Python Programming

Python

- It was developed in December 1989 by Guido van Rossum
- It was named Python after their favourite show Monty **Python's** Flying Circus, a BBC Comedy series.
- Python does not support pointers, it has references
- Python is based on indentation.

Zen Of Python(Extra)

In [2]:
```
import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Data Types

There are 6 data Types in Python

- Null: when no value is present
- Numeric: Int, Float, Complex
- Sequence: List, Tuples, Strings
- Mapping: Dictionary
- Sets: it stores unique values
- Boolean: bool

Operations in Python

Arithmetic Operations

- Addition : ‘+’
- Subtraction : ‘-’
- Division : ‘/’
- Float Division : ‘//’
- Multiplication : ‘*’
- Square : ‘**’
- Increment : ‘++’
- Decrement : ‘--’
- In python instead of using i=i++ we use i+=1 and i=i– is replaced with i-=1

Logical Operations

- AND : ‘and’, ‘&’
- OR : ‘or’ , ‘|’
- NOT : ‘not’
- Binary one’s complement : ‘~’ (tilled sign) #Extra
- Binary Left Shift : ‘<<’
- Binary Right Shift : ‘>>’

Boolean Operation

- Bool = 1 : True
- Bool = 0 :False

```
In [7]: a = bool(1)
        print(a)
```

```
True
```

```
In [8]: a = bool(0)
        print(a)
```

```
False
```

Comparison Operation

- Greater than : ‘>’
- Less than : ‘<’
- Equal to : ‘==’
- Greater than equal to : ‘>=’
- Less than equal to : ‘<=’
- Not equal to : ‘!=’

Chaining Comparison Operator

- 1 < x < 10

```
In [95]: x=4
         1<x<10

Out[95]: True
```

Condition Operation (is)

The is operator works on reference of a variable and gives a Boolean output if condition is true or false.

```
In [92]: a="hello"
         b="hello"
         a is b

Out[92]: True
```

```
In [96]: a=[1,2,3,4,5]
         b=[1,2,3,4,5]
         a is b

Out[96]: False
```

```
In [98]: a=(1,2,3,4,5)
         b=(1,2,3,4,5)
         a is b

Out[98]: False
```

Is (continued)

In [107]:
```
a=(1,2,3,4,5)
b=(1,2,3,4,5)
c=a
print(id(a))
print(id(b))
print(id(c))
print(a is b)
print(a is c)
```

```
1657168777464
1657147106936
1657168777464
False
True
```

In [102]:
```
a="hello"
b="hello"
print(id(a))
print(id(a))
a is b
```

```
1657120923184
1657120923184
```

Out[102]: True

Is (continued)

```
In [109]: x=5.6
          type(x) is int

Out[109]: False

In [110]: type(x) is not int

Out[110]: True
```

Conditional Operator(If-Else)

```
if condition:
    logic statements
else:
    logic statements
```

In [111]:
```
a=10
b=20
if a == b:
    print("Equal")
else:
    print("Not Equal")
```

```
Not Equal
```

Membership Operation(in)

The ‘in’ operator is used to check if a value exists in a sequence or not. Evaluates to true if it finds a variable in the specified sequence and false otherwise.

```
In [112]: a=[1,2,3,4,5]
          print(4 in a)

          True
```

How to take an input

```
In [*]: #Input Value
        a=input('Enter Value')
```

```
Enter Value
```

```
In [3]: a=input('Enter Value')
```

```
Enter Valueapurva
```

```
In [*]: #Input Value
        print('Enter Value')
        a=input()
```

```

Enter Value
```

```
In [2]: print('Enter Value')
        a=input()
```

```
Enter Value
apurva
```

Using the End Statement

```
In [*]: print('Enter Value',end =' : ')
        a=input()
```

```
Enter Value :
```

```
In [1]: print('Enter Value',end =' : ')
        a=input()
```

```
Enter Value : apurva
```

Eval

```
In [5]: a=eval(input("Enter no"))
        print(type(a))
```

```
Enter no16
<class 'int'>
```

Loops(For Loop)

The for loop is used when a particular set of instructions are to be executed repeatedly.

In [114]:
```
fruits=["apple","cherry","orange"]
for i in fruits:
    print(i)
```

```
apple
cherry
orange
```

Enumerate in For Loop(Extra)

Python's built-in enumerate function allows us to loop over a list and retrieve both the index and the value of each item in the list:

```
In [30]: # Know the index faster
         vowels=['a','e','i','o','u']
         for i, letter in enumerate(vowels):
             print (i, letter)
```

```
0 a
1 e
2 i
3 o
4 u
```

Else for a For loop(Extra)

- We can use an else statement with a for loop.

In [116]:
```
def func(array):
     for num in array:
        if num%2==0:
            print(num)
            break # Case1: Break is called, so 'else' wouldn't be executed.
     else: # Case 2: 'else' executed since break is not called
        print("No call for Break. Else is executed")

print("Ans :")
a = [3,2,4,1]
func(a)
```

```
Ans :
2
```

Loops(While)

In [115]:
```
i = 1
while i < 6:
  print(i)
  i += 1
```

```
1
2
3
4
5
```

len() function

- It is used to find the length of a particular string, list, tuple.
- Syntax : len(name of string/list/tuple)

String

- It is defined in double(“”) or single(‘’) inverted commas.
- Mutable Datatype.

Sorting a String

- It is used to sort the components of a string according to the ascii values
- Syntax : sorted(string)

```
In [6]: string="Apurva2970"
        sorted(string)

Out[6]: ['0', '2', '7', '9', 'A', 'a', 'p', 'r', 'u', 'v']
```

Slicing a String

- It is done to get a particular part of a particular string
- Syntax : string[start index : end index : step size]

```
In [8]: n="apurva_suri"
        print(n[::-1])          # to reverse a string
        print(n[::3])           # prints the elements at a step size of 3
        print(n[2:])            # prints from the 2nd indexed element
        print(n[:2])            # prints everything before 2nd indexed element
        print(n[0:5:2])         # prints from 0th indexed to 5th indexed element in a step size of 2
        print(n[11::-1])        # used to reverse an array

        irus_avrupa
        ar_r
        urva_suri
        ap
        auv
        irus_avrupa
```

Indexing of String

- It is used to find the index of any given element in the string.
- Syntax : string.index(“element”)

```
In [13]: a="Alphaisalpha"
         print(a.index("a"))
```

```
4
```

Concatenation

- It is done to join two string together into a single string.
- Syntax : string1 + string2

```
In [14]: a = "Python"
         b = "Language"
         c=a+b
         print(c)
```

```
PythonLanguage
```

Repeating String

- Syntax : string * no of times to be repeated

```
In [15]: a = "Hello"
         b = a*2
         print(b)
```

```
HelloHello
```

Removing Spaces from String

- Syntax : string.strip() # for removing space from left and right side of string
  string.lstrip() # for removing space from left side of string
  string.rstrip() # for removing space from right side of string

```
In [17]: a="  Python  "
         print(a.lstrip(),'has length:',len(a.lstrip()))
         print(a.rstrip(),'has length:',len(a.rstrip()))
         print(a.strip(),'has length:',len(a.strip()))

Python   has length: 8
  Python has length: 8
Python has length: 6
```

Replacing one String with another

- Syntax : string.replace(replaced word , word that is replacing it)

```
In [19]: a="I like Python. Python is a good language"
         b="Python"  #word that is going to replace
         c='C++'       #word to be replaced
         print(a.replace(b,c))

         I like C++. C++ is a good language
```

Changing Case of String

- We can change the case of a String by using inbuilt functions such as :
- upper()
- lower()
- title()
- swapcase()

```
In [21]: b="python"
         print("uppercase = " +b.upper())
         print("lowercase = " +b.lower())
         print("swapcase = " +b.swapcase())
         print("Title = " +b.title())
```

```
uppercase = PYTHON
lowercase = python
swapcase = PYTHON
Title = Python
```

Checking Condition in String

In [26]:
```
w=" "
y="python478061"
x="Python"
z="628931"
print(y.islower())
print(y.isalpha())
print(x.isalpha())
print(y.isdigit())
print(z.isdigit())
print(y.isalnum())
print(y.islower())
print(y.istitle())
print(x.istitle())
print(y.isspace())
print(w.isspace())
```

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

Format() Function

- **str.format()** is one of the *string formatting methods* in Python3, which allows multiple substitutions and value formatting. This method lets us concatenate elements within a string through positional formatting.
- Syntax: ***{ } .format(value)***
- *s – strings*
- *d – decimal integers (base-10)*
- *f – floating point display*
- *c – character*
- *b – binary*
- *o – octal*
- *x – hexadecimal with lowercase letters after 9*
- *X – hexadecimal with uppercase letters after 9*
- *e – exponent notation*

Format() Continued

In [120]:
```
print ("This site is {0:f}% securely {1}!!". format(100, "encrypted"))
print ("My average of this {0} was {1:.2f}%".format("semester", 78.234876))
print ("My average of this {0} was {1:.0f}%" .format("semester", 78.234876))
print("The {0} of 100 is {1:b}".format("binary", 100))
print("The {0} of 100 is {1:o}".format("octal", 100))
```

```
This site is 100.000000% securely encrypted!!
My average of this semester was 78.23%
My average of this semester was 78%
The binary of 100 is 1100100
The octal of 100 is 144
```

In [122]:
```
name=input("Enter name")
i_d=int(input("Enter id no"))
sal=int(input("Enter salary"))
string="My name is {}. My id is {} and i have a salary of ${}".format(name,i_d,sal)
print(string)
```

```
Enter nameRahul
Enter id no12046
Enter salary25000
My name is Rahul. My id is 12046 and i have a salary of $25000
```

Format() Continued

In [152]:
```
print("{0:15}, is the computer science portal for {1:8}!".format("GeeksforGeeks", "geeks"))
print("It is {0:5} degrees outside !".format(40))
print("{0:19} was founded in {1:16}!".format("GeeksforGeeks", 2009))

# To demonstrate aligning of spaces
print("{0:^26} was founded in {1:<4}!".format("GeeksforGeeks", 2009))
print("{:*^20s}".format("Geeks"))
```

```
GeeksforGeeks  , is the computer science portal for geeks   !
It is    40 degrees outside !
GeeksforGeeks       was founded in             2009!
      GeeksforGeeks        was founded in 2009!
*******Geeks********
```

Format() Continued

In [42]:
```
a = int(input("Enter lower range :-\n"))
b = int(input("Enter upper range :-\n"))
print("Not using Formatter")
for i in range (a, b):
        print ( i, i**2, i**3, i**4 )
print("Using Formatter")
for i in range (a, b):
        print("{:6d} {:6d} {:6d} {:6d}" .format(i, i ** 2, i ** 3, i ** 4))
# Using formatters to give 6 spaces to each set of values
```

```
Enter lower range :-
2
Enter upper range :-
5
Not using Formatter
2 4 8 16
3 9 27 81
4 16 64 256
Using Formatter
     2      4      8     16
     3      9     27     81
     4     16     64    256
```

Counting occurrence of a letter in String

- It is used to count how many time a particular substring has occurred in a string or even if it is present in a particular section of the string or not.
- Syntax : string.count(substring,start index,end index)

  or string.count(substring)

```
In [59]: a='mississippi'
         b='i'
         print(a.count(b,0,6))
         print(a.count(b))
```

```
2
4
```

Defining Infinity in Python(Extra)

- We can define infinity in Python.

In [79]:
```
# Positive Infinity
p_infinity = float('Inf')

if 999999999999 > p_infinity:
    print("The number is greater than Infinity!")
else:
    print("Infinity is greatest")

#Negetive Infinity
n_infinity = float('-Inf')
if -9999999999999 < n_infinity:
    print("The number is lesser than Negative Infinity!")
else:
    print("Negative Infinity is least")
```

```
Infinity is greatest
Negative Infinity is least
```

Lists

- It is defined in [ ] square braces and each element is separated by a comma(,).
- Mutable Datatype
- It can contain different data types in a single list.
- We can have an int, string, float, list, tuple, dictionary inside the same list.

Inputting a List

In [66]:
```
lst=[1,2,3,4,5]
lst1=eval(input("Enter elements of list seperated by comma"))
print(lst,lst1)
print(type(lst1))
```

```
Enter elements of list seperated by comma[6,7,8,9,10]
[1, 2, 3, 4, 5] [6, 7, 8, 9, 10]
<class 'list'>
```

Append()

- It adds an element to the end of the list
- Syntax : `list.append(item)`

```
In [67]: lst=[]
         for i in range(0,5):
             lst.append(int(input("Enter no")))
         print(lst)
```

```
Enter no1
Enter no2
Enter no3
Enter no4
Enter no5
[1, 2, 3, 4, 5]
```

```
In [70]: lst=[1,2,3,4,5]
         lst.append(6)
         lst.append([7,8])
         print(lst)
```

```
[1, 2, 3, 4, 5, 6, [7, 8]]
```

Split()

- Syntax : `str.split(separator, maxsplit)`
- separator *: The is a delimiter. The string splits at this specified separator. If is not provided then any white space is a separator.*
- maxsplit *: It is a number, which tells us to split the string into maximum of provided number of times. If it is not provided then there is no limit.*
- *Return* : returns a list of strings after breaking the given string by the specified separator.

Split() Continued

```
In [62]: name1=input("Enter names")
         emailid=input("Enrt id")
         lst=name1.split(",")
         print(name1.split(",",2))
         print(lst)
         lst1=emailid.split("@")
         print(lst1)
```

```
Enter namesApurva,Ashwin,Drishya,Prateek
Enrt id198@287@674@878
['Apurva', 'Ashwin', 'Drishya,Prateek']
['Apurva', 'Ashwin', 'Drishya', 'Prateek']
['198', '287', '674', '878']
```

Join()

- Syntax : `string.join(iterable)`
- It returns a string.

```
In [11]: lst=["Python","Java","C++"]
         new="".join(lst)
         s="-".join(lst)
         print(new)
         print(s)

PythonJavaC++
Python-Java-C++
```

Sorting a List

- **Syntax :** sorted(iterable, key, reverse)
- **Parameters :** sorted takes three parameters from which two are optional.
- *Iterable* : sequence (list, tuple, string) or collection (dictionary, set, frozenset) or any other iterator that needs to be sorted.
- *Key*(optional) : A function that would server as a key or a basis of sort comparison.
- *Reverse*(optional) : If set true, then the iterable would be sorted in reverse (descending) order, by default it is set as false.

Sorting Continued

In [21]:
```
def secondsort(x):
    return(x[1])
def thirdsort(x):
    return(x[2])
lst=[2,3,0,8,4,7,-1]
print(sorted(lst))
print(sorted(lst,reverse=True))
lst1=[[1,2,6],[3,0,1],[4,6,-1],[2,2,2]]
lst1.sort(key=secondsort)
print(lst1)
lst1.sort(key=thirdsort)
print(lst1)
```

```
[-1, 0, 2, 3, 4, 7, 8]
[8, 7, 4, 3, 2, 0, -1]
[[3, 0, 1], [1, 2, 6], [2, 2, 2], [4, 6, -1]]
[[4, 6, -1], [3, 0, 1], [2, 2, 2], [1, 2, 6]]
```

Indexing in List

- It is used to find the index of a particular element in the list.
- Syntax : list.index(element)

```
In [34]: lst=[19,23,45,21,57,24,9,56]
         lst.index(9)

Out[34]: 6
```

Repeating a List

In [89]:
```
#Repeating List
#synatx = list*integerno
x=list(range(3,30,7))
print("List is",x)
print("Repetation twice",x*2)
print("Repetation 5 times",x*5) #one list with repeated elements
```

```
List is [3, 10, 17, 24]
Repetation twice [3, 10, 17, 24, 3, 10, 17, 24]
Repetation 5 times [3, 10, 17, 24, 3, 10, 17, 24, 3, 10, 17, 24, 3, 10, 17, 24, 3, 10, 17, 24]
```

Slicing in List

- Slicing in List is the same as in String.

```
In [35]: lst=[1,2,3,4,5,6,7,8,9]
         print(lst[1:5])
         print(lst[::-1])
         print(lst[2::2])
```

```
[2, 3, 4, 5]
[9, 8, 7, 6, 5, 4, 3, 2, 1]
[3, 5, 7, 9]
```

Reversing a List

In [33]:
```
lst=[4,2,7,8,1,10,3]
lst.reverse()
print(lst)
```

```
[3, 10, 1, 8, 7, 2, 4]
```

In [101]:
```
#Reversig of list
days=['Sunday','monday','Tuesday','wednesday',\
      'thursday','friday','saturday'] #to continue from next line
print(days[len(days):0:-1])
```

```
['saturday', 'friday', 'thursday', 'wednesday', 'Tuesday', 'monday']
```

Delete()

- Syntax : del list(index of element)
- It deletes the element present at that index

Remove()

- Syntax : list.remove(element)
- It removes a particular element if present in list

Pop()

- Syntax : list.pop()
- It deletes the element from the end of the list

```
In [39]: lst=[1,2,3,4,5,6,1,7,8,9,10,2,11,12,3]
         lst.remove(2)
         print(lst)
         del lst[5]
         print(lst)
         lst.pop()
         print(lst)
```

```
[1, 3, 4, 5, 6, 1, 7, 8, 9, 10, 2, 11, 12, 3]
[1, 3, 4, 5, 6, 7, 8, 9, 10, 2, 11, 12, 3]
[1, 3, 4, 5, 6, 7, 8, 9, 10, 2, 11, 12]
```

Copy()

- We have to copy a list to another as if we use the = sign it creates a reference and any changes made in one list will show in the other.
- Syntax : list1=list.copy()

```
In [40]: a=[1,2,3,4,5]
         b=a
         c=a.copy()
         b.append(6)
         a.remove(2)
         print(a)
         print(b)
         print(c)
```

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

Insertion in List

- Syntax : list.insert(element,index)
- It is used to insert an element at a particular index of a list.

```
In [41]: a=[1,2,3,4,5,6,7]
         a.insert(3,18)
         a.insert(5,"Python")
         print(a)

         [1, 2, 3, 18, 4, 'Python', 5, 6, 7]
```

Tuples

Difference Between Tuple and List

Tuple

- Faster
- More secure
- Immutable

List

- Slower
- Less Secure
- Mutable

Inputting a tuple

In [2]:
```
# One way of creation
tup = 'python', 'geeks'
print(tup)
print(type(tup))
# Another for doing the same
tup = ('python', 'geeks')
print(tup)
tup1=eval(input("enter tuple in () brackets"))
print(tup1)
```

```
('python', 'geeks')
<class 'tuple'>
('python', 'geeks')
enter tuple in () brackets(1,2,3,4,"Asia",5.6)
(1, 2, 3, 4, 'Asia', 5.6)
```

Basic functions of tuple

```
In [9]: tup=(1,3,0,4,-1,6)
        tup1=(1,3,4,-1,6)
        print(min(tup))
        print(max(tup))
        print(sum(tup))
        print(any(tup))
        print(all(tup))
        print(any(tup1))
        print(all(tup1))
```

```
-1
6
13
True
False
True
True
```

Concatenating a tuple

- Like in String and List the + sign is used to concatenate tuples.
- tup2= tup+tup1

```
In [1]: tup = (1,2,3,4,5)
        tup1 = ("apple","cherry","rose")
        tup2 = tup + tup1
        print(tup2)

        (1, 2, 3, 4, 5, 'apple', 'cherry', 'rose')
```

Nesting a tuple

```
In [2]: tup = (1,2,3,4,5)
        tup1 = ("apple","cherry","rose")
        tup2 = tup,tup1
        print(tup2)
```

```
((1, 2, 3, 4, 5), ('apple', 'cherry', 'rose'))
```

Repetition in tuple

```
In [3]: tup = ("apple","cherry","rose")
        tup1 = tup*3
        print(tup1)
```

```
('apple', 'cherry', 'rose', 'apple', 'cherry', 'rose', 'apple', 'cherry', 'rose')
```

Slicing in tuple

```
In [11]: tup=(1,2,3,4,5,6,7,8,9,10)
         print(tup[2:7])
         print(tup[::-1])
         print(tup[:8:3])
```

```
(3, 4, 5, 6, 7)
(10, 9, 8, 7, 6, 5, 4, 3, 2, 1)
(1, 4, 7)
```

Converting list to tuple

- A list can be converted to a tuple simply by writing the key word tuple.
- Syntax : tuple(list)

```
In [26]: lst=[1,2,3,"apple","cherry",5,6,7]
         print(lst)
         print(type(lst))
         tup=tuple(lst)
         print(tup)
         print(type(tup))
```

```
[1, 2, 3, 'apple', 'cherry', 5, 6, 7]
<class 'list'>
(1, 2, 3, 'apple', 'cherry', 5, 6, 7)
<class 'tuple'>
```

string to tuple

```
In [28]: string="Geeks"
         tup=tuple(string)
         print(tup)
```

```
('G', 'e', 'e', 'k', 's')
```

Insertion in tuple

- As we know tuples are immutable so once a tuple is made no changes can be done, i.e., no insertion, deletion or replacement.
- So, here we do slicing.

```
In [18]: #Insertion
         tup = (1,2,3,4,5,6,7,8,9)
         n = int(input("Enter the position where you want to insert : "))
         lst = [input("Enter element to be inserted : ")]
         e = tuple(lst)
         tup_new = tup[:n-1] + e + tup[n-1:]
         print(tup_new)

         Enter the position where you want to insert : 7
         Enter element to be inserted : python
         (1, 2, 3, 4, 5, 6, 'python', 7, 8, 9)
```

Deletion in tuple

In [19]:
```
#Deletion
tup = (1,2,3,4,5,6,7,5.6,8,9)
n = eval(input("Enter the element you want to delete : "))
for i in range(len(tup)):
    if tup[i] == n:
        tup_new = tup[:i] + tup[i+1:]
print(tup_new)
```

```
Enter the element you want to delete : 5.6
(1, 2, 3, 4, 5, 6, 7, 8, 9)
```

Replacement in tuple

In [19]:
```
a="I like Python. Python is a good language"
b="Python"  #word that is going to replace
c='C++'       #word to be replaced
print(a.replace(b,c))
```

```
I like C++. C++ is a good language
```

Dictionary

- It is a mutable data type
- It is declared using curly “{}” braces and has a key value pair.
- Syntax : {key : value , key : value}
- Every key has a value
- All keys are unique.

Inputting a dictionary

In [29]:
```
d = { 1 : "Python" , 2 : "C++" , 3 : "Java"}
d1 = eval(input("Enter Dictionary in {}"))
print(d)
print(d1)
```

```
Enter Dictionary in {}{"Name" : "Apurva" , "Age" : 20 , "Id" : 12345}
{1: 'Python', 2: 'C++', 3: 'Java'}
{'Name': 'Apurva', 'Age': 20, 'Id': 12345}
```

Accessing a value through its key and adding a key: value pair

- Syntax to access: dict_name[Key]
- Syntax to add: dict_name[new_key]=new_value

In [34]:
```
d1 = {"Name" : "Apurva" , "Age" : 20 , "Id" : 12345}
print(d1["Name"])
d1["Year"]=2021
print(d1)
```

```
Apurva
{'Name': 'Apurva', 'Age': 20, 'Id': 12345, 'Year': 2021}
```

To change the value of a key

- Syntax : dict_name(key) = new_value

```
In [35]: d1 = {'Name': 'Apurva', 'Age': 20, 'Id': 12345, 'Year': 2021}
         d1["Year"]=2019
         print(d1)

         {'Name': 'Apurva', 'Age': 20, 'Id': 12345, 'Year': 2019}
```

Delete a particular key : value pair or entire dictionary

In [37]:
```
d1 = {'Name': 'Apurva', 'Age': 20, 'Id': 12345, 'Year': 2021}
del d1["Year"]
print(d1)
d1.clear()
print(d1)
```

```
{'Name': 'Apurva', 'Age': 20, 'Id': 12345}
{}
```

In [38]:
```
d1 = {'Name': 'Apurva', 'Age': 20, 'Id': 12345, 'Year': 2021}
print(d1)
del d1
print(d1)
```

```
{'Name': 'Apurva', 'Age': 20, 'Id': 12345, 'Year': 2021}
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-38-68001c6189e6> in <module>()
      2 print(d1)
      3 del d1
----> 4 print(d1)

NameError: name 'd1' is not defined
```

List of keys

- Syntax : dict_name.keys()

```
In [52]: d1 = {'Name': 'Apurva', 'Age': 20, 'Id': 12345, 'Year': 2021}
         print(d1.keys())

         dict_keys(['Name', 'Age', 'Id', 'Year'])
```

Unpacking(Extra)

- It is done to take out the elements of a list or a dictionary and send them to a function or print them.

In [22]:
```
def point(x, y, z):
    print(x,y,z)

foo_list = (2, 3, 4)
bar_dict = {'y': 3, 'x': 2, 'z':'4'}

point(*foo_list) # Unpacking Lists
point(**bar_dict) # Unpacking Dictionaries
```

```
2 3 4
2 3 4
```

Sets

- They hold unique values, i.e., no repetition of elements.
- They are mutable data types.

Inputting set

In [56]:
```
a = {12,0,90,34,67,58,45,12,67,38,18,27}
s = eval(input("Enter set in {}"))
print(a)
print(s)
```

```
Enter set in {}{"Python","java","C++","R","Pearl","Python","R"}
{0, 34, 67, 58, 38, 12, 45, 18, 90, 27}
{'java', 'R', 'C++', 'Python', 'Pearl'}
```

Converting to set

- We can convert a string, list, tuple into a set simply by writing the keyword set before it.
- Syntax : set(string/list/tuple)

```
In [57]: a = [1,2,89,5,4,3,12,1,3,1,31,2,5,6,45]
         b = "Mississippi"
         c = (1,2,3,4,5,1,3,9,6,7,9,1,3,0,2)
         s=set(a)
         print(s)
         s1=set(b)
         print(s1)
         s2=set(c)
         print(s2)
```

```
{1, 2, 3, 4, 5, 6, 12, 45, 89, 31}
{'M', 'i', 'p', 's'}
{0, 1, 2, 3, 4, 5, 6, 7, 9}
```

Adding element to a set

- Syntax : set.add(element)

```
In [60]: s = {'java', 'R', 'C++', 'Python', 'Pearl'}
         s.add("C")
         print(s)
```

```
{'java', 'C', 'R', 'C++', 'Python', 'Pearl'}
```

Union

- Syntax : set1.union(set2) or set1|set2

```
In [58]: set1 = {"Rahul","Rohit","Priya","Riya"}
         set2 = {"Mohan","Ram","Riya"}
         set3 = set1.union(set2)
         set4 = set1|set2
         print(set3)
         print(set4)
```

```
{'Riya', 'Mohan', 'Rahul', 'Rohit', 'Priya', 'Ram'}
{'Riya', 'Mohan', 'Rahul', 'Rohit', 'Priya', 'Ram'}
```

Intersection

- Syntax : set1.intersection(set2)

```
In [59]: set1 = {"Rahul","Rohit","Priya","Riya"}
         set2 = {"Mohan","Ram","Riya"}
         set3 = set1.intersection(set2)
         print(set3)
```

```
{'Riya'}
```

Difference

- Syntax : set1.difference(set2)

  or set1- set2

```
In [64]: set1 = {"Rahul","Rohit","Priya","Riya"}
         set2 = {"Mohan","Ram","Riya"}
         set3 = set1.difference(set2)
         set4 = set2.difference(set1)
         set5 = set1 -set2
         set6 = set2 -set1
         print(set3)
         print(set4)
         print(set5)
         print(set6)
```

```
{'Rohit', 'Priya', 'Rahul'}
{'Mohan', 'Ram'}
{'Rohit', 'Priya', 'Rahul'}
{'Mohan', 'Ram'}
```

Functions

- Functions are used to avoid repetition of lines.
- We don’t write the same code again and again we just call the function and it does the work.
- **Syntax:**

```
def function_name(parameters):
    logic statements
```

Return Statement

- You can return multiple values in Python

File Handling

- file can store any type of data
- Collection of files is called a record
- if we open a file in write mode even if file doesn’t exist it will run
- backlog of write is that it will overwrite if data is present so we have to use append

Modes in File

- r : Read
- w : Write
- a : Append
- r+ : first u read then write
- w+ : first write then we will read
- a+ : first u write then we will read from start

Open and Close a file

- To use a particular file we have to open it first and then close it.
- Closing a file is very important because if we don't close it, it may damage the information we store in that file.
- Closing the file permanently saves the data.

```
In [1]: f=open('myfile.txt','w')
        str = input("Enter Text : ")
        f.write(str)
        f.close()

        Enter Text : I am learning Python
```

```
In [2]: f=open('myfile.txt','r')
        str=f.read()
        print(str)
        f.close()

        I am learning Python
```

File

In [3]:
```
f=open("myfile.txt",'a+')
print("Enter text(@ at end)")
while str !='@':
    str=input()
    if str !='@':
        f.write(str +'\n')
f.close()
```

```
Enter text(@ at end)
Python is a great language.
It is easy to learn.
@
```

In [6]:
```
with open("file.txt",'w') as f:
    f.write("I am a leraner\n")
    f.write('python is attractive')
print("Reading using with")
with open("file.txt",'r') as f:
    for line in f:
        print(line)
```

```
Reading using with
I am a leraner

python is attractive
```

Exception Handling

- It runs the program even if there is an error and gives in log to make program robust we use exception handling ,if the user gets an error in the program and he can try to remove it then it is called an exception and if user gets an error and he is not able to solve it is called an error.
- **BLOCKS IN EXCEPTION HANDLING**
- try block : all the logical statements or logic have to be written in this block only
- if programmer thinks there might be error he can write those lines of code also
- except block : all the exceptions of the categorized errors to be written in this block
- we can use more than 2 exception classes in one block
- final block : statements that need to be performed despite error have to be written in this block
- example:file closing

ERRORS

- Syntax error : These are all those errors that occur due to wrong syntax or indentation
- it is executed during compile time.
- runtime error : These errors are identified during runtime by python interpreter (inline or online interpreter)
- example: zero division error.
- Out of Range/Bound :
- Logical Error: These error depends on the programmer logic only and only programmer is the sole person responsible to remove it
- if there is a multiple line comment in loop indentation is to be seen

Facts

- A try block can have multiple except blocks
- it can have a except and finally block but not mandatory
- it can work without except and finally
- but except and finally cant work without try
- there is only one finally block