COMP110

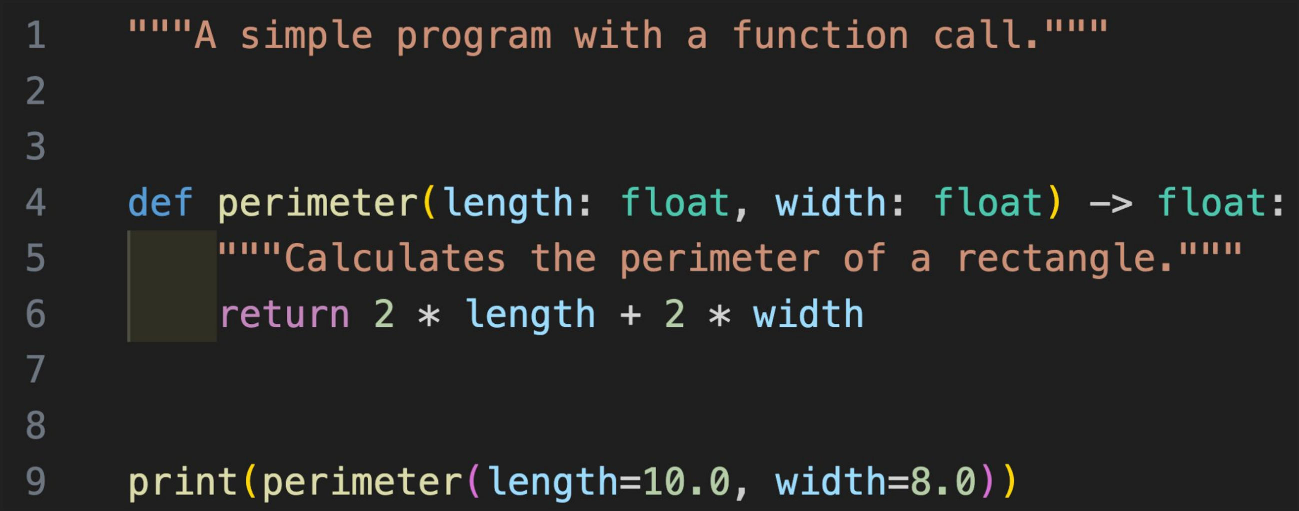
Functions and Memory Diagrams

Please be ready to write on a piece of paper or tablet!

Announcements

- Reminder: Labor Day on Monday; no class!
  - Office Hours and Tutoring canceled on Sunday and Monday
- Quiz 00 on Friday
  - Ways to prepare:
    - Quiz expectations on course site
    - Practice problems
    - Office Hours: 11am-5pm today
    - Tutoring: 5-7pm today
    - Hybrid review session: 6-7pm today in FB009 and online
- CQ00: Memory Diagram
  - Upload a .pdf of the memory diagram we do in class by 11:59pm

Warm-up: write down at least one line number for each:



```python
1   """A simple program with a function call."""
2
3
4   def perimeter(length: float, width: float) -> float:
5       """Calculates the perimeter of a rectangle."""
6       return 2 * length + 2 * width
7
8
9   print(perimeter(length=10.0, width=8.0))
```

1. Docstring
2. Function call(s)
3. Return statement
4. Function definition
5. Arguments
6. Use of a parameter's name in an *expression*

The `return` statement vs. calls to `print`

- **The return statement is *for your computer*** to send a result back to the function call's "bookmark" *within your program*
  - A bookmark is dropped when you *call* a function with a return type. When that function's body reaches a *return statement*, the returned value replaces the function call and the program continues on
- **Printing is *for humans* to see**. To share some data with the user of the program, you must output it in some way
- If you have a function, **`my_func`**, that returns some value, you can print the value it returns by:
  1. Printing its return value directly with **`print(my_func())`** or
  2. (Later in the course) by storing the returned value in a variable and *later* printing the variable

Tracing programs by hand: Intro to memory diagrams!

- The evaluation of a program depends on many interrelated values
- As any non-trivial program is evaluated, what needs to be kept track of includes:
  1. The current line of code, or expression within a line, being evaluated
  2. The trail of function calls that led to the current line and “frame of execution”
  3. The names of parameters/variables and a map of the values they are bound to
  4. More!
- As humans, this quickly becomes more information than we can mentally keep track of.
  - Good news: Memory diagrams will help you keep track of it all on paper!

Memory diagrams

- A program’s runtime *environment* is the mapping of *names* in your program to their *locations* in memory
- A program’s *state* is made up of the *values stored* in those locations
- You can use memory diagrams to visually keep track of both the environment and its state
- Memory diagrams will help you keep track of how function calls are processed.
    - Where was the function called?
    - What was the return value, and where was it returned to?
    - (and more!)

```python
"""A simple program with a function call."""


def perimeter(length: float, width: float) -> float:
    """Calculates the perimeter of a rectangle."""
    return 2 * length + 2 * width


print(perimeter(length=10.0, width=8.0))
```

```python
"""A program with *two* function calls."""

def perimeter(length: float, width: float) -> float:
    """Calculates the perimeter of a rectangle."""
    return 2 * length + 2 * width

def square_perimeter(side: float) -> float:
    """Calculates the perimeter of a square."""
    return perimeter(length=side, width=side)

print(square_perimeter(side=4.0))
```

(a second example, in case you want to try this on your own!)

CQ00: Submitting the memory diagram to Gradescope

From your phone:

1. Open the CQ00 assignment and make a submission
2. Upload a photo/PDF of your memory diagram
3. Complete your submission (and please make sure your image is in the correct orientation!)