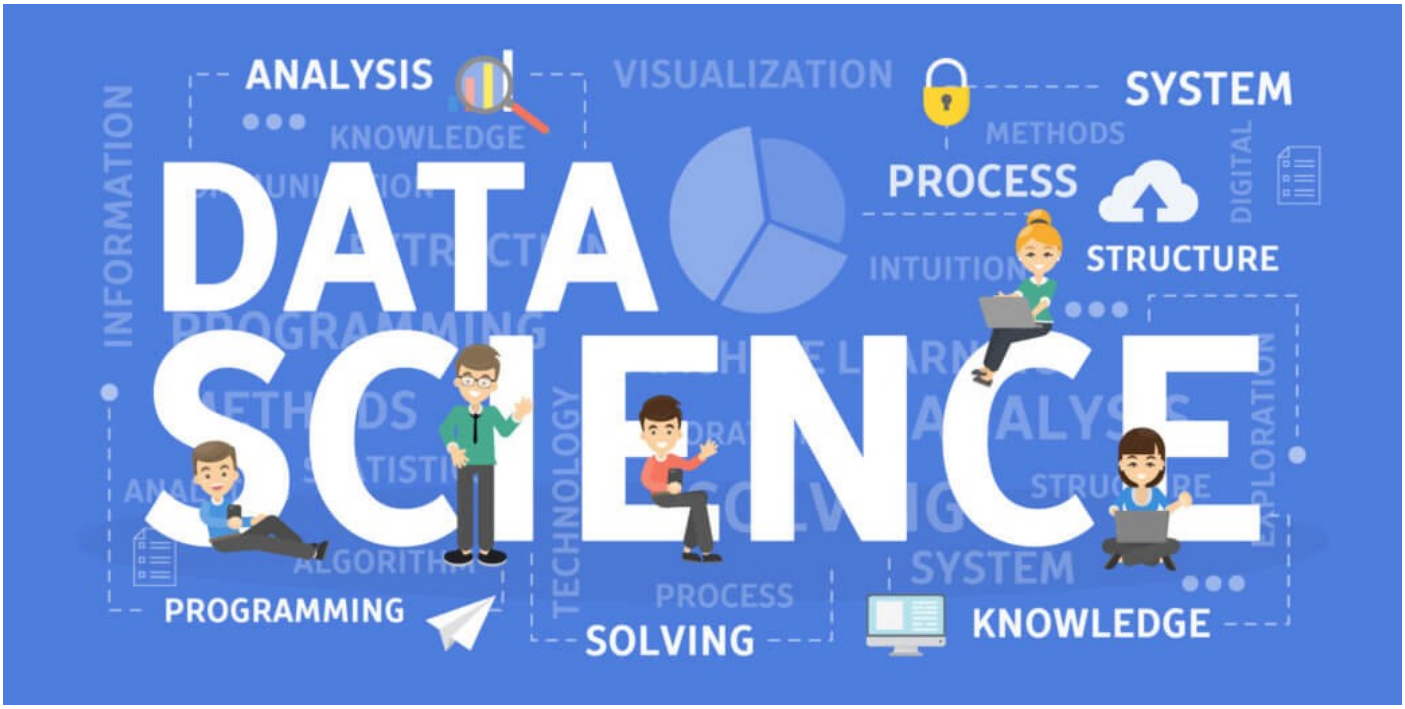
Road to Data Science in 50 Days - Day 9

Python for Data Science



Python

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL). This tutorial gives enough understanding on Python programming language.

Why to Learn Python?

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

Python is a MUST for students and working professionals to become a great Software Engineer specially when they are working in Web Development Domain. I will list down some of the key advantages of learning Python:

- **Python is Interpreted** − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive** − You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented** − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language** − Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

Characteristics of Python

Following are important characteristics of Python Programming −

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

Why Python for Data Science ?

“Data science” is just about as broad of a term as they come. It may be easiest to describe what it is by listing its more concrete components:

Data exploration & analysis.

- Included here: Pandas; NumPy; SciPy; a helping hand from Python’s Standard Library.

Data visualization.

A pretty self-explanatory name. Taking data and turning it into something colorful.

- Included here: Matplotlib; Seaborn; Datashader; others.

Classical machine learning.

Conceptually, we could define this as any supervised or unsupervised learning task that is not deep learning (see below). Scikit-learn is far-and-away the go-to tool for implementing classification, regression, clustering, and dimensionality reduction, while StatsModels is less actively developed but still has a number of useful features.

- Included here: Scikit-Learn, StatsModels.

Deep learning.

This is a subset of machine learning that is seeing a renaissance, and is commonly implemented with Keras, among other libraries. It has seen monumental improvements over the last ~5 years, such as AlexNet in 2012, which was the first design to incorporate consecutive convolutional layers.

- Included here: Keras, TensorFlow, and a whole host of others.

Data storage and big data frameworks.

Big data is best defined as data that is either literally too large to reside on a single machine, or can’t be processed in the absence of a distributed environment. The Python bindings to Apache technologies play heavily here.

- Apache Spark; Apache Hadoop; HDFS; Dask; h5py/pytables.

Odds and ends.

Includes subtopics such as natural language processing, and image manipulation with libraries such as OpenCV.

- Included here: nltk; Spacy; OpenCV/cv2; scikit-image; Cython

Basics

Basics 1:- Variable declaration.

A variable declaration is generally done using the assignment operator (equals to is also called assignment operator). In the example below, x is a variable that is assigned a constant value of 10

In [1]:
```
x = 10
print(x)
```

```
10
```

Basics 2 :- Input operations

We can ask user to input a value

In [2]:
```
x = input('Enter the value of x:- ')
print('The value assigned to x by user is ',x)
```

```
Enter the value of x:- 10
The value assigned to x by user is  10
```

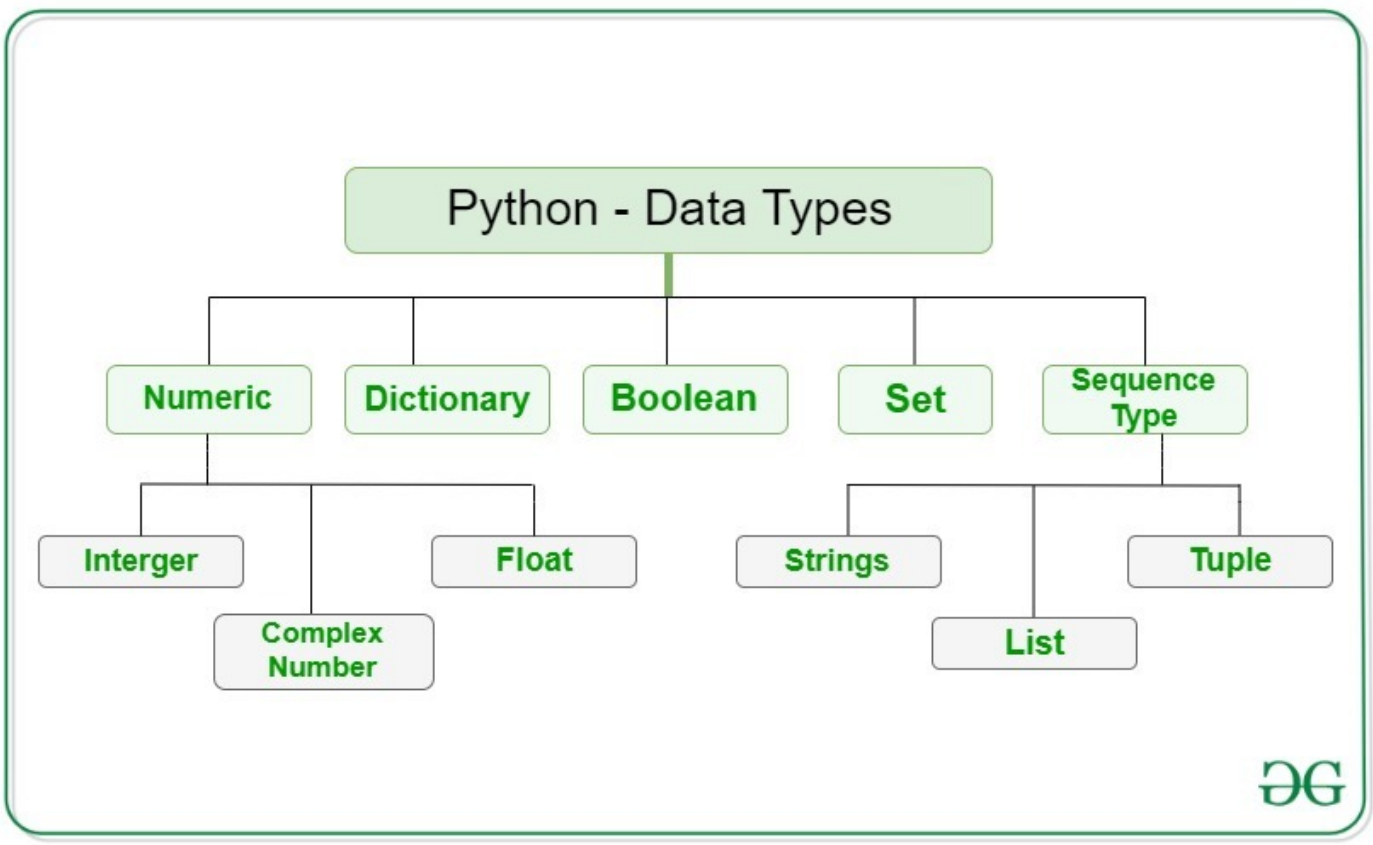
Basics 3 :- Binary operations

In [3]:
```
number1 = 120
number2 = 40
number3 = 7

print('Number obtained by adding number1 and number2 will be ',number1+number2
)
print('Number obtained by subtracting number 2 from number 1 will be ',number1
-number2)
print('Number obtained by multiplying number1 and number2 ',number2*number1)
print('Number obtained by dividing number1 by number2 will be ',number1/number
2)
print('The remainder obtained on dividing number2 by number 3 will be ',number
2%number3)
print('The number obtained on raising 9 to the power of 3 will be ',9**3)
```

```
Number obtained by adding number1 and number2 will be  160
Number obtained by subtracting number 2 from number 1 will be  80
Number obtained by multiplying number1 and number2  4800
Number obtained by dividing number1 by number2 will be  3.0
The remainder obtained on dividing number2 by number 3 will be  5
The number obtained on raising 9 to the power of 3 will be  729
```

Data Structures in Python



We will talk about the 3 main data types used in our daily programming routine in python:-

- Integers
- Float
- Strings

Integer: The integer representation is used for all numbers which can be represented without a decimal.

In [4]:
```
c = 19
print('This is an example of an integer number:- ',c)
```

```
This is an example of an integer number:-  19
```

Float: The float datatype is identified by a decimal notation used in the representation of numbers. The numbers are Real Numbers, integers and fractions depending upon the representation. The way the floating point numbers are shown is below

In [5]:
```
b = 45.6778
print('This is an example of a floating point number',b)
```

```
This is an example of a floating point number 45.6778
```

Strings: The strings are sequences of character datatypes used in Python. They are usually written within quotes or double quotes.

In [6]:
```
a = 'This is a generic string statement used in python.'
print(a)
```

```
This is a generic string statement used in python.
```

Typecasting: Changing the datatype of an object in python is called Typecasting.

For example, changing float to integer or changing integer to float and so on.

We consider these 3 datatypes and demonstrate the typecasting on them

In [7]:
```
strings  = 'a string'
integers = 12
floats   = 14.567

print('Converting a float to integer:-  ',int(floats))
print('Converting an integer to float:- ',float(integers))
print('Converting an integer to string:-',str(integers))
```

```
Converting a float to integer:-   14
Converting an integer to float:-  12.0
Converting an integer to string:- 12
```

Exercise One: Take a string '12' and add 15 to it.

In [8]:
```
# Solution to exercise one
a = '12'
solution = int(a)+15
print('The solution on typecasting a string \'12\' to integer and adding 15 to
it is: ',solution)
```

```
The solution on typecasting a string '12' to integer and adding 15 to it is:
27
```

Exercise Two: Add two floating point numbers (12.909 and 15.607) and round them to an integer value.

In [9]:
```
print('The necessary solution(without rounding) was:- ',(12.909+15.607))
print('The necessary solution(after rounding) is   :- ',int(12.909+15.607))
```

```
The necessary solution(without rounding) was:-  28.516
The necessary solution(after rounding) is   :-  28
```

A Short review about Boolean types: Boolean type is either True of False.

It is commonly used in ascertaining the validity of a particular expression. For example, we know 3 is less than 5 and the outcome of it must be True. A sample demonstration is given below

In [10]:
```
print('Sample demonstration of TRUE boolean type')
print('3<5 is',3<5,'and the datatype of the result is of type', type(3<5))

print('\n')
print('Sample demonstration of False boolean type')
print('14<5 is',14<5,'and the datatype of the result is of type', type(3<5))
```

```
Sample demonstration of TRUE boolean type
3<5 is True and the datatype of the result is of type <class 'bool'>


Sample demonstration of False boolean type
14<5 is False and the datatype of the result is of type <class 'bool'>
```

The Boolean datatype is mainly used in if-else conditions while checking multiple conditions at the same time

Lists

A list is a data structure that holds an ordered collection of items i.e. you can store a sequence of items in a list. This is easy to imagine if you can think of a shopping list where you have a list of items to buy, except that you probably have each item on a separate line in your shopping list whereas in Python you put commas in between them.

In [11]:
```
# This is my shopping list
shoplist = ['apple', 'mango', 'carrot', 'banana']

print('I have', len(shoplist), 'items to purchase.')

print('These items are:', end=' ')
for item in shoplist:
    print(item, end=' ')

print('\nI also have to buy rice.')
shoplist.append('rice')
print('My shopping list is now', shoplist)

print('I will sort my list now')
shoplist.sort()
print('Sorted shopping list is', shoplist)

print('The first item I will buy is', shoplist[0])
olditem = shoplist[0]
del shoplist[0]
print('I bought the', olditem)
print('My shopping list is now', shoplist)
```

```
I have 4 items to purchase.
These items are: apple mango carrot banana
I also have to buy rice.
My shopping list is now ['apple', 'mango', 'carrot', 'banana', 'rice']
I will sort my list now
Sorted shopping list is ['apple', 'banana', 'carrot', 'mango', 'rice']
The first item I will buy is apple
I bought the apple
My shopping list is now ['banana', 'carrot', 'mango', 'rice']
```

How It Works

The variable **shoplist** is a shopping list for someone who is going to the market. In **shoplist**, we only store strings of the names of the items to buy but you can add any kind of object to a list including numbers and even other lists.

We have also used the **for..in** loop to iterate through the items of the list. By now, you must have realised that a list is also a sequence. The speciality of sequences will be discussed in a later section.

Notice the use of the **end** parameter in the call to **print** function to indicate that we want to end the output with a space instead of the usual line break.

Next, we add an item to the list using the **append** method of the list object, as already discussed before. Then, we check that the item has been indeed added to the list by printing the contents of the list by simply passing the list to the **print** function which prints it neatly.

Then, we sort the list by using the **sort** method of the list. It is important to understand that this method affects the list itself and does not return a modified list - this is different from the way strings work. This is what we mean by saying that lists are mutable and that strings are immutable.

Operations of List

- **Append** We can add elements to a list by using append functionality. Append will mean that we are extending a list by an element.
- **Extend** It helps in adding a list to a given list.
- **Insert** We can insert an item at a given index by using this function

  list_name.insert(index,value)
- **Remove** We can remove a value from a list by using the remove functionality
- **Index** We can also return the index of the first matched item of a list.
- **Clear** We can remove the items from a list by using Clear functionality

  list_name.clear()
- **Sort** We can sort a list in whichever order possible using Sort functionality
- **Count** We can get the count of a value from a list using count functinoality

---

Tuple

Tuples are used to hold together multiple objects. Think of them as similar to lists, but without the extensive functionality that the list class gives you. One major feature of tuples is that they are immutable like strings i.e. you cannot modify tuples.

Tuples are defined by specifying items separated by commas within an optional pair of parentheses.

Tuples are usually used in cases where a statement or a user-defined function can safely assume that the collection of values (i.e. the tuple of values used) will not change.

In [12]:
```
# I would recommend always using parentheses to indicate start and end of tupl
e
# even though parentheses are optional.
# Explicit is better than implicit.

zoo = ('python', 'elephant', 'penguin')
print('Number of animals in the zoo is', len(zoo))

new_zoo = 'monkey', 'camel', zoo    # parentheses not required but are a good
 idea
print('Number of cages in the new zoo is', len(new_zoo))
print('All animals in new zoo are', new_zoo)
print('Animals brought from old zoo are', new_zoo[2])
print('Last animal brought from old zoo is', new_zoo[2][2])
print('Number of animals in the new zoo is',
      len(new_zoo)-1+len(new_zoo[2]))
```

```
Number of animals in the zoo is 3
Number of cages in the new zoo is 3
All animals in new zoo are ('monkey', 'camel', ('python', 'elephant', 'pengui
n'))
Animals brought from old zoo are ('python', 'elephant', 'penguin')
Last animal brought from old zoo is penguin
Number of animals in the new zoo is 5
```

How It Works

The variable **zoo** refers to a tuple of items. We see that the **len** function can be used to get the length of the tuple. This also indicates that a tuple is a sequence as well.

We are now shifting these animals to a new zoo since the old zoo is being closed. Therefore, the *new_zoo* tuple contains some animals which are already there along with the animals brought over from the old zoo. Back to reality, note that a tuple within a tuple does not lose its identity.

We can access the items in the tuple by specifying the item's position within a pair of square brackets just like we did for lists. This is called the indexing operator. We access the third item in **new_zoo** by specifying **new_zoo[2]** and we access the third item within the third item in the new_zoo tuple by specifying **new_zoo[2][2]**. This is pretty simple once you've understood the idiom.

Dictionary

A dictionary is like an address-book where you can find the address or contact details of a person by knowing only his/her name i.e. we associate keys (name) with values (details). Note that the key must be unique just like you cannot find out the correct information if you have two persons with the exact same name.

Note that you can use only immutable objects (like strings) for the keys of a dictionary but you can use either immutable or mutable objects for the values of the dictionary. This basically translates to say that you should use only simple objects for keys.

Pairs of keys and values are specified in a dictionary by using the notation **d = {key1 : value1, key2 : value2 }**. Notice that the key-value pairs are separated by a colon and the pairs are separated themselves by commas and all this is enclosed in a pair of curly braces.

Remember that key-value pairs in a dictionary are not ordered in any manner. If you want a particular order, then you will have to sort them yourself before using it.

The dictionaries that you will be using are instances/objects of the **dict** class.

In [13]:
```
# 'ab' is short for 'a'ddress'b'ook

ab = {
    'Swaroop': 'swaroop@swaroopch.com',
    'Larry': 'larry@wall.org',
    'Matsumoto': 'matz@ruby-lang.org',
    'Spammer': 'spammer@hotmail.com'
}

print("Swaroop's address is", ab['Swaroop'])

# Deleting a key-value pair
del ab['Spammer']

print('\nThere are {} contacts in the address-book\n'.format(len(ab)))

for name, address in ab.items():
    print('Contact {} at {}'.format(name, address))

# Adding a key-value pair
ab['Guido'] = 'guido@python.org'

if 'Guido' in ab:
    print("\nGuido's address is", ab['Guido'])
```

```
Swaroop's address is swaroop@swaroopch.com

There are 3 contacts in the address-book

Contact Swaroop at swaroop@swaroopch.com
Contact Larry at larry@wall.org
Contact Matsumoto at matz@ruby-lang.org

Guido's address is guido@python.org
```

How It Works

We create the dictionary **ab** using the notation already discussed. We then access key-value pairs by specifying the key using the indexing operator as discussed in the context of lists and tuples. Observe the simple syntax.

We can delete key-value pairs using our old friend - the **del** statement. We simply specify the dictionary and the indexing operator for the key to be removed and pass it to the **del** statement. There is no need to know the value corresponding to the key for this operation.

Next, we access each key-value pair of the dictionary using the **items** method of the dictionary which returns a list of tuples where each tuple contains a pair of items - the key followed by the value. We retrieve this pair and assign it to the variables **name** and **address** correspondingly for each pair using the **for..in** loop and then print these values in the for-block.

We can add new key-value pairs by simply using the indexing operator to access a key and assign that value, as we have done for Guido in the above case.

We can check if a key-value pair exists using the **in** operator.

Set

Sets are unordered collections of simple objects. These are used when the existence of an object in a collection is more important than the order or how many times it occurs.

Using sets, you can test for membership, whether it is a subset of another set, find the intersection between two sets, and so on.

In [14]:
```
bri = set(['brazil', 'russia', 'india'])

#Check if india is in set
print('india' in bri)

print('usa' in bri)

bric = bri.copy()

bric.add('china')

# Check if Set 2 is SuperSet of Set 1
bric.issuperset(bri)

bri.remove('russia')

#Operations on Set

print('Intersection of Set 1 and Set 2', bri & bric) #OR bri.intersection(bri
c)

print('Union of Set 1 and Set 2', bri | bric) #OR bri.union(bric)
```

```
True
False
Intersection of Set 1 and Set 2 {'brazil', 'india'}
Union of Set 1 and Set 2 {'brazil', 'china', 'india', 'russia'}
```

How It Works

If you remember basic set theory mathematics from school, then this example is fairly self-explanatory. But if not, you can google "set theory" and "Venn diagram" to better understand our use of sets in Python.

References

When you create an object and assign it to a variable, the variable only refers to the object and does not represent the object itself! That is, the variable name points to that part of your computer's memory where the object is stored. This is called binding the name to the object.

Generally, you don't need to be worried about this, but there is a subtle effect due to references which you need to be aware of:

In [15]:
```
print('Simple Assignment')
shoplist = ['apple', 'mango', 'carrot', 'banana']
# mylist is just another name pointing to the same object!
mylist = shoplist

# I purchased the first item, so I remove it from the list
del shoplist[0]

print('shoplist is', shoplist)
print('mylist is', mylist)
# Notice that both shoplist and mylist both print
# the same list without the 'apple' confirming that
# they point to the same object

print('Copy by making a full slice')
# Make a copy by doing a full slice
mylist = shoplist[:]
# Remove first item
del mylist[0]

print('shoplist is', shoplist)
print('mylist is', mylist)
# Notice that now the two lists are different
```

```
Simple Assignment
shoplist is ['mango', 'carrot', 'banana']
mylist is ['mango', 'carrot', 'banana']
Copy by making a full slice
shoplist is ['mango', 'carrot', 'banana']
mylist is ['carrot', 'banana']
```

How It Works

Most of the explanation is available in the comments.

Remember that if you want to make a copy of a list or such kinds of sequences or complex objects (not simple objects such as integers), then you have to use the slicing operation to make a copy. If you just assign the variable name to another name, both of them will "refer" to the same object and this could be trouble if you are not careful.

More About Strings

We have already discussed strings in detail earlier. What more can there be to know? Well, did you know that strings are also objects and have methods which do everything from checking part of a string to stripping spaces? In fact, you've already been using a string method... the format method!

The strings that you use in programs are all objects of the class str. Some useful methods of this class are demonstrated in the next example. For a complete list of such methods, see help(str).

In [16]:
```
# This is a string object
name = 'Swaroop'

if name.startswith('Swa'):
    print('Yes, the string starts with "Swa"')

if 'a' in name:
    print('Yes, it contains the string "a"')

if name.find('war') != -1:
    print('Yes, it contains the string "war"')

delimiter = '_*_'
mylist = ['Brazil', 'Russia', 'India', 'China']
print(delimiter.join(mylist))
```

```
Yes, the string starts with "Swa"
Yes, it contains the string "a"
Yes, it contains the string "war"
Brazil_*_Russia_*_India_*_China
```

How It Works

Here, we see a lot of the string methods in action. The **startswith** method is used to find out whether the string starts with the given string. The **in** operator is used to check if a given string is a part of the string.

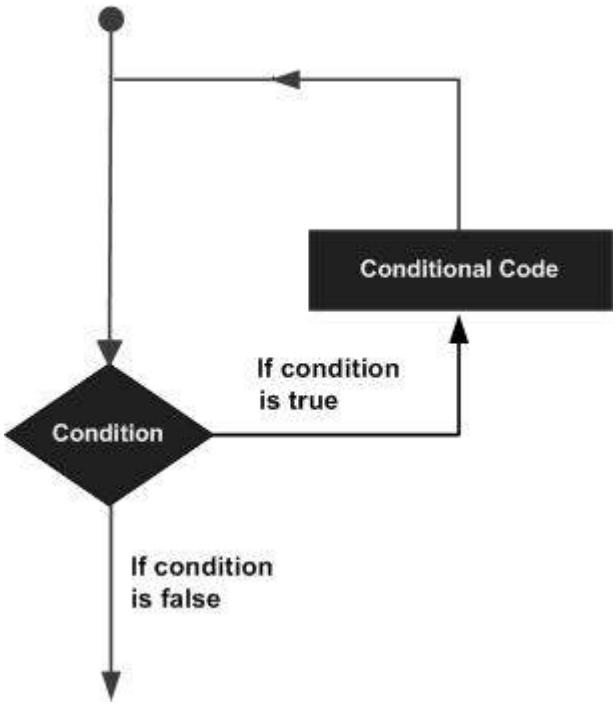
The **find** method is used to locate the position of the given substring within the string; **find** returns -1 if it is unsuccessful in finding the substring. The **str** class also has a neat method to **join** the items of a sequence with the string acting as a delimiter between each item of the sequence and returns a bigger string generated from this.

Loops

In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on. There may be a situation when you need to execute a block of code several number of times.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times. The following diagram illustrates a loop statement −



Python programming language provides following types of loops to handle looping requirements

- **While loop**: Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.
- **For loop**: Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
- **Nested loop**: You can use one or more loop inside any another while, for or do..while loop.

Loop Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

Python supports the following control statements. Click the following links to check their detail.

- **break statement**: Terminates the loop statement and transfers execution to the statement immediately following the loop.

- **continue statement**: Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
- **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.

Let's see some practical examples:

In [17]:
```
# We take a list and try to print all the elements of that list using a for lo
op

a = [1,2,3,4,5,6,7,8,9,10]
for i in range(0,len(a),1):
    print('index:- ',i,' values:- ',a[i])
```

```
index:-  0  values:-  1
index:-  1  values:-  2
index:-  2  values:-  3
index:-  3  values:-  4
index:-  4  values:-  5
index:-  5  values:-  6
index:-  6  values:-  7
index:-  7  values:-  8
index:-  8  values:-  9
index:-  9  values:-  10
```

Meaning of the above statement

- Here i is the index of the loop which will allow the loop to iterate through the list. The value of the index will tell the location in the list for example if i = 2 then we will access a[2]
- The for loop is written in following format :- for i in range(start_value,end_value,increment)
- The start value will tell the for loop as to where it shall tell the index i to begin.
- The end_value will tell the for loop as to where it shall tell the index i to stop.
- The increment will tell the for loop to tell the index i as to what is the length of steps it needs to take

An exmple to print the datatype of elements in a list

In [18]:
```
sample_list = [112,1,45,'vampires','good vampires','blood thirsty vampires','h
ybrid vampires',12.22,90.999,93.2,'original vampires']
for i in range(0,len(sample_list),1):
    print('value ',sample_list[i],'          Data Type',type(sample_list[i]))
```

```
value  112           Data Type <class 'int'>
value  1           Data Type <class 'int'>
value  45           Data Type <class 'int'>
value  vampires           Data Type <class 'str'>
value  good vampires           Data Type <class 'str'>
value  blood thirsty vampires           Data Type <class 'str'>
value  hybrid vampires           Data Type <class 'str'>
value  12.22           Data Type <class 'float'>
value  90.999           Data Type <class 'float'>
value  93.2           Data Type <class 'float'>
value  original vampires           Data Type <class 'str'>
```

An example to print all numbers between 0 to 10 except 7

In [19]:
```
for i in range(0,11,1):
    if i!=7:
        print(i)
```

```
0
1
2
3
4
5
6
8
9
10
```

An example to get all odd numbers between 1 to 20

In [20]:
```
odd_numbers = []
for i in range(1,21,2):
    odd_numbers.append(i)
print('Odd numbers between 0 and 20 are ',odd_numbers)
```

```
Odd numbers between 0 and 20 are  [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
```

Multiplication table.

In [21]:
```
number = int(input('Enter a number for which you need a multiplication table:-
'))
for i in range(11):
    print(number,' x',i,' = ',number*i)
```

```
Enter a number for which you need a multiplication table:- 9
9  x 0  =  0
9  x 1  =  9
9  x 2  =  18
9  x 3  =  27
9  x 4  =  36
9  x 5  =  45
9  x 6  =  54
9  x 7  =  63
9  x 8  =  72
9  x 9  =  81
9  x 10  =  90
```

Assignment

1. Print all the leap years between 2 years taken as input
2. Print Fibonacci series using for loop.

Solve some more problems to practise python basics more from: https://www.w3resource.com/python-exercises/python-basic-exercises.php?passed=passed (https://www.w3resource.com/python-exercises/python-basic-exercises.php?passed=passed)

References and Links to Refer

Python tutorials by TutorialsPoint: https://www.tutorialspoint.com/python/index.htm (https://www.tutorialspoint.com/python/index.htm)

Python Datastructures: https://python.swaroopch.com/data_structures.html (https://python.swaroopch.com/data_structures.html)

Video Tutorials:

Python Tutorials by Corey Schafer (I learned from him): https://www.youtube.com/watch?v=YYXdXT2l-Gg&list=PL-osiE80TeTt2d9bfVyTiXJA-UTHn6WwU (https://www.youtube.com/watch?v=YYXdXT2l-Gg&list=PL-osiE80TeTt2d9bfVyTiXJA-UTHn6WwU)

Python Tutorials by Sentdex: https://www.youtube.com/watch?v=oVp1vrfL_w4&list=PLQVvvaa0QuDe8XSftW-RAxdo6OmaeL85M (https://www.youtube.com/watch?v=oVp1vrfL_w4&list=PLQVvvaa0QuDe8XSftW-RAxdo6OmaeL85M)

Python Tutorials by Telusko(Indian Author - Recommended): https://www.youtube.com/watch?v=QXeEoD0pB3E&list=PLsyeobzWxl7poL9JTVyndKe62ieoN-MZ3 (https://www.youtube.com/watch?v=QXeEoD0pB3E&list=PLsyeobzWxl7poL9JTVyndKe62ieoN-MZ3)