PYthon

C	C++	Java	Python
① Procedural → functions	OOP → class + obj → close to hardware → OS → EX	OOPs → Syntax → secure → long	OOPs → Syntax → Web → AI → ML → Library → Django
• exe ✗	• exe ✗	• exe ✓	②
* fastest	fastest	less than C++ (after)	Dynamic
Compiler	Compiler	Compiler + interpreter	compiler + interpreter
No operator overloading	operator loading	NO Operator loading	platform independent
pointers	pointer	NO pointer	• exe ✓
No multiple inheritance	Yes Multiple inheritance	No multiple inheritance	slower execution
			Compile + interpreter
			Yes Operator overloading
			NO pointer
			Yes Multiple inheritance

* How python achieves plateform independent?

one ⇒

a.py → Compiler → Byte code → interpreter / PVM → .exe → Machine code

Just because of byte code python is plateform independent.

Byte code → PVM

but one condition is you have to install PVM (python virtual Machine, then you will able to achieve independent Plateform.

① what is plateform independent:
plateform independent means executive file of one operating system will be run on another operating system that is called plateform independent.

⇒ How it is plateform independent: python is plateform independent because it provides Byte code and pvm (python virtual machine) it is condition, using, both thing we will able to run python exe file from one system to another system

Note: PVM is also plateform dependent.

Major minor micro (fixed bugs)

3.1.12 version

* python 2 vs python 3

why python is becoming so popular day by day.

Python features

* plateform independent: write once run anywhere.

(1) Free and open sources: Python is freely available on python.org and it's code is openly available on the internet you can edit and take it into your work.

(2) Easy to code: Easy to code because it's syntax is very easy to write

(3) Easy to read: There is no any concept of {}, (), <>, ; and headfile, datatype declaration but in the python indentation is available that helps to identify how much portion of a block of code. that's why it is easy to read.

(4) Multi paradigm language: It supports both procedural programming and object oriented programming language that's why it is Multi paradigm language.

(5) Library support: Python have much and more libraries like Django → website, Tensorflow → AI, pandas → Data scientist.

(6) Portable language / plateform independent: Write once run anywhere

(7) Interpreted language: Python is interpreted language so compile code line by line and easy to debugging it is a good advantage.

Integrated |

(8) Extensible: Python can by used with C++, Java and it can be used with any other languages as well.
Ex⇒ Google, Netflix.

(9) Dynamic memory allocation: In the python memory allocation happen at run time that's why it is dynamic memory allocation.

a=15; ⇒ int a; at runtime

* Limitations of python.

(1) Slow language: Interpreter compile code one by one language that's why it is slow language.

(2) High memory consumption: Because of dynamic memory allocation a huge libraries are used and all the memories allocated at the runtime that's why it consume more memory.

(3) Mobile App development is not common: In the python App development is not easy, App development is easy in Java.

* Syntax Error in python.

Data type in python

↳ Numeric → int = 56..... No limit (Complex number)
↳ int
↳ float → only float No L/CM
↳ complex
↳ 2+3i

↳ Seqvence (ordered datatype) → List, tuple, String

List ⇒ L = [] → string, int, float = [1, 'Ri', 1.6]
(indexes: 0, 1, 2 / -3, -2, -1)
print(L[1]) = o/p = 1.6
print(L[-3]) = o/p = 1
↳ mutable

Tuple ⇒ T = () ⇒ (1, 2.5, 'Rav')
T[1] = 3.5 ✗ immutable
Cannot be change.

* String : No char in python
s = 'a'
s = 'Hi'
s = "I am Son's of"
s = """Hi I am "Rayshan" """ Multi line string
(indexes: 0, 1, 2, 3)
index

unordered datatyp : → Set
→ Dictionary

* Set : • No duplicate value • It is mutable
• No concept of index
s = {1, 1, 2} → print(s) = 1, 2 or 2, 1
s = {1, 1.5, 'Rav'}

Imp | s = {}, print(type(s)) o/p = dictionary
s = set() print o/p = set

using loop you can access set.
for i in s: print(i)

* Dictionary : No index in this, access using key only, key can be anything, any value can be anything, dictionary is mutable.
D = {key : value, Name : 'Rav', age : 15}
print(key)
print('Rav') = error we can not access key using value
Assignment —

Q1. How to create array in python How to access element in array

Q2. What is the use of Negative index in list

Q3. What is the boolean datatype in python.

Operators

division float ↑ int division ↑

↳ Arithmetic : + − * % / // → floor

print(5/2) — 2.5
" (5//2) — 2
" (−5/2) — −2.5
" (−5//2) — −3

कभी भी floor division low floor लेता है
so −2.5 नहीं −3 आएगा

* Comparison Operator :

< ≤ == a=5
> ≥ != b=4
a>b T

* logical operator : ✓And ✓OR ✓ ! (Not)

a=5
b=4
T

print(a and b) # 4
print(a or b) # 5
print(a != b) # True

[redacted]

* Bitwise operator : &, |, ~, ^, <<, >>

a=5
b=4

& (AND)
| (OR)
^ (XOR)
<< (Left shift)
>> (Right shift)
~ (Not)

a = 0101
b = 0100
& = 0100 = 4
| = 0101 = 5
^ = 0001 = 1
<< =

a & b → 4
a | b → 5
~a → −6 (2's complement)
a ^ b → 1
a << 2 → 4 → 0101 → 0100 = 4
a >> 2 → 1 → 0101 → 0001 = 1

5
0101
1010
1c → 0101
2c → +1
0110 = −6

a = 1100 = −4
b = 0100 = 4
a & b = 4
a | b = −4
!a = 3 → 1100 → ! → 0011 → 3
a^b = −8
a<< = −16
a>> = −1

1100
0011
1100

1101 =

|−3| = 3

1100 << 0100 =
00001100 << 00000100 =

shift
Incase of right if the no is negative instead of putting 00 you have to put (11).

Q = a = −3 a << 3
b = −2 a >> 3
~a

* Identity operator: Identity operators are used to check if two values are located on the same part of the memory

Identity → is
→ is not

a=5
b=5 print(a is b) o/p = T

a = [10, 20, 30] print(a is b) ⇒ False
b = [10, 20, 30] print(a is not b) ⇒ True

x = a (assigning list to x variable)
print(x is a) # True
print(x is not a) # False

* print(id(a)) integer] memory address
print(id(b)) float] same

list
tuple
dictionary] memory address same

In simple data type variable share same memory address like → int, float but in advance python like list, tuple, dictionary variable share different memory address. a = [1,2,3]
b = [1,2,3]
print(a == b) = T

* Membership Operator → in
not in

x = 20
L = [10,20,30]
print(x in L) = True
print(x not in L) = false

* Turnary operator: a if a > b else b

* Operator precedence:

		Associativity
1	**	right to left
2	~ bitwise NOT	L → R
3	*, /, //, %	"
4	+, -	"
5	<< >>	"

Assignment = (R to L)

6	&	bitwise and	L → R
7	^	" xor	
8	|	" or	
9	identity + membership + Assignment operator		
10	Not		
11	and		boolean Not, and, or operator
12	or		

good question

(2) Name = "abc"
Age = 5
if name == "abc" or name == "def" and age > 10
print("a")
else print("2")

O/P ⇒ 1

if → (if name == "abc" or name == "def") and age > 10
then o/p = 2

(1) 5 + (100 / 10) - 3 * 2
5 + 10 - 3 * 2
5 + 10 - 6
15 - 6 = 9.0

(3) x = 6
y = 2
print(6//2) O/P: 3
print(6 ** 2) O/P: 36

(4) a = 4 = 100
b = 11 = 1011
print(~a) = -5
print(a|b) = 15

print(a >> 2) = 1
print(-13 || 4)
= -5

4 | 11
↓ or

(5) x=10 y=20
print(x and y) = 20
print (x or y) = 10

(6) x=10 y=50
if(x ** 2>100 and y==50)
print("Yes") // No output

(7) x=1
y=2
x=y+=4 = x=y=y+4
print(x) = x=y=6
x=6
// x = 6 X
Syntax error

Q Assignment what are non associative operators:

* for in → for i in — / L,T,D, :
while

L=[1,2,3,4]
T=(1,2,3,4)
S={1,2,3,4}
D={1:"H", 2:"by"}
for i in T/L/S/D :
print(i)

but in dictionary

for i in D :
print(i, D[i])
key value

Range

* range(start, stop, step)

for i in range(len(L)):
print(L[i])

* while loop | while(count<3):
while exp : | print("Hi") lesson
count=0

while (count<9)
print("Hi")
count ++;

जो भी variable use होगा उसको define करना जरूरी है।

* i=0
while(i<5):
print(i)
i=i+1
if i==3:
break
else:
print(0)

0
1
2
0

* loop control statement:
break, continue, pass

```
i=1
while True:
    if(i%7==0)
        break
    print(i)
    i=i+1
```

O/P: 1, 2, 3, 4, 5, 6

```
i=0
while(i<3):
    print(i)
    i=i+1
else:
    print(0)
```

O/P:
0
1
2
0

```
x=['ab','cd']
for i in x:
    print(i.upper())
```

['ab','cd']

AB
CD

```
x='abcdef'
while i in x:
    print(i)
```

i not defined

```
x='abcdef'
i=i+1
while(i in x):
    print(i)
```

no output

```
x='abcd'
i=0
while i in x:
    print(i)
```

take while of condition infinite, a print

```
i=0
while(i<5):
    print(i)
    i=i+1
    if(i==3):
        continue
    else
        print(i)
```

O/P:
0
1
2
4
0

Abish

* difference blw continue and break

* Nesting loop:

```
i = 1, j = i
for i in range(1,5):
    for j in range(i):
        print(j)
```

```
for i in range(1,5):
    for j in range(i):
        print(i, end=' ')
    print()
```

1
2 2
3 3 3
4 4 4 4

* Condition statement
  - if
  - if else
  - if elif else
  - Nested if

(1)

```
if age > 10:
    ==
else:
    ==
```

```
if a>b:
    ==
elif:
    ==
else:
    ==
```

```
if a>b:
    if ==
        ==
```

(2) Accept three sides of a triangle and check whether it is equilateral, isosceles or scalene. (3) triangle.

```
side1 = int(input("1st side"))
side2 = int(input("2nd side"))
side3 = int(input("3rd side"))

if (side1 == side2 == side3):
    print('equilateral triangle')
elif (side1 != side2 != side3):
    print('scalene triangle')
else
    print('isosceles triangle')
```

using if, else

100 − 30 = 70

Page:
Date: / /

Accept following from the user and calculate the % of the daily attendance
1. total no of lecture
2. total no of days he/she absent
3. After calculating percentage, show that if % is less than 75 then user will not able to sit in the exam.

```
lec = input("Enter lecture No:")
days = input("Enter total days absent:")
total_attendance = lec - days
wow = (total_attendance / lecture) / 100 * 75
if (wow >= 75)
    print("You can sit")
else
    print("You cannot sit")
absent_percen = (days / lec) * 100
present = 100 - absent_percen
if (present >= 75)
    print("You can sit")
else
    print("You cannot sit")
```

Take an integer n and perform following options:
1. if n is odd → weard
2. if n is even and in the inclusive range of 2 to 5 print Not weird.
3. if n is even and in the inclusive range of 6 to 20 print Weird.
4. if n is even and greater than 20 print not weird.

```
n = int(input("Enter n"))
if (n%2 != 0)
    print("weard")
if (n%2 == 0) if (n > 20) print("Not weird")
elif n in range(2,6)
    print("Not weird")
elif n in range(6,21)
    print("weird")
```

9
21
4

part-1

* String methods :

(i) lower() : (ii) upper() (iii) Capitalize()

(iv) title() → s = "Second year"
Second Year

limitation → "student's data"
→ Student'S Data
↑ limitation

(v) swapcase() → lower to upper, upper to lower
s = "Second YEAR"
o/p = ""

(vi) casefold() → similar to lower, but more agressive, ß को ss lower नहीं करता है।
s = "ß"
s.lower() → ß
s.casefold() → ss

(vii) center() → this method creates and return a new string that is paded with the specified character.

& Assign → diff b/w isdecimal(), isdigit(), isnumeric() string -

s = " This is D2IT "
s1 = s.center(16) || (16, #) o/p
print(s1)
o/p = ___This is D2IT___
actual len = 12 → 16-12 = 4

(viii) count() → count function returns the no of occurances of a substring within a string and it is case sensitive.
Ex ⇒
s = " This is D2IT "
s.count(i) || o/p
s.count(i, start, end);

(ix) endswith() : This method returns true if a string ends with the given suffix otherwise returns false.
s = "This is GNDEC."
print(s.endswith("DEC")) → F
" (s.endswith("GNDEC.")) → F
("GNDEC.") → T
("GNDEC") → F
("is GNDEC") T
("This is GNDEC.") → T

(10) find().

(11) index().

(2 marks) write a program to distinguished b/w find and index method of a string

find()	index()
(i) find() methode return the lowest index of the first occurance of the substring.	index() method return the index of first occurance of the substring.
(ii) find() method if the substring in not found by find() method then it return -1.	if the substring is not found by index() method then it throws exception in this case.
(iii) Syntax: (a) s="find one if you can" print(s.find("me")) → 5 (b) print(s.find("me", 4, 10))	(a) print(s.index("me")) → 5 (b) print(s.index("me", 4, 10))
(iv) It can take three parameters.	this also can take three parameters.

(3) WAP to count no of whitespace in the given string
var = "Hi 9 am"
count = var.count(" ")
print(count)

(3). WAP To find the no of whitespaces in the given string.

part-2

String methods:

(1) isprintable(): This method returns true, if all the characters of the string are printable or the string is empty.

S = "This is DeIT" → F
S.isprintable() → F
S1 = " "
S1.isprintable() → T

(2) isspace():

S = "This is DeIT"
count=0
for i in S:
    if(i.isspace()==True):
        count++
print(count) | o/p = 2

(3) istitle() → if the given string is title case (UpperCase) it returns true otherwise false

(4) join() → It is used to join elements of the sequence seprated by a string seprater

#, !!& $

```
s = '_'.join("D2IT")
O/P → _D_2_I_T
s = {1, 2, 3, 4, 5, 6, 7}
s1 = '#'
print(s1.join(s))
O/p → #1, #2, #3 - - -
```

increase of dictionary

```
s1 = '#'
s1 = {1:'A', 2:'B'}
print(s1.join(s))
```

1#2#

(5) ljust() & rjust() → sister of center()

lj → padding on right side
rj → padding on left side

Ex →

```
s = "D2IT"
s.rjust(6) → ##D2IT
s.ljust(7,'#') → D2IT###
```

(6) (i) partition() → This method splits the string at the first occurance of the separator and returns a tuple containing the part before the separator, the separator and the part after the separator.

```
s = "I love GNDEC Ludhiana"
s.partition("love")
O/P → ('I', 'love', 'GNDEC Ludhiana')
      L ←——→ R
s = "I love GNDEC and I love coffee"
s.partition("love")
O/p → ('I ', 'love', ' GNDEC and I love coffee')
```

if separator is not present

```
("this", ' ', ' ')
```

(ii) rpartition() → from right side

```
s = "I love you and I love cricket"
s.rpartition("love")
('I love you and ', 'love', ' cricket')
('I love you and ', 'love', ' cricket')
```

⑦ replace() →

S = "Hello world Hello"

S.replace("Hello", "Bye")

o/p → Bye world Bye

S.replace("Hello", "Bye", 2)
↳ 2 times

o/p → Bye Bye world Bye Bye.

⑧ startswith() → true / false

S = "Greeks for Greeks"
0 1 2 3 4

S.startswith("Greek") → True

S.startswith("for", 6, 9) → True

S.startswith("for", 6, 8) → false.
(n-1)

⑨ %find() & %index() →

File handling

* How to open a file:

```
with open("file.txt", "r") as f:
    print(f.read())
```

file.txt: Hello world

you can pass the value (2) → He

read() display whole content of file.

* Read line →

```
with open("file.txt", "r") as f:
    print(f.readline())
```

readline() displays the first line of file only.

* Readlines →

```
with open("file.txt", "r") as f:
    print(f.readlines())
```

Hello world

o/p → ["Hello", "world"]

readlines displays whole file in list.

* WAP that will display third line of file.

```
with open("file.txt", "r") as f:
    L = f.readlines()
    print(L[2])
```

* How to create a file:

```
with open("file.txt", "x") as f:
```

(Q12) capability of write fun यदि file exist नहीं करती है तो create कर देती है

```
with open("file.txt", "w") as f:
    f.write("Hellow in world in Bye")
```

write → It override the old content

In append mode

```
with open("file.txt", "a") as f:
    f.write("Hello in world")
```

* Remove operation.

```
import os
os.remove("file1.txt")
```

* How to create csv file in python.

* WAP that will count and display total no of words in a text file.

```
count=0;
with open("file.txt", "r") as f: data=file.read()
for line in f:
    words = line.split(" ");
    count = count + len(words);
print("No of char:"+str(count))
```

dekhlena

```
with open(f.txt") as f:
content = f.read()
print(len(content.split()))
```

* WAP that will display the count of 'The' word in your text file.

① MDSL

② immutable → ImFlaFB

(Pyq) What are the immutable data types in python?

ans ⇒ Mutable data types can be modified after creation

Mutable data types

① List ② Dictionaries ③ Sets

Immutable data types

Immutable data types cannot be changed once they created

① strings ② Tuples ③ integers ④ floats ⑤ Booleans

(Pyq) difference b/w isdecimal(), isdigit(), ~~isalpha()~~ and ~~isnum~~ isnumeric() string functions.

isdecimal()	isdigit()	isnumeric()
(i) It returns True if all characters in the string are decimal digit (0–9)	It returns True if all characters in the string are digits.	It returns True if all characters in the string are numeric.
(ii) It doesn't accept superscript or subscript digits.	It accepts superscript and subscript digits.	It accepts superscript or subscript and as well as fractions.
(iii) example: isdecimal("1234") o/p: True isdecimal("$1^2 3^4$") o/p: False	example: isdigit("1234") and isdigit("$1^2 3^4$") o/p: True isdigit("1/2") o/p: false	Example: isnumeric("1234"), isnumeric("$1^2 \frac{3^4}{5}$"), and isnumeric("1/2") o/p: True

(PY8) write a program to accept a number from a user and calculate the product of all the digits
n=56 o/p = 30

```
n = int(input("Enter a number:"))
sum = 1
while n > 0:
    digit = n % 10
    sum = sum * digit
    n = n // 10;
print("The product of the digits is:", sum)
```

(PY9) write a program to accept a number from user and calculate the ~~product of all the digits of the given~~ the sum of all numbers from 1 to the given number.

any ⇒
```
n = int(input("Enter a number:"))
total_sum = sum(range(1, n+1))
print("The sum is:", total_sum)
```

OR ⇒
```
sum = 0
n = int(input("Enter a number:"))
for i in range(n+1):
    sum = sum + i
print("Sum is:", sum)
```

(PY10) Write short Note on Operator precedence vs operator Associativity.

Operator precendence	Operator Associativity
It dictates the order in which operations are performed when multiple operators are present in an expression.	It determines how to group operands when multiple operators have the same precendence.
Exponential **	Right to left
unary plus/Minus, bitwise not +, -, ~	Right to left
*, /, //, %	left to right
Addition, Subtraction +, -	"
bitwise shift left, right <<, >>	"
<, <=, >, >=	"
==, !=	"
and, or	"
Assignment operators =, +=, -=, *=, /=, //=, %=, **=, &=, ^=	Not applicable
	Not applicable

Python ascii

0 →	48	A → 65	a → 97
1 →	49	B → 66	b → 98
2 →	50	C → 67	c → 99
3 →	51	!	!
4 →	52	:	:
5 →	53	Y → 89	x → 121
6 →	54	Z → 90	z → 122
7 →	55		
8 →	56		
9 →	57		

```
o/p=? L=['a','b','c','d','D']
L.sort(reverse = True)
print(L)
o/p: ['d','c','b','a','D']
L.sort(reverse = False)
o/p: ['D','a','b','c','d']
```

Q7 (PYQ) How to read and write into a Text files in python.

ans ⇒

To read from a text file in python

```
with open('filename.txt', 'r') as f:
    content = f.read()
    print(content)
```

To write to a text file

```
with open('filename.txt', 'w') as f:
    f.write('Hello, I am mayuthem.')
```

Q8 (PYQ) what is difference b/w : count(), and length(), function in List ?

count()	length()
(i) count() method is used with list to count the occurrences of a specified element.	len() gives the total no of element in the list
(ii) list = [1,2,3,1,4,1,5] target = 1 count = list.count(target) print(count) o/p → 3	(2) list = [1,2,3,1,4,1,5] count = len(list) print(count) o/p → 7

MST-2

* list → list are used to store multiple items in a single variable

Ex ⇒ list1 = []

list - items are ordered, changable and allows duplicate values.

list[5] = Not found
list[-1] = 2
list[n-1] = 2
list[-4] = 1
list[2] = "GOOD"

* append(), insert(), remove(), pop(), extend(), count(), reverse(), sort(), clear(), length(), sum(), min(), max().

append() केवल एक ही insert कर सकते हैं,
extend() multiple value can insert

* Nested list : list inside list

list = [1,2,3,[4,5,[7,8],9],10,11]

* Diff b/w list, tuple, set, dictionary Reema

List	tuple	Set	dictionary
mutable	immutable	mutable	~~Dictionary~~ Mutable
Allow duplicate value	Allow	doesn't allow	Doesn't allow
Empty list = []	~~tuple~~ tuple = ()	set = set()	Dictionary = { }
It can store any data type even list, tuple, set	same	It can store data types (int, str, tuple, but not list set, dic	key value (int, str and tupple only values can be of any type.
ordered & indexed	same	unordered	unordered

* Adding key value pairs to the dictionary

my_dic["name"] = "John"

my_dic["age"] = 30

my_dic["Hi"] = "New York"

* # Access value by key

print("age", my_dic["age"])

this_dict = {"brand": Ford", "Model": mustang"

"year": 1964, "year": 2020}

✗ ✓ = selected item

* check if key exist in dictionary

if "name" in any_dict:
    print("key 'name' exist in the dictionary")

else:
    print("key "name" does not exist in the dict")

Dictionary: Dictionary are used to store data values in key value pair, the values are in dictionary can be any thing and key must be unique and must be immutable object like num, string & tupple.

* function: function help to organize code into logical block that perform specific task benefits of function increase code readability, reuseability.

def function name (parameters):
    # statement
    return expression.

* add two no.

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

print(add(5,5))
```

* add multiple no using python (*args) arbitary argument

```
def add_multiple(*args):
    return sum(args)

result = add_numbers(1,2,3,4,5)
print(result)
```

op: 15

* Arguments in python function

(i) default arguments: default argument is a parameter if a value is not provided.

Ex=>

(ii) keyword arguments: The idea is for allow the the called the specified argument name with value

keyword argument is a parameter passed to a function with the parameter name. This allow you to provide value for specific parameters by their name.

(iii) positional argument: We use the argument during function call so that the first argument is assign to name, the second argument is assign to second value, by changing the position or you forget the order of position.

default argument example

```
def my_fun(x, y=50):
    print("x:", x)
    print("y:", y)

my_fun(10)
```

Ex=>

keyword →

```
def Statement(first_name, second_name)
    print(first, second_name)

Statement(first_name="John", second_name="Rau")
```

positional

```
def namage(name, age)
    print("Hi, I am:", name)
    print("my age is", age)
print("case 1")
name Age("Suraj", 27)
Point("case 2")
name Age(27, "Suraj")
```

*

Arbitary keyword Argument

*args → It allows you to pass a variable no of positional arguments to a function, It collects these positional arguments in tuple.

```
def my-fun(*args)
    for age in args:
        print(args).
my-fun(1,2,3,4,7)
```

**kwargs : It collect these keywords into a dictionary.

diff b/w args & kwargs

*args	**kwargs
(i) It collects arguments in tuple.	It collects argument in dictionary.
(ii) It captures an arbitary number of positional arguments	It captures an arbitary number of keyword arguments
(iii) def my-fun(*args): for arg in args: print(arg) my-fun(1,2,3,4)	def my-fun(**kwargs): for key, value in kwargs.items(): print(f"{key}:{value}") my-fun(name="Ram", age=25 city="london") o/p: name: Ram age: 25 city: london
(iv) There is no any concept of key, value pair	There is concept of key, value pair.

practical # → loops - 1,2,3,10
strings - 1,2,7
list - 9,4

(1) Append VS extend Assignment Tues 5th week
(2) pop VS remove
(3) rev reverse list

* lambda function = It is also known as anonymous function, It is used to define anonymous function, It is defined using 'lambda' keyword

```
def add(a,b)
    return a+b
addl = lambda x,y: x+y
minuy = lambda x,y: x-y
print(minuy(3,4)) ds=5
print(addl(5,4)) print(add(5,9))
def disp()
    def show()
        print("show fun")
    print("Display fun")
```

* lambda fun is concise way to create small anonymous fun it is defind by 'lambda' keyword lambda fun are often used for short-term operations where full function definition is not necessary.

* GUI : Python offers multiple option for developing GUI, out of all the tkinter is most commonly used method. It is a standard python interface to the Tk GUI toolkit shipped with python and easiest way to create GUI to tkinter, python.

(1) Import the module - tkinter.
(2) create the main window (container)
(3) Add any no of widgets to the main window.
(4) Apply event trigger on the widgets.

Im → python 2.x is 'Tkinter'
Im → python 3.x 'tkinter'

Widgets in GUI

(1) Button ✓
(2) Canvas
(3) checkButton ✓
(4) Entry ✓
(5) Frame
(6) Label ✓
(7) Listbox
(8) Menubox
(9) Menu
(10) radiobutton ✓
(11) Messagebox ✓
(12) Message

* Two main methods which the user needs to remember while creating a Python Application.

(1) m = tkinter.tkU
(2) m.mainloop();

* tkinter also offer access to the geometrical configuration of the widgets which can organize the widgets in the parent window.
methods

(1) Pack U : Organize the widgets in blocks before placing in the parent widgets

(2) Grid() : It organize the Widgets in grid table like structure plays before placing in the parent widgets

(3) place : It organize the widgets by placing them on specific position directed by the programm.

append

* list1 = [1, 2, 3]
list1.append(4) ⇒ [1, 2, 3, 4]
list1.append([5, 6]) ⇒ [1, 2, 3, 4, [5, 6]]

Extend → list2 = [1, 2, 3]
list3.append([4, 5])
# [1, 2, 3, 4, 5]

(8 Marks) Design GUI using Tkinter to order a pizza from dominos. choose data and widgets accordingly.

ans⇒

```
import tkinter as tk
from tkinter import ttk

def order_pizza():
    pizza_size = size_var.get()
```

(Pg 9) Using ttk widgets over standard Tkinter offers several advantages in terms of appearance and functionality.

ans⇒ Indeed using ttk widgets over standard Tkinter offers several advantages

(i) Modern Appearance: ttk widgets have a more modern appearance compared to the standard Tkinter widgets, making application look more visually appearing

(ii) Native look and feel: ttk widgets adapt to the native look and feel of the underlying operating system providing more consistent user experience

(iii) Customization: ttk widgets offer more customization options, fonts, and styles to match the design requirement

(iv) Additional functionality: ttk widgets often come with additional functionality that was not available in standard Tkinter widgets. such as themed buttons, comboboxes and progress bars.

(v) Improved performance: ttk widgets often ~~come with additional functionality~~ more lightweight and can offer better performance compared to standard Tkinter widgets.

Page:
Date: / /

* Tkinter module
File dialogue module -
Color chooser ->
ttk module
(i) Note book
(ii) progress bar
(iii)

(8) (Pgm) write a program to find LCM of two two numbers using function in py.

Soln:

```
def find_gcd(x,y):
    while(y):
        x, y = y, x%y
    return x
def find_lcm(x,y):
    gcd = find_gcd(x,y)
    lcm = (x*y) // gcd
    return lcm
num1 = int(input("Enter first number: "))
num2 = int(input("Enter second number"))
print("The LCM of", num1, "and", num2, "is:",
find_lcm(num1, num2))

# num1 = 6, num2 = 8 o/p: 24
```

This program defines two fun^{n} find_gcd() to find the Greatest Common Divisor (GCD) of two numbers and find_lcm() to find the LCM using formula (x*y)//gcd(x,y)

Page:
Date: / /

(Pgm) write a program using fun^{n} to multiply all the numbers in a list sample use = [8,2,3,-1,7]
expected o/p: -336.

Soln:

```
def multiply(numbers):
    result = 1
    for num in numbers:
        result = result * num
    return result

sample_list = [8,2,3,-1,7]
output = multiply(sample_list)
print("Expected output:", output)
```

This program defines a function called multiply that takes a list of all numbers of input. it initialize a variable result to 1 and then iterate through each number in the list.

(Pgm) Write short Note on following with suitable ~~syntax~~
(a) Constructor in python
(b) Multilevel inheritance

(a) Constructor: A constructor in python is a special method that initializes objects of a class, it is called automatically when an object is created. In python, the constructor method is '__init__()'.

Example -

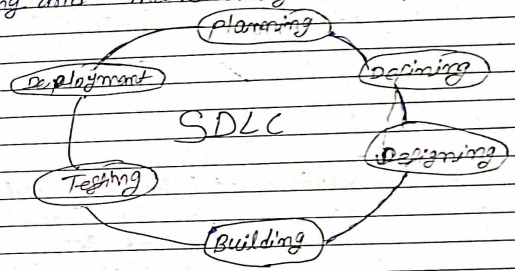
```
class Myclass:
    def __init__(self, arg1, arg2):
        self.arg1 = arg1
        self.arg2 = arg2
# creating an instance of Myclass
obj = Myclass("Hello", 42)
# accessing instance variables
print(obj.arg1) # o/p: Hello
print(obj.arg2) # 42
```

(b) Multiple inheritance: Multiple inheritance in python refers to the concept where a derived class inherits from a base class, and then another class inherits from that derived class.

```
class Base:
    def __init__(self, x):
        self.x = x

class Derived1(Base):
    def __init__(self, x, y):
        super().__init__(x)
        self.y = y

class Derived2(Derived1):
    def __init__(self, x, y, z):
        super().__init__(x, y)
        self.z = z

obj = Derived2(1, 2, 3)
print(obj.x) # o/p: 1
print(obj.y) # o/p: 2
print(obj.z) # o/p: 3
```

(Q9) what is the need of SDLC and stages of SDLC life cycle in python?

ans) SDLC is a method, approach or process that is followed by a software development organization while developing any software. SDLC comprises a detailed description or step-by-step plan for designing, developing, testing and maintaining the software.



Stages of the software Development Life Cycle
SDLC specifies the task to be performed at various stages by a software engineer or developer.

Stage 1	Stage 2	Stage 3	Stage 4	Stage 5
planning & Requirement	Defining Requirement	Design	Develop ment	Testing

Stage 6
Deployment & maintenance

Page:
Date: / /

① Stage 1: planning and Requirement Analysis: planning in software development involves gathering requirements from customers and market surveys, forming the foundation of the project. The quality of the project relies on this stage.

Planning → Define project scope → Set objectives and Goals →

Resource planning

② Defining Requirements: In this stage, all the requirements for the target software are specified. These requirements get approval from customers, market analysts and stakeholders. This is a sort documents that specifies all those things that need to be defined and created during the entire project cycle.

Defining → functional Requirement → Technical Requirement → Requirement Reviewed & Approved

③ Stage 3: Designing Architecture: The software requirements specifications (SRS) guide software designers in creating optimal architecture. The design Document specifications (DDS) contain multiple designs for the product architecture based on the defined requirement in the SRS.

Design → low level Design → High Level Design

Stage 4: Developing product: ~~At this stage~~ During this stage, developers begin the core development of the product by implementing specific programming tools such as compilers, interpreters, and debuggers are utilized.

Development → Coding standard → scalable Code →

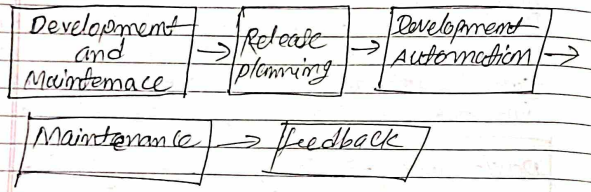
version control → Code Review

Stage 5: Product Testing and Integration: After the development of the product, testing of the software is necessary to ensure its smooth execution. Although minimal testing is conducted at every stage of SDLC. Therefore at this stage, all the probable errors are tracked, fixed and retested.

System Testing → Manual Testing → Automated Testing

Stage 6: Development and Maintenance of products:

After testing, the product is deployed in phases according to the organization's strategy and tested in real world industrial environment to ensure smooth performance. if it performs well, the organization sends out the product as a whole.



Q. ~~Caesar cipher~~

Caesar_cipher_emcrypt program:

```
def caesar_cipher_emcrypt(text, distance):
    encrypted_text = ''
    for char in text:
        if char.isprintable():
            encrypted_char = chr((ord(char)-32+distance)%95+32)
            encrypted_text += encrypted_char
        else:
            encrypted_text += char
    return encrypted_text

text = input("Enter the text: ")
distance = int(input("Enter the distance value: "))
encrypted_text = caesar_cipher.encrypt(text, distance)
Print("Encrypted text: ", encrypted_text)
```

* caesar_cipher_decrypt program.

```
def caesar_cipher_decrypt(encrypted_text, dist):
    decrypted_text = ''
    for char in encrypted_text:
        if char.isprintable():
            decrypted_char += decrypted_char
        else:
            decrypted_text += char
    return decrypted_text

encrypted_text = input("Enter the encrypted text: ")
distance = int(input("Enter the distance value: "))

decrypted_text = caesar_cipher_decrypt(encrypted_text, distance)
print("Decrypted text:", decrypted_text)
```

* what is class: A class in python is a blueprint for creating objects. It defines the attributes and methods that the objects of the class will have. To create a class in py we can use class keyword followed by the name, you can declare attributes and methods that belong to that class.

```
Class MyClass:
    def __init__(self, x, y)
        self.x = x
        self.y = y
```

2 Marks final Questions

Parameter	List	Tuple	Set	Dictionary
Mutability	Mutable	immutable	Mutable	Mutable
Duplicate value	Allowed	Allowed	Not Allowed	Not allowed
key value pair	NO	NO	NO	Yes
Arrangement	ordered	ordered	unordered	unordered
Empty lists	list = []	tuple = ()	Set = {}	Dictionary = {}
Access	print(list[0])	print(tup[0])	print(set[0])	print(dict['age'])
Example	list = [1, "RAM", 3.1]	tuple = (10, "RAM", True)	set = {20, "RAM", 20} set.add(10)	dic = {"name": "RAM", "age": 30} dic['age'] = 19
Modification	list[0] = 2	~~tuple[0] = 2~~ X	~~set[0] = 20~~	

① What is the purpose of setting up path and environment variable for python?

ans> Accessing python globally: Setting up the path enables you to run python from any directory without specifying its full path.

Running python scripts: Environment variables allow you to execute python scripts from any location without specifying the full path

Accessing installed packages: Python is installed in specific directories setting up environment variables ensure python can locate and use these packages.

② diff b/w a string and a numeric data type in py

Features	string	numeric
Definition	Represents set of characters	Represents numerical values
Example	"Hello", 'python', "123"	123, 3.14, -5
Operations	Concatenation, slicing, formatting etc.	Arithmetic operations (+, -, /)
TYPE	str	int, float
immutability	strings are immutable	Numeric are mutable
Use	used to represent textual data	used to represent arithmetic calculations

③ How can syntax errors be detected and corrected in python.

ans⇒ detect
- Error Messages: python provides descriptive error messages that pinpoint the location of syntax errors.
- IDE features: IDE often highlight syntax errors in real time, making it easy to identify and correct.

correct
- Code review: Self review of code helps in correct and identifying syntax errors.
- Testing: Running the code and observing it's behavior can help to catch errors and correct error.

④ what is the difference bw a class and an object in python.

Features	class	object
Definition	Blueprint for creating objects	Instance of a class
Creation	Defined using the 'class' keyword	Created using the class constructor.
Properties	It contains methods and variables	It contains state and behavior defined by its class.
Usage	Acts as a template for creating objects	Object is instance of class with their unique data
Example	class Car:	my_car = Car()

⑤ Suppose your script attempts to print the value of a variable that has not yet been assigned a value. How does the python interpreter react?

ans→ When attempting to print the value of a variable that has not yet been assigned a value, python raise a 'NameError'. This errors indicates that the variable name is not defined in the current scope. It does not have a value associated with it. So trying to access it results in a runtime error.

⑥ Discuss four string manipulations methods.

(i) str.upper() → Convert all characters to uppercase
(ii) str.lower() → Convert all characters to lowercase
(iii) str.find → find a characters with in string
(iv) str.count → count no of characters
(v) str.strip → removes whitespace from a string

⑥ In what way is a recursion design different from a top down design?

Features	Recursive Design	Top-down Design
Definition	A problem-solving approach where a function calls itself	A problem solving approach where the problem is broken down into smaller subproblems
Implementation	Implemented using recursive functions	Implemented using function decomposition.
Termination	It requires a base case to terminate recursion	Relies on breaking down the problem until it becomes trivial
Complexity	Can be complex due to function calls within function calls	less complex due to linear progress.
Error handling	Can be challenging due to nested function calls	Easier to handle error in specific component

⑦ what is object instantiation? what are the options at the programmer's disposal during the process?

ans⇒ Object instantiation is the process of creating an instance of a class also known as object.
- During this process programmers have several options
- Constructor: They can provide arguments to the class constructor to initialize the object with specific values.

Default Constructor

- Default values: They can define default values for constructor parameters, allowing instantiation without providing all arguments
- Inheritance: They can inherit attributes and methods from parent classes during instantiation.

(8) Describe two fundamental differences b/w terminal based user interface and GUIs.

features	Terminal based UI	Graphical user interface
Interaction	Interaction through text input and output	interaction through graphical elements such as buttons and windows.
Complexity	Generally simpler and more lightweight	More visually and complex
Input methods	keyboard	keyboard and mouse
Speed	faster	slower
Information display	Text-based	text, images, videos, buttons, forms.

(9) Explain relationship between a function and its arguments.

ans ⇒ A function is a block of reusable code that performs a specific task. Arguments or parameters are variables passed to a function to provide it necessary information for execution. The relationship between a function and its arguments is that the function receives input from its arguments, process that input and return a result on the basis of provided arguments.

(10) what happens when the print function prints a string literal with embedded newline characters

ans ⇒ when the print function encounters a string literal with embedded newline characters Ex = ("\n") it interprets them as instructions to move to the next line before continuing to print. Therefore each occurrence of "\n" in the string will cause the subsequent text to be printed on a new line

(11) when would you make a data field read-only and how would you do this?

ans ⇒ we would make a data field read-only when I want to restrict external code from modifying its value directly, ensuring that it remains unchanged after initialization.

Ex ⇒

```
Class Myclass:
    def __init__(self, value):
        self.value = value

    @property
    def value(self):
        return self.value

Obj = Myclass(20)
print(obj.value) # 20
obj.value = 30 # AttributeError
```

(12) what is slicing in python? Explain with example

ans ⇒ slicing in python refers to the technique of extracting a portion of a sequence of string, number, by specifying a range of indices.

Syntax: Sequence[start:stop:step]

Example:

```
text = "Hello, world!"
substring = text[7:12] # world
print(substring)        # 12-1
# slicing with step.
numbers = [0,1,2,3,4,5,6,7,8,9]
even_num = numbers[::2]
print(even_num) #[0,2,4,6,8]
```

(13) why is the "pass" keyword used for in python

ans ⇒ The "pass" keyword in python is a null operation keyword used it does nothing when executed. It is often used as a placeholder where syntactically as placeholder required but no action is needed. It allows for future you to create empty code blocks without causing a syntax error.

Ex ⇒

```
if condition:
    pass # placeholder for future code
else:
    print("condition is false")
```

pass statement executed, nothing happens but you avoid getting an error when empty code is not allowed in the code

(14) what are iterators in python?

ans ⇒ Iterators in python are objects that allows you to traverse a container such as list, tuple, dictionary) one element at a time. Iterators implement two methods --iter--() and --next--() which allows iteration over the containers.

in other words ⇒

```
My_list = [1,2,3]
My_iter = iter(my_list)
print(next(my_iter)) # output 1
print(next(my_iter)) # output 2
print(next(my_iter)) # output 3
```

(15) what is the ~~scope~~ scope of a variable? Give an example?

ans ⇒ The scope of a variable in python refers to the region of code where the the variable is accessible. Local variables are accessible only within the block of code where they are defined, while global variables are accessible throughout the entire program.

```
def my_function():
    x = 10
    print("value of x:", x)

x = 20 → global scope of x
print("value of x:", x) # 20
my_function() # 10
```

local scope of x

(16) why do we use join() function in python?

ans ⇒ The join() function in python is used to concatenate element of an iterable (such as list, tuple) into a single string using a specified separator.

Ex ⇒

```
my_list = ["apple", "banana", "orange"]
result = ", ".join(my_list)
print(result) # apple, banana, Orange
```

join in py is an in-built method used to join an iterable's elements, separated by a separator which is specified by you.

9 Marks

1 (a) with the help of code snippets:
(a) Differentiate b/w structural equivalence and object identity.
(b) Differentiate b/w function and method

(a)	Structural Equivalence	object identity
(i)	Two objects are considered equivalent if they have the same structure and content	Two objects are considered identical if they refer to the same memory location
(ii)	Ex⇒ list1 = [1, 2, 3] list2 = [1, 2, 3] print(list1 == list2) # o/p: True	list1 = [1, 2, 3] list2 = list1 print(list1 is list2) # o/p: True

(b)	function	method
(i)	A block of code that performing a specific task	A function associated with an object (belonging to a class)
(ii)	called by its name	called on an object using dot notation

(iii)	def my_fun():
	print("Hello")
	
	my_fun()
	# o/p: Hello
		class myclass:
		def my_fun(self):
		print("Hello")
		
		obj = myclass()
		obj.my_fun() # Hello

2 Develop a code that inputs a text file. The code should print the unique words in the file in alphabetical order. The code should print the no of characters and digits

```
def count_character_digits(file_path):
    unique_words = set()
    num_char = 0
    num_digits = 0
    with open(filepath, 'r') as file:
        for line in file:
            for word in line.split():
                unique_words.add(word)

                for char in word:
                    if char.isalpha():
                        num_chars += 1
                    elif char.isdigit():
                        num_digits += 1

    print("unique words in alphabetical order:")
    for word in sorted(unique_words):
        print(word)
    print("No of characters:", num_chars)
    print("No of digits :", num_digits)
file_path = "sample_text.txt"
count_character_digits(file_path)
```

3 Create a recursive function that accept a pathname as an argument. The pathname can be either the name of a file or the name of a directory. If the pathname refers to a file, its name is displayed, followed by its contents. Otherwise, if the pathname refers to a directory, the function is applied to each name in the directory. Test this function in a new program.

os.path.isfile(path)
os.path.isdir(path)
os.listdir(path)

Page
Date / /

cont>
```
import os
def display_path_contents(path):
    if os.path.isfile(path):
        print("File:", path)
        with open(path, 'r') as file:
            print(file.read())

    elif os.path.isdir(path):
        print("Directory:", path)
        for item in os.listdir(path):
            display_path_contents(os.path.join(path, item))

    else:
        print("Invalid path")

path = "Example_directory"
display_path_contents(path)
```

④ write a python code to calculate income tax based on the user's input

ans⇒
```
def calculate_tax(income):
    if income <= 18900:
        tax = 0
    elif income <= 45000:
        tax = (income - 18,900) * 0.19
    elif income <= 120000
        tax = 5,092 + (income - 45000) * 0.32
    else: tax = 29467 + (income - 120000) * 0.37
    Print("Your income tax is: $", tax)
calculate_tax(float(input("Enter your income:")))
```

Page
Date / /

⑤ what are the different types of loops and selection statements available in python?

1. loops:
   - for loop: Executes a * while
   - (ii) selection statements →
   - if statement • if-else statement
   - if elif statement • nested if statement
   - ternary operator (conditional expression)
   - switch statement

for loop →
```
for i in range(5):
    print(i)
```

while loop →
```
n = 0
while n<5:
    print(n)
    n += 1
```

if →
```
x = 10
if x > 0:
    print("x is positive")
```

if-else - statement
```
y = -5
if y > 0:
    Print("y is positive")
else:
    print("y is non-posit")
```

⇒ if-elif-else statement
```
grade = 85
if grade >= 90:
    print('A')
elif grade >= 80:
    Print('B')
elif grade >= 70:
    print('C')
else:
    print('D')
```

Nested if state→
```
a = 10
b = 5
if a > 0:
    if (b > 0):
        print("both a and b are positive")
```

*Switch statement: Not directly available in python but can be complemented using dictionaries or if-elif-else statement.

```
def switch_case(argument):
    switch = { 1:"case1",
               2:"case2",
               3:"case3"
    return switch.get(argument, "Invalid case")
print(switch_case(2)) # case2
```

Ternary operator

```
age = 25
adult = "yes" if age >= 18 else "NO"
print("Is the person adult?", adult)
```

Q6 write a python program to approximate the square root of a given number using a while loop.

```
def squareroot(number):
    guess = 1.0
    while abs(guess * guess - number) > 0.0001:
        guess = (guess + number/guess) / 2

    print("The square root of", number, "is approximately:", guess)

squareroot(89)  # 6.011...
```

Q7 Explain the concept of recursive functions in python and provide an example that demonstrate their usage. Discuss the key elements required for designing recursive functions and highlight the importance of defining base cases.

ans⇒ Recursive functions in python are functions that call themselves within their definition. They are powerful tools for solving problems that can be broken down into smaller, similar subproblems. Key ele for designing recursive fun^{n}.

① Base case: This is the terminating condition that prevents the function from calling itself indefinitely. It is crucial to define a base case to ensure that the recursion stops at some point.

② recursive case: This is where the function calls itself with a small or simple input, moving progress towards the base case.

③ Progress Towards Base Case: Each recursive call should move the function closer to the base case. otherwise the recursion may result in infinite loop.

④ Return statement value: The return value of the function should be defined in terms of the base case and the recursion case.

Example:

```
def factorial(n):
    if n==0:        # -> Base case
        return 1
    else:           # -> recursive case
        return n * factorial(n-1)

number = int(input("Enter a number:"))
print("factorial of", number, "is", factorial(number))
```

Q8 with a suitable program, elaborate compilation and linking process in python.

ans⇒ python is an interpreted language so it doesn't have a compilation process; python code can be compiled to bytecode which is then interpreted by the python runtime.

The compilation and linking process

1. Compilation to Bytecode: when you write a python script (.py file) it first needs to be compiled to bytecode. This process is handled by the python interpreter when you run the script.

The bytecode is stored in ~~__pycache__~~ __pycache__ directory as (.pyc) files for future use, improving the execution speed on subsequent runs.

2. Interpreter Execution: The bytecode generated in the compilation step is executed by the python interpreter. The interpreter reads the bytecode instructions and executes them one by one.

3 Dynamic Linking: Python also supports dynamic linking through module imports. When you import a module, python searches for it in the directories once found, python dynamically loads the module and links it to your script.

module.py
```
def greet(name):
    print("Hello", name)
```
main.py
```
import module
module.greet("Alice")
```
When you run main.py what happens

① Compilation: Both module.py and main.py are compiled to bytecode by the python interpreter.

② Interpreter Execution: The bytecode of main.py is executed by the python interpreter. It encounters the import module statement, so it searches for module.py in the directories listed. Compiles it if necessary and loads it into memory.

③ Dynamic Linking: The import module statement dynamically links main.py to module.py allowing main.py to call functions defined in module.py.

④ Execution: The module.greet("Alice"). statement in main.py is executed resulting in the o/p: "Hello, Alice"

⑨ write a program to accept a number from 1 to 12 and display the name of the month and days in that month like 1 for january and the number of days 31 and so on.

```
month_num = int(input("Enter a no from 1 to 12:"))
```
ans ⇒
```
month = ["january", "february", "March", "April",
"May", "June", "July", "August", "September",
"October", "November", "December"]

days_in_month = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]

if month_num >= 1 and month_num <= 12:
    month_name = month[month_num - 1]
    no_of_days = days_in_month[month_num - 1]
    print("Month:", month_name)
    print("No of days:", no_of_days)
else:
    print("Invalid input")
```

⑩ write a python program to search a specific part of a string for a substring.

ans ⇒
```
main_string = "This is a sample string"
substring = "sample"
if substring in main_string:
    print("substring found at index:",
    main_string.index(substring))
else:
    print("substring not found.")
```

(11) How are classes created in python? explain with coding example.

ans →

```
Class Car:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model

    def drive(self):
        print(f"Driving {self.brand} {self.model}")

my_car = Car("Toyota", "Corolla")
my_car.drive()
```

(12) what is the usage of help() and dir() function in python? Give programming example.

(i) help(): This function is used to get help information about modules, functions, classes and methods in python. It prints out a help page containing information about the specified object.

(ii) dir(): This function returns a list of valid attributes for the specified object. It is used to get a list of attributes and methods associated with an object.

Ex →

```
Class MyClass:
    def __init__(self, name):
        self.name = name
    def greet(self):
        print("Hello", self.name)
obj = MyClass("Rayhan")
print("Attributes and methods of obj:")
print(dir(obj))
print("Help information about the greet method")
help(obj.greet)
```

(13) Assume that a file containing integers separated by newlines. write a code segment that opens the file and prints the average value of the integers.

```
with open("file.txt", "r") as file:
    numbers = [int(line) for line in file]
    average = sum(numbers) / len(numbers)
    print("Average value:", average)
```

(14) write a pro that computes and prints the average of the numbers in a text file. you should make use of two heigher order function to simplify the design

```
def average(numbers):
    return sum(numbers) / len(numbers)

def read_numbers(filename):
    with open(filename, "r") as file:
        return [int(line) for line in file]

def calculate(filename):
    numbers = read_numbers(filename)
    avg = average(numbers)
    print("Average:", avg)

calculate("file.txt")
```

Page :
Date : / /

(Q15) write a while loop in py that computes the factorial of a given integer N.

```
def factorial(N):
    result = 1
    while N > 1:
        result = result * N
        N = N - 1
    return result
print("factorial of", N, "is :", factorial(N))
```

(Q16) class B extends class A. class B defines an str method that returns the string representation of its instance variable. class B defines a single instance variable named age, which is an integer. write the code to define the str method for class B. This method should return the combined string information from both classes. Label the data for age with the string "Age:"

ans:

```
class A:
    def __init__(self, name):
        self.name = name
class B(A):
    def __init__(self, name, age):
        super().__init__(name)
        self.age = age
    def print_string(self):
        return f"Name is: {self.name}, age: {self.age}"
obj = B("Raushan", 18)
print(obj.print_string)
```

(Q17a)

Q12 Why is it a good idea to write and test the code for laying out a window's components before you add the methods that perform computations in response to events.

(1) Focus on separation of concern: Separating layout logic from event handling keeps your code organized. You can focus on the visual structure of the window without worrying about how components will react to user interaction.

(2) Easier Testing: Testing the layout is simpler, you can visually verify the placement and arrangement of components without needing to simulate user events. This allows you to identify and fix layout issue early on.

(3) Debugging Efficiency: If you encounter issues later, then the problem became easier. You can determine if it's a layout problem or an event handling by testing each part independently.

(4) Reusable layouts: By testing the layout independently, you can ensure it works consistently. This makes the layout reusable across different window or scenarios where you might need the same visual structure.

12 Marks

① Result Preparation system for 20 students data includes Name, Rollno, Marks in 3 subjects. The percentage and grade are to be calculated from following info

marks → 80 to 100 — A
60 to 80 — B
45 to 60 — C
Less than 45 — D

```
Class Student:
    def __init__(self, name, rollno, marks):
        self.name = name
        self.rollno = rollno
        self.marks = marks
        self.percentage = self.calculate_percentage()
        self.grade = self.calculate_grade()

    def calculate_percentage(self):
        total_marks = sum(self.marks)
        percentage = (total_marks)/(3*100))*100
        return percentage

    def calculate_grade(self):
        if(self.percentage<=100 and self.percentage>=80):
            return 'A'
        elif(self.percentage<=80 and self.percentage>=60):
            return 'B'
        elif(self.percentage<=60 and self.percentage>=45):
            return 'C'
        else:
            return 'D'

student_data = [
{"name": "Raghav", "roll_no": 101, "marks": [65,75,90]},
{"name": "Emma", "roll_no": 102, "marks": [70,80,65]},
  # Add data for other student
]

students = []
for data in student_data:
    student = Student(data["name"], data["roll_no"],
                      data["marks"])
    students.append(student)

for student in students:
    print(f"Name: {student.name}, Roll No:
    {student.rollno}, Percentage: {student.percentage}%,
    Grade: {student.grade}")
```

Q. Elaborate the concept of dictionary in python
Python. How will you add and access ele to a Dictionary? write a program to concatenate the following dictionary to create a new one.

```
#ADD Element →
my_dic = {'Apple': 5, 'banana': 10, 'orange': 3}
my_dic =
my_dic['grape'] = 8
my_dic['banana'] = 12
# Accessing element
print(my_dic['apple']) # 5
print(my_dic['banana'])
print(my_dic.get('banana')) #10
print(my_dic.get('orange')) #3
```

Concatenate the dictionary

```
dic1 = {1: 10, 2: 20}
dic2 = {3: 30, 4: 40}
dic3 = {5: 50, 6: 60}
result dict = {}
for d in (dic1, dic2, dic3):
    result_dic.update(d)
print("Dictionary :", result_dic)
# O/P : {1:10, 2:20, 3:30, 4:40, 5: 50, 6:60}
```

Q3 write python program that accept a string and calculate the number of digits and letters. Sample Data : python 3.2
Expected O/p : letters = 6
digit = 2

```
def count_letter_digit(string):
    digit = 0
    letter = 0
    for char in string:
        if char.isdigit():
            digit += 1
        elif char.isalpha():
            letter += 1
    return letter, digit
sample = "python 3.2"
letter, digit = count_letter_digit(sample)
print("Letters:", letter)
print("Digits:", digit)
```

Q4 Write a script named diff.py This code should prompt the users for the names of two text files and compare the contents of the two files to see if they are the same. if they are not, the script should output "N", followed by the first line of each file that differ from each other. The input loop should read and compare lines from each file. The loop should break as soon as a pair of different lines is found.

```
def compare_files(file1, file2):
    with open(file1) as f1, open(file2) as f2:
        for line1, line2 in zip(f1, f2):
            if line1 != line2:
                return False, line1.strip(), line2.strip()
    return True, None, None

file1 = input("Enter first file :")
file2 = input("Enter second file :")
are_equal, line1, line2 = compare_files(file1, file2)
if are_equal:
    print("Yes")
else:
    print("No")
    print("First Difference found: {line1}, {line2}")
```

Q5 Define and test myrange function.

```
def myRange(start, stop=None, step=1):
    if stop is None:
        stop = start
        start = 0
    result = []
    current = start
    while (step > 0 and current < stop) or (step < 0 and current > stop):
        result.append(current)
        current += step
    return result
```

```
print(myRange(5)) # [0, 1, 2, 3, 4]
print(myRange(1, 5) # [1, 2, 3, 4]
print(myRange(1, 10, 2)) # [1, 3, 5, 7, 9]
```

Q6 Design a program in python to accomplish the following tasks.

(i) Read a text file containing a paragraph of text.

(ii) Implement a loop to iterate over each word in the paragraph to count the occurence of each unique word and store the word count in a dictionary.

(iii) write the updated word count dictionary to a new text file, with each word and its count on a separate line.

(i)
```
def read_paragraph(file):
    with open(file,'r') as f:
        paragraph = f.read()
    return paragraph
```

(ii)
```
def count_words(paragraph):
    word_count = {}
    words = paragraph.split()
    for word in words:
        if word in word_count:
            word_count[word] += 1
        else:
            word_count[word] = 1
    return word_count
```

(iii)
```
def write_word_count(word_count, outputfile):
    with open(outputfile,'w') as f:
        for key,val in word_count.items():
            f.write(f"{key}:{val}\n")
```

Q7 Define and test a function myrange. This function should behave like python's standard range function, with the required and optional arguments, but it should return a list. Do not use the range function in your implementation.

```
def myrange(start, stop=None, step=1):
    if stop is None:
        stop = start
        start = 0
    result = []
    curr = start
    while curr < stop if step > 0 else curr > stop:
        result.append(curr)
        curr += step
    return result
print(myrange(5))
print(myrange(1,5))
print(myrange(1,20,2))
print(myrange(5,1,-1))
```