[301] Creating Functions

Tyler Caraza-Harter

Learning Objectives Today

Function syntax:
- basics, return, tabbing

Input/output:
- parameters
- three types of arguments
- print vs. return

Tracing:
- What happens when?
- PythonTutor

Please continue reading
Chapter 3 of Think Python

Also read 301 bonus:
"Creating Fruitful Functions"

link on schedule

Main Code:
1. Put 2 in the "moves" box
2. Perform the steps under "Move Code", then continue to step 3
3. Rotate the robot 90 degrees to the right (so arrow points to right)
4. Put 3 in the "moves" box
5. Perform the steps under "Move Code", then continue to step 6
6. Whatever symbol the robot is sitting on, write that symbol in the "resut" box

Move Code:
A. If "moves" is 0, stop performing these steps in "Move Code", and go back to where you last were in "Main Code" to complete more steps
B. Move the robot forward one square, in the direction the arrow is pointing
C. Decrease the value in "moves" by one
D. Go back to step A

how do we write functions
like move code?

**Functions are like "mini programs",
as in our robot worksheet problem**
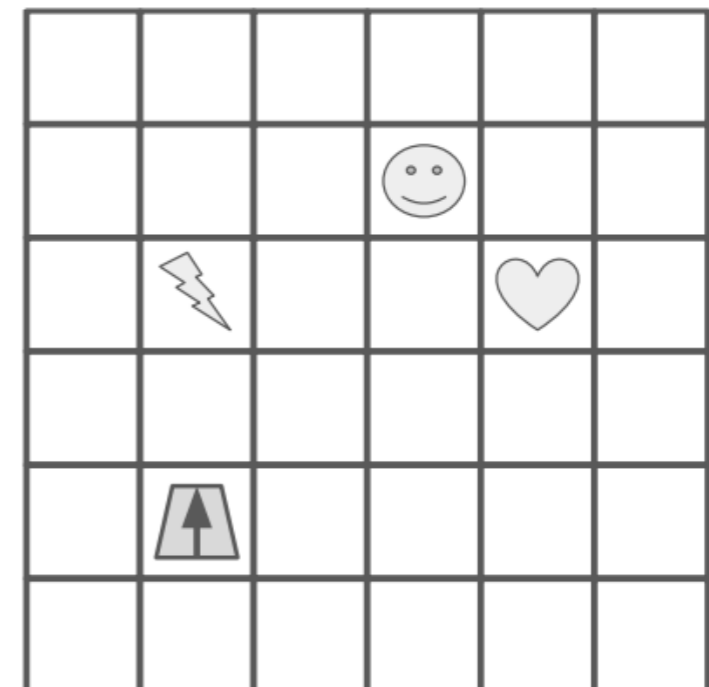
Types of functions

Sometimes functions **do** things
- Like "Move Code"
- May produce output with print
- May change variables

Sometimes functions **produce** values
- Similar to mathematical functions
- Many might say a function "returns a value"
- Downey calls these functions "fruitful" functions
  (we'll use this, but don't expect people to generally be aware of this terminology)

Sometimes functions do both!

Types of functions

Sometimes functions **do** things
- Like "Move Code"
- May produce output with print
- May change variables

Sometimes functions **produce** values
- Similar to mathematical functions
- Many might say a function "returns a value"
- Downey calls these functions "fruitful" functions
  (we'll use this, but don't expect people to generally be aware of this terminology)

Sometimes functions do both!

Math to Python

Math: $f(x) = x^2$

Python:
```python
def f(x):
    return x ** 2
```

Math to Python

Math: $f(x) = x^2$

Python:
```
def f(x):
    return x ** 2
```

Function name is "f"

Math to Python

Math: $f(x) = x^2$

Python:
```
def f(x):
    return x ** 2
```

It takes one parameter, "x"

Math to Python

Math: $\boxed{f(x) =}\ x^2$

Python:

```
def f(x):
    return x ** 2
```

① In Python, start a function definition with "def" (short for definition), and use a colon (":") instead of an equal sign ("=")

Math to Python

Math: $f(x) = \boxed{x^2}$

Python:
```
def f(x):
    return x ** 2
```

2 In Python, put the "return" keyword before the expression associated with the function

Math to Python

Math: $f(x) = x^2$

Python:
```
def f(x):
    return x ** 2
```

3 In Python, indent before the statement(s)

Math to Python

Math: $g(r) = \pi r^2$

Python:

```
def g(r):
    return 3.14 * r ** 2
```

4   Computing the area from the radius

Math to Python

Math: $g(r) = \pi r^2$

Python:
```
def get_area(radius):
    return 3.14 * radius ** 2
```

5 In Python, it's common to have longer names for functions and arguments

Math to Python

Math: $g(r) = \pi r^2$

Python:
```python
def get_area(diameter):
    radius = diameter / 2
    return 3.14 * radius ** 2
```

6 It's also common to have more than one line of code (all indented)

Can we implement our own version of popular math functions?

```
abs(x)
```

```
sqrt(x)
```

```
pow(base, exp)
```

demos...

Rules for filling parameters...

function declaration

```
def foo(x, y=-1):
```

x = ???

y = ???

not actual code, but imagine parameters as variables that are automatically initialized for you

function invocation
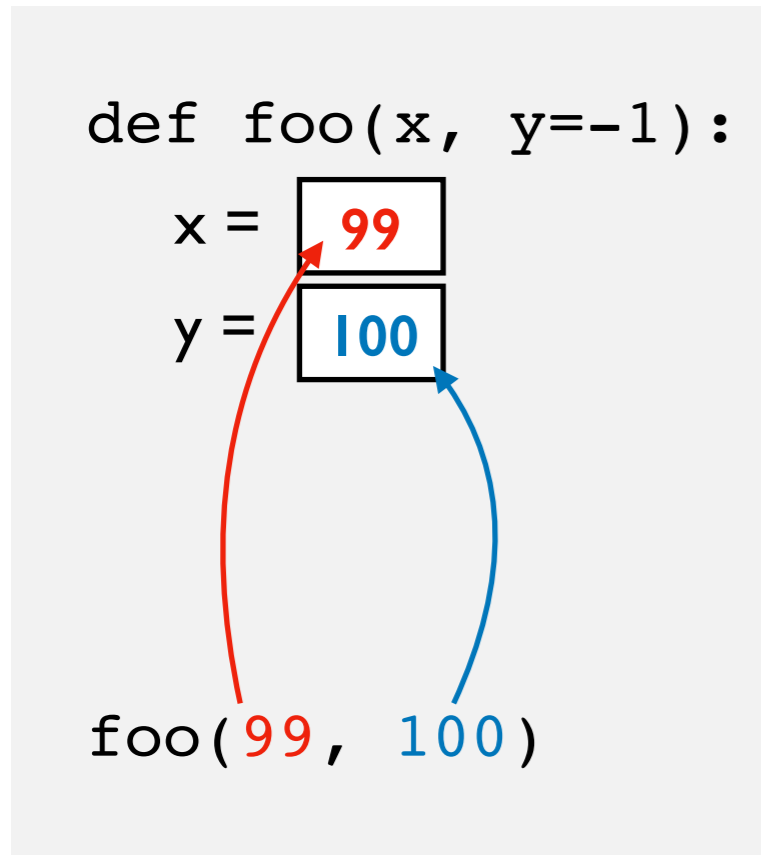
```
foo(99, 100)
```

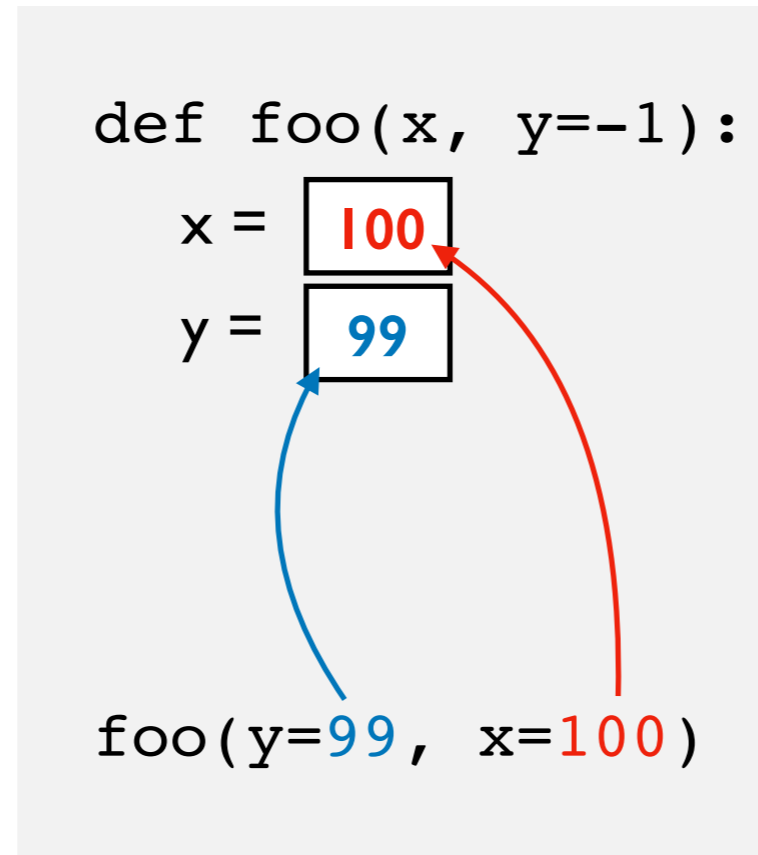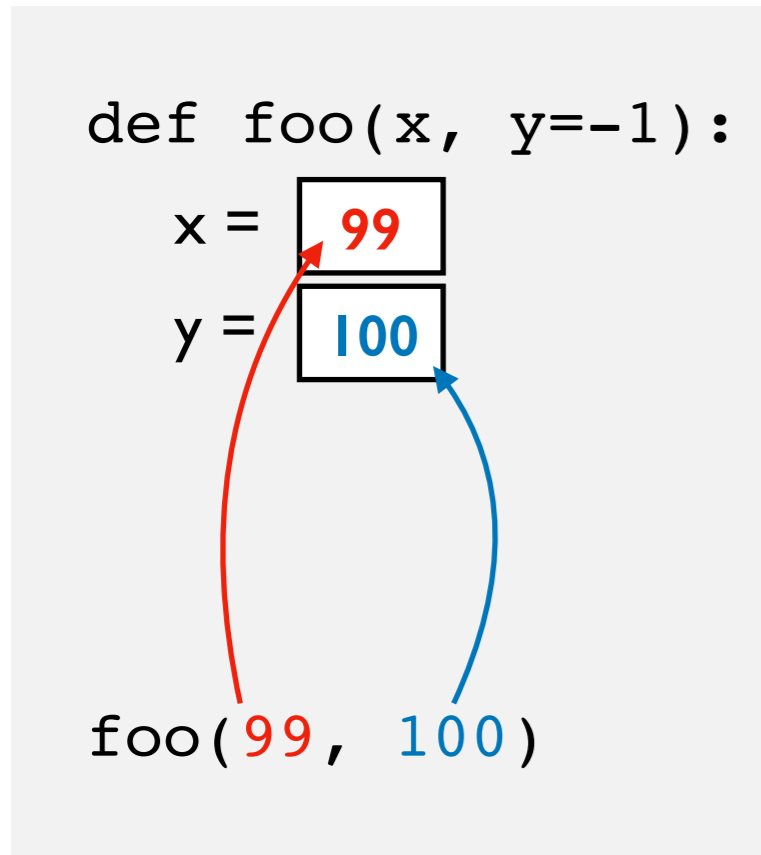**1** positional arguments

Rules for filling parameters...

```
def foo(x, y=-1):
    x =  99
    y =  100
```

foo(99, 100)

1 positional arguments

Rules for filling parameters...

```
def foo(x, y=-1):
    x =  99
    y =  100



    foo(99, 100)
```

```
def foo(x, y=-1):
    x =  100
    y =  99



    foo(y=99, x=100)
```

1 positional arguments   **2** keyword arguments

Rules for filling parameters...

```
def foo(x, y=-1):
    x = [ 99 ]
    y = [ 100 ]



    foo(99, 100)
```

```
def foo(x, y=-1):
    x = [ 100 ]
    y = [ 99 ]



    foo(y=99, x=100)
```

```
def foo(x, y=-1):
    x = [ 99 ]
    y = [ -1 ]



    foo(99)
```
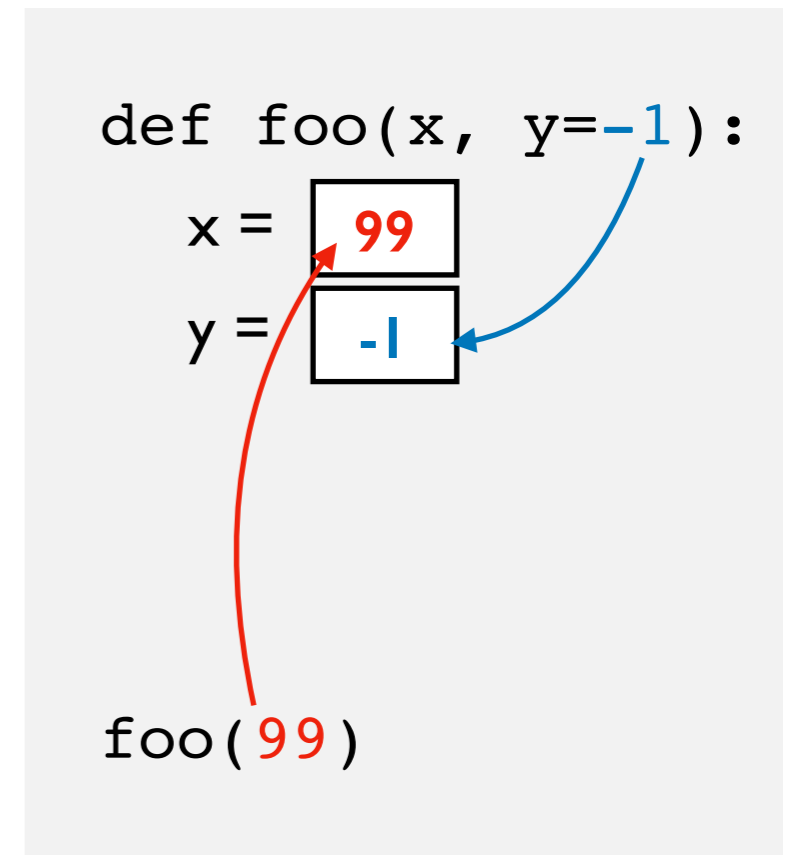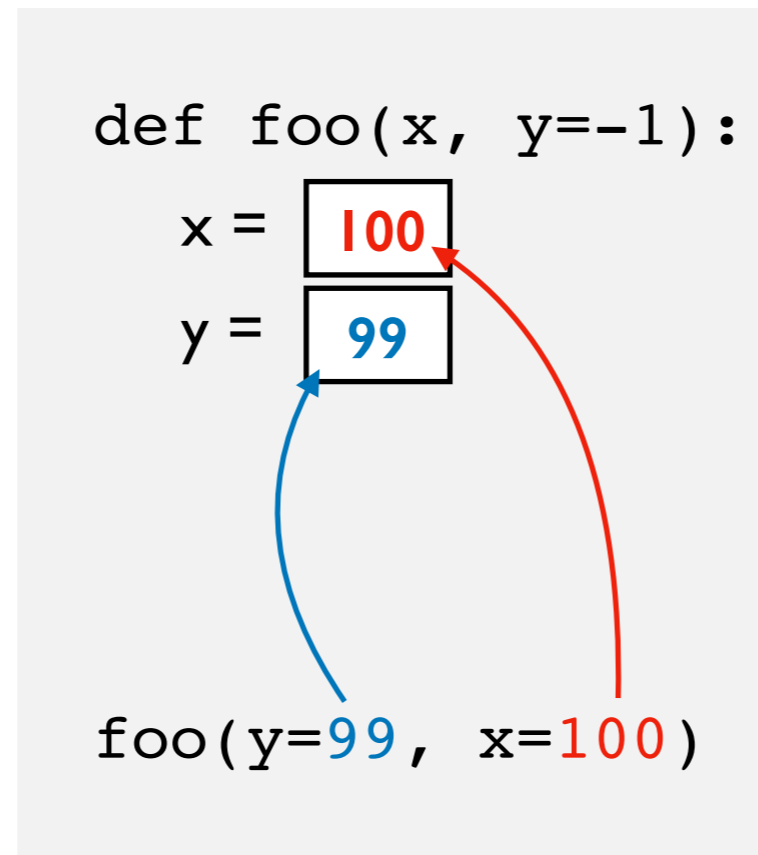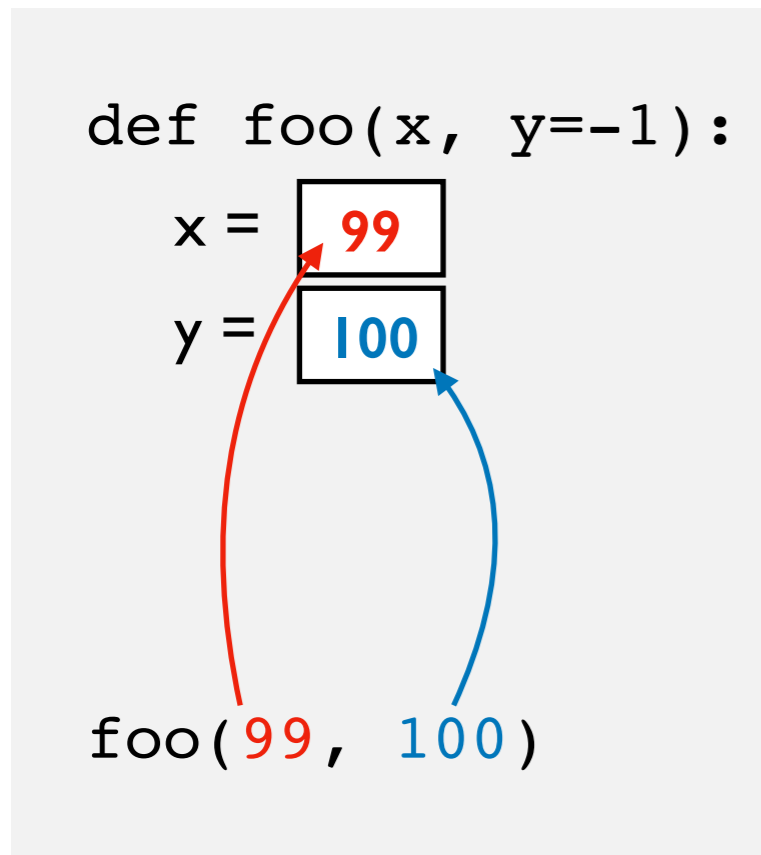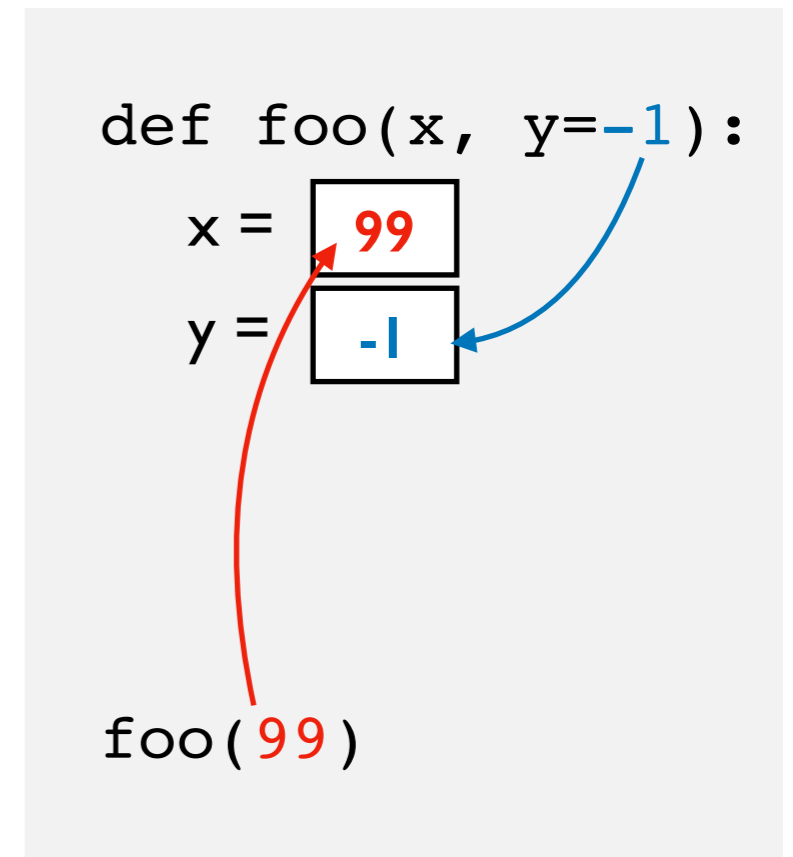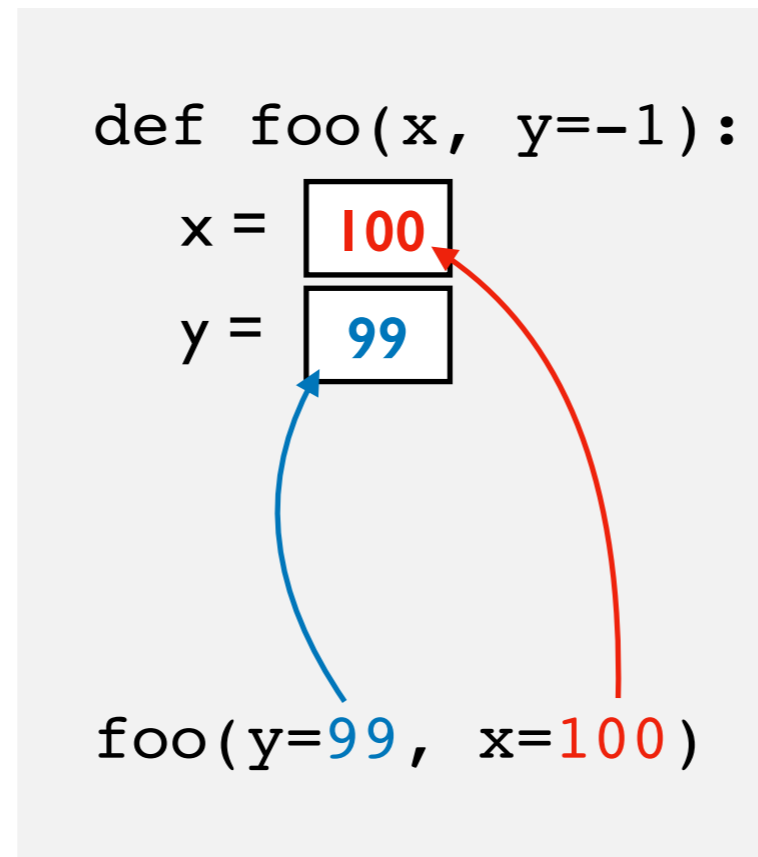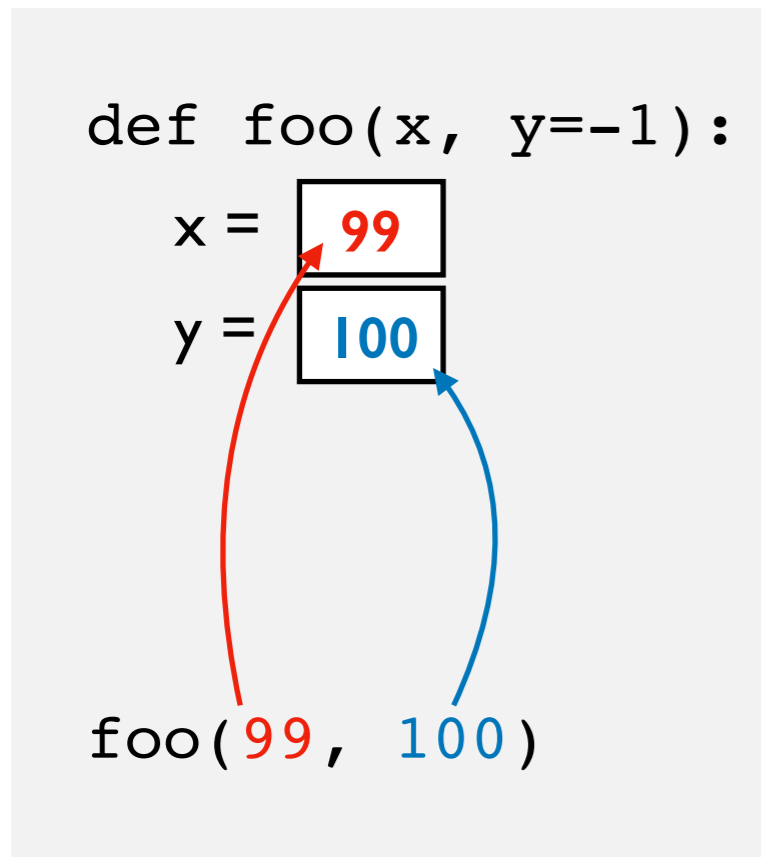
1 positional arguments **2** keyword arguments **3** default arguments

common pitfall: *confusing keyword arguments and default arguments*

Rules for filling parameters...


```
def foo(x, y=-1):
    x = [ 99 ]
    y = [ 100 ]


    foo(99, 100)
```

```
def foo(x, y=-1):
    x = [ 100 ]
    y = [ 99 ]


    foo(y=99, x=100)
```

```
def foo(x, y=-1):
    x = [ 99 ]
    y = [ -1 ]


    foo(99)
```

1 positional arguments   **2** keyword arguments   **3** default arguments

worksheet practice...

pre-installed (e.g., math)
- sqrt()
- sin(), cos()
- pi, *etc.*

built in
- input()
- print()
- len()
- *etc.*

Where do **modules** *come from?*
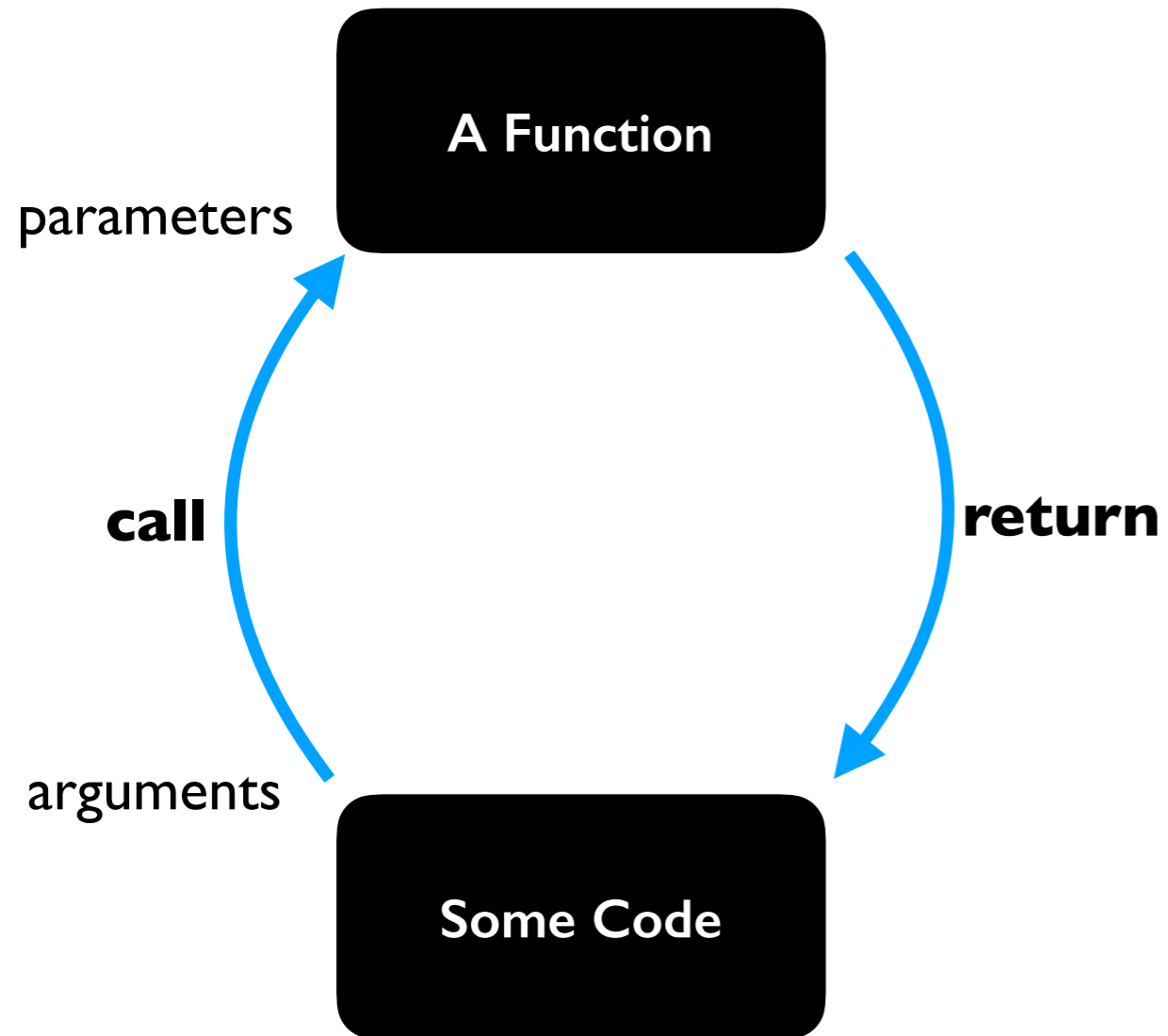
installed (e.g., jupyter)
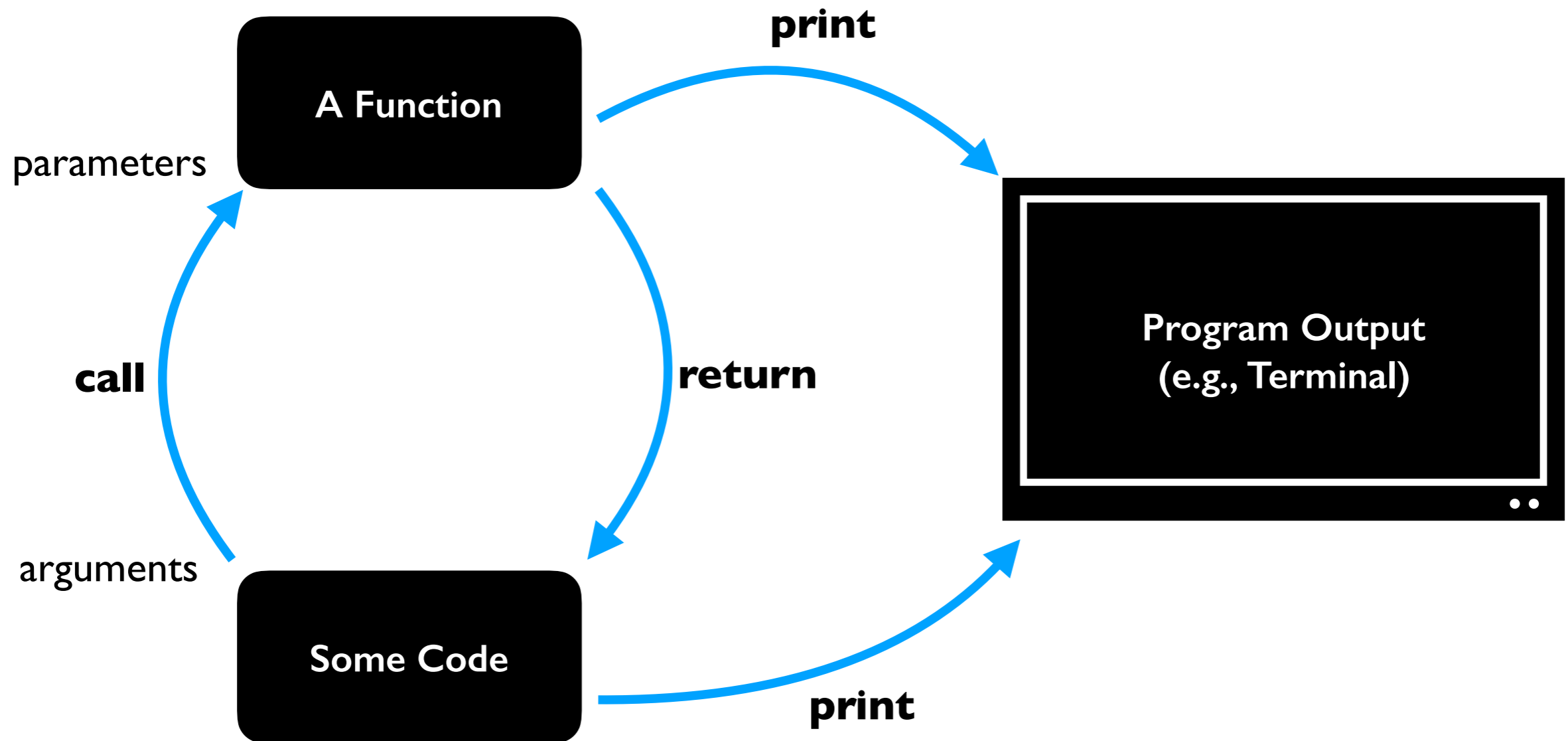- pip install jupyter
- pip install ...

custom
- dog
- cat
- ...

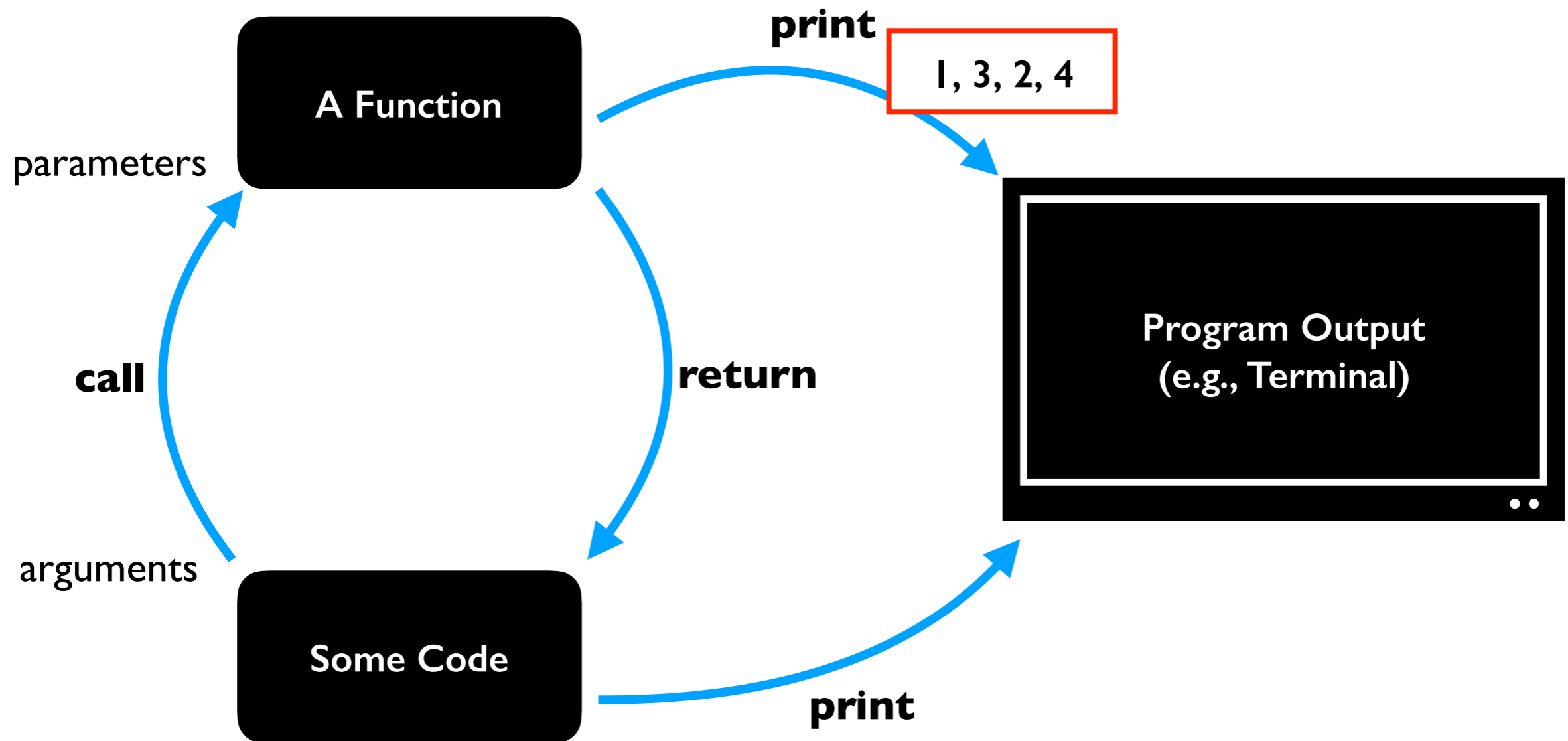*demos...*

Print vs. Return

Print vs. Return

A Function

parameters

call

arguments

Some Code

print

return

print

**Program Output
(e.g., Terminal)**

we could call print from multiple places

Print vs. Return

Print vs. Return
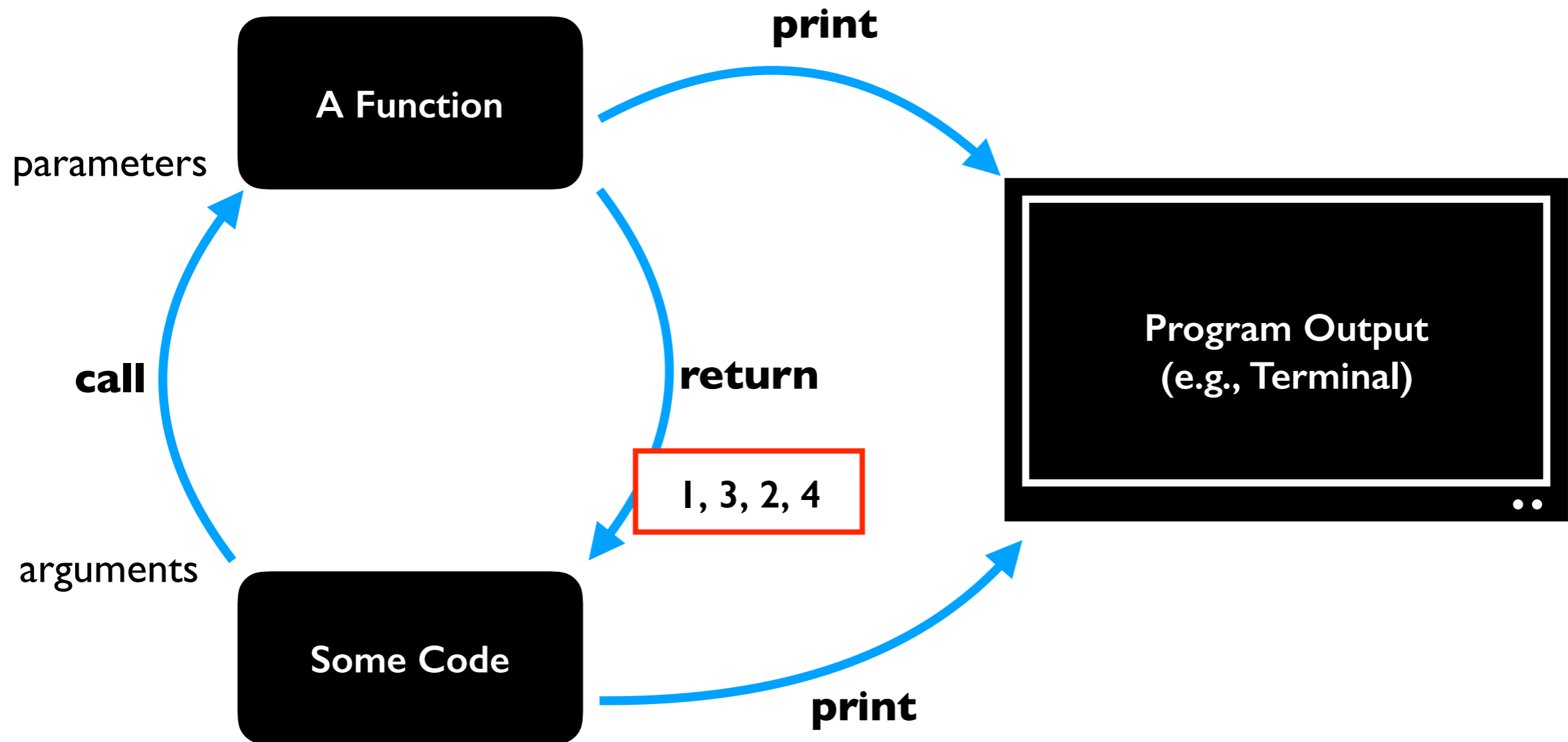
A Function

parameters

call

return

1, 3, 2, 4

arguments

Some Code

print

print

Program Output
(e.g., Terminal)

returning, instead of **printing**, gives callers different options for how to use the result
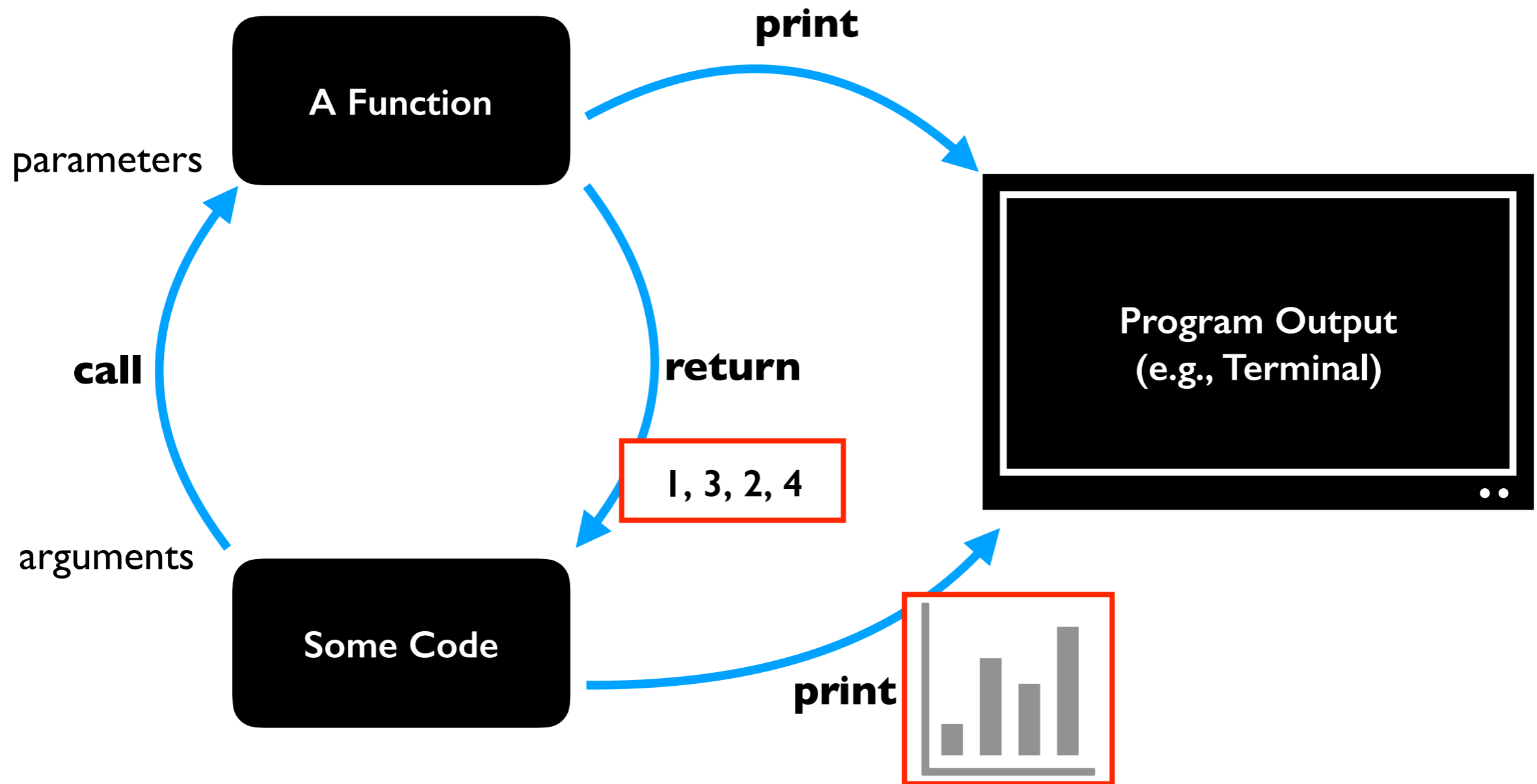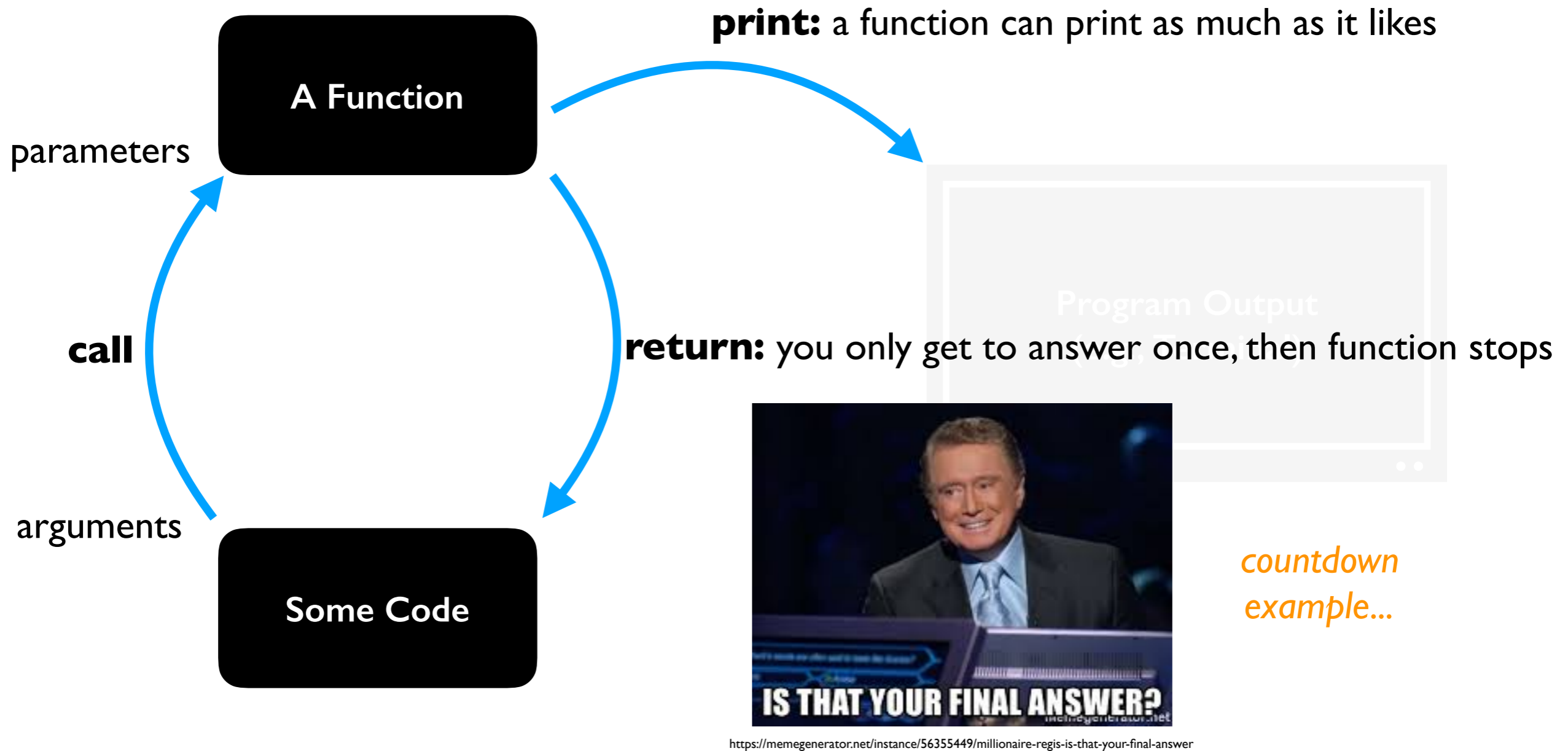
Print vs. Return

returning, instead of **printing**, gives callers different options for how to use the result

Print vs. Return

A Function

parameters

call

arguments

Some Code

print: a function can print as much as it likes

return: you only get to answer once, then function stops

Program Output

countdown example...

IS THAT YOUR FINAL ANSWER?

https://memegenerator.net/instance/56355449/millionaire-regis-is-that-your-final-answer

returning, instead of **printing**, gives callers different options for how to use the result

Checking Examples with PythonTutor

Wed: Creating Functions (Sep 18)

- Positional Params
- Keyword Params
- Return Values

Read: Downey Ch 3 ("Adding New Functions" to "Flow of Execution" and "Fruitful and Void Functions")

Read: Creating Fruitful Functions

Link to Slides

Link to Worksheet

Interactive Exercises

Due: P2

Assigned: Lab-P3, P3

Worksheet Problem 19

Python 3.6

```
→  1  print("A")
   2
   3  def foo():
   4      print("B")
   5  print("C")
   6  foo()
   7  print("D")
   8  foo()
```

Edit this code

→ line that has just executed
→ next line to execute

...

Demo: Approximation Program

input: a number from user

output: is it approximately equal to an important number? (pi or zero)

```
python approx.py
please enter a number: 3.14
close to zero?   False
close to pi?     True
```

```
python approx.py
please enter a number: 0.000001
close to zero?   True
close to pi?     False
```

```
python approx.py
please enter a number: 3
close to zero?   False
close to pi?     False
```

what is error between 4 and 8?
- 100%
- 50%

$$\frac{abs(8 - 4)}{max(abs(4), abs(8))}$$