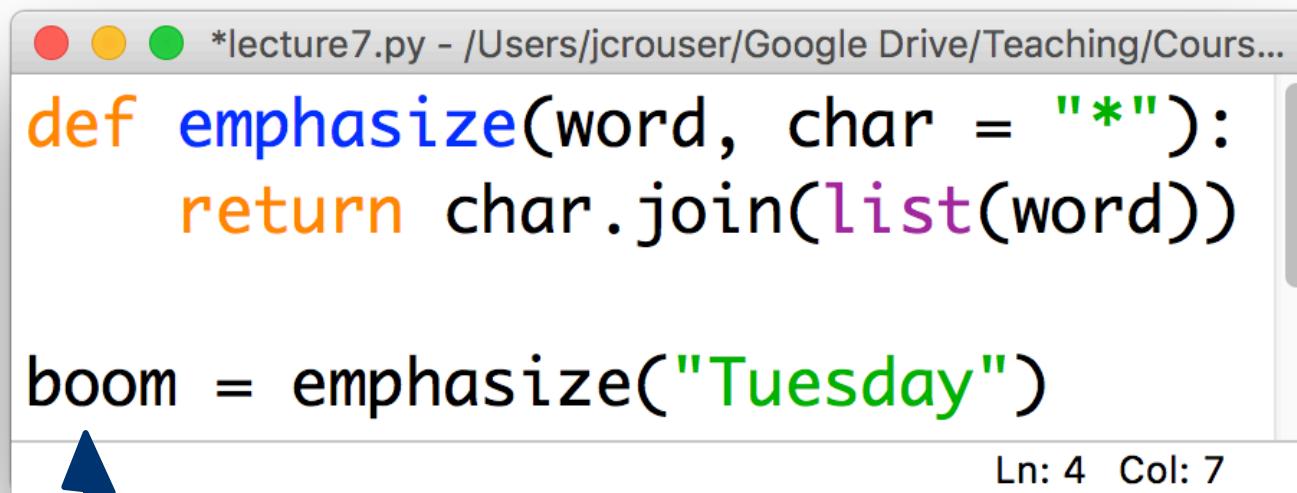
only one parameter

- Result:

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

Returning values

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



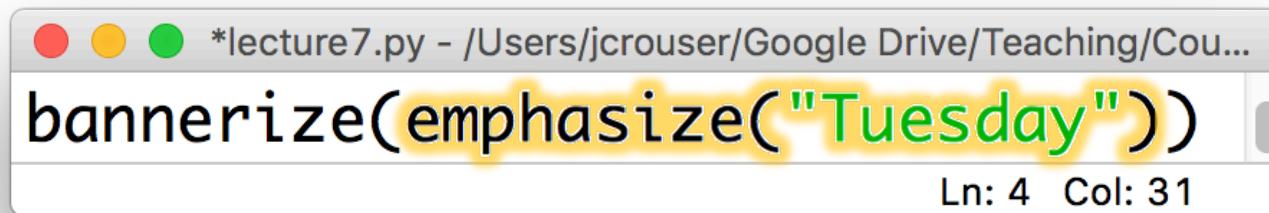
```
*lecture7.py - /Users/jcrouser/Google Drive/Teaching/Cours...
def emphasize(word, char = "*"):
    return char.join(list(word))

boom = emphasize("Tuesday")
Ln: 4 Col: 7
```

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

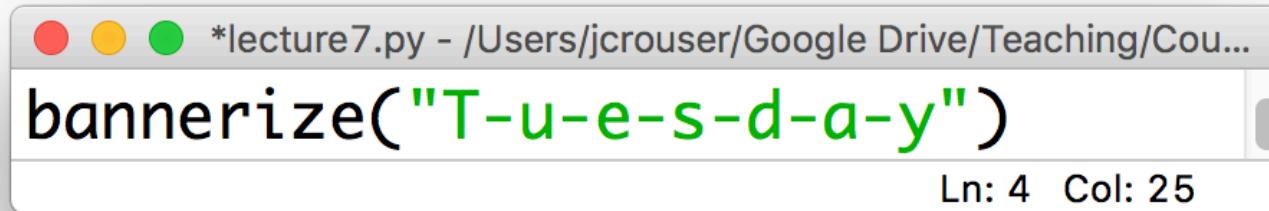
Advanced: chaining functions

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



A screenshot of a Mac OS X desktop showing a terminal window titled '*lecture7.py - /Users/jcrouser/Google Drive/Teaching/Cou...'. The terminal contains the following code:
`bannerize(emphasize("Tuesday"))`

Ln: 4 Col: 31



A screenshot of a Mac OS X desktop showing a terminal window titled '*lecture7.py - /Users/jcrouser/Google Drive/Teaching/Cou...'. The terminal contains the following code:
`bannerize("T-u-e-s-d-a-y")`

Ln: 4 Col: 25

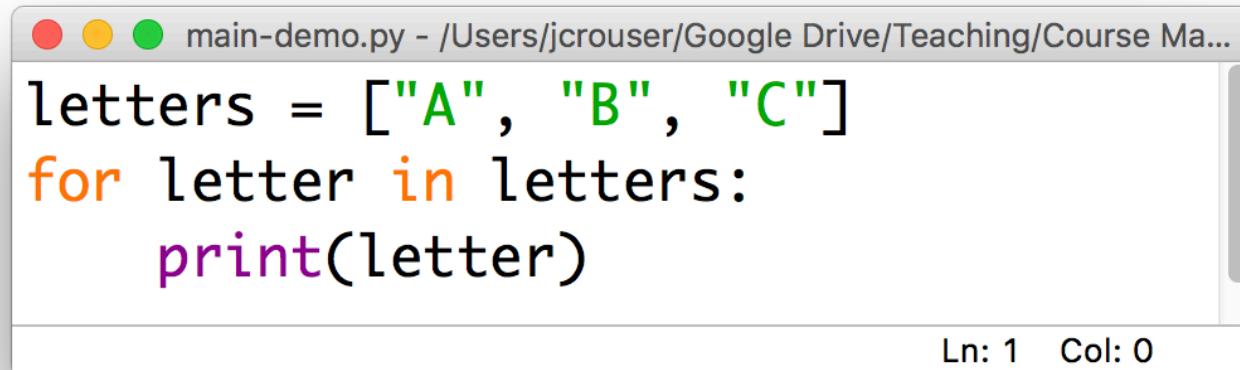
Recap: functions

- 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:



A screenshot of a terminal window titled "main-demo.py - /Users/jcrouser/Google Drive/Teaching/Course Ma...". The window contains the following Python code:

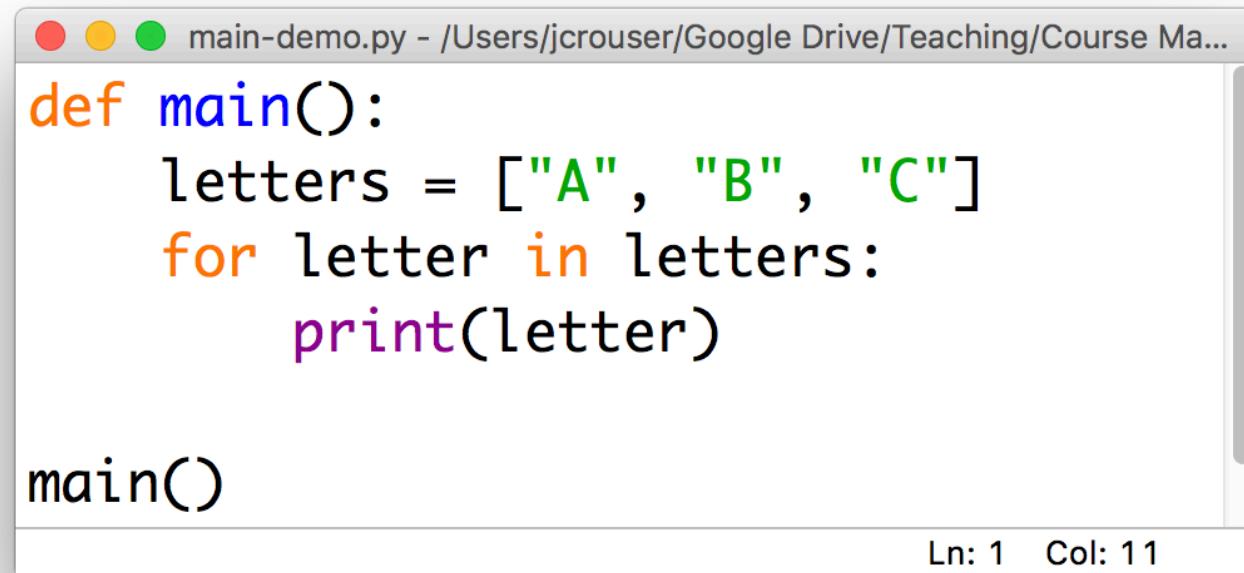
```
letters = ["A", "B", "C"]
for letter in letters:
    print(letter)
```

The status bar at the bottom right of the terminal window shows "Ln: 1 Col: 0".

- 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.



A screenshot of a terminal window titled "main-demo.py - /Users/jcrouser/Google Drive/Teaching/Course Ma...". The window contains the following Python code:

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

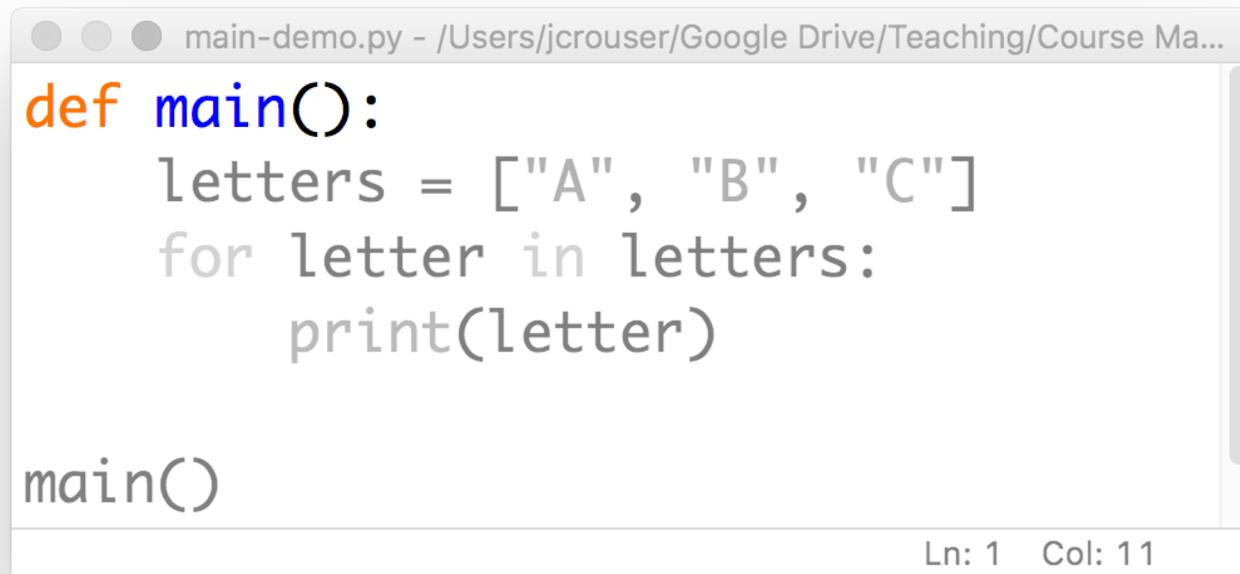
main()
```

The status bar at the bottom right of the terminal window shows "Ln: 1 Col: 11".

- 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



A screenshot of a code editor window titled "main-demo.py - /Users/jcrouser/Google Drive/Teaching/Course Ma...". The code in the editor is as follows:

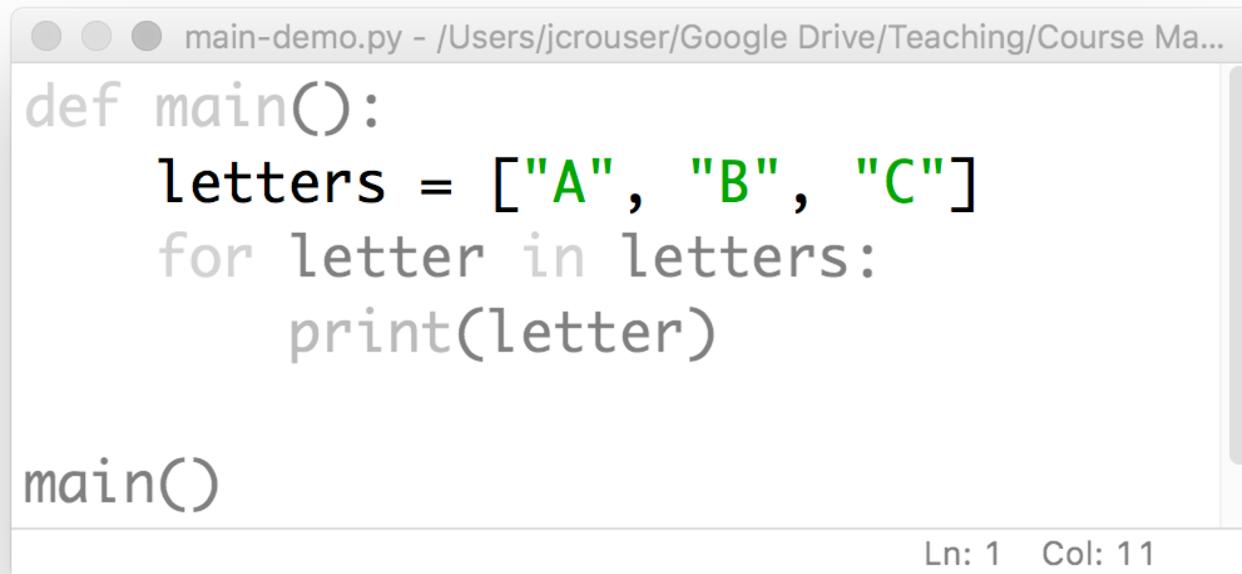
```
def main():
    letters = ["A", "B", "C"]
    for letter in letters:
        print(letter)

main()
```

The word "def" is highlighted in orange, and "main" is highlighted in blue. The status bar at the bottom right shows "Ln: 1 Col: 11".

How this works

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



A screenshot of a code editor window titled "main-demo.py - /Users/jcrouser/Google Drive/Teaching/Course Ma...". The code in the editor is:

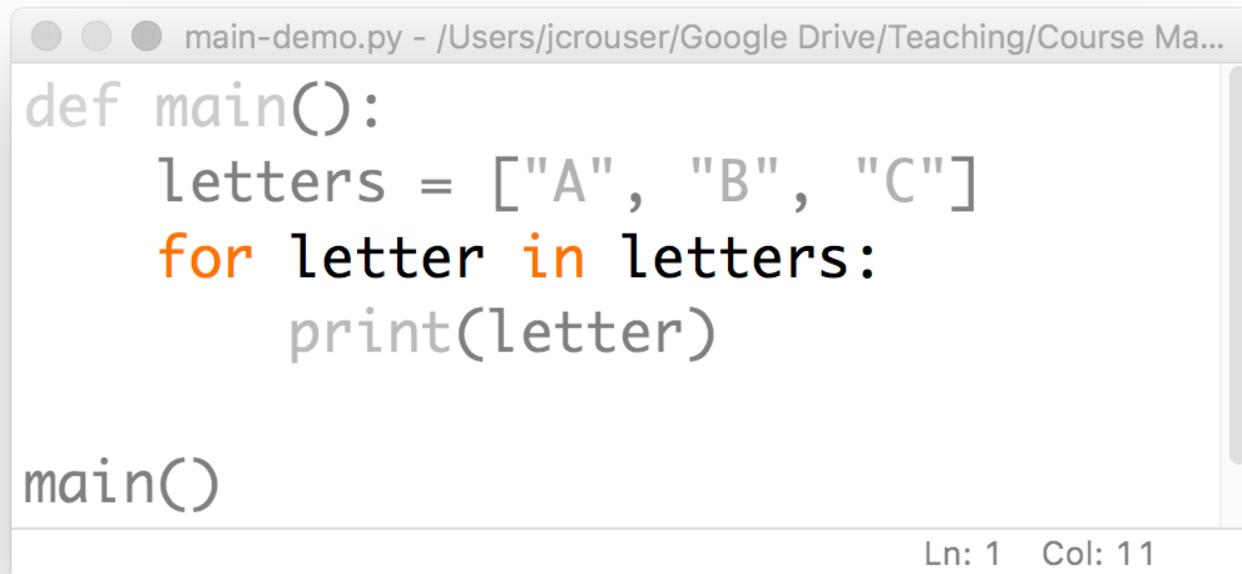
```
def main():
    letters = ["A", "B", "C"]
    for letter in letters:
        print(letter)

main()
```

The code editor shows the file path at the top, the code itself in the main text area, and a status bar at the bottom indicating "Ln: 1 Col: 11".

How this works

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



The image shows a terminal window with the title bar "main-demo.py - /Users/jcrouser/Google Drive/Teaching/Course Ma...". The window contains the following Python code:

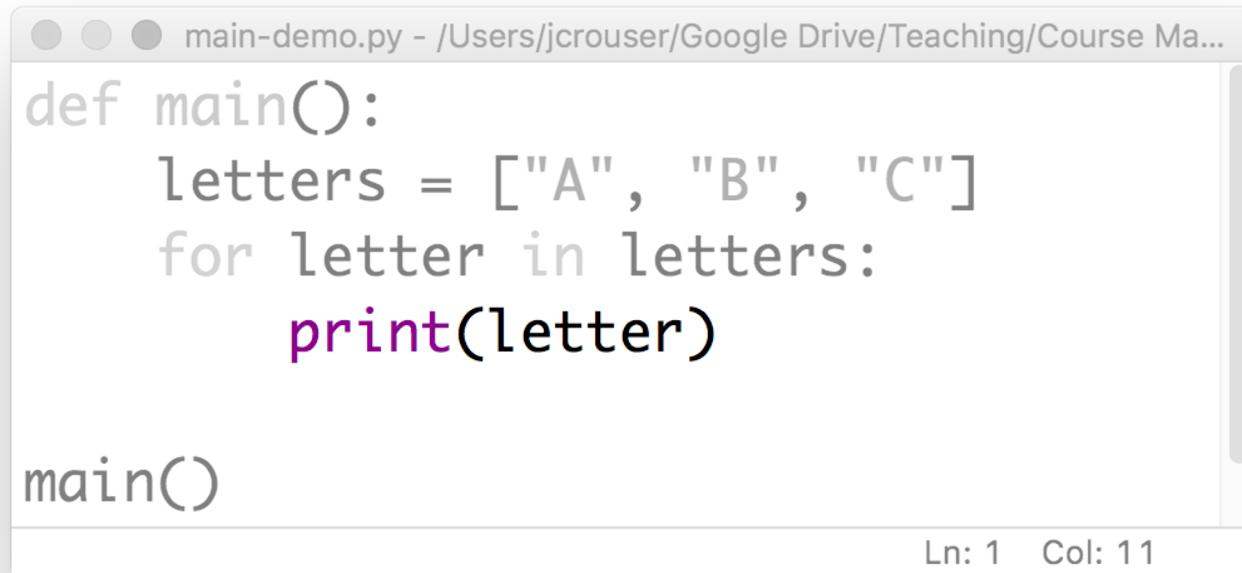
```
def main():
    letters = ["A", "B", "C"]
    for letter in letters:
        print(letter)

main()
```

At the bottom of the terminal window, the status bar displays "Ln: 1 Col: 11".

How this works

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



A screenshot of a terminal window titled "main-demo.py - /Users/jcrouser/Google Drive/Teaching/Course Ma...". The window contains the following Python code:

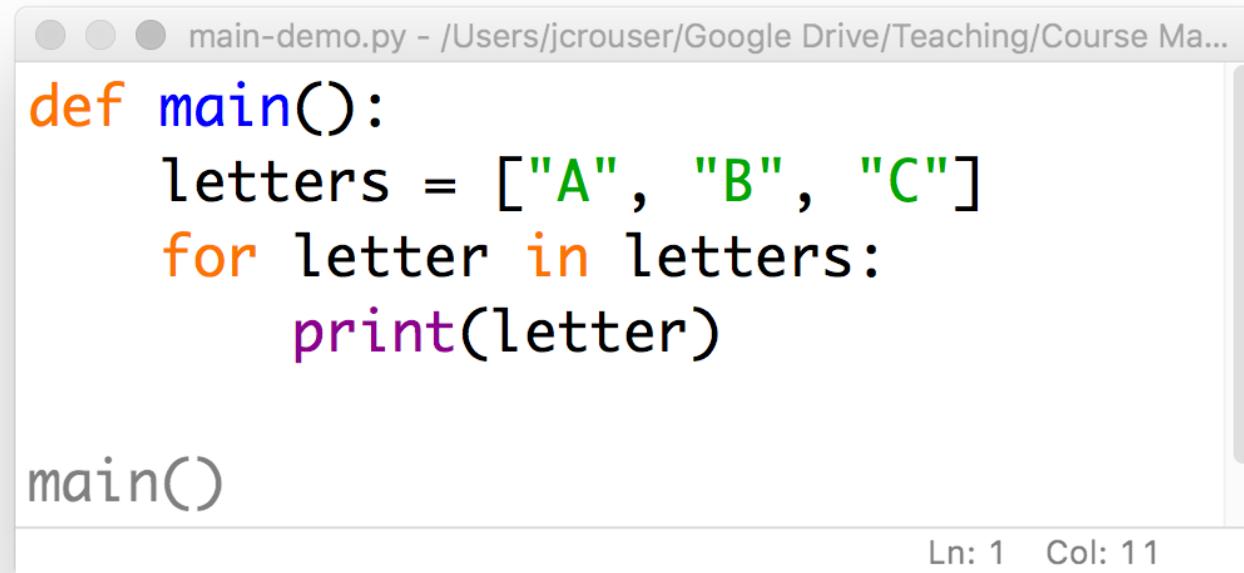
```
def main():
    letters = ["A", "B", "C"]
    for letter in letters:
        print(letter)

main()
```

The word "print" is highlighted in purple. At the bottom right of the terminal window, it says "Ln: 1 Col: 11".

How this works

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



The image shows a screenshot of a code editor window titled "main-demo.py - /Users/jcrouser/Google Drive/Teaching/Course Ma...". The code in the editor is:

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

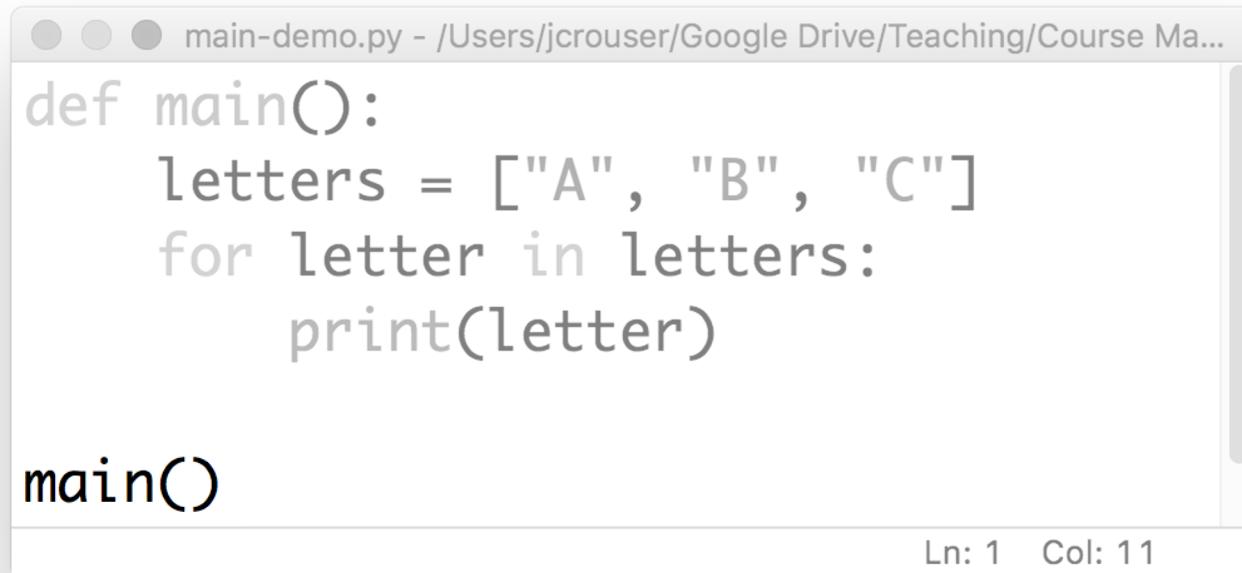
main()
```

The code is color-coded: "def", "main", "for", "in", "print" are in purple; "letters" is in blue; and the string values "A", "B", "C" are in green. The status bar at the bottom right of the editor window displays "Ln: 1 Col: 11".

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

How this works

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



A screenshot of a code editor window titled "main-demo.py - /Users/jcrouser/Google Drive/Teaching/Course Ma...". The code in the editor is:

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

main()
```

The cursor is positioned at the end of the word "main" in the final line. The status bar at the bottom right shows "Ln: 1 Col: 11".

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

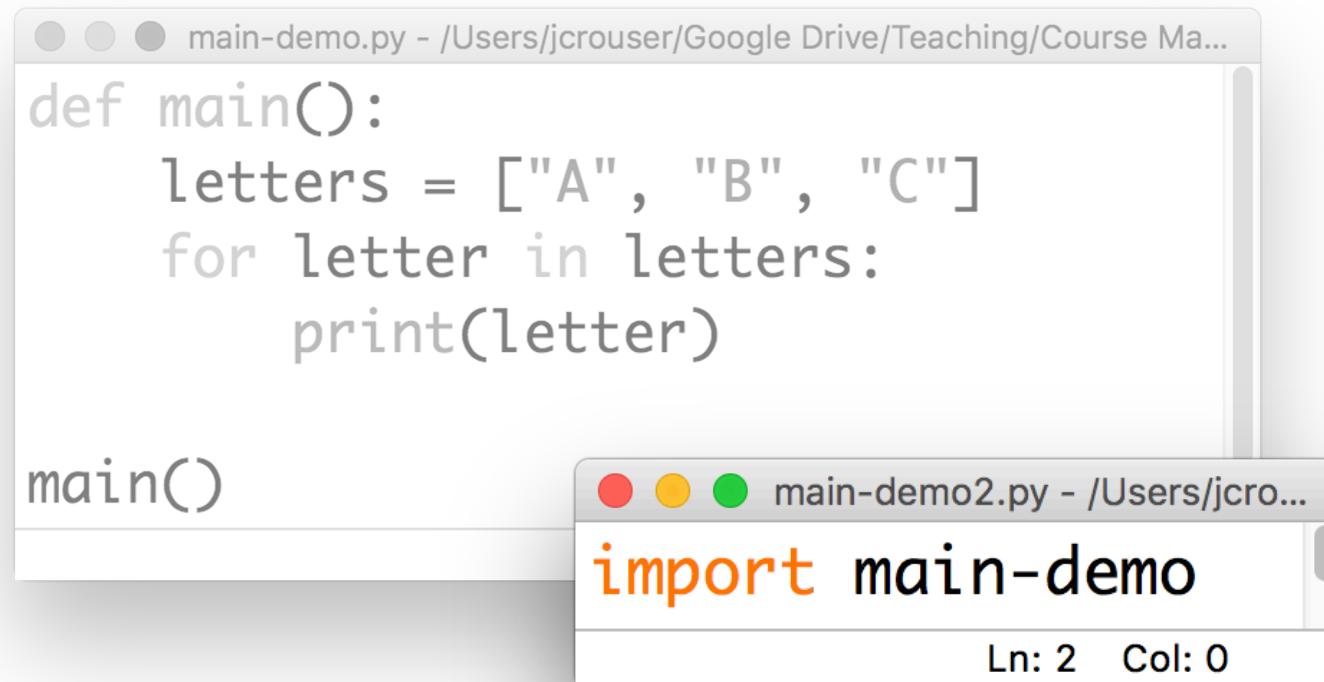
Discussion

Why bother?



Just one more thing...

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



The image shows two separate Python code editors side-by-side. The left editor has a title bar 'main-demo.py - /Users/jcrouser/Google Drive/Teaching/Course Ma...' and contains the following code:

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

The right editor has a title bar 'main-demo2.py - /Users/jcrouser/Google Drive/Teaching/Course Ma...' and contains the following code:

```
import main-demo
```

Both editors show a vertical scroll bar on the right side.

Discussion

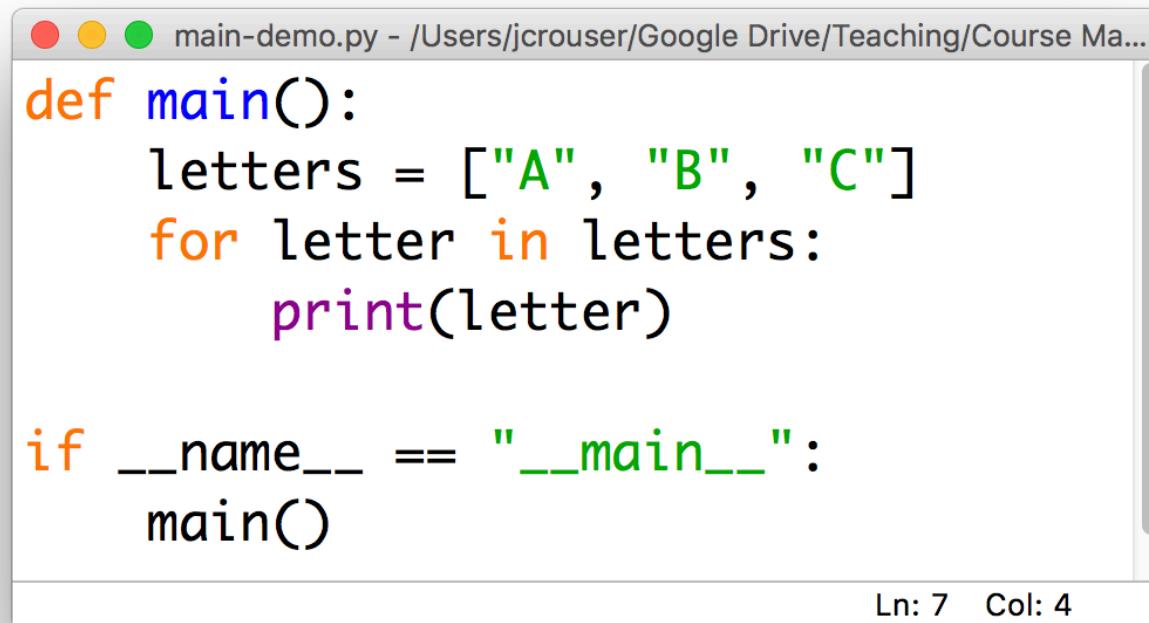
- **What we need:** a way to tell python to behave one way when we **run it as a “stand-alone” program**, and a different way when we **import** it

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



A screenshot of a Python code editor window titled "main-demo.py - /Users/jcrouser/Google Drive/Teaching/Course Ma...". The code is as follows:

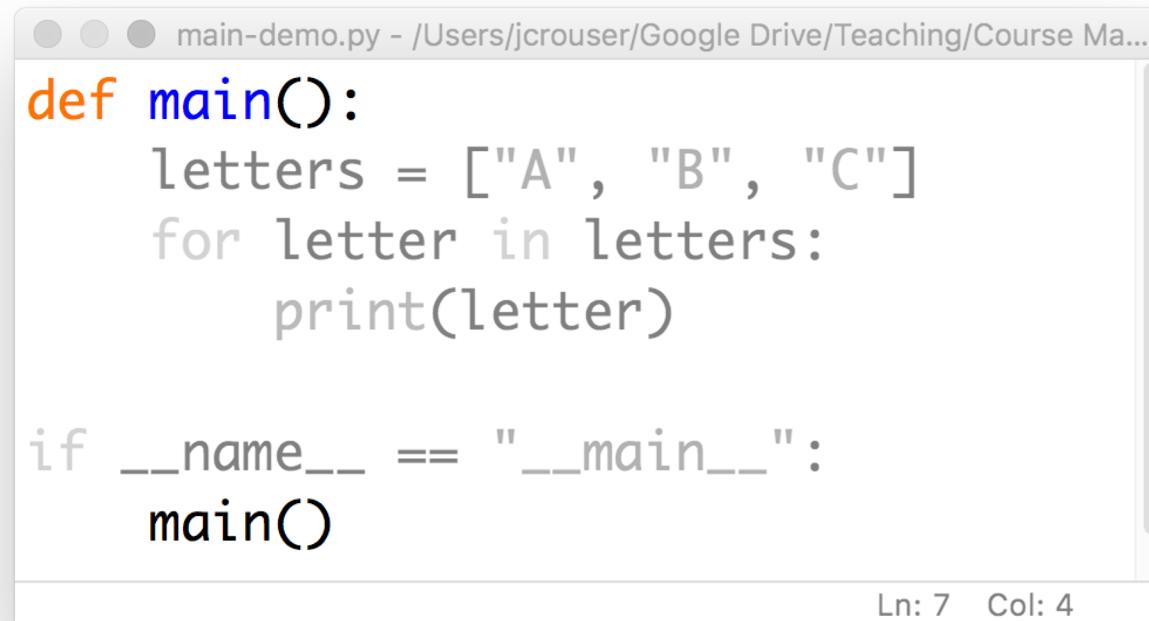
```
def main():
    letters = ["A", "B", "C"]
    for letter in letters:
        print(letter)

if __name__ == "__main__":
    main()

Ln: 7 Col: 4
```

Python convention

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



A screenshot of a Python code editor window titled "main-demo.py - /Users/jcrouser/Google Drive/Teaching/Course Ma...". The code in the editor is:

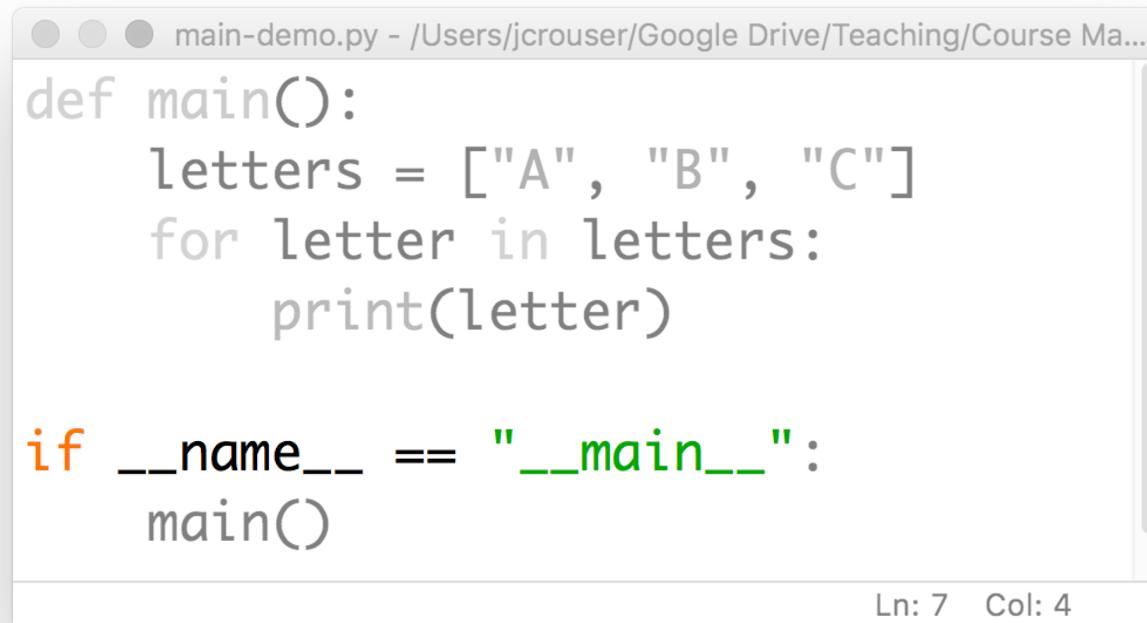
```
def main():
    letters = ["A", "B", "C"]
    for letter in letters:
        print(letter)

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__`"



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

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

Ln: 7 Col: 4

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

To the documentation!

The screenshot shows a web browser window displaying the Python documentation for the `__main__` module. The URL in the address bar is https://docs.python.org/3/library/__main__.html. The page title is `__main__ — Top-level script environment`. The left sidebar contains links to "Previous topic" (`builtins`), "Next topic" (`warnings`), and "This Page" (Report a Bug, Show Source). The main content explains that `'__main__'` is the name of the scope in which top-level code executes. It shows a code snippet for running a script:

```
if __name__ == "__main__":
    # execute only if run as a script
    main()
```

For a package, the same effect can be achieved by including a `__main__.py` module, the contents of which will be executed when the module is run with `-m`.

At the bottom, there is footer information: "Copyright 2001-2018, Python Software Foundation.", "The Python Software Foundation is a non-profit corporation. Please donate.", "Last updated on Sep 26, 2018. Found a bug?", and "Created using Sphinx 1.7.6".

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 `imported`, it prints out the phrase:

 "This program has been imported!"

Discussion

What did you come up with?



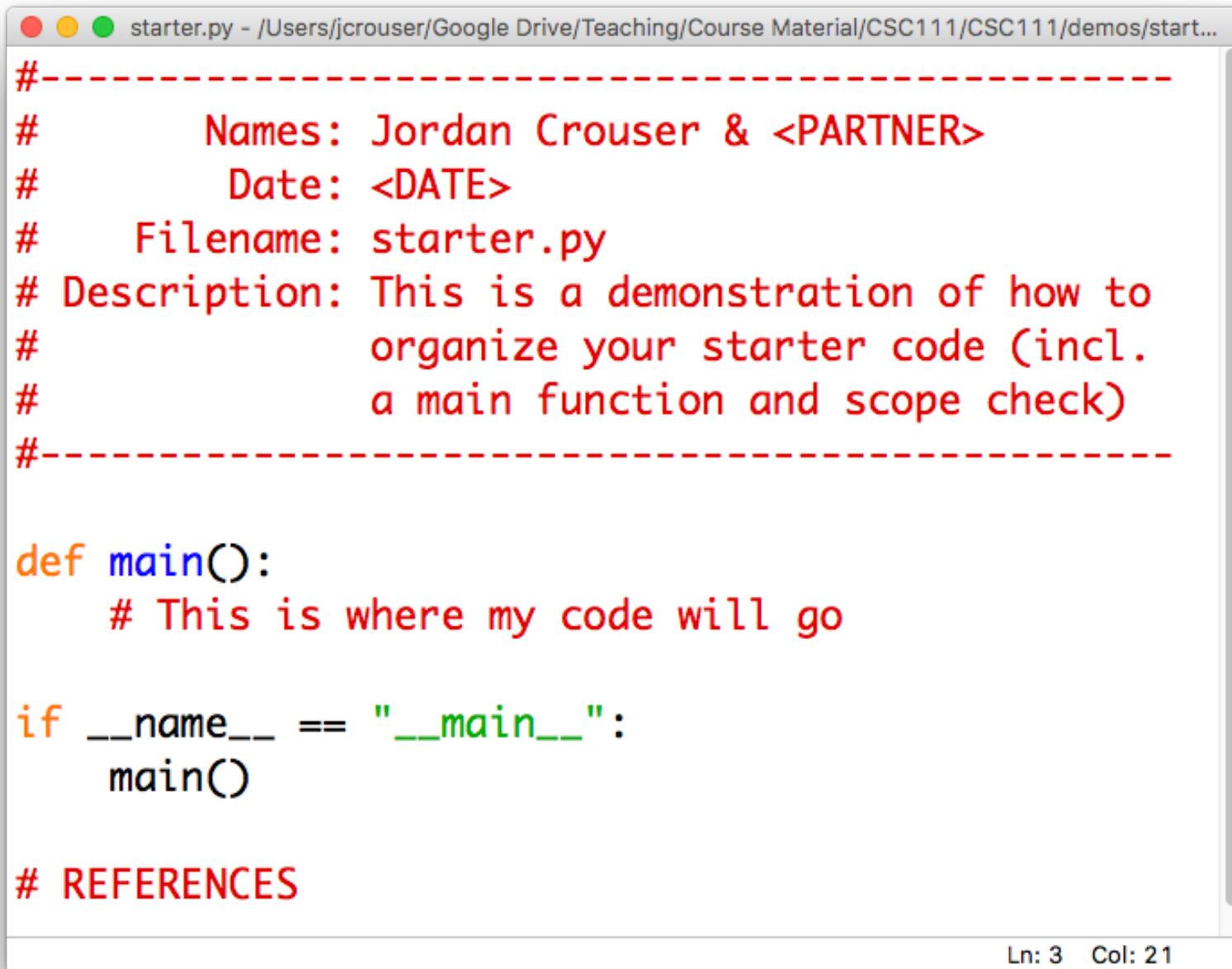
Takeaways

- 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



The screenshot shows a code editor window with a tab bar at the top labeled "starter.py - /Users/jcrouser/Google Drive/Teaching/Course Material/CSC111/CSC111/demos/start...". The code itself is a Python script with the following content:

```
#-----#
#      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
```

In the bottom right corner of the editor window, there is a status bar displaying "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