# Week 7

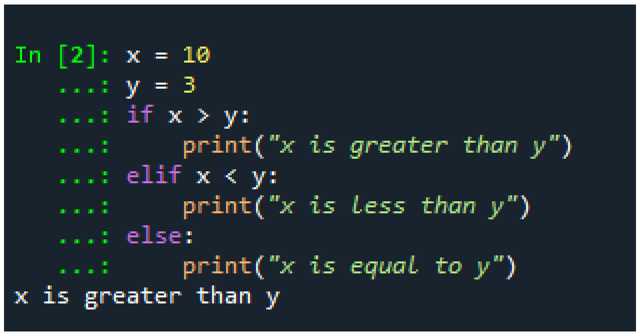
1. If-else
2. Loops
3. Functions
4. Classes

## If-else

The "if" statement is primarily used in controlling the direction of our program. It is used in skipping the execution of certain results that we don't intend to execute.

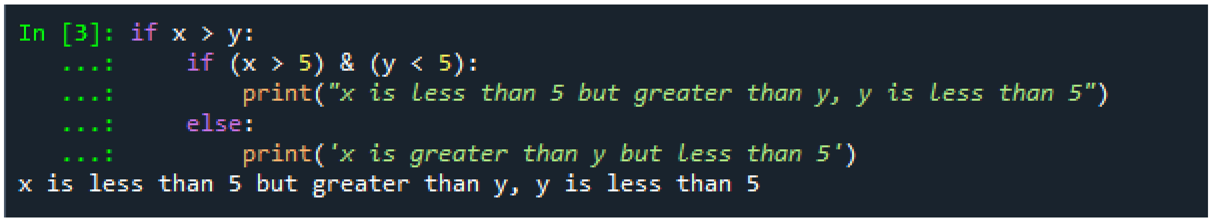
The basic structure of an "if" statement in python is typing the word "if" (lower case) followed by the condition with a colon at the end of the "if" statement and then a print statement regarding printing our desired output. Any additional conditionals within the current control-flow should utilize the "elif" statement. The "else" statement is always optional. It is also possible to nest additional conditional statements within our for-loop. If you have multiple conditions within a single conditional statement, make sure each is enclosed in parenthesis. Python is case sensitive, too, so "if" should be in lower case.

Basic for-loop



```
In [2]: x = 10
   ...: y = 3
   ...: if x > y:
   ...:     print("x is greater than y")
   ...: elif x < y:
   ...:     print("x is less than y")
   ...: else:
   ...:     print("x is equal to y")
x is greater than y
```

Nested for loop (notice the second if-else is indented):



```
In [3]: if x > y:
   ...:     if (x > 5) & (y < 5):
   ...:         print("x is less than 5 but greater than y, y is less than 5")
   ...:     else:
   ...:         print('x is greater than y but less than 5')
x is less than 5 but greater than y, y is less than 5
```

## Loops

Python loops are virtually identical to loops in R. While the syntax is slightly different, the functionality is the same; the object is to iterate and do some type of work. Python supports for loops and while loops. Python allows the programmer to utilize three control statements within their loops:

1 break statement

Terminates the loop statement and transfers execution to the statement immediately following the loop.

2 continue statement

Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

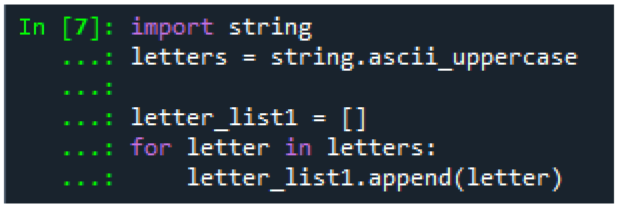
3 pass statement

The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.

### For loop

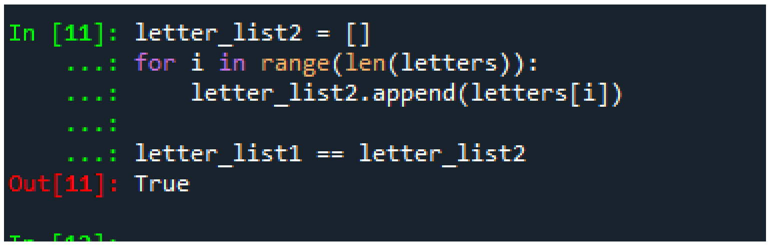
For loops function just like R for loops but have a slightly different syntax. We saw last week that we can iterate over each character of a string and then append those individual values to a list using a for-loop. Like if statements for loops are case sensitive. The basic syntax is: for <iterable> in <object>: do something.

Example 1: In the below example we create a string of letters and iterate over them individually appending them to an empty list, letter_list. Our iterable in this case is letter which is the individual letter at each iteration. This can be called anything and is typically called ‘i’.



```
In [7]: import string
   ...: letters = string.ascii_uppercase
   ...:
   ...: letter_list1 = []
   ...: for letter in letters:
   ...:     letter_list1.append(letter)
```

An equivalent method for writing this is to create a range() object that consists of the starting and ending index of your object. By default, range() will pick up the starting index if you provide the last index element. In most cases you can simply use the length of the object. When you do this your iterable becomes your index position at each iteration so you will need to subset your letters list with that index. Different programming scenarios will require you to use this methodology.

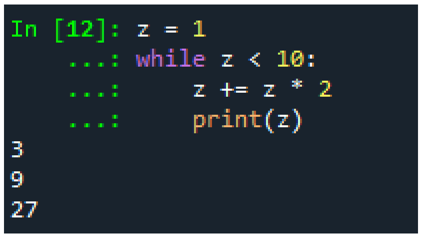


```
In [11]: letter_list2 = []
    ...: for i in range(len(letters)):
    ...:     letter_list2.append(letters[i])
    ...:
    ...: letter_list1 == letter_list2
Out[11]: True
```

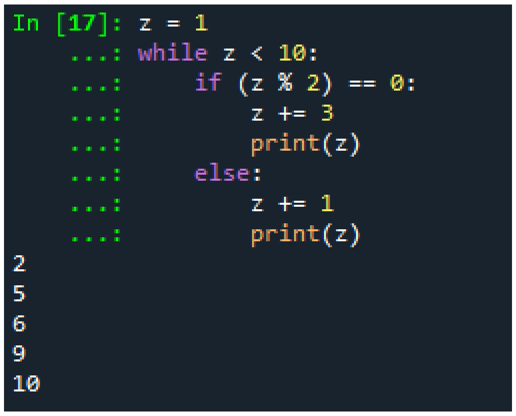
**While loops**

In Python, the while loop statement repeatedly executes a code block while a particular condition is true.

In a while-loop, every time the condition is checked at the beginning of the loop, and if it is true, then the loop's body gets executed. When the condition became False, the controller comes out of the block.



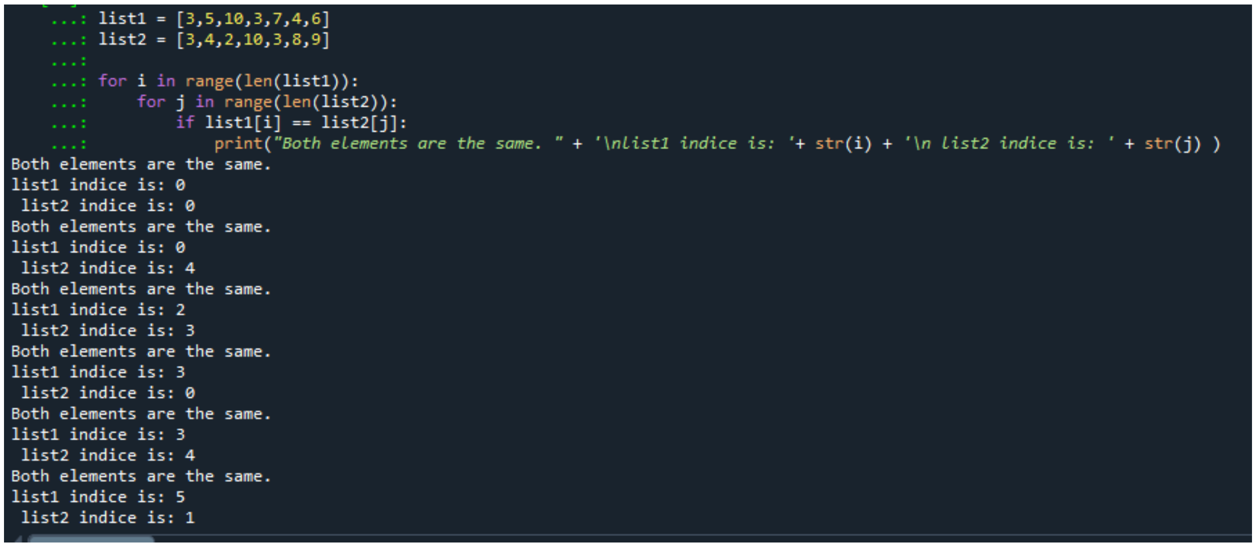
```
In [12]: z = 1
    ...: while z < 10:
    ...:     z += z * 2
    ...:     print(z)
3
9
27
```



```
In [17]: z = 1
    ...: while z < 10:
    ...:     if (z % 2) == 0:
    ...:         z += 3
    ...:         print(z)
    ...:     else:
    ...:         z += 1
    ...:         print(z)
2
5
6
9
10
```

**Nested Loops**

You can nest loops relatively easily in Python. However, remember this is a On^2 operation and we can perform the same task by hashing!



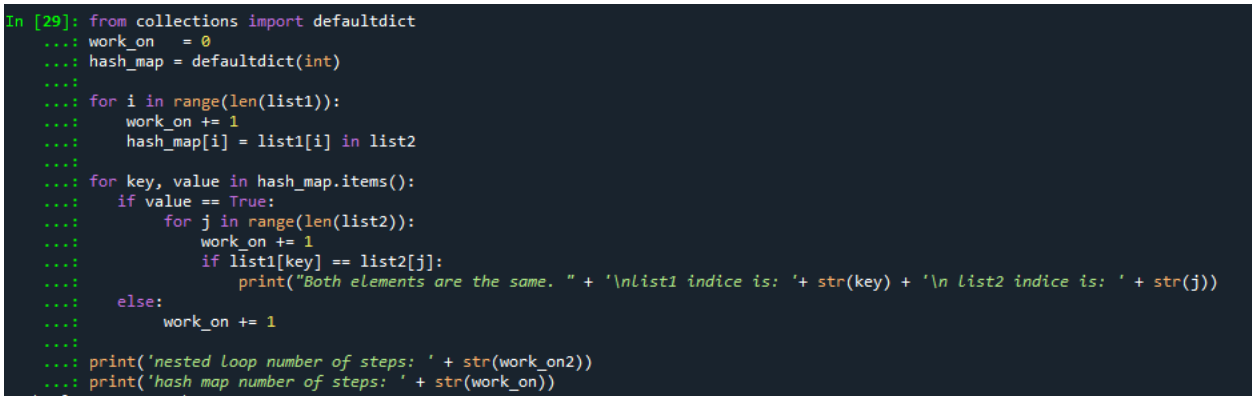
```
    ...: list1 = [3,5,10,3,7,4,6]
    ...: list2 = [3,4,2,10,3,8,9]
    ...:
    ...: for i in range(len(list1)):
    ...:     for j in range(len(list2)):
    ...:         if list1[i] == list2[j]:
    ...:             print("Both elements are the same. " + '\nlist1 indice is: '+ str(i) + '\n list2 indice is: ' + str(j) )
Both elements are the same.
list1 indice is: 0
 list2 indice is: 0
Both elements are the same.
list1 indice is: 0
 list2 indice is: 4
Both elements are the same.
list1 indice is: 2
 list2 indice is: 3
Both elements are the same.
list1 indice is: 3
 list2 indice is: 0
Both elements are the same.
list1 indice is: 3
 list2 indice is: 4
Both elements are the same.
list1 indice is: 5
 list2 indice is: 1
```

**Hashing**

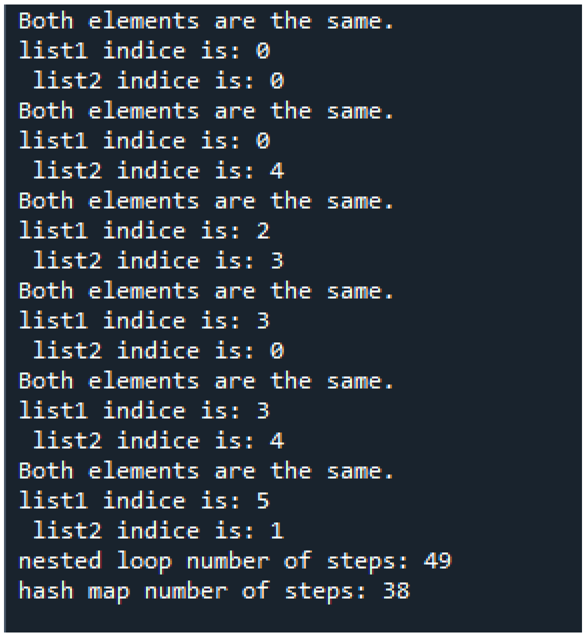
Hashing in Python is much easier than in R due to the defaultdict function we learned last week!

In our previous example we had to scan both arrays to determine which indices matched the other. In the following example using defaultdict and hashing we can use the dictionary to find the indices in list one that match values in list 2. Once we know that a value matches in list 1 to list we can iterate over the key value pairs and list to determine the indices of the matches while avoiding the additional step of iterating over the unmatched values.

(this is your example for iterating over dictionaries 😊 )



```
In [29]: from collections import defaultdict
    ...: work_on    = 0
    ...: hash_map = defaultdict(int)
    ...:
    ...: for i in range(len(list1)):
    ...:     work_on += 1
    ...:     hash_map[i] = list1[i] in list2
    ...:
    ...: for key, value in hash_map.items():
    ...:    if value == True:
    ...:         for j in range(len(list2)):
    ...:             work_on += 1
    ...:             if list1[key] == list2[j]:
    ...:                 print("Both elements are the same. " + '\nlist1 indice is: '+ str(key) + '\n list2 indice is: ' + str(j))
    ...:    else:
    ...:         work_on += 1
    ...:
    ...: print('nested loop number of steps: ' + str(work_on2))
    ...: print('hash map number of steps: ' + str(work_on))
```



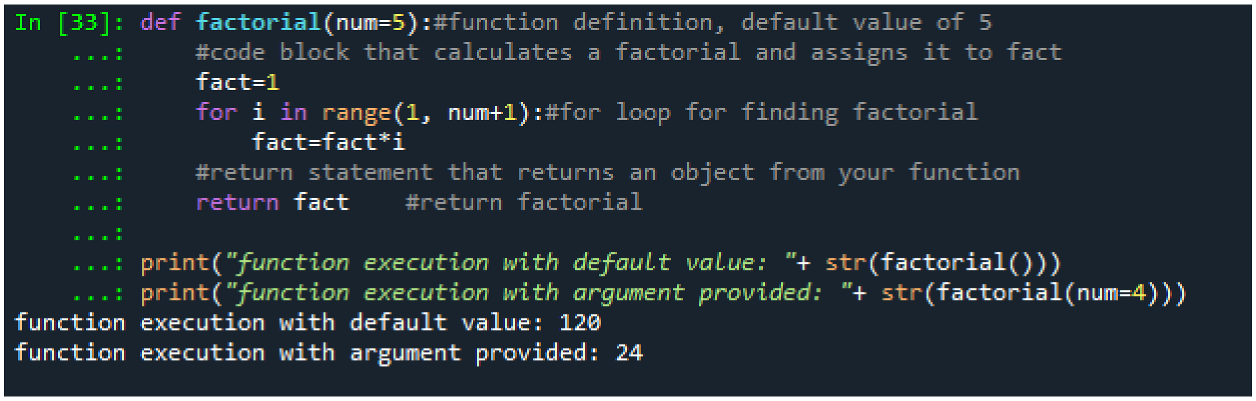
```
Both elements are the same.
list1 indice is: 0
 list2 indice is: 0
Both elements are the same.
list1 indice is: 0
 list2 indice is: 4
Both elements are the same.
list1 indice is: 2
 list2 indice is: 3
Both elements are the same.
list1 indice is: 3
 list2 indice is: 0
Both elements are the same.
list1 indice is: 3
 list2 indice is: 4
Both elements are the same.
list1 indice is: 5
 list2 indice is: 1
nested loop number of steps: 49
hash map number of steps: 38
```

## Functions

Defining a function in Python simply means to create a function. Functions in Python are created and denoted using the keyword def, followed by a function name.

**Function Components:**

1. **Arguments**
   a. Information can be passed into functions as arguments.
   **b.** Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma. Functions can have default values for arguments. To do this you simply define an argument with an equal sign and a default value.
2. **Code block**
   **a.** The code block is the code you are executing within the function
3. **Return statement**
   a. The object or values you are returning as a result of your code block



```
In [33]: def factorial(num=5):#function definition, default value of 5
    ...:     #code block that calculates a factorial and assigns it to fact
    ...:     fact=1
    ...:     for i in range(1, num+1):#for loop for finding factorial
    ...:         fact=fact*i
    ...:     #return statement that returns an object from your function
    ...:     return fact    #return factorial
    ...:
    ...: print("function execution with default value: "+ str(factorial()))
    ...: print("function execution with argument provided: "+ str(factorial(num=4)))
function execution with default value: 120
function execution with argument provided: 24
```

## Classes

In Python, we use classes to create objects. A class is a tool, like a blueprint or a template, for creating objects. It allows us to bundle data and functionality together. Since everything is an object, to create anything, in Python, we need classes. Let us look at the real-life example to understand this more clearly.

Class Components

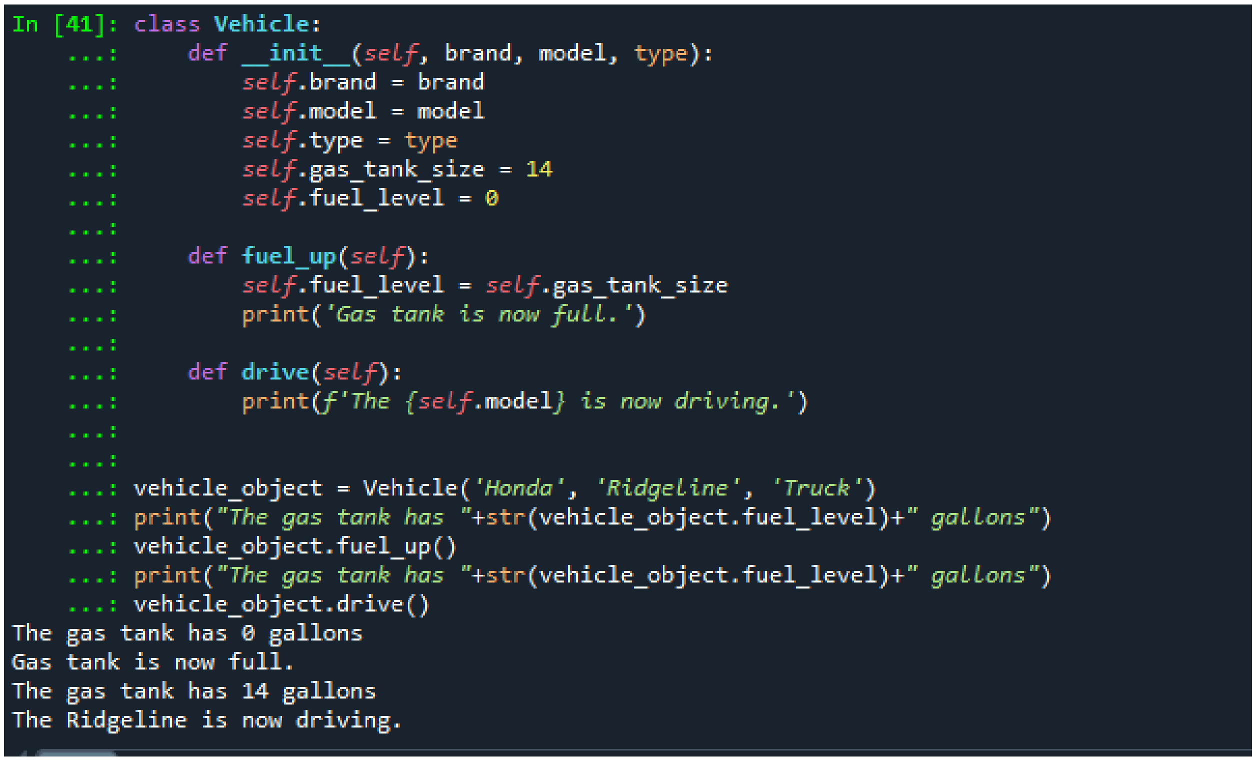
1. **__init__() method in Python**
    a. It is similar to a constructor in Java and C++. It gets executed as soon as we create an object.
    b. The values for the parameters in the __init__() method should be given while creating the object.
2. **Dot operator**
    **a.** In python, the dot operator is used to access variables and functions which are inside the class.
3. **Self**
    a. Almost all methods in the class have an extra parameter called self. It references the current object of the class.
    b. Unlike other parameters, the value of the self is provided automatically by Python. Instead of self, we can name it anything we want as long as it is the first parameter of the method.
4. **Attributes**
    a. Variables that are declared inside a class are called Attributes. Attributes are two types: Class Attributes and Instance Attributes.
        i. Class attributes are variables that are declared inside the class and outside methods. These attributes belong to the class itself and are shared by all objects of the class.

        ii. Instance attributes provide a state to an object. They are declared inside the __init__() method. Instance attributes are unique to each object and are not shared by other objects. We can declare instance attributes in the __init__() method by using the dot operator and the self parameter.

5. **Restricting access**
    a. We can prevent an object from accessing an attribute by prefixing double underscores to the attribute name. When an object tries to access such an attribute, it raises an AttributeError. This applies to both class attributes and instance attributes.
6. **Methods**
    a. Functions inside a class are called Methods. They provide behavior to the class. Most of the methods have an extra parameter called self. We can access them in the same way we access attributes.

## Class Attributes vs Instance Attributes

| Class Attributes | Instance Attributes |
|---|---|
| Shared by all objects of the class. | Not shared by all objects of the class. |
| Same value for all objects. | A Unique value for each object. |
| Declared outside the __init__() method. | Declared inside the __init__() method. |
| Dot operator and self parameter are not needed. | Dot operator and self parameter are needed to declare instance attributes. |
| Can be accessed with the class name and object name. | Can only be accessed with object name. |



```
In [41]: class Vehicle:
    ...:     def __init__(self, brand, model, type):
    ...:         self.brand = brand
    ...:         self.model = model
    ...:         self.type = type
    ...:         self.gas_tank_size = 14
    ...:         self.fuel_level = 0
    ...:
    ...:     def fuel_up(self):
    ...:         self.fuel_level = self.gas_tank_size
    ...:         print('Gas tank is now full.')
    ...:
    ...:     def drive(self):
    ...:         print(f'The {self.model} is now driving.')
    ...:
    ...:
    ...: vehicle_object = Vehicle('Honda', 'Ridgeline', 'Truck')
    ...: print("The gas tank has "+str(vehicle_object.fuel_level)+" gallons")
    ...: vehicle_object.fuel_up()
    ...: print("The gas tank has "+str(vehicle_object.fuel_level)+" gallons")
    ...: vehicle_object.drive()
The gas tank has 0 gallons
Gas tank is now full.
The gas tank has 14 gallons
The Ridgeline is now driving.
```