Creating Functions

Introduction

Now that we learned about loops, it would be nice to have the ability to *reuse* our code to help us solve different problems. Functions allow us to do just that. They also give us the ability to name a sequence of operations (or block of code), thus making our code expressive. Let's see how this works, and why something like this is useful.

Objectives

You will be able to:

- Declare and use a basic function

Our problem so far

Imagine that we have a group of employees who have just joined our company.

```
In [1]: new_employees = ['jim', 'tracy', 'lisa']
        new_employees
```

```
Out[1]: ['jim', 'tracy', 'lisa']
```

> Press shift + enter to run this code.

We want to send each of them a nice welcome message. We could use a `for` loop to create a list of `welcome_messages`.

```
In [2]: welcome_messages = []
        for new_employee in new_employees:
            welcome_messages.append("Hi " + new_employee.title() + ", I'm so glad to be w

        welcome_messages
```

```
Out[2]: ["Hi Jim, I'm so glad to be working with you!",
         "Hi Tracy, I'm so glad to be working with you!",
         "Hi Lisa, I'm so glad to be working with you!"]
```

In [8]:
```
# PRACTICE
graduants = ['chebii' , 'dee' , 'bonnie' , 'kim']
congratulations_messages = []
for graduant in graduants:
    congratulations_messages.append('Hi ' + graduant.title() + ', congratulations
congratulations_messages
```

Out[8]:
```
['Hi Chebii, congratulations on your graduation, keep going guys',
 'Hi Dee, congratulations on your graduation, keep going guys',
 'Hi Bonnie, congratulations on your graduation, keep going guys',
 'Hi Kim, congratulations on your graduation, keep going guys']
```

Then a couple of weeks later, a few more employees join, and we want to send messages to them as well.

In [3]:
```
new_employees = ['steven', 'jan', 'meryl']
```

Well to accomplish welcoming the new employees, we would likely copy our code from above.

In [3]:
```
welcome_messages = []
for new_employee in new_employees:
    welcome_messages.append("Hi " + new_employee.title() + ", I'm so glad to be w

welcome_messages
```

Out[3]:
```
["Hi Jim, I'm so glad to be working with you!",
 "Hi Tracy, I'm so glad to be working with you!",
 "Hi Lisa, I'm so glad to be working with you!"]
```

If each time we wanted to reuse code we would have to copy and paste the code and maintain a lot more code than is necessary. Also, each time we recopied it is another opportunity to make a mistake. So what if there was a way to write that code just one time, yet be able to execute that code wherever and whenever we want? Functions allow us to do just that.

Here is that same code wrapped in a function:

In [5]:
```
def greet_employees():
    welcome_messages = []
    for new_employee in new_employees:
        welcome_messages.append("Hi " + new_employee.title() + ", I'm so glad to

    return welcome_messages
```

In [6]:
```
greet_employees()
```

Out[6]:
```
["Hi Jim, I'm so glad to be working with you!",
 "Hi Tracy, I'm so glad to be working with you!",
 "Hi Lisa, I'm so glad to be working with you!"]
```

> Make sure to press shift + enter for the two cells above.

There are two steps to using a function: defining a function and executing a function. Defining a function happens first, and afterward when we call `greet_employees()` we execute the function.

In [7]:
```
new_employees = ['Jan', 'Joe', 'Avi']
greet_employees()
```

Out[7]:
```
["Hi Jan, I'm so glad to be working with you!",
 "Hi Joe, I'm so glad to be working with you!",
 "Hi Avi, I'm so glad to be working with you!"]
```

Ok, let's break down how to define, or declare, a function. Executing a function is fairly simple, just type the function's name followed by parentheses.

In [8]:
```
greet_employees()
```

Out[8]:
```
["Hi Jan, I'm so glad to be working with you!",
 "Hi Joe, I'm so glad to be working with you!",
 "Hi Avi, I'm so glad to be working with you!"]
```

Declaring and using functions

There are two components to declaring a function: the function signature and the function body.

In [8]:
```
def name_of_function(): # signature
    words = 'function body' # body
    print(words) # body
```

Function Signature

The function signature is the first line of the function. It follows the pattern of `def` , `function name` , `parentheses` , `colon` .

```
def name_of_function():
```

The `def` is there to tell Python that you are about to declare a function. The name of the function indicates how to reference and execute the function later. The colon is to end the function signature and indicate that the body of the function is next. The parentheses are important as well, and we'll explain their use in a later lesson.

Function Body

The body of the function is what the function does. This is the code that runs each time we execute the function. We indicate that we are writing the function body by going to the next line and indenting after the colon. To complete the function body we stop indenting.

In [10]:
```
def name_of_function():
    words = 'function body' # function body
    print(words) # function body
# no longer part of the function body
```

Let's execute the `name_of_function()` function.

In [9]:
```
name_of_function()
```

```
function body
```

> Press shift + enter

Did it work? Kinda. The lines of our function were run. But our function did not return anything. Functions are designed so that everything inside of them stays inside. So for example, even though we declared the `words` variable, `words` is not available from outside of the function.

In [10]:
```
words
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/tmp/ipykernel_241/2660519150.py in <module>
----> 1 words

NameError: name 'words' is not defined
```

To get something out of the function, we must use the `return` keyword, followed by what we would like to return. Let's declare another function called `other_function()` that has a body which is exactly the same, but has a return statement.

In [11]:
```
def other_function(): # signature
    words = 'returned from inside the function body' # body
    return words
```

In [12]:
```
other_function()
```

Out[12]:
```
'returned from inside the function body'
```

Much better. So with the return statement we returned the string `'returned from inside the function body'`.

> We will learn more on what is available from inside and outside of the function, so, don't worry if it feels a little confusing right now.

See it again

Now let's identify the function signature and function body of our original function, `greet_empoyees()`.

In [15]:
```
def greet_employees(): # function signature
    welcome_messages = [] # begin function body
    for new_employee in new_employees:
        welcome_messages.append("Hi " + new_employee.title() + ", I'm so glad to

    return welcome_messages # return statement

# no longer in function body
```

As you can see, `greet_employees()` has the same components of a function we identified earlier: the function signature, the function body, and the return statement. Each time we call, `greet_employees()`, all of the lines in the body of the function are run. However, only the return value is accessible from outside of the function.

In [13]:
```
greet_employees()
```

Out[13]:
```
["Hi Jan, I'm so glad to be working with you!",
 "Hi Joe, I'm so glad to be working with you!",
 "Hi Avi, I'm so glad to be working with you!"]
```

Summary

In this lesson we saw how using a function allows us to reuse code without rewriting it. We saw that to declare a function we first write the function signature, which consists of the `def` keyword, the function name, parentheses, and a colon. We indicate the body of the function by indenting our code and then writing the code that our function will execute. To execute the function, we write the function's name followed by parentheses. Executing the function will run the lines in the body of the function.