The Complete Python Learning

Python is a **dynamically typed programming language** This means, it automatically interprets the details while executing.
It is an open source software.
Python can be executed in 2 ways:

1. Shell Scripting
    - Cannot write multiple lines of code
    - Cannot save the code
    - Cannot share the code
    - This brings the need for Text Editors
2. IDE (Shell + Text editor)
    - Well known IDEs:
        - Jupyter Notebook, Google Colaborator: Saves file as ipynb (Python Notebook)
        - Spyder, Pycharm: Saves file as .py (Python File)

Markdown: Anything used for headings & text in a notebook.
Single line comment: Anything used after # in a code is a single line comment. (Example below)
Paragraph Comment: Cntl + / is used to comment complete selected paragraph

help(function) gives the details on how to use the function

In [1]:

```
# This is a comment written by Sourabh and will not be executed
```

Primitive Data Types

1. **bool (Boolean):** Contacts 0/False and 1/True
2. **int (Integer):** Contains all integers including negative, positive and 0
3. **float (Decimals):** Contains all decimals
4. **complex:** In the form of 1+2x
5. **str (String):** Contains characters incl. alphabel, symbol, words, sentences, para. Syntax: "this is a string"

Sequential Data Types (Covered after operators)

In [2]:

```
x = True
print(x) # used to return the data
print(type(x)) # type is used to return datatype for given variable
```

```
True
<class 'bool'>
```

In [3]:

```
x = 12
print(x)
print(type(x))
```

```
12
<class 'int'>
```

In [4]:

```
x = 2.4
print(x)
print(type(x))
```

```
2.4
<class 'float'>
```

In [5]:

```
x = 'python'
print(x)
print(type(x))
```

```
python
<class 'str'>
```

In [6]:

```
x = 2+3j
print(x)
print(type(x))
```

```
(2+3j)
<class 'complex'>
```

Strings

In [7]:

```
x = 'python'
y = "Python"
print(x,y)
```

```
python Python
```

Note: If we wish to use one of the quotes(') or double quotes(") within a string, we can use the other one to start and close the string.
If we are required to use both quotes(') and double quotes(") together within a string, then we are required to use escape sequence character, which is "\".
Escape Sequence: \ is used as an escape sequence. It means that it will not execute the next character. \n is used for next line character. \t is used for tab. See example below:

In [8]:

```
sent1 = "python's awesome"
```

In [9]:

```
sent1 = 'python\'s awesome'
print(sent1)
```

```
python's awesome
```

```
In [10]: sent1 = "python\'s \\awesome"
         print(sent1)
```

```
python's \awesome
```

Writing a paragraph: Triple double quotes ("""Paragraph""") are used to write a paragraph.

```
In [11]: poem = """twinkle twinkle little star
         how I wonder What you are"""
```

```
In [12]: print(poem)
```

```
twinkle twinkle little star
how I wonder What you are
```

String Concatenation: If we wish to print 2 or more strings together, it can be done by using comma seperated variables in print function.
If we wish to concat and assign strings to a variable, + operator can be used to do that. However, it does not put a space between them. (Refer example below)

```
In [13]: str1 = "It's"
         str2 = 'good'
         str3 = 'example'
```

```
In [14]: print(str1,str2,str3)
```

```
It's good example
```

```
In [15]: print(str1+str2+str3)
```

```
It'sgoodexample
```

```
In [16]: x = 2
```

```
In [17]: print('The value from operation is',x)
```

```
The value from operation is 2
```

String indexing and slicing: We can select a particular part of string with usage of indexes. Indexing starts from first character of string and starts with 0.\

Negative Indexing: For larger datasets, if we wish to index from reverse, we can start indexing from last character with index -1. Note: If nothing is mentioned before or after colon, it takes 0 or last index as default

In [18]:

```python
var1 = 'Python Language'
print(var1[0])
print(var1[0:3])
print(var1[2:])
print(var1[:4])
print(var1[-3])
print(var1[-3:-1])
print(var1[-3:])
```

```
P
Pyt
thon Language
Pyth
a
ag
age
```

Len Function is used to print the length of the provided variable.
Spaces are also considered in length.

In [19]:

```python
len(var1)
```

Out[19]:

```
15
```

Usage of string methods

In [20]:

```python
var1.capitalize() # Used to capitalize only first letter
```

Out[20]:

```
'Python language'
```

In [21]:

```python
var1.upper() # Converts complete string to upper case
```

Out[21]:

```
'PYTHON LANGUAGE'
```

In [22]:

```python
var1.lower() # Converts complete string to lower case
```

Out[22]:

```
'python language'
```

In [23]:

```python
var1 = var1.replace('a','qq') # replaces second arg with the first. Syntax: va
```

In [24]:

```python
var1
```

Out[24]:

```
'Python Lqqnguqqge'
```

In [25]:

```python
var1.count('g') #Returns the no. of occurances of arg. Syntax: var.count("what_
```

Out[25]:

```
2
```

In [26]:

```python
var1.find('t') #Returns the index of arg. Syntax: var.find("what_to_find")
```

Out[26]:

```
2
```

```
In [27]: var1*3 #gives the string 3 times without spaces
```

```
Out[27]: 'Python LqqnguqqgePython LqqnguqqgePython Lqqnguqqge'
```

We use the **input function** to take the input from user dynamically.

```
In [28]: name = input("Enter the name: ")
```

```
Enter the name: Sourabh
```

```
In [29]: print(name)
```

```
Sourabh
```

```
In [30]: print(name.strip()) #Strip function is used to remove any heading or tailing sp
```

```
Sourabh
```

TypeCasting

```
In [31]: x * 2
```

```
Out[31]: 4
```

```
In [32]: str(2) * 2 # It gives 22 as (string 2) x2 is 22 and (int2)x2 is 4
```

```
Out[32]: '22'
```

bool --> int --> float --> complex --> string

float --> int --> bool

```
In [33]: str(complex(float(int(True))))
```

```
Out[33]: '(1+0j)'
```

```
In [34]: float(complex('2')) #Gives error as it is not possible to change complex to fl
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[34], line 1
----> 1 float(complex('2'))

TypeError: float() argument must be a string or a real number, not 'complex'
```

Practice question: Find the area of circle with radius taken from user

In [35]:

```
r = input("Please enter radius: ")
area = 3.14*float(r)*float(r)
print("Area of the given circle is: ", area)
```

```
Please enter radius: 3
Area of the given circle is:  28.259999999999998
```

Operators

1. **Arithimatic Operators:** +,-,,/,//,%,*

   - / - division gives float value
   - // - Division gives gives int value, called as **floor division**
   - % - Gives remainder after division, called as **Mod operator**
   - ** - exponent

2. **Assignment Operators:** =, +=, -=, *=, /=

   - x += 3 stands for x=x+3 and similar for other assignment operators.

3. **Relational Operators:** ==, !=, >=, <=, >, > etc.

   - Returns boolean value only

4. **Logical Operators:** and, or, not

   - Used to make decisions generally
   - Returns boolean value only

5. **Bitwise Operators:** &, !

   - Changes to binary and performs operation
   - & is bitwise whereas and is logical

6. **Identity Operator:** is, is not

   - is: Returns true if the variables are matching, else false.
   - Syntax: a is b

7. **Membership Operator:** in, not in

   - returns true if first variable is part/sub-part of 2nd variable.
   - Syntax: x in y

Arithimatic Operators Example

In [36]:

```
2 ** 3 # exponential
```

Out[36]:

```
8
```

In [37]:

```
10 % 5 # remainder
```

Out[37]:

```
0
```

In [38]:
```
9 / 2
```
Out[38]:
```
4.5
```

In [39]:
```
9 // 2 # floor division
```
Out[39]:
```
4
```

Assignment Operator Example

In [40]:
```
# x = x+2
x += 2
```

Relational Operators Example

In [41]:
```
a = 10
b = 5
```

In [42]:
```
print(a > b)
print(a < b)
print(a >= b)
print(a <= b)
print(a == b) # equal to
print(a != b) # Not equal to
```

```
True
False
True
False
False
True
```

Logical Operators Example

In [43]:
```
print((a>2) and (b>1))
```

```
True
```

In [44]:
```
not True
```
Out[44]:
```
False
```

In [45]:
```
not (a >11)
```
Out[45]:
```
True
```

Bitwise Operators Example

```
In [46]: print((a>2) and (b>1))

         True
```

```
In [47]: print((a>2) & (b>1)) # bitwise operator

         True
```

```
In [48]: 10 and 5

Out[48]: 5
```

```
In [49]: a & b

Out[49]: 0
```

Convert from Binary to decimal and reverse.

Bin(var)[2:] changes var to binary
int("binary_var",2) changes var to int

```
In [50]: bin(a)[2:]

Out[50]: '1010'
```

```
In [51]: bin(b)[2:]

Out[51]: '101'
```

```
In [52]: a | b

Out[52]: 15
```

```
In [53]: int('1111',2)

Out[53]: 15
```

Identity Operator Example

```
In [54]: 1 is 1

         <>:1: SyntaxWarning: "is" with a literal. Did you mean "=="?
         <>:1: SyntaxWarning: "is" with a literal. Did you mean "=="?
         C:\Users\srbhk\AppData\Local\Temp\ipykernel_13832\1033434568.py:1: SyntaxWarn
         ing: "is" with a literal. Did you mean "=="?
           1 is 1

Out[54]: True
```

```
In [55]: 'Hello' is 'hello'
```

```
<>:1: SyntaxWarning: "is" with a literal. Did you mean "=="?
<>:1: SyntaxWarning: "is" with a literal. Did you mean "=="?
C:\Users\srbhk\AppData\Local\Temp\ipykernel_13832\2248926702.py:1: SyntaxWarn
ing: "is" with a literal. Did you mean "=="?
  'Hello' is 'hello'
```

```
Out[55]: False
```

Membership Operator Example

```
In [56]: 'x' in 'python'
```

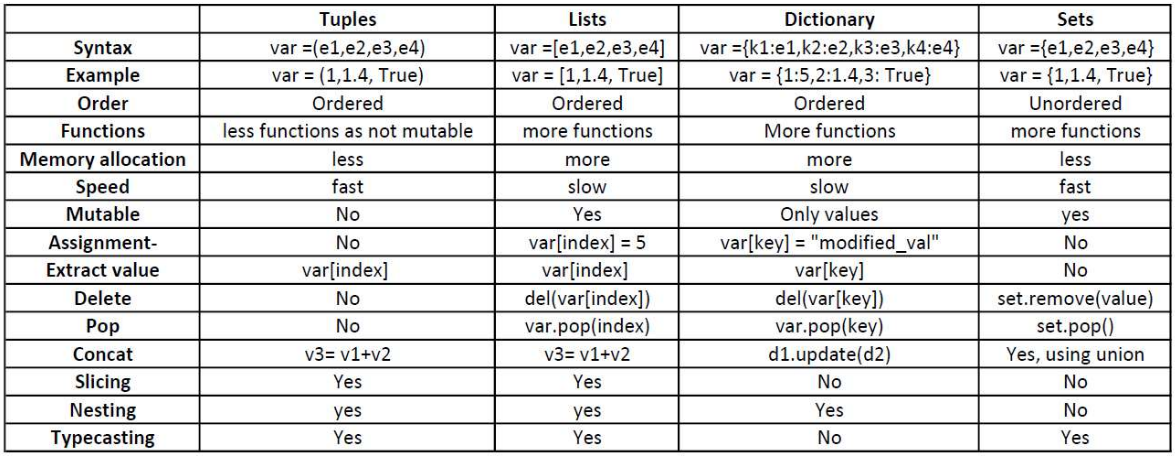
```
Out[56]: False
```

```
In [57]: 'x' not in 'python'
```

```
Out[57]: True
```

Sequential Data Types

1. Tuples
2. Lists
3. Dictionary
4. Sets



	Tuples	Lists	Dictionary	Sets
Syntax	var =(e1,e2,e3,e4)	var =[e1,e2,e3,e4]	var ={k1:e1,k2:e2,k3:e3,k4:e4}	var ={e1,e2,e3,e4}
Example	var = (1,1.4, True)	var = [1,1.4, True]	var = {1:5,2:1.4,3: True}	var = {1,1.4, True}
Order	Ordered	Ordered	Ordered	Unordered
Functions	less functions as not mutable	more functions	More functions	more functions
Memory allocation	less	more	more	less
Speed	fast	slow	slow	fast
Mutable	No	Yes	Only values	yes
Assignment-	No	var[index] = 5	var[key] = "modified_val"	No
Extract value	var[index]	var[index]	var[key]	No
Delete	No	del(var[index])	del(var[key])	set.remove(value)
Pop	No	var.pop(index)	var.pop(key)	set.pop()
Concat	v3= v1+v2	v3= v1+v2	d1.update(d2)	Yes, using union
Slicing	Yes	Yes	No	No
Nesting	yes	yes	Yes	No
Typecasting	Yes	Yes	No	Yes

Lists

Methods used for both lists and tuples

1. **var.count('element_to_count')** gives the count of provided element
2. **var.index('key_to_find')** gives the 1st index of the key in the list/tuple.
3. **Typecasting:** We can convert list to tuple and vice versa. Ex List1 = list(tuple_to_convert)

4. **Accessing nested lists/tuples:** We can access nested objects with help of sub-indexes. Example: A[3][2] returns 3rd element of 4th object in A.

Methods used for lists

1. **list.append(object)** adds a single object(list/tuple/int/char etc.) at the end of the list. Increases length by 1 only.
2. **list.extend(object)** extends or concats the list with as many objects passed (list1+list2 can also be used)
3. **list.insert(index, object)** inserts the object at the mentioned index
4. **del(var[index])** deletes the object at the mentioned index
5. **list.pop(index)** removes the element and also prints it from the mentioned index. If no index mentioned(empty arg.), it pops the last element
6. **list.split(separator)** Seperates the list after each occurance of separator
7. **list.sort(reverse=True|False, key=myFunc)** Sorts the list. Reverse=True will sort the list descending. Default is reverse=False which sorts in ascending. key is a function to specify the sorting criteria
8. **Copying list:** B = A[:]
9. **Ammending an element in a nested list:** We can change or add element in a nested list. Ex: list[1][2].append(element_to_add)

In [58]:
```
list2 = [1,1.2,'python',True]
tup = tuple(list2)
```

In [59]:
```
list2[2] = 'Java' #Assigning the list index 2
print(list2)
```

```
[1, 1.2, 'Java', True]
```

In [60]:
```
list2.append('python') # it would add only one element in last position
```

In [61]:
```
list2.insert(1,200) # by using index value we can insert the required data
list2
```

Out[61]:
```
[1, 200, 1.2, 'Java', True, 'python']
```

In [62]:
```
list2.pop() # removes last element in list and returns it
```

Out[62]:
```
'python'
```

In [63]:
```
list2.pop(3)
```

Out[63]:
```
'Java'
```

In [64]:
```
list1 = ['a','b','c','d']
list2 = [1,2,3,4]
```

In [65]:

```
list3 = list1 + list2
print(list3)
```

```
['a', 'b', 'c', 'd', 1, 2, 3, 4]
```

In [66]:

```
list1.extend(list2)
```

In [67]:

```
list1
```

Out[67]:

```
['a', 'b', 'c', 'd', 1, 2, 3, 4]
```

In [68]:

```
list1.append(list2)
list1
```

Out[68]:

```
['a', 'b', 'c', 'd', 1, 2, 3, 4, [1, 2, 3, 4]]
```

In [69]:

```
list1[8][1]
```

Out[69]:

```
2
```

In [70]:

```
list1 = ['a','b','b','c','d','d']
list1.count('b')
```

Out[70]:

```
2
```

In [71]:

```
list1 = [20,40,1,-5,100]
list1.sort()
list1
```

Out[71]:

```
[-5, 1, 20, 40, 100]
```

In [72]:

```
list1.sort(reverse = True)
list1
```

Out[72]:

```
[100, 40, 20, 1, -5]
```

In [73]:

```
tup1 = (1,2,3,4)
type(tup1)
```

Out[73]:

```
tuple
```

In [74]:

```
list(tup1) #Typecasting
```

Out[74]:

```
[1, 2, 3, 4]
```

Practice Questions

In [75]:

```
list1 = [10,20,[30,40,[50,60,70],80],90,100]
# add number 72 after 70 in list1
list1[2][2].append(72)
list1
```

Out[75]:

```
[10, 20, [30, 40, [50, 60, 70, 72], 80], 90, 100]
```

In [76]:

```python
# write a program to find the value 3 in list and if its present replace it wi
# 300
list1 = [1,2,3,4,5,6,3,4]
x= list1.index(3)
list1[x]= 300
list1
```

Out[76]:

```
[1, 2, 300, 4, 5, 6, 3, 4]
```

Tuples

In [77]:

```python
tup2 = (1,1.2,'python',True)
print(tup2)
```

```
(1, 1.2, 'python', True)
```

In [78]:

```python
tup2[2]
```

Out[78]:

```
'python'
```

In [79]:

```python
tup2[2] = 'Java' #Gives error as tuple does not support assignment
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[79], line 1
----> 1 tup2[2] = 'Java'

TypeError: 'tuple' object does not support item assignment
```

Dictionary

1. Have unique and immutable keys
2. Values are mutable
3. Lists, tuples, Sets or Dict can be nested

Functions and methods for Dictionary

1. **dict[key] = 'new_value'** can be used to access and assign value
2. **key in dict** finds the mentioned key. If not found, returns -1
3. **dict.keys()** returns all keys in the dictionary
4. **dict.values()** returns all values in the dictionary
5. **del(dict[key])** deletes the mentioned key and corrosponding value
6. **dict.pop(key)** pops the corrosponding key and value
7. **dict[new_key]= [values]** is used to add new key and corrosponding values

In [80]:

```python
dict1 = {'names': 'xyz','age': 20}
```

```
In [81]: print(dict1)

         {'names': 'xyz', 'age': 20}
```

```
In [82]: dict1 = {'names': ['xyz','pqr'],'age': [20,24]}
```

```
In [83]: print(dict1)

         {'names': ['xyz', 'pqr'], 'age': [20, 24]}
```

```
In [84]: dict1.pop('age')

Out[84]: [20, 24]
```

```
In [85]: dict1

Out[85]: {'names': ['xyz', 'pqr']}
```

```
In [86]: dict1['age'] = [20,24] #Adding a new key and corrosponding values
```

```
In [87]: print(dict1)

         {'names': ['xyz', 'pqr'], 'age': [20, 24]}
```

```
In [88]: dict1.keys()

Out[88]: dict_keys(['names', 'age'])
```

```
In [89]: dict1.values()

Out[89]: dict_values([['xyz', 'pqr'], [20, 24]])
```

```
In [90]: dict1['age'] = 40
```

```
In [91]: dict1

Out[91]: {'names': ['xyz', 'pqr'], 'age': 40}
```

Sets

1. Unordered- do not record position
2. Has unique values - removed duplicates even if we assign
3. No item assignment as unordered

Functions and methods of sets

1. **set.update(value)** or **set.add(value)** adds mentioned value to the set
2. **set.remove(value)** removes mentioned value from the set
3. **set.pop()** pops random value from set
4. **set1= set(list)** can be used to **typecast**
5. **set1.union(set2)** gives the union of 2 sets

6. **set1.intersection(set2)** gives the intersection of 2 sets
7. **set1.difference(set2)** or **set1 - set2** gives the difference of 2 sets i.e. removes elements from set1, which are also present in set2.
8. **set1.symmetric_difference(set2)** gives the union of both sets and removes the intersection part
9. **set1.issubset(set2)** - returns True if set1 is subset of set2
10. **set1.issuperset(set2)** - returns True if set1 is superset of set2
11. **value in set1** - returns true if mentioned value is present in set

```
In [92]: set1 = {'a','b','c','d'}
         set1

Out[92]: {'a', 'b', 'c', 'd'}
```

```
In [93]: set1[0] = 'b' ##Gives error as item assignment not possible

         ---------------------------------------------------------------------------
         TypeError                                 Traceback (most recent call last)
         Cell In[93], line 1
         ----> 1 set1[0] = 'b'

         TypeError: 'set' object does not support item assignment
```

```
In [94]: set1.add('d')
         set1

Out[94]: {'a', 'b', 'c', 'd'}
```

```
In [95]: set1 = {'a','b','b','c','d'}
         set2 = {'c','d','e','f'}
         print(set1,set2)

         {'d', 'b', 'a', 'c'} {'e', 'd', 'f', 'c'}
```

```
In [96]: set1.update((20,))
         set1

Out[96]: {20, 'a', 'b', 'c', 'd'}
```

```
In [97]: set1.remove(20)
         set1

Out[97]: {'a', 'b', 'c', 'd'}
```

```
In [98]: print(set1,set2)

         {'d', 'a', 'c', 'b'} {'e', 'd', 'f', 'c'}
```

```
In [99]: set1.union(set2) # combines everything

Out[99]: {'a', 'b', 'c', 'd', 'e', 'f'}
```

```
In [100]: set1.intersection(set2)

Out[100]: {'c', 'd'}
```

```
In [101]: set1.difference(set2)

Out[101]: {'a', 'b'}
```

```
In [102]: set5 = set1 - set2
```

```
In [103]: set1.symmetric_difference(set2)

Out[103]: {'a', 'b', 'e', 'f'}
```

```
In [104]: set5.issubset(set1)

Out[104]: True
```

```
In [105]: set5.issuperset(set1)

Out[105]: False
```

Conditional Statements

Input --> Rule --> Output

If its raining take your umbrella

else dont take it

if condition: return

else: return

```
In [106]: raining = False
          if raining:
              print('Take umbrella')
          else:
              print('Dont take umbrella')

          Dont take umbrella
```

```
In [107]: if 3>0:
              print('Positive Number')

          Positive Number
```

In [108]:

```
if -3>0:
    print('Positive Number')
print('this is always printed')
```

```
this is always printed
```

In [109]:

```
# in this program we are checking whether number is positive or negative
num = 3
if num > 0:
    print('Positive Number')
elif num == 0:
    print('Zero')
else:
    print('Negative Number')
```

```
Positive Number
```

Q. write a program to print even or odd

In [110]:

```
num = 2
if num%2 == 0:
    print('Even Number')
else:
    print('Odd Number')
```

```
Even Number
```

In [111]:

```
# Nested If
num = 0
if num >= 0:
    if num > 0:
        print('Positive Number')
    else:
        print('Zero')
else:
    print('Negative Number')
```

```
Zero
```

Defining and calling a function

def name_of_function(arguments): body of function

Example: max([1,2,3])

Here, name of function is max

list [1,2,3] is the argument that your function is expecting

In [112]:

```
def chk_nmbr(num):
    if num >= 0:
        if num > 0:
            print('Positive Number')
        else:
            print('Zero')
    else:
        print('Negative Number')
chk_nmbr(3)
```

```
Positive Number
```

Practice question. Write a Fizz Buzz Program with below conditions:

1. if the number is divisible by 3 --> Fizz
2. if the number is divisible by 5 --> Buzz
3. if the number is divisible by 3,5 --> FizzBuzz
4. if its not divisible by any of mentioned above numbers it needs to return the same number

In [113]:

```
def FizzBuzz(x):
    if x%3==0 and x%5==0:
        print("FizzBuzz")
    elif x%3==0:
        print("Fizz")
    elif x%5==0:
        print("Buzz")
    else:
        return x
```

In [114]:

```
num = int(input('Enter the number: '))
FizzBuzz(num)
```

```
Enter the number: 4
```

Out[114]:

```
4
```

Loops

In [115]:

```
snacks = ['pizza','Burger','Shawarma','Franky']
```

In [116]:

```
print('Current snack is ',snacks[0])
print('Current snack is ',snacks[1])
print('Current snack is ',snacks[2])
print('Current snack is ',snacks[3])
```

```
Current snack is  pizza
Current snack is  Burger
Current snack is  Shawarma
Current snack is  Franky
```

In [117]:

```
for snack in snacks:
    print('Current snack is',snack) # we do not have to use indices as Python
```

```
Current snack is pizza
Current snack is Burger
Current snack is Shawarma
Current snack is Franky
```

In [118]:

```
list(range(5)) #Creating a list using Range with 5 objects
```

Out[118]:

```
[0, 1, 2, 3, 4]
```

In [119]:

```
for i in range(4):
    print(i)
```

```
0
1
2
3
```

In [120]:

```
snacks
```

Out[120]:

```
['pizza', 'Burger', 'Shawarma', 'Franky']
```

In [121]:

```
len(snacks)
```

Out[121]:

```
4
```

In [122]:

```
snacks[0]
```

Out[122]:

```
'pizza'
```

In [123]:

```
for i in range(len(snacks)): #len(snacks) give 4, so range(4) gives 0,1,2,3 an
    print('current snack is',snacks[i])
```

```
current snack is pizza
current snack is Burger
current snack is Shawarma
current snack is Franky
```

In [124]:

```
str1 = 'Python'
for letter in str1:
    print(letter)
```

```
P
y
t
h
o
n
```

While loop

In [125]:

```
i = 0
while i<3:
    print('Inside while loop')
    i +=1 # i = i+1
```

```
Inside while loop
Inside while loop
Inside while loop
```

In [126]:

```
n = 10
sum = 0
i = 1

while i<=n:
    sum = sum+i
    i = i+1
else:
    print('Summation of first',n,'natural numbers is',sum)
```

```
Summation of first 10 natural numbers is 55
```

1. **break -** Whenever executed, the loop will be terminated and execution comes out of loop
2. **continue -** Skips the current iteration and and goes to next iteration in the same loop
3. **pass -** Used to create place holder for later use. It does nothing. Empty iteration will give error so pass helps in avoiding error

In [127]:

```
for i in [1,2,3,4,5,6,7,8,9,10]:
    print(i,end = ' ')
    if i == 7:
        break
```

```
1 2 3 4 5 6 7
```

In [128]:

```
for i in [1,2,3,4,5,6,7,8,9,10]:
    if i == 7:
        continue
    print(i,end = ' ')
```

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

In [129]:

```
for i in [1,2,3,4,5,6,7,8,9,10]:
    if i == 7:
        pass
    print(i,end = ' ')
```

```
1 2 3 4 5 6 7 8 9 10
```

List comprehensions

- They are used to reduce execution time and code size/space.
- They reduce readability

- Due to this reason, list comprehensions are used when we require fast execution and dropped where we require readability

```
In [130]: def for_loop():
              list1 = []

              for i in range(30):
                  if i%2 == 0:
                      list1.append(i)
              return list1

          for_loop()
```

```
Out[130]: [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28]
```

```
In [131]: def list_comp():
              return [i for i in range(30) if i%2 == 0]

          list_comp()
```

```
Out[131]: [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28]
```

timeit Function

- It is used to get execution time of code/snippets
- Syntax: timeit.timeit(stmt, setup,timer, number) # setup, timer and number Arguments are optional
- stmt: This will take the code for which you want to measure the execution time. The default value is “pass”.
- setup: This will have setup details that need to be executed before stmt. The default value is “pass.”
- timer: This will have the timer value, timeit() already has a default value set, and we can ignore it.
- number: The stmt will execute as per the number is given here. The default value is 1000000.

```
In [132]: import timeit
```

```
In [133]: timeit.timeit(for_loop,number = 1000)
```

```
Out[133]: 0.016016199995647185
```

```
In [134]: timeit.timeit(list_comp,number = 1000)
```

```
Out[134]: 0.019959099998231977
```