



## CL03: Introduction to Memory Diagrams

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

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

```
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*

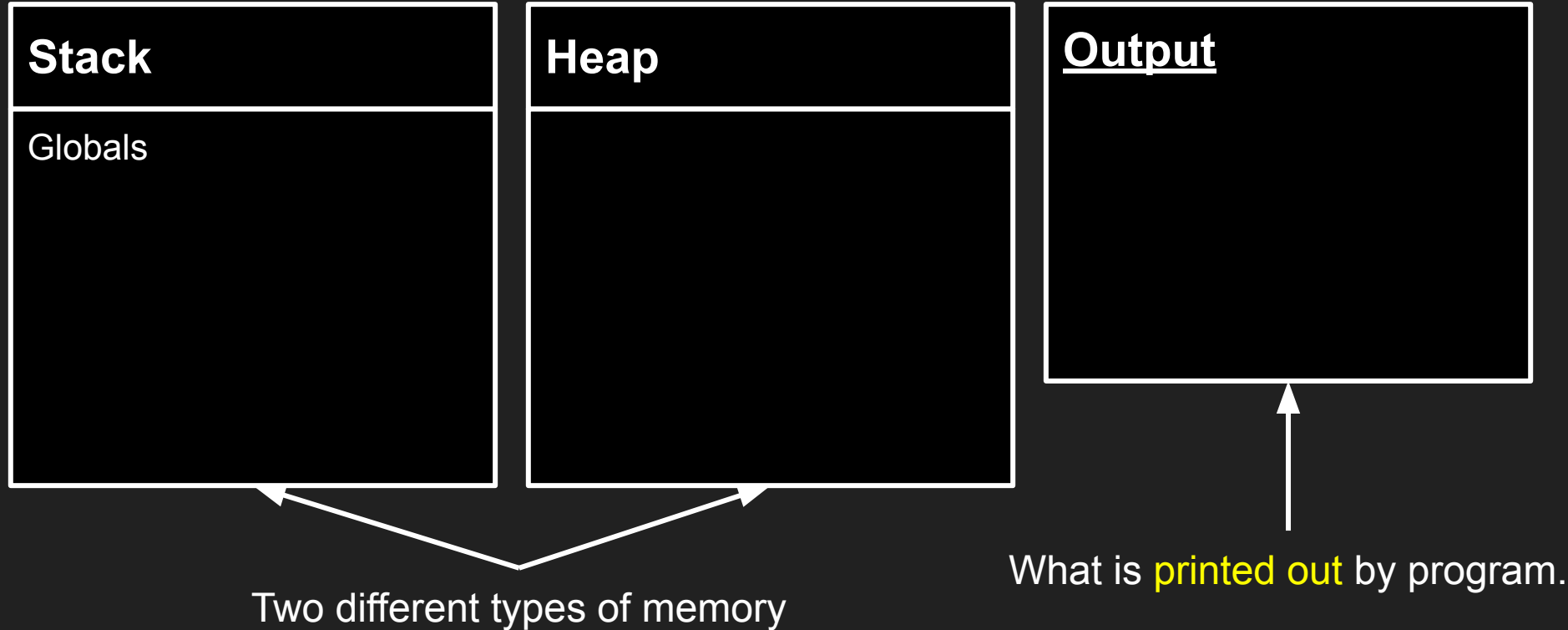
# The `return` statement vs. calls to `print`

- **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

# Motivation

- As a program is evaluated, what needs to be kept track of includes:
  - What am I currently doing?
    - The current **line of code**, or expression within a line, being evaluated
    - The names of **parameters/variables** and their value assignments
  - Where did I come from/Where do I go next?
    - The trail of **function calls** that led to the current line and “frame of execution”
- **Memory diagrams** help us to trace code in memory
- Helps us to understand **what** our code is doing and **why**

# Memory Diagram Components



# Stack vs. Heap

- Stack: variables, primitive types
- Heap: definitions, certain mutable types (more on this later)

```
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))
```

Question: How would my diagram differ if my last line didn't call `print()` ?

## Stack

Globals

## Heap

## Output

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

Question: How would my diagram differ if my last line didn't call `print()` ?

## Stack

Globals

## Heap

## Output

# Function Call Frame Anatomy

- Return Address tracks the line on which the function was *called*
  - This answers the question of “Where did I come from/Where do I go next?” because it tells us where to go when our function returns
- Local Variables track the variables defined in the scope of the function call
- Return Value is the value that is returned by our function
  - Recall that a function call is an expression.  
The return value is what the function call *evaluates* to.

# Function Call Steps

- Establish the frame
  - Add frame on stack labeled with function name
  - Add Return address
  - Copy over arguments (these become our local variables)
- Enter the body of the function definition and follow the code logic until you **return**

```
1  """Practice with function writing."""
2
3  ✓ def silly_sum(num1: int, num2: int) -> int:
4      num3: int = num1
5      num1 = num2
6      num2 = num3
7      num3 = num3 + 1
8      return num1 + num2
9
10  print(silly_sum(num1=4, num2=5))
```

## Stack

Globals

## Heap

## Output

```
1 def get_tax(price: int, tax_rate: float) -> float:
2     |     return price * tax_rate
3
4 def total_cost(cost: int, tax: float) -> float:
5     |     return cost + get_tax(price=cost, tax_rate=tax)
6
7 print(total_cost(cost=100, tax=0.07))
```

## Stack

Globals

## Heap

## Output

## More Practice

- On the course site go to [Resources](#) > [Practice Memory Diagrams](#)
- Quiz practice questions on course site
- Review session tomorrow (Wednesday) at 6 pm
  - Link on course site
  - Link should be replaced with recording after (*be patient!*)