Week 2: How Programs Run

And also a review of Week 1

More on Submitting Assignments

- Github will still be used for posting projects, sharing materials including files and example scripts, etc…
- For this week, either:
  - Send me an email with your assignments (.ipynb)
  - Share it with me through Google (Gmail in syllabus)
- And finally, feel free to help each other out with assignments!
  - Would there be any interest in an online forum or group chat?

Overview of the Week

- **Mondays**: Review of previous week and lecture on this week’s contents
- **Wednesdays**: Time to work on the assignment of the week, live-coding examples
- **Fridays**: Assignment of the week due

Final Notes

- Zoom Ground rules?
    - Eating is fine
    - Use the participation tools - I’ll try my best to keep an eye on them!
    - Feel free to unmute or interject with questions

Week 1: Review

Week 1: Variables, Data Types, and Assignment

- Four primary data types of Python: integers (ints), floats, booleans, and strings
  - There are more, but those are considered more complex, most of which are built-in to Python but not necessarily other languages.
- Operators: Numerical operators, logical operators, and comparisons
- Syntax for declaring variables and how to use expressions to perform operations on data

Week 2: Flow of Execution & Functions

By the end of today’s session, you should be able to...

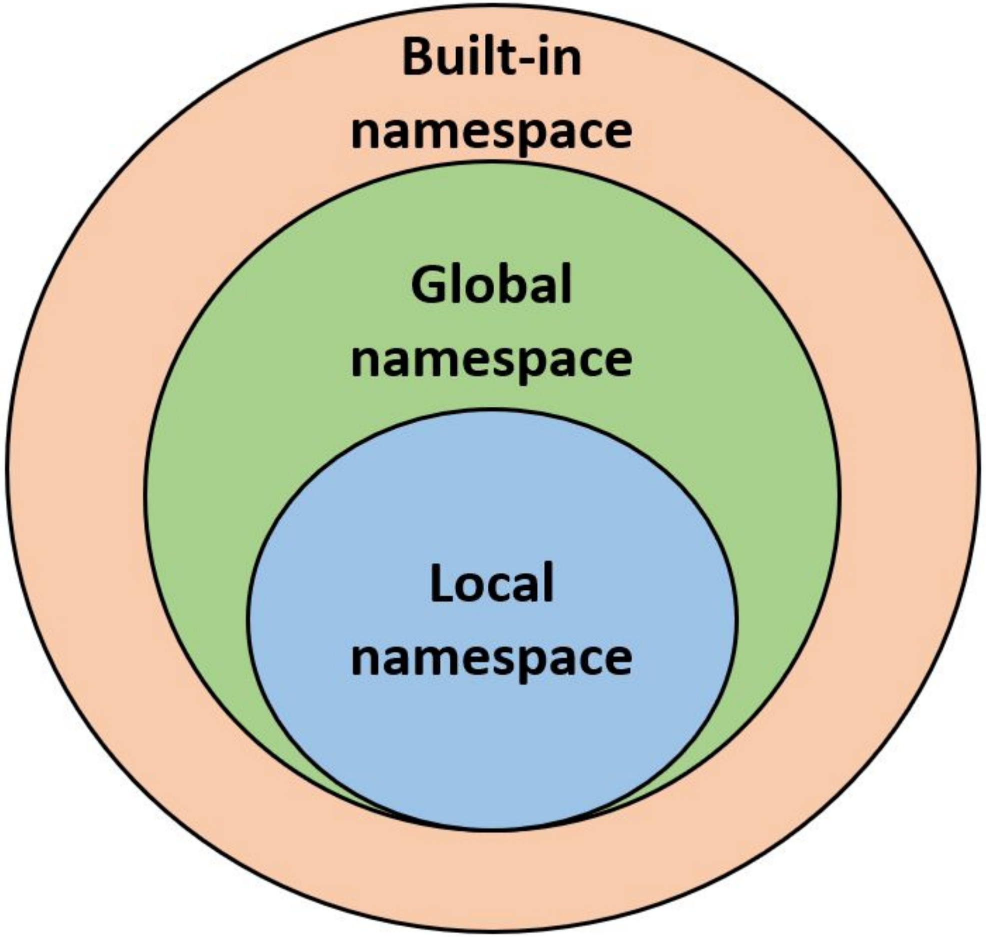
1. Understand how functions, whitespace, and scope may change how the interpreter sees code
2. Set up a function

Flow of Execution

- Python executes code **line by line** using an interpreter (Week 1)
- To learn more about how programs execute, we need to learn about **whitespace**, **functions**, and **scope**…
- Knowing how code executes and how functions, whitespace, and scope may change that is essential for things like…
    - Debugging!
    - Writing smooth and efficient code

Scope

- “**Built-in**”: This is all the built-in keywords/functions the Python interpreter already knows - like print() or ‘and’. They are accessible anywhere.
- “**Global**”: These are variables/functions you might create or **import** from a library which are accessible from anywhere.
- “**Local**”: These are variables/functions that only exist in local spaces such as in functions or within a for-loop.



Type of Namespaces

Functions

Here are three important terms to know when referring to a function:

1. The function **header**.
   a. Contains the function name and function parameters
2. The function **body**.
   a. Contains the ‘instructions’ of the function, any documentation, and any return statements
3. The function **call**.
   a. This is how you use a function

Function Analogy

You can think of a python function like a mathematical function. For example:

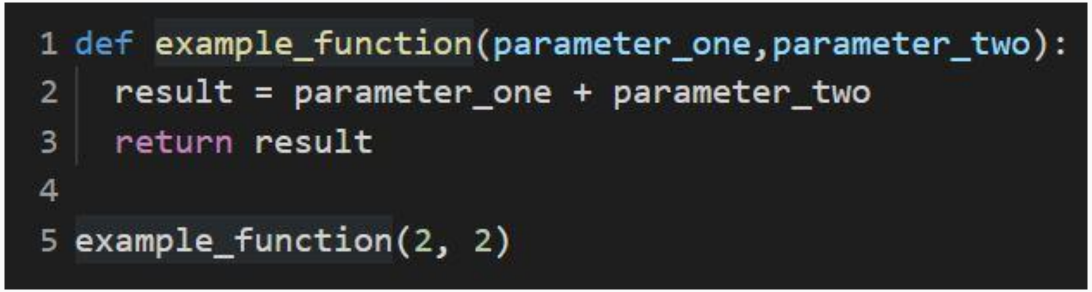
f(x,y) = 2x + 2y

- **f** is the function name. The function could have just as easily been called ‘**g**’ or ‘**h**’.
- (x,y) are the function parameters. This is input or arguments the function receives and will ‘do something’ with.
- 2x + 2y is the function body. It tells you what the function will do with its given inputs
- The evaluation of “2x + 2y” is the function return value. It is the output produced by the function.
- All of the above make up the **function definition**.

Python Syntax for Functions

Function header →
Function body →

Function call →



```
1 def example_function(parameter_one,parameter_two):
2   result = parameter_one + parameter_two
3   return result
4
5 example_function(2, 2)
```

- Line 1: Function header
  - ‘def’ is a python keyword which marks the beginning of a function
  - After ‘def’ is where you place the function **name** and **parameters**
  - Colon ‘:’ marks the end of the function ‘header’ and the start of the function ‘body’
- Line 2-3: Function body
  - Python statements that define what the function does with its given parameters
  - Notice how each line that is a part of the function body is indented
  - Line 3 uses the keyword ‘return’ which indicates the function produces an output
- Line 5: Function **call**
  - Notice the how the function call is not indented with the function body

“Fruitful” functions versus void functions

- A function which uses the keyword **return** is a “fruitful” function, meaning it produces a value
- Some functions perform actions but do not produce a value to return - these are called void functions.
- Since fruitful functions return a value, you can assign fruitful functions to a variable like such…
    - x = example_function(2,2)
    - square = evaluate_square(3) # this would assign 9 to the variable ‘square’
- Void functions return no value - which is indicated by a special value called ‘None’

Do functions alter the flow of execution?

- Python reads your code line by line
- Functions do not interfere with that, but…
- **Statements inside a function are not executed until the function is called**.

One last word on whitespace and indentations

Python distinguishes one “section” of code from another using indentations

- Recall functions:

```
1 def example(parameter):
2       result = parameter + 2
3       return result

4 print(example(2))
```

```
Output:
4
```

- You would not be able to refer to the variable ‘result’ outside of the function ‘example’
- This is because it is in the function body, indicated with indentations
- The indentations tell python the variable ‘result’ belongs in a different namespace than lines 1 or 4

To-do by Wednesday:

- Week 2 will be posted in an hour
- Look over and get Week 2 started
- Come to Wednesday prepared with any questions and Colab open in another tab!