1) Easy to learn
2) Platform Independent - (Same Output - always come)
3) Dynamic memory management - (jitna need utna apply hoga)
4) Open Source - (licecanee free)
5) Variety of pre-Defined functions - (Only need to value pass)
6) Huge number of application
7) Integrated with any software
8) Light weight (no need to learn more syntax)

for Python - Anaconda

In Anaconda
- Jupyter Notebook
- Spyder
- pycharm
- visual studio code

i) Introduction
ii) Features
iii) Print ( )
iv) Type ( )
v) Addition Operations

Basic Data Types

1) Integer : -1111, 12345, 13482, -1234
2) String : 'RAHUL', 'RAJ', '1235', 'Raj123@'
3) Float : 12.22, -123.3, -123.0
4) Boolean : True, False 1,0
5) Complex : a+jb , 2+3.j

a & b → real number
a → real part
jb → imaginary part
j → sqrt -1

''' Sequence Data Types

1- LIST
2- DICTIONARY
3- TUPLE
4- SET

'''

Ex ~~list~~ List → [123, 12.2, "abc", True, 3

String → '', ""

* m = ['a', 'b']

len(m)

out[1] : 2

Data Types-

* A data types is a classification of data based on the type of value it contains.
It tell the computer what kind of data is stored in a particular variable or constant & how it should be treated or processed.

or

* It is used in connection with static typing of variables in programming languages.

* Sequence data types is combination of basic data types.

Ex- of List-

List → [123, 12.2, 'abc', True, 3+3.j] - It is combination of 5 elements.

List 1

out: [123, 12.2, 'abc', True, (3+3j)]

* List 1 = [123, 12.2, 'abc', True, 3+3.j]
len (list1)

out: 5

* List1 = [[123, 12.2, 'abc'], True, 3+3.j] (inner list)
len (list1)

out: 3

List inside list is called ~~nested~~ nested list

- Hyphen
_ Underscore

Index- Every elements that is present in a list is index value.

* list1 = [[123, 12.2, 'abc'], True, 3+3j]
  * list1[0] → out: [123, 12.2, 'abc']
  * list1[1] → out: True
  * list1[2] → out: 3+3j

* list1 = [[123, 12.2, 'abc'], True, 3+3j]

list1[0][0] → out: 123

list1[0][1] → out: 12.2

Slicing:-

A slice object is used to specify how to slice a sequence.

take a sublist out of a original list.

* list1 = [[123, 12.2, 'abc'], True, 3+3j]

list1[0][::-1]

out: ['abc', 12.2, 123] reverse

slicing: 3 parameter [start index: end index: jump]

list[0:2:1]

out: [[123, 12.2, 'abc'], True]

[0:2:1] ← exclusive

list[0][2::-1]

out: ['abc', 12.2, 123]

* list1 = ['567', True, 3+3j]

list1[0][2:0:-1]

out: 76

list1[0][2::-1]

out: 765

* x = [122, 980, 8900, 8787, 7676]
  0 1 2 3 4

x [4::-1]

out [7676, 8787, 8900, 980, 122] reverse

* z = [['abc', 'def'], 98, 87]

z [2]

out: 87

str(z[2])[::-1]

output: '78'

* y = '87'

y [1::-1]

out '78'

* z = 'hello'

z [4::-1]

out: 'olleh'

* Z = [56, 90]

Z [-1::-1]

out [90, 56]

List

Example: List indexing

my_list = ['p', 'r', 'o', 'b', 'e']

Print(my_list[0] ▷ p

Print(my_list[2] ▷ o

Print(my_list[4] ▷ e

Nested List

n_list = ["Happy", [2,0,1,5]]

Nested indexing

Print(n_list[0][1]) : out a

Print(n_list[1][3]) : out 5

- n underscore 0 of 1

Example: Negative indexing in lists

Python allows negative indexing for its sequences.

The index of -1 refers to the last item, -2 to the second last item and

```
my_list = ['p', 'r', 'o', 'b', 'e']

Print (my_List [-1])      # e
Print (my_List [-5])      # P
```

```
* my_list = ['p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'n', 'g']
# index    [ 0,   1,   2,   3,   4,   5,   6,   7,   8,   9]
# index    [-10  -9   -8   -7   -6   -5   -4   -3   -2   -1]

Print (my_list [2:5])      out ['o', 'g', 'r']
Print (my_list [:-5])      out ['p', 'r', 'o', 'g', 'r']
Print (my_list [5:])       out ['a', 'm', 'i', 'n', 'g']

z = [0:9]      out ['p', 'g']
z = [-10::9]   out ['p', 'g']
z = [5:0:-2]   out ['a', 'g', 'r']
z = [-5:-10:-2]  out ['a', 'g', 'r']
z = [5:8:2]    out ['a', 'i']
z = [5:8:-2]   out [] (Empty)
```

Jupitor Notebook - IDE

Functions

1) all(list) - The method all() method returns True when all elements in the given iterable are true. If not, it returns False.

Example: How all() works for lists?

```
z = [1,2,3,4,5]
Print(all(z))          # True : all values true

z = [0, False]
Print(all(z))          # False : all values false

z = [1,3,4,0]
Print(all(z))          # False : one false value

z = [0, False, 5]
Print(all(z))          # False : one true value

z = [ ]
Print(all(z))          # True : empty iterable
```

2) any(list) - any() function returns True if any element of an iterable is True. If not, any() returns False.

Ex :- True since, 1, 3 & 4 (at least one) is true

```
x = [1, 3, 4, 0]
Print(any(x))          # True

x = [0, False]
Print(any(x))          # False : both are False

x = [0, False, 5]      Print(any(x))   # True : 5 is true

x = [ ]                Print(any())    # False : iterable is empty
```

3) sorted(dict) - The sorted() function sorts the elements of a given iterable in a specific order (either ascending or descending) & returns the sorted iterable as a list.

Example: vowels list

```
my_list = ['e', 'a', 'u', 'o', 'i']
print(sorted(my_list))          # ['a', 'e', 'i', 'o', 'u']
Print(sorted(my_list, reverse=True))   # ['u', 'o', 'i', 'e', 'a']
```

4) sort() - The sort() method sorts the elements of a given list in a specific ascending or descending order.

Example

```
List 1 = [22, 30, 100, 300, 399]
list 2 = ['z', 'ab', 'abc', 'a', 'b']
List 1. sort()
list 2 . sort()
Print(list 1)   # [22, 30, 100, 300, 399]
List 1. sort (reverse = True)
Print (list 1)   # [399, 300, 100, 30, 22]
Print (list 2)   # ['a', 'ab', 'abc', 'b', 'z']
```

5) min(list) - this method is used to get min value from the list. In Python 3 list elements type should be same otherwise compiler throw type Error.

Ex

```
List 1 = ['a', 'b', 'c']
List 2 = [1, 2, 3]
~~list 3 = ['a', 'b', 'c', 1, 2, 3]~~
List 3 = ['a', 'b', 'c', 1, 2, 3]
```

```
Print (min (list1))      # a
Print (min (list2))      # 1
Print (min (list3))      # Type Error : '<' not supported
                           b/w instances of 'int' & 'str'
```

6) max (list) - The max() method returns the elements from the list with maximum value.

Ex :-

```
list1 = ['a', 'b', 'c']
list2 = [1, 2, 3]
list3 = ['a', 'b', 'c', 1, 2, 3]

Print (max (list1))      # c
Print (max (list2))      # 3
Print (max (list3))      # TypeError
```

7) len (list) - The len() method returns the number of elements in the list.

Ex :-

```
list 1 = ['a', 'b', 'c']
list 2 = []
list 3 = ['a', 'b', 'c', 1, 2, 3]

Print (len (list1))      # 3
Print (len (list2))      # 0
Print (len (list3))      # 6
```

8) append() :- ~~The~~ It is method ~~data~~ 'a' adds an item to the end of the list

Ex :-

```
list 1 = [1, 2, 3]
list 1 . append (4)
list 1 . append ('helloworld')
list1 . append (['a', 'b', 'c'])
Print (list 1)      # [1, 2, 3, 'helloworld', ['a', 'b', 'c']]
```

9) extend() - The extend() method adds all the elements of an iterable (list, tuple, string etc.) to the end of the list.

Ex :-

```
list1 = [1, 2, 3]
list1.extend([5])
list1.extend('helloworld')
list1.extend(['a', 'b', 'c'])
Print (list1)   # [1, 2, 3, 5, 'h', 'e', 'l', 'l', 'o', 'w', 'o', 'r', 'l', 'd', 'a', 'b', 'c']
```

10) Insert() - The insert() method inserts an elements to the list at the specified index.

Ex :-

```
list1 = ['helloworld', 'python', 'java', 'c++', 1, 2, 3]
list1.insert(3, 'c#')
Print (list1)   # ['helloworld', 'python', 'java', 'c#', 'c++', 1, 2, 3]
```

11) remove() - The remove() method removes the first matching element (which is passed as an argument) from the list.

Ex :-

```
list1 = ['helloworld', 'python', 'java', 'c++', 1, 2, 3]
list1.remove("java")
Print (list1)   # ['helloworld', 'python', 'c++', 1, 2, 3]

list1.remove(2)
print (list1)   # ['helloworld', 'python', 'c++', 1, 3]
```

12) POP() - The method removes the items at the given index from the list & returns the removed item.

Ex-

```
list1 = ['helloworld', 'python', 'java', 'c++', 1,2,3]
list.pop()          # pop last element
Print(list1)        # ['helloworld', 'python', 'java', 'c#', 1,2]
list.pop(2)         # pop element with index 2
Print(list1)        # ['helloworld', 'python', 'c++', 1,2]
```

13) Clear() - It is the method removes all items from the list.

```
list1 = ['helloworld', 'python', 'java', 'c++', 1,2,3]
list1.clear()
Print('list1:', list1)    # list1: []
```

14) Index() - this method returns the index of the specified element in the list.

```
list1 = ['helloworld', 'python', 'java', 'c++', 1,2,3]
Print("index of java:", list1.index('java'))    # index of java : 2
Print("index of 2 :", list1.index(2))           # index of 2 : 5
```

15) count() :- The count() method returns the number of time the specified element appears in the list.

```
list = [1,2,3, 'a', 'b', 'c', 1,2 3]
Print(list1, count(1))      # 2
Print(list1, count('b'))    # 1
print(list1, count(4))      # 0
```

16) reverse() : this method reverses the elements of the list

```
list1 = ['helloworld', 'Python', 'java', 'c++', 1,2,3]
list1.reverse()
Print(list1)   # [3,2,1, 'c++', 'java', 'Python', 'helloworld']
```

17) copy() : this method returns a shallow copy of the list

```
list1 = ['helloworld', 'python', 'java', 'c++', 1,2,3]
list2 = list1.copy()
Print('list1:', list1)   # list1: ['helloworld', 'Python', 'java', 'c++', 1, 2, 3]
Print('list2:', list2)   # list2: ['helloworld', 'Python', 'java', 'c++', 1, 2, 3]
```

—x—

1) List Membership Test

we can test if an item exists in a list or not, using the keyword in.

```
my_list = ['p', 'r', 'o', 'b', 'l', 'e', 'm']
# Output: True
Print('p' in my_list)   # True
# Output: False
Print('a' in my_list)   # False
# Output: True
Print('i' not in my_list)   # True
```

Tuple()

z = (1, 2, 3, 4, 5, 6)
type(z)
len(z) : out 6

z = (1, 2,)
type(z)
out : tuple

z = (90)
type(z)
out: int

z = (90, 10)
type(z)
out: tuple

Different types of tuples :-

```
# Empty tuple
my_tuple1 = ( )
Print (my_tuple1)          # out : ( )

# Tuple having integers
my_tuple2 = (1, 2, 3)
Print (my_tuple2[0])       # out : 1

# tuple with mixed datatypes
my_tuple3 = (1, "Hello", 3.4)
Print (my_tuple3[1][3]     # out : l

# nested tuple
my_tuple4 = ("mouse", [8, 4, 6], (1, 2, 3))
my_tuple4 [1][2]           # out : 6
#Print (my_tuple4)         # ("mouse", [8, 4, 6], (1, 2, 3))
 Len (my_tuple4)           # 3
```

Example!-

```
my_tuple = 3, 4.6, "dog", 5-6      # :: Paranthesis() is not mandatory
Print (my_tuple)                   # (3, 4.6, "dog", 56)
```

tuple unpacking is also possible

```
a, b, c, d = my_tuple
Print (a)      # 3
Print (b)      # 4.6
Print (c)      # dog
```

Example!

```
t1 = ('ab', 1, 2, 3.14, "Helloworld")
t2 = "a", "b", "c", "d"
```

tuple contain other list & tuple

```
t3 = (1, 2, 3, ['a', 'b', 'c'], ('z', 26))
Print (t1[4][-1])      # out : d
Print (t2[3])          # out : d
Print (t3[3][1])       # out : b
# len (t3[3])          # out : 26
```

(Doubt)

* # Accessing tuple elements using indexing

```
my_tuple = ('p', 'e', 'r', 'm', 'i', 't')
Print (my_tuple[0])                          # out! 'p'
Print (my_tuple[5])                          # out! 't'
Print ("Last index :", len (my_tuple)-1)     # Last index : 5
Print ("Last element :", my_tuple[len(my_tuple)-1])
                                             # Last element : t
```

(Doubt)

```
# Index must be within range
# Print (my_tuple[6])    # IndexError: List index out of range

# Index must be an integer
# my_tuple[2.0]    # TypeError: tuple indices must be integers or slices, not float
```

Ex 1

```
n_tuple = ("mouse", [8,4,6,], (1,2,3))

# nested index
Print (n_tuple[2][0])    # 1   ∵ element with index 2 & within sub-element with index 0
Print (n_tuple[0][3])    # 5   ∵ element with index 0 & and within sub-element with index 3
```

Example:

```
t = (3,7,4,2)
Print (t[2])        # 4
Print (t[1:3])      # (7,4)
Print (t[-3])       # 7
Print (t[1:-2])     # (3,7)
Print (t[-1::-1])   # (2,4,7,3)   ∵ start from -1 & step=-1 so it basicly return
```

Example:1)

Deleting tuples

```
my_tuple = ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'n', 'g')

# Can't delete items
del my_tuple[3]    # TypeError: 'tuple' object doesn't support item deletion

# Can delete an entire tuple
# del my_tuple
# Print (my_tuple)    # NameError: name 'my_tuple' is not defined
```

Example 2(1):-

Python Built-in Tuple Functions

Function	Description
len( )	Returns number of elements in a tuple
max( )	Returns item from the tuple with max value
min ( )	Return item from the tuple with min value.
Sorted( )	Return a new sorted list of sequence in the tuple
tuple( )	Converts a sequence into tuple
Comp( )	Compares items of two tuples. (Not available in Python 3)

```
z = ('b', 'n', 'i', 'w')
y = list(z)
Print (y)          # ['b', 'n', 'i', 'w']
y.append('a')
Print (y)          # ['b', 'n', 'i', 'w', 'a']
b = tuple (y)
Print (b)          # ('b', 'n', 'i', 'w', 'a')
```

*
```
t1 = (1, 2, 3, 4, 5)
t2 = ('x', 'y', 'z', [1, 2], 3)
Print (len (t1))   # 5
Print (len (t2))   # 5
```

* max(tuple)

```
t1 = (1, 2, 3, 4, 5)
t2 = ('x', 'y', 'z')
Print (max(t1))    # 5
Print (max(t2))    # z
```

* min(tuple)

```
t1 = (1, 2, 3, 4, 1)
t2 = ('x', 'y', 'z')
Print (min(t1))      # 1
Print (min(t2))      # x
```

* sorted (duct) –

```
Py_tuple = ('e', 'a', 'u', 'o', 'i')
Print (sorted (Py_tuple))                    # ['a', 'e', 'i', 'o', 'u']
Print (sorted (Py_tuple, reverse= True)      # ['u', 'o', 'i', 'e', 'a']
```

* tuple (seq) – The tuple( ) method converts a list of items into tuples.

```
S = "helloworld"
t1 = tuple (S)
List = [1, 2, 3]
t2 = tuple (List)
Print (t1 + t2)      #('h', 'e', 'l', 'l', 'o', 'w', 'o', 'r', 'l', 'd')
Print (t2)           # (1, 2, 3)
```

more ex! z = 'hello'
tuple (z) out # ('h', 'e', 'l', 'l', 'o')

```
z = 'hello'
y = 'hi'
tuple (z+y)  # ('h', 'e', 'l', 'l', 'o', 'h', 'i')
```

```
z = "hello"
p = "hi"
y = list(z)
y.extend("hi")
tuple(y)
```

out : ('h', 'e', 'l', 'l', 'o', 'h', 'i')

```
z = "woo"
y = z[2::-1]
Print(z == y)
```

out : False

Tuple membership test

```
my_tuple = ('a', 'p', 'p', 'l', 'e',)
```

In Operation

```
Print('a' in my_tuple)      # True
Print('b' in my_tuple)      # False
```

Not in Operation

```
Print('g' not in my_tuple)  # True
```

Set { }

Example 1: Different types of sets in Python

```
# Set of integers
my_set = {5, 7, 1, 2, 3}
Print (my_set)          # {1, 2, 3, 5, 7}

# Set of mixed datatypes
my_set = {1, 0, 2, 6, 2, {1.0, 2.6, 2.1, "Hello", (1,2,3), 9, 8}
Print (my_set)          # {1.0, 'Hello', (1,2,3), 9, 8, 2.6, 2.1}
```

```
z = [90, 98]
Set (z)        out: {90, 98}

z = [90, 98]
tuple(z)       out: (90, 98)
```

Example - 2

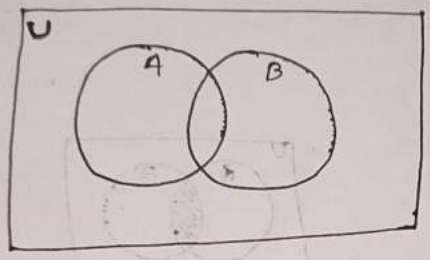
```
# set cannot have deeplicates
my_set = {1, 2, 3, 4, 5, 3, 2}
Print (my_set)          # {1, 2, 3, 4, 5}

# We can make set from a list
my_set = set ([1, 2, 3, 2])
Print (my_set)          # {1, 2, 3}

# set can have immutable items
# here (3,4) is a immutable list
```

```
my_set = {1, 2, (3,4)}
Print (my_set)          # {1,2,(3,4)} ∵ tuple is immutable

# set cannot have mutable items
# here [3, 4] is a mutable list
my_set = {[3,4]}        # this will cause an error ∵ list is mutable
my_set

#Example:- Distiguish set & dictionary while creating empty set
# initialize a with {}
a = {}
# Check data type of a
Print (type(a))         # <class 'dict'> ∵ dict also use {}
# initialize a with set ()
a = set ()
# Check data type of a
Print (type(a))         # <class 'set'>

* y = {}                y = set()          y = {8, 9, 89}
  type (y)              type (y)           type (y)
out: dict               out: set           out: set

# initialize my_set
my_set = {1, 3}
Print (my_set)          # {1, 3}
# my_set[0]
# if you uncomment above line, you will get an error
#TypeError: 'set' object does not support indexing
```

```
# add an element
my_set.add(3.5)
Print(my_set)          # {3.5, 1, 3}

# add multiple elements
my_set.update([2,3,4])
Print(my_set)          # {1, 2, 3.5, 3, 4}

# add list and set
my_set.update([1,7],(8,9,10),{1,5,9,10})
Print(my_list)         # {1, 2, 3, 4, 5, 7, 8, 9, 10}

* z = {9,8,87,98}
  z
  out {8, 9, 87, 98}

z = {9,8,87,98}
z.add(90.9)
: out : {8, 9, 87, 90.9, 98}

z = {1, 3, 3.5}
Print(z)
z.add(2.5)
Print(z)
out : {3, 1, 3.5}
out : {3, 1, 2.5, 3, 5}

# Example :
# initialize my_set
myset = set("Helloworld")
Print(my_set)          # unordered set of unique elements
out : {'W','H','r','d','e','o','l'}
# Pop an element
Print(my_set.pop())    # removes a random element
out: W
```

```
# pop another element
my_set.pop()
print(my_set)          # out: {'r', 'd', 'e', 'o', 'l'}

# clear my_set
my_set.clear()
print(my_set)          # out: set()
```

Set Union (|)



Union of A & B is a set of all elements from both sets. union is performed using | operator. Same can be accomplished using the union() method.

* Example 1:

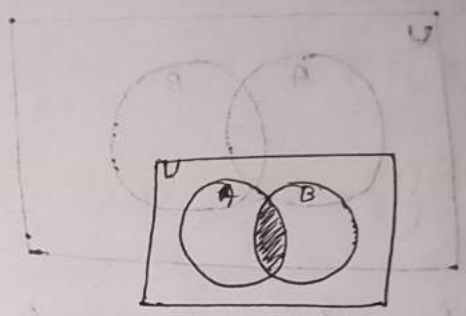
```
# set union method
# initialize A & B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}
# use | operator
A = A | B
Print(A)
```

out: {1, 2, 3, 4, 5, 6, 7, 8}

Example :2

```
# use union function
A = A.union(B)
Print(A)
out: {1, 2, 3, 4, 5, 6, 7, 8}
# use union function on B
B.union(A)
Print(B)
out: {1, 2, 3, 4, 5, 6, 7, 8}
```

Set Intersection (&)



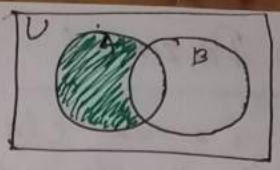
Example - 1:-

```
# intersection of sets
# initialize A & B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}
# use & operator
Print (A & B)        out: {4, 5}
```

or

```
# use intersection function on A
A.intersection(B)    # {4, 5}
# use intersection function on B
B.intersection(A)
out: {4, 5}
```

Set Difference (-)

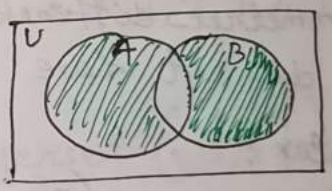



```
# Difference of two sets
# initialize A & B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# use - operator on A
Print (A-B)          # {1, 2, 3}
Print (B-A)          # out: {8, 6, 7}
```

or

```
# use difference function on A
A. difference (B)    # {1, 2, 3}
# use difference function on B
B. difference (A)    # {6, 7, 8}
```

Set Symmetric Difference (^)




```
# symmetric difference of two sets
# initialize A & B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# use ^ operator
Print(A ^ B)        out # {1, 2, 3, 6, 7, 8}
```

or

```
# use symmetric difference function on A
A. symmetric_difference (B)   # {1, 2, 3, 6, 7, 8}
# use symmetric difference function on B
B. symmetric_difference (A)   # {1, 2, 3, 6, 7, 8}
```

Python Frozenset

Frozenset is a new class that has the characteristics of a set, but its elements cannot be change once assigned. While tuples are immutable lists, frozensets are immutable sets.

Set being mutable are unhashable, so they can't be used as dictionary keys. On the other hand, frozensets are hashable & can be used as keys to a dictionary.

Frozensets can be created using the frozenset() function.

The **Frozenset()** function returns an immutable frozenset object initialized with elements from the given iterable.

The data type supports methods like copy(), difference(), intersection(), isdisjoint(), issubset(), issuperset(), symmetric_difference() & union(). Being immutable, it does not have methods that add or remove element.

Syntax:

```
Frozenset([iterable])
```

```
# Example:
# Frozensets
# initialize A & B
A = frozenset([1,2,3,4])
B = frozenset([3,4,5,6])
A.isdisjoint(B)        # False : they have common intersection Points
```

Out: False

```
A.difference(B)        # frozenset({1,2})
```

Out: frozenset({1,2})

A | B

out : frozenset({1, 2, 3, 4, 5, 6})

A.add(3) # Error

'frozenset' object has no attribute 'add'

*Example 1- Frozensets

initialize A & B

A = frozenset([1, 2, 3, 4])

B = frozenset([3, 4, 5, 6])

Copying a frozenset

C = A.copy()

Print(C) # frozenset({1, 2, 3, 4})

Union

Print(A.union(B)) # frozenset({1, 2, 3, 4, 5, 6})

intersection

Print(A.intersection(B)) # frozenset({3, 4})

difference

Print(A.difference(B)) # frozenset({1, 2})

Symmetric_difference

Print(A.symmetric_difference(B))

out! # frozenset({1, 2, 5, 6})

Other frozenset methods :-
isdisjoint(), issubset(), & issuperset()

```
# Example 2 : Frozensets
# initialize A, B & C
A = frozenset([1,2,3,4])
B = frozenset([3,4,1,5])
C = frozenset([5,6])

# isdisjoint() method
Print(A.isdisjoint(C))          # out : True

# issubset() method
Print(C.issubset(B))            # out : True

# issuperset() method
Print(C.issuperset(B))          # out :- True
```

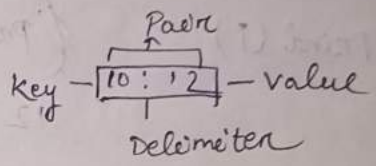
Dictionary (Class - 6)

Python Dictionary:- In this class, you'll learn everything about python dictionaries; how they are created, accessing, adding, removing elements from them & various built-in methods.

* Python dictionary is an unordered collection of items. Each item of an dictionary has a key/value pair.

* d = {10 : 12, 20 : 30, 'a' : 50, 'b' : 70, 'c' : 65}

len (d)



Key
value
Delimeter

d['c'] ∴ out: 65

d['d'] = 50

Print (d)

out: {10:12, 20:30, 'a' : 50, 'b' : 70, 'c' : 65, 'd' : 50}

* z = {'a' : 80, 'b' : 90, 'a' : 98}

z['b'] = 54

z out : {'a' : 98, 'b' : 54}

* z = {90 : 'a', 87 : 'b'}

z[90] = 'x'

z[85] = 'c'

z ∴ out: {90 : 'x', 87 : 'b', 85 : 'c'}

List 1 = [12, 13, 14, 15, 16, 17]

x = tuple (List 1)

x

out: (12, 13, 14, 15, 16, 17)

LOOPS

FOR LOOP

WHILE LOOP

x = [12, 13, 14, 15, 16, 17]

```
for i in x :
    Print (i)
```

```
12
13
14
15
16
17
```

```
Print (i, end=" ")
```

```
12 13 14 15 16 17
```

```
Print (i end = "a ")
```

```
12a 13a 14a 15a 16a 17a
```

z = {90 : 99, 98 : 97}

```
for i in z :
    Print (i)
```

out : 90
98

```
Print (z[i])
```

99
97

x = {88 : a, 90 : c, 92 : e}

```
for i in x
    Print (i)
```

out : 88
90
92

```
x = [12, 13, 14, 15, 16, 17]
y = [ ]
z = [ ]
d = { }
for i in x[::2]:
    y.append(i)
for i in x[1::2]:
    z.append(i)
for i in range(0, 3):          or (0, len(z))
    d[y[i]] = z[i]
d
out : {12:13, 14:15, 16:17}
```

* `z = (10, 20, 30, 40, 50, 60)`

`z = z[::-1]`

`z` `#(60, 50, 40, 30, 20, 10)`

```
y = [ ]
x = [ ]
d = { }
for i in x[0::2]:
    y.append(i)
for i in x[1::2]:
    x.append(i)
for i in range(0,3):
    d[y[i]] = z[i]
```

Print (y)
Print (z)
d

out [60, 40, 20]
[50, 30, 10]

d: {60:50, 40:30, 20:10}

Example 1: How all() works with Python dictionaries?
In case of dictionaries, if all keys (not values) are true or the dictionary is empty,
all() returns True. Else, it returns false for all ~~are~~ other cases.

```
my_dict = {2 : 'False', 1 : 'False'}
Print (all (my_dict))          # ~~False~~ True

my_dict = {1 : 'True', 2 : 'True'}
Print (all (my_dict))          # True

my_dict = {1 : 'True', False : 0}
Print (all (my_dict))          # False

my_dict = { }
Print (all (my_dict)           # True
```

0 is False
'0' is True

```
my_dict = {'0' : 'True'}
Print (all (my_dict))          # True
```

Example 2 any()

```
my_dict = {0 : 'False'}
Print (any (my_dict))          # False : 0 is false

my_dict = {0 : 'False', 1 : 'True'}
Print (any (my_dict))          # True : 1 is True
```

```
my_dict = {0 : 'False', False : 0}
Print (any (my_dict))          # False : both of False are false

my_dict = {}
Print (any (my_dict))          # False : iterable is empty

# 0 is False
# '0' is True
my_dict = {'0' : 'False'}
Print (any(my_dict))           # True
```

Sorted (dict).

```
my_dict = {'e' : 1, 'a' : 2, 'u' : 3, 'o' : 4, 'i' : 5}
Print (sorted(my_dict))                  # ['a', 'e', 'i', 'o', 'u']
Print (sorted(my_dict, reverse = True))  # ['u', 'o', 'i', 'e', 'a']
```

Str (dict)

```
my_dict = {1 : 'Python', 2 : 'Java', 3 : 'C++', 4 : 'PHP'}
Str (my_dict)
Out: "{1 : 'Python', 2 : 'Java', 3 : 'C++', 4 : 'PHP'}"
```

```
Print (type (my_dict))
<Class 'dict'>          type ( )
```

Clear ( ) :

```
my_dict = {1 : 'Pytho', 2 : 'Java', 3 : 'C++', 4 : 'PHP'}
Print (Str (my_dict)
my_dict . clear ( )
Print (Str (mydict)          # { }
```

Copy()

```
dict1 = {1: 'Python', 2: 'Java', 3: 'C++', 4: 'PHP'}
dict2 = dict1.copy()
print(dict2)
```

out: {1: 'Python', 2: 'Java', 3: 'C++', 4: 'PHP'}

item()

```
my_dict = {1: 'Pytho', 2: 'Java', 3: 'C++', 4: 'PHP'}
Print(my_dict.items())
```

Doubt

out:- dict_item([(1, 'Python'), (2, 'Java'), (3, 'C++'), (4, 'PHP')])

Keys()

```
my_dict = {1: 'Python', 2: 'Java', 3: 'C++', 4: 'PHP'}
all_keys = my_dict.key()
print(all_keys)
```

out: dict_keys([1, 2, 3, 4])

Value()

```
dict1 = {1: 'Python', 2: 'Java', 3: 'C++', 4: 'PHP'}
Values = dict1.value()
Print(values)
```

out: dict_values(['Python', 'Java', 'C++', 'PHP'])

Membership Test for Dict keys

```
squares = {1:1, 3:9, 5:25, 7:99, 9:81}
print (1 in squares)          # True
print (2 not in squares)      # True
# membership tests for key only not value
print (99 in squares)         # False
```

—x—

```
z=[1,2,3,4,5]
i=0
while (i< len(z)):
    sum = sum +z[i]
    print (sum)
    i = i+1
```

0 + 1 = 1

1 2 + 1

3

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

```
num = 10
sum = 0
i = 1
While i <= num:
    sum = sum + i
    i = i+1
```

0 + 1 = 1

1 + 1 = 2

3 + 2

2

5

Control Structures

(Class no. 07)

Conditional Statements

IF
ELSE

IF
ELIF
ELSE

IF
IF

IF |

if → NESTED LOOP

ITERATIVE STATEMENTS
FOR
WHILE

SEQUENTIAL STATEMENTS
BREAK
CONTINUE
PASS

'''

* grade = 70
if grade <= 70:
    print ("passing grade")

out! Passing grade

* If the number is positive, we print an appropriate message.

```
num = 3
if (num<0):          # if condition is TRUE: enter the body of if
    Print (num, "is a positive number.")
Print ("This is always printed.")
```

```
num = -1
if num > 0:          # if condition is FALSE: do not enter the body of if
    print (num, "is a negative number.")
Print ("This is also always printed.")
```

Out: This is always printed
This is also always printed.

* Calculate the square of a number if it greater than 6

```
number = int(input ('Enter a number'))
if number > 6:
    # Calculate square
    Print (number * number)
Print ('Next lines of code')
```

out: Enter a number 7
49
Next lines of code

* Ex- Check if num1 is less than num2

```
num = int (input ('Enter a number'))
num = int (input ('Enter a number'))
if (num1 < num2):
    Print ("num1 is less than num2")
```

Out! Enter a number 4
Enter a number 10
num1 is less than num2

(*) num = int(input('Enter a number'))
if (num % 3 == 0):
    Print ('The number is divisible by 3')

Out: Enter a number 15
The number is divisible by 3

(*) num = int(input('Enter a number'))
if (num % 2 == 0):
    Print ('Number is even')

Out: Enter a number 12
Number is even

* x = 12
if x > 10 :
    Print ("Hello")

out: Hello

* if 5 != 3 * 6 :
    Print ("Hooray!")

(!=) - not equal to

Out: Hooray!

* if 5 == 15/3 :
    print ("Horray!")
Print ("Hi")
Out: Horray!
Hi

(*) num1 = int(input('Enter a number for num1'))
num2 = int(input('Enter a number for num2'))
temp = num1
num1 = num2
num2 = temp
Print('The value in num1 is', num1)
Print('The value in num2 is', num2)

Out:- Enter a number for num1 25
Enter a number for num2 50
The value in num1 is 50
The value in num2 is 25

(*) num1 = int(input('Enter a number for num1'))
num2 = int(input('Enter a " " num2'))
num1, num2 = num2, num1
Print(num1, num2)

Out; Enter a number for num1 12
" " num2 14.
14 12

* # num1, num2, num3 :
num1 = int(input('Enter a number for num1'))
num2 = int(input(" " num2'))
num3 = int(input(" " num3'))

```
if (num1 < num2 and num1 < num3):
    print (num1)
if (num2 < num1 and num2 < num3):
    print (num2)
if (num3 < num1 and num3 < num2):
```

Out: Enter a number for num1 24
" " " num2 25
" " " num3 12
12

if - else statement :

```
x = 1
if x > 3:
    print ("Case 1")
if x <= 3:
    print ("case 2")
```

Out: Case2

Ex:-

```
grade = 60
if grade >= 65:
    print ("passing grade")
else:
    print ("Failing grade")
```

Out:- Failing grade.

* Ex- Program checks if the number is positive or negative & displays an appropriate

```
num = -90
# Try these two variations as well.
# num = -5
# num = 0
if num >= 0:
    Print ("Positive or zero")
else:
    Print ("Negative number")
```

Out: Negative number

* Ex- Program to check if a num1 is less than num2

```
num1, num2 = 6, 5
if (num1 < num2):
    Print ("num1 is less than num2")
else:
    Print ("num2 is less than num1")
```

Out: num2 is less than num1

* hungry = 1

```
x = "Feed the bear now!" if hungry else "Do not feed the bear."
Print (x)        # Do not feed the bear

hungry = 0
if hungry:
    Print ('Feed the bear now!')
else:
    Print ('Do not feed the bear')
```

Out: Feed the bear now!

* x = 12
if 10 < x > 11:
    print ("hello")
else:
    print ("world")
out: (hello)

* Program to check if num1 is less than num2

num1, num2 = 5, 5
if (num1 < num2):
    Print ("num1 is less than num2")
else:
    Print ("num2 is less than num1")

"""
output: num2 is lesser than num1
"""

if elif else :-

Ex-

In this program, we check if the number is positive or negative or zero & display an appropriate message.

num = 0
# Try these two variations as well:
# num = 0
# num = -4.5
if num > 0:
    print ("positive number")
elif num == 0
    print ("zero")
else:
    print ("negative number")

print ('outside')

Out! zero
outside

* num1, num2 = 5, 5
if (num1 > num2):
    print ("num1 is greater than num2")
elif (num1 < num2):
    print ("num1 is equal to num2")
else:
    print ("num1 is less than num2")

Out: num1 is less than num2

* x = 10
y = 12
if x > y:
    print ("x > y")
elif x < y:
    print ("x < y")
else:
    print ("x = y")

Out: x < y

* grade = 67
if grade >= 90:
    print ("A grade")
elif grade >= 80:
    print ("B grade")
elif grade >= 70:
    print ("C grade")

```
elif grade >= 65:
    Print ("D grade")
else:
    Print ("Failing grade")
```

Out: D grade

if-elif-else statements with logical operators. (and, or)

*
```
a = 0
if a>0 or a%2 == 0
        Print ('A is an even & positive integer')
elif a>0 and a%2!=0:
        Print ('A is a positive integer')
elif a==0:
    Print ('A is zero')
else:
    Print ('A is negative')
```

Out: A is an even & positive integer

*
```
x = str(input ('Enter the fruit')
fruits = ['banana', 'orange', 'mango', 'pear']
if x in fruits:
    Print ('That fruit already exist in the list')
else:
    fruit.append(x)
    Print ('The modified fruits are', fruits)
```

out: Enter the fruit kiwi
The modified fruits are ['banana', 'orange', 'mango', 'pear', 'kiwi']

H.W

* a = 10
if a >= 20: # Condition FALSE
    Print ("Condition is True")
else: # Code will go to ELSE body
    if a >= 15: # Condition FALSE
        Print ("Checking second value")
    else: # Code will go to ELSE body
        Print ("All conditions are false")

out: All conditions are false

* x = 10
y = 12
if x > y:
    Print ("x > y")
elif x < y:
    print ("x < y")
    if x == 10:
        Print ("x = 10")
    else:
        Print ("invalid")
else:
    print ("x = y")

out: x < y
invalid

```
num1 = 0
if (num1 != 0):          # For zero condition is FALSE
    if(num1 > 0):
        print ("num1 is a positive number")
    else:
        print ("num1 is a negative number")
else:                    # For zero condition is TRUE
    print ("num1 is neither positive nor negative")
```

Out: num1 is neither positive nor negative

* Program to find whether a number is positive or no

```
z = int(input('Enter a number:'))
if (z>0):
    if(z>10):
        print ('Number is greater than 10')
    else:
        print ('Number is less than 10')
else:
    print ('Negative')
```

Out: Enter a number : 5

Num is less than 10

* Program to find a number is even or not

```
z = int(input('Enter a number'))
if (z%2 ==0)
    print ('even')
else:
    print ('Odd')
```

Out: Enter a number 20
Even

* Program to find the first 3 multiples of 2

```
z = 21
if (z==2 and z==4 and z==6):
    print('First 3 multiples')
else:
    print('Wrong Output')
```

Out: Wrong Output

Python While Loop :-

* Print numbers less than 5

```
Count = 1    # variable --> 1
# run loop till count is less than 5
While count < 5 :   #4
    print (Count) #4
    Count = Count + 1  #5
```

Out: 1
2
3
4

* for i in range (0,5):
    print(i)

out: 0
1
2
3
4

* i=0
while (i<5):
    print (i)
    i = i+1

out: 0
1
2
3
4

* i = 2
while (i <= 20):
    Print (i)
    i = i + 3

out: 2
5
8
11
14
17
20

* z = [21, 34, 21, 23, 45, 6, 7, 888, 909]
i = 1
while (i < len (z)):
    Print (z[i])
    i = i + 2

out: 34
23
6
888

* z = [1, 2, 3, 4, 5]
i = 0
sum = 0
while (i < len (z)):
    sum = sum + z[i]
    i = i + 1
Print (sum / len (z))

out: 3.0

* Sum of first ten numbers

num = 10
sum = 0
i = 1
while i <= num:
    sum = sum + i
    i = i + 1
Print ("sum of first 10 number is :", sum)

out: Sum of first 10 number is : 55

09]

```
* p = int(input("Enter a number"))
  i = 1
  while (i<p):
      print(i**2)
      i = i+1
```

Out! Enter a number 5
1
4
9
16

```
* a = 10          # 'a' is my variable
  while a>0:      # Enter the body of while loop becz
                    condition is TRUE
      print("Value of a is", a)
      a = a-2
  Print("Loop is completed")
```

Out! Value of a is 10
" " " 8
" " " 6
" " " 4
" " " 2
Loop is completed

* How many times a given number can be divided by 3 before it is less than or equal to 10.

```
Count = 0
number = 180
while number > 10:
      # devide number by 3
      number = number /3
      # increase count
      Count = Count + 1
Print('Total iteration required', Count)
```

Out! Total iteration required 3

no-8) (*) z = [1, 12, 14, 15, 18, 19]

```
Even_list = [ ]
Odd_list = [ ]
for i in z:
    if (i % 2 == 0):
        Even_list.append(i)
    else:
        Odd_list.append(i)
Print ('Even list values are', Even_list)
Print ('Odd list values are', Odd_list)
```

Out:- Even List values are [12, 14, 18]

Odd List values are [1, 15, 19]

(*) words = ['one', 'two', 'three', 'four', 'five']

```
for i in words:
    print(i, end=" ")
Print(i)                    # out: one two three four five
```

(*) numbers = [10, 20, 30, 40, 50]

```
s=0
for i in range(2, 5):
    s = s + numbers[i]
Print(s)                    # out: 120
```

(*) Calculate the average of List of numbers

numbers = [10, 20, 30, 40, 50]

```
sum = 0
for i in numbers:
    sum = sum + i
```

sum = 0+10 --> 10
sum = 10+20 --> 30
sum = 30+30 --> 60
sum = 60+40 --> 100
sum = 100+50 --> 150

```
List_size = len(numbers)
average = sum / List_size
print (average)
```

Out: 30.0

(*) for i in range(2,12,2):
    Print(i)

out: 2 4 6 8 10

(*) num = 5
for a in range(1,6) # (1,2,3,4,5)
    Print(num * a) # 2*1 # 2*2 # 3*2

out: 2 4 6 8 10

Break in for Loop

(*) numbers = (10, 11, 12, 13, 14, 15)
for number in numbers:
    if number > 10 and number <= 13:
        print(number)
    else:
        break
Print('Done') # out: Done

(*) Color = ['Blue', 'Pink', 'Green', 'Pink', 'Yellow']
for i in colour:
    if (i == 'Green' and i == 'Blue'):
        Print(i)
        Break
Print(i) out: Yellow

(if 'or' Blue Blue)

(*) numbers = [1, 4, 7, 8, 15, 20, 35, 41, 55]
for i in numbers:
    if i > 15:
        break
    else:
        Print(i)

out: 1 4 7 8 15

(*) for i in range(0,3):
    for j in range(0,2):
        if j == i:
            break
        Print(i,j)

out: 1 0
2 0
2 1

Continue in for loop

```
color = ['Green', 'Pink', 'Blue', 'Yellow', 'Red']
for i in color:
    if (i == 'Pink'):
        Continue
    else:
        print(i)
```

out: Green
Blue
Yellow
Red

```
numbers = (0, 1, 2, 3, 4, 5)
for number in numbers:
    print (number)
    if number == 3:
        Continue
    print ('next number should be', number+1) if number != 5 else print("Loop's end")
print ('outside the loop')
```

out:
```
0
next number should be 1
1
Next " " " 2
2
" " " 3
3
4
Next " " " 5
5
loop's end
outside the Loop
```

Pass in loop

```
for number in range(6):
    pass
if num > 5:
    print 'hello'
```

```
num = [1, 4, 5, 3, 7, 8]
for i in num:
    pass
    print(i)
```

out: 1 4 5 3 7 8

recursion

Class - 9

What is a function in Python?

A function is a organized block of code.

(meaning - we are writing something within a given block or boundry)

Types of Functions:

i) Built - in function (which is already present or define)

ii) User - defined function (which are define by us)

different functions are ~~range~~

range(), print(), input(), type(), id(), eval()

```
def function name (parameter 1, parameter 2):
    # function body
    # Write some action
return value
```

(imp thing - ✓)

(def → define or definition)

* any type of logic we have to input on function body.

Example:

```
def add(num1, num2):
    Print("Number 1: ", num1)
    Print("Number 2: ", num2)
    addition = num1 + num2
    return addition

res = add(2, 4)
Print("Result:", res)
```

function name: 'add', parameter 'num1', 'num2'

Function body

return value

Function call

Defining a() without any parameters

def - defination

(Class no. - 9)

```
def add_number (x, y, z):
    w = x + y + z
    return w
print (add_numbers(2,6,7))
```

Out :- 15

```
z = [1,2,3,4,5]
s = 0
for i in z:
    s = s + i   #1
print(s)
```

Out : 15

(for loop)

```
def List_addition_numbers(z):
    s = 0
    for i in z:
        s = s + i
    return s
z = [1, 2, 3, 4, 5]
List_addition_numbers(z)
```

Out :- 15

```
def generate_product ():
    product = 1
    for i in range (1,7):
        product = product * i    # 1*2*3*4*5*6
    return product
generate_product ()
```

Out : 720

```
def generate_product (a):
    product = 1
    for i in range(
```

(x) def even_lost():
    z = []
    for i in range(1,11):
        if i%2==0:
            z.append(i)
    return z

even_lost()

out: [2,4,6,8,10]

(i%2!=0
odd_lost())

(*) z = [14, 19, 22, 87, 90, 100]

def div_five(a):
    for i in a:
        if i%5==0:
            print(i)

div_five(z)

(*) z = [12, 15, 18, 21, 91, 98, 97, 90]

def dive_del_two(a):
    for i in a:
        if i%2==0:
            z.remove(i)

    return z

div_del_two(z)

out! [15, 21, 91, 97]

```
def greet(x,y):
    print("Welcome to Python for data Science",x,y)
# call function using its name
greet('Its over','The End')
```

Out: Welcome to Python for data Science Its over The End

```
def two_numbers(w,z):
    y = w/z - w   # BODMAS
    return y
print(two_numbers(12,6))
```

Out: 10.0

Types of functions-

1) No Parameter, But Some Return Value
2) No Parameter, No Return value
3) Some parameter, No Return value
4) Some parameter, Some Return value

```
# Some Parameter, Some Return Value
def two_num(x,y):
    z = x+y
    return z
print(two_num(8,9))
# Some Parameter, No Return Value
def two_num(x,y):
    z = x+y
    print(z)
two_num(8,9)
# No. Parameter, No Return Value
def two_num():
```

```
x = 9
y = 8
z = x+y
print (z)
two_num()
```

No Parameter, But some Return value

```
def two_num():
    x = 9
    y = 8
    z = x+y
    return z
two_num()
```

out: 17
17
17
17

(*) def greet (name):

```
    """
    This function greets to the person passed in
    as a parameter
    """
    print ("Hello," + name + ". Good morning!")
greet ("HI")
```

out: Hello.. HI . Good morning!

(*) def sum_of_numbers (n):

```
    total = 0
    for i in range (n+1):
        total = total + i
    Print (total)
sum_of_number (10)
sum_of_number (100)
```

out:- 55
5050

```
def course (name, course_name):
    Print("Hello", name, "Welcome to Python for Data Science")
    Print("Your course name is," course_name)
course ('Arthur', 'Python')
```

out: Hello Arthur Welcome to Python for Data Science
Your course name is Python

```
def greetings (name):          # single parameter
    message = name + ', welcome to Python for Data Science'
    return message
Print(greetings ('Nilaan'))
```

out: Nilaan, welcome to Python for Data Science

```
def add_ten (num):          # single parameter
    ten = 10
    return num + ten
Print (add_ten(90))
```

out: 100

```
x = int(input ('Enter your number'))
Choice = int (input ('Enter your choice'))

def square_number (x, choice):
    if (choice == 1):
        return x ** 2
    elif choice == 2:
        return x ** 3
```

```
else:
    (x+x)
print(square_number(x, choice))
```

Out: Enter your number 4
" " Choice 5

8
None

```
def square_number (x, choice):
    if (choice == 1):
        if (x % 5 == 0):
            return ('x is divisible by 5')
    elif (choice == 2):
        if (x % 10 == 0):
            print ('x is divisible by 10')
    elif (choice == 3):
        if (x % 2 == 0):
            print ('x is divisible by 2')
            return 10

    else:
        print ('x is not divisible by 5 & 2')
x = int(input('Enter your number'))
choice = int (input ('Enter your choice'))
square_number (x, choice)
```

Out- Enter your number 20
" " choice 2
x is divisible by 10

Types of Arguments / parameters

1- Required Parameters
2- Keyword "
3- Default "
4- Variable length "

(*) def add_sum (x, y):
        return x+y
    add_sum (12, 14)

out: 26

(*) def add_sum (x, y):
        return x+y
    add_sum (y=9, x=10)

out: 19

(*) def add_sum (x=10, y=20):
        return x+y
    add_sum( )

out: 30

(*) def add_sum (*z):
        for i in z
            print(i)
    add_sum (11, 12, 10, 8, 9)

*z - address of z

out: 11
12
10
8
9

Class-10

Global Variables

```
def two_num(x, y):
    z = x + y
    return z
two_num(10, 20)
print(z)
```

out: 30

(print(x) - name 'y' is not define)
(local variable)

Create a global variable

```
global_var = 999
def fun1():
    print("value is 1st function:", global_var)
def fun2():
    print("value is 2nd function:", global_var)
fun1()
fun2()
```

Out: value in 1st function : 999
value in 2nd function : 999

```
def fun():
    print("x inside :", x)
x = "hello"
fun()
print("x outside:", x)
```

~~Print("x~~
out: x inside : hello
x outside : hello

```
x = 'global'
def fun():
    z = x * 2
    print(z)
fun()
```

out! globalglobal

```
x = 'global'
def fun():
    x = 20
    z = x * 5
    print(z)
fun
```

out! 100

if x=30 output 200

```
global_lang = 'DataScience'
def var_scope_test():
    local_lang = 'Python'
    print(local_lang)
var_scope_test()          # output 'Python'
# outside of function
print(global_lang)        # Output 'DataScience'
print(local_lang)         # NameError: name 'local_lang' is not defined
```

```
a = 90                    # Global variable
def print_data():
    a = 6                 # local variable
    b = 30
    print("(a, b):(",a,",",b,")")
print_data()
print("Global a:",a)      # Global x : 50
print("Local b:", b)      # b is local variable - throw NameError
```

out: (a, b) : (6, 30)
Global a: 90
Error

* Ex1:- Accessing local variable outside the scope

```
def fun():
    y = "local"
fun()
Print(y)
```

out: name 'y' is not defined

*
```
def fun1():
    loc_var = 999        # local variable
    Print ("Value is :", loc_var)
def fun2():
    Print ("Value is :", loc_var)
fun1()
fun2()
```

Out: Value is 999
Error

* Ex- Using Global & local variable in same code

```
x = "global"
def fun():
    global x
    y = "local"
    x = x * 2
    Print (x)
    Print (y)
fun()
Print(x)
```

(doubt)

out: globalglobal
local
globalglobal

(*) Ex2: Global variable & Local variable with same name

```
x = 9
def fun():
    x = x+1
    print("Local x:", x)
fun()
print("global x:", x)
```

out: local variable 'x' referenced before assignm

(*) def my_func():

```
    x = 10
    Print("value inside the body of function:", x)
x = 20
my_func()
Print("value outside of function:", x)
```

out: value inside the body of function: 10
value outside of function: 20

(*) Create a nonlocal variable

```
x1 = "y"
def outer_fun():
    x1 = "local"
    Print(variable type for outer function:", x1)
    def inner_fun():
        global x1
        x1 = "nonlocal"
        print("variable type for inner function:", x1)
    inner_fun()
outer_fun()
print("variable type of x1:", x1)
```

out: Variable type for outer function: local
" " " inner " : nonlocal
" " of x1 : nonlocal

```
def outer_fun()
    # global x
    x = 999
    def inner_fun():
        # local variable now acts as global variable
        global x
        #x = 900
        Print("value of x inside inner function is:", x)
    inner_fun()
    Print("value of x inside outer function is:", x)
x = 1000
outer_fun()
x = 1000
Print('The outer value of x is', x)
```

out: 1000
999
1000

Recursion :-

Function call with in a function. Class-11

Example 1:

```
def factorial(n):
    if n==1:
        return 1
    else:
        return (n * factor(n-1))
# 5 * factorial(4)    4* factorial(3) ....

num = 5
Print ("The factorial of", num, "is", factorial(num))
```

out! The factorial of 5 is 120

n=1

```
for i in range (1,6):
    n = n*i      # 120
    n
```

out ! 120

(*) Fibonacci_sequence (n): 0+1, 1+2, 2+3, 3+4 ....

```
def fibonacci_sequence(n):
    if n<=1:
        return n
    else:
        return (fibonacci_sequence(n-1)+ fobonacci_sequence(n-2))

num = int(input('Enter a Number'))   (work start from here)
if ntem <=0:
    print ('Invaled')
else:
    print ('Fibonacci sequence:')
```

```
for i in range (num):
    print (fibonacci_sequence(i))
```

out:- Enter a number 5
Fibonacci sequence:
0
1
1
2
3

Python Exceptions Handling Using try, except & finally statement:

Catching Exceptions in Python

* Exception : It is a type of Error
* Error is not an Exception
* Porpose to read - To handle it -- Handle exceptions

4 blocks in case of exception handling

try: if we feel a code any part is break, then we write on try block

except: if code break, then it come except

else; if code run sucessfully then it come else

finally: It is always get execuated

```
* try:
    print ('10'+'5')
  except:
    print ('You cannot add a integer to a strong')
```

out:- You cannot add a iteger to a strong.

```
try:
    x = 1/int('z')
except:
    print('you cannot add a integer to a String')
```

out: ValueError: invalid literal for int() with base 10: 'z'

```
import sys
randomList = ['a', 5, 0]
for entry in randomList:
    try:
        print("The entry is", entry)
        r = 1/int(entry)
        # break
        # continue
        pass
    except:
        print("oops!", sys.exc_info()[1], "occurred.")
        print("Next entry.")
        print()
print("The reciprocal of", entry, "is", r)
```

from sys import *

Then sys is removed

Out: The entry is a
oops! invalid literal for int() with base 10: 'a' occurred.
Next entry.

The entry is 5
The entry is 0
oops! division by zero occurred
Next entry

The reciprocal of 0 is 0.2

```
import sys
randomList = ['a', 0, 2]
for entry in randomList:
    try:
        print("The entry is", entry)
        r = 1/int(entry)
        break
    except Exception as e:
        print("Oops!", e.__class__, "occurred.")
        print("Next entry.")
```

Out:-

```
The entry is a
Oops! <class 'ValueError'> occurred.
Next entry.

The entry is 0
Oops! <class 'ZeroDivisionError'> occurred
Next entry.
```

```
try:
    name = input('Enter your name:')
    year_born = input('Year you were born:')
    print(type(year_born))
    age = 2022 - year_born
    # print(f'you are {name}. And your age is {age}.')
    print(name, age)
except:
    print('something went wrong')
```

out! Enter your name : Ram
Year you were born : 3
<Class 'str'>
Something went wrong

* try:
    name = str (input ('Enter your name :'))
    year_born = (input ("Year you were born :"))
    age = 2022 + year_born
    print (f'You are {name}. And your age is {age}')
except TypeError :
    print ('Type error occured')
except ValueError :
    print ('value error occured')
except ZeroDivisionError :
    print ('zero division error occured')

* try:
    a = int (input ("Enter a positive integer :"))
    if a<=0:
        raise ValueError ("That is not a positive number!")
except ValueError as ve:
    print (ve)

Out Enter a positive integer : -9
That is not a positive number!

(*) try:
```
    name = str(input('Enter your name:'))
    yearborn = int(input('Year you born:'))
    age = 2022 - (year_born);
    print(f'you are {name}. And your age is {age}.');
except TypeError:
    print('Type error occur')
except ValueError:
    print('value error occured')
except ZeroDivisionError:
    print('zero division erro occur')
else:
    print('I usually run with the try block')
finally:
    print('I always run!')
```
out: Enter your name: q
Year you born: 2000
You are q. and your age is 22.
I usually run with the try block
I always run.

(*) try:
```
name = input('Enter your name:')
year_born = input('year you born:')
age = 2022 - int(year_born)
print(f'you are {name}. And your age is {age}.
```

```
except Exception as e:
    print (e)
```

out: Enter your name : Rahul
Year you born : 1923
You are Rahul. And your age is 99.

Argument of an Exception -

(*)
```
def temp_convert (var):
    try:
        return int (var)
    except ValueError as Argument:
        Print ("The argument does not contain numbers",
temp_convert ("xyz")                                Argument)
```

out: The argument does not contain numbers
invalid literal for int() with base 10: 'xyz'

(*)
```
def functionName (level):
    if level < 1:
        raise Exception (level)
    return level

try:
    l = functionName (-90)
    Print ("level = ", l)
except Exception as e:
    Print ("error on level argument", e.args[0])
```

out: error on level argument -90

* Packing & Unpacking Arguments in Python

We use two operators:

- * for tuples
- ** for dictionaries

Unpacking Lists

```
def sum_of_five_nums(a,b,c,d,e):
    return a+b+c+d+e
lst = [1,2,3,4,5]
Print(sum_of_five_nums(lst))
```

output: TypeError

```
def sum_of_five_nums(a,b,c,d,e):
    return a+b+c+d+e
lst = [1,2,3,4,5]
Print(sum_of_five_nums(*lst))
```

output is-

```
numbers = range(2,7)
print(list(numbers))
args = [2,7]
numbers = range(*args)
print(numbers)
```

output [2,3,4,5,6]

range(2,7)

```
countries = ['Finland', 'Sweden', 'Norway', 'Denmark', 'Iceland']
fin, swe, nor, *rest = countries
print(fin, swe, nor, rest)

numbers = [1, 2, 3, 4, 5, 6, 7]
one, *middle, last = numbers
print(one, middle, last)
```

out: Finland Sweden Norway ['Denmark', 'Iceland']
1 [2, 3, 4, 5, 6] 7

```
def unpacking_person_info(name, country, city, age):
    return f'{name} lives in {country}, {city}. He is {age} year old.'

dct = {'name': 'Milaan', 'country': 'England', 'city': 'London', 'age': 96}
print(unpacking_person_info(**dct))
```

out: Milaan lives in England London. He is 96 years old.

```
def sum_num(*l):
    for i in l:
        print(i)
sum_num(90, 9, 8, 7, 7, 6)
```

out: 90
9
8
7
7
6

```
def sum_num(a, b):
    print(a)
    print(b)
z = {'a': 12, 'b': 90}
sum_num(**z)
```

out: 12
90

Enumerate

```
for index, item enumerate([20, 30, 40]):
```

```
for i, j in enumerate([20, 30, 40]):
    Print(i, j)
```

out:
```
0  20
1  30
2  40
```

Zip

```
fruits = ['banana', 'orange', 'mango']
vegetables = ['Tomato', 'Potato', 'Cabbage']
fruits_and_veges = []
for f, v in zip(fruits, vegetables):
    fruits_and_veges.append({'fruit': f, 'veg': v})
Print(fruits_and_veges)
```

suppose ({f+v})

out: `[{'fruit': 'banana', 'veg': 'Tomato'}, {'fruit': 'orange', 'veg': 'potato'}, {'fruit': 'mango', 'veg': 'cabbage'}]`

out: `['banana Tomato', 'orange Potato', 'mango Cabbage']`

Class-12

```
age = input('Enter Your Age:')
try:
    if page <= 18:
        print('eligible')
    else:
        print('not eligible')
except:
    print('Something went wrong')
```

Out: Enter Your Age : 15
<class 'str'>
Something went wrong

```
a = int(input())
b = int(input())
try:
    print(a/b)
except TypeError as e:
    print(e)
except ValueError as e:
    print(e)
except IndexError as e:
    print(e)
except KeyError as e:
    print(e)
except SyntaxError as e:
    print(e)
except ZeroDivisionError as e
    Print(e)
except Exception as e:
    print(e)
```

```
try:
    z = '5' / '0'
except TypeError as e:
    print(e)
```

out: unsupported operand type(s) for /: 'str' and 'str'

```
age = input()
if age.isnumeric():
    if int(age) >= 18:
        print('elegible')
    else:
        raise IndexError('not elegbe')
else:
    raise ValueError('Enter only Numeric Value')
```

out: 12
IndexError: not elegobe

```
try:
    name = 'nishad'
    print(name, name[10])
except (NameError, ValueError, IndexError) as err:
    print(err)
finally:
    print('done')
```

out: String index out of range
done

```
try:
    name = 'neshad'
    print (namee[10])
except NameError as err:
    print (err)
except ValueError as err:
    print (err)
except IndexError as err:
    print (err)
```

out: name 'namee' is not defined

```
try:
    name = 'neshad'
    print (name, name.index('x'))
```

out: substring not found

Opening Files in Python:

(file handle)

- Open a file
- Close the file
- Wrote onto file
- Read contents of files

- one, top of a file

Perform operation

* Various mode present in file handle

Mode	Description
r	Read - Opens a file for reading only.
t	text - Opens in text mode. (default)
b	Binary - Opens in binary mode (images)
x	Create - Opens a files for exclusive creation if the file already exists, the operation fails
rb	Opens file for reading only in binary format
r+	Opens a file for both reading and writing
rb+	Opens a file for both reading and writing in binary format.
w	write - Opens a file for writing only. Overwrites the file if the exists. Create a new file
wb	Opens a file for writing only in binary format. Overwrites the file if the file exists
w+	Opens a file for both writing and reading Overwrites the existing file if the file ex
wb+	Opens a file for both writing & reading in binary format. Overwrite the existing file
a	Append - Opens a file for appending. The file pointer is at the end of the file if the file exists.
ab	Opens a file format. The file pointer is at the end of the file if the file exists.
a+	Opens a file for both appending & reading.
ab+	Opens a file for both appending & reading in binary format. if not exists than create new.

(*) f = Open ("test.txt")
print (f)

Out: <_io.TextIOWrapper name='test.text' mode='r' encoding='cp1252'>

(*) f = Open ("test.text", 'w')
print (f)

Out: <_io.TextIOWrapper name='test.txt' mode='w' encoding='cp1252'>

(*) # Open a file
data = Open("data.text", "wb")
print ("Name of the file:", data.name)
print ("Closed or not:", data.closed)
print ("Opening mode:", data.mode)
data.close() # closed data.text file

Out: Name of the file: data.txt
Closed or not: False
Opening mode: wb

(PWD - Present Working directry)

Cd Documents - change to Documents folder

(3) Writing to files on Python

* with Open ("test-1.txt", 'w') as f:
f.write ("my first file\n")
f.write ("This file\n\n")
f.write ("contains three lines\n")

* z = "hello\n"
z
Out: 'hello\n'

or

z = "hello\n"
print (z, end=" ")
out: hello

Various function

Method	Description
Close( )	Closed an opened file.
detach( )	Separates the underlying binary buffer from the TextIOBase and returns it.
fileno( )	Returns an integer number of the file.
flush ( )	Flushes the write buffer of the file stream.
isatty ( )	Returns True if the file stream is interactive
read(n)	Returns Reads at most n characters from the file. Reads till end of file if it is negative or None.
readable ( )	Returns True if the file stream can be read form.
readline(n=-1)	Reads, returns a ~~list~~ one of lines from the file.
readlines(n=-1)	" " a list " " "
seek(offset, from = SEEK_SET)	Changes the file position to offset bytes.
Seekable ( )	Returns True if the file stream supports random access.
tell ( )	Returns the current file location
truncate (size = None)	Resizes the file stream
Writable ( )	Returns True if the file stream can be written to.
write (s)	writes the string s to the file & returns the number of characters written
Writelines (lines)	Writes a list of lines to the file.

(*) Remove the file - import os
os.remove('example.txt')

Example=

1) Read/Text -- rt
2) WRITE -- w
3) BINARY -- b
4) APPEND -- a

* PWd
out 'C:\\Users\\TANMAYA SAHU\\Documents'
* w= open ('file1', 'w') ('file1.text', 'w')
Print (w)
out: <_io.TextIOWrapper name='file1.txt' mode='w' encoding='cp125
* w= open ('file1.txt', 'r')
Print (w)
out: <_io.TextIOWrapper name='file1.txt' mode='r' encoding='cp125
* with open ('file1.txt', 'w') as q:
q.write ('This is a random text\n')
q.write ('This is another random text')
* ~~with~~ w = open ('file1.text', 'r')
print (w.read())
out: This is a random text \n'
This is another random text
* with open ('file1.txt', 'a') as q:
q.write ('\nThis is a random text \n')
q.write ('This is another random text')
* w = open ('file1.txt', 'r')
Print (w.read(10)) (only 10 character)
out: This is a
* w = open ('file1.txt', 'r') (read only 1st line)
print (w.readline()) out: This is a random text
* Print (w.readline()) out: This is another random text (2nd line)

Then open with two

Class-13

Python Logical Errors (Exceptions)

type of errors in exceptions

* if a<3 out: SyntaxError : invalid syntax (missing :)
* Print 'hello world' out: SyntaxError : missing parentheses
* 1/0 out: ZeroDivisionError: division by zero
* r = open ('imgonary.txt', 'w')
  out: FileNotFoundError : No such file or directory: 'imaginary.txt'
* from math import power
  out: ImportError - Cannot import name 'power' from 'math'
* Print (age) out: NameError
* numbers = [1, 2, 3, 4, 5]
  numbers [5] out: IndexError
* 6 + '3' out:- TypeError
* ('19a') Out:- ValueError

* AssertionError
* AttributeError
* EOFError
* FloatingPointError
* GeneratorExit
* ImportError
* IndexError
* KeyError
* KeyboardInterrupt
* MemoryError
* NameError
* NotImplementedError
* OSError
* OverflowError
* ReferenceError
* RuntimeError
* StopIteration
* IndentationError
* TabError
* SystemError
* systemExit
* UnboundLocalError
* UnicodeError
* UnicodeTranslateError
* UnicodeEncodeError
* UnicodeDecodeError

Object Oriented Programming :

An Object has two characteristics:
- attributes (Properties)
- behaviour (function)

ex:- • name, age, color as attributes
• singing, dancing as behaviour

CLASS → Combination Of multiple functions & variables

OBJECT → INSTANCE of A CLASS

Class → multiple objects

FUNCTION → MANY FUNCTION CALLS

Class → Parameters (n parameters)

N functions → Class Methods

N # variable → Class variables

Function → Call

Class → Object

Function →

Definition → def

Body → logic / implementation

Return type → Output

Class → function → return

object of the Class → function

def employee :

Class employee :

e1 = employee()
e2 = employeee()

e1 → Object of the class employee
e2 → Object of the Class
e1 & e2 are independent

- Group of functions → Modules
  Group of modules. → Libraries

request

(*) Example:

```
import math
# math is a module
math.pi
x = math.sin(90)
x
import random
# random is a module
z
```

out: 3.141592653.....

Class

In Python, everything is an object.

```
Class Person:
    pass
Print (Person)
def Person():
    Print (Hi)
```

out: <Class '__main__.Person'>

python → Main Class
Execution → Main

```
* Class Student:
    """ This is student class with data """
    def learn(self):                    # A sample method
        Print("Welcome to class on Python programming")
Stud = Student()                        # Creating object.
Stud.learn()                            # Calling method
Stud1 = Student                         # Creating object
Stud1.learn()
out: Welcome to class on Python Programming
     "    "    "    "    "    "

* Class Person:
    def __init__(self, name, age):
        # self allows to attach parameter to the class
        self.name = name
        self.age = age
    def display(self):
        Print(self.name)
P = Person('Rahul', 98)
P.display()
# z = person('w', 78)
# Print(z.name)
# Print(z.t)
# Print(P)
out: Rahul
```

```
class Person:
    def __init__(self, firstname, lastname, age, country, city):
        self.firstname = firstname
        self.lastname = lastname
        self.age = age
        self.country = country
        self.city = city
    def marks(self):
        print("My Name is", self.firstname, self.lastname,
              "whose age is", (self.age))
P = Person('RAHUL', 'KOHLI', 96, 'England', 'London')
print(P.firstname)
print(P.lastname)
P.marks()
print(P)
```

out:
```
RAHUL
KOHLI
My Name is RAHUL KOHLI whose age is 96
<class '__main__.Person'>
```

Instance Variables & Methods

Class-14

if the value of a variable varies from object to object, then such variables are called instance variables.

```
def mark (self):                                   instant variables
    print ("My Name is", self.firstname self.Lastname,
Instant method                "whose age is", (self.age))

def __init__ (self, name, percentage):
    constructor
```

#Ex Creating class & object in Python

```
Class Student:
    def __init__ (self, name, percentage):
        self.name = name
        self.percentage = percentage
    def show(self):                    ← (the values are fixied so show is not instant method)
        a = 10
        b = 20
        c = 30
        return a,b,c

stud = Student ('Arthur', 90)
print (stud.name)
print (stud.percentage)
x,y,z = stud.show()
print (x,y,z)
```

Out:
```
Arthur
90
10 20 30
```

```
class Parrot:
    species = "bird"                    # class attribute
    def __init__(self, name, age):     # instance attribute
        self.name = name
        self.age = age

# instatiate the Parrot class
blu = Parrot("Blu", 10)
woo = Parrot("woo", 15)

# access the class attributes
print("Blu is a {}".format(blu.species))
print("woo is also a {}".format(woo.species))

# access the instance attributes
print("{} is {} years old".format(blu.name, blu.age))
print("{} is {} years old".format(woo.name, woo.age))
```

(string formatting)

Out:
```
Blu is a bird
woo is also a bird
Blu is 10 years old
woo is 15 years old
```

```
class Person:
    def __init__(self, firstname, lastname, age, country, city):
        self.firstname = firstname
        self.lastname = lastname
        self.age = age
        self.county = country
        self.city = city
```

```
def Person_info(self):
    return f'{self.firstname} {self.lastname}
or  is {self.age} years old. He lives in {self.city},
                                {self.country}
P = Person('Motra', 'Rahul', 96, 'England', 'London')
print(P.Person_info())
```

Out: Motra Rahul is 96 years old. He lives on London, England

String formate & dot formate

(*) Class Parrot:

```
    # instance attributes
    def __init__(self, name, age):
        self.name = name
        self.age = age
    # instance method
    def sing(self, song):
        return "{} sings {}".format(self.name, song)
    def dance(self, z):
        return "{} is now dancing".format(z)
# instantiate the object
blu = parrot("Blu", 10)
# call our instance methods.
print(blu.sing("'Happy'"))
print(blu.dance('Hello'))
```

Out: Blu sings 'Happy'
Hello is now dancing

(*)
```
class Person:
    def __init__(self, firstname='Rohit', lastname='Gill',
                 age=96, country='England', city='London'):
        self.firstname = firstname
        self.lastname = lastname
        self.age = age
        self.country = country
        self.city = city
    def Person_info(self):
        return f'{self.firstname} {self.lastname} is
                {self.age} years old. He lives on
                {self.country}, {self.city}'
P1 = Person()
print(P1.person_info())
P2 = Person('Ben', 'Doe', 30, 'Finland', 'Tampere')
print(P2.person_info())
```

Out: Rohit Gill is 96 years old. He lives on London, England.

Ben Doe is 30 years old. He lives on Tampere, Finland

(*) Class Person:

```
    def __init__(self, firstname='Rahul', lastname=
                 ='Kohli', age=96, country='England',
                 city='London'):
        self.firstname = Rahul
        self.lastname = Kohli
        self.age = 96
```

```
        self.country = country
        self.city = city
        self.skills = []
    def person_info(self):
        return f"{self.firstname} {self.lastname} is
        {self.age} years old. He loves in
        {self.country}, {self.city}"
    def add_skill(self, skill):
        self.skills.append(skill)
P1 = person()
Print(P1.Person_info())
P1.add_skill('Python')
P1.add_skill('MATLAB')
P1.add_skill('R')
P2 = Person('Ben', 'Doe', 30, 'Finland', 'Tampere')
Print(P2.person_info())
Print(P1.skills)
P2.add_skill('Java')
P2.add_skill('Mendix')
P2.add_skill('C++')
Print(P2.skills)
```

Out! Rahul Kohli is 96 years old. He loves in London, England

Ben Doe is 30 ~~Finland~~ years old. He loves in Finland, Tampere

[Python, MATLAB, R]

[Java, Mendix, C++]

① Inheritance

It is a process of inheriting the properties the base class into a derived class.

Examples -

```
Class ClassOne:                          # Base Class
    def func1(self):
        Print('This is parent Class')
Class ClassTwo (ClassOne):               # Derived Class
    def func2 (self):
        Print ('This is Child Class')
obj = ClassTwo()
obj.func1()
obj.func2()
```

Base Class or Parent Class

Class ClassTwo (ClassOne)

Child Class or Derived Class

out: This is parent Class
This is Child Class

(*)

Class Student (person):

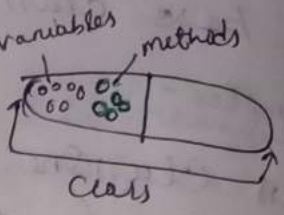
Object Oriented features:
- i) Inheritance
- ii) Encapsulation
- iii) Polymorphism
- iv) Abstraction

(ii) Encapsulation

Class- 15

encapsulation is a method of wrapping data & function into a single entity.

Encapsulation means the internal representation of an object is generally hidden from outside of the object's ~~programe~~ definition.



Two type of excess modifier. - public - excess anywhere
private - limited

Need of Encapsulation

It provides security by the

(*) Example -

```
Class Computer:
    def __init__(self):                    # It private attribute
        self.__maxprice = 900              # self.y = 900 public attribut
    def sell(self):
        print("selling price: {}".format(self.maxpr
    def setMaxprice(self, price):
        self.__maxprice = price

C = Computer()
C.sell()

#c.__maxprice = 1000
# c.setMaxprice(1000)
c.sell()
```

# C.__maxprice
C.setMaxprice(2000)
C.sell()

#

-: → Private attribute
x>90 → Public "

out: Selling Price: 900
Selling Price: 900
Selling Price: 2000

*

Class Employee:

def __init__(self, name, salary):
self.name = name
self.__salary = salary

def show(self):
Print("Name is ", self.name, "and salary is", self.__salary)

(return) → "Name is", 'Bella' and salary is 6000

outside Class

E = Employee("Bella", 6000) (object)

# E.printName()
print(E.name)

print(E.show())

AttributeError: 'Employee' object has no attribute '_salary'

out: Bella
Name is Bella and salary is 6000
None

(iii) Polymorphism

Poly - many
morphism - form

(same class many forms of ( ) but called by object)

(*) Ex-

```
Class Parrot:
    def fly(self):
        Print("Parrot can fly")
    def swim(self):
        Print("parrot can't swim")

Class Penguin:
    def fly(self):
        Print("penguin can't fly")
    def swim(self):
        Print("penguin can swim")

# common interface
def flying_test(bird):
    bird.fly()
# instantiate objects
blu = parrot()
Peggy = penguin()

# passing the object
flying_test(blu)
flying_test(Peggy)
```

out: Parrot can fly
penguin can't ~~swim~~ fly

```
class Circle:
    Pi = 3.14
    def __init__(self, radius):
        self.radius = radius
    def calculate_area(self):
        print("Area of circle:", self.Pi * self.radius
                                 * self.radius)
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width
    def calculate_area(self):
        print("Area of Rectangle:", self.length *
                                    self.width)
cir = Circle(9)
rect = Rectangle(9,6)
cir.calculate_area()
rect.calculate_area()
```

out: Area of circle: 254.34
Area of Rectangle : 54

Types of Inheritance

1) Single Inheritance — one parent class & one child class
2) Multiple "
3) Multilevel "
4) Hierarchical "
5) Hybrid "

1) **Single Inheritance:** One parent class & one child class

```
# Base class
class Vehicle:
    def Vehicle_info(self):
        Print('Inside Vehicle class')

# Derived class
class Car(Vehicle):
    def car_info(self):
        Print('Inside Car class')

# Create object of car
Car = Car()
# acces 'vehicle' info using car object
Car.Vehicle_info()
Car.car_info()
```

Out: Inside Vehicle class
Inside Car class

2) Multiple Inheritance:

We have more than parent class but single child class

```
* Class Person:
    def person_info(self, name, age):
        Print('Inside person class')
        Print('Name:', name, 'Age:', age)
  Class Company:
    def company_info(self, company_name, location):
        Print('Inside Company class')
        Print('Name:', Company name, 'location:', location)
# Derived class
  Class Employee(Person, Company):
    def Employee_info(self, salary, skill):
        Print('Inside Employee class')
        Print('Salary:', salary, 'skill:', skill)
    def Person_info(self, name, age):
        Print('Inside Employee class')
        Print('Name:', name, 'Age:', age)

emp = Employee()
emp.person_info('Rahul', 33)
emp.company_info('Google', 'Atlanta')
emp.Employee_info(19000, 'Machine Learning')
```

out: Inside Person Class
Name: Rahul Age: 33
Inside Company class
Name: Google location: Atlanta
Inside Employee class
Salary: 19000 skill: Machine Learning

iii) Multilevel Inheritance
Base Class Derive class Derive class 2
3 Gen - Grand parent, Parent, Child
level 1 level 2 level 3

```
* # Base Class
  Class Vehicle:
      def Vehicle_info(self):
          Print('Inside Vehicle Class')

  # Parent Class
  Class Car(vehicle):
      def car_info(self):
          Print('Inside SportsCar Class')

  # Child Class
  Class SportsCar(car):
      def sports_car_info(self):
          Print('Inside sportscar class')
  # Create object of sportscar
  S_car = SportsCar()
  # access Vehicle's and Car info using SportCar object
  S_car.vehicle_info()
  S_car.car_info()
  S_car.sports_car_info()
```

Out:-
```
Inside Vehicle Class
Inside Car Class
Inside SportsCar class
```

```
class Animal:                          # grandparent class
    def eat(self):
        Print('Eating...')
class Dog(Animal):                     # Parent class
    def bark(self):
        Print('Barking...')
class BabyDog(Dog):                    # Child class
    def weep(self):
        Print('Weeping...')
    def bark(self):
        Print('Inside BabyDog class')
d = BabyDog()
d.eat()
d.bark()
d.weep()
```

Out:- Eating...
Inside BabyDog class
Weeping...

iv) Hierarchical Inheritance :-

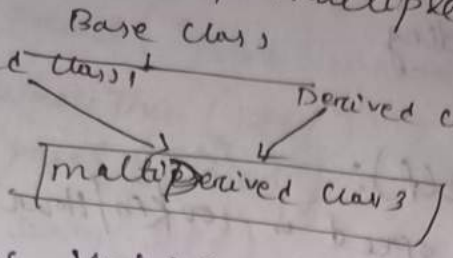
There will be more the Child Class but single parent class.

```
Class Vehicle :
    def info (self):
        Print ("This is Vehicle")

Class Car (Vehicle) :
    def car_info. (self, name):
        Print ("Car name is :", name)

Class Truck (Vehicle):
    def truck_info (self, name):
        print ("Truck name is:", name)

obj 1 = Car ( )
obj 1. info ( )
obj 1. car_info ('BMW')

obj 2 = Truck ( )
obj 2. info ( )
obj 2. truck_info ('Ford')
```

out : This is Vehicle
Car Name is : BMW
This is Vehicle
Truck name is : Ford

v) Hybrid Inheritance:

Combination of multiple inheritance




```
Class Vehicle:
    def vehicle_info(self):
        Print("Inside Vehicle class")
Class Car(Vehicle):
    def car_info(self):
        Print("Inside Car class")
Class Truck(vehicle):
    def truck_info(self):
        Print("Inside Truck class")
# Sports car can inherits properties of vehicle & car
Class SportsCar(car, vehicle):
    def sports_car_info(self):
        print("Inside SportsCar class")
# Create Object
s_car = SportsCar()
s_car.vehicle_info()
s_car.car_info()
s_car.sports_car_info()
```

Out: Inside Vehicle Class
Inside car Class
Inside SportsCar Class

Class-17 (Date - 25/03/2023)

Method Overriding

```
Class Vehicle:
    def max_speed(self):
        Print("max speed is 100 km/Hour")
Class Car(Vehicle):
    # overriden the implementation of vehicle class
    def max_speed(self):
        Print("max speed is 200 km/Hour")
# Creating Object of Car class.
Car = Car()
Car.max_speed()
```

out: max speed is 200 km/Hour

```
Class Parent:  # define parent class
    def myMethod(self):
        Print('Calling parent method')
Class Child(Parent):  # define child class
    def myMethod(self):
        Print('Calling child method')

C = Parent()  # Instance of Child
C.myMethod    # child calls overridden method
```

out: Calling Child method

```
class Bird:    # Parent class
    def __init__(self):
        print("Bird is ready")
    def whoisThis(self):
        print("Bird")
    def swim(self):
        print("swim faster")
# Child class
class Penguin(Bird):
    def __init__(self):
        # call super() function to run the __init__()
          method of the parent class using the
        super().__init__()
        print("Penguin is ready")
    def whoisThis(self):
        print("Penguin")
    def run(self):
        print("Run faster")
Peggy = Penguin()
Peggy.whoisThis()
Peggy.swim()
Peggy.run()
issubclass(Penguin, Bird)
isinstance(peggy, Bird)
```

Out:
```
Bird is ready
Penguin is ready
Penguin
swim faster
Run faster
True
```

(*) Class Point:

```
def __init__ (self, x=0, y=0):
    self.x = x
    self.y = y
P1 = Point (1,2)
P2 = Point (2,3)
print (P1 + P2)
```

out :- TypeError: unsupported operand types

(*) Class Book:

```
def __init__ (self, pages):
    self.pages = pages
def __add__ (self, other):
    print (self.pages)
    print (other.pages)
    return self.pages + other.pages
b3 = Book(200)
b1 = Book(150)
b2 = Book(100)
# b2.__add__ (b1, b3)
print ("Total Number of pages:", b1+b3)
```

Out: 150
200
Total Number of pages: 350

(*) Class Vector:

```
def __init__ (self, a, b):
    self.a = a
    self.b = b
def __str__ (self):
    return 'Vector (%d, %d)' % (self.a, self.b)
    ('vector: {} {}'.format(self.a, self.b)
```

```
def __add__(self, other):
    return Vector(self.a + other.a, self.b + other.b)
v1 = Vector(52, 10)
v2 = Vector(58, -2)
print(v1 + v2)
print(v1)
```

out: Vector(110, 8)
Vector(52, 10)

Various operation for method overloading

Operator	Expression	Internally	Magic method
Addition	P1 + P2	P1.__add__(P2)	__add__(self, other)
Subtraction	P1 - P2	P1.__sub__(P2)	__sub__(self, other)
Multiplication	P1 * P2	P1.__mul__(P2)	__mul__(self, other)
Power	P1 ** P2	P1.__pow__(P2)	__pow__(self, other)
Increment	P1 += P2	P1.__iadd__(P2)	__iadd__(self, other)
Decrement	P1 -= P2	P1.__isub__(P2)	__isub__(self, other)
Product	P1 *= P2	P1.__imul__(P2)	__imul__(self, other)
Division	P1 /= P2	P1.__idiv__(P2)	__idiv__(self, other)
Modulus	P1 %= P2	P1.__imod__(P2)	__imod__(self, other)
Power	P1 **= P2	P1.__ipow__(P2)	__ipow__(self, other)
Division	P1 / P2	P1.__truediv__(P2)	__div__(self, other)
Floor Division	P1 // P2	P1.__floordiv__(P2)	__floordiv__(self, other)
Remainder (modulo)	P1 % P2	P1.__mod__(P2)	__mod__(self, other)
Bitwise Left shift	P1 << P2	P1.__lshift__(P2)	__lshift__(self, other)
Bitwise Right shift	P1 >> P2	P1.__rshift__(P2)	__rshift__(self, other)
Bitwise AND	P1 & P2	P1.__and__(P2)	__and__(self, other)
Bitwise OR	P1 | P2	P1.__or__(P2)	__or__(self, other)
Bitwise XOR	P2 ^ P2	P1.__xor__(P2)	__xor__(self, other)
Bitwise NOT	~P1	P1.__invert__()	__invert__(self)

```
Class Point:
    def __init__(P, x=0, y=0):
        P.x = x
        P.y = y
    def __str__(P):
        return "{0},{1}".format(self.x, self.y)
    def __lt__(P, other):
        P_mag = (P.x ** 2) + (P.y ** 2)
        other_mag = (other.x ** 2) + (other.y ** 2)
        return P_mag < other_mag
P1 = Point(1,1)
P2 = Point(-2,-3)
P3 = Point(1,-1)
Print(P1 < P2)
Print(P2 < P3)
Print(P1 < P3)
```

Out: True
False
False

(iv) Abstraction

```
Class InstanceCounter(object):
    Count = 0
    def __init__(self, val):
        self.val = val
        InstanceCounter.count += 1
    def set_val(self, newval):
        self.val = newval
    def get_val(self):
        return self.val
```

```
    def get_count(self):
        return InstanceCounter.count
a = InstaceCounter(5)
b = InstaceCounter(10)
c = InstanceCounter(15)
for obj in (a, b, c):
    print("Value of object: %s" % (obj.get_val()))
    print("count: %s " % (obj.get_count()))
```

out:
```
Value of object : 5
count : 3
Value of object : 10
count : 3
Value of object : 15
count : 3
```

(*)
```
class MyClass(object):
    @classmethod
    def class_1(cls):
        print("class method 1")
    def class_2(self):
        print("self/Instance method 1")
print("calling the class 'my class' directly without an Instance:")
MyClass.class_1()
#MyClass.class_2()
print("\n Calling the instance 'MyClass()':")
MyClass().class_1()
MyClass().class_2()
```

if I have to call both of them then use 2nd function also @classmethod

out:

Calling the class 'MyClass' directly without an instance:
Class method 1

Calling the instance 'MyClass()':
Class method 1
self/Instance method 1

```
class MyClass(object):
    count = 0
    def __init__(self, val):
        self.val = val
        MyClass.count += 1
    def set_val(self, newval):
        self.val = newval
    def get_val(self):
        return self.val
    @classmethod
    def get_count(cls):
        return cls.count
object_1 = MyClass(10)
Print("\nValue of object : %s" %. object_1.get_val())
Print(MyClass.get_count())

object_2 = MyClass(20)
Print("\nvalue of object : %s" %. object_2.get_val())
Print(MyClass.get_count())
```

```
object_3 = MyClass(40)
print("\n Value of object : %s" %. object_3.get_val())
print(MyClass.get_count())
```

out:
Value of Object : 10
1
value of object : 20
2
value of object : 40
3

```
Class MyClass(object):
    count = 0
    def __init__(self, val):
        self.val = self.filterint(val)
        MyClass.count += 1
    @staticmethod
    def filterint(value):
        if not isinstance(value, int):
            print("Entered value is not an INT,
                  value set to 0")
            return 0
        else:
            return value

a = MyClass(5)
b = MyClass(10)
c = MyClass(15)
print(a.val)
print(b.val)
print(c.val)
print(a.filterint(100))
```

out:
5
10
15
100

Dt 26/03/2023 Class - 18

```
import abc
Class My_ABC_Class(object):
    __metaclass__ = abc.ABCMeta
    @abc.abstractmethod
    def set_val(self, val):
        return
    @abc.abstractmethod
    def get_val(self):
        return
Class MyClass(My_ABC_Class):
    def set_val(self, input):
        self.val = input
    def get_val(self):
        Print("\n Calling the get_val() method")
        Print("I'm part of the Abstract method define in My_ABC_Class()")
        return self.val
    def hello(self):
        ~~Print("\n Call~~
        Print("\n Calling the hello() method")
        Print('I'm *not* part of the Abstract Method define in My_ABC_Class()")
my_class = MyClass()
my_class.set_val(10)
Print(my_class.get_val())
myclass.hello()
```

out:-
calling the get_val() method
I'm part of the Abstract methods defined in My_ABC_Class()
10
calling the hello() method
I'm *not* part of the Abstract Method defined in My_ABC_Class()

* 4 types of object oriented features
  i) Inheritance ii) encapsulation iii) Polymorphism
  iv) abstraction

* parent, child → single multiple multi level hierarchical hybrid
* entity → class → methods and variables
* polymorphism → many forms → function → implemented in multiple classes
* abstraction → abstract class → abstract methods - implement the abstract methods in the child class

* import abc # Pre-defined module → abstraction

class My_ABC_Class(object):
    __metaclass__ = abc.ABCMeta

* The 'abc' module provide feature to create
* Abstract Base Classes.

* To create an Abstract Base Class, set the '__metaclass__' magic method
* to 'abc.ABCMeta'. This will mark the respective class as an Abstract Base Class.
* Now, in order to specify the methods which are to be enforced on the child classes, i.e. to create Abstract Method we use the decorator @abc.abstractmethod

on the method we need.

* The child class which inherits from an Abstract Base class can implement methods of their own, but * should always * implement the methods defined in the parent ABC class.

Regular expressions :-

text = The person's phone number is 408-555-1234. Call soon!

'Phone' in text

Out : True

Phone not in text

Out : False

```
import re          # regular expressions
pattern = 'Phone'
re.search(patter, text)
```

re - stand for name of the module

Out : <_sre.SRE_Match object; span(13,18), match='Phone'>

```
pattern = "NOT IN TEXT"
re.search(pattern, text)
```

Out : nothing

```
pattern = 'Phone'
match = re.search(pattern.text)
match
```

Out :- <_sre.SRE_Match object; span = (13, 18), match='Phone'>

```
match.span()
```

Out : (13,18) (for starting index)

```
match.start()
```

Out : 13

```
match.end()
```

out : - 18

* text = "my Phone is a Phone new Phone"
match = re.search("Phone", text).
match.span()
out: (3,8)

* matches = re.findall("Phone", text)
matches
out: ['Phone', 'Phone', 'Phone']

* for i in re.finditer("Phone", text):
print(i.span())

out:- (3,8)
(14,29)
(24,29)

for corresponding index we have to take for Loop

6 identifiers

Characters	Description	Example Pattern code	Example match
\d	A digit	file_\d\d	file_25
\w	Alphanumeric	\w-\w\w\w	A-b_1
\s	white space	a\sb\sc	a b c
\D	A non digit	\D\D\D	ABC
\W	Non-alphanumeric	\W\W\W\W\W	*-+=)
\S	Non-white space	\S\S\S\S	Yoyo

* import re
z = re.search('\d\d file_\d\d', '37 file_25')
z
out:- <re.Match Object; span=(0,9), match='37file_25'>

* import re
z = re.search('\d\d file\d', '37 file_25')
match = '37 file_2'>
- ('\d file_', '37file_25') - match = '7file_'
- ('\d file_\d', '37file_25') - match - 7file_2

* ('\d abc \d', '4abc5') - match = '4abc5'
('abc \d', '4abc5') - match = 'abc5'
('abc', '4abc5') - match = 'abc'
('c\d', '4abc5') - match = 'c5'

* import re
z = re.search('\d \d$', '37file_25')
out: <re.Match object; span=(7,9), match='25'>

* ('\d$', '37file_25') - match = '5'
* '\w\w$', '37file_25' - match = '25'
* '\w\w$', '37file_25abc7') - match = 'c7'
* '_\d\d', '37file_25abc7') - match = '_25'
* ('\w\w\w', '37file_25abc7')

* ('\d\d\w', '37 file_25abc7') - match- '37f'
* ('\w\w\w', '37 file_25abc7') - match = '37f'
* ('\d\d\w\w\w\d', '37 file_25abc7') - match = '25abc7'
* ('\w\d', '37file_25abc_c7') - match = '37'

* import re
x = re.search('\d\d\d-\d\d an address of \d\d\d', 'John has 789-90 an address of 718 country')

out - <re.Match object; span=(9, 33), match='789-90 an address of 718'>

(*) matches = re.findall("\w", 'Jo@hn mathy has 789-90 an address of 718 Country")

out - 'J'
'o'
'h'
'n'
'm'
'a'
'n'
'y'
'h'
'a'
's'
'7'
'8'
'9'
'9'
'o'
'a'
'n'
'a'
'd'
'd'
'r'
'e'
's'
's'
'o'
'f'
'7'
'1'
'8'
'C'
'o'
'u'
'n'
't'
'r'
'y'

\d - ['7' '8' '9' '9' 'o' '7' '1' '8']

\s - [' ' ' ' ' ' ' ' ' ' ' ' ' ']

(*) import re
x = re.findall ('\s\w\w', 'John many has 789-90 an address of 718 country')

out - [' ma', ' ha', ' 78', 'an', 'ad', 'of', ' 71', 'co']

* \d\w\w - ['789', '718']

Character	Description	Example Pattern code	Example Match
+	Occurs one or more time	Version \w-\w+	Version A-b2
{3}	occurs exactly 3 time	\D{3}	abc
{2,4}	Occurs 2 to 4 times	\d{2,4}	123
{3, }	occurs 3 or more	\w{3, }	anycharacter
*	Occurs zero or more time	A*B*C*	AAACC
?	Once or none	Plural?	Plural

(Quantifiers)

Dt-27/03/2023 Class-19

(*) import re

```
z = "The sun rises in the east at 05:32 AM"
x = re.findall("[^a-zA-Z0-9]",z)
```

[^ - not operator]

out- [' ', ' ', ' ', ' ', ' ', ' ', ' ', ':', ' ']

x= re.findall("[^a-zA-z]", z) - then only no & spaces are come

(*) import re

```
z= re.search('\d*', '1234a561')    # occurs zero or more time
z
```

out- <re.Match object; span=(0,4), match='1234'>

* z= re.search('\s*', '1234a561')
  match=' '

* z= re.search('\d{3}$', '1234a561') → match- 561

* z= re.search('\w{3, }$', '1234a561') - match-'1234a561'
  minimum 3 & maximum anything

* ('\w{3,5}$', '1234a561') - match- 4a561

+

```
import re
z = re.search('\W+', '12345&61')
z
out - match = '12345&61'
```

occurs one or more times

* ('\d+', '12 3456') - match - '12'
* ('[a-z]+', '1234a561') - match - 'a'
* ('[3-4]+', '1234a561') - match - '34'

? (once or none)

z = re.search('\d?', '1234a561') - match - 1
z = re.search('\W?', '1234a561') - match - ' '
z = re.search('\W{3}?', 'Pattern') - match - 'Pat'
z = re.search('\W{3,}?', 'Pattern') - match - Pat
'\W{3,5}?', 'pattern' - Pat

* z = "India is a democratic Country"
  x = re.findall("^.ndia.is", z)
  x
  Out - ['India is']

. indicate one character all things
^ - starts with

- "^...", z - Ind
- "...$", z - try

* z = 'Indddddda'
  x = re.findall('Ind+a', z) out - Indddddda

- z = ①

or operator |

* re.search - (r"man|woman", "This woman was there.")
  out - match - woman
* re search - (r"man|woman", "This woman or man was there.")
  out - woman - # first occurance

The Wildcard Character:-

* re.findall (r".at", "The cat in the hat sat here.")
  out - ['cat', 'hat', 'sat']
* re.findall (r"..at", "The bat went splat.")
  out - [' bat', 'plat']
* re.findall (r'\Stat', "The bat went splat")
  out - ['bat', 'Splat']

starts with and Ends with

re.findall (r'\w$', 'This ends with a number 2s')
out - ['s']

re.findall (r'^\dt', '1345567 is the loneliest number.')
out - ['1345567']

Parentheses for Multiple Options:-

text = 'Hello, would you like some catfish?'
texttwo = "Hello, would you like to take a catnap?"
text three = "Hello, would you have this caterpillar?"

re.search (r'cat (fish|nap|claw)', text)
out- re.Match object; span = (17, 23), match = 'catfish'>

re.search (r'cat (fish|nap|claw)', texttwo
out - 'catnap'

* z = "Now the time is 10:00 AM"
  x = re.findall ["[A-M]", z) out :- 'A', 'M'
* z = "Now the time is + 10:90 AM"
  x = re.findall ("[+]", z)
* z = "Now the time is 10:90 AM"
  x = re.findall ("[0-6]", z) out :- ['1', '0', '0']

Split ( ) & Substitute ( )

```
z = "Spain is a 9 AM country present in Europe"
x = re.split (" ", z)
x
out - 'Spain', 'is', 'a', '9', 'AM', 'country', 'present', 'in', 'Europe']
x = re.split ("e", z)
x
out - [Spain is a 9 AM country pr', 's', 'nt in Europ', '']
```

(*) z = Spain is a 9 AM country present in Europe"

```
x = re.split ("s", z, 2)
x
out: ['Spain', 'is', 'a 9 AM country present in Europe']
```

(*) z = "Spain is a 9AM country present in Europe"

```
x = re.sub ('a', 'm', z)
x
out - 'Spmin is m 9 AM country present in Europe'
(*)x = re.sub (" ", "p", z)
x
out = 'Spainispap9pAMpcountryppresentpinpEurope'
*x=re.sub ("\s", "|", z, 3)
out - 'Spain|is|a|9 AM country present in Europe'
```

Class-20 Dt-29/03/2023

Collection Module :-

```
from collections import counter
# syntax 1
import re
import collections
Collection. counter ( )
#syntax 2
import collections as c
c. counter ( )
# syntax 3
from collection import counter
Coutter ( )
or
from collection import *
counter ( )
list ( )
```

Combination of function → module
combination of modules → Library
Application → Logen → sign up / sign in
Authentication → verify (Password)

Counter() with lists

lst = [1,2,2,2,2, 3,3,3,1,2,1,12,3,2,32,1,21,1,223,1]

z = Counter(lst)

z

out: Counter({1:6, 2:6, 3:4, 12:1, 32:1, 21:1, 223:1})

* Counter('aabsbsbsbhshhbbishs')

out: Counter({'a':2, 'b':7, 's':6, 'h':3})

* S = "How many times does each word show up in this sentence word times each each word"

words = s.split()

Counter(words)

out: Counter({'How': 1
'many': 1
'times': 2
'does': 1
'each': 3
'word': 3
'show': 1
'up': 1
'in': 1
'this': 1
'sentence': 1})

c = Count(words)

C.most_common()

[('each', 3),
('word', 3)
('times', 2)
('How', 1)
('many', 1)
('does', 1)
('show', 1)
('up', 1)
('in', 1)
('this', 1)
('sentence', 1)]

c = counter(words)

c.most_common()

sum(c.values())

list(c)

c.items()

c = counter(words)

c.most_common()

sum(c.values())

out: dict_items([('How', 1), ('many', 1), ('times', 2), ('does', 1), ('each', 3), ('word', 3), ('show', 1), ('up', 1), ('in', 1), ('this', 1), ('sentence', 1)])

defaultdict -

```
from collections import defaultdict
d = {}
d['one']        out: KeyError - 'one'
d = defaultdict(object)
d['one'] = 10
out - <object at 0x2166e276bf0>
for item in d:
    Print(item)   out - one
d = defaultdict(lambda: 20)
d['12']          d['14']
out: 20          out: 20
```

any value is come only 20 because it is defaultdict

namedtuple:-

```
t = (12, 13, 14)
t[0]
out-12
from collections import namedtuple
Dog = namedtuple('CAT', ['age', 'breed', 'name'])
sam = Dog(age=2, breed='Lab', name='Sammy')
frank = Dog(age=2, breed='shephard', name='Frankie')
sam
out - CAT(age=2, breed='Lab', name='Sammy')
sam.age
out - 2
Sam.breed
'Lab'
```

Timing your code -

* def func_one(n):
    '''
    Given a number n, returns a list of string integers
    ['0', '1', '2', ... 'n']
    '''
    return [str(num) for num in range(n)]

* [str(num) for num in range(n)]

```
z = []
for num in range(n):
    z.append(str(num))
```

* func_one(10)

out: ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']

* def func_two(n):
    '''
    Given a number n, returns a list of string integers
    ['0', '1', '2', ... 'n']
    '''
    return list(map(str, range(n)))

func_two(10))

out- ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']

* import time

```
# step 1 - Get start time
start_time = time.time()
# step 2: Run your code you want to time
result = func_one(1000000)
# step 3: Calculate total time elapsed
end_time = time.time() - start_time
```

integers

```
import time
z = time.time()
z
out :- 1630093799.6347192
```

It's our current time

```
* end - time
out - 0.14962792396941
```

Timeit Module :-

```
import timeit
* timeit.timeit(stmt, setup, number=100000)
out - 0.13553508300003614
* setup = '''
def func_one(n):
    return [str(num) for num in range(n)]
'''
stmt = 'func_one(5)'
```

integers

Useful Math functions :-

```
import math
* help math
```

```
import time
help time
```

```
import timeit
help timeit
```

Rounding Numbers

```
Value = 4.35
* math.floor(value)
out- 4
* math.ceil(value)
out- 5
```

```
* round(value,1)
out - 4.0
```

Mathematical Constants

* math.pi
  out- 3.141592653589793
* math.e
  out- 2.718281828459045
* math.log(10)
  out- 2.302585092994046
* math.log(100,10)
  out- 2.0

Trigonometrics Functions

* math.sin(10)
  out:- 0.5440211108893698.
* math.degrees(pi/2)
  out- 90.0
* math.radians(180)
  out:- 3.141592653589793
* import random
  random.randint(0,100)
  out- 3
  always run and random value
* random.seed(101)
  random.randint(0,100)
  out- 74
  once came then it will not change
* mylist = list(range(0,20))
  my list
  out [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]