JamCoders: Week 1

Lecture 3B:

- Functions
- Print vs Return
- None

Lecture Objective

- Learn what a function is and why we use them
- Recognize common functions we've already been using
- Get comfortable with defining and calling basic functions

Next lecture we'll learn more details about functions. Today we'll focus on the core details.

Routine

What do you do when you get up in the morning? What is your morning routine?

1. Check my phone
2. Brush my teeth
3. Shower
4. Get dressed
5. Make my bed
6. Eat breakfast

Encapsulation

Encapsulation is how we (humans) deal with *complexity*.

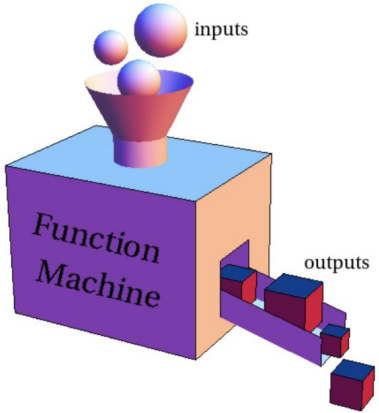
Parent tells you to “brush your teeth”, rather than “pick up the tooth brush, put toothpaste on it, put it in your mouth, move it around for 120 seconds, …”

Parent can do that because at some point in the past, we all learned that “brush your teeth” means doing all of those other steps.

Functions

Functions

Functions are a way of ***encapsulating*** some code to use it elsewhere in our program.



This allows us to accomplish two things at once:

- Write repeatable code: We can write a function once, then use it anywhere.
- Write readable code: it is clear at a glance what each part of the code is responsible for.

Calling Functions

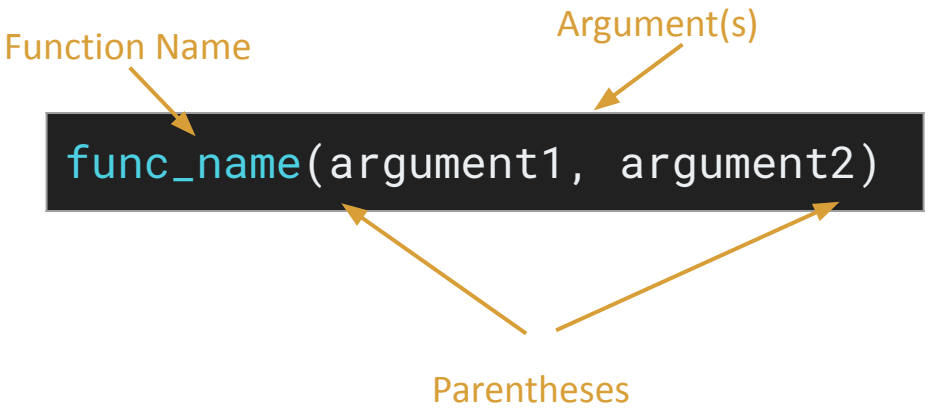
Functions

You've already been *using* functions all along, you just didn't know it.

```
int("8")
input("Name?")
str(100)
print("Hello")
```

Using Functions

To use a function, we **call** the function...



Calling a function

A function call is another kind of expression, just like `a + b` or `is_raining and is_cloudy`, and therefore gets evaluated.

A function call **evaluates** to the **return value** of the function

Calling a function

```
name = input("What is your name? ")
```

We say `input` is a function that **returns a string**, because the function call evaluates to a string.

Calling a function

```
number = int("8")
```

We say `int` is a function that **returns an integer**, because the function call evaluates to an integer.

Calling a function

```
print("Hello world")
```

So...what does print **return**? What does it evaluate to?

Nothing! Functions don't *have* to return something, they can just do something.

Calling a function

```
rounded = round(2.5662, 2)
```

`round` is an example of a function that takes multiple **arguments**, separated by commas.

The number of arguments that a function takes is specified in the function definition.

Defining Functions

Defining Functions

We know how to use existing functions that Python provides us...how about making our own?

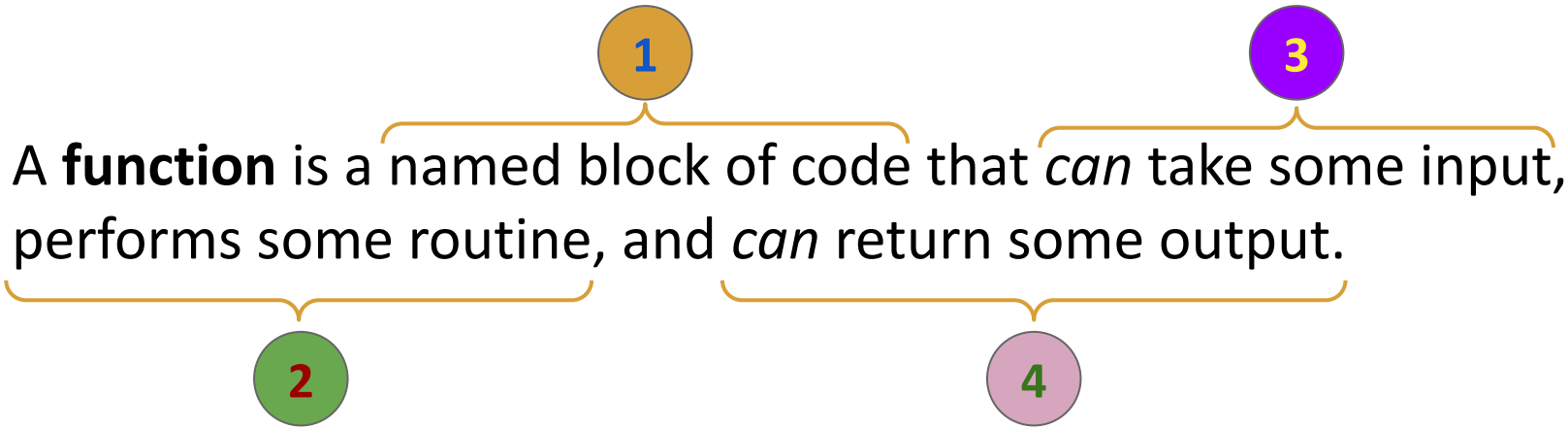
Called **defining** a function.

Functions

The technical definition...

A **function** is a named block of code that *can* take some input, performs some routine, and *can* return some output.

Functions

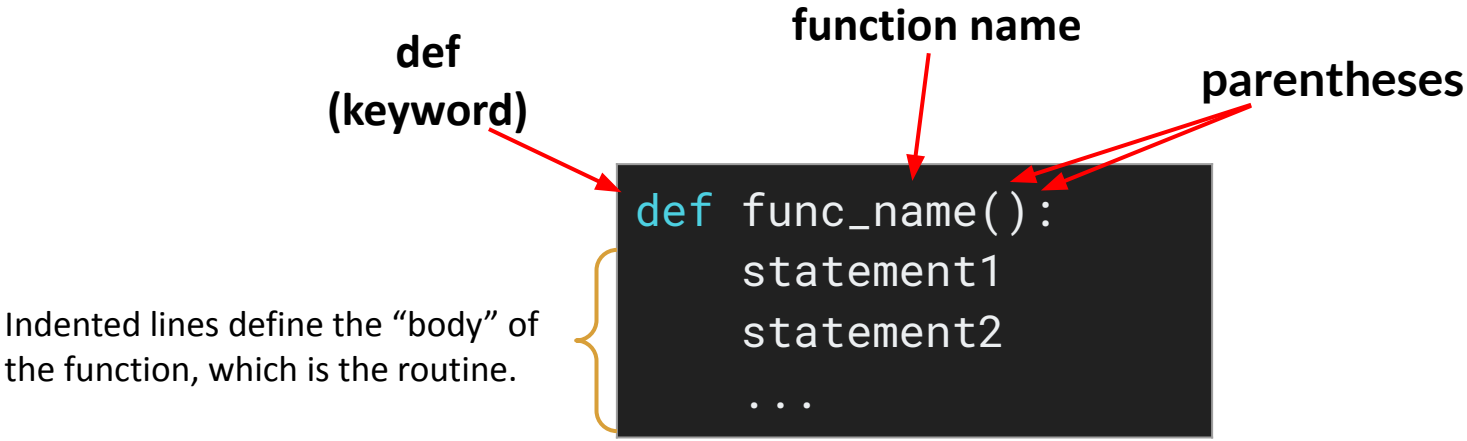

1
3
A **function** is a named block of code that *can* take some input,
performs some routine, and *can* return some output.
2
4

Functions 1 2

A **function** is a named block of code...that performs some routine

Functions 1 2

A **function** is a named block of code...that performs some routine



(def stands for “define”)

Functions 1 2

A **function** is a named block of code...that performs some routine

For example, making this block of code into a function...

```
num = -12
if num >= 0:
    print(num)
else:
    print(-num)
```

```
def absolute_value():
    num = -12
    if num >= 0:
        print(num)
    else:
        print(-num)
absolute_value()
```

Functions 3

Functions are powerful because they *can* take some *input* to use in the routine.

Functions 3

Functions are powerful because they *can* take some *input* to use in the routine.

```
def func_name(param1, param2):
    statement1
    statement2
    ...
```

Zero or more **parameters**, which are given separated by commas in the parentheses. These are then variables that can be used *only* within the function.

Functions 3

Functions are powerful because they *can* take some *input* to use in the routine.

```
def absolute_value():
    num = -12
    if num >= 0:
        print(num)
    else:
        print(-num)
absolute_value()
```

```
def absolute_value(num):
    if num >= 0:
        print(num)
    else:
        print(-num)
absolute_value(-12)
```

Now, this routine will work for any given number, not just -12.

Functions 4

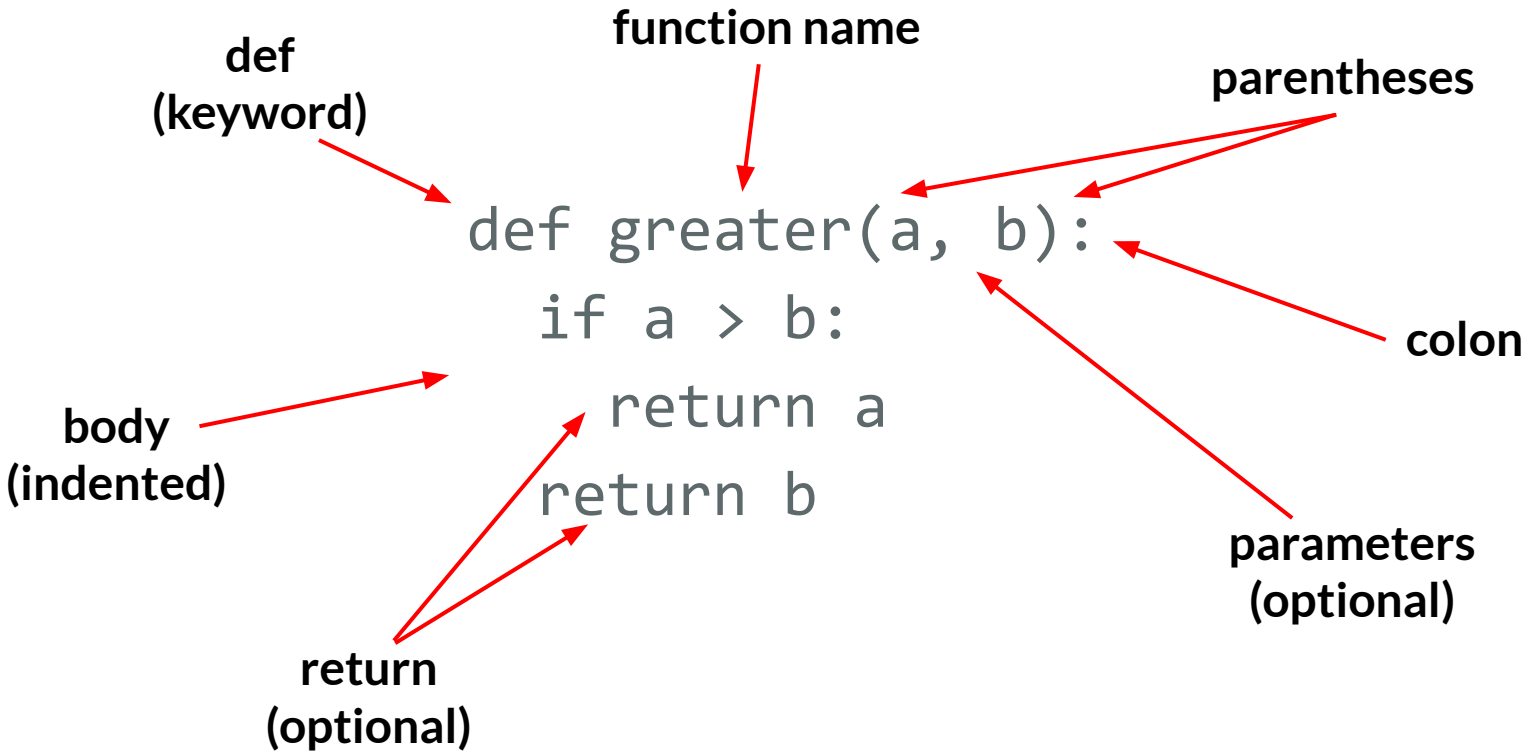
A function...can return some output

```
def func_name(param1, param2):
    x = param1 + param2
    return x
```

`return` is a reserved keyword used to return output from a function.

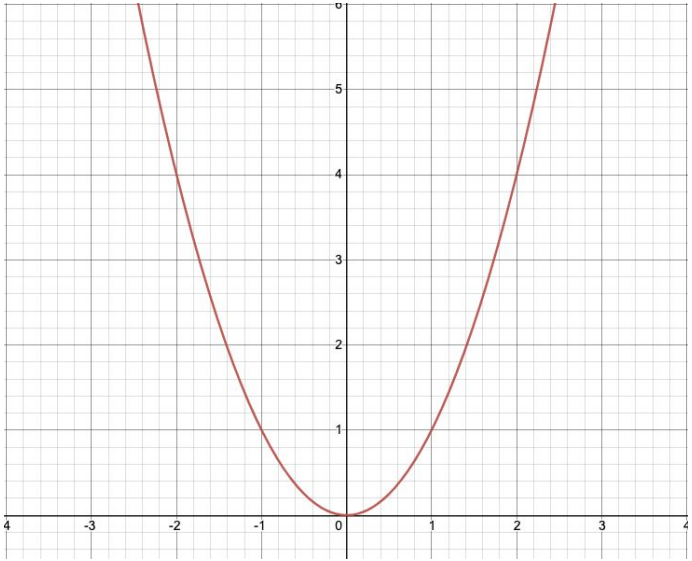
- It can only be used within a function.
- Function execution stops as soon as soon as return line is reached.

Function Anatomy



Functions

Another perspective...similar to “functions” in math.



$f(x) = x^2$

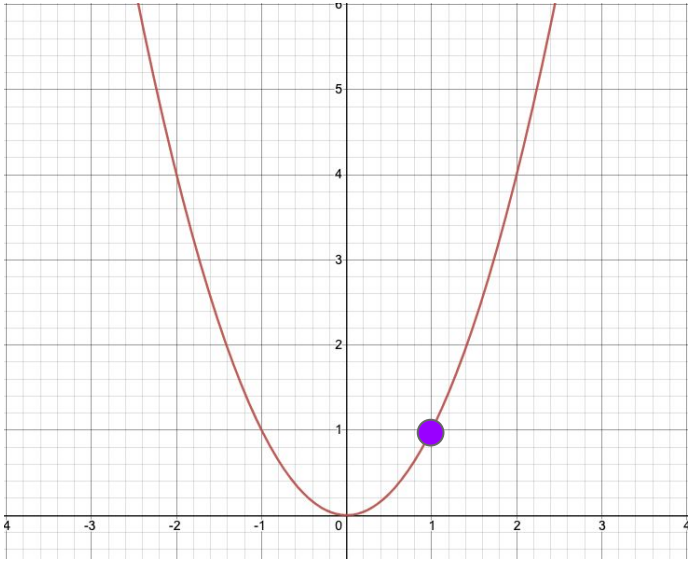
Functions

Another perspective...similar to “functions” in math.

Input: 1

“Call” the function: f(1)

Output: 1



$f(x) = x^2$

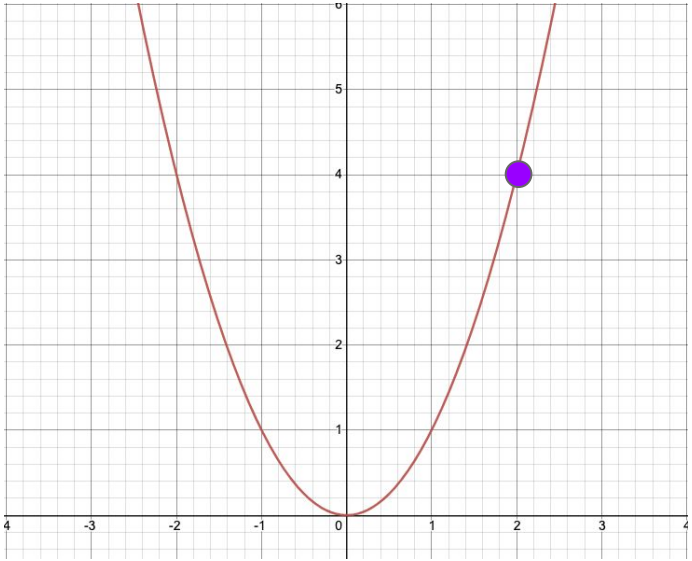
Functions

Another perspective...similar to “functions” in math.

Input: 2

“Call” the function: f(2)

Output: 4



$f(x) = x^2$

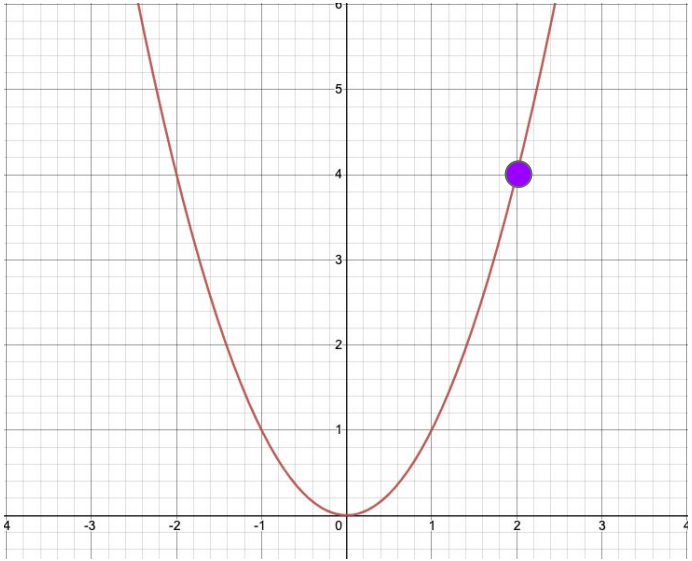
Functions

Another perspective...similar to “functions” in math.

Input: 2

“Call” the function: f(2)

Output: 4



x^2 is the body

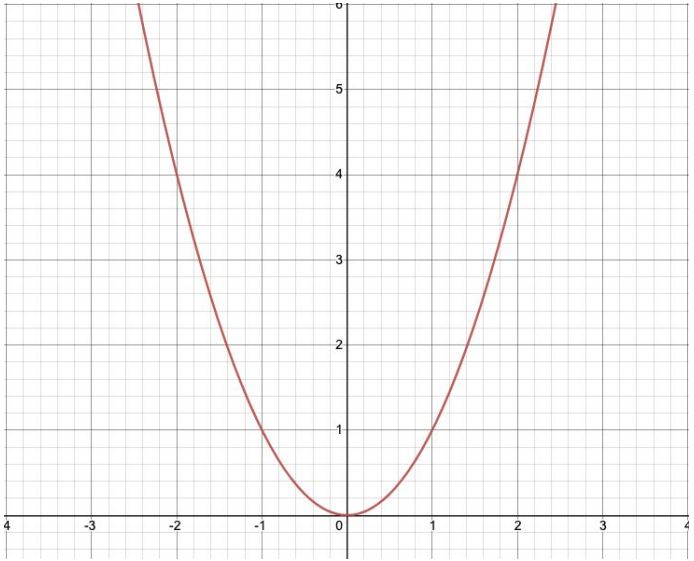
x is the parameter

2 is the argument

4 is the return value

$f(x) = x^2$

Functions



$f(x) = x^2$

```
def parabola(x):
    return x * x

print("f(1) ==", parabola(1))  # == 1
print("f(2) ==", parabola(2))  # == 4
```

Function Examples

Function Examples

Write a function to introduce a person with their name and age.

```
def introduction(name, age):
    age_string = str(age) + " years old"
    print("Hi this is " + name + ". They are " + age_string)

introduction("Sonya", 41)
introduction("James", 19)
introduction("Greg", 28)
introduction("Alexandria", 76)
```

Function Practice

Write a function that takes a string as a parameter and returns the word repeated twice.

Sample input: “hello”

Sample output: “hellohello”

Quick Takeaways

1. Think of a function as a (sub)routine.
2. As soon as you see yourself doing the same sort of thing in different places, think function.
3. Use functions to make your code more readable via “encapsulation”

Vocab:

- Using a function is called **calling** or **executing** the function
  - Function calls are **expressions** that need to be evaluated
  - The input values given in the function call are called **arguments**
- Creating a new function is called **defining** it
  - The input variables in the definition are called **parameters**
  - We can **return** a value for the function to evaluate to

Live Coding Demo

Print vs Return

Print vs Return

Common question…what's the difference between print and return?

Print vs Return

Print

Return

Print vs Return

Print

```
def print_hello():
  print("Hello World")
value = print_hello()
print("Return value:", value)
```

```
~/CSCI100-Repl$ python3 main.py
Hello World
Return value: None
```

`print` is a function that gives output from your entire program to the computer to print to the terminal.

- Can be called from *anywhere*
- Function doesn't return any value
- Communicates outside of the program (to the terminal)

Return

Print vs Return

Print

```
def print_hello():
  print("Hello World")
value = print_hello()
print("Return value:", value)
```

```
~/CSCI100-Repl$ python3 main.py
Hello World
Return value: None
```

`print` is a function that gives output from your entire program to the computer to print to the terminal.

- Can be called from *anywhere*
- Function doesn't return any value
- Communicates outside of the program (to the terminal)

Return

```
def return_hello():
  return "Hello World"

value = return_hello()
print("Return value:", value)
length = len(value)
if length > 5:
  print(value)
```

```
~/CSCI100-Repl$ python3 main.py
Return value: Hello World
Hello World
```

`return` is a statement that specifies the value that the function will evaluate to when called.

- Can only be used within a function
- Does **not** print anything to the screen, only specifies value that the function call evaluates to
- Someone else can print the return value if they want to.

None

None

What happens when a function doesn't return anything? What does it evaluate to?

```
def print_hello():
  print("Hello World")
value = print_hello()
print("Return value:", value)
```

```
~/CSCI100-Repl$ python3 main.py
Hello World
Return value: None
```

None is a special value, used for when there is no value to set. Most commonly, you'll see this when a function doesn't return anything.

It is of type "NoneType", a new type that can only have one value, None.

What is this?

None

You can manually set a value to None.

```
ta_name = None
if ta_name == None:
  ta_name = input("Enter your TA's name: ")
print(ta_name)
```