[220 / 319] Creating Functions

Meena Syamkumar
Andy Kuemmel
Cole Nelson

Readings:

Parts of Chapter 3 of Think Python,
Chapter 5.5 to 5.8 of Python for Everybody
Creating Fruitful Functions

Learning Objectives

Explain the syntax of a function header:

- def, ( ), :, tabbing, return

Write a function with:

- correct header and indentation
- a return value (fruitful function) or without (void function)
- parameters that have default values

Write a function:

- knowing difference in outcomes of print and return statements

Determine result of function calls with 3 types of arguments:

- positional, keyword, and default

Trace function invocations, to determine control flow

*Where do **modules** come from?*

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

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

installed (e.g., jupyter)
- pip install jupyter
- pip install ...

Anaconda did these installations for us

custom
- project (lab-p3)

Main Code:

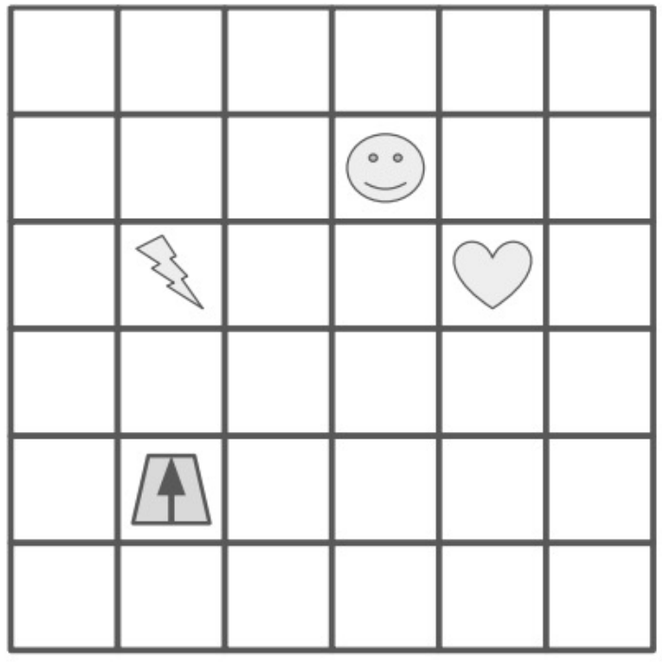
1. Put 2 in the “moves” box
2. Perform the steps under “Move Code”, then continue to step 3
3. Rotate the robot 90 degrees to the right (so arrow points to right)
4. Put 3 in the “moves” box
5. Perform the steps under “Move Code”, then continue to step 6
6. Whatever symbol the robot is sitting on, write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

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: $f(x) = x^2$

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: $f(x) = x^2$

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 (tab space) before the statement(s)

Math to Python

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

Python:

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


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:

```
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)

Let's implement functions

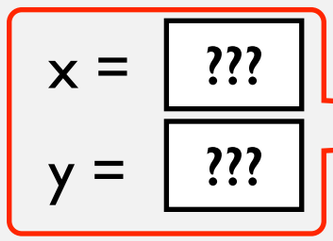
```
cube(side)
```

```
is_between(lower, num, upper)
```

jupyter / PythonTutor demos ...

Rules for filling parameters...

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



function declaration

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

```
foo(99, 100)
```

function invocation

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)
```

```
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*

worksheet practice...

Generating grid for game like Battleship

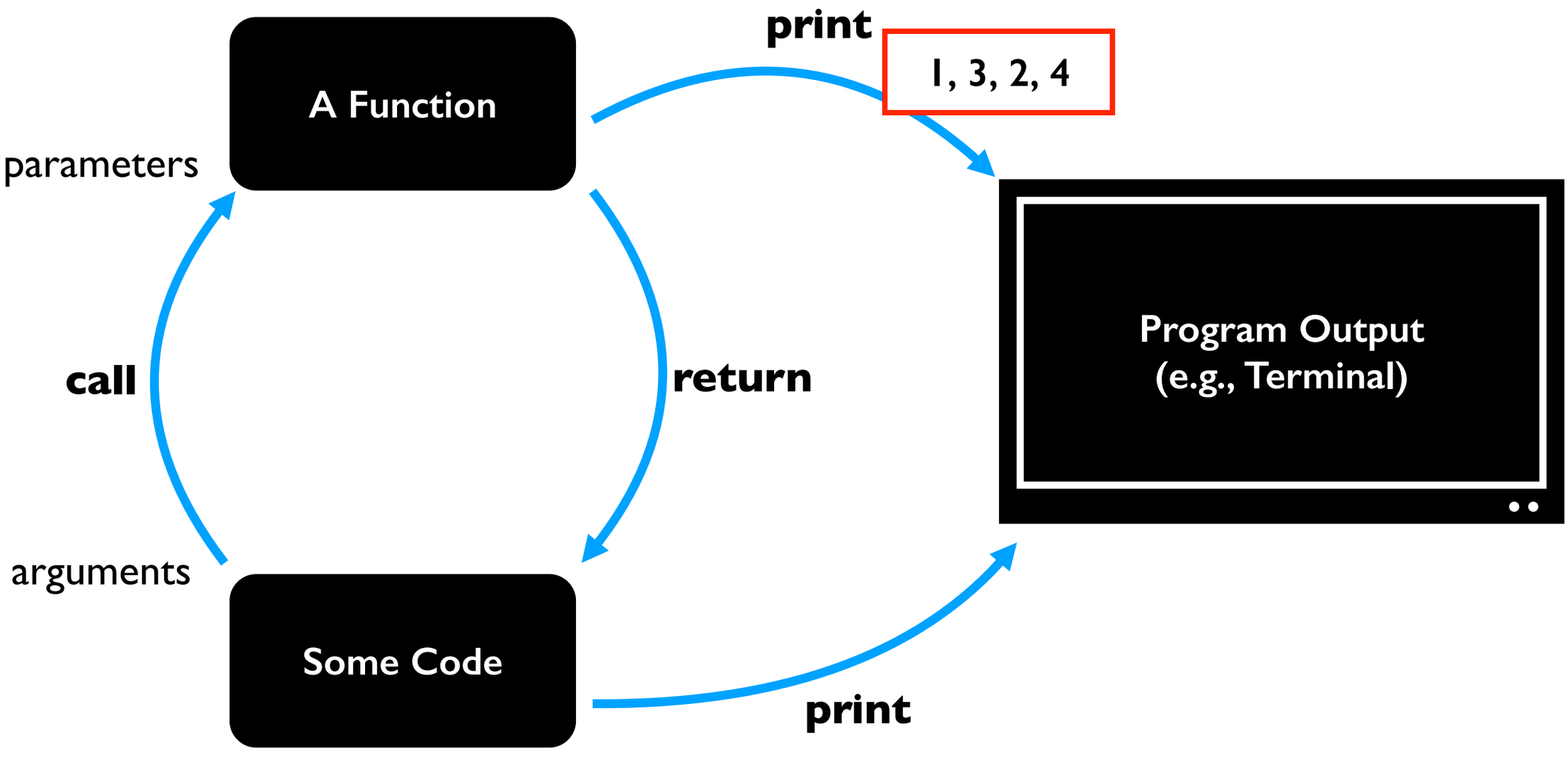
Grid:
##########
##########
##########
##########
##########
##########
##########
##########

10 x 8 grid

```
get_grid(width, height, symb = '#', title = 'Grid:')
```

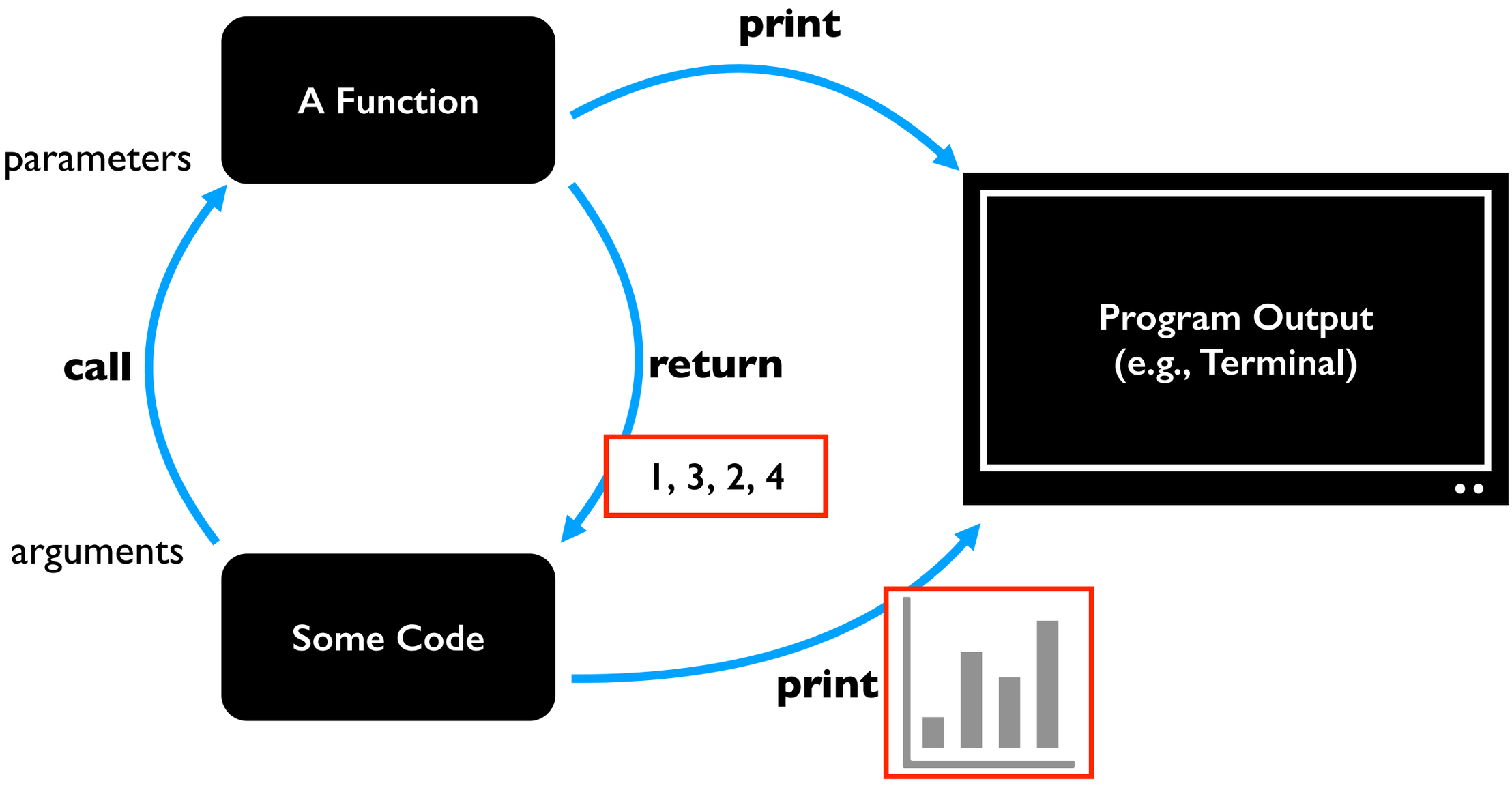
PythonTutor demo...

Print vs. Return



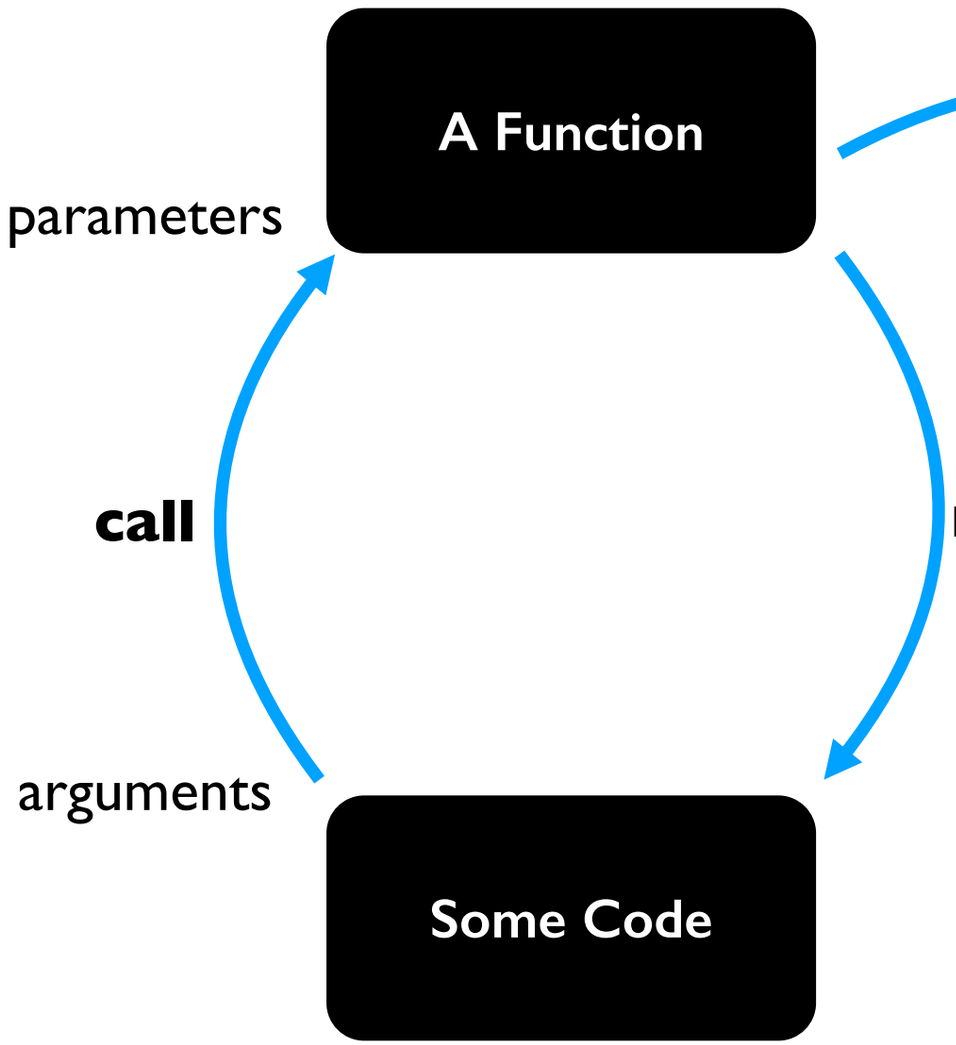
we could call print from multiple places

Print vs. Return



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

Print vs. Return



print: a function can print as much as it likes

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

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

bad_add3 example...

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

Interactive Examples with PythonTutor

Course website schedule page entry for "Creating Functions"

```
def func_c():
    print("C")

def func_b():
    print("B1")
    func_c()
    print("B2")

def func_a():
    print("A1")
    func_b()
    print("A2")

func_a()
```

Let's trace this example

Challenge: Approximation Program

input: a number from user

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

```
please enter a number: 3.14
close to zero?  False
close to pi?    True
```

```
please enter a number: 0.000001
close to zero?  True
close to pi?    False
```

```
please enter a number: 3
close to zero?  False
close to pi?    False
```

what is error between 4 and 8?

- 100%
- 50%

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