Python

* Variables -

- Containers for storing data values.
- eg.- name = "Alice"

* Datatypes -

int - 10, -5
float - 3.14, -0.5
str - "hello"
bool - True, False
list - [1,2,3]
tuple - (2,3,4)
dict - {"a": 1}

* Typecasting (Type Conversion) -

```
#Str to int
x = int ("10")    #10

# float to int
z = int (3.9)    # 3 (truncates, not rounds)

# list to string
lst = list ("abc")  # ['a', 'b', 'c']
```

* type() -

```
print (type ("hello"))    #str
```

* String -

- name = "Alice"

  Class = 'X'

  message = """ This is

  a multiline

  string."""

* String Indexing and Slicing -

text = "Python"

text [0] # 'P'

text [-1] # 'n'

text [0:2] # 'Py'

text [:3] # 'Pyt'

text [3:] # 'hon'

* String Immutability -

- String cannot be changed after creation

- text [0] = 'j' # Error

* String methods -

- Str. lower() - Convert to lowercase
- Str. upper() - " " uppercase
- Str. strip ( ) - Removes leading/trailing space
- Str. replace (old, new) - Replace substring
- Str. split (sep) - Split string into list
- Str. join (list) - join ~~list~~ ~~string~~ list into string
- Str. find (sub) - Returns ~~first~~ index of first Occurance.
- Str. startwith (prefix) - Counts Occurance of substring - Str. count (sub) •
- Str. endwith (suffix) - Check if string starts with value.
  - " " " ends " " .

- Str.isdigit() - check all chars are digits
- Str.isalpha() - " " " letters
- Str.isalnum() - " " " letters / digits.

* String formatting (f-strings) :

```
name = "Alice"
age = 30
f"Hello, {name}. You are {age} years old."
# 'Hello, Alice. You are 30 years old.'
```

* Strings Methods with Examples -

- Str.lower() -

```
'Hello'.lower()        # 'hello'
'Python123'.lower()    # 'python123'
```

- Str.upper() -

```
'hello'.upper()        # 'HELLO'
'Python'.upper()       # 'PYTHON'
```

- str.split(sep) -

```
"one, two, three".split(",")   # ['one', 'two', 'three']
'Hello World'.split()          # ['Hello', 'World']
```

- Str.join(list) -

```
",".join(["a", "b", "c"])      # 'a,b,c'
" ".join(['Python', 'Rocks'])  # 'Python Rocks'
```

- Str.strip() - ~~Removes~~

```
"  hello  ".strip()            # hello
'\n Python\n'.strip()          # Python
```

- Str.replace(old, new) -

```
"apple pie".replace("pie", "juice")   # apple juice
"aaabbb".replace("a", "b")            # bbbbbb
```

- Str. find (sub) –

  "notebook". find ("book") # 4

  "banana". find ("a") # 1

- Str. count (sub) –

  "banana". count ("a") # 3

  "hello world". count ("l") #3

- Str. Startswith (prefix) –

  "Python". Startswith ("Py") # True

  "Coding". Startswith ("code") # false

- Str. endswith (suffix) –

  "filename.txt". endswith (".txt") #True

  "image.png". endswith (".jpg") # False

- Str. isdigit () –

  "12345". isdigit () # True

  "12a23". isdigit() # False

- Str. isalpha () –

  "Python". isalpha () # True

  "Python3". isalpha() # false

- Str. isalnum () –

  "abc 123". isalnum () # True

  "abc@123". isalnum() # False

* Operators in Python –

1. Arithmetic Operators

```
a = 10
b = 5
print (a +b)     # 15
print (a-b)      # 5
print (a * b)    # 50
print (a /b)     # 2.0
print (a//b)     # 2 → floor division
print (a %b)     # 0 → Modulus (remainder)
print (a **b)    # 100000 → Exponentiation
```

10 % 3 ⇒ 3.33
10//3 ⇒ 3

9 % 2 ⇒ 4.5
9 //2 ⇒ 4

2. Comparison operators -

```
a = 10
b = 5

print (a == b)      # false
print (a != b)      # True
print ( a > b)      # True
print (a < b)       # False
print (a >= b)      # True
print (a <= b)      # False
```

3. Logical operators -

```
X = True
Y = False

print (x and y)     # False
print ( x and y)    # True
print (not x)       # False
```

4. Bitwise operators -

```
a = 5     # binary = 0101
b = 3     # binary = 0011
result1 = a & b     # 0001 => 1
print (result1)
result2 = a | b
print (result2)     # 0111 → 7
```

5. Assignment Operators -

```
a = 10
a += 5      # a = a + 5
print (a)   # 15
```

6. Membership Operators -

```
lst = [1,2,3]
X = 2
print (x in lst)        # True
print (4 not in lst)    # True
```

Identity operators -

```
a = [1,2]
b = [1,2]
c = a
print (a is b)      # False (Diff^n objects in memory)
print (a is c)      # True (Same object)
print (a is not b)  # True (not the same object)
```

* Taking input from user

```
name = input ("Enter your name:")
print ("Hello", name)
```

• Type Conversion -

```
age = int (input (" Enter your age : "))
price = float (input (" Enter the price : "))
```

* Operator Precedence -

Python follows - Parenthesis, Exponents, Multiplication / Division, Addition / Substraction (PEMDAS)

example :

```
result = 10 + 2 * 3
print (result) # 16

res = (10 + 2) * 3
print(res) # 36

res = 2 ** 3 ** 2    # Right to left exponentiation
print (res)  # 512
```

* If - else Conditionals :

ex -

```
age = 25
if age < 18:
    Category = "Minor"
else if age < 65:
    Category = "Adult"
else :
    Category = "Senior Citizen"
print ("Category", Category)
```

Example : Applying Conditions on Pandas Dataframe

```
import pandas as pd
data = { 'Name' : ['Alice', 'Bob', 'Charlie'], 'Score' :
        [65, 40, 75] }
df = pd. Dataframe (data)
df ['Result'] = df ['Score']. apply (lamdo x : 'Pass'
        if x ≥ 50 else 'fail')
print (df)
```

* Match Case in Python -

```
def http_status(code):
    match code:
        case 200:
            return "OK"
        case 400:
            return "Bad Request"
        case 500:
            return "Internal Server Error"
        case _:
            return "Unknown Status"
```

case _ → act as default case

```
print(http_status(200))  # OK
print(http_status(400))  # ~~Not found~~
                         Bad Request
```

* String formatting in Python -

i) using .format() -

```
# Positional arguments
print("{0} loves {1}".format("Alice", "Python"))
#O/P: Alice loves Python

# keyword arguments
print("{name} is {age} years old".format(name="Bob", age=30))
# O/P: Bob is 30 years old.
```

ii) using f-strings -

```
name = "John"
printf(f"my name is {name}")
#O/P- My name is John

a,b = 10,5
print(f"The sum is {a+b}")
# O/P- The sum is 15
```

• Formating Numbers –

Use → :.2f , :.3f → for decimal control.

Example:

```
pi = 3.14159
print(f"Rounded Pi : {pi :.2f} ")
#O/P: Rounded Pi : 3.14
```

```
Value = 0.875
print(f" Completion : {Value :.0%} ")
# O/P: Completion : 88%
```

: → applying formatting
.0 → Show 0 decimal places
% → Convert the no. to % (* by 100 and add %)

- 0.875 X 100 = 87.5%
- Rounded to 0 decimal places → 88%

• Padding & Alignment –

Use → <, >, ^ with width to align content.

Example:

```
#center align
print(f"{Python :^ 10 }")
#O/P:  '   Python   '

# Right align with width
print( f"{ 'Code' : >8}")
#O/P :  '    Code'
```

* Functions in Python

- is a block of reusable code that performs a specific task.

- **Syntax:**

```
def fun-name(parameters):
    # Code block
    return result
```

- **example:**

```
def greet(name):
    return f"hello, {name}!"
print(greet("Gaurav"))
#O/P - Hello, Gaurav!
```

* Loops

- **for loops-**

→ fruits = ["apple", "banana", "cherry"]

```
for fruit in fruits:
    print(fruit)
```

O/P - apple
banana
cherry

→ for i in range(1,6):

```
    print(i)
```

O/P - 1
2
3
4
5
6

→ for i in range(10, 0, -2):

```
    print(i)
```

O/P - 10
8
6
4
2

- while loop -

```
count = 1
while count <= 3:
    print (count)
    count += 1
```

O/P - 1
2
3

- Break statment -

```
for i in range (5):
    if i == 3:
        break
    print (i)
```

O/P - 0
1
2

- Continue statment -

```
for i in range (5):
    if i == 2:
        continue
    print (i)
```

O/P - 0
1
3
4

- else with loop -

```
for i in range (3):
    print (i)
else:
    print ("loop finished")
```

O/P - 0
1
2
loop finished

- Nested loop -

```
for i in range (1,3):
    for j in range (1,4):
        print (f"{i} x {j} = {i*j}")
```

O/P - 1 x 1 = 1
1 x 2 = 2
1 x 3 = 3
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6

- looping through Dictionary -

→ person = {"name" : "Gaurav", "age" : 21}

for key, value in person.items():

print (f"{key} : {value}")

o/p - ~~Gaurav~~
Name : Gaurav
age : 21

- looping through string -

→ for char in "Python":

print (char)

o/p -
P
Y
T
H
O
N

- List comprehension -

→ square = [x * x for x in range (1, 6)]
print (square)

o/p -
[1, 4, 9, 16, 25]

* Lists

- Ordered, mutable (changable), and iterable collection of items.
- List can hold any data type: number, list, even other list.

ex- fruits = ["apple", "banana", "cherry"]
0 1 2
-3 -2 -1

- Creating Lists -

numbers = [1, 2, 3, 4]

mixed = [1, 'hello', 3.5, True]

nested = [[1, 2], [3, 4]]

print (~~numbers~~ mixed) # [1, 'hello', 3.5, True]

- Accessing items in a list -

```
fruits = ["apple", "banana", "cherry"]
print (fruits[0]) # apple
print (fruits[-1]) # cherry
```

- Modifying list elements -

```
fruits[1] = "mango"
print (fruits) # ['apple', 'mango', 'cherry']
```

- List slicing -

```
print (fruits[0:2]) # ['apple', 'mango']
print (fruits[:]) # ['apple', 'mango', 'cherry']
```

- <u>List Methods</u> -

- append () - add element to the end

```
fruits.append ("orange")
print (fruits) # ['apple', 'mango', 'cherry', 'orange']
```

- insert (index, value) - Insert at a position

```
fruits.insert (1, "kiwi")
print (fruits) # ['apple', 'kiwi', 'mango', 'cherry', 'orange']
```

- extend ( ) - add multiple elements

```
fruits.extend (["grape", "melon"])
print (fruits) # ['apple', 'kiwi', 'mango', 'cherry', 'orange', 'grape',
                 'melon']
```

- remove () - Remove specific items

```
fruits.remove ("mango")
print (fruits) # ['apple', 'kiwi', 'cherry', 'orange', 'grape', 'melon']
```

~~•Jumovt~~

• pop () - Remove last item (or index)

```
fruits.pop()
print (fruits) # ['appli', 'kiwi', 'chuny', 'orangi', 'grapi']

fruits.pop (1)
print(fruits) # ['appli', 'chuny', 'orangi', 'grapi']
```

• index (value) - find index of an item

```
print (fruits.index ('orangi') # 2
```

• count () - Count occurances

```
numbers = [1, 2, 2, 3, 4, 2]
print (numbers.count (2)) # 3
```

• sort () - Sort the list in place

```
nums = [4, 2, 9, 1]
nums.sort ()
print (nums) # [1, 2, 4, 9]
nums.sort (reverse = True) # [9, 4, 2, 1]
```

• Sorted () - Sort but return a new list

```
nums = [4, 2, 9, 1]
new_nums = sorted (nums)
print (new_nums) # [1, 2, 4, 9]
```

• ~~Sorted~~ reverse () - reverse the list

```
nums = [1, 2, 3]
nums.reverse ()
print (nums) # [3, 2, 1]
```

- copy() - Make a shallow copy

```
a = [1,2,3]
b = a.copy()
b.append(4)

print("a:", a)    # a : [1,2,3]
print("b:", b)    # b : [1,2,3,4]
```

- clear() - Remove all elements

```
nums.clear()
print(nums)  # []
```

- Looping through a list -

```
for item in ['pen', 'book', 'Laptop']

print(item)  # pen
             book
             Laptop
```

* Tuple -

- tuple is immutable meaning you can't change, add or remove items after creation

- creating a tuple -

```
fruits = ("apple", "banana", "cherry")
print(fruits) # ('apple', 'banana', 'cherry')
```

- Single element tuple -

```
single = ("hello",)
print(type(single)) # <class 'tuple'>
```

- Accessing tuple items -

```
print(fruit[0]) # apple
print(fruit[-1]) # cherry
```

- Looping through tuples -

```
for item in fruits:
    print(item) # apple
                banana
                cherry
```

- Tuple slicing -

```
print(fruits[1:]) # ('banana', 'cherry')
```

* Tuple Methods -

- count() -

```
numbers = (1, 2, 3, 2, 2, 4)
print(numbers.count(2)) # 3
```

- index() - find the first index of an item

```
print (numbers.index(3))   # 2
```

- Concatination -

```
a = (1,2)
b = (3,4)
c = a+b
print (c)   # (1,2,3,4)
```

- Repetition -

```
repeat = ("Hi",)*3
print (repeat) #  ('Hi', 'Hi', 'Hi')
```

- checking for items Existance -

```
if "banana" in fruits :
        print ("Yes, banana is in tuple")
```

o/p - # Yes, banana is in tuple

- ~~checking~~ Convert list to tuple & vice versa -

```
my_list = [1,2,3]
my_tuple = tuple (my_list)
print (my_tuple)  # (1, 2, 3)
```

```
my_tuple = (4,5,6)
my_list = list (my_tuple)
print (my_list) # (4,5,6)
```

- immutability test -

```
nums = (1,2,3)
#nums [0] = 100   x this will raise an error
# TypeError : 'tuple' object dosnot support items assignment.
```

* Set -

- Collection of unique elements
- Unordered (no - indexing)
- Mutable (can add or remove items)
- Ex - my_set = {1, 2, 3, 2, 1}

  print (my_set) # {1, 2, 3}

  # Duplication are removed

- **Creating Sets -**

  fruits = {"apple", "Banana", "cherry"}

  print (fruits) # {"Banana", "apple", "cherry"}

  # Order may differ - set are unordered

- **Creating an Empty set -**

  empty_set = set()

  print (type (empty_set)) # <class 'set'>

- **Looping through Sets -**

  for fruits in fruits:

  print (fruit) # banana
  apple
  cherry

- set methods —

- add() - add a single element.

```
fruits.add("orange")
print(fruits) # {'banana', 'apple', 'cherry', 'orange'}
```

- update() - add multiple items.

```
fruits.update(['grapes', 'melon'])
print(fruits) # {'banana', 'apple', 'grapes', 'cherry', 'orange', 'melon'}
```

- remove() - Remove item (error if not found)

```
fruits.remove("apple")
print(fruits) # {'banana', 'grapes', 'cherry', 'orange', 'melon'}
# will through keyerror if item not found.
```

- discard() - Remove item (no error if not found)

```
fruits.discard('papaya') # no error
```

- pop() - Remove random element

```
item = fruits.pop()
print(item)      # banana
print(fruits)    # {'grapes', 'cherry', 'orange', 'melon'}
```

- clear() - Empty the set

```
fruits.clear()
print(fruits)    # set()
```

- **Set Operations** –

  - union() – combine two sets (no duplicates)

    a = {1,2,3}

    b = {3,4,5}

    print (a.union(b)) # {1,2,3,4,5}

  - intersection() - common elements

    print (a.intersection(b)) # {3}

  - difference() - items in A not in B.

    print (a.difference(b)) # {1,2}

  - symmetric_difference() - item in either, but not both

    print (a.symmetric_difference(b)) # {1,2,4,5}

- **Set Membership Test** –

  if 3 in a:

    print ('Yes') # Yes

* Dictionary -

- unordered, mutable, indexed collection of key-value pairs
- real life dictionary: you search by word (key) and get its meaning (value).
- ex - student = {

```
"name" : " Gaurav ,
"age" : 20,
"course" : " BCA".
}
```

```
print (student ["name"])   # Gaurav
```

- **Creating a Dictionary -**

```
data = {
    "name": " Alice,
    "marks": 95 ,
    "passed" : True
}
```

```
print (data)   # {'name' : 'Alice' , ' marks' : 95 , 'passed' : True}
```

- **Dictionary Methods -**

- dict.key() → return all keys.

```
data: {'name' : ' Alice' , 'age' : 21}
print (data.keys ())   # dict-keys (['name' , 'age'])
```

- dict.values() → Return all values.

```
print (data.values()) # dict_values (['Alice', 21])
```

- dict.items() → Return list of key-value tuples.

```
print (data.items()) # dict_items ([('name', 'Alice'), ('age', 21)])
```

- dict.get(key) → Safely get value (return None if not found)

```
print (data.get ('age')) # 21
print (data.get ("address")) #None
```

- dict.update() → Update dictionary with new data.

```
data.update ({'age' : 22, "city" : "Lucknow"})
print (data) # {'name' : 'Alice', 'age' : 22, 'City' : 'Lucknow'}
```

- dict.pop(key) → Remove key and remove its value

```
age = data.pop ('age')
print (age) #22
print (data) # {'name' : Alice ; 'city' : 'Lucknow'}
```

- dict.popitem() → Removes the last inserted key-value pair

```
item = data.popitem()
print (item) # ('city', 'Lucknow')
```

- dict.clear() → Remove all items

```
data.clear()
print (data) # {}
```

• dict.copy() → Returns a shallow copy of dictionary.

```
original = {'a': 1, 'b': 2}
copy_dict = original.copy()
print(copy_dict)  # {'a': 1, 'b': 2}
```

• dict.fromkeys(keys, value) → Create dictionary from list of keys.

```
keys = ['name', 'age', 'city']
new_dict = dict.fromkeys(keys, 'unknown')
print(new_dict) # {'name': 'unknown', 'age': 'unknown', 'city': 'unknown'}
```

Looping through Dictionary -

```
Student = {'name': 'Gaurav', 'age': 20}

for key, value in Student.items():
    print(f"{key}: {value}")  # name: Gaurav
                                age: 20
```

Dictionary use cases -

Student Record, JSON Data, Word counter, Configuration settings.

* File handling -

- It allows to create read, write, update and delete files using python
- types of files in Python -
  - → text Files (.txt, .log, etc) - plain human readable
  - → Binary files (.jpg, .png, .exe, etc) - contain data in binary form

- Opening a files -

```
file = open ("example.txt", "r")
```

Mode	Meaning	Description
r	read	Default mode. File must exist
w	write	Creates a new file or truncates existing
a	append	Adds content at the end
x	Create	Creates file, fails if file exists
b	Binary	used for binary files (rb, wb, etc)

- Reading a doc/ file -

a) .read () - Reads entire content

```
file = open ("example.txt", "r")
print (file.read())
file.close()
```

O/P - This is a file.

b) .readline () - Read one line at a time

```
file = open("example.txt", 'r')
print (file.readline ())
file.close ()
```

O/P - this is a file.

c) .readlines() - Reads all lines as a list

```
file = open ("example.txt", 'r')
lines = file.readlines()
print (lines)
file.close()
```

O/P - ['this is a sample file\n', 'Second line\n

- write to a file - .write()

a) overwrite using 'w' mode:

```
file = open ("example.txt", "w")
file.write ("New content here!")
file.close ()
```

old content in file is replaced with -
New content here!

b) Append using 'a' mode:

```
file = open ("example.txt", "a")
file.write ("\n Another new line")
file.close ()
```

Adds new line at end -
New content here!
Another new line

- Best practice - with statement -

```
with open ("example.txt", 'r') as file:
    content = file.read()
    print (content)
```

- File Pointer Methods

Methods	Description
file.tell()	Returns current file pointer position
file.seek(offset)	change the file pointer position

Ex -
```
with open ("example.txt", 'r') as f:
    print (f.tell())    # 0
    f.read (5)
    print(f.tell())     # 5
    f.seek (0)
    print (f.read (4))  # Nun
```

- Handling file Exception -

```
try :
    with open ("notfound.txt", 'r') as f:
        data = f.read ()
except FileNotFoundError :
    print ("The file does not exists")
```
o/p- the file does not exists

- Difference b/w write() & writelines() -

```
lines = ['Line 1\n', 'Line 2\n']
with open ("example.txt"), 'w') as f:
    f.writelines (lines)
```
o/p- Line1
Line2

- Working with Binary files -

```
with open ('image.jpg', 'rb') as f:
    binary_data = f.read (10)
    print (binary_data)
```

O/P - b'\xff\xd8\xff\x00\x10JFIF'

* JSON MODULE:

- json → javascript object notation
- lightweight data-interchange format
- used to store and exchange data b/w server and client (eg. in web APIs)
- import -

```
import json
```

- functions in json module -

  - json.dumps( ) - Convert python object to json string.

```
import json
data = {"name": "Gaurav", "age": 20, "is_student": True}
json_string = json.dumps(data)
print (json_string)
```

O/P - {'name': 'Gaurav', 'age': 20, 'is_student': true}

NOTE: • Python 'True' becomes true in JSON
• Dic^n becomes JSON object

- json.loads() - Convert JSON string to Python object.

```
json_data = '{"Name": "Gaurav", "age": 20, "is_student": true}'
python_data = json.loads(json_data)
print(python_data)
```

O/P- {'name': 'Gaurav', 'age': 20, 'is_student': True}

- json.dump() - write json data to a file

```
data = {"City": "Lucknow", "State": "UP"}
with open("data.json", "w") as file:
    json.dump(data, file, indent=4)
```

O/P- create a ~~json~~ data.json file with.

```
{
    "City": "Lucknow",
    "State": "UP"
}
```

- json.load() - Read JSON data from a file

```
with open("data.json", "r") as file:
    content = json.load(file)
    print(content)
```

O/P- {'City': 'Lucknow', 'State': 'UP'}

- Python → JSON type Conversion Table -

Python	JSON
dict	obj
list, tuple	array
str	string
int, float	number
True	true
False	false
None	Null

- handling Error with Try - Except -

```
json_data = '{"name": "Gaurav", "age": 20}' # Incorrect json

try:
    parsed = json.loads(json_data)
except json.JSONDecodeError as e:
    print("JSON Error:", e)
```

O/P -

JSON Error : Expecting ':' delimiter : Line 1 Column 20 (char 19)

- Real world Example : fetch from API -

```
import json
import urllib.request

url = "https://jsonplaceholder.typicode.com/todos/1"
response = urllib.request.urlopen(url)
data = json.loads(response.read())

print(data)
```

o/p -

{'userId' : 1, 'id' : 1, 'title' : 'delectus aut autem',

'completed' : False}

* Object Oriented Program (OOP) -

- OOP is programming paradim that organizes codes using object and classes.
- It helps in writting clean, modular, and reusable code.

- **4 pillars of OOP in Python -**

Encapsulation	Binding data and methods into a single unit
Abstraction	hiding internal details, showing essentials
Inheritance	Reusing code through class hierarchy
Poly morphism	Same function behaves differently.

- **Class & Object -**

class : Blueprint for creating objects..

object : Instance of a class.

```
class Student :
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def greet(self):
        print(f"hello, my name is {self.name} and I'm {self.age}
                                                years old.")

S1 = Student("Gaurav", 23)
S1.greet()
```

O/P - Hello my name is Gaurav and I'm 23 years old

• Constructor :

Called automatically when object is created.

```
class car:
    def __init__ (self, brand):
        self.brand = brand
        print ("car object created")

car1 = car ("BMW")
print (car1.brand)
```

O/P - car object created
BMW

• Encapsulation (Private Members) :

```
class BankAccount :

    ~~def __get~~
    def __init__ (self, balance):
        self.__balance = balance

    def get_balance (self):
        return self.__balance

acc = BankAccount (5000)
print (acc.get_balance ())
```

O/P - 5000

- **Inheritance** -

```
class Animal:
    def speak(self):
        print("Animal speaks")

class Dog(Animal):
    def bark(self):
        print("Dog barks")
```

```
d = Dog()
d.speak()
d.bark()
```

O/P - Animal speaks
Dog barks

- **Polymorphism (method overriding)** -

```
class Bird:
    def sound(self):
        print("Some bird sound")

class sparrow(Bird):
    def sound(self):
        print("chirp chirp")
```

```
b = sparrow()
b.sound()
```

O/P- chirp chirp

• Abstraction (using ABC method) -

```
from abc import ABC, abstractmethod

class shape (ABC):
    @abstractmethod
    def area (self):
        pass

class Circle (Shape):
    def area (self):
        return 3.14 * 5 * 5

c = Circle ()
print (c.area ())
```

O/P- 78.5.

• Super () keyword -

- used to call parent class method

```
class Parent :
    def show (self):
        print (" This is Parent")

class child (Parent):
    def show (self):
        super (). show()
        print (" This is child")

c = child()
c.show ()
```

O/P-

This is Parent
This is child

- **Class vs Instance Variable -**

```
class Example :
    class_var = " I am class variable"
    def __init__ (self):
        self.inst_var = " I am instance variable"

e1 = Example()
e2 = Example()

e1.class_var = "changed"
print (e1.class_var)
print (e2.class_var)
```

O/P - changed
I am class variable

- **Dunder method** - (__str__ , __len__, etc)

```
class Book :
    def __init__ (self , title):
        self.title = title
    def __str__ (self):
        return f" Book title : {self.title}"

b = Book ( " Python Mastry")
print (b)
```

O/P - Book title: Python Mastry

* List Comprehension :-

- It is a shorter and more readable way to create list in python.
- Syntax: [expression for item in iterable if condition]
- example:

```
numbers = []
for i in range(5):
    numbers.append(i**2)

numbers_lc = [i**2 for i in range(5)]
print(numbers)        # [0, 1, 4, 9, 16]
print(numbers_lc)     # [0, 1, 4, 9, 16]
```

- with condition:

```
evens = [x for x in range(11) if x%2==0]
print(evens)  # [0, 2, 4, 6, 8, 10]
```

- with Else :

```
status = ["Even" if x%2==0 else "odd" for x in range(6)]
print(status) # ['Even', 'odd', 'Even', 'odd', 'Even', 'odd']
```

- Nested List Comprehensions (List inside list):

```
table = [[x*y for y in range(1,4)] for x in range(1,4)]
print(table) # [[1,2,3], [2,4,6], [3,6,9]]
```

- Flatten a 2D lists -

```
matrix = [[1,2], [3,4], [5,6]]
flattened = [num for row in matrix for num in row]
print (flattened)  # [1,2,3,4,5,6]
```

- Using function in list comprehension -

```
def square (n):
    return n*n
result = [square (n) for x in range (1,6)]
print (result)  # [1, 4, 9, 16, 25]
```

- List comprehension with strings -

```
word = "Python"
letters = [ch. upper () for ch in word]
print (letters)  # ['P', 'Y', 'T', 'H', 'O', 'N']
```

*

* Lambda function in Python -

- Lambda function are anonymous (nameless) function in Python.
- mainly used when you need a small, short-term function and don't want to formally define it with def.

- Syntax: `lambda arguments : expression.`

  keyword to define the fun — input parameters (can be multiple) — single expression to be evaluated and return

- Ex - • ~~add~~ f

```
def add (a, b):
    return a+b

add_lambda = lambda a, b : a+b

print (add (3,5))          # 8
print (add_lambda (3,5))   # 8
```

- Ex - Without Arguments -

```
greet = lambda : "Hello, Python"
print (greet())   # Hello, Python
```

- Ex - With Conditional Logic -

```
max_num = (lambda x, y : x if x>y else y
print (max_num (10,20))   # 20
```

* Decorators

- It is a function that takes another function as an argument, add some functionality to it, and return it.
- You can think of it like wrapping a function to give it extra power without changing its original code.
- ex -

→ without decorators -

```
def shout(text):
    return text.upper()

def greet(func):
    message = fun("hello")
    return message + "!!!"

print(greet(shout))
```

O/P - HELLO !!!

→ @decorator-name -

```
def uppercase_dec(func):
    def wrapper():
        result = func()
        return result.upper()
    return wrapper

@uppercase_dec
def say_hello():
    return "hello world"

print(say_hello())
```

O/P - HELLO WORLD