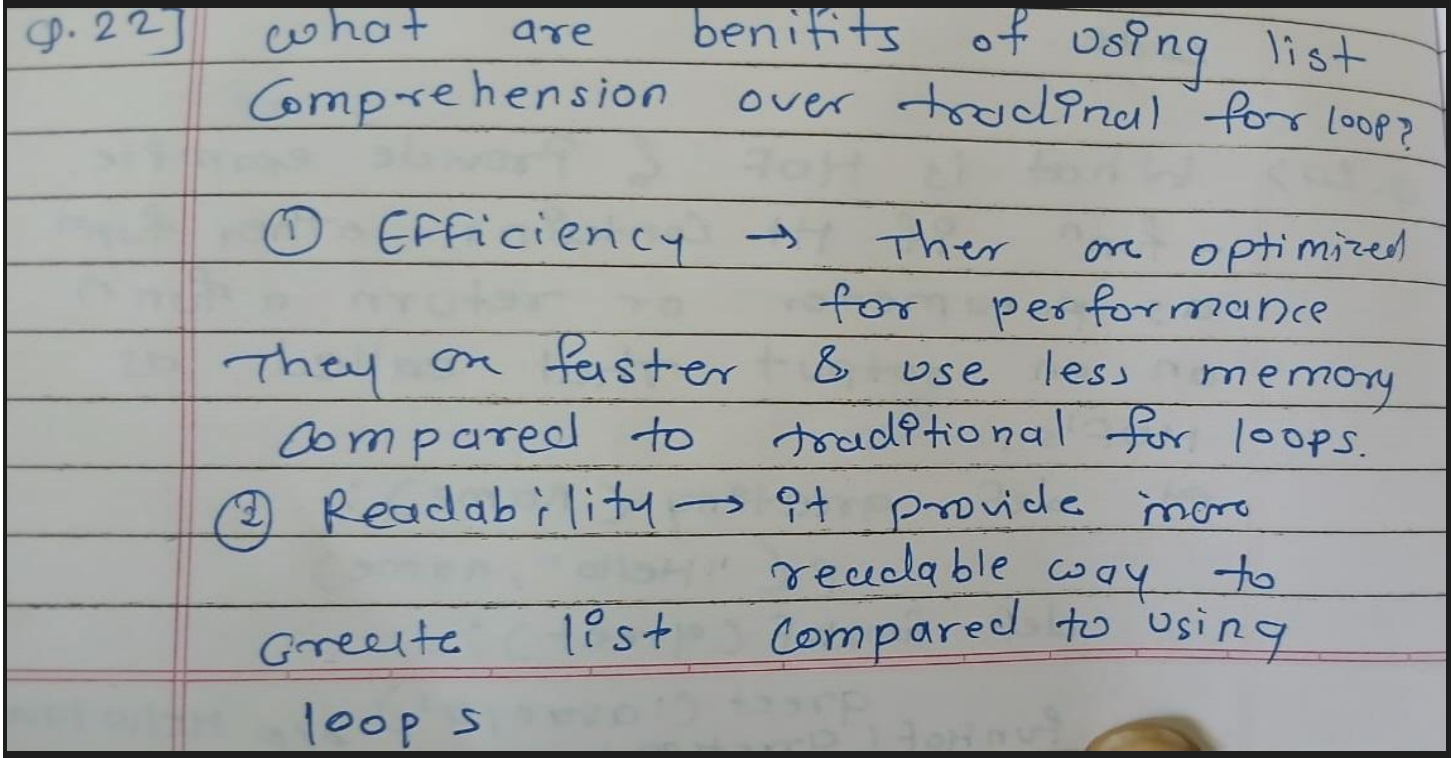
Section A: Theory (10 Questions)

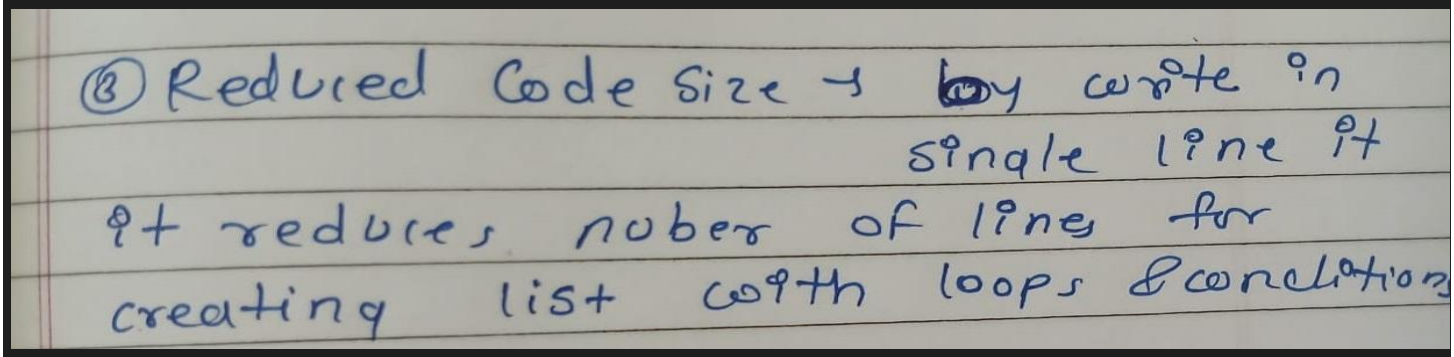
Q1. What are the benefits of using list comprehensions over traditional for loops?



Q.22] what are benifits of using list Comprehension over tradinal for loop?

① Efficiency → Ther are optimized for performance They are faster & use less memory compared to traditional for loops.

② Readability → it provide more readable way to create list compared to using loops



③ Reduced Code Size → by write in single line it it reduces nober of line for creating list with loops & conditions

Q2. Hpw do you handle multiple exception in a single try block? Provide an example.

creating list with loops & conditions

Q.23) How do you handle multiple exceptions in single try block ? provide an example ?

→ we can handle multiple exceptions in single try block by except clouse.

```
try :
        x = 10/0
        y = int("not a number")
except ZeroDivisionError :
        print("not possible by zero")
except ValueError :
        print("not value valid")
```

o/p not possible by zero

each except block handle specific type of exception

you can write multiple exception in tuple

```
except (zer...., Typ...., value):
    & print single warning.
```

Q3. What is File Handling?

Q.24) What is file handling?

- It refers the process of reading from and writing to files stored on a disk.
- It allows us to interact with external files. such binary or text
- Python Provides built in methods to manage files efficiently
- TO work with files we use open function file we use open() which returns file object.
- This object provides read write & maniplate methods to file contents.
- The file must be closed after the completing the operations. with help of close()
or we can use (with statement) which handles automatic file closing.

① opening file we use open() specifying file ~~methods~~ name & mode (r) (w) (a) (rw) (b)

② Reading from file :- with help of read() readline() & readlines() we can read file content.

③ writeing to file :- with help of we can write into file. write() & writelines()

Q4. What are the advantages of using a tuple over a list

Date

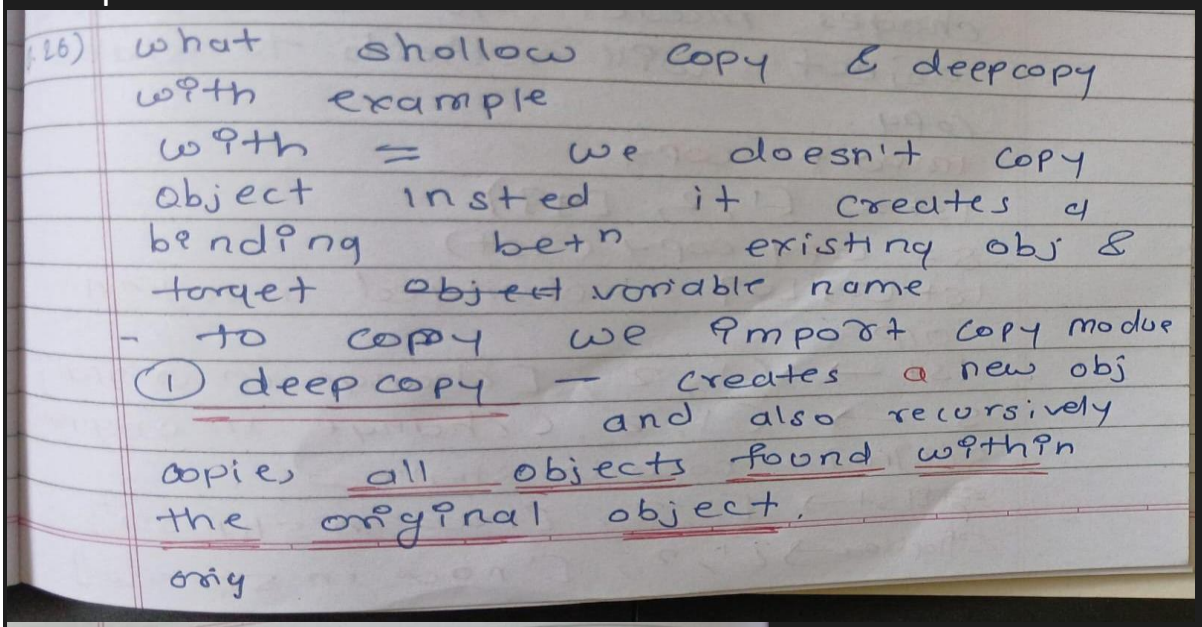
(15) what are the advantages of using tuple over a list.

① Immutability - once they are created, their values cannot be changed, this makes tuple safer we can use tuple in that senario where data doesn't change.

② Performance :- Since tuple are immutable they generally faster than list

③ Data Integrity :- By using tuple you ensure that the data remains unchanged throughout its lifetime which helps to prevent accidental modification & improves readability of code.

Q5. What is difference between shallow copy and deep copy of a list. Provide example.

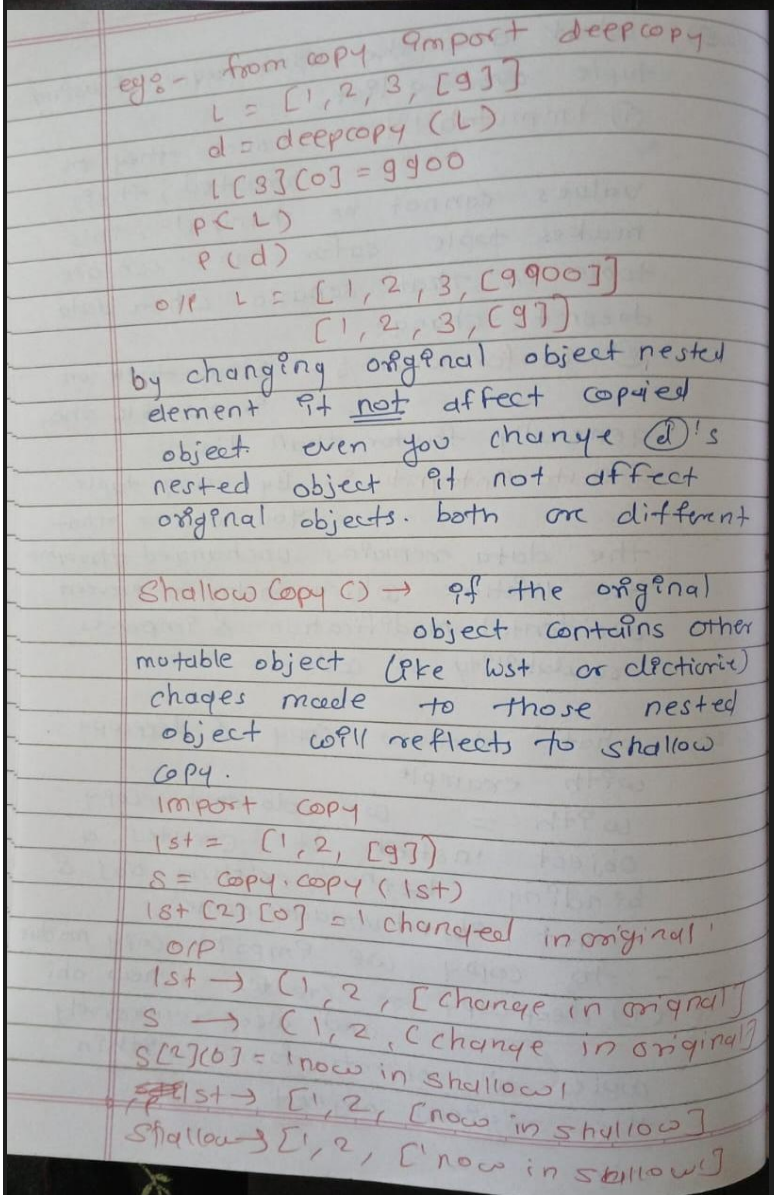

(26) what shallow copy & deepcopy with example

with = we doesn't copy object insted it creates a binding betn existing obj & target object variable name

- to copy we import copy module

(1) deepcopy — creates a new obj and also recursively copies all objects found within the original object.

orig

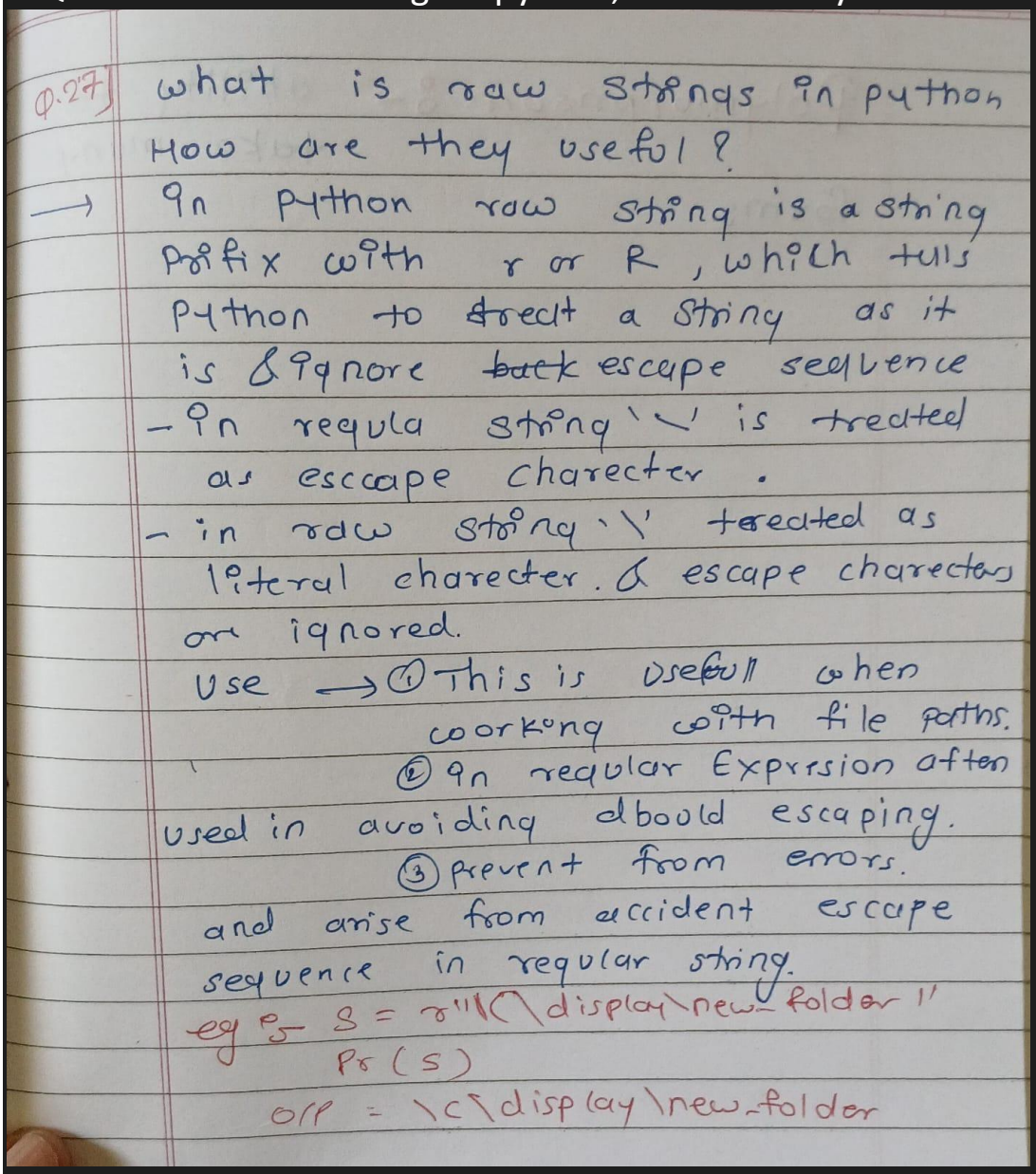

eg:- from copy import deepcopy
L = [1, 2, 3, [9]]
d = deepcopy (L)
L[3][0] = 9900
P(L)
P(d)
o/p L = [1, 2, 3, [9900]]
[1, 2, 3, [9]]

by changing original object nested element it not affect copied object even you change (d)'s nested object it not affect original objects. both are different

Shallow Copy () → if the original object contains other mutable object (like list or dictionrie) chages made to those nested object will reflects to shallow copy.

import copy
lst = [1, 2, [9]]
S = copy.copy (lst)
lst[2][0] = 'changed in original'
o/p
lst → [1, 2, [change in original]
S → [1, 2, [change in original]
S[2][0] = 'now in shallow'
lst → [1, 2, [now in shallow]
Shallow → [1, 2, ['now in shallow']

Q6. What are raw strings in python, How are they useful?



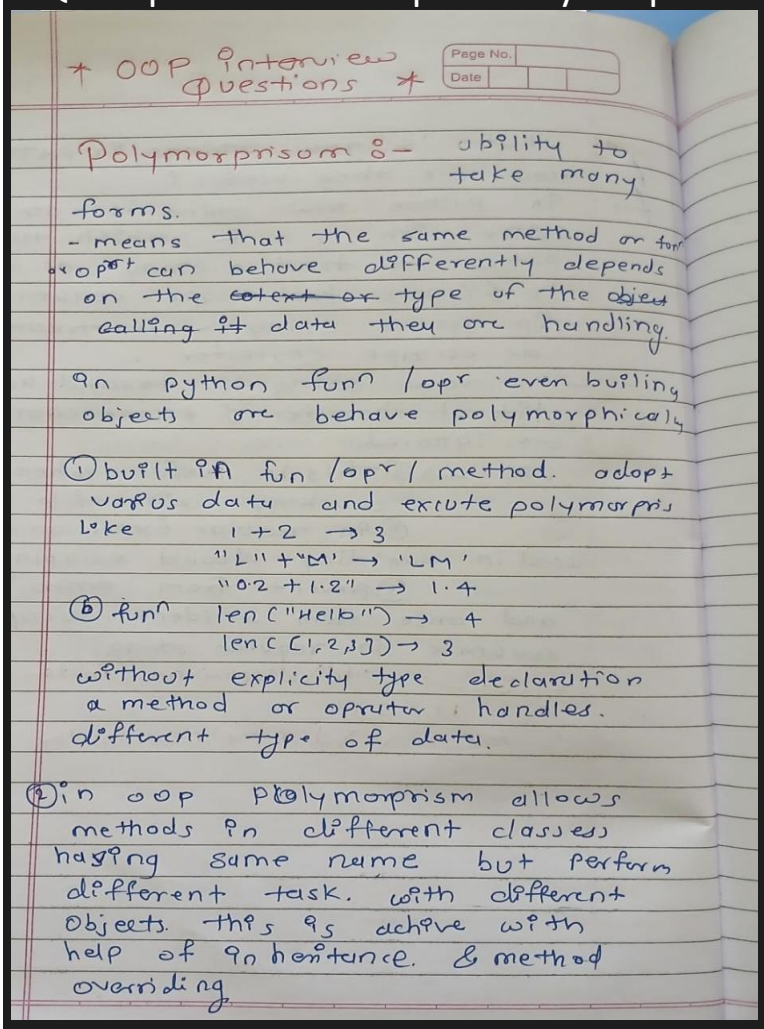
Q.27] what is raw strings in python How are they useful?

→ In python raw string is a string prefix with r or R, which tells python to treat a string as it is & ignore ~~back~~ escape sequence
- In regula string '\' is treated as escape character.
- in raw string '\' treated as literal character. & escape characters are ignored.

Use → ① This is useful when working with file paths.
② In regular Expression often used in avoiding dbould escaping.
③ Prevent from errors.
and arise from accident escape sequence in regular string.

eg :- S = r"\c\display\new_folder"
Pr(S)
o/p = \c\display\new_folder

Q7. Explain the concept of Polymorphism in Python with example.



* OOP interview Questions *

Page No.
Date

Polymorprisom :- ability to take many forms.

- means that the same method or fun opert can behave differently depends on the ~~extent or~~ type of the ~~objects~~ calling it data they are handling.

In python fun / opr even builing objects are behave polymorphically

① built in fun / opr / method. adopt various data and exicute polymorpis like
1 + 2 → 3
"L" + "M" → 'LM'
"0.2 + 1.2" → 1.4

② fun len("Helb") → 4
len([1,2,3]) → 3
without explicity type declaration a method or opratw handles. different type of data.

③ in oop polymorprism allows methods in different classes having same name but perform different task. with different objects. this is achive with help of inheritance. & method overriding.

eg:-
```
class shape:
    def area():
        print("Base no area is defined")
class circle(shape):
    def area(self,r)
        print(r*r*3.14)
class rectangle(shape):
    def area(self,L,b)
        print(L*b)
c = circle()
c.area(10)  → 1000
r = rectangle()
c.area(10,20) → 220
```

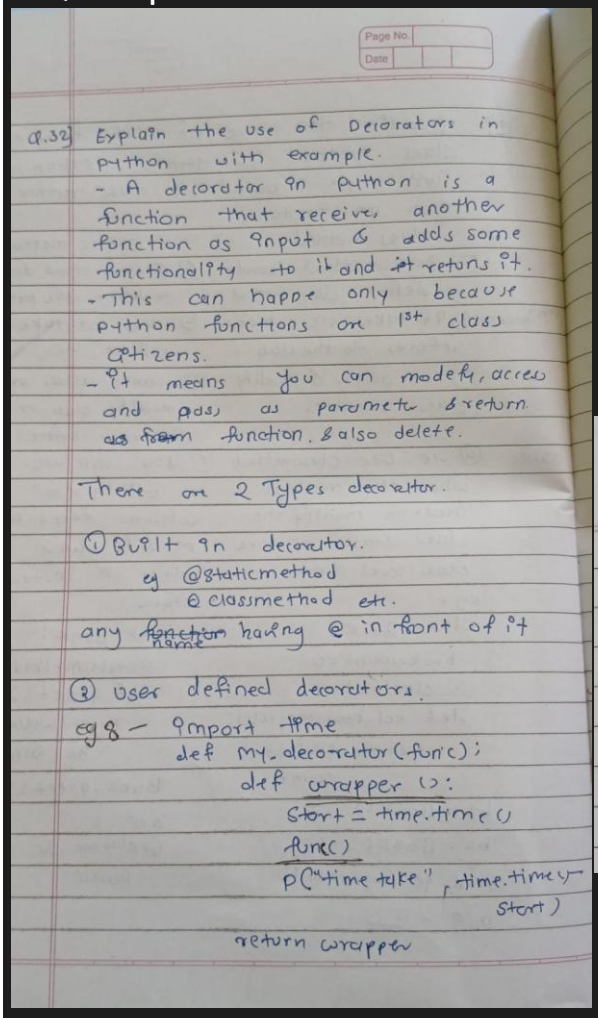
Here area method behave differently when object c is called and Overrides area method to return Circles area.
Here the area is called at runtime executes diffent method based on type of object which calls the area method.

Q8. What is the role of __init__ in python.

q.30] what is the role of __init__ in python

__init__ is a Constructor method in py & is automatically called to allocate memory when a new object is created. All classes have a __init__ method associated with them. it helps in distinguishing method if you not exciplicitly defined __init__ method in class python will automatically privide default constructer. & obj will be created without initizing any attributes value.

Q9. Explain the use of Decorators in Python with example.



Q.32 Explain the use of Decorators in python with example.

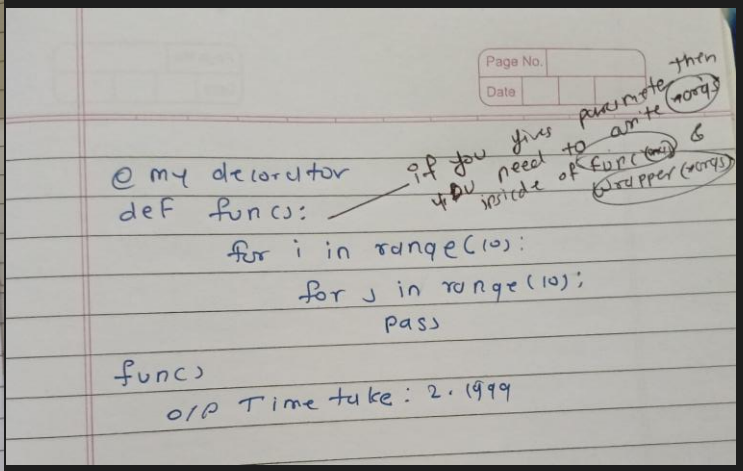
- A decorator in python is a function that receive, another function as input & adds some functionality to it and it returns it.
- This can happe only because python functions are 1st class citizens.
- It means you can modify, access and pass as parameter & return as from function. & also delete.

There are 2 Types decorator.

① Built in decorator.
eg @staticmethod
@classmethod etc.
any function name having @ in front of it

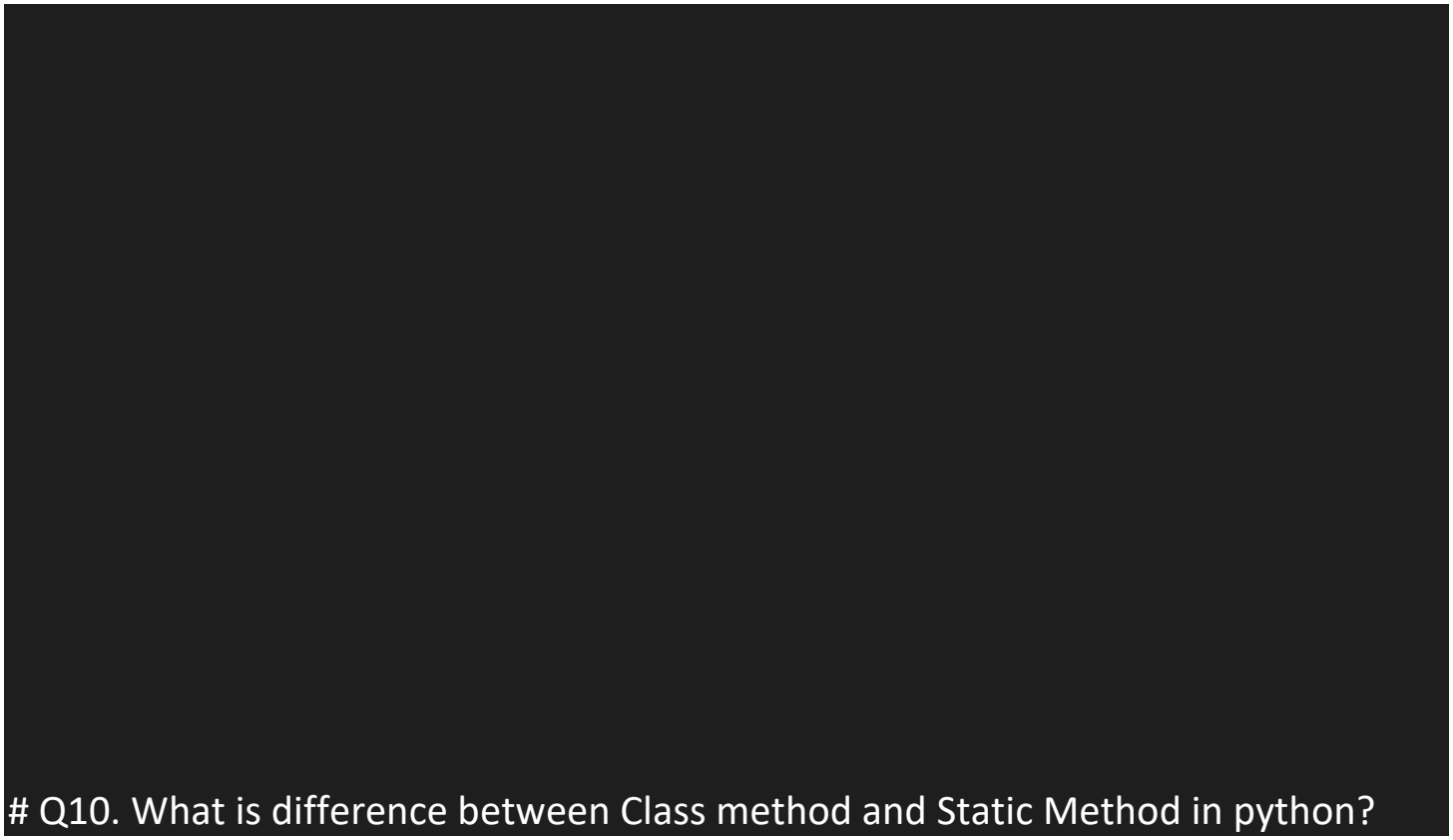
② user defined decorators.

```
eg8 - import time
      def my_decorator(func):
          def wrapper():
              start = time.time()
              func()
              p("time take", time.time() - start)
          return wrapper
```



```
@ my decorator
def func():
    for i in range(10):
        for j in range(10):
            pass
func()
o/p Time take: 2.1999
```

if you gives parameter then you need to write (args) & inside of func(args) & wrapper(args)



Q10. What is difference between Class method and Static Method in python?

Page No.
Date

Q.39] Explain the use of Diff between class method & static method in Py. Both are bound to class rather than an instance.

class method	Static method
① @classmethod decorator to define class method	① @staticmethod decorator to define class method
② It takes cls, which refers to the class	② Does not take self or cls.
③ Can access & modify class variables	③ Can't access or modify class or instance state.
④ We use classmethods when the method needs to modify the class state or access class-level data.	⑤ You can use as utility fun^n which doesn't modify any class or instance state.

eg:

```
class Book:
    book_count = 0
    @classmethod
    def get_bookcount(cls):
        return cls.book_count
b1 = Book()
b2 = Book()
Book.get_book_count()
```

o/p - 2

```
class Bank:
    @staticmethod
    def greet():
        print("Welcome to Bank")
Bank.greet()
```

o/p
Welcome to Bank.

Section B: Correct the Code (10 Questions)

```
# Q1.

# numbers = [1, 2, 3, 4, 5]
# for i in range(len(numbers)):
#     print(numbers[i+1])
# ANS :List out of index
# numbers = [1, 2, 3, 4, 5]
# for i in range(len(numbers)):
#     print(numbers[i])


# Q2.
# count = 5
# while count >= 0:
#     print(count)
# ANS  infinite loop
# count = 5
# while count >= 0:
#     print(count)
#     count-=1


# Q3.
# for i in range(10):
#     if i % 2 == 0:
#         continue
#     print(i)
# ANS  no need to change


# Q4.
# for i in range(3):
#     for j in range(3):
#         if i == j:
#             print(i, j)
#             break
# ANS
#  for i in range(3):
#     for j in range(3):
#         if i == j:
```

```
#             print(i, j)
            # break


# Q5.
# num = 15
# if num % 3 == 0:
#     print("Divisible by 3")
# elif num % 5 == 0:
#     print("Divisible by 5")
# ANS
# num = 15
# if num % 3 == 0:
#     if num%5==0:
#         print("Divisible by both")
#     else:
#         print("Divisible by 3")
# elif num % 5 == 0:
#     print("Divisible by 5")


# Q6.
# x = 10
# if x > 5:
# print("Greater than 5")
# else:
#     print("Less than or equal to 5")
# ANS
# x = 10
# if x > 5:
#     print("Greater than 5")
# else:
#     print("Less than or equal to 5")


# Q7.
# a, b = 5, 10
# if a > 0 and b < 5:
#     print("Condition met")
# else:
#     print("Condition not met")
# ANS
# a, b = 5, 10
```

```
# if a > 0 and b > 5:
#     print("Condition met")
# else:
#     print("Condition not met")


# Q8.
# class Parent:
#     def show(self):
#         print("Parent class")

# class Child(Parent):
#     def show(self):
#         print("Child class")
#         super.show()

# obj = Child()
# obj.show()
# ANS
# class Parent:
#     def show(self):
#         print("Parent class")

# class Child(Parent):
#     def show(self):
#         print("Child class")
#         # super.show()

# obj = Child()
# obj.show()


# Q9.
# class Car:
#     def __init__(self, brand):
#         brand = brand

# c1 = Car("Toyota")
# print(c1.brand)
# ANS
# class Car:
#     def __init__(self, brand):
```

```
#         self.brand = brand

# c1 = Car("Toyota")
# print(c1.brand)



# Q10.
# class Animal:
#     def __init__(self, name):
#         self.name = name

# class Dog(Animal):
#     def __init__(self, breed):
#         self.breed = breed

# dog = Dog("Labrador")
# print(dog.name)
# # ANS
# class Animal:
#     def __init__(self, name):
#         self.name = name

# class Dog(Animal):
#     def __init__(self, breed):
#         self.breed = breed
#         super().__init__(breed)

# dog = Dog("Labrador")
# print(dog.name)
```

Section C: Write Code For (10 Questions)

```
# Q1. Define a class person with attributes for name and age. Implement a subclass Employee that adds an attribute for salary and method to calculate the annual bonus.
#     (eg: 10% of salary)
# class Person:
#     def __init__(self,name,age):
#         self.name=name
```

```
#         self.age=age
# class Employee(Person):
#     def calannualbonus(self,salary):
#         print(self.name,self.age)
#         print(f"Anual bonus: {salary*1/10}")
# e=Employee('lavanya',24)
# e.calannualbonus(1200000)


# Q2. Implement a function that generates a random password of given length, The password should include uppercase letters, lowercase letters, digits and special character.
# len=int(input("Enter length "))
# import string
# import random as rd
# def generaterandompassword(l):
#     upperchr='ABCDEFGHIJKLMNOPQRSTUVWXYZ'
#     lowerchr='abcdefghijklmnopqrstuvwxyz'
#     digit='1234567890'
#     specail=string.punctuation
#     allchar=upperchr+lowerchr+digit+specail
#     requred=[]
#     requred.append(rd.choice(upperchr))
#     requred.append(rd.choice(lowerchr))
#     requred.append(rd.choice(digit))
#     requred.append(rd.choice(specail))
#     pwd=[]
#     pwd.extend(requred)
#     for i in range(len-4):
#         pwd.append(rd.choice(allchar))
#     print(pwd)
#     rd.shuffle(pwd)
#     return "".join(pwd)
# print(generaterandompassword(len))
# output:
# Enter length 10
# ?u2TN.eB(D
# Enter length 8
# pJ<-f?8n


# Q3. Create a function that takes a string and returns a dictionary with the frequency count of each character.
```

```
# s=input("enter string ")
# print({i:s.count(i) for i in s})
```

Q4.Write a NumPy program to test whether none of the elements of a given array are zero.

```
# import numpy as np
# arr=np.array([1,2,3,10,9,7])
# f=1
# for i in np.nditer(arr):
#     if i==0:
#         f=0
#         break
# if f==1:
#     print("Test worked")
# else:
#     print("Test Failed")
```

Output:Test worked

Q5. Check a number is Automorphic or not.

```
# n=int(input("Enter no "))#76
# sqr=n**n#5576
# temp=n
# r=0
# while(n!=0):
#         if sqr%10==n%10:
#             r=r*10+n%10
#         n//=10
#         sqr//=10

# result=0
# while(r!=0):
#       result=result*10+r%10
#       r//=10

# if result==temp:
#       print(result,temp,"Automorphic number")
# else:
#       print("Not Automorphic ")
```

```
# output:
# Enter no 76
# 76 76 Automorphic number
# Enter no 12
# Not Automorphic


# Q6.  Write a NumPy program to test a given array element-wise for finiteness (not infinity or not a number).
# arr=np.array([10,0,np.inf,np.inf,12,1,np.nan])
# for i in arr:
#     if np.isfinite(i):
#         print(i,"Finite")
#     else:
#         print(i,"not finite")
# output:
# 10.0 Finite
# 0.0 Finite
# inf not finite
# inf not finite
# 12.0 Finite
# 1.0 Finite
# nan not finite


# Q7.Write a Python program to combine two dictionary by adding values for common keys
# d1={'a':10,'b':5,'c':20}
# d2={'a':5,'d':30,'c':20}
# merged={}
# for i in d1:
#     if i in d2:
#         merged.update({i:d1[i]+d2[i]})
#         d2.pop(i)
#     else:
#         merged.update({i:d1[i]})
# for i in d2:
#     merged.update({i:d2[i]})
# print(merged)
# output:{'a': 15, 'b': 5, 'c': 40, 'd': 30}
```

```
# Q8. Write a NumPy program to test element-wise for complex numbers, real numbers in a given array. Also test if a given number is of a scalar type or not.
# import numpy as np
# arr=np.array([12,12+0j,23+1j,67+34j])

# for i in arr:
#     if np.iscomplex(i):
#         print(i,"Complex")
#     elif np.isreal(i):
#         print(i,"Real")
#     else:
#         print(type(i))
# output:
# (12+0j) Real
# (12+0j) Real
# (23+1j) Complex
# (67+34j) Complex

# Q9. Create a class "Library" that maintains a list of books. Implement a method add a book. Borrow a book and list of books are currebtly available in library.
# class Library:
#     def __init__(self):
#         self.dict={}
#     def addbook(self,name,author):
#         self.dict.update({name:author})
#     def listbook(self):
#         print(self.dict)
#     def borrowbook(self,name,author):
#         if (name in self.dict) and self.dict[name]==author:
#             self.dict.pop(name)
#             print("Borrowd Succefully")
# l=Library()
# l.addbook('DeepWork','D R VINS')
# l.addbook('Frog',"Trowe")
# l.listbook()
# l.borrowbook('Frog',"Trowe")
# l.listbook()
# Output:
# {'DeepWork': 'D R VINS', 'Frog': 'Trowe'}
# Borrowd Succefully
# {'DeepWork': 'D R VINS'}
```

```
# Q10. Write a function that takes a string and returns a new string with all duplicate characters removed.
# def dupremove(s):
#     se=[]
#     for i in s:
#         if i not in se:
#             se.append(i)
#     return "".join(se)
# print(dupremove('lavanya'))
# output: lavnya
```