

Lecture 9:

# THE `main()` FUNCTION

---

CSC111: Introduction to CS through Programming

R. Jordan Crouser

Assistant Professor of Computer Science

Smith College

# Announcements

## INTRODUCTION TO GOOGLE CLOUD PLATFORM



ICE CREAM AND TREATS WILL BE SERVED  
Campus Center 204 | 10.01.2018 | 4:30 pm

Come to learn more about the platform, and  
get free credits and swag!



**Microsoft®**

New England Research  
& Development Center

Student Day @ the NERD on Monday 9/24

- RECAP?**
- Tour the facility, visit with Boston-area Smith alums, see cool demos, check out the maker space, win cool prizes!
  - Travel support / carpooling available
  - Interested? Fill out the form posted to Slack or talk to Asmita (your CS Department Liaison)

# About Assignment 2

- 92 / 92 assignments submitted on time (!!!)
- Almost everyone included a header and had appropriate variable names (nice work)
- Most submissions passed all automated tests on **integers**
- Currently working on testing **floats**
- A few notes on **attributing resources**:

# A reminder from the syllabus

- **Attribution:**
  - The names of all collaborating students should be listed at the top of the submission. [Note: this includes help from TAs.]
  - If you worked alone, please state: “*I did not collaborate with anyone on this assignment.*”
- **“References”**
  - In-line 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 10% of A2 submissions had one ☹

# 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

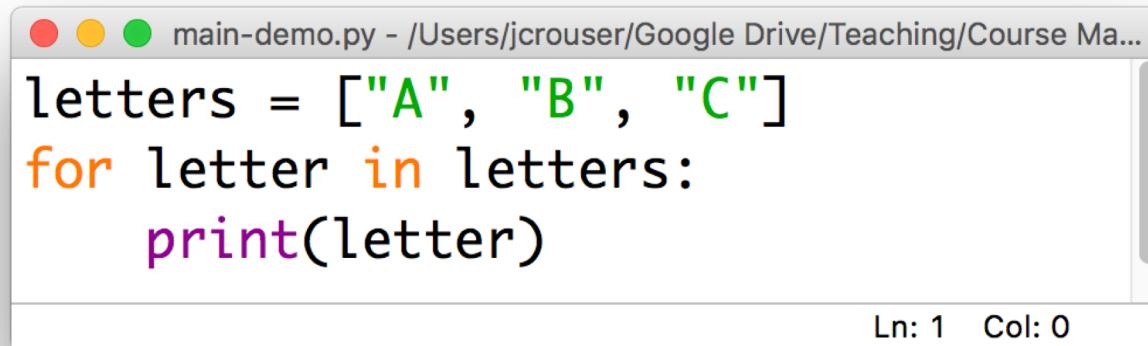
```
documentations.py - /Users/jcrouser/Google Drive/Teaching/Course Material/CSC111/CSC111/demos/documentations....  
#-----  
#      Names: Jordan Crouser (I did not collaborate  
#              with anyone on this assignment)  
#      Date: 26 September 2018  
#      Filename: demo.py  
# Description: 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
```

# Overview

- ✓ Recap of assignment 2
- ✓ Strings
  - ✓ operations on strings
  - ✓ accessing individual letters
  - ✓ handy methods
- The `main()` function
- Lab: Pretty Printing
- Life skill #2: debugging

# Recap

- 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:

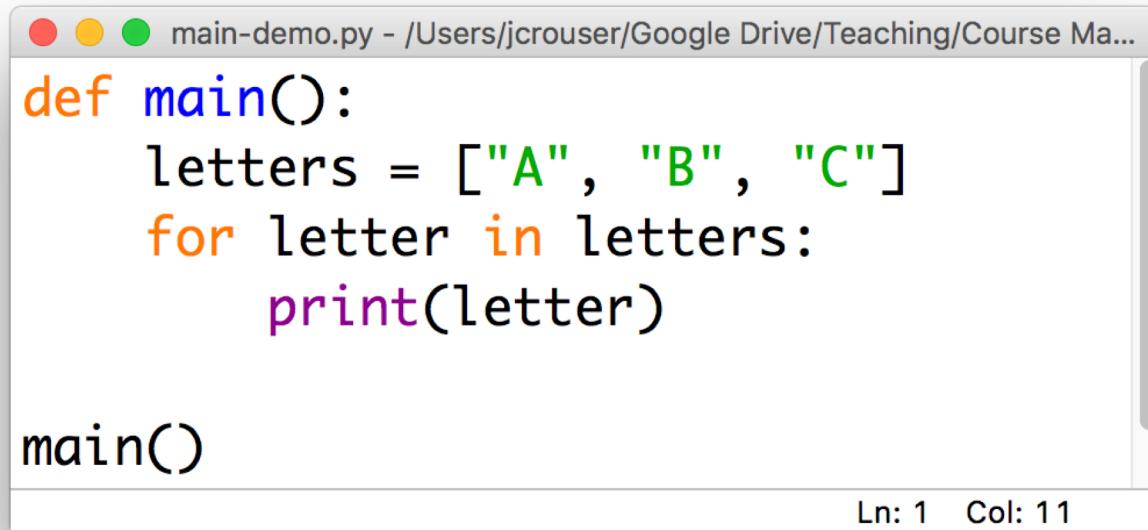
```
letters = ["A", "B", "C"]
for letter in letters:
    print(letter)
```

The status bar at the bottom right 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)

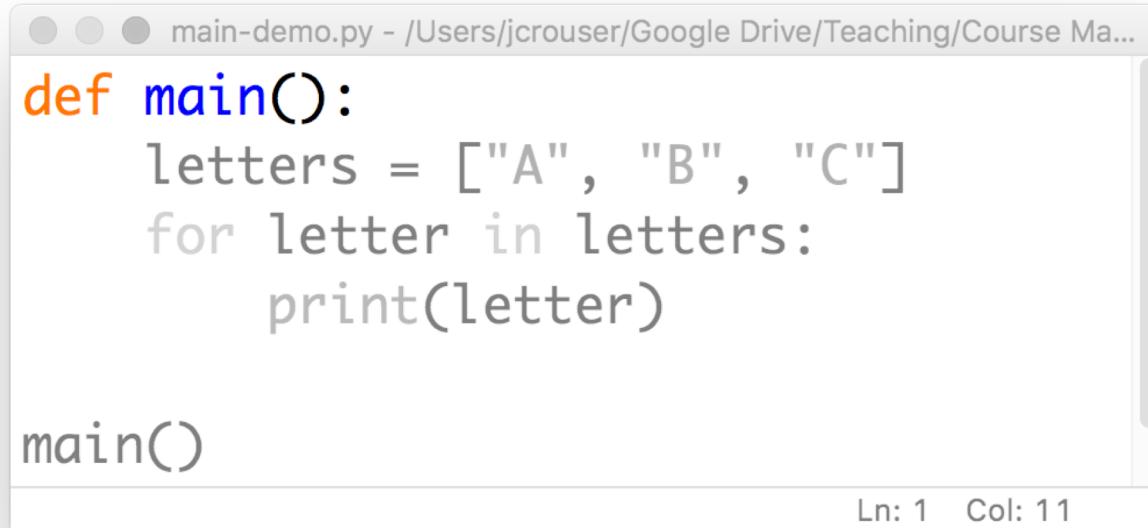
main()
```

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:

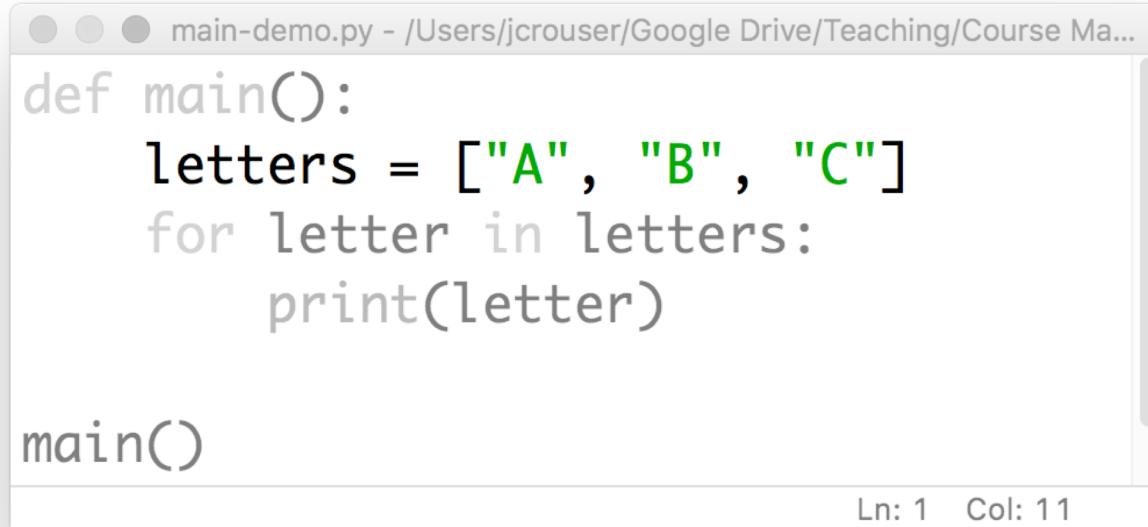
```
def main():
    letters = ["A", "B", "C"]
    for letter in letters:
        print(letter)

main()
```

The status bar at the bottom right of the editor 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:

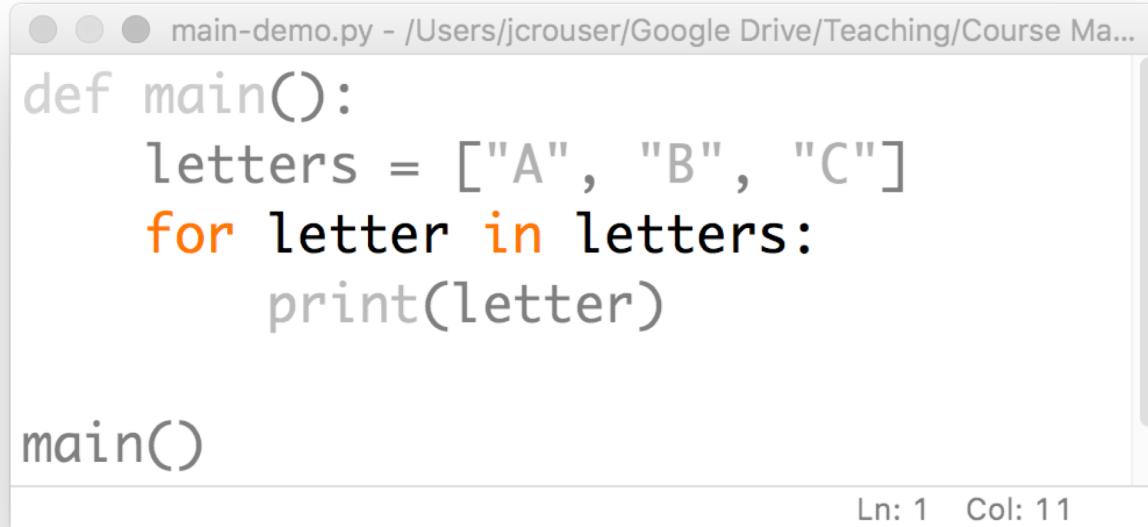
```
def main():
    letters = ["A", "B", "C"]
    for letter in letters:
        print(letter)

main()
```

The code editor shows the cursor at the beginning of the first line of the function definition. At the bottom right of the editor window, there is a status bar displaying "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)

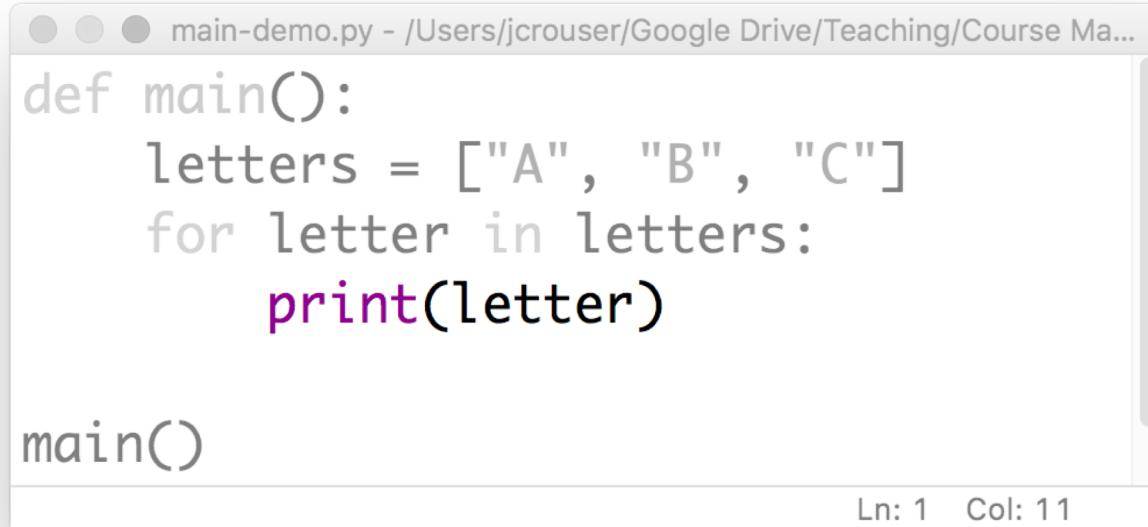
main()
```

The line "for letter in letters:" is highlighted in orange, indicating it is the current line being executed or selected.

At the bottom right of the editor window, there is a status bar with the text "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:

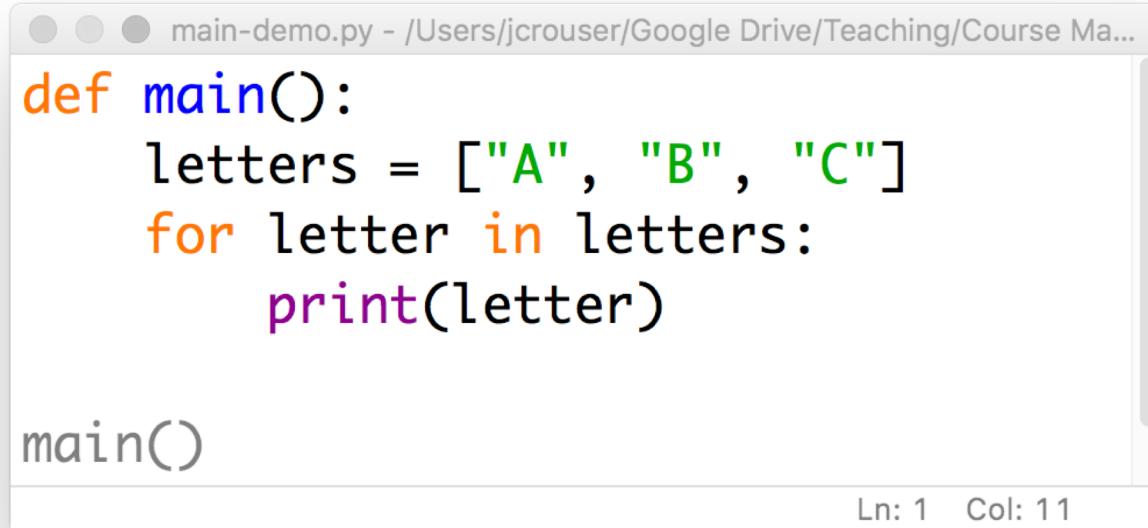
```
def main():
    letters = ["A", "B", "C"]
    for letter in letters:
        print(letter)

main()
```

The word "print" is highlighted in pink. At the bottom right of the editor 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()`



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)

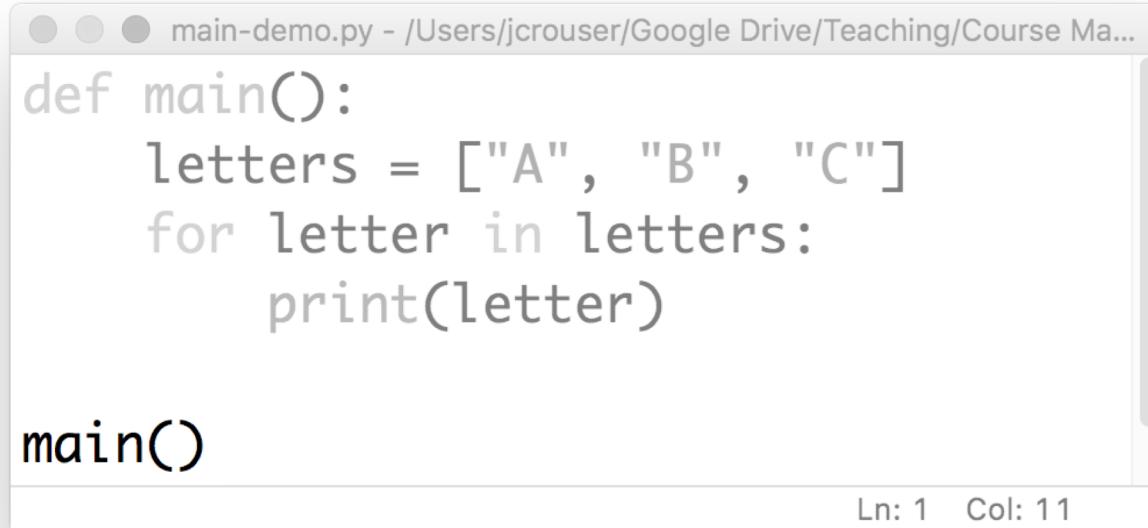
main()
```

The status bar at the bottom right shows "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 word "main()" is highlighted in the editor. In the bottom right corner of the editor window, it says "Ln: 1 Col: 11".

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

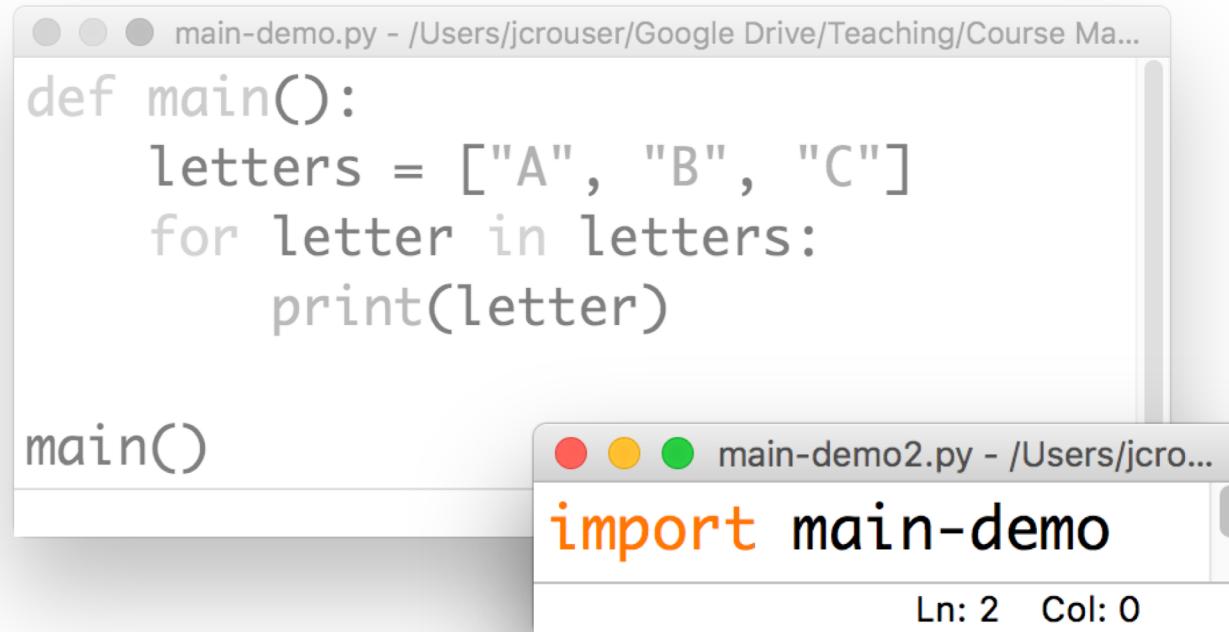
# Discussion

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

```
import main-demo
```

Both editors show the status bar at the bottom with 'Ln: 2 Col: 0'.

# Discussion

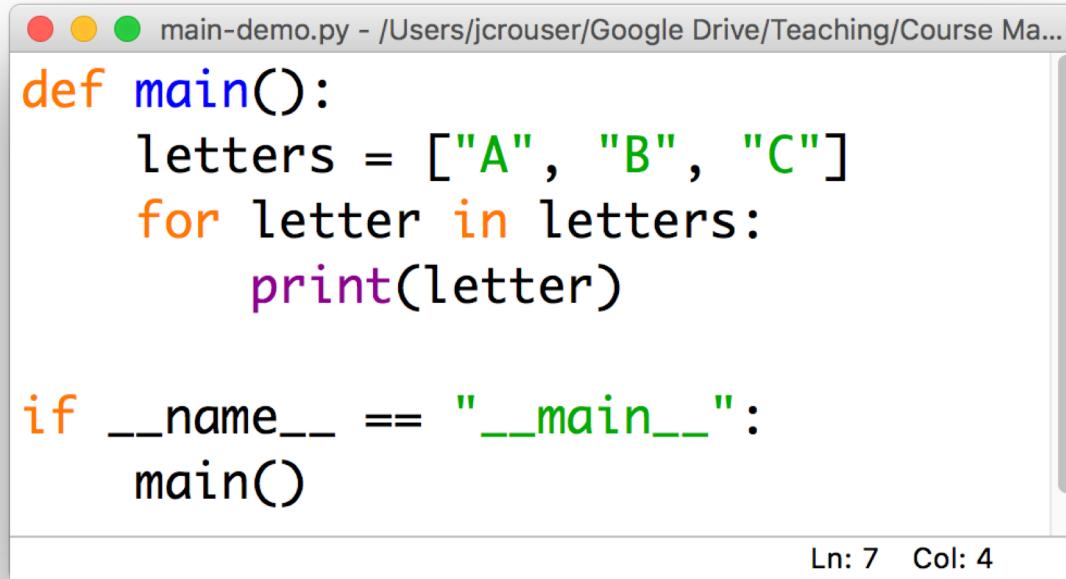
- **What we need:** a way to tell python to behave one way when we **run it as a “stand-alone” program**, and a different way when we **import** it

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



The image shows a screenshot of a Python code editor. The title bar reads "main-demo.py - /Users/jcrouser/Google Drive/Teaching/Course Ma...". The code itself is:

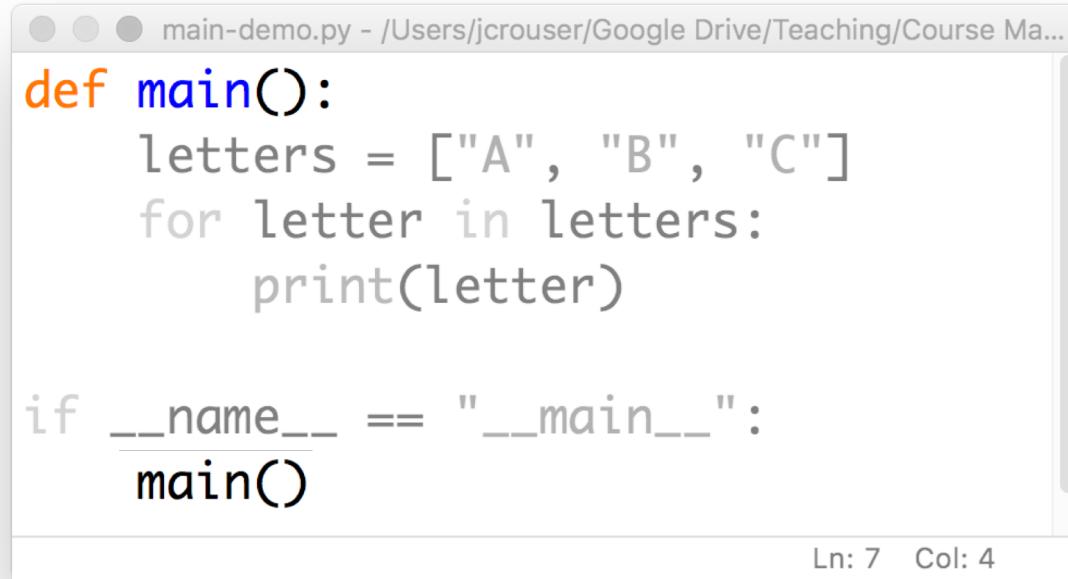
```
def main():
    letters = ["A", "B", "C"]
    for letter in letters:
        print(letter)

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

The code uses color-coded syntax highlighting: orange for **if**, **def**, and **for**; blue for **main()** and **\_\_name\_\_**; green for strings; and purple for **print**. The status bar at the bottom right indicates "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()**



The image shows a screenshot of a Python code editor. The title bar reads "main-demo.py - /Users/jcrouser/Google Drive/Teaching/Course Ma...". The code itself is:

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

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

At the bottom right of the code editor window, it says "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()
```

- (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 is [https://docs.python.org/3/library/\\_\\_main\\_\\_.html](https://docs.python.org/3/library/__main__.html). The page title is `__main__ — Top-level script environment`. The left sidebar contains links for "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. Quick search, Go, The Python Software Foundation is a non-profit corporation. Please donate, Last updated on Sep 26, 2018. Found a bug?, Created using Sphinx 1.7.6.

D E M O

T Y M E

# 15-minute exercise

- Write a program that contains a **main()** function, which contains instructions for printing out the phrase:  
**"Today is not mountain day :-( "**
- 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:

**"Maybe today...?"**

# 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

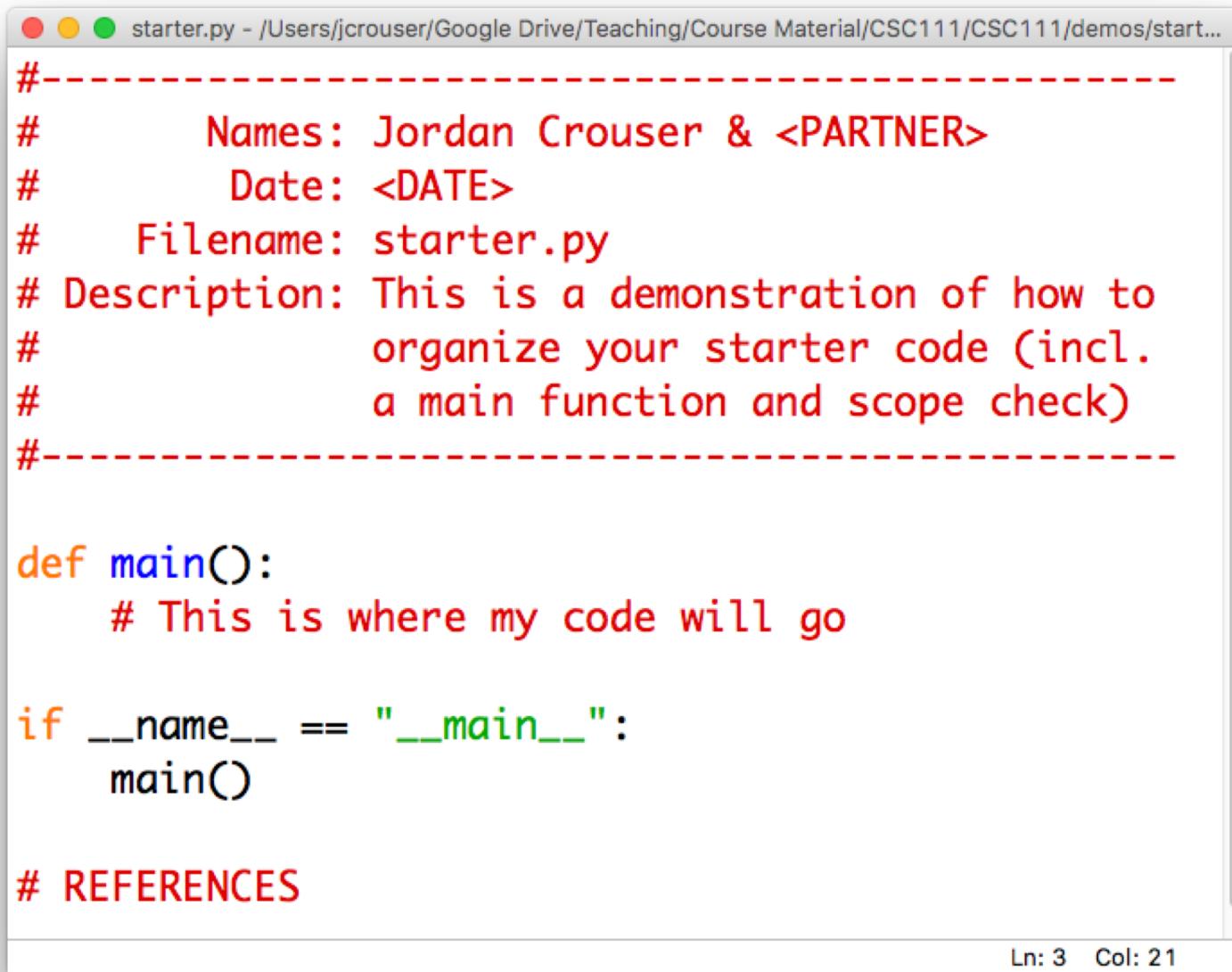
# Takeaways

- 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



A screenshot of a Mac OS X terminal window titled "starter.py - /Users/jcrouser/Google Drive/Teaching/Course Material/CSC111/CSC111/demos/start...". The window displays a Python script with syntax highlighting. The code includes a multi-line comment at the top with metadata, a main function definition, an if \_\_name\_\_ == "\_\_main\_\_": block, and a "# REFERENCES" section at the bottom.

```
#-----
#      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

# Overview

- ✓ Recap of assignment 2
- ✓ Strings
  - ✓ operations on strings
  - ✓ accessing individual letters
  - ✓ handy methods
- ✓ The `main()` function
- Lab: Pretty Printing
- Life skill #2: debugging