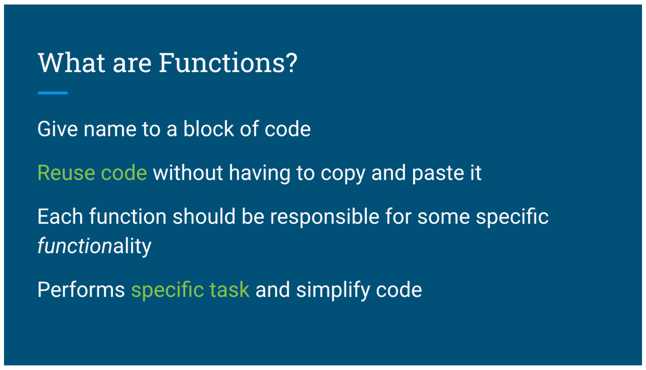
Functions (Definitions)

Date: Sep 9, 2024

Defining Functions



What are Functions?

Give name to a block of code

Reuse code without having to copy and paste it

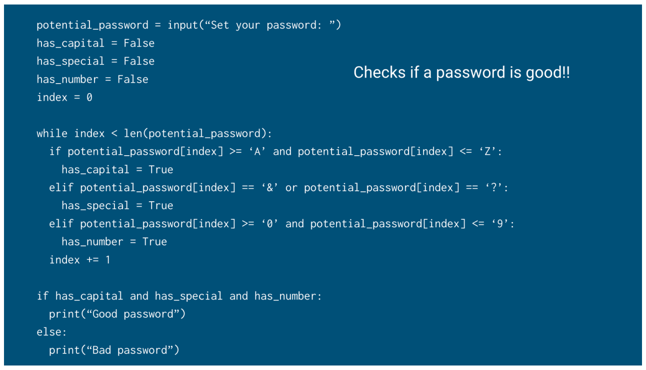
Each function should be responsible for some specific *functionality*

Performs specific task and simplify code

Functions essentially give a name to a block of code.

We can reuse the code in a function by calling it and this helps us avoid duplicating code! Why does this matter? If we need to update something in this code, we only need to update it once!

A function should perform some specific task!



```
potential_password = input("Set your password: ")
has_capital = False
has_special = False
has_number = False
index = 0

while index < len(potential_password):
  if potential_password[index] >= 'A' and potential_password[index] <= 'Z':
    has_capital = True
  elif potential_password[index] == '&' or potential_password[index] == '?':
    has_special = True
  elif potential_password[index] >= '0' and potential_password[index] <= '9':
    has_number = True
  index += 1

if has_capital and has_special and has_number:
  print("Good password")
else:
  print("Bad password")
```

Checks if a password is good!!

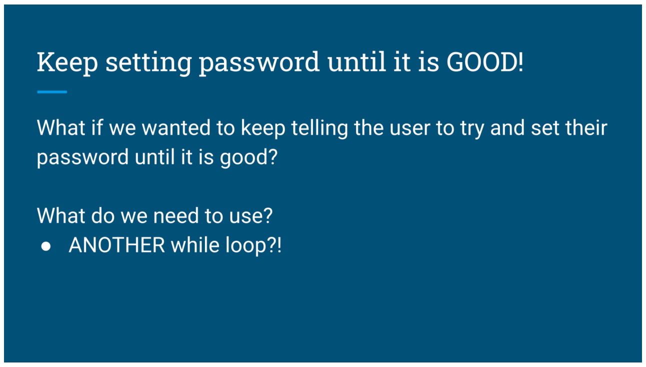
Let's take a look at this code and figure out what it is doing.

(1) Looks like it is asking the user to set a password.

(2) It iterates through the chosen password character by character to make sure that the password contains:

- capital letter
- special character
- number

(3) Tells the user if the password is "good" or "bad" depending on if it contains ALL of those characters.

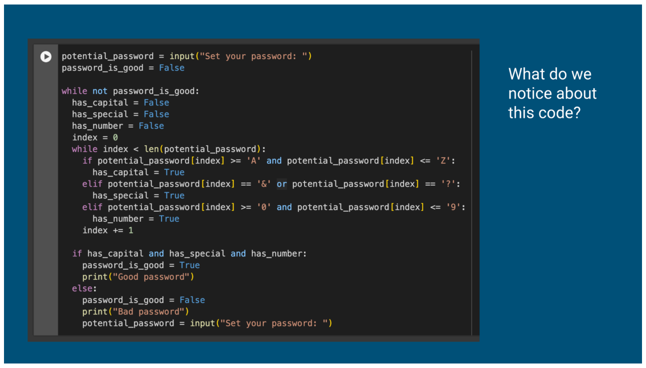


Keep setting password until it is GOOD!

What if we wanted to keep telling the user to try and set their password until it is good?

What do we need to use?

- ANOTHER while loop?!



```
potential_password = input("Set your password: ")
password_is_good = False

while not password_is_good:
  has_capital = False
  has_special = False
  has_number = False
  index = 0
  while index < len(potential_password):
    if potential_password[index] >= 'A' and potential_password[index] <= 'Z':
      has_capital = True
    elif potential_password[index] == '&' or potential_password[index] == '?':
      has_special = True
    elif potential_password[index] >= '0' and potential_password[index] <= '9':
      has_number = True
    index += 1

  if has_capital and has_special and has_number:
    password_is_good = True
    print("Good password")
  else:
    password_is_good = False
    print("Bad password")
    potential_password = input("Set your password: ")
```

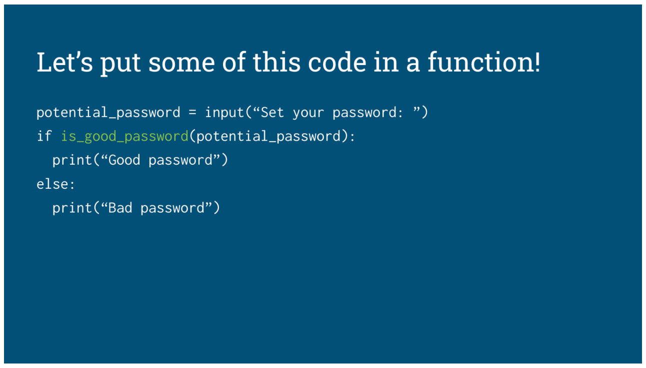
What do we notice about this code?

If we use ANOTHER while loop, then the code kinda is hard to read!!

- Lots of levels of indentation!
- Nested while loops!

If we are repeating code over and over again, let's use a function to reuse this code!

This not only helps us make the code easier to read, but also makes it more MODULAR! Put blocks of code into different modules/parts!



Let's put some of this code in a function!

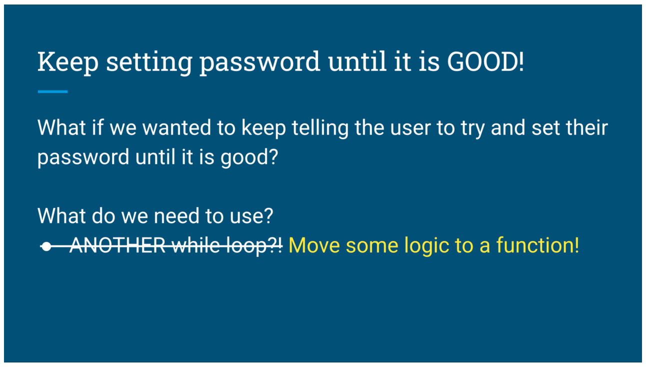
```
potential_password = input("Set your password: ")
if is_good_password(potential_password):
  print("Good password")
else:
  print("Bad password")
```

If we put the logic that checks if the password is "good" in a function, then the code looks like this!

This looks so much simpler and cleaner! The function can handle the task to iterate through the characters and make sure all the conditions are satisfied!

If we used the function in our nested while loop example, we get rid of the nested loops!

We can put the logic that does one of the while loops in a function so we no longer add all these layers of indentation to our code!

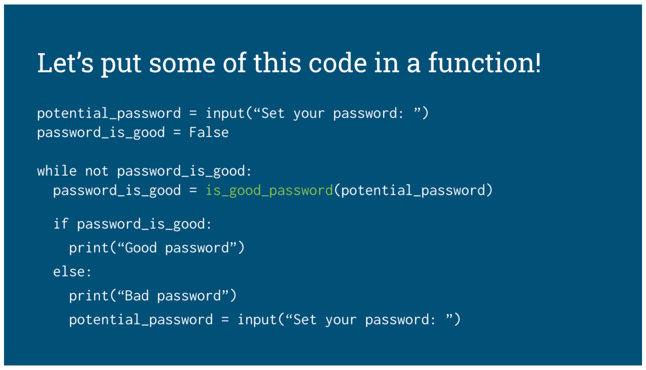


Keep setting password until it is GOOD!

What if we wanted to keep telling the user to try and set their password until it is good?

What do we need to use?

- ~~ANOTHER while loop?!~~ Move some logic to a function!



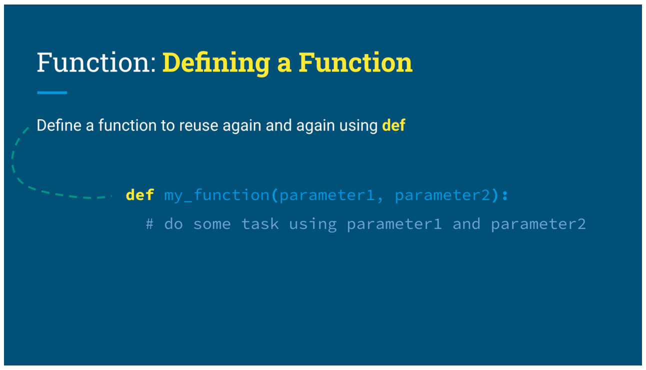
Let’s put some of this code in a function!

```
potential_password = input("Set your password: ")
password_is_good = False

while not password_is_good:
  password_is_good = is_good_password(potential_password)

  if password_is_good:
    print("Good password")
  else:
    print("Bad password")
    potential_password = input("Set your password: ")
```

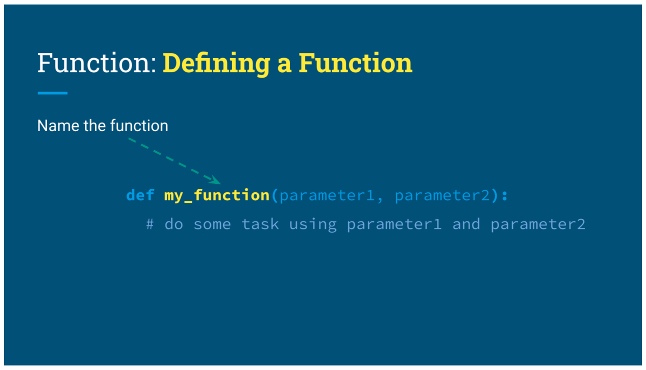
How do we define a function in Python?



Function: **Defining a Function**

Define a function to reuse again and again using **def**

```
def my_function(parameter1, parameter2):
  # do some task using parameter1 and parameter2
```



Function: **Defining a Function**

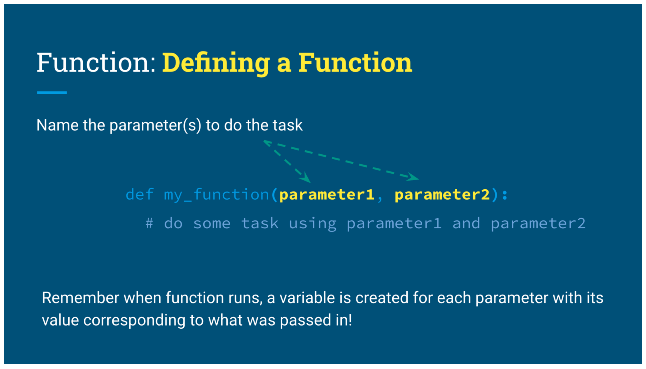
Name the function

```
def my_function(parameter1, parameter2):
  # do some task using parameter1 and parameter2
```

If you want to define/create a function, you need to use the `def` keyword!

Then that is followed by the name you want to give the function that is descriptive to what the function is doing!

Then give variable names to the parameters that you will need as input values/data in order for the function to do its job!

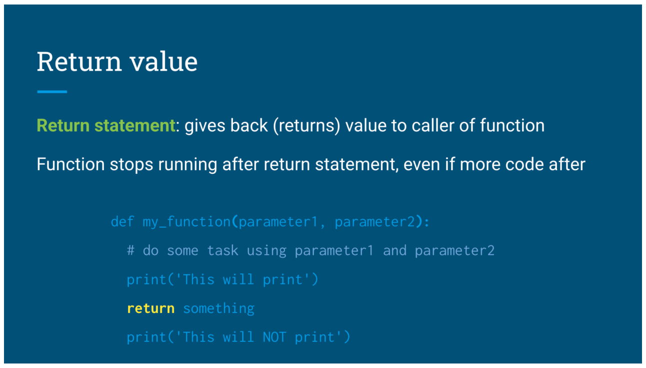


Function: **Defining a Function**

Name the parameter(s) to do the task

```
def my_function(parameter1, parameter2):
  # do some task using parameter1 and parameter2
```

Remember when function runs, a variable is created for each parameter with its value corresponding to what was passed in!



Return value

Return statement: gives back (returns) value to caller of function

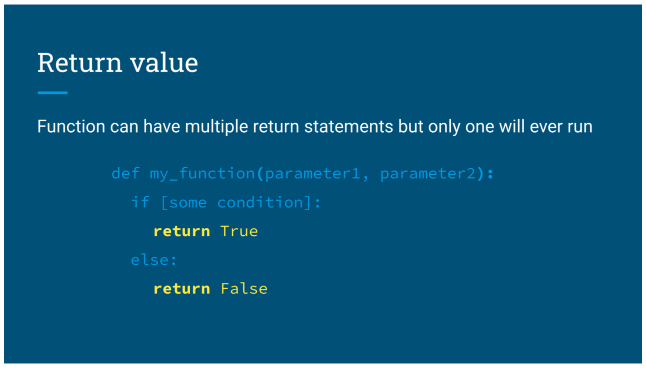
Function stops running after return statement, even if more code after

```
def my_function(parameter1, parameter2):
  # do some task using parameter1 and parameter2
  print('This will print')
  return something
  print('This will NOT print')
```

The `return` statement will tell you what values/data this function returns/outputs!

Keep in mind that `return` is also a keyword in Python!

The function stops running once the computer sees the `return` statement EVEN IF there is more code after the `return`!



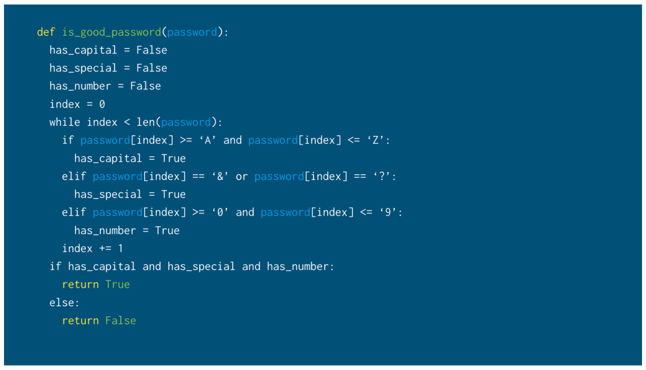
Return value

Function can have multiple return statements but only one will ever run

```
def my_function(parameter1, parameter2):
  if [some condition]:
    return True
  else:
    return False
```

Note that functions CAN have MULTIPLE `return` statements! BUT only ONE will ever run!

Once one is reached in the program, the function will stop!



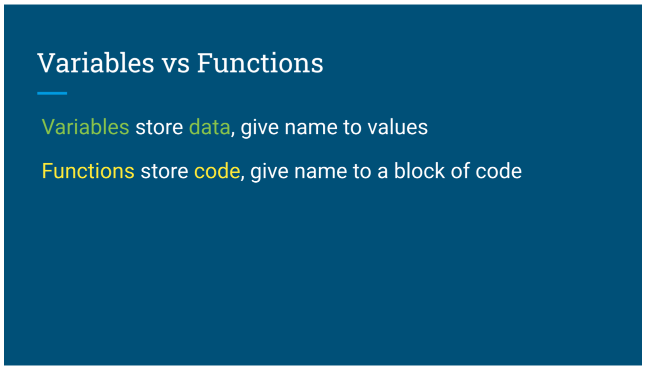
```
def is_good_password(password):
  has_capital = False
  has_special = False
  has_number = False
  index = 0
  while index < len(password):
    if password[index] >= ‘A’ and password[index] <= ‘Z’:
      has_capital = True
    elif password[index] == ‘&’ or password[index] == ‘?’:
      has_special = True
    elif password[index] >= ‘0’ and password[index] <= ‘9’:
      has_number = True
    index += 1
  if has_capital and has_special and has_number:
    return True
  else:
    return False
```

Let’s go back to our code and put it in a function!

Note: The parameter/argument that this function needs is a password! Because the point of this function is to check if a password is “good”! That variable is used in this function which stores the value that would be passed in the function when it is called!

Notice that this function has two `return` statements!

We can actually write this multiple ways (see the colab)!



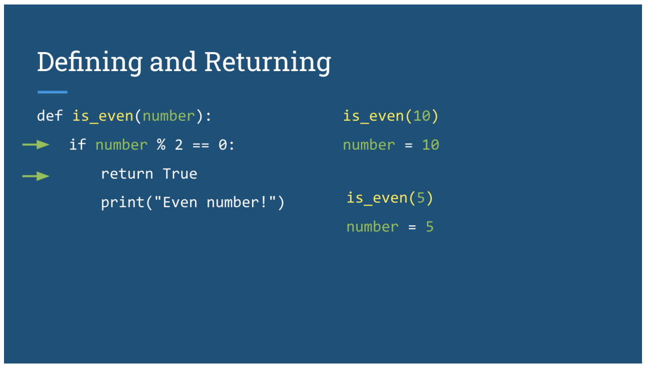
Variables vs Functions

Variables store data, give name to values

Functions store code, give name to a block of code

Remember how variables store data and store a specific value? They give a name to a value.

Similarly, functions store code. They give name to a block of code/set of instructions for the computer that does a specific task.



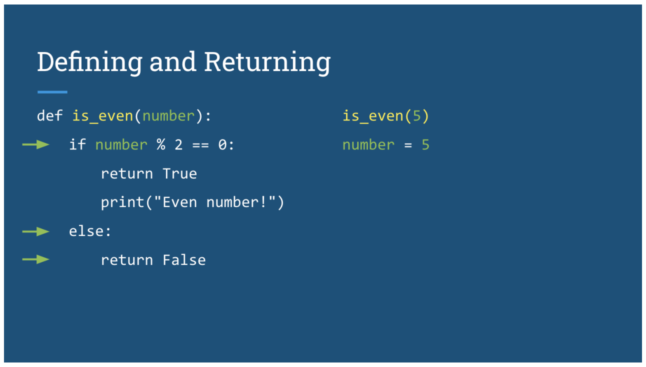
Defining and Returning

```
  def is_even(number):               is_even(10)
→   if number % 2 == 0:              number = 10
→       return True
        print("Even number!")        is_even(5)
                                     number = 5
```

In this first example, we are defining a function that returns `True` if the argument passed into the function is an even number.

Note: The lines of code after the `return` statement WILL NOT RUN because once you hit the `return` in a function, the function stops.

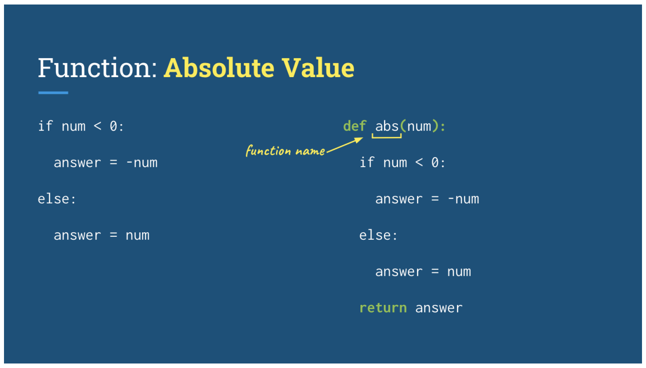
`is_even(10)` returns `True`
`is_even(5)` returns `None`



Defining and Returning

```
def is_even(number):                is_even(5)
    if number % 2 == 0:             number = 5
        return True
        print("Even number!")
    else:
        return False
```

In the second example, `is_even(5)` returns `False`!



Function: **Absolute Value**

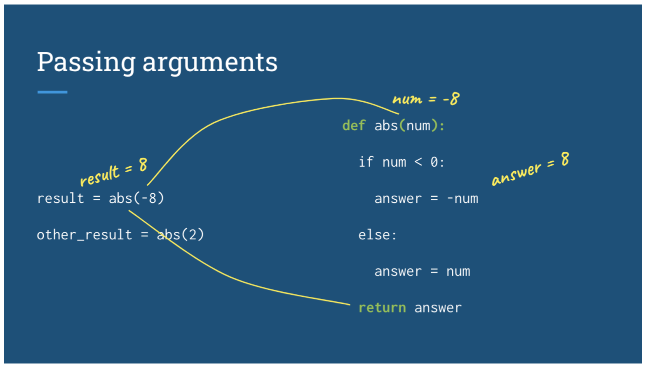
```
if num < 0:                          def abs(num):
  answer = -num                        if num < 0:
else:                                    answer = -num
  answer = num                         else:
                                         answer = num
                                       return answer
```

Let's say we want to write a function that takes a number as input and outputs the absolute value of this number.

Remember you need:

- **Name**: The keyword `def` to tell the computer you are defining a function followed by the name you want to give it!
- **Parameters**: Specify parameters needed for the function to do its job!
- **Return**: A `return` statement if you want this function to have an output!

This function probably needs to take in a number as a parameter to calculate the absolute value of it and it also probably wants to return that value as an output.



Passing arguments

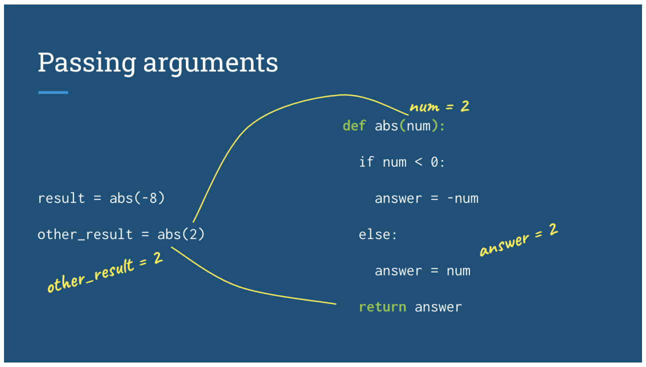
```
                                     def abs(num):
                                       if num < 0:
result = abs(-8)                         answer = -num
other_result = abs(2)                  else:
                                         answer = num
                                       return answer
```

Let's say we have this line of code:

```
result = abs(-8)
```

We know that we are calling this function and passing in the argument of `-8` as the input data!

Then when we go into the function and run the code there, we are creating a variable `num` that



Passing arguments

```
result = abs(-8)
other_result = abs(2)
```

```
def abs(num):
  if num < 0:
    answer = -num
  else:
    answer = num
  return answer
```

stores the number `-8`! Then we use this variable to run the code in the function!

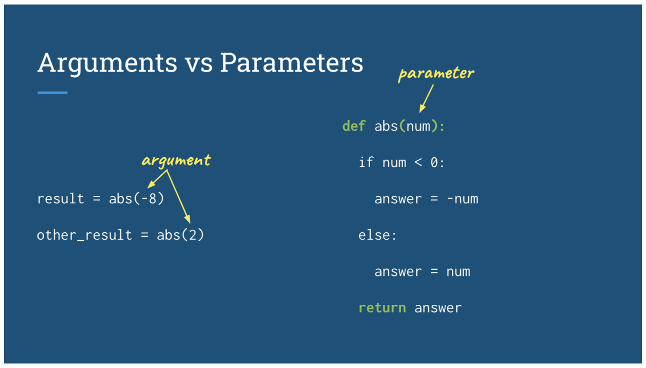
So it looks like `answer = 8` because `answer = -num`!

We return the answer so then the output of this function is the integer `8` which means `result` stores this `8`!

Similarly we do this same process for the next line of code:

```
other_result = abs(2)
```

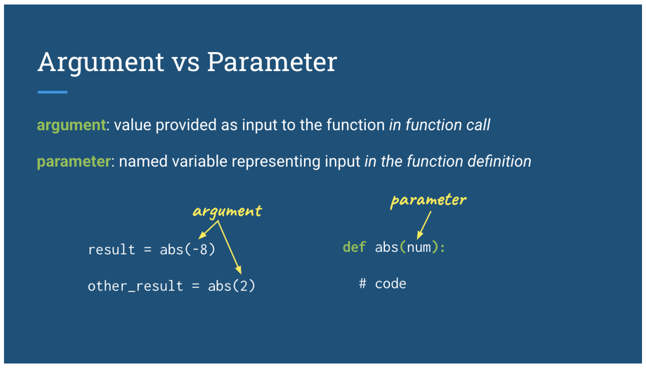
So after calling the function and running the code in the function, `other_result = 2`!



Arguments vs Parameters

```
result = abs(-8)
other_result = abs(2)
```

```
def abs(num):
  if num < 0:
    answer = -num
  else:
    answer = num
  return answer
```



Argument vs Parameter

argument: value provided as input to the function *in function call*

parameter: named variable representing input *in the function definition*

```
result = abs(-8)
other_result = abs(2)
```

```
def abs(num):
  # code
```

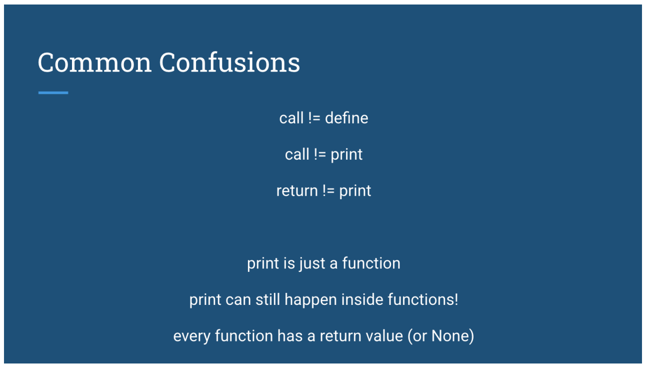
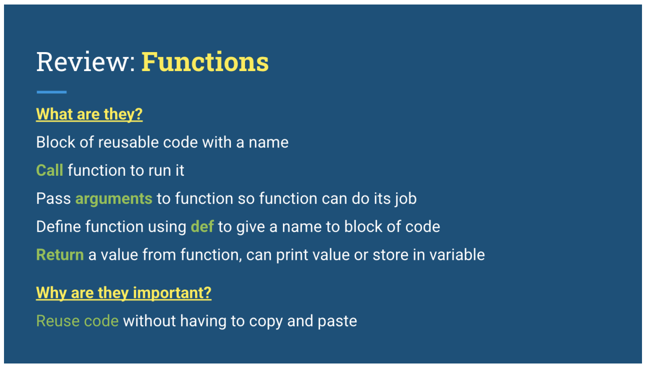
So what exactly is the difference between an argument and a parameter?

The data/input you pass into the function call are the arguments.

You can think of arguments as the actual values passed to the function when it's called.

The variables (in the function definition) that store this data/input value that is then used inside the function code are the parameters.

You can think of parameters as the variables listed within the parentheses in a function's definition that establishes the kind of input the function expects. They act as placeholders for the values the function will work with. The function's code uses these parameter names to refer to the values it receives.

 Common Confusions call != define call != print return != print print is just a function print can still happen inside functions! every function has a return value (or None)	We added one more common confusion to this list. Note that calling a function is not the same thing as defining a function. The main key difference is the `def` keyword.
 Review: **Functions** **What are they?** Block of reusable code with a name **Call** function to run it Pass **arguments** to function so function can do its job Define function using **def** to give a name to block of code **Return** a value from function, can print value or store in variable **Why are they important?** Reuse code without having to copy and paste	Functions allow us to take a block of code that does some specific task and REUSE IT! We can give this block of code a name and then define it using the `def` keyword. Remember when defining a function, you also need to specify the parameters needed for this function to do its job. When you call the function, you need to pass in the arguments you want as input data/values. If there is a `return` statement in the function definition, then the output/return value should be stored in a variable when you call the function!

TODOs

- [ ] HW 3 - While Loops & Strings
    - Due Wednesday Sep 11, 2024 @ 11:59pm!

- [ ] Project 1 (pt. 3) - Blackjack!
    - Due Sunday Sep 15, 2024 @ 11:59pm!
- [ ] Midterm Exam 1 – COMING SOON
    - Next Wednesday on Sep 18, 2024 !

- [ ] STEP Internship @ Google
    - Applications open Sep 30, 2024 and close on Oct 25, 2024 !!
    - For 1st year and 2nd year students, applications will be available at g.co/jobs/step!
    - All of the available intern opportunities can be found at google.com/students!