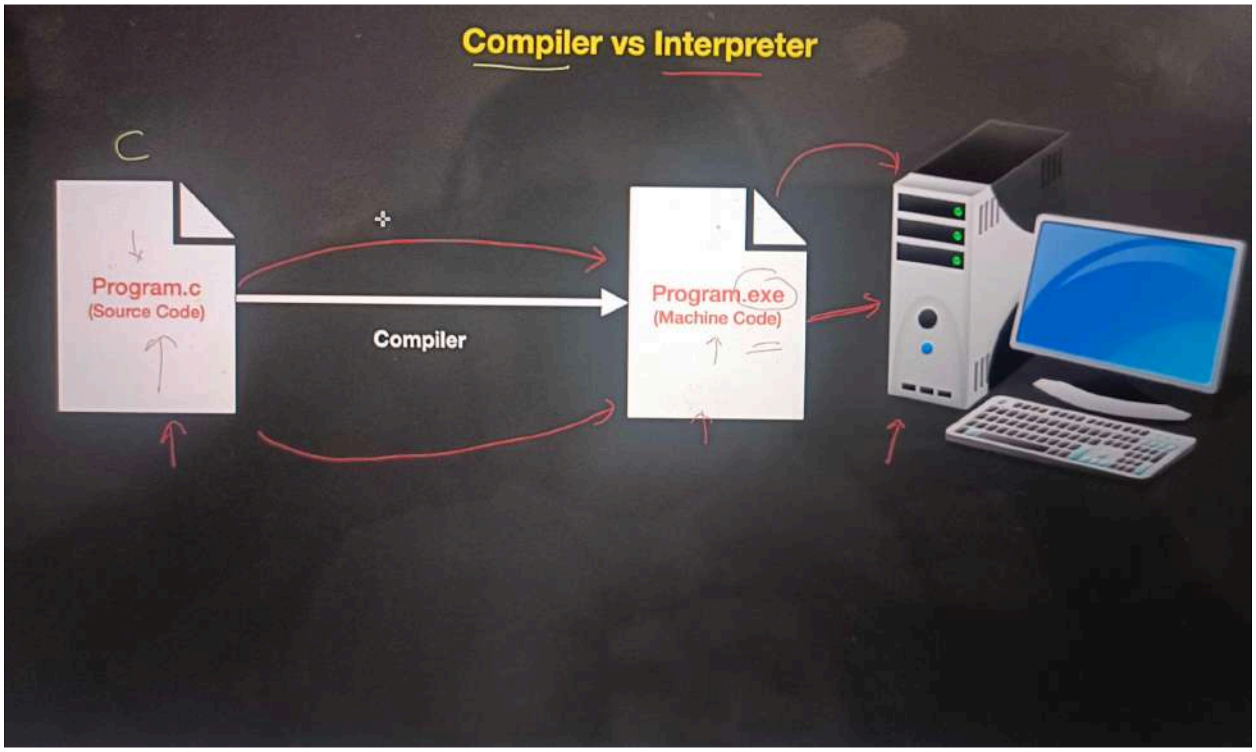
Introduction to python

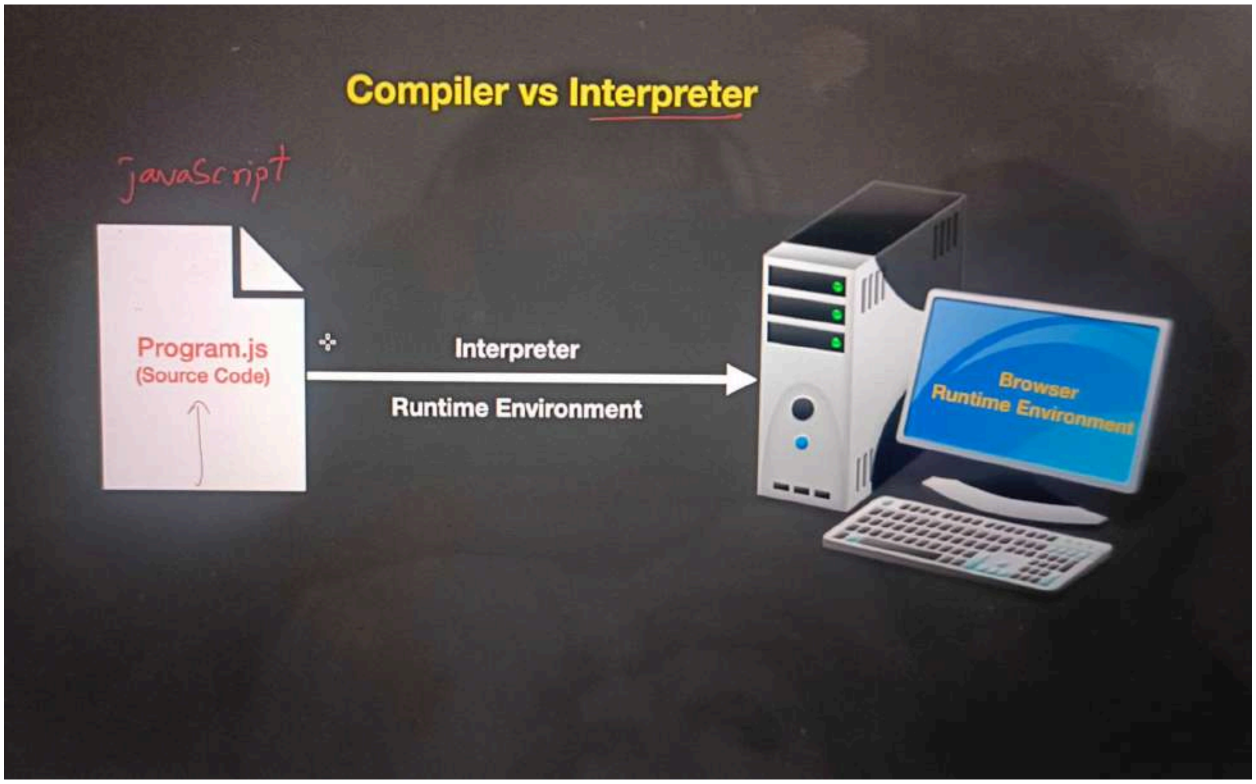
1. **Types of programing languages**
   a**)** Low level language : machine level and assembly
   b) High level language: compiler based, interpreter based, hybrid
   Note: python will come under Hybrid

2. **Compiler VS interpreter**
   a**)** Compiler is used to translate to exe file which machine will understand
   b) Interpreter will directly execute its runtime environment
   Note: (for ex in Interpreter when we run code it will take single piece of code or all code and directly execute it in machine code but in compiler will convert first to exe and we have to execute when we want)
   c. Example javascript run inside browser so here browser is interpreter

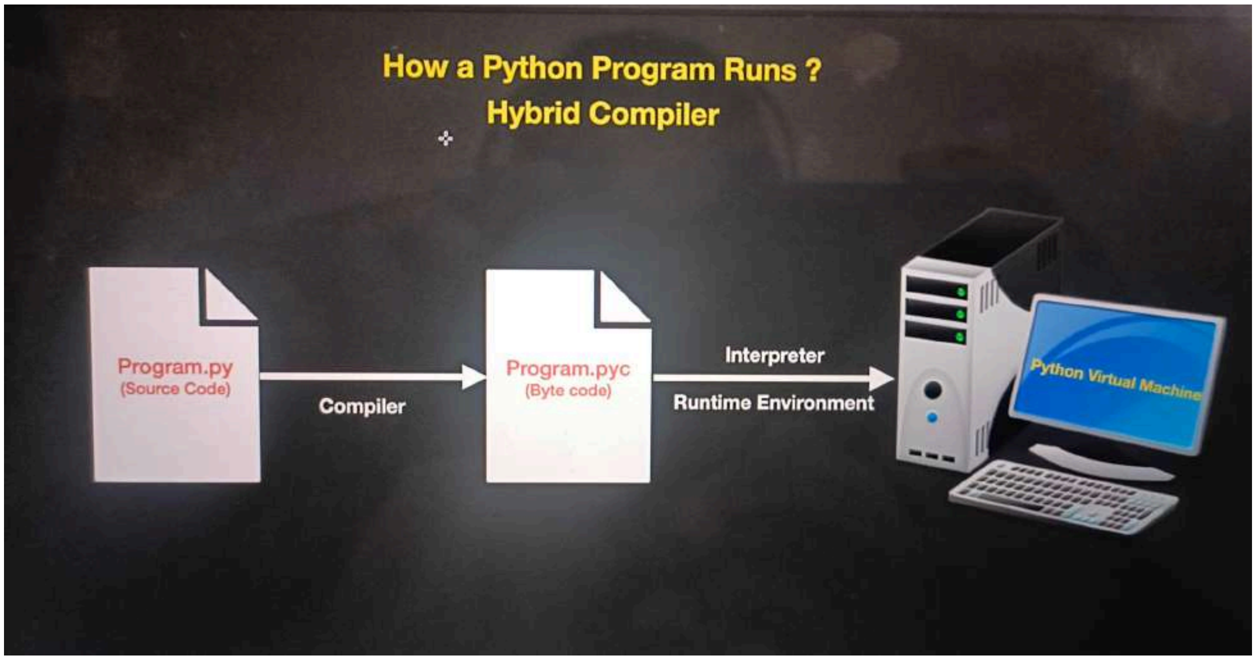
Compiler



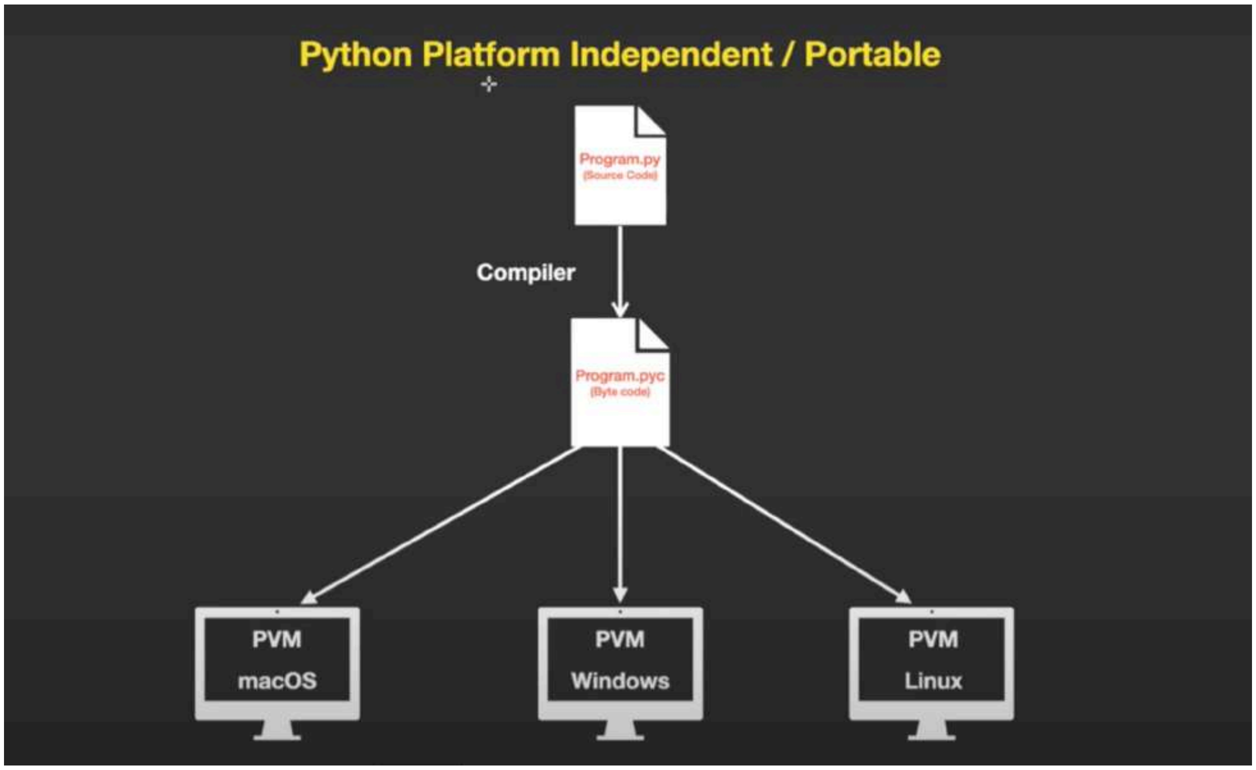
Interpreter



3. **Hybrid Compiler**
   a. When we run python code it will convert source code to intermediate code called bytecode and then byte code will convert to machine code through interpreter (PVM)
   b. Byte code will create as hidden file in memory in the file extension program.pyc
   Ex: java, python



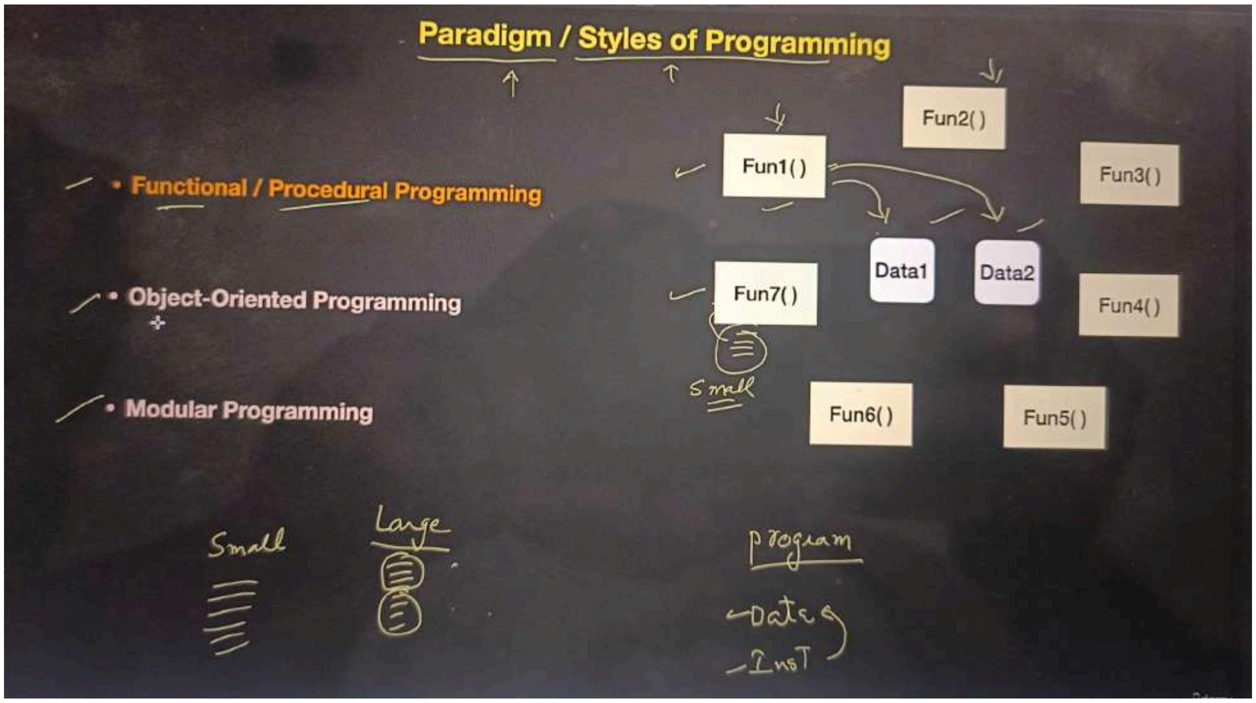
4. **Python platform independent**
   Python will run on different different platform for same code



5. **Paradigm / style of programming**
   a) When we write large number of code its difficult to manage hence we have to adopt bellow methods to organize our code its called as Paradigm
   b) Python support bellow all Paradigm

   <u>Paradigm have various methods</u>

   a. Functional / procedural programing :
   a.1) large program broken down into functions, function is piece of code which perform specific task
   a.2) it is function centric more focus on function

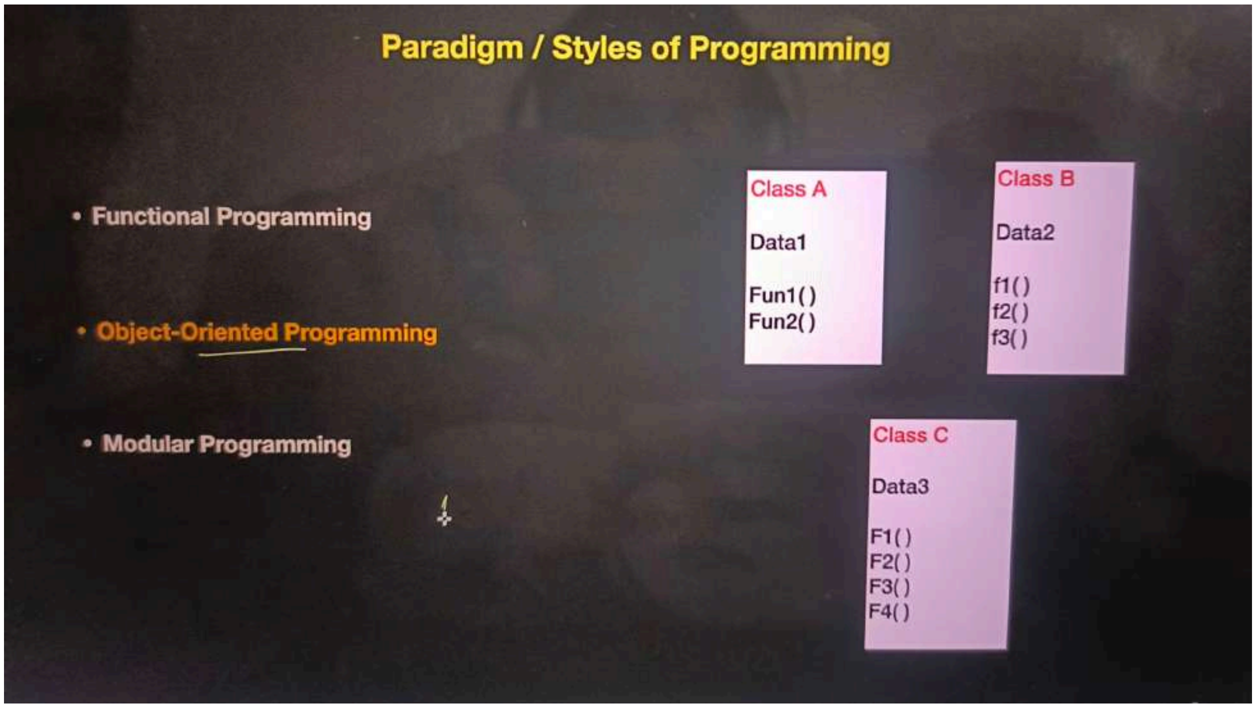

Paradigm / Styles of Programming

- Functional / Procedural Programming
- Object-Oriented Programming
- Modular Programming

Fun1()
Fun2()
Fun3()
Fun4()
Fun5()
Fun6()
Fun7()
Data1
Data2
Small
Small
Large
program
Data
Inst

b. Object oriented programing

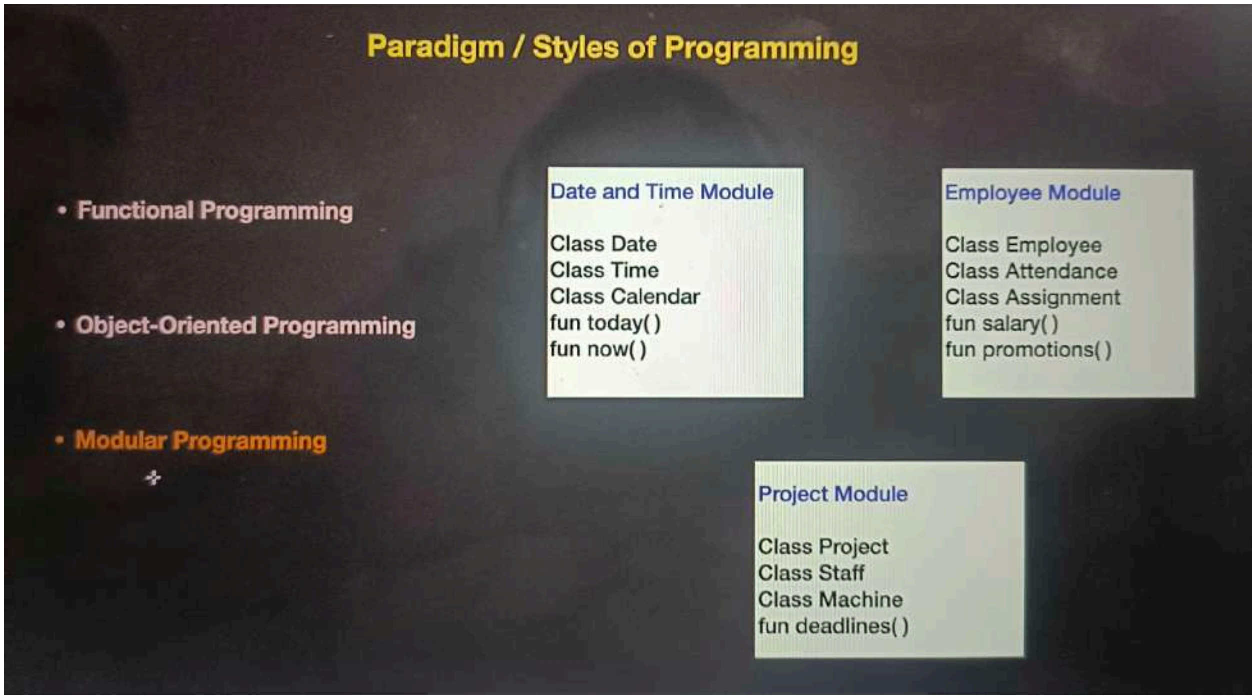
b.1) We have a class called A the class contain Data1 and function the function used to call data


Paradigm / Styles of Programming

- Functional Programming
- Object-Oriented Programming
- Modular Programming

Class A
Data1
Fun1()
Fun2()
Class B
Data2
f1()
f2()
f3()
Class C
Data3
F1()
F2()
F3()
F4()

c. modular programing:
a big program break down into modules and module may contain class and functions together



Paradigm / Styles of Programming

- Functional Programming
- Object-Oriented Programming
- Modular Programming

Date and Time Module

Class Date
Class Time
Class Calendar
fun today( )
fun now( )

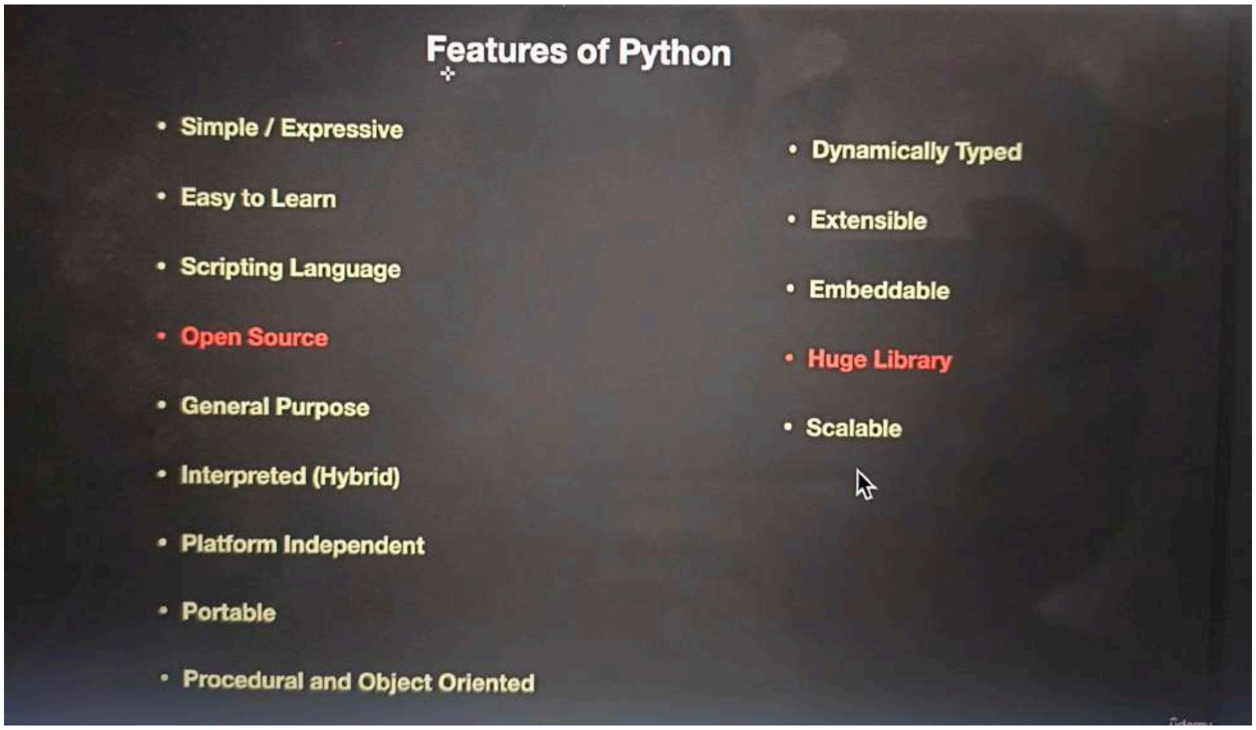
Employee Module

Class Employee
Class Attendance
Class Assignment
fun salary( )
fun promotions( )

Project Module

Class Project
Class Staff
Class Machine
fun deadlines( )

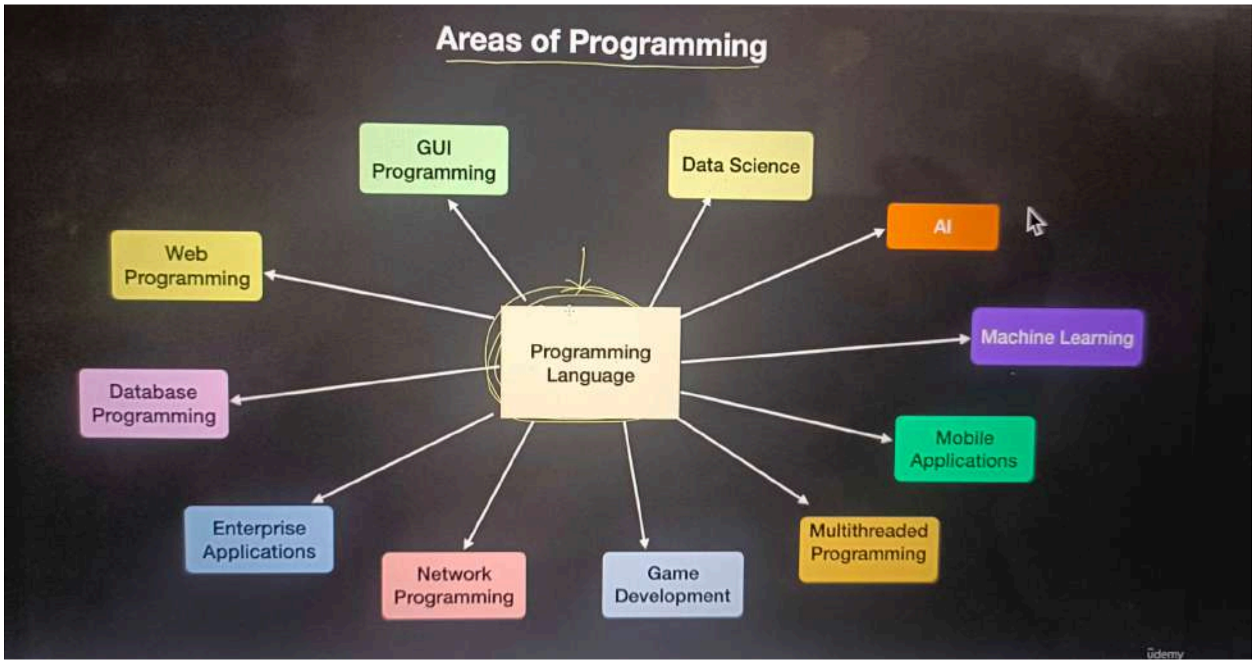
6. **Features of python**



Features of Python

- Simple / Expressive
- Easy to Learn
- Scripting Language
- Open Source
- General Purpose
- Interpreted (Hybrid)
- Platform Independent
- Portable
- Procedural and Object Oriented
- Dynamically Typed
- Extensible
- Embeddable
- Huge Library
- Scalable

7. **Areas of programing**



Python installation and Setup

1. **Installing python**
2. **Installing pycharm**
3. **Installing VS code**
4. **Installing and using jupyter Notebook**

Python DataTypes

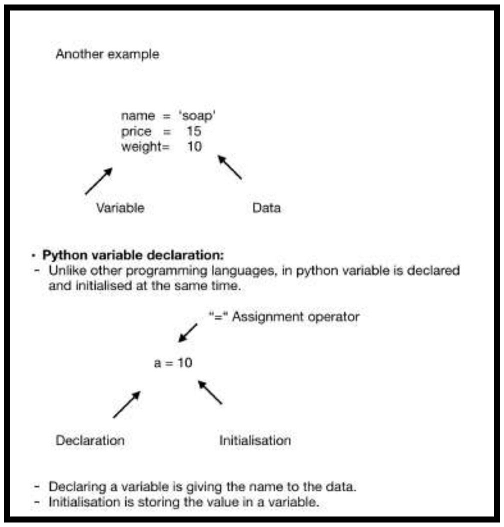
1. Section introduction

1.a) Numeric Types
2.b) variables
3.c) Literals
4.d) conversions

2. What are variables

a.) Variables are names given to data or referral to data
b.) Variables are the names given to the data stored in a memory location.
Ex a=10, name='soap'
Note: in above example 'a' called as declaration and assigning value to a is called as initialization

Declaration of multiple variables
a, b, c=15, 10, 12 or X=Y=Z=1
Name, price, weight='cv', 50, 100



Another example

```
name  =  'soap'
price  =   15
weight=    10
```

Variable

Data

- **Python variable declaration:**
  - Unlike other programming languages, in python variable is declared and initialised at the same time.

"=" Assignment operator

```
a = 10
```

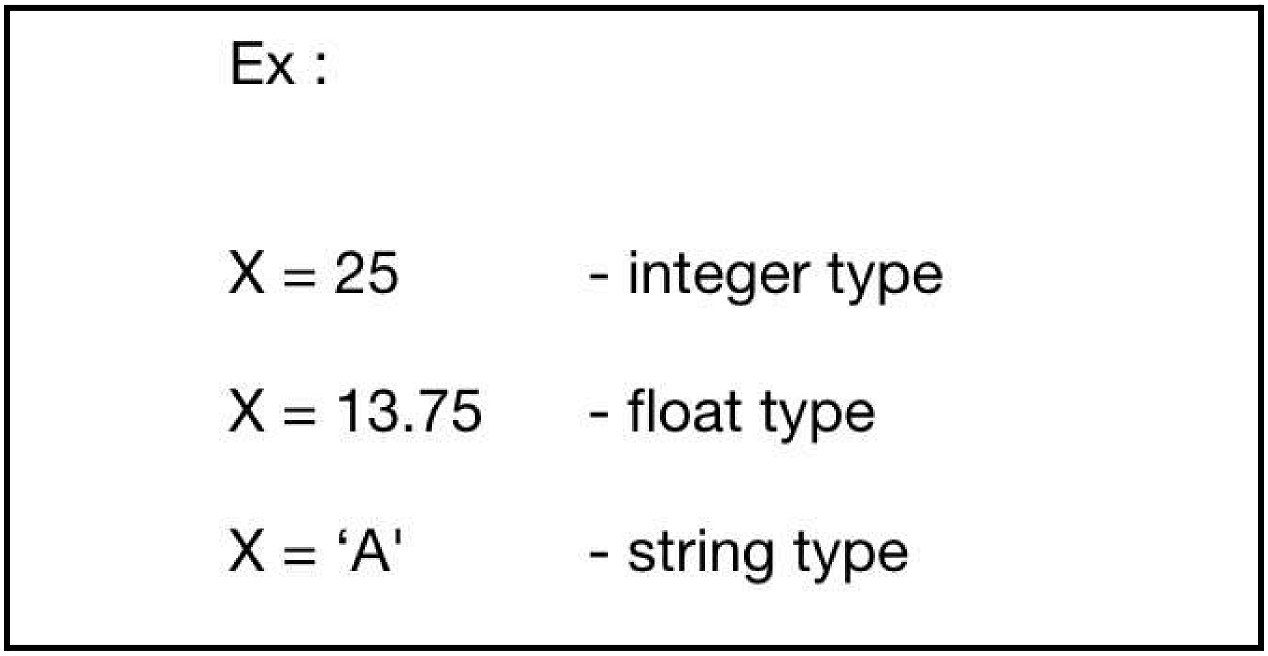
Declaration

Initialisation

- Declaring a variable is giving the name to the data.
- Initialisation is storing the value in a variable.

3. **Dynamically typed language:** Python is called a dynamically typed language because you don't have to declare the data type of a variable when you create it. Instead, the type of a variable is determined at runtime

   Variables do not have a specific data type , it's type depends on the value assigned to it



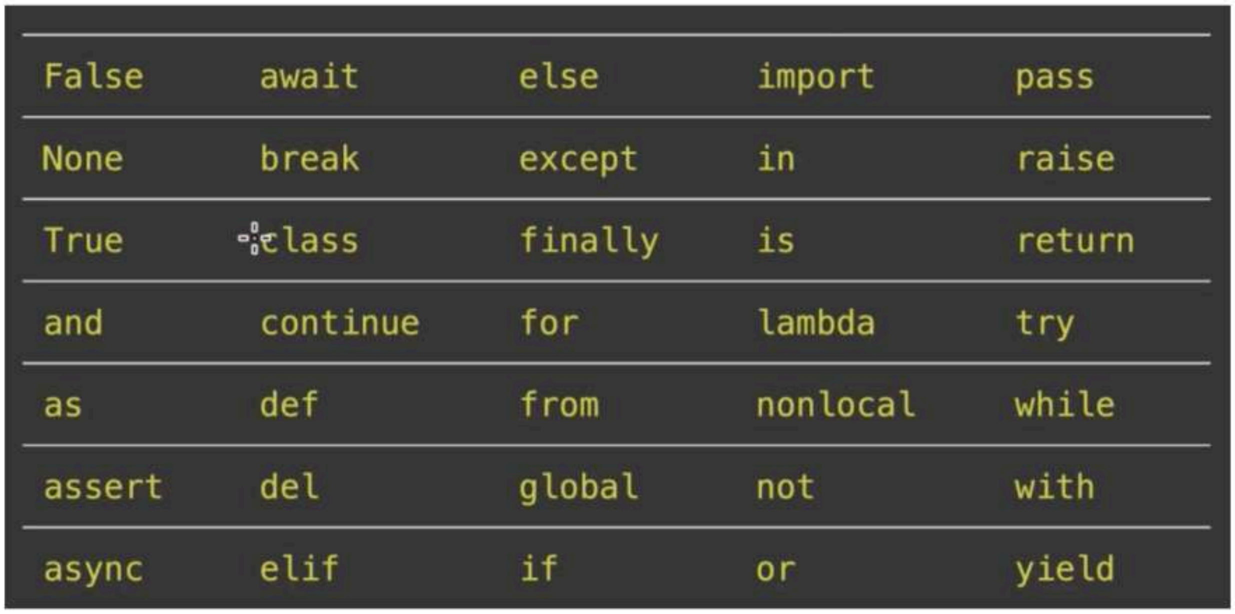
Ex :

```
X = 25          - integer type

X = 13.75       - float type

X = 'A'         - string type
```

4. **Rules for Declaring a variable**

   a) we cannot use keywords to declare a variable. These words cannot be used while declaring the variable.

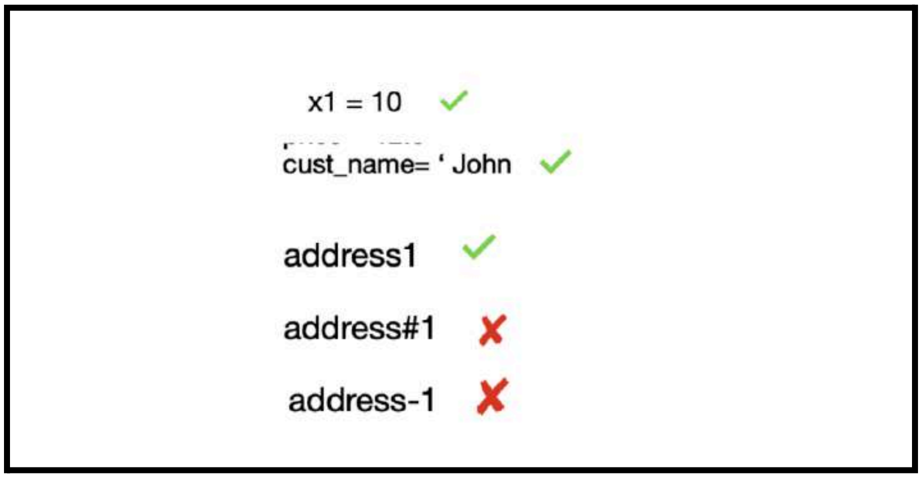
   a.1) Keywords : the words which are predefined in the language are called keywords or reserved words



False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

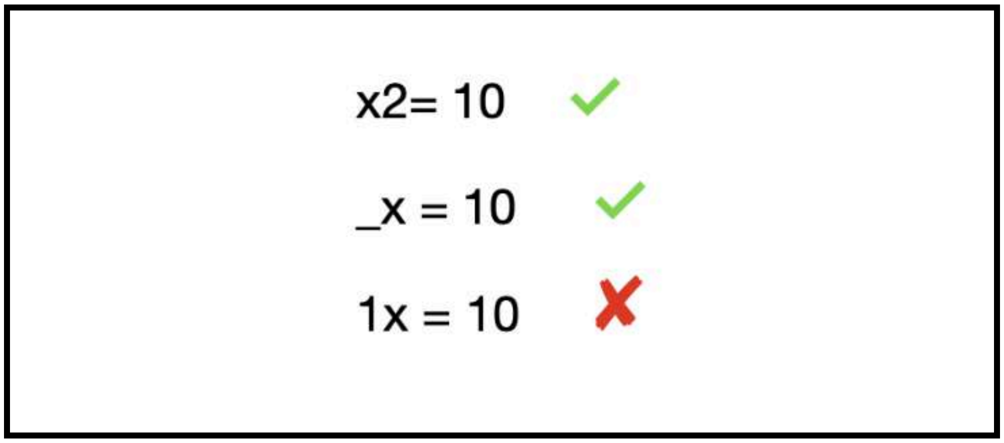
b) The first rule says that we can mix alphabet and numbers while declaring the variables, we can even use an underscore.

c) However we cannot use any special symbol like $, &,@,#,- etc…



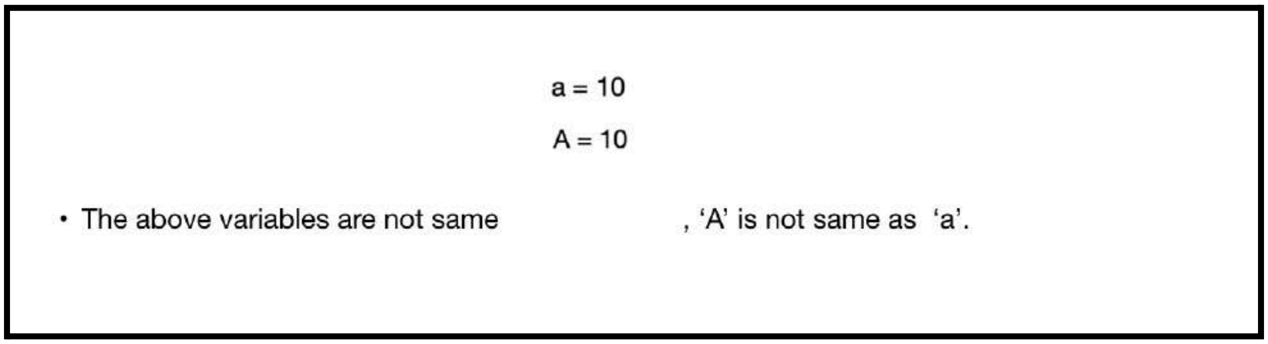
```
x1 = 10 ✓
cust_name= ' John ✓
address1 ✓
address#1 ✗
address-1 ✗
```

d) The second rule says that the variable must start with a letter or underscore character only.



```
x2= 10   ✓
_x = 10  ✓
1x = 10  ✗
```

e) The last rule states that variables are case sensitive,



```
a = 10
A = 10
```

- The above variables are not same , 'A' is not same as 'a'.

5. Python dataTypes

a) Numeric: it can support one value

a.1) int: storing integer value called integer, we can store how much lengthy we want (no sizeLimit in python) and also we can store negative value as well
EX: a=1234567890, b=10, c = -10
a.2) float: storing decimal value called float
EX: a=12.59, b= -12.59, c=12345.67890
a.3) bool: it is logical data like True and False
Note: True value is 1 and False value is 0
a.4) complex:
EX: x=3+4j, c=complex(12,9)

b) Sequence : collection of values, value could be integer, float, or char, but in sequence every values have index

b.1) List: list is collection of values here value can be modified (mutable)
b.2) tuple: tuple is also sequence of collection of value but in tuple value can not be modified (immutable)
b.3) Strings: collection of numbers symbols, words or char
b.4) Bytes: it contain one byte of data and each value should be in between 0 to 255 only it is immutable

b.5) Bytearray: it contain one byte of data and each value should be in between 0 to 255 only it is mutable

c) set : set is also collection of values but set values are not index based

c.1) Set: set is collection of values but value are un-ordered and it is mutable

c.2) frozenset: it is set of value but it is freezed value (cannot be modified) (im-mutable)

d) Dictionary:

d.1) Disct: it contain key and value, it is very useful for storing data and retrieving Data

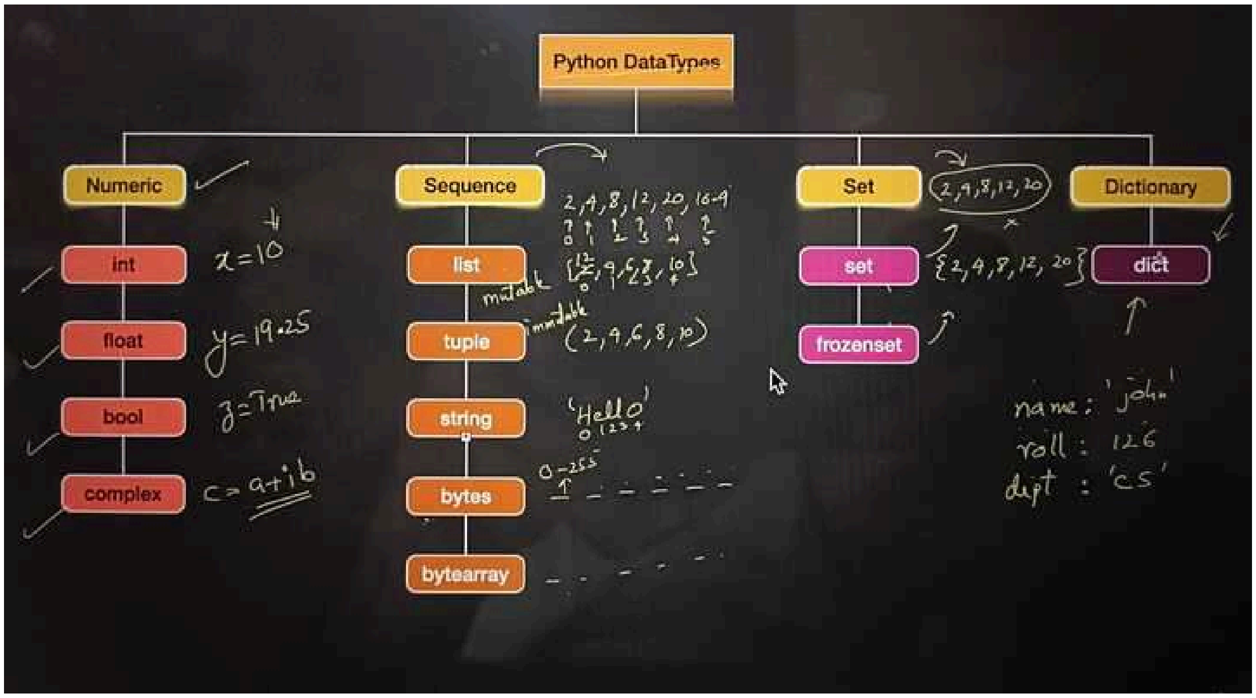
Example: name : 'cv' (name is key, value is CV)

Note:

mutable means value can be changed

im-mutable means previous value can not be modified

For Ex: first variable "A" value assigned 10, later we changed "A" value to 20, in python previous value of "A" will not changed it will store in memory but only change is "A" will start pointing to new value 20 its called **immutable**



6. Literals and constant

a) Literals: Literals are simply the raw data values that are used to represent data types. They are fixed values that are directly written in the code.

EX: Examples of literals

```
number_literal = 42  # Numeric literal
string_literal = "hello"  # String literal
boolean_literal = True  # Boolean literal
```

In these examples, 42, "hello", and True are literals representing an integer, a string, and a boolean value, respectively. They are directly written in the code and represent specific values.

b) Constants: Constants are identifiers (variable names) that are used to represent fixed values in the code. They are usually used to make the code more readable and maintainable by giving meaningful names to fixed values.

Example of constants (by convention)

```
TAX_RATE = 0.2  # Constant representing tax rate
MAX_SPEED = 100  # Constant representing maximum speed
```

In this example, TAX_RATE and MAX_SPEED are constants representing the tax rate and the maximum speed, respectively. By convention, we use uppercase names with underscores to indicate that these variables should be treated as constants

7. Base conversions

Base conversion in Python involves converting a number from one numerical base system to another

bin() converts a decimal number to its binary representation.
int() with base 2 converts a binary string to its decimal equivalent.
hex() converts a decimal number to its hexadecimal representation.
int() with base 16 converts a hexadecimal string to its decimal equivalent.

8. Type conversions

Converting one data type to another data type using below mentioned functions is called as Type conversions

1) Implicit
2) Explicit
implicit : In Implicit type conversion of data types , the Python interpreter automatically converts one data type to another without any user involvement.
Explicit : The programmer has to convert and programmer have to mention

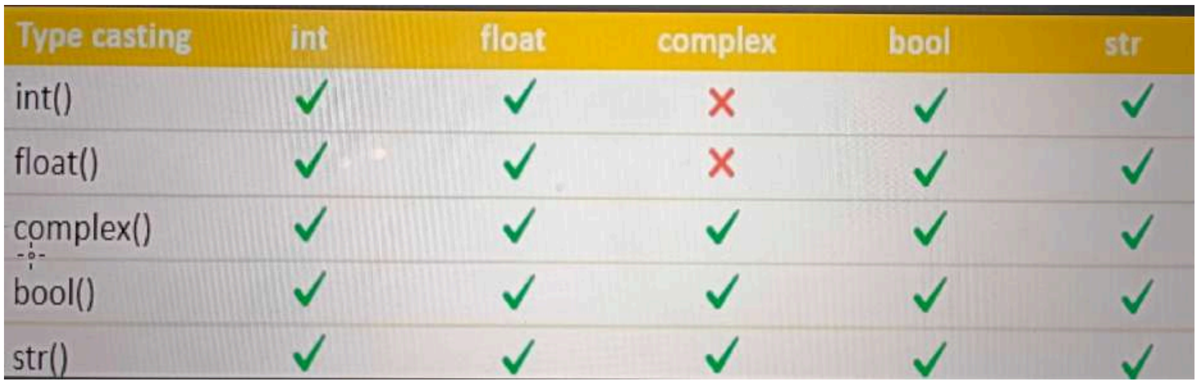
Functions: int(), float(), Bool(), complex(), str()

Note:

a) complex number cannot be converted to integer, float

b) String also cannot be converted into integer, float, complex if its alphabets but if the string variable is in digits then its possible to convert. (for ex a='1234')

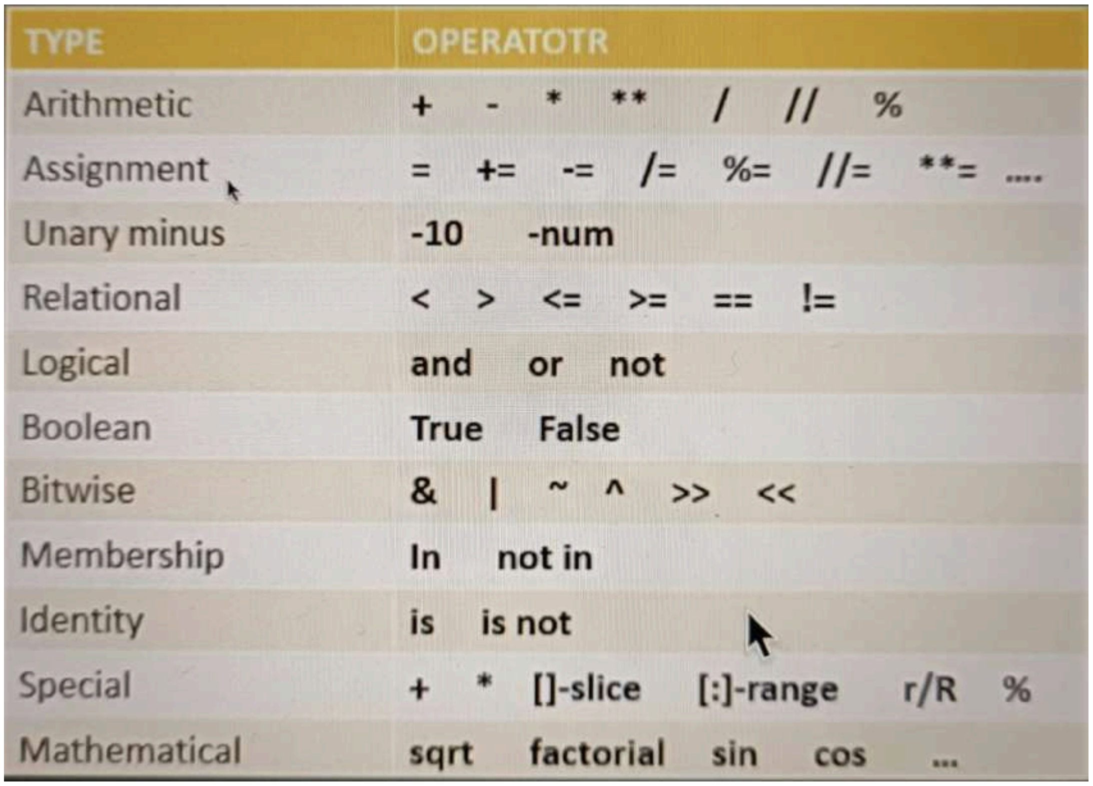
c) all data types can be converted to Bool() and str()



Type casting	int	float	complex	bool	str
int()	✓	✓	✗	✓	✓
float()	✓	✓	✗	✓	✓
complex()	✓	✓	✓	✓	✓
bool()	✓	✓	✓	✓	✓
str()	✓	✓	✓	✓	✓

Operators and Expression

1. Operators:

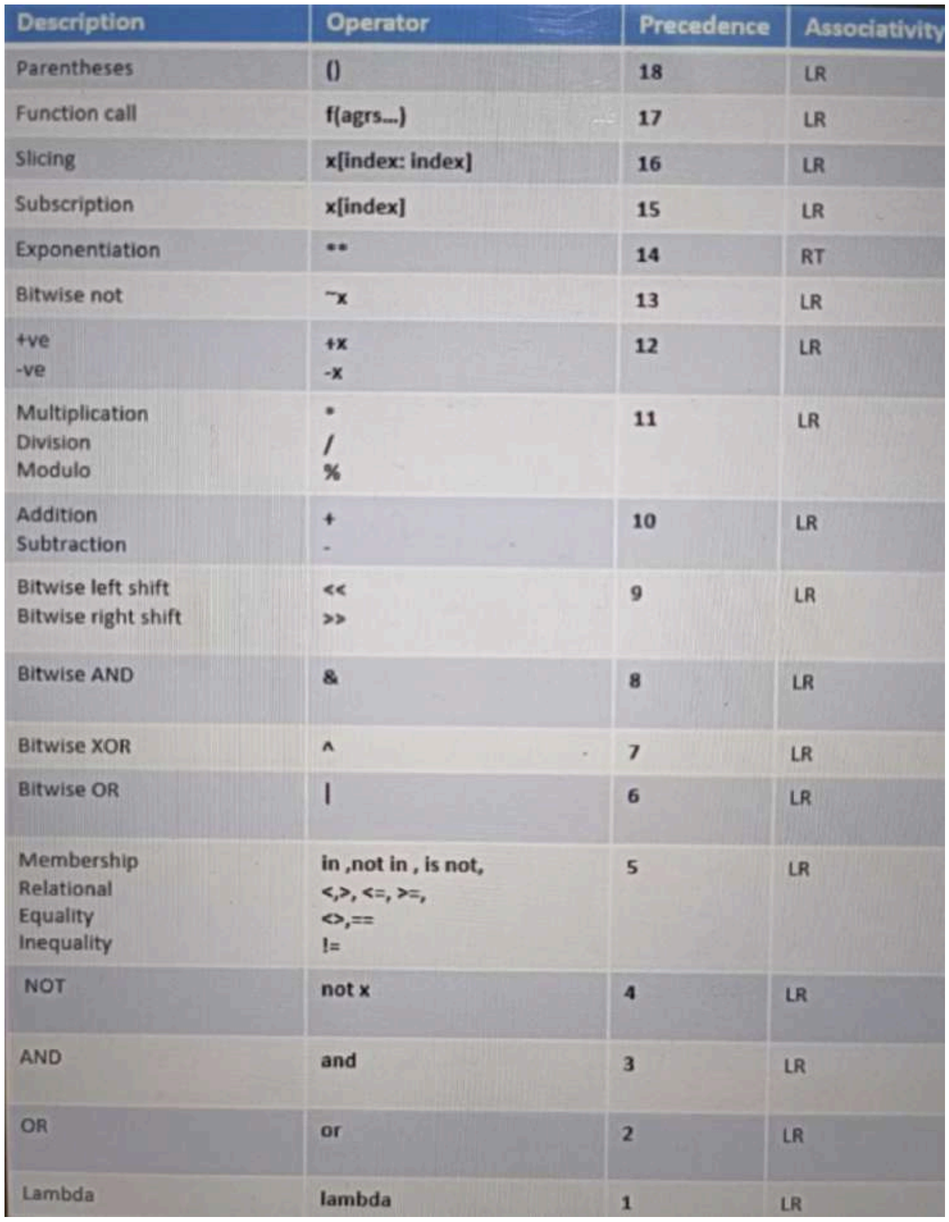


TYPE	OPERATOTR
Arithmetic	+ - * ** / // %
Assignment	= += -= /= %= //= **= ....
Unary minus	-10 -num
Relational	< > <= >= == !=
Logical	and or not
Boolean	True False
Bitwise	& | ~ ^ >> <<
Membership	In not in
Identity	is is not
Special	+ * []-slice [:]-range r/R %
Mathematical	sqrt factorial sin cos ...

2. Expression:

an expression is a combination of values, variables, and operators that evaluates to a single value. Expression in python can be used in a variety of ways, such as in assignment statements, function arguments, and conditional statements

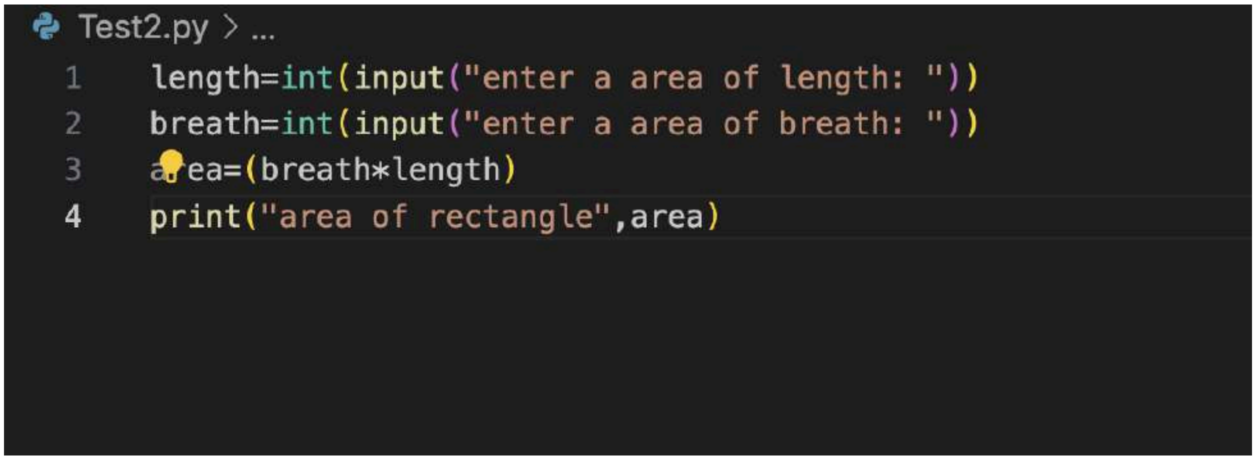
In Python, expressions are executed based on precedence, which determines the order in which operators are evaluated in an expression. Operators with higher precedence are evaluated before operators with lower precedence. If operators have the same precedence, the expression is evaluated from left to right.



Description	Operator	Precedence	Associativity
Parentheses	()	18	LR
Function call	f(agrs...)	17	LR
Slicing	x[index: index]	16	LR
Subscription	x[index]	15	LR
Exponentiation	**	14	RT
Bitwise not	~x	13	LR
+ve -ve	+x -x	12	LR
Multiplication Division Modulo	* / %	11	LR
Addition Subtraction	+ -	10	LR
Bitwise left shift Bitwise right shift	<< >>	9	LR
Bitwise AND	&	8	LR
Bitwise XOR	^	7	LR
Bitwise OR	|	6	LR
Membership Relational Equality Inequality	in ,not in , is not, <,>, <=, >=, <>,== !=	5	LR
NOT	not x	4	LR
AND	and	3	LR
OR	or	2	LR
Lambda	lambda	1	LR

3. Programs using Expressions

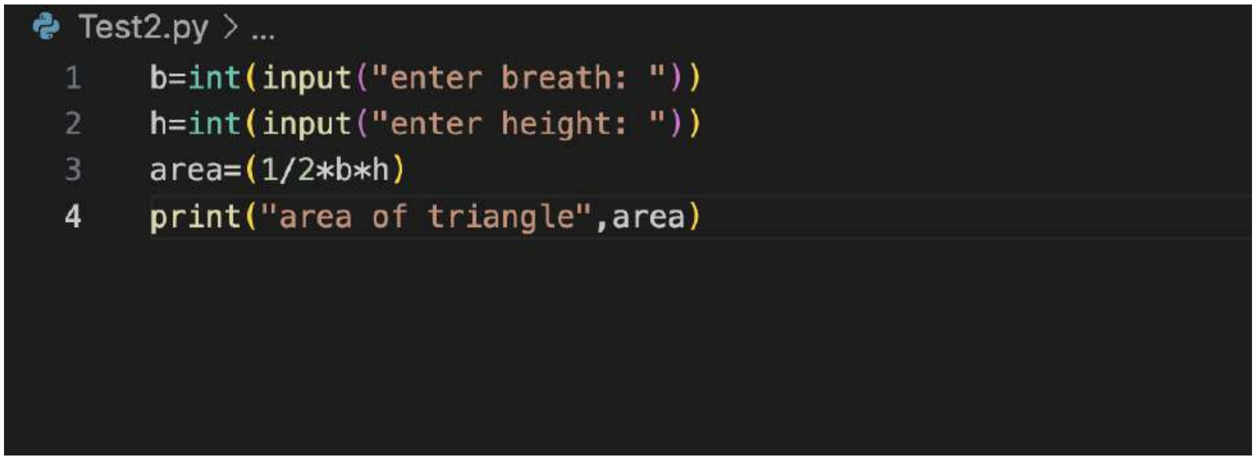
Write program on area of rectangle
Formula for rectangle: (area=length * breath)



```python
length=int(input("enter a area of length: "))
breath=int(input("enter a area of breath: "))
area=(breath*length)
print("area of rectangle",area)
```

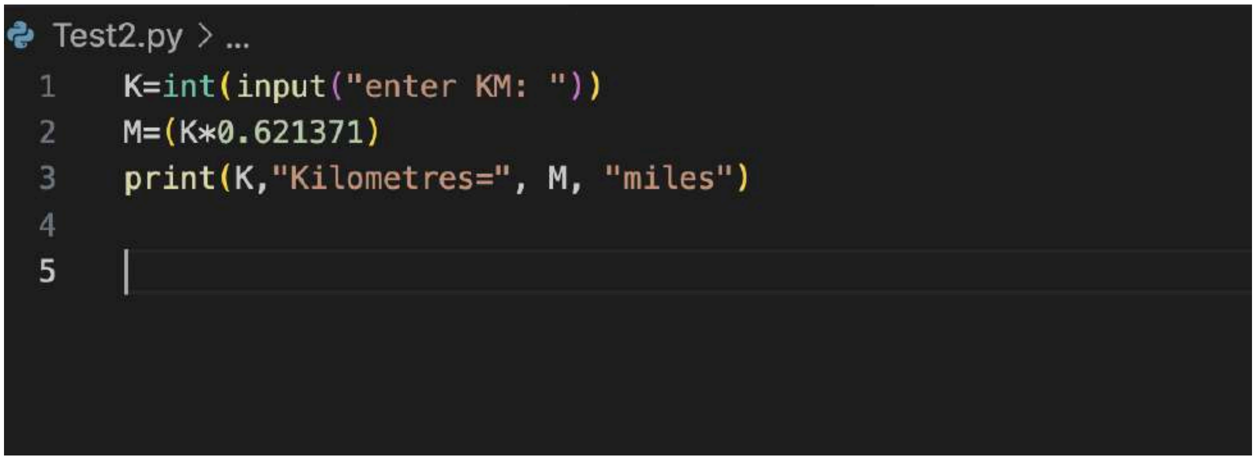
4. Challenges:

a) find Area of triangle



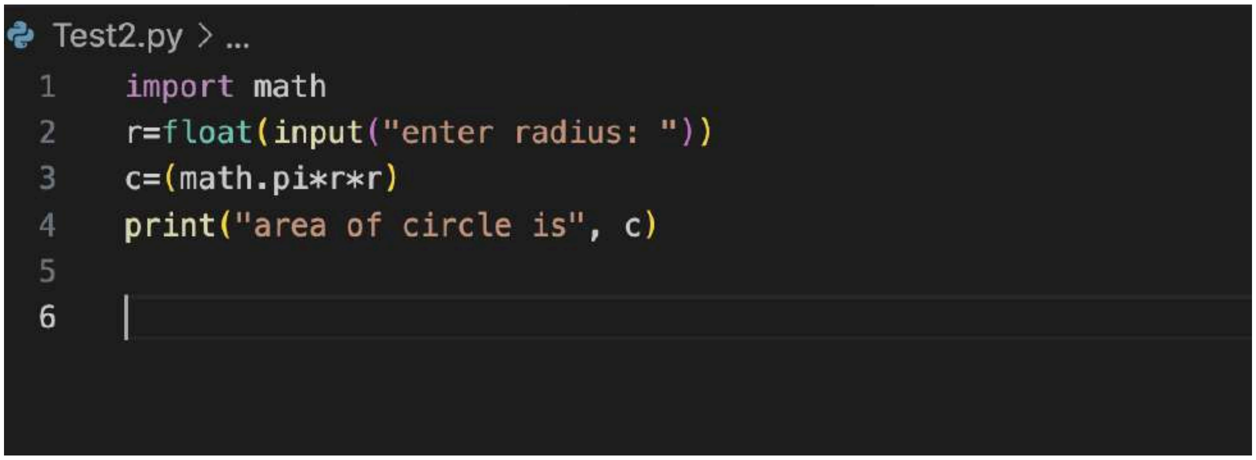
```python
b=int(input("enter breath: "))
h=int(input("enter height: "))
area=(1/2*b*h)
print("area of triangle",area)
```

b) converting KM to Miles



```python
K=int(input("enter KM: "))
M=(K*0.621371)
print(K,"Kilometres=", M, "miles")


```

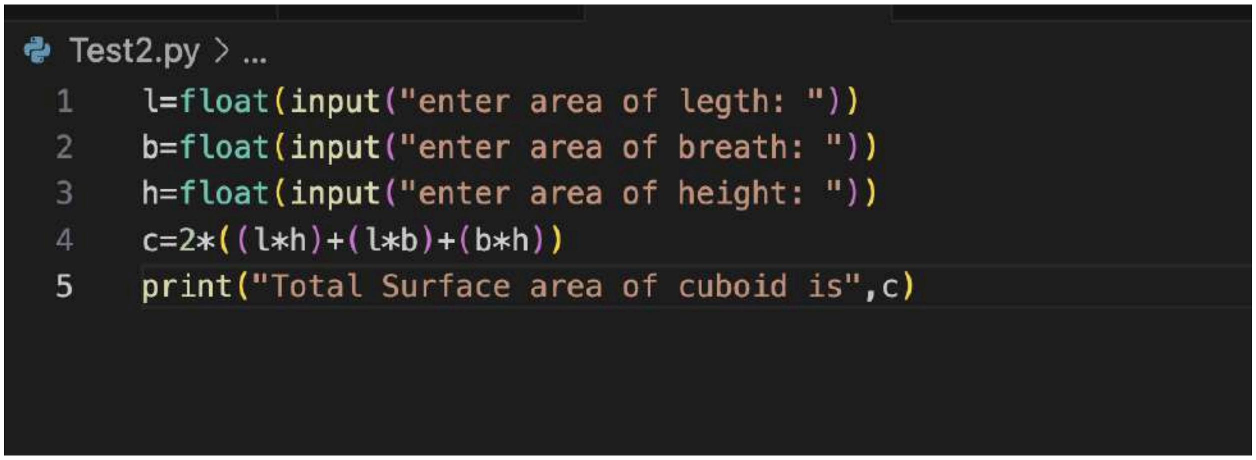
c) write program on find Area of circle



Test2.py > ...

```python
import math
r=float(input("enter radius: "))
c=(math.pi*r*r)
print("area of circle is", c)

```

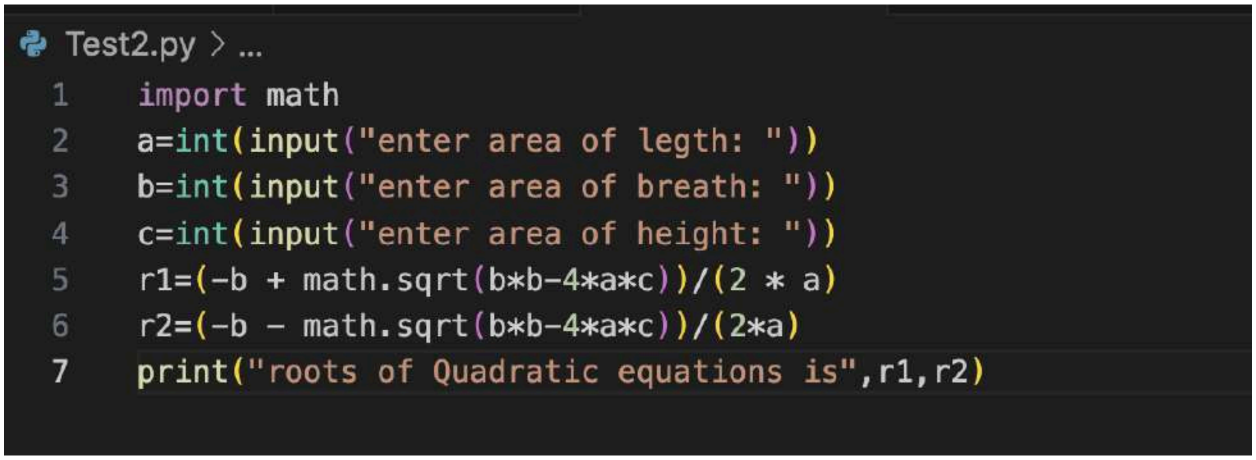
d) Total surface area of cuboid



Test2.py > ...

```python
l=float(input("enter area of legth: "))
b=float(input("enter area of breath: "))
h=float(input("enter area of height: "))
c=2*((l*h)+(l*b)+(b*h))
print("Total Surface area of cuboid is",c)
```

e) finding roots of Quadratic equations

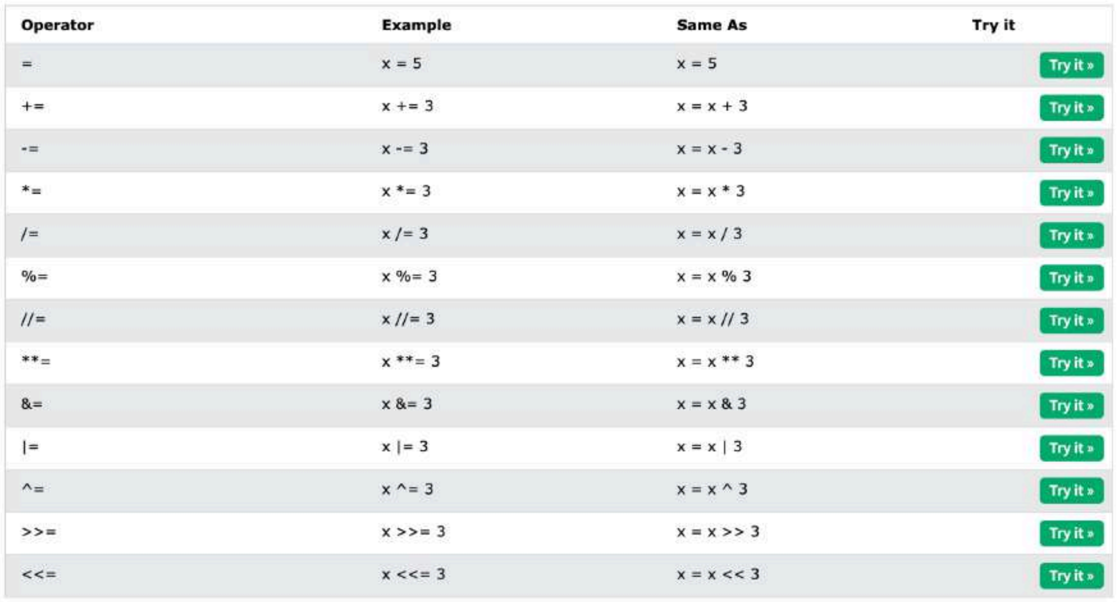


Test2.py > ...

```python
import math
a=int(input("enter area of legth: "))
b=int(input("enter area of breath: "))
c=int(input("enter area of height: "))
r1=(-b + math.sqrt(b*b-4*a*c))/(2 * a)
r2=(-b - math.sqrt(b*b-4*a*c))/(2*a)
print("roots of Quadratic equations is",r1,r2)
```

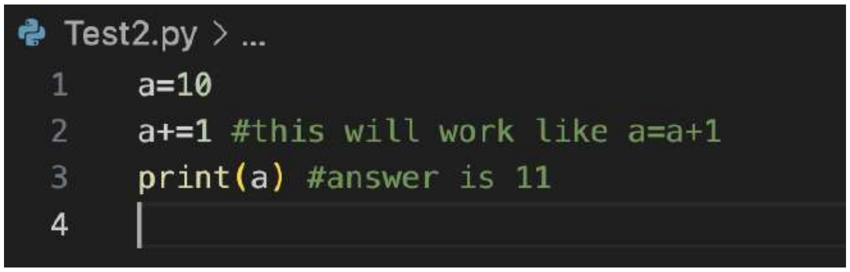
5. Assignment Operators

In Assignment Operators if we use “a+=1” it will first calculate a+1 and then store the result value into the “a”. like this it will work for all the below mentioned operators



Operator	Example	Same As	Try it
=	x = 5	x = 5	Try it »
+=	x += 3	x = x + 3	Try it »
-=	x -= 3	x = x - 3	Try it »
*=	x *= 3	x = x * 3	Try it »
/=	x /= 3	x = x / 3	Try it »
%=	x %= 3	x = x % 3	Try it »
//=	x //= 3	x = x // 3	Try it »
**=	x **= 3	x = x ** 3	Try it »
&=	x &= 3	x = x & 3	Try it »
|=	x |= 3	x = x | 3	Try it »
^=	x ^= 3	x = x ^ 3	Try it »
>>=	x >>= 3	x = x >> 3	Try it »
<<=	x <<= 3	x = x << 3	Try it »

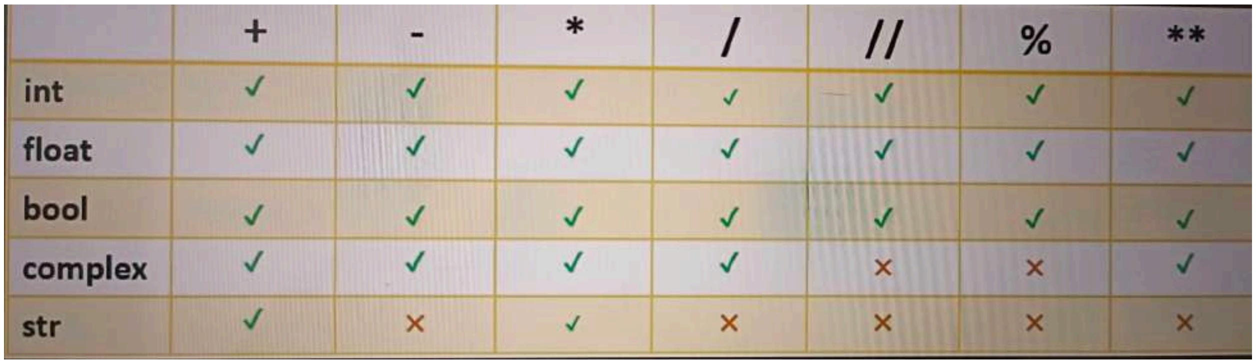
EX:



Test2.py > ...

```python
a=10
a+=1 #this will work like a=a+1
print(a) #answer is 11
```

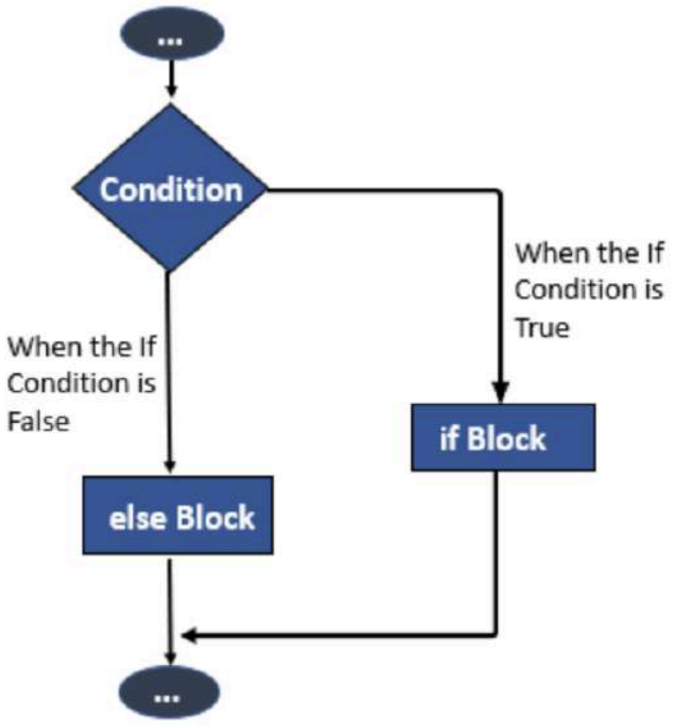
6. Arithmetic with all Data Types



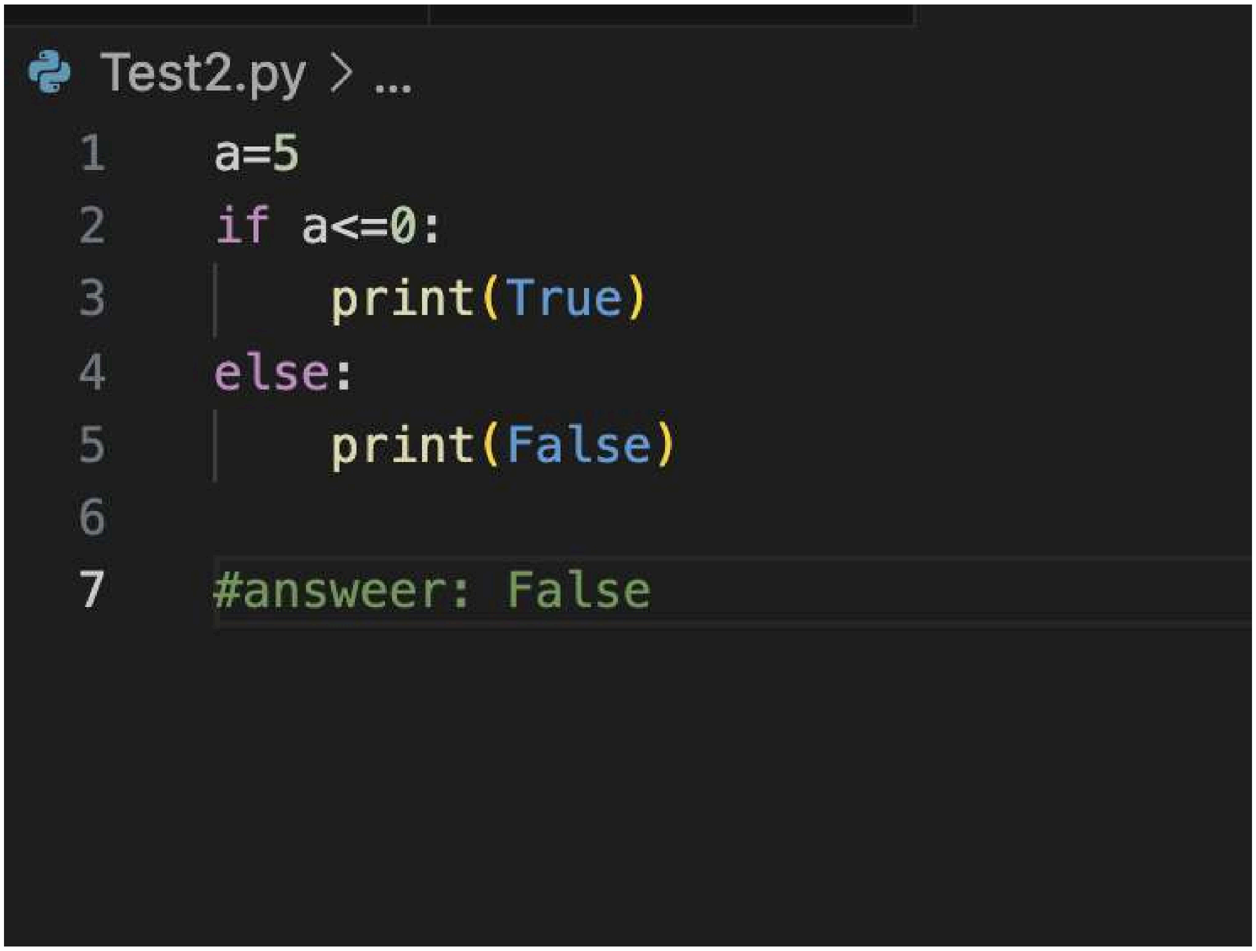
	+	-	*	/	//	%	**
int	✓	✓	✓	✓	✓	✓	✓
float	✓	✓	✓	✓	✓	✓	✓
bool	✓	✓	✓	✓	✓	✓	✓
complex	✓	✓	✓	✓	✗	✗	✓
str	✓	✗	✓	✗	✗	✗	✗

Conditional Statement

1. **If else Conditional**

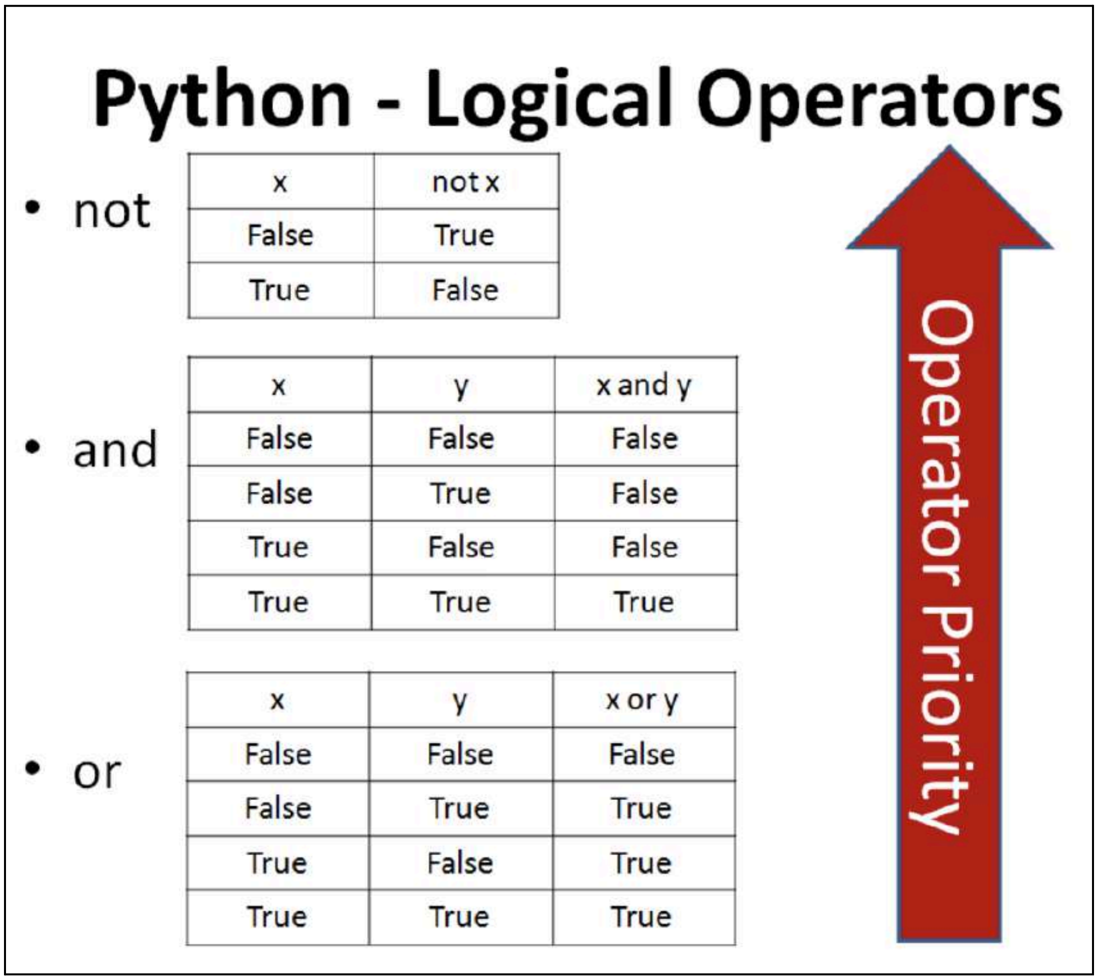


EX: for if else condition



```python
a=5
if a<=0:
    print(True)
else:
    print(False)

#answeer: False
```

2. Logical operator in if else Condition



Python - Logical Operators

- not

x	not x
False	True
True	False

- and

x	y	x and y
False	False	False
False	True	False
True	False	False
True	True	True

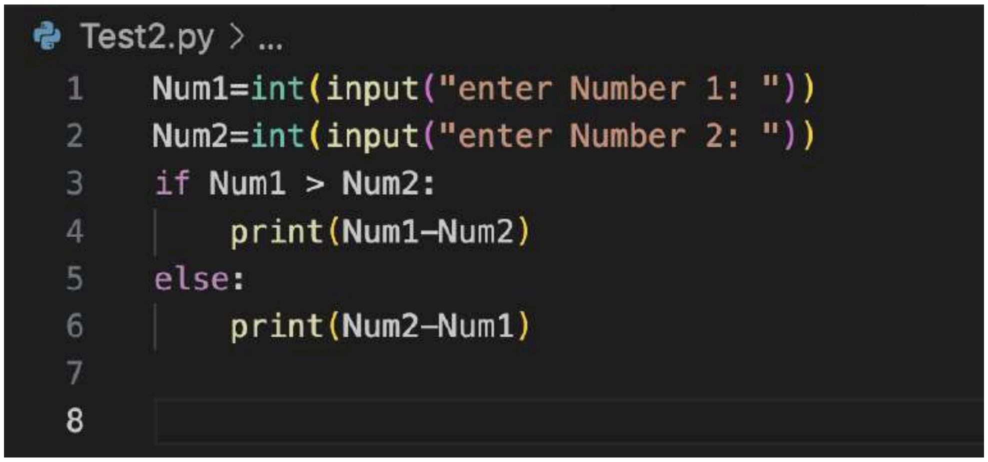
- or

x	y	x or y
False	False	False
False	True	True
True	False	True
True	True	True

Operator Priority

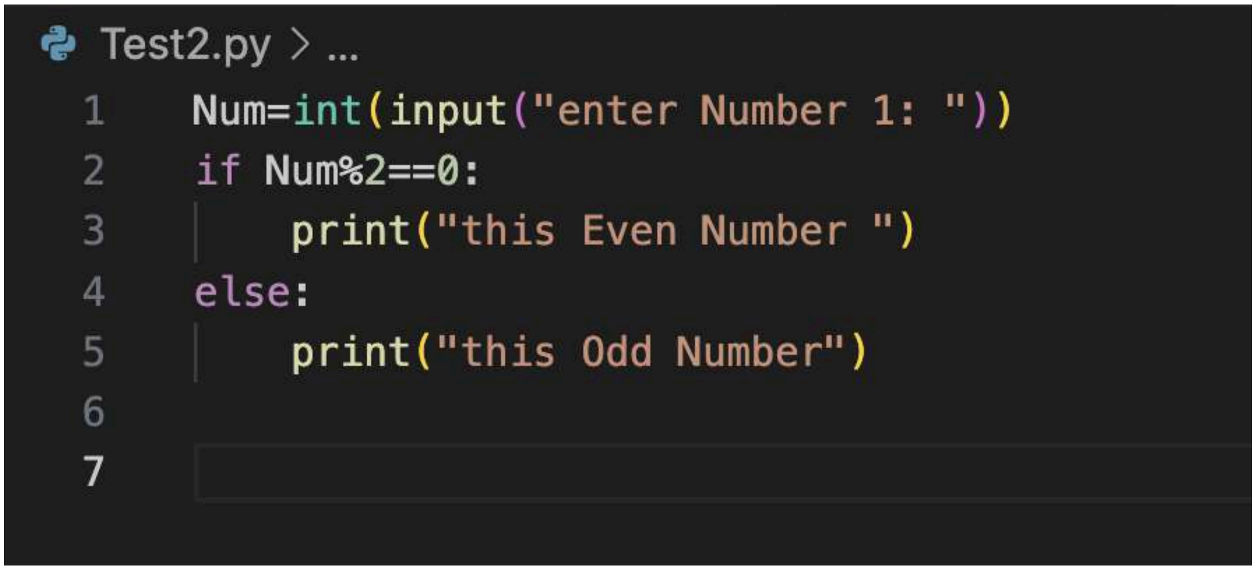
3. Challenges on if else condition

a) Find difference between two number and the result value should be in positive value



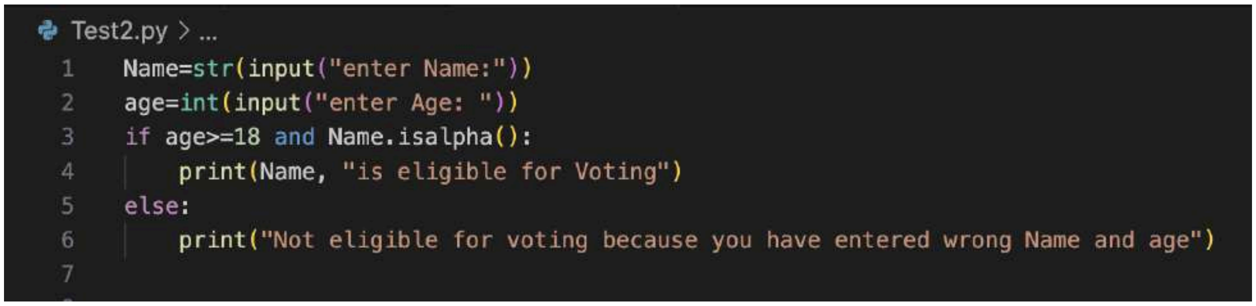
```python
Num1=int(input("enter Number 1: "))
Num2=int(input("enter Number 2: "))
if Num1 > Num2:
    print(Num1-Num2)
else:
    print(Num2-Num1)
```

b) Check if number is odd or Even



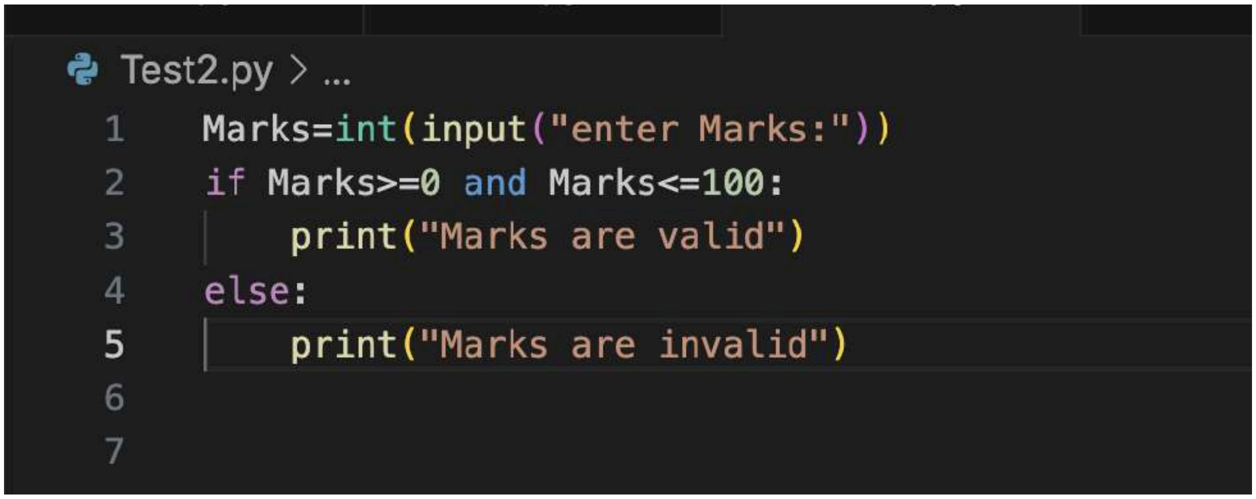
```
Test2.py > ...
Num=int(input("enter Number 1: "))
if Num%2==0:
    print("this Even Number ")
else:
    print("this Odd Number")
```

c) Check for Age Eligibility for casting Vote



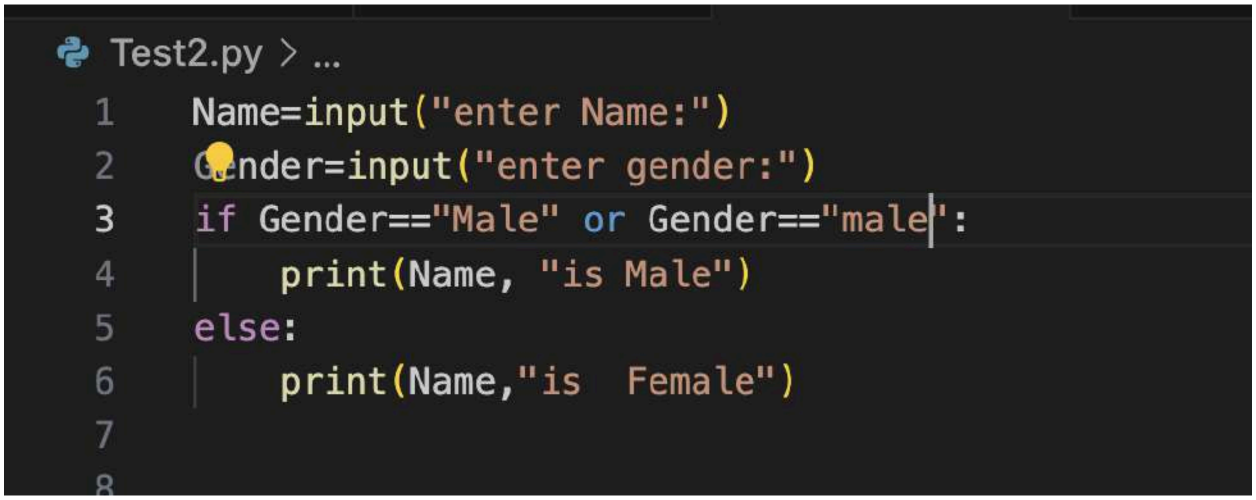
```
Test2.py > ...
Name=str(input("enter Name:"))
age=int(input("enter Age: "))
if age>=18 and Name.isalpha():
    print(Name, "is eligible for Voting")
else:
    print("Not eligible for voting because you have entered wrong Name and age")
```

d) check if marks of subject are within range 0-100

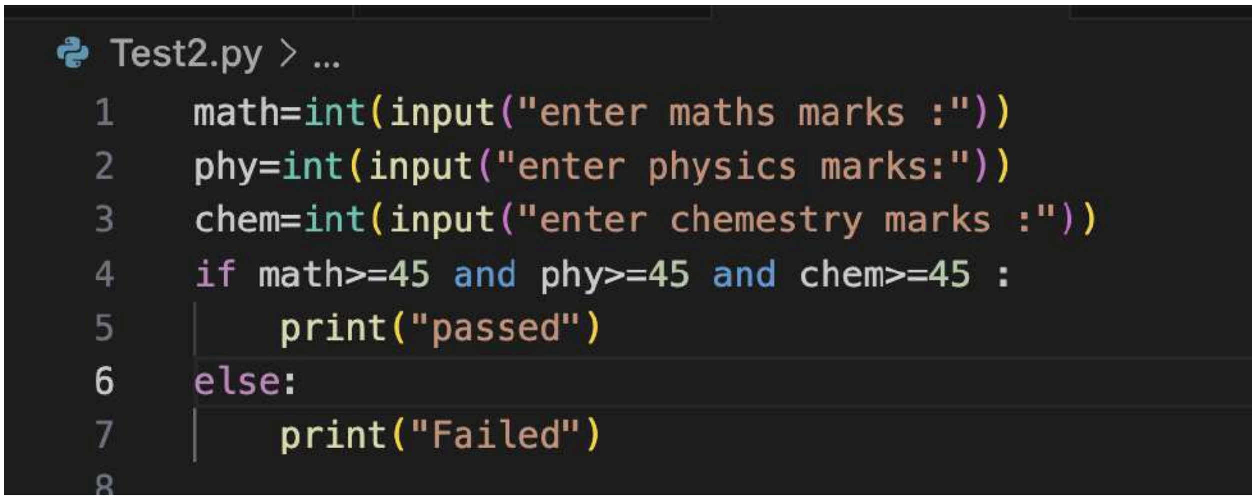


```
Test2.py > ...
Marks=int(input("enter Marks:"))
if Marks>=0 and Marks<=100:
    print("Marks are valid")
else:
    print("Marks are invalid")
```

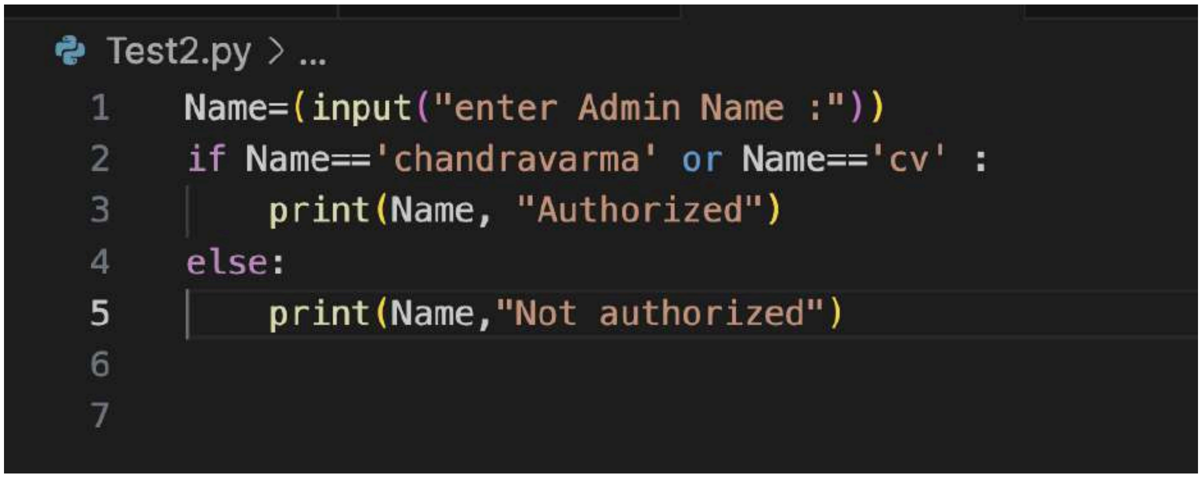
e) check if person is Male or Female


Test2.py > ...
```python
Name=input("enter Name:")
Gender=input("enter gender:")
if Gender=="Male" or Gender=="male":
    print(Name, "is Male")
else:
    print(Name,"is  Female")
```

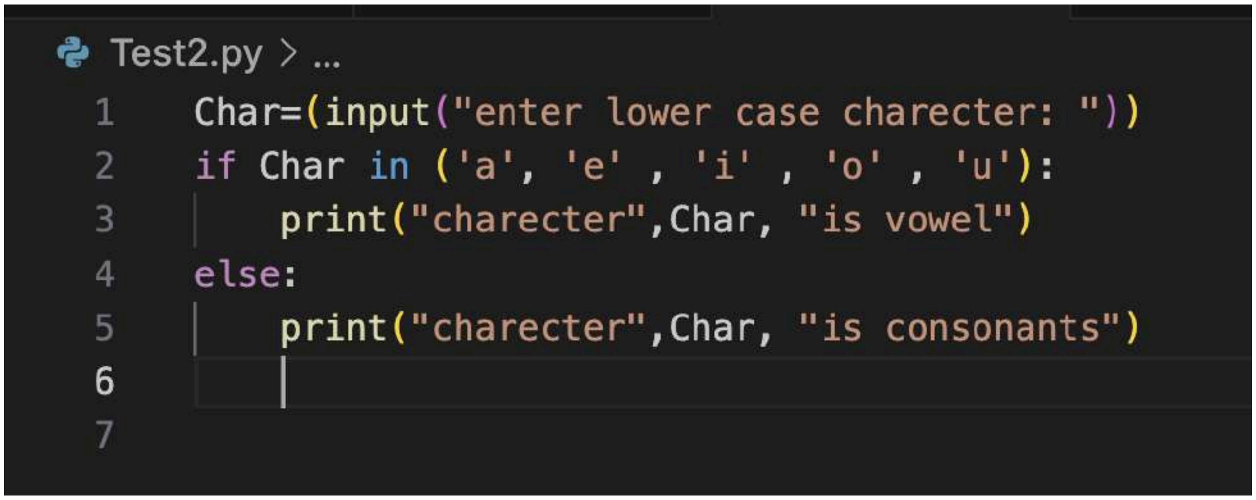
f) check student has passed or failed, by taking marks in 3 subjects


Test2.py > ...
```python
math=int(input("enter maths marks :"))
phy=int(input("enter physics marks:"))
chem=int(input("enter chemestry marks :"))
if math>=45 and phy>=45 and chem>=45 :
    print("passed")
else:
    print("Failed")
```

g)check if person is authorized for admin access


Test2.py > ...
```python
Name=(input("enter Admin Name :"))
if Name=='chandravarma' or Name=='cv' :
    print(Name, "Authorized")
else:
    print(Name,"Not authorized")
```

h)check if given lower case character is vowel or consonants

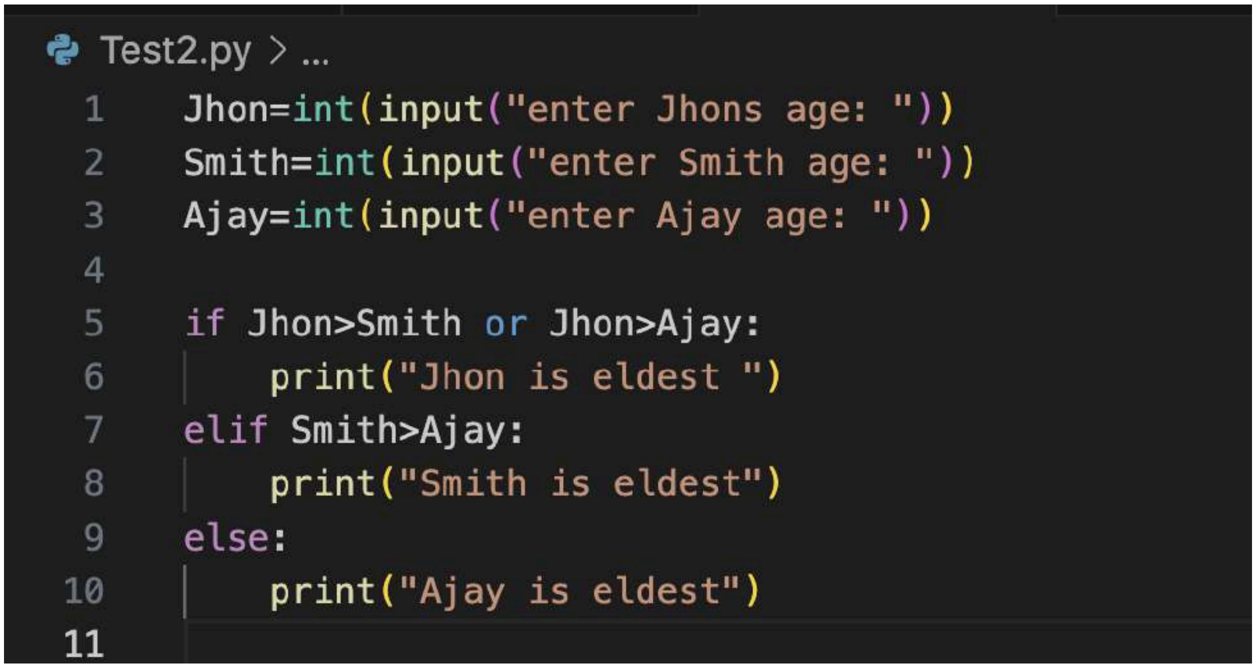


Test2.py > ...

```python
Char=(input("enter lower case charecter: "))
if Char in ('a', 'e' , 'i' , 'o' , 'u'):
    print("charecter",Char, "is vowel")
else:
    print("charecter",Char, "is consonants")

```

4. Nested if and elif Statement

a) check who is eldest in Jhon, Smith, Ajay using Nested if and elif

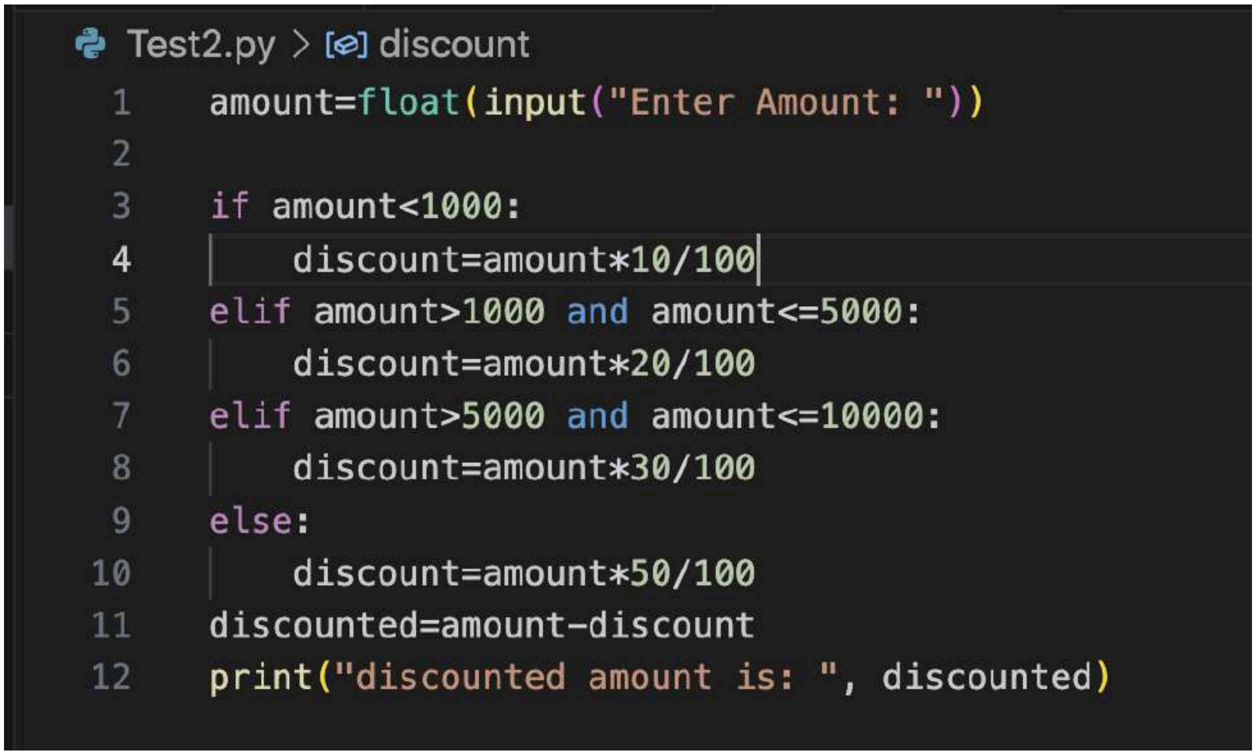


Test2.py > ...

```python
Jhon=int(input("enter Jhons age: "))
Smith=int(input("enter Smith age: "))
Ajay=int(input("enter Ajay age: "))

if Jhon>Smith or Jhon>Ajay:
    print("Jhon is eldest ")
elif Smith>Ajay:
    print("Smith is eldest")
else:
    print("Ajay is eldest")

```

b)Calculate Discounted amount for given Discount Details using elif condition

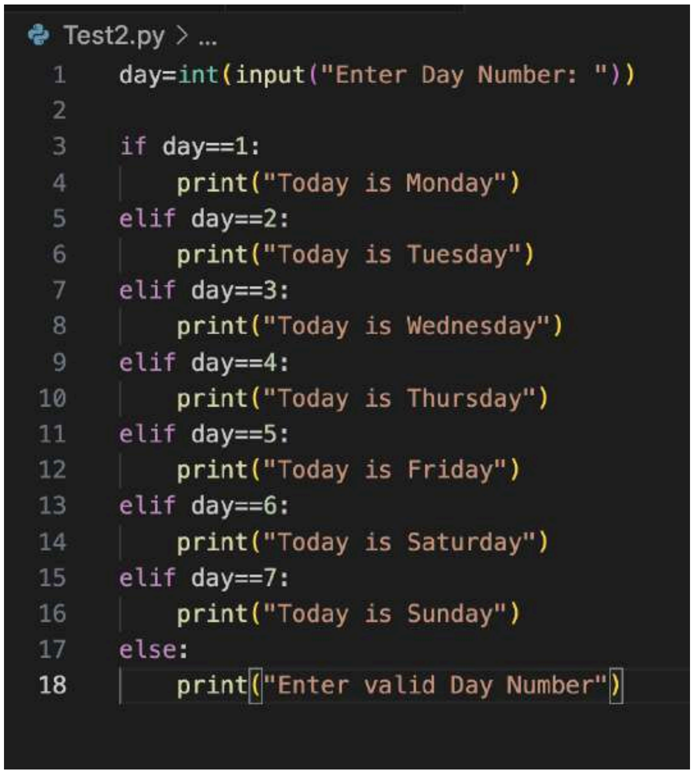
amount <1000 discount is 10%
amount >1000 and <=5000 discount is 20%
amount >5000 and <=10000 Discount is 30%
amount >10000 Discount is 50%



Test2.py > discount

```python
amount=float(input("Enter Amount: "))

if amount<1000:
    discount=amount*10/100
elif amount>1000 and amount<=5000:
    discount=amount*20/100
elif amount>5000 and amount<=10000:
    discount=amount*30/100
else:
    discount=amount*50/100
discounted=amount-discount
print("discounted amount is: ", discounted)
```

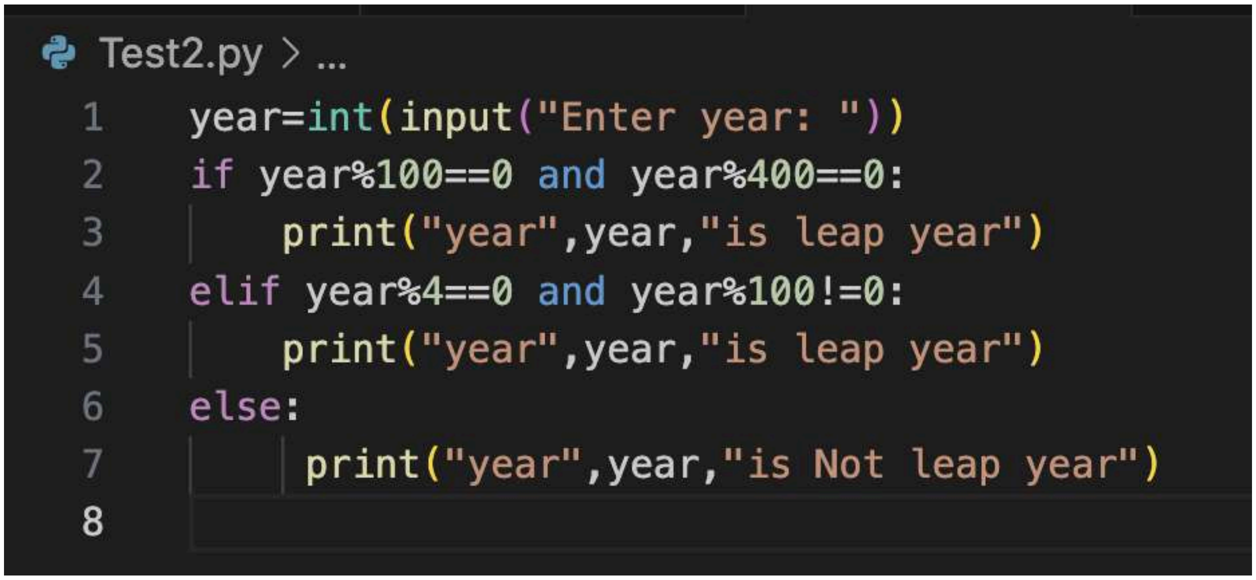
c)Take a day number 1 to 7 and display day name using else if



Test2.py > ...

```python
day=int(input("Enter Day Number: "))

if day==1:
    print("Today is Monday")
elif day==2:
    print("Today is Tuesday")
elif day==3:
    print("Today is Wednesday")
elif day==4:
    print("Today is Thursday")
elif day==5:
    print("Today is Friday")
elif day==6:
    print("Today is Saturday")
elif day==7:
    print("Today is Sunday")
else:
    print("Enter valid Day Number")
```

d)check whether the year is leap year or not

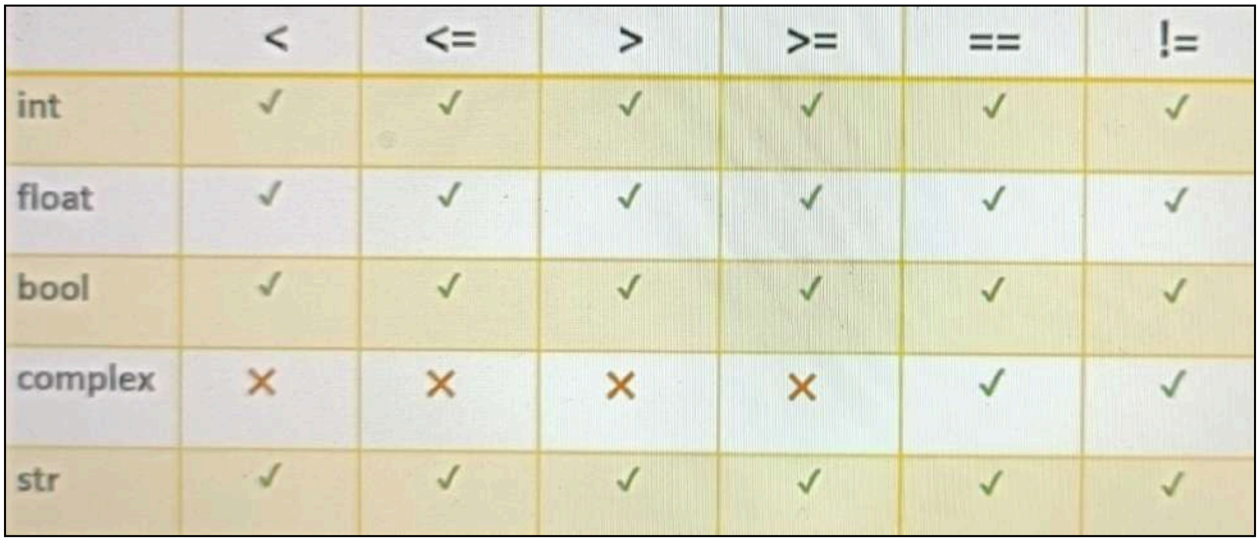


Test2.py > ...

```python
year=int(input("Enter year: "))
if year%100==0 and year%400==0:
    print("year",year,"is leap year")
elif year%4==0 and year%100!=0:
    print("year",year,"is leap year")
else:
      print("year",year,"is Not leap year")
```

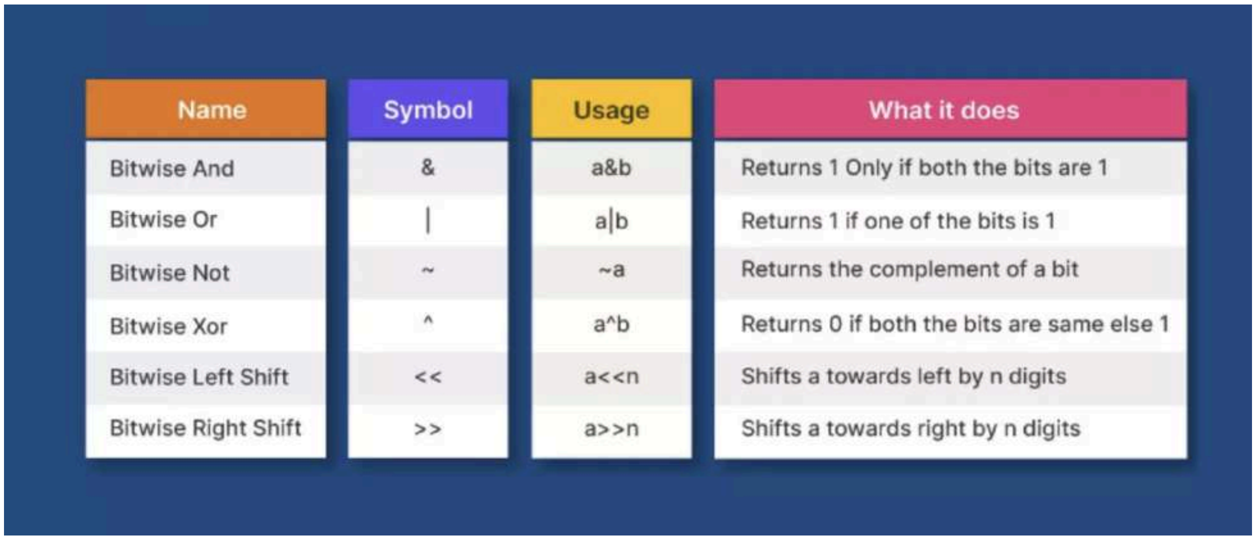
5. Relational operators

Relational operator with Data Types



	<	<=	>	>=	==	!=
int	✓	✓	✓	✓	✓	✓
float	✓	✓	✓	✓	✓	✓
bool	✓	✓	✓	✓	✓	✓
complex	✗	✗	✗	✗	✓	✓
str	✓	✓	✓	✓	✓	✓

6. Bitwise Operator



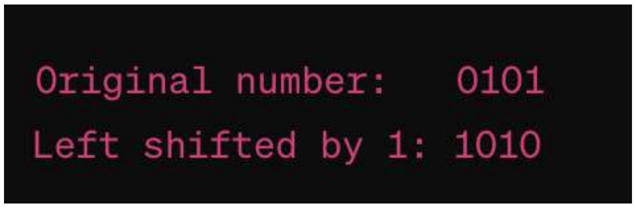
Name	Symbol	Usage	What it does
Bitwise And	&	a&b	Returns 1 Only if both the bits are 1
Bitwise Or	|	a|b	Returns 1 if one of the bits is 1
Bitwise Not	~	~a	Returns the complement of a bit
Bitwise Xor	^	a^b	Returns 0 if both the bits are same else 1
Bitwise Left Shift	<<	a<<n	Shifts a towards left by n digits
Bitwise Right Shift	>>	a>>n	Shifts a towards right by n digits

a)Bitwise And operator in binary (&)
1 and 1 =1
1 and 0 =0
0 and 1 =0
0 and 0 =0

b)Bitwise OR operator in binary (|)
1 OR 1 =1
1 OR 0 =1
0 OR 1 =1
0 OR 0 =0

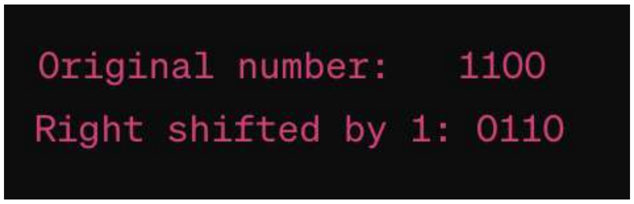
c)Bitwise XOR operator in binary (^)
1 XOR 1 =0
1 XOR 0 =1
0 XOR 1 =1
0 XOR 0 =0

d)Bitwise Left Shift operator in binary (<<)
Consider the binary number 0101 (which represents the decimal number 5 in binary). If we perform a left shift by 1 bit (0101 << 1), it would look like this:



```
Original number:    0101
Left shifted by 1: 1010
```

e)Bitwise Right Shift operator in binary (>>)
Consider the binary number 1100 (which represents the decimal number 12). If we perform a right shift by 1 bit (1100 >> 1), it would result in:

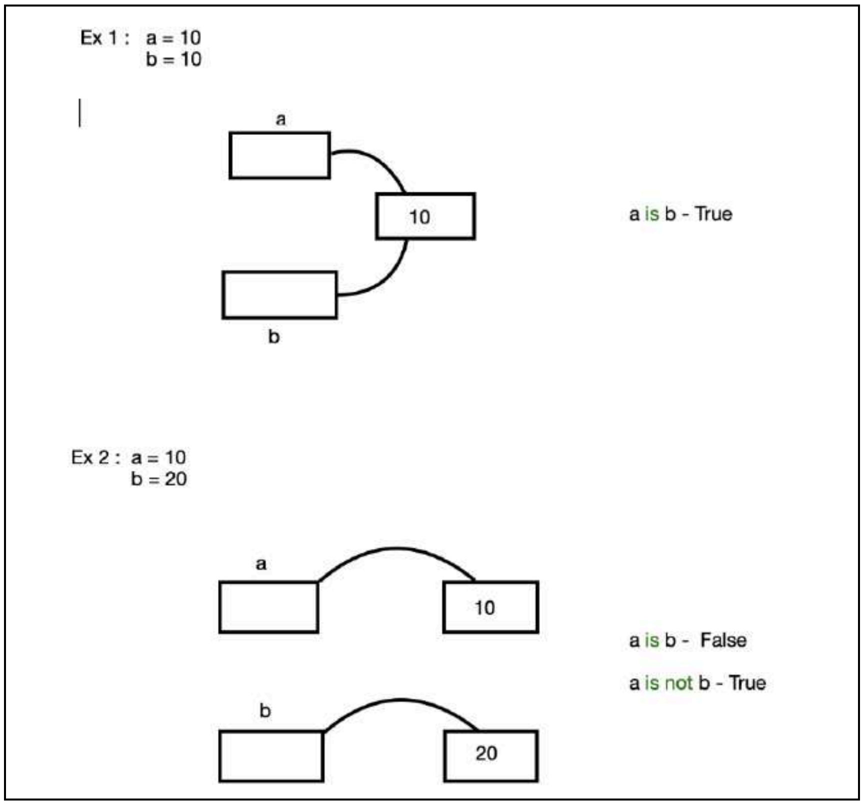


```
Original number:   1100
Right shifted by 1: 0110
```

7. Identity Operator

Operator	Description
is	It returns true if two variables point the same object and false otherwise
is not	It returns false if two variables point the same object and true otherwise

Note: it will work when you directly assign value to a variable or using "Int" data type to accept input but when you accept input values from the user without data type declaration python creates new memory even when the given values are the same.
EX:



Loops / Control flow

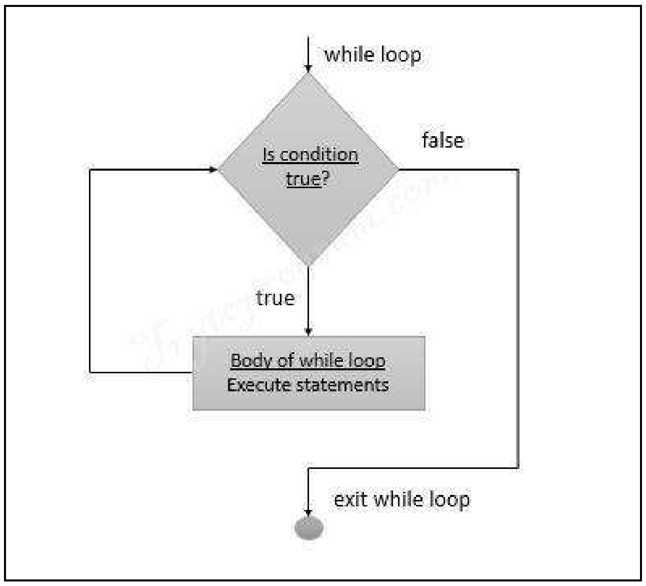
1) Control Flow

a) While loop
b) For loop
c) Continue, break, pass
d) Infinite loop
e) Match case

2) Introduction to Loop's

a) Loops are also called repeating statements.
b) If you want a set of statements to repeat again and again in the program then we use loops The statements can repeat either 'for number of times' or 'As long as the condition is true'

While Loop

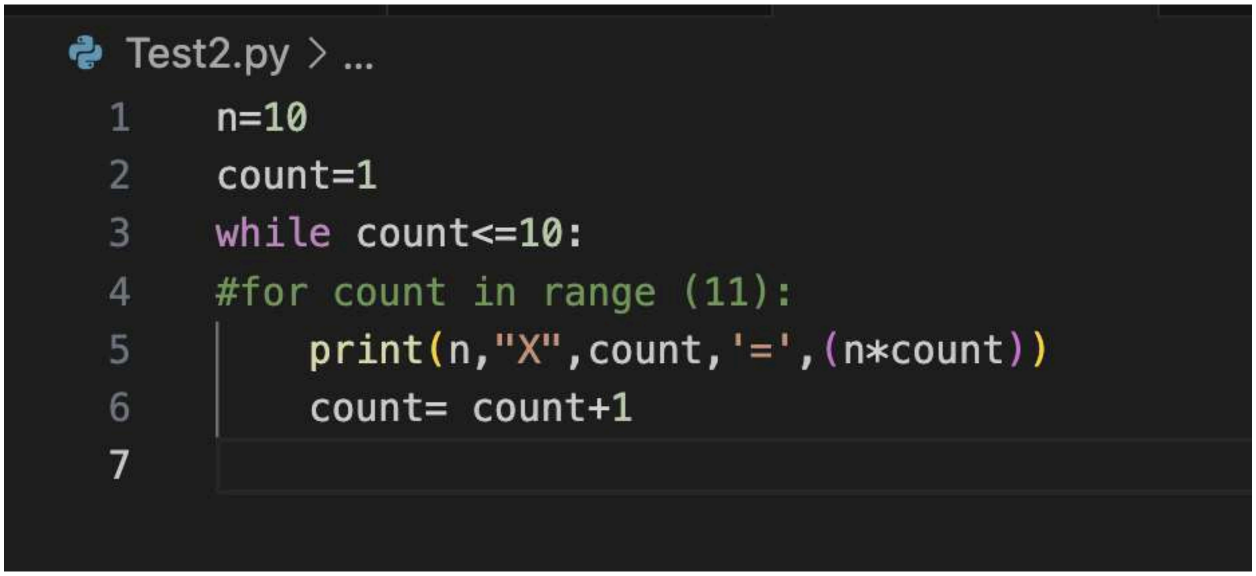


EX: print hello 10 time

```
count=0
while count<10:
        print("Hello")
        count= count+1
```

3) While loop Challenges

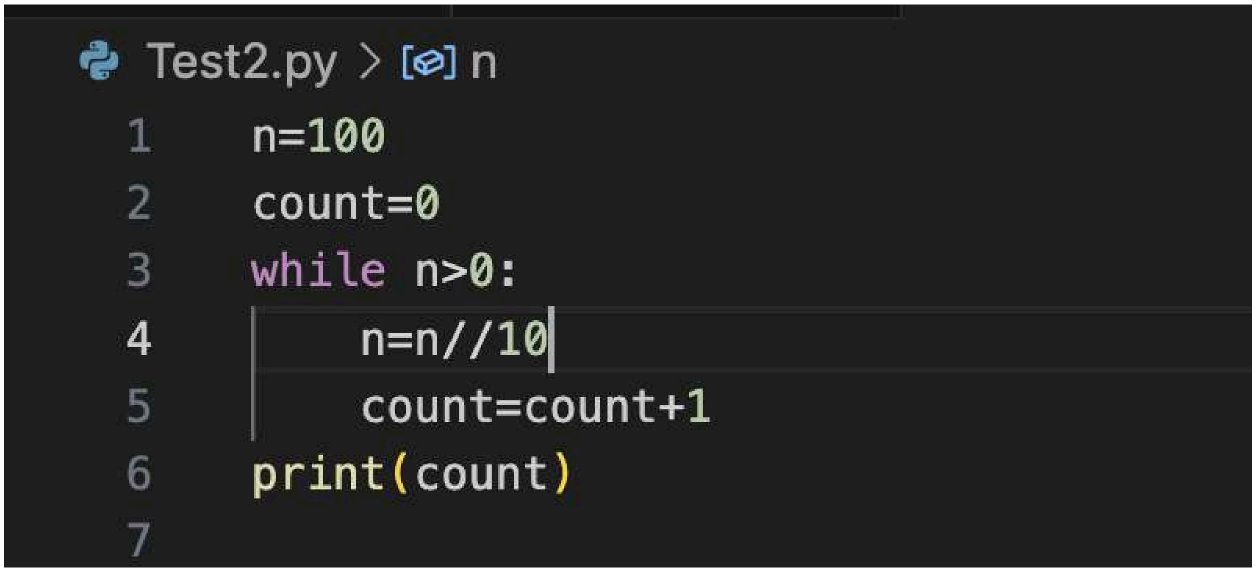
3.1) Display Multiplication table for given number



Test2.py > ...

```python
n=10
count=1
while count<=10:
#for count in range (11):
    print(n,"X",count,'=',(n*count))
    count= count+1

```

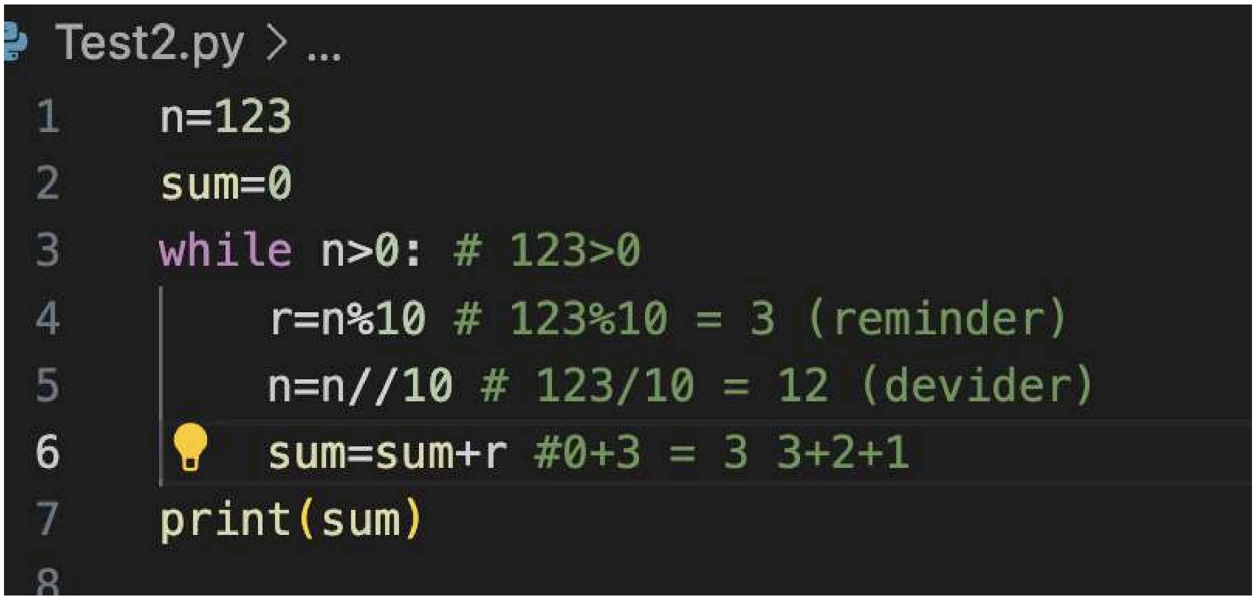
3.2) counting number of digits in number



Test2.py > [⊘] n

```python
n=100
count=0
while n>0:
    n=n//10
    count=count+1
print(count)

```

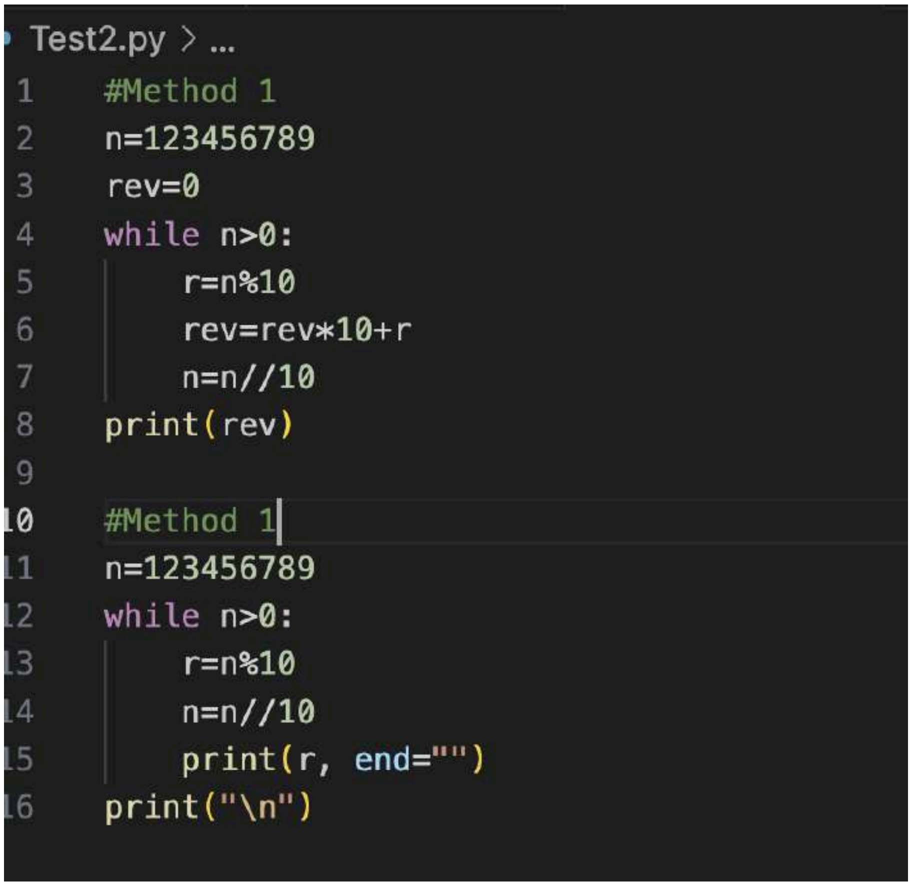
3.3) find sum of digit in number



Test2.py > ...

```python
n=123
sum=0
while n>0: # 123>0
    r=n%10 # 123%10 = 3 (reminder)
    n=n//10 # 123/10 = 12 (devider)
    sum=sum+r #0+3 = 3 3+2+1
print(sum)
```

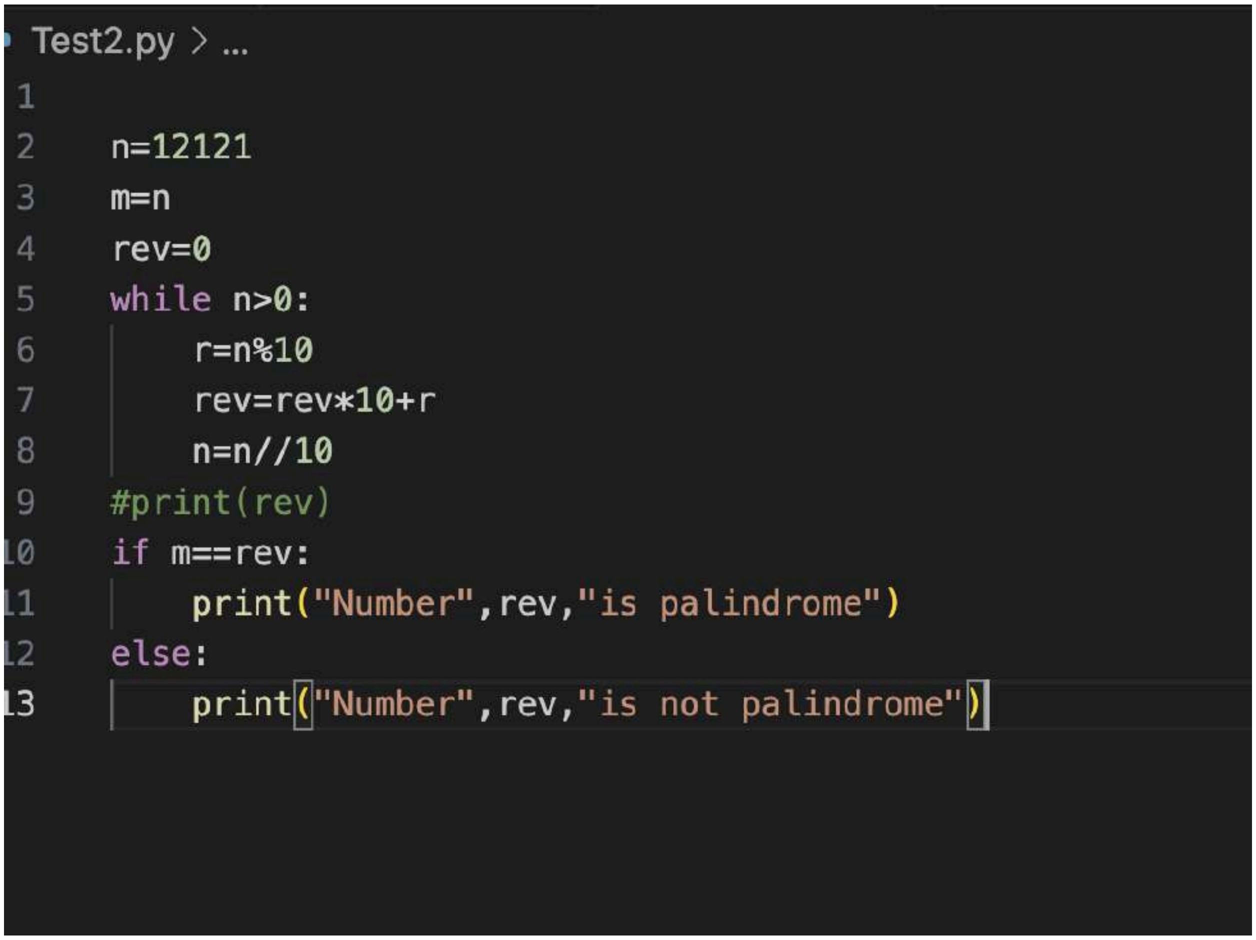
3.4) Reversing number



Test2.py > ...

```python
#Method 1
n=123456789
rev=0
while n>0:
    r=n%10
    rev=rev*10+r
    n=n//10
print(rev)

#Method 1
n=123456789
while n>0:
    r=n%10
    n=n//10
    print(r, end="")
print("\n")
```

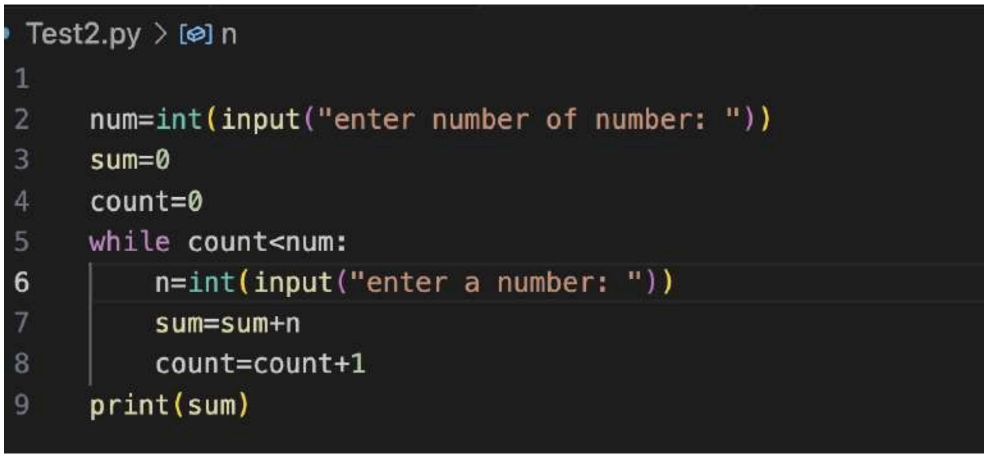
3.5) check given number is palindrome or not



Test2.py > ...

```python

n=12121
m=n
rev=0
while n>0:
    r=n%10
    rev=rev*10+r
    n=n//10
#print(rev)
if m==rev:
    print("Number",rev,"is palindrome")
else:
    print("Number",rev,"is not palindrome")
```

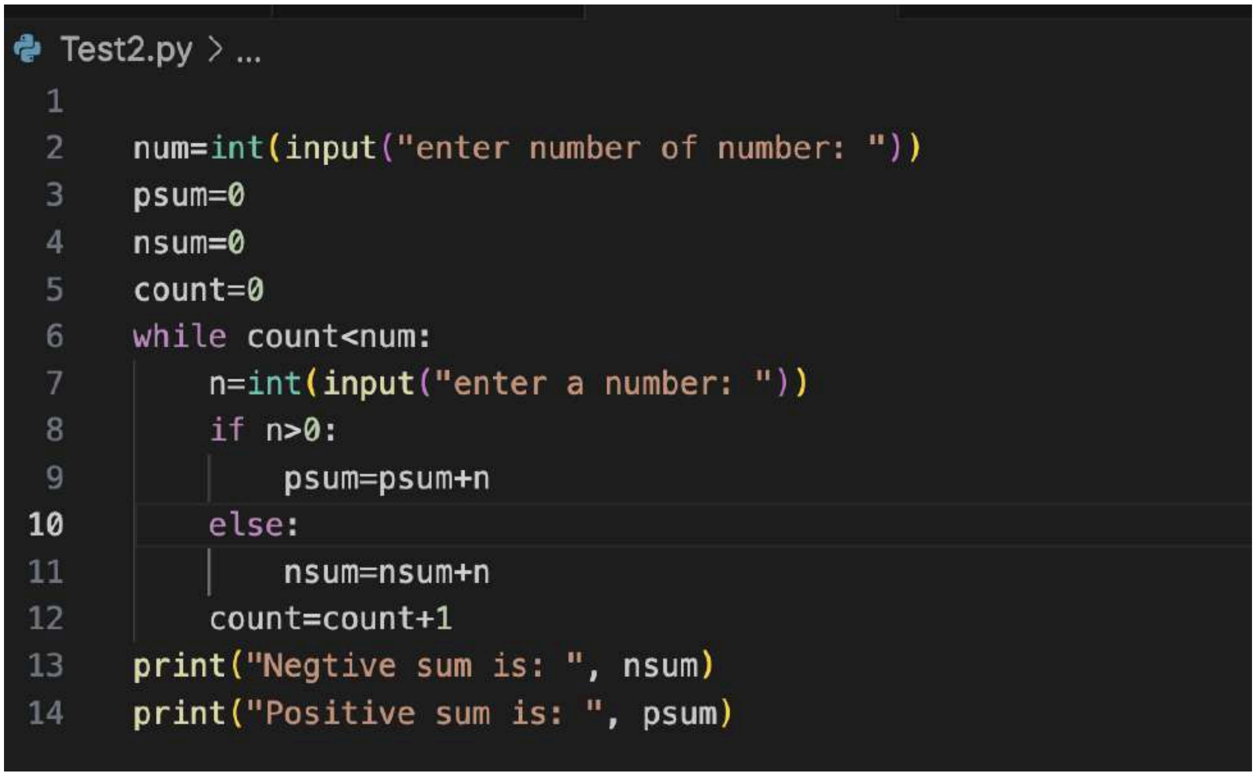
3.6) find sum of given number as input
Note:
First take number from user how many time he want to enter number
Then make sum of entered number



Test2.py > n

```python

num=int(input("enter number of number: "))
sum=0
count=0
while count<num:
    n=int(input("enter a number: "))
    sum=sum+n
    count=count+1
print(sum)
```

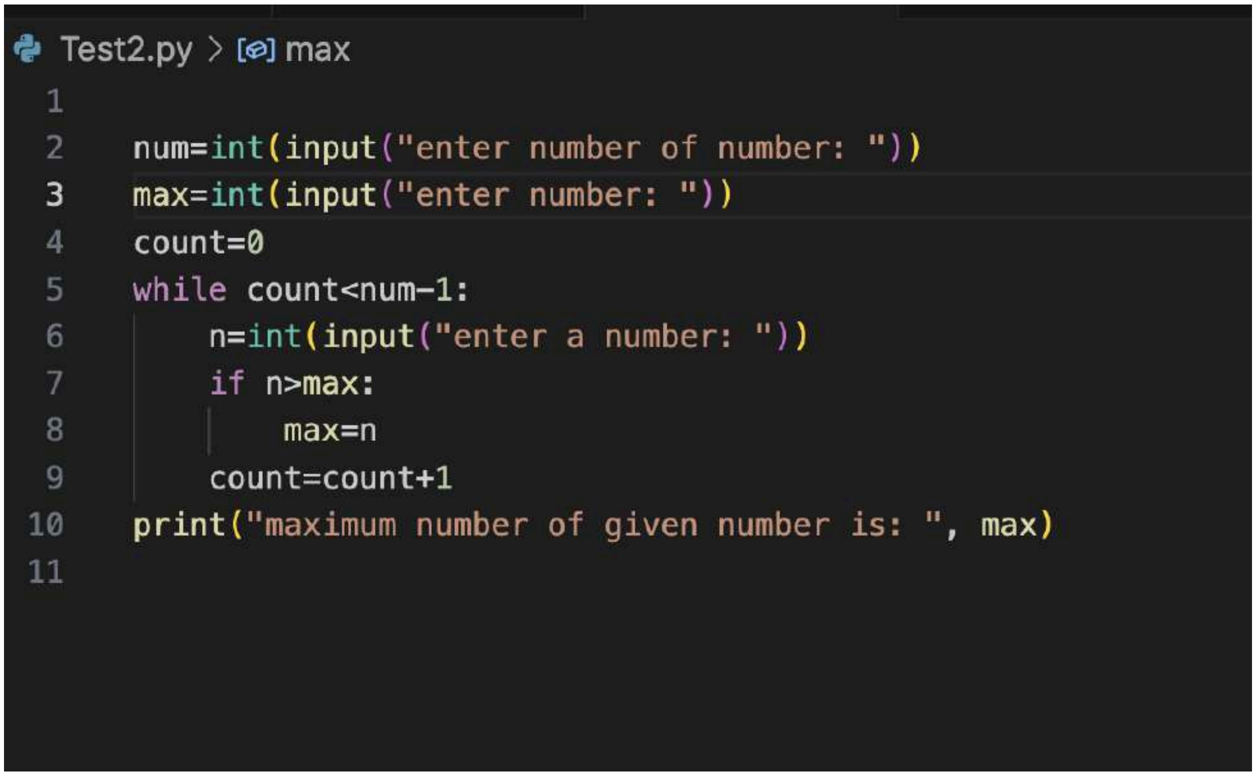
3.7)print sum of -ve value and +ve value



Test2.py > ...

```python

num=int(input("enter number of number: "))
psum=0
nsum=0
count=0
while count<num:
    n=int(input("enter a number: "))
    if n>0:
        psum=psum+n
    else:
        nsum=nsum+n
    count=count+1
print("Negtive sum is: ", nsum)
print("Positive sum is: ", psum)
```

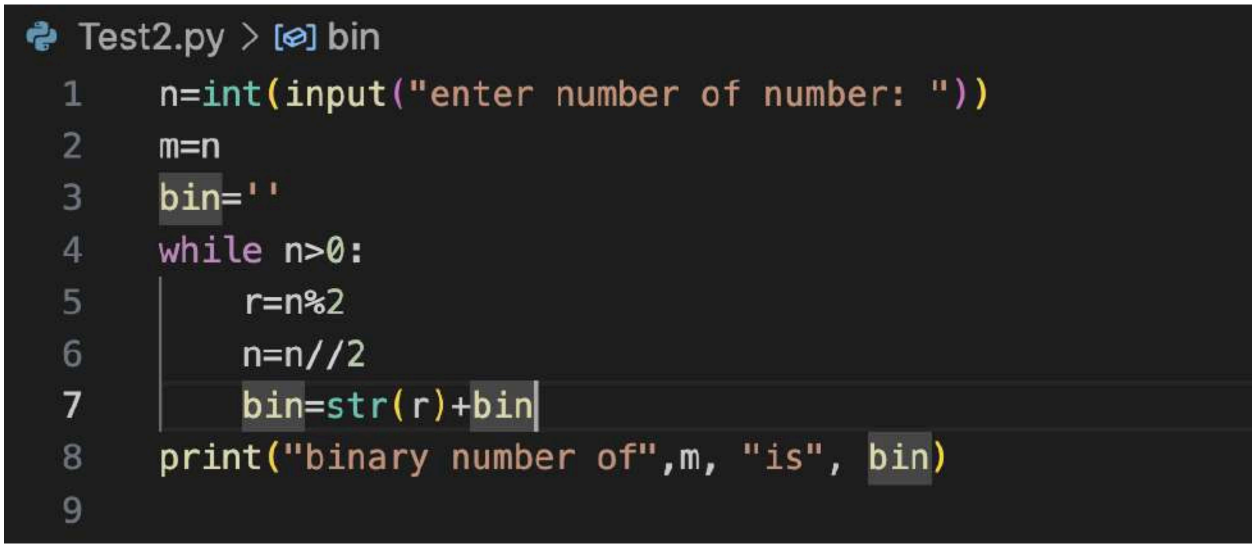
3.8) find Maximum number of given numbers



Test2.py > max

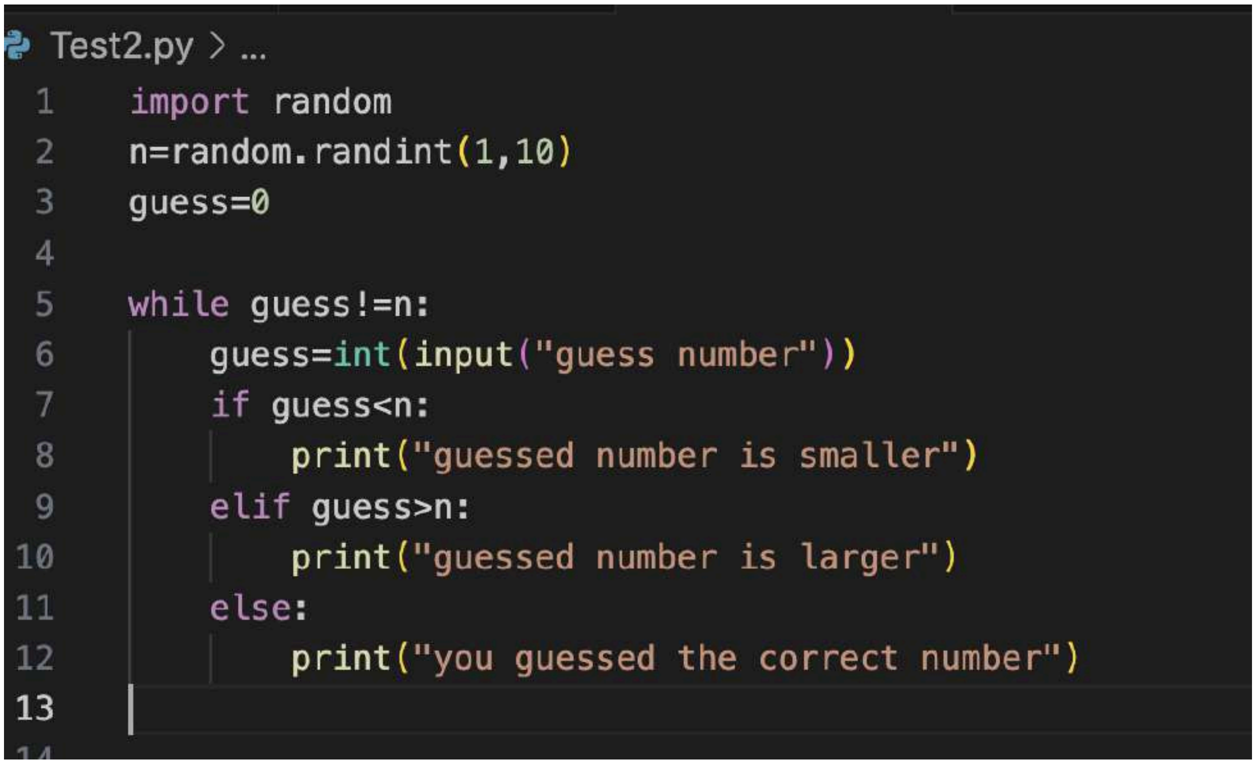
```python

num=int(input("enter number of number: "))
max=int(input("enter number: "))
count=0
while count<num-1:
    n=int(input("enter a number: "))
    if n>max:
        max=n
    count=count+1
print("maximum number of given number is: ", max)

```

3.9) convert decimal number to binary



```python
n=int(input("enter number of number: "))
m=n
bin=''
while n>0:
    r=n%2
    n=n//2
    bin=str(r)+bin
print("binary number of",m, "is", bin)
```

3.10) guess number between 1 to 10

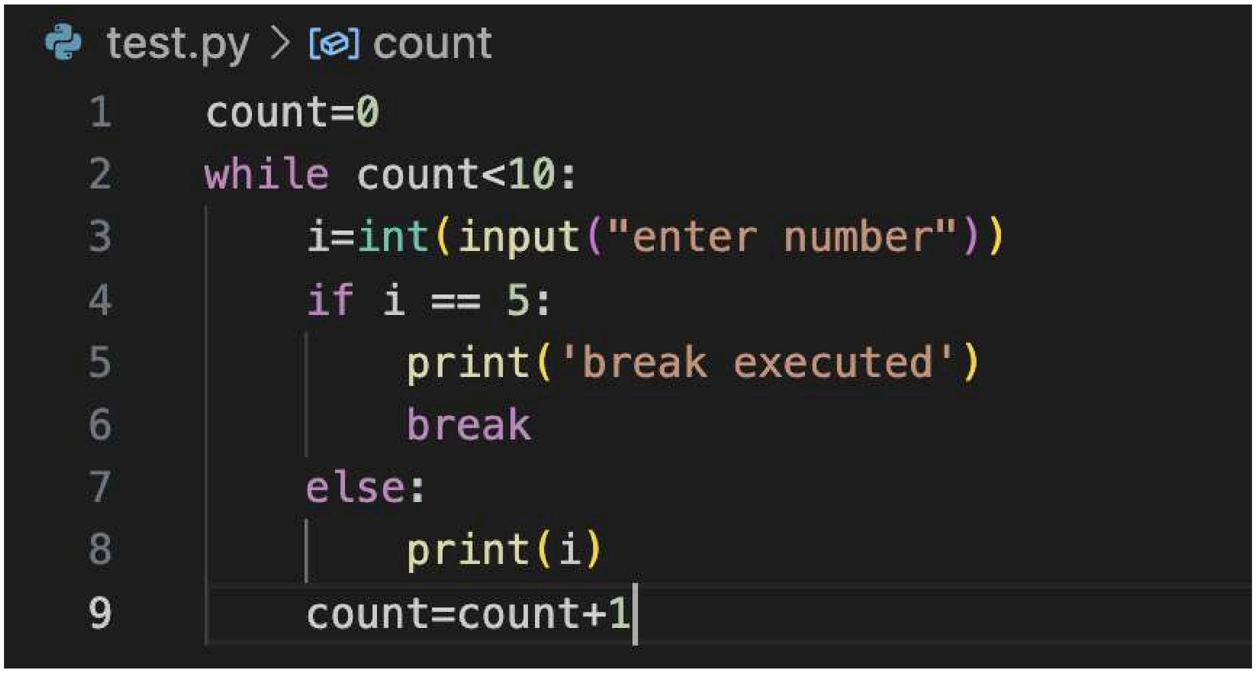


```python
import random
n=random.randint(1,10)
guess=0

while guess!=n:
    guess=int(input("guess number"))
    if guess<n:
        print("guessed number is smaller")
    elif guess>n:
        print("guessed number is larger")
    else:
        print("you guessed the correct number")
```

4) infinite loop - Break - Continue - Pass with while loop

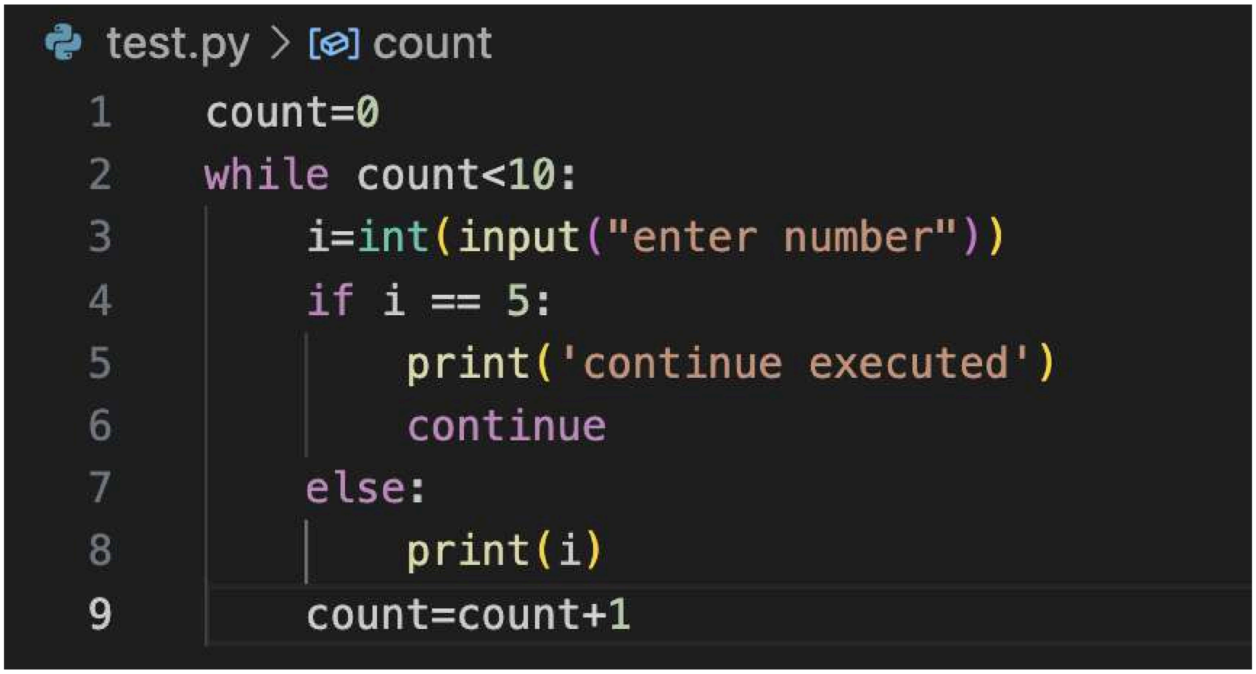
4.1) Break:

In the Break if condition is true then it will print “break executed” and then it will break statement and come out of loop, if its fail it will print(i) value until while loop will get fails



```python
count=0
while count<10:
    i=int(input("enter number"))
    if i == 5:
        print('break executed')
        break
    else:
        print(i)
    count=count+1
```

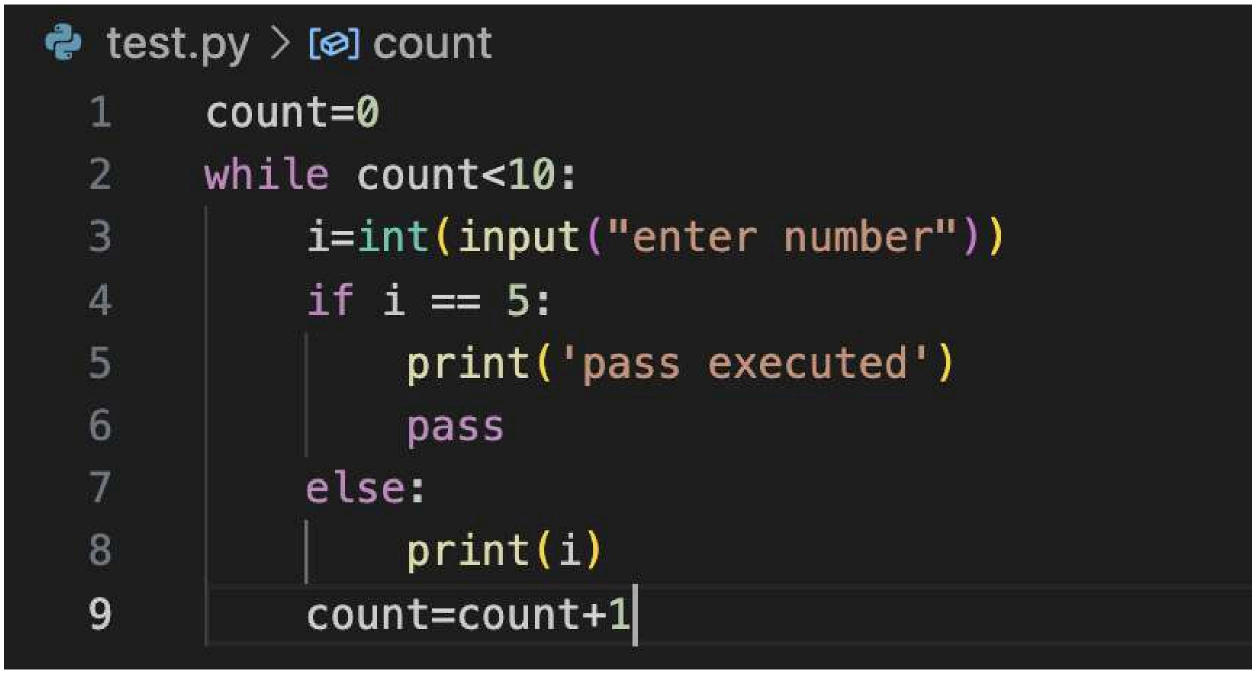
4.2) Continue:
In continue, based on bellow example it will not increment count value bcz it will continue back to loop if condition is true, when if condition will fail then it will print else part and increment count value



```python
count=0
while count<10:
    i=int(input("enter number"))
    if i == 5:
        print('continue executed')
        continue
    else:
        print(i)
    count=count+1
```

4.3) Pass
In pass, based on bellow example it will check if condition if it is true and then it will pass

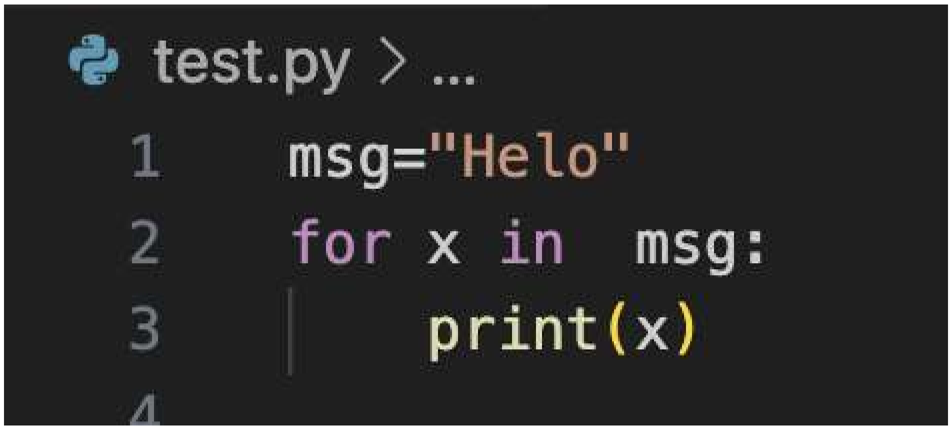
cursor to the count increment (but in continue cursor will not move to count increment statement if condition is true)


test.py > count

```python
count=0
while count<10:
    i=int(input("enter number"))
    if i == 5:
        print('pass executed')
        pass
    else:
        print(i)
    count=count+1
```

5) For Loop:

A Python for loop is a type of loop which is used to iterate a set of sequential statements multiple times (like a list, tuple, or string) Iterating over a sequence means going through each element one by one

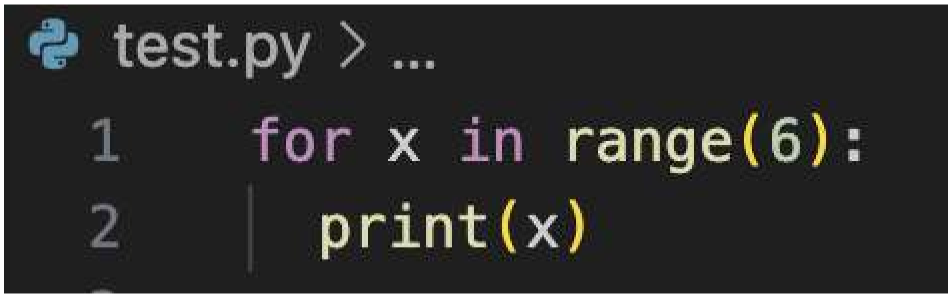

test.py > ...

```python
msg="Helo"
for x in  msg:
    print(x)
```

5.1) For loop range() Function

5.1.1) The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

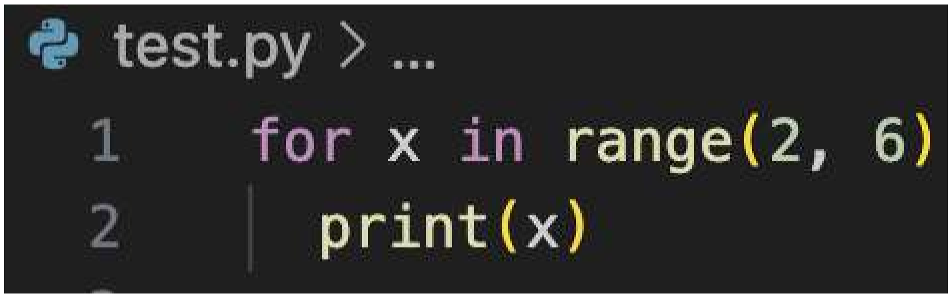
Ex:



```python
for x in range(6):
    print(x)
```

5.1.2) The range() function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: range(2, 6), which means values from 2 to 6 (but not including 6)

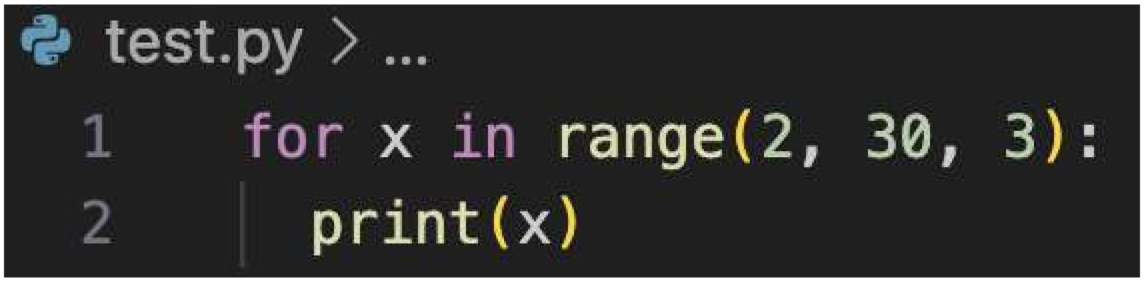
Ex:



```python
for x in range(2, 6)
    print(x)
```

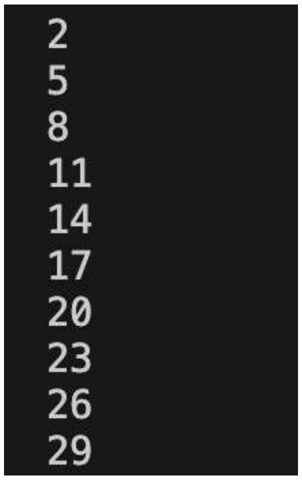
5.1.3)The range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: range(2, 30, 3)

Ex:



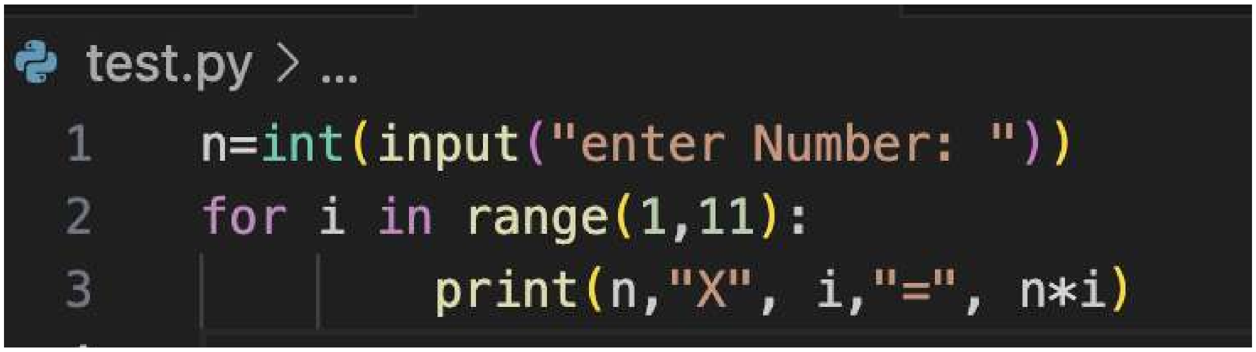
```python
for x in range(2, 30, 3):
    print(x)
```

Output:


```
2
5
8
11
14
17
20
23
26
29
```

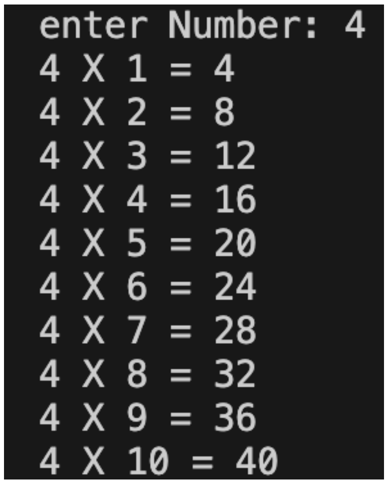
6) For loop Challenges

6.1) Display Multiplication table for given number

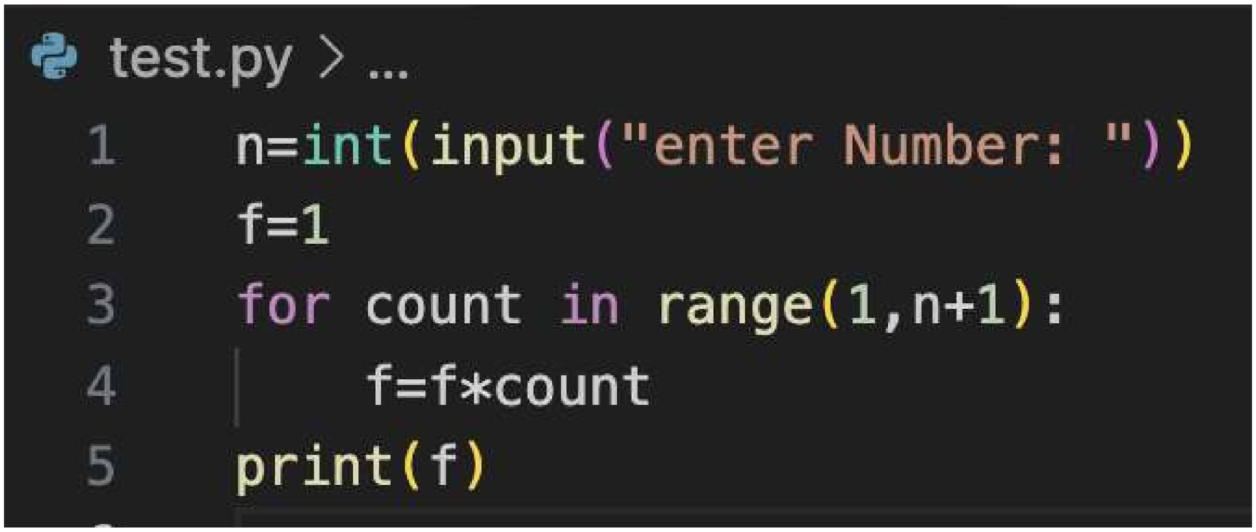


```python
n=int(input("enter Number: "))
for i in range(1,11):
        print(n,"X", i,"=", n*i)
```

Output:


```
enter Number: 4
4 X 1 = 4
4 X 2 = 8
4 X 3 = 12
4 X 4 = 16
4 X 5 = 20
4 X 6 = 24
4 X 7 = 28
4 X 8 = 32
4 X 9 = 36
4 X 10 = 40
```

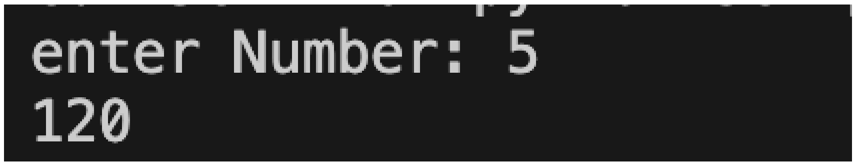
6.2) Find Factorial of given number



test.py > ...

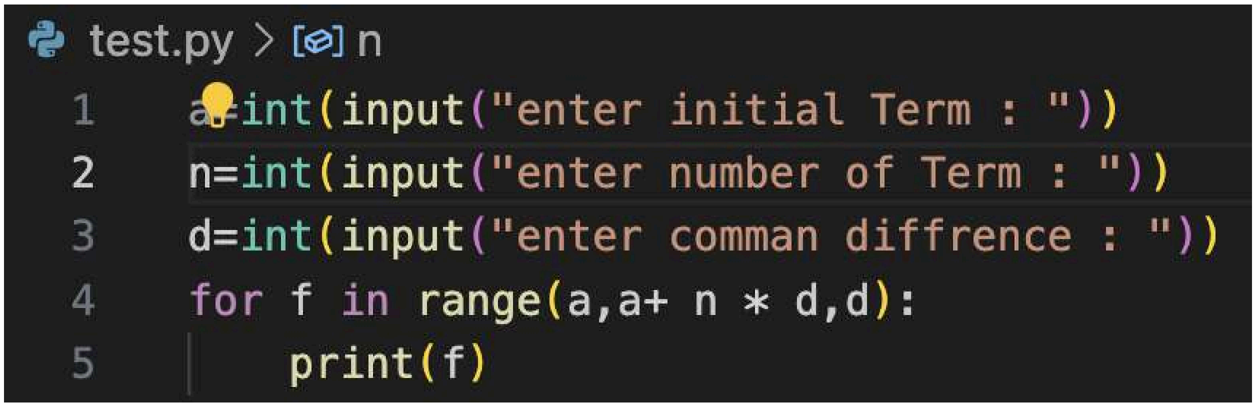
```python
n=int(input("enter Number: "))
f=1
for count in range(1,n+1):
    f=f*count
print(f)
```

Output:



```
enter Number: 5
120
```

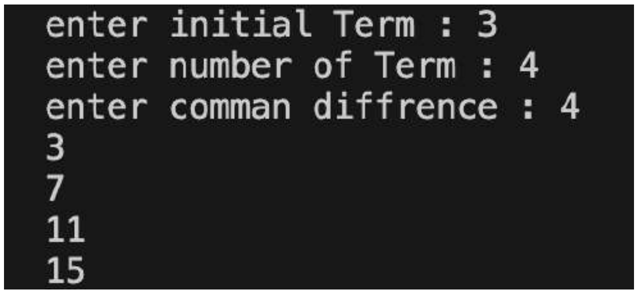
6.3) Print n term of AP (Arithmetic progression) series



test.py > n

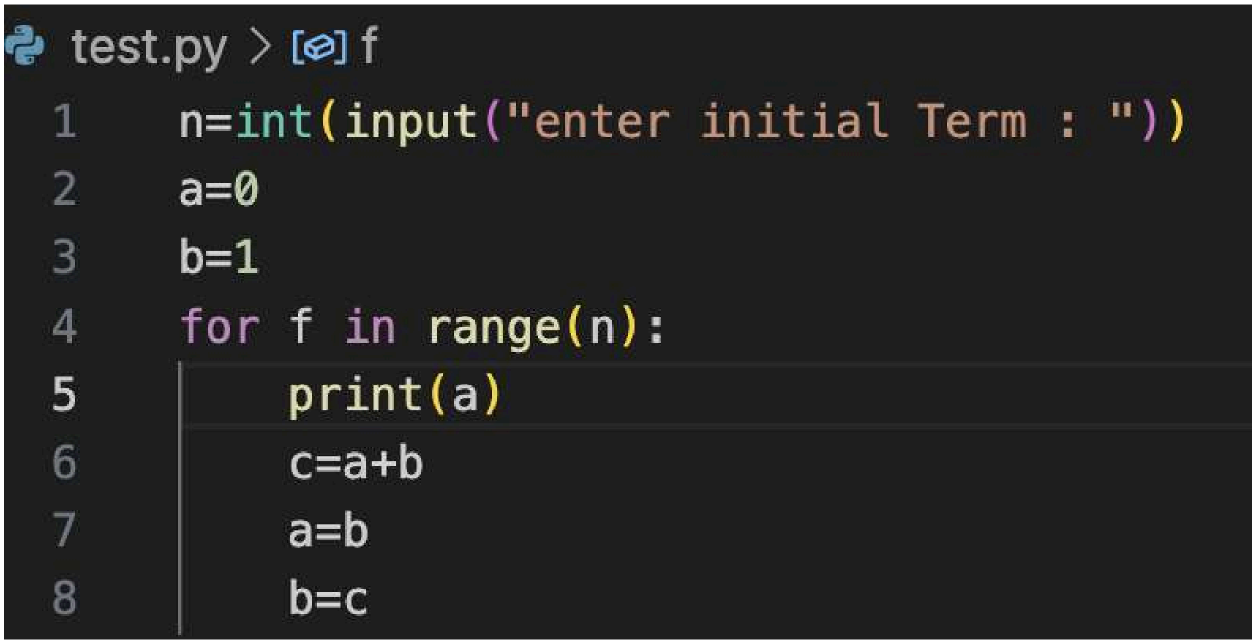
```python
a=int(input("enter initial Term : "))
n=int(input("enter number of Term : "))
d=int(input("enter comman diffrence : "))
for f in range(a,a+ n * d,d):
    print(f)
```

Output:



```
enter initial Term : 3
enter number of Term : 4
enter comman diffrence : 4
3
7
11
15
```

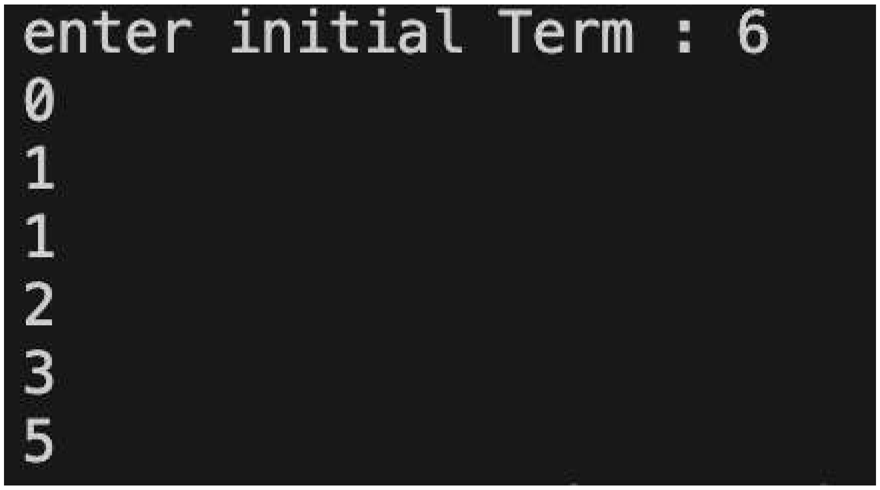
6.4) print n term of fibonacci series



test.py > f

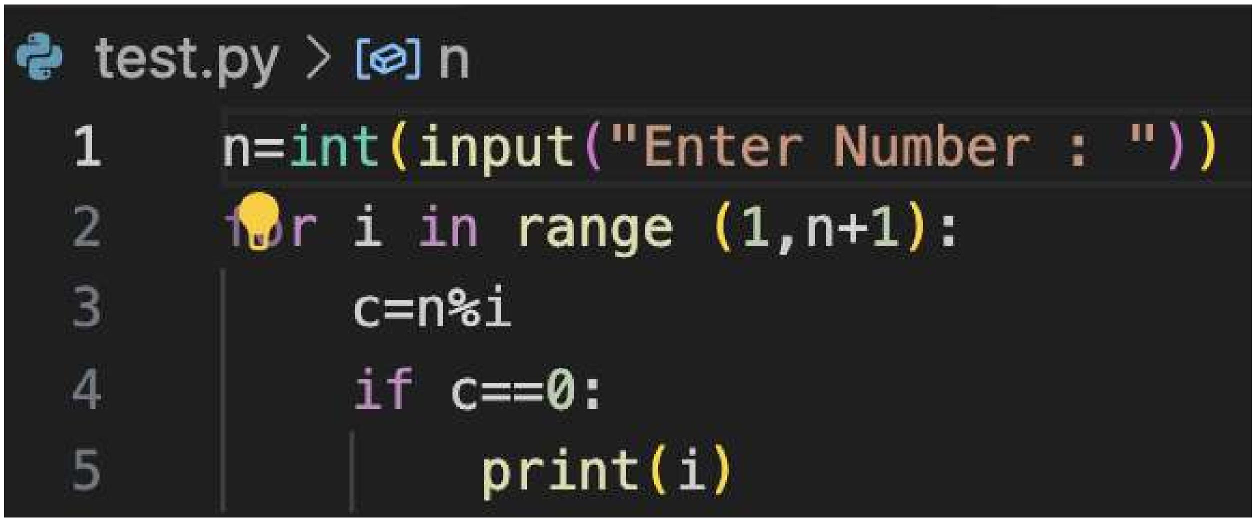
```python
n=int(input("enter initial Term : "))
a=0
b=1
for f in range(n):
    print(a)
    c=a+b
    a=b
    b=c
```

Output:



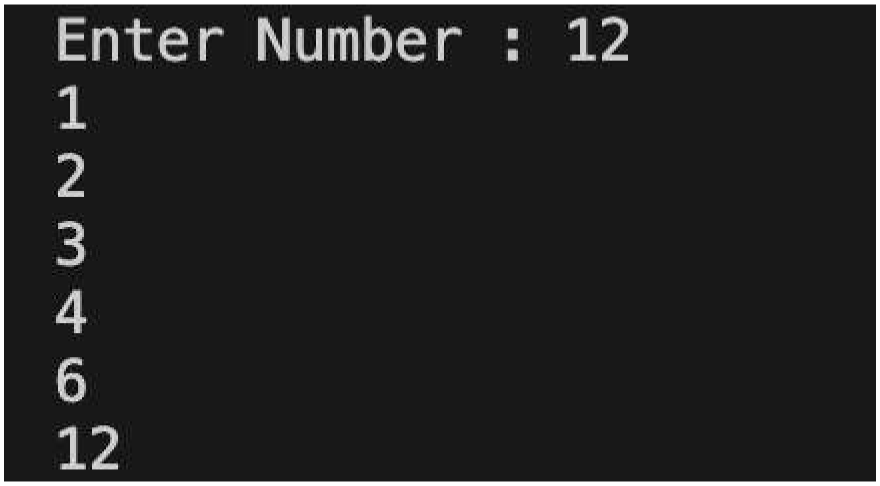
```
enter initial Term : 6
0
1
1
2
3
5
```

6.5) Find the factor of number



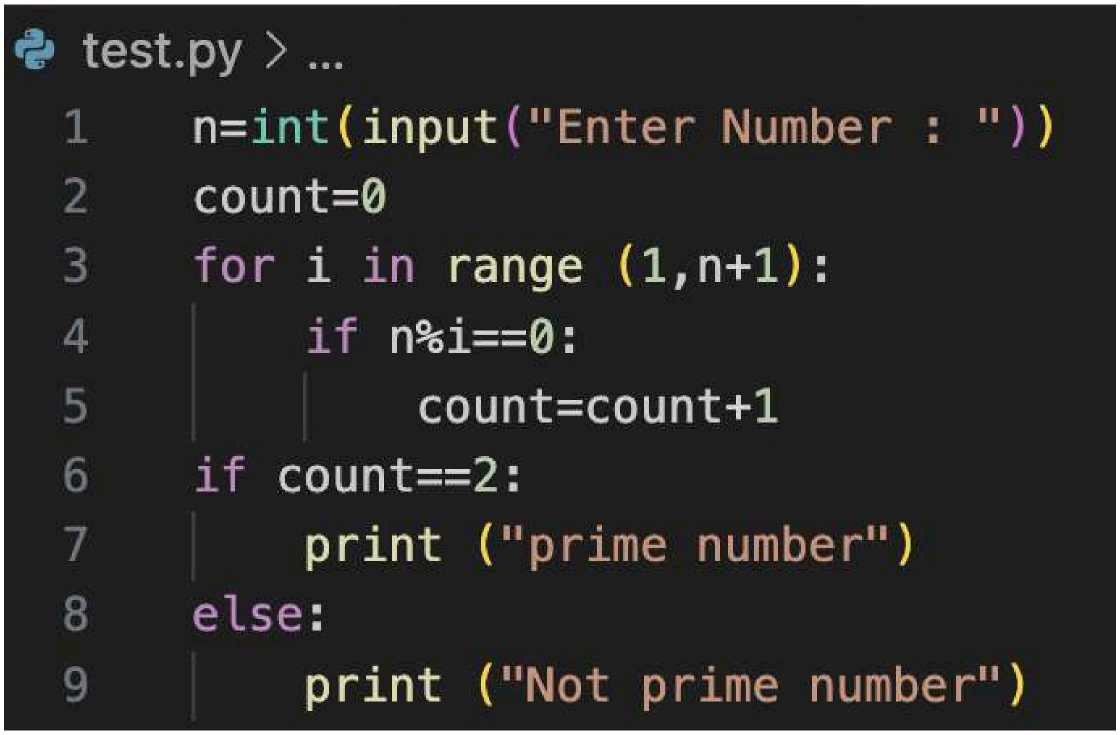
```python
n=int(input("Enter Number : "))
for i in range (1,n+1):
    c=n%i
    if c==0:
        print(i)
```

Output:



```
Enter Number : 12
1
2
3
4
6
12
```

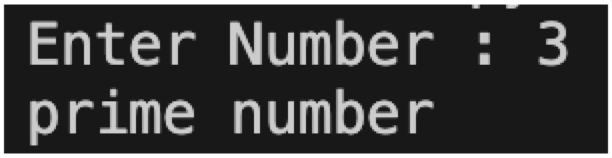
6.6) check if number is prime or not



test.py > ...

```python
n=int(input("Enter Number : "))
count=0
for i in range (1,n+1):
    if n%i==0:
        count=count+1
if count==2:
    print ("prime number")
else:
    print ("Not prime number")
```

Output:

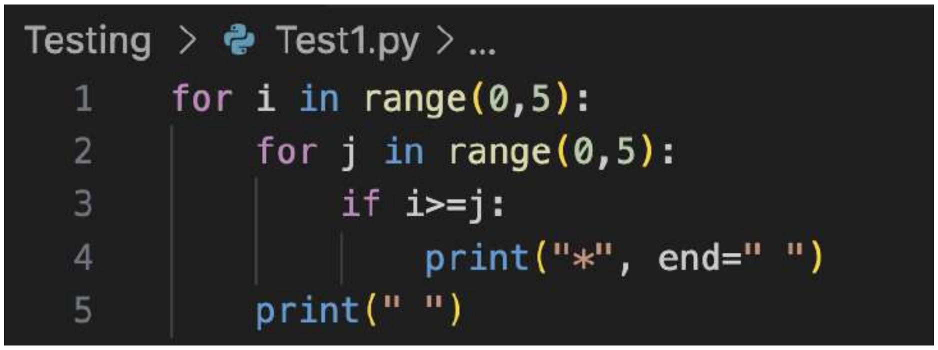


```
Enter Number : 3
prime number
```

7) Break , Continue, Pass with For loop

Break, continue, and pass work like same as above explain in while loop
Refer (infinite loop - Break - Continue - Pass with while loop)

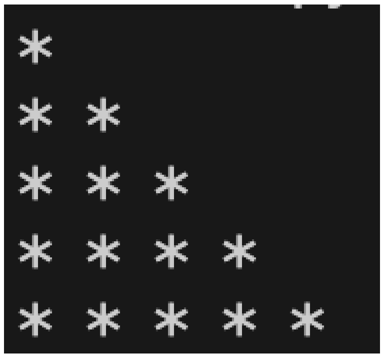
8) Nested Loop: A nested loop is a loop inside another loop. This means that one loop is contained within another loop. Each time the outer loop executes, the inner loop will execute completely.
Example 1:



Testing > Test1.py > ...

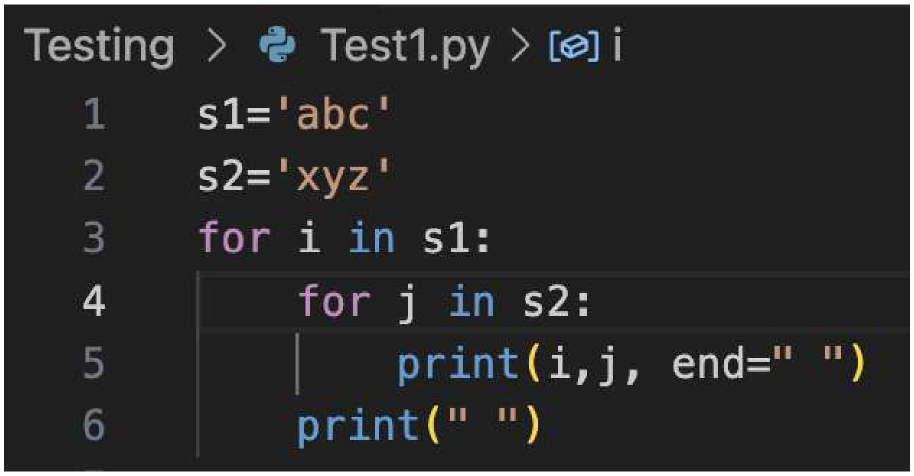
```python
for i in range(0,5):
    for j in range(0,5):
        if i>=j:
            print("*", end=" ")
    print(" ")
```

Output



```
*
* *
* * *
* * * *
* * * * *
```

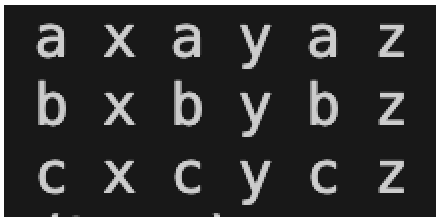
Example 2:



Testing > Test1.py > i

```python
s1='abc'
s2='xyz'
for i in s1:
    for j in s2:
        print(i,j, end=" ")
    print(" ")
```

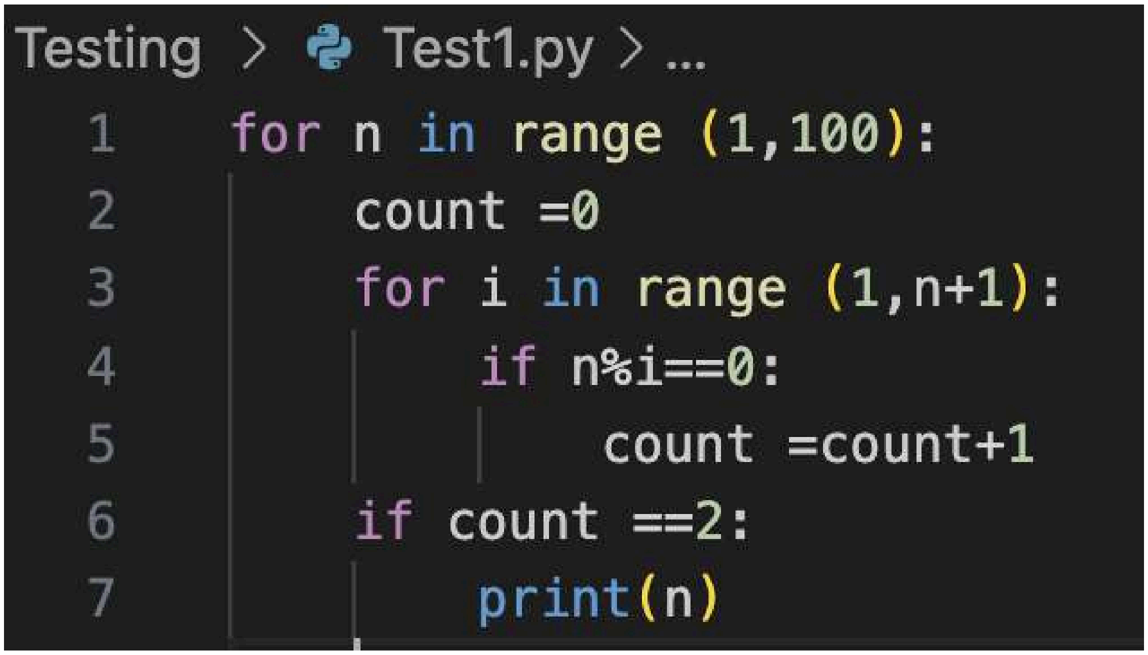
Output:



```
a x a y a z
b x b y b z
c x c y c z
```

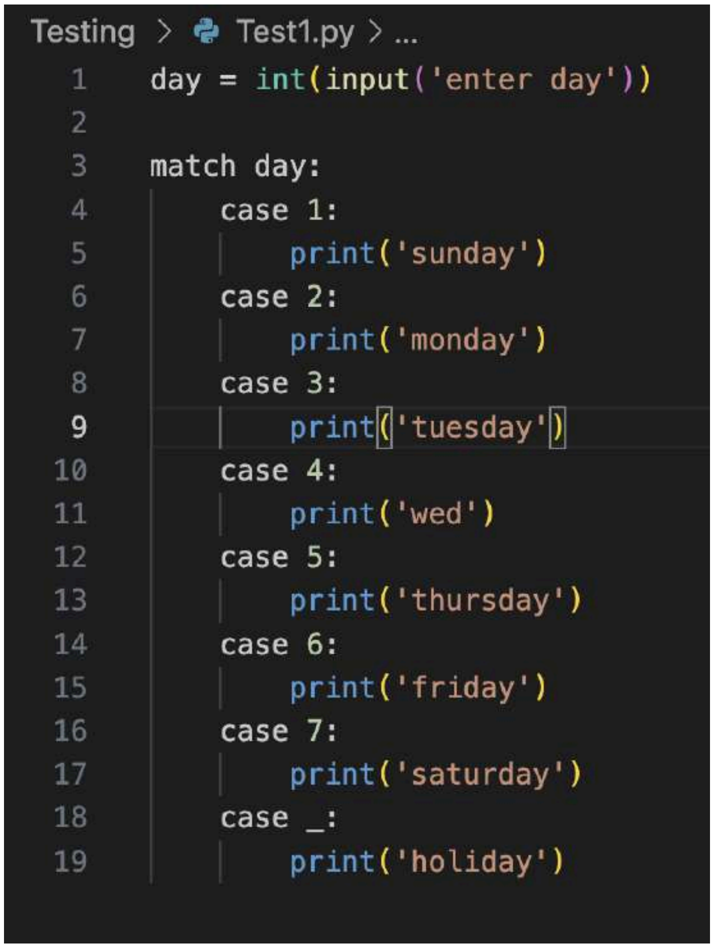
8.1) challenges on Nested loop

8.1.1) Print prime number from 1 to 100 (prime number is number is only divided by 1 and 2)



```python
for n in range (1,100):
    count =0
    for i in range (1,n+1):
        if n%i==0:
            count =count+1
    if count ==2:
        print(n)
```

9) Match Case



```python
day = int(input('enter day'))

match day:
    case 1:
        print('sunday')
    case 2:
        print('monday')
    case 3:
        print('tuesday')
    case 4:
        print('wed')
    case 5:
        print('thursday')
    case 6:
        print('friday')
    case 7:
        print('saturday')
    case _:
        print('holiday')
```

Strings and its Methods

1) introduction to string

a string is a sequence of characters enclosed within either single quotes (') or double quotes (").
EX: a='cv' b="Jhon's"
Note: if Apostrophe ' is used in string we have to use double quotes

2) Operator on string

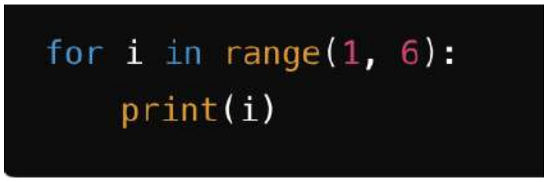
2.1) concatenation

It means combining two or more strings/lists into one longer string. You can do this using the + operator:
Ex: a= 'how'+'are'+'you' (how are you)

2.2) Repetition

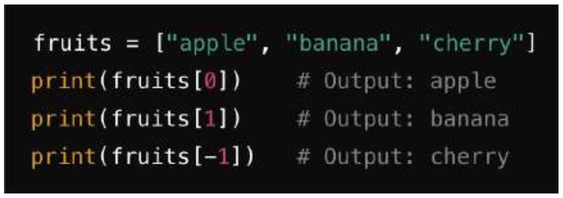
Repetition in Python is commonly handled using loops, which allow you to execute a block of code multiple times

Ex:



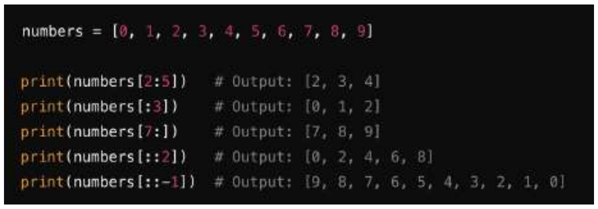
```
for i in range(1, 6):
    print(i)
```

2.3) indexing: Indexing in Python refers to accessing elements of a sequence using their positions. Python sequences that support indexing include lists, tuples, strings, and more
EX:

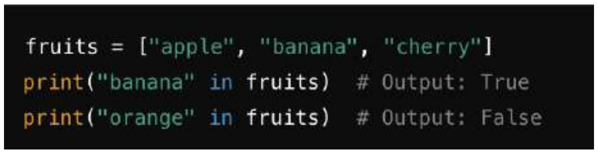


```
fruits = ["apple", "banana", "cherry"]
print(fruits[0])    # Output: apple
print(fruits[1])    # Output: banana
print(fruits[-1])   # Output: cherry
```

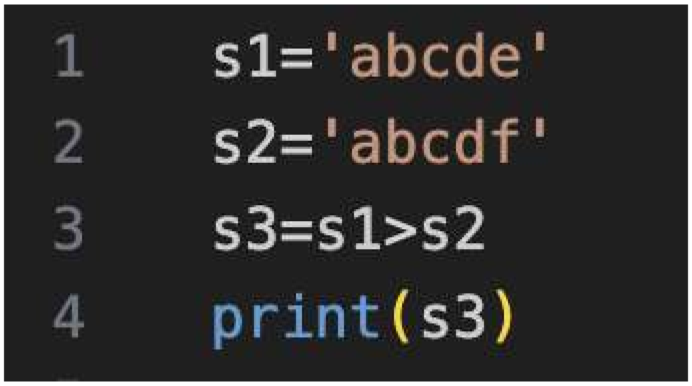
2.4) slicing: Slicing in Python allows you to access a subset of elements from sequences like lists, tuples, and strings. Slicing is done using the colon : operator within square brackets [].
EX:


```
numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

print(numbers[2:5])   # Output: [2, 3, 4]
print(numbers[:3])    # Output: [0, 1, 2]
print(numbers[7:])    # Output: [7, 8, 9]
print(numbers[::2])   # Output: [0, 2, 4, 6, 8]
print(numbers[::-1])  # Output: [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

2.5) in and not in : used to check if a value is present in a sequence (like a list, tuple, string, or set). These operators return True or False based on whether the specified value is found in the sequence.
Ex:


```
fruits = ["apple", "banana", "cherry"]
print("banana" in fruits)  # Output: True
print("orange" in fruits)  # Output: False
```

3) Relational Operator on String


```
s1='abcde'
s2='abcdf'
s3=s1>s2
print(s3)
```

4) String Methods

Mthods	Description
capitalize()	Converts the first character to uppercase
casefold()	Converts string into lower case
center()	Returns a centered string
count()	Returns the number of times a specified value occurs in a string
encode()	Returns an encoded version of the string
endswith()	Returns true if the string ends with the specified value
expandtabs()	Sets the tab size of the string
find()	Searches the string for a specified value and returns the position of where

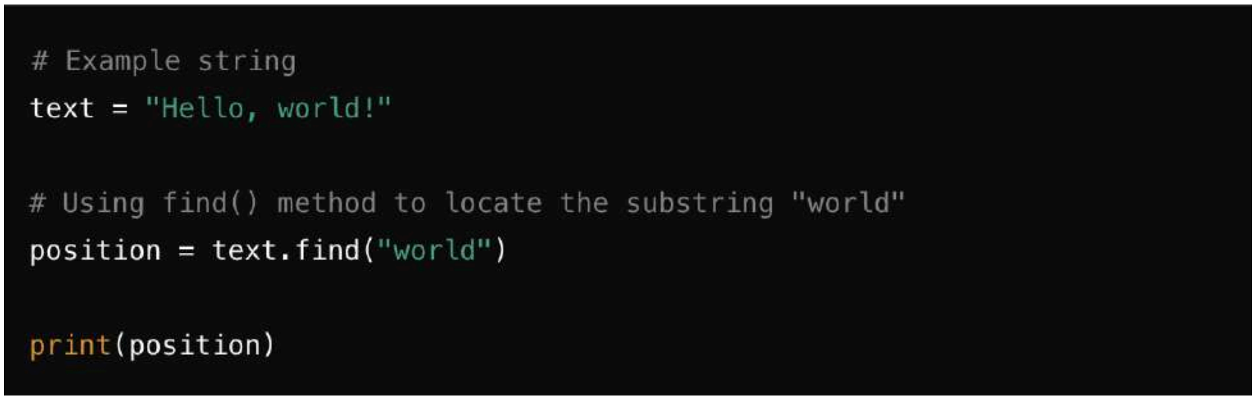
	it was found
format()	Formats specified values in a string
format_map()	Formats specified values in a string
index()	Searches the string for a specified value and returns the position of where it was found
isalnum()	Returns True if all characters in the string are alphanumeric
isalpha()	Returns True if all characters in the string are in the alphabet
isascii()	Returns True if all characters in the string are ascii characters
isdecimal()	Returns True if all characters in the string are decimals
isdigit()	Returns True if all characters in the string are digits
isidentifier()	Returns True if the string is an identifier
islower()	Returns True if all characters in the string are lower case
isnumeric()	Returns True if all characters in the string are numeric
isprintable()	Returns True if all characters in the string are printable
isspace()	Returns True if all characters in the string are whitespaces
istitle()	Returns True if the string follows the rules of a title
isupper()	Returns True if all characters in the string are upper case
join()	Converts the elements of an iterable into a string
ljust()	Returns a left justified version of the string
lower()	Converts a string into lower case
lstrip()	Returns a left trim version of the string
maketrans()	Returns a translation table to be used in translations
partition()	Returns a tuple where the string is parted into three parts
replace()	Returns a string where a specified value is replaced with a specified value
rfind()	Searches the string for a specified value and returns the last position of where it was found
rindex()	Searches the string for a specified value and returns the last position of where it was found
rjust()	Returns a right justified version of the string
rpartition()	Returns a tuple where the string is parted into three parts
rsplit()	Splits the string at the specified separator, and returns a list
rstrip()	Returns a right trim version of the string
split()	Splits the string at the specified separator, and returns a list
splitlines()	Splits the string at line breaks and returns a list
startswith()	Returns true if the string starts with the specified value

strip()	Returns a trimmed version of the string
swapcase()	Swaps cases, lower case becomes upper case and vice versa
title()	Converts the first character of each word to upper case
translate()	Returns a translated string
upper()	Converts a string into upper case
zfill()	Fills the string with a specified number of 0 values at the beginning

5) String method Examples

5.1 Find() : this method used to find the substring in the string

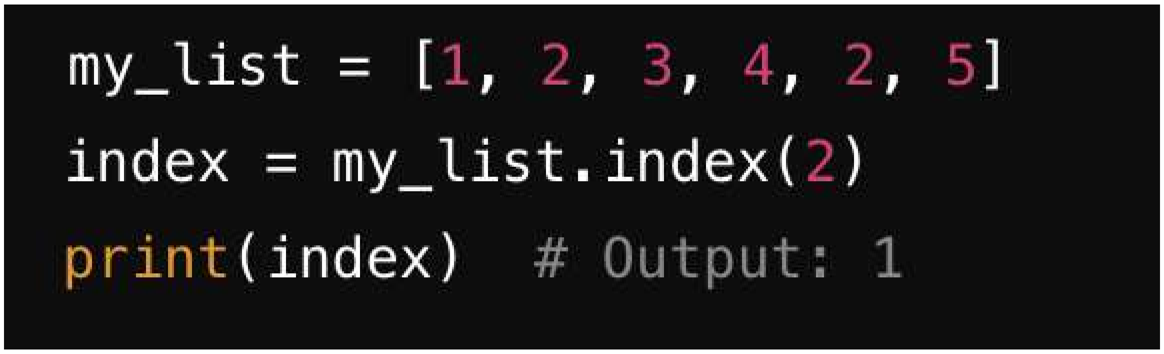
Ex:



```
# Example string
text = "Hello, world!"

# Using find() method to locate the substring "world"
position = text.find("world")

print(position)
```

5.2 index(): the index() method is used to find the index of the first occurrence of a specified value within a sequence such as a list, tuple, or string

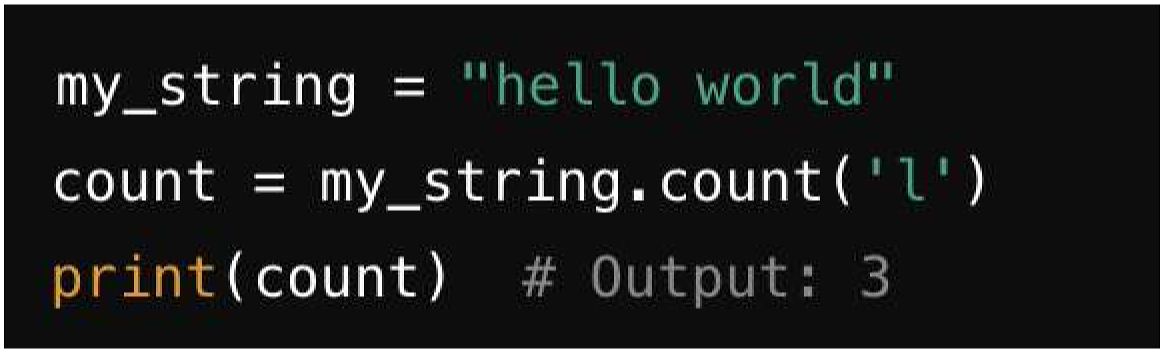
EX:



```
my_list = [1, 2, 3, 4, 2, 5]
index = my_list.index(2)
print(index)  # Output: 1
```

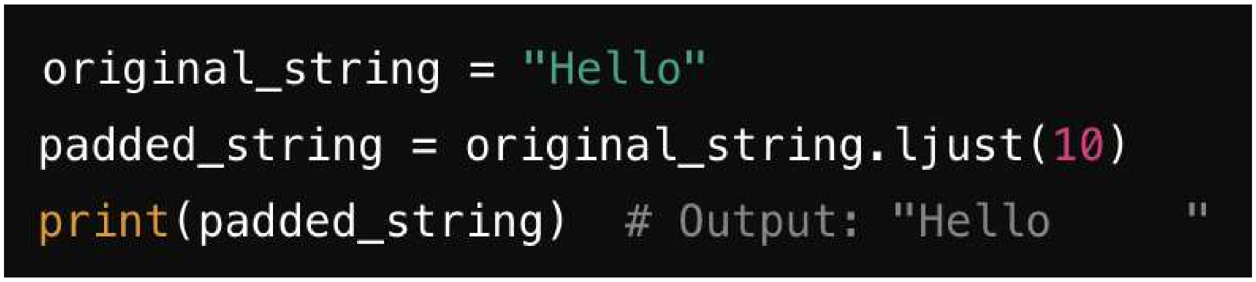
5.3 count(): The count() method in Python is used to count the number of occurrences of a specified element in a sequence such as a list, tuple, or string

Ex:



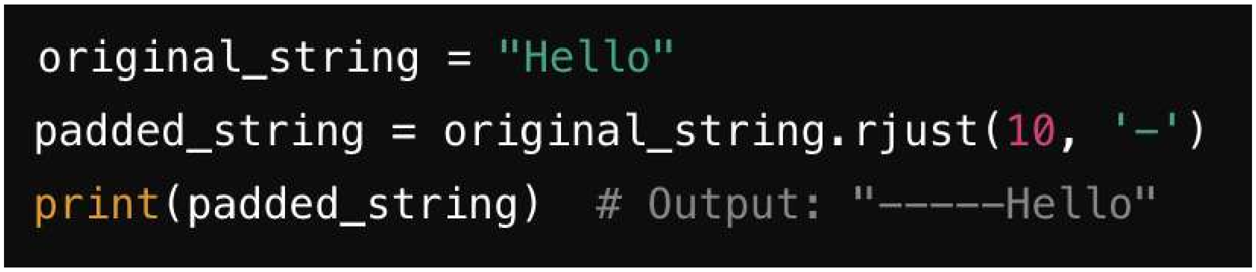
```python
my_string = "hello world"
count = my_string.count('l')
print(count)  # Output: 3
```

5.4 ljust():
EX:



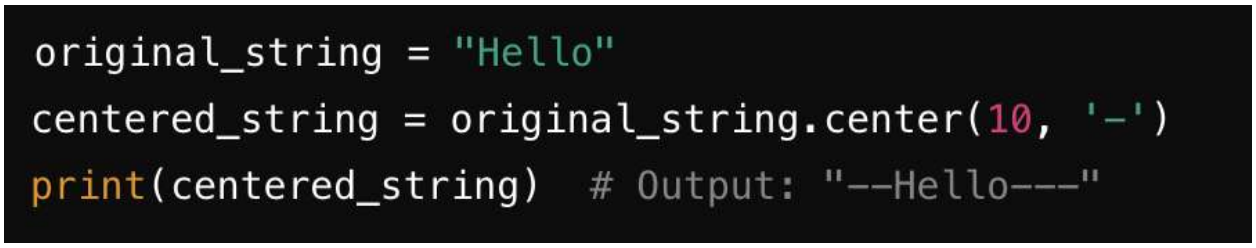
```python
original_string = "Hello"
padded_string = original_string.ljust(10)
print(padded_string)  # Output: "Hello     "
```

5.5 rjust():
Ex:



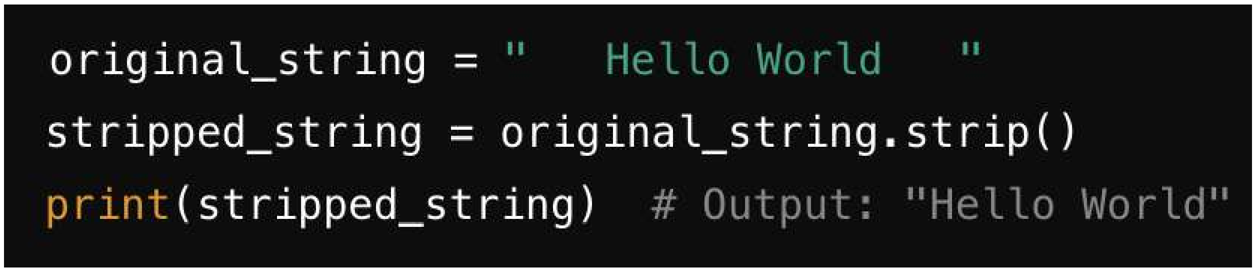
```python
original_string = "Hello"
padded_string = original_string.rjust(10, '-')
print(padded_string)  # Output: "-----Hello"
```

5.6 Center():
Ex:



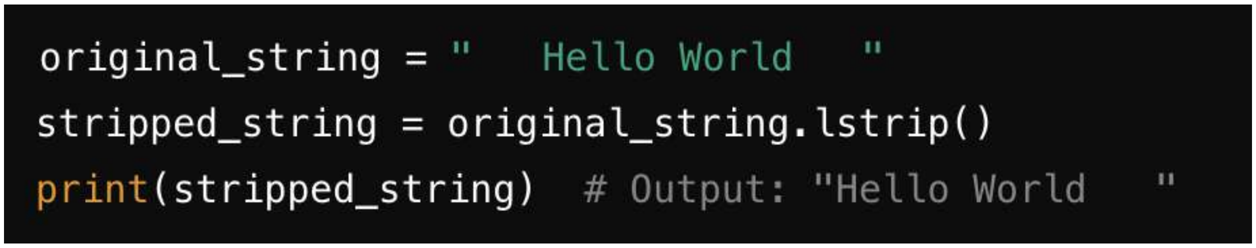
```python
original_string = "Hello"
centered_string = original_string.center(10, '-')
print(centered_string)  # Output: "--Hello---"
```

5.7 strip(): remove space from both side
Ex:



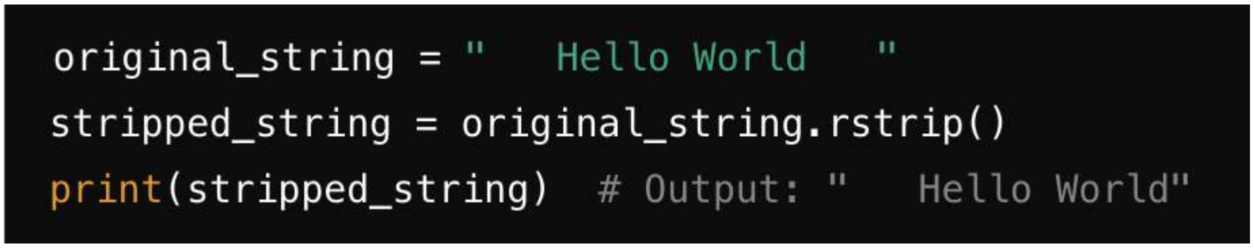
```python
original_string = "   Hello World   "
stripped_string = original_string.strip()
print(stripped_string)  # Output: "Hello World"
```

5.8 lstrip(): remove space from left side



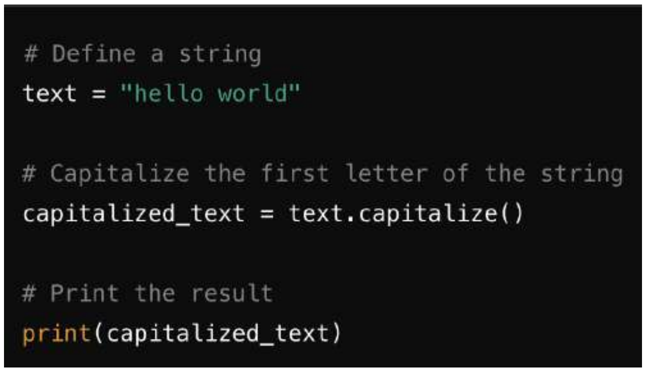
```python
original_string = "   Hello World   "
stripped_string = original_string.lstrip()
print(stripped_string)  # Output: "Hello World   "
```

5.9 rstrip(): remove space from right side



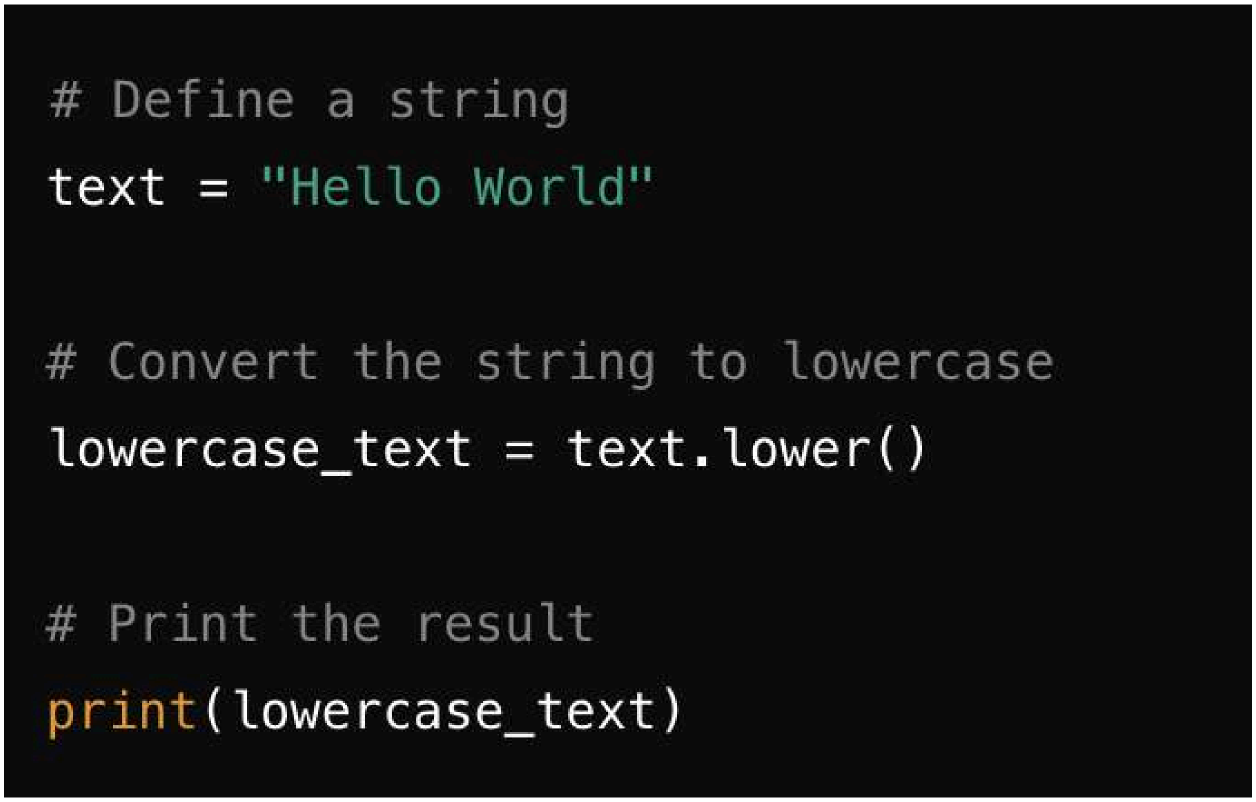
```python
original_string = "   Hello World   "
stripped_string = original_string.rstrip()
print(stripped_string)  # Output: "   Hello World"
```

5.10 Capitalize():



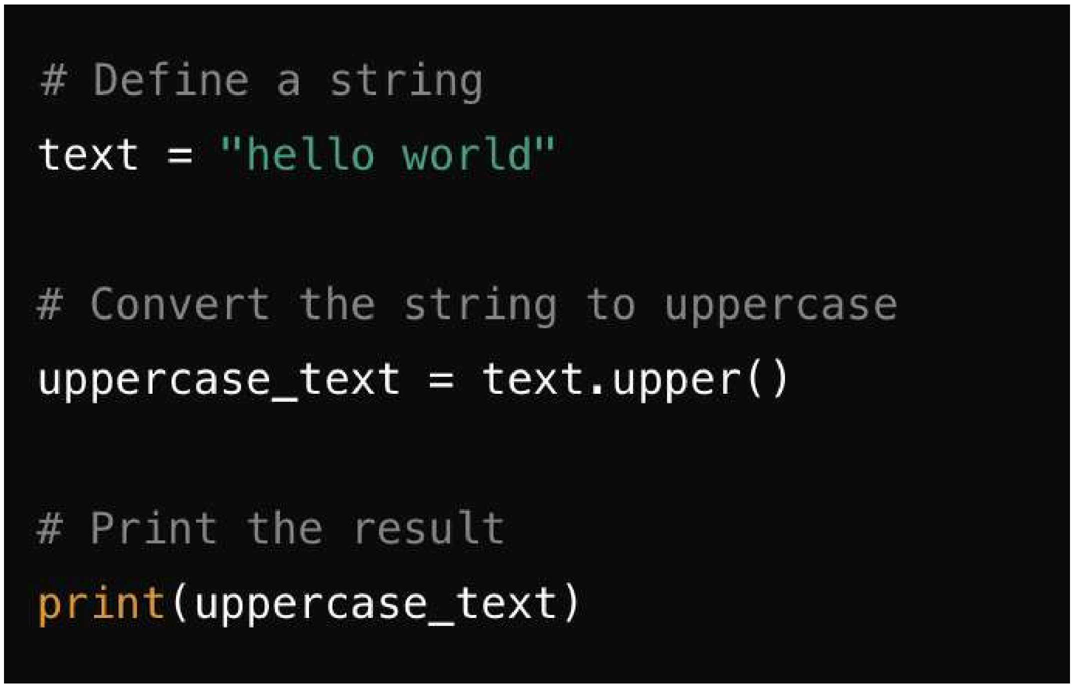
```python
# Define a string
text = "hello world"

# Capitalize the first letter of the string
capitalized_text = text.capitalize()

# Print the result
print(capitalized_text)
```

5.11 Lower():



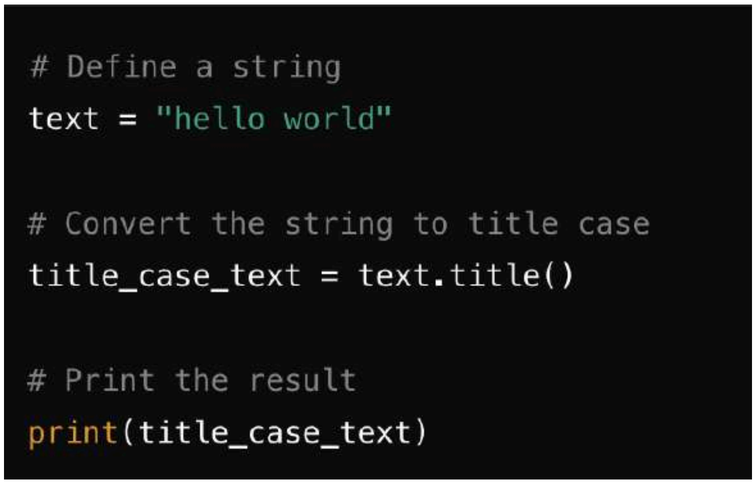
```python
# Define a string
text = "Hello World"

# Convert the string to lowercase
lowercase_text = text.lower()

# Print the result
print(lowercase_text)
```

5.12 Upper():



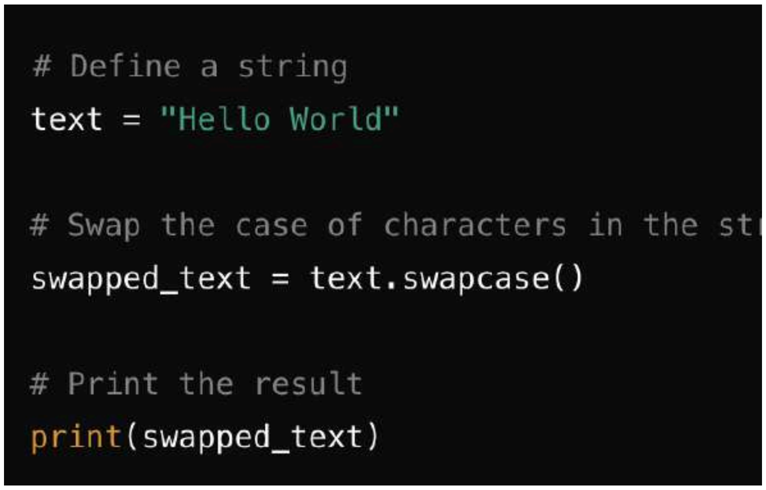
```python
# Define a string
text = "hello world"

# Convert the string to uppercase
uppercase_text = text.upper()

# Print the result
print(uppercase_text)
```

5.13 title():



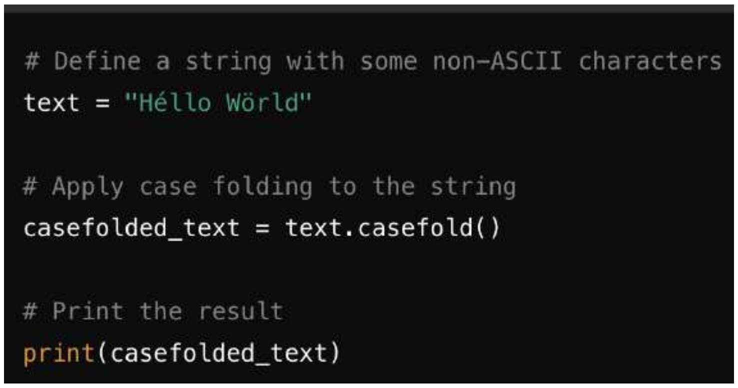
```python
# Define a string
text = "hello world"

# Convert the string to title case
title_case_text = text.title()

# Print the result
print(title_case_text)
```

5.14 swapcase():



```python
# Define a string
text = "Hello World"

# Swap the case of characters in the st
swapped_text = text.swapcase()

# Print the result
print(swapped_text)
```

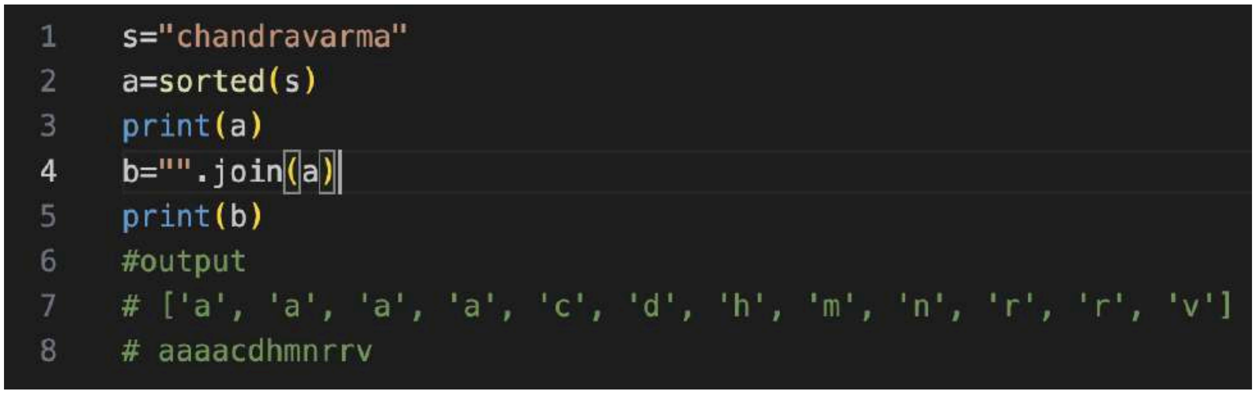
5.15 casefold()



```python
# Define a string with some non-ASCII characters
text = "Héllo Wörld"

# Apply case folding to the string
casefolded_text = text.casefold()

# Print the result
print(casefolded_text)
```

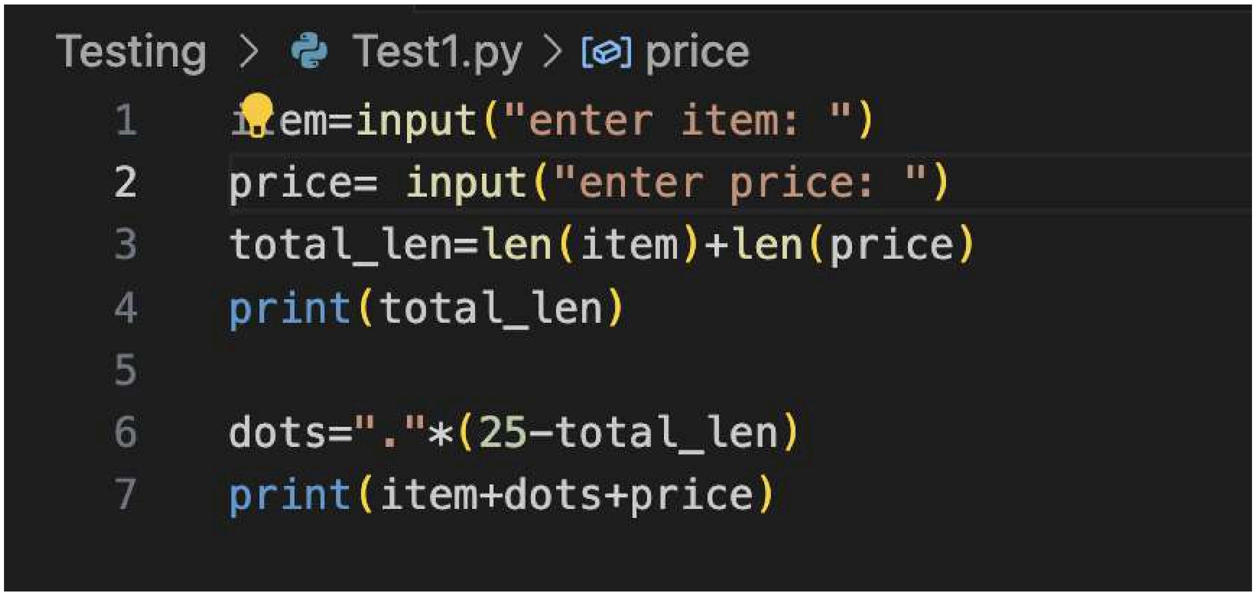
6) Challenges

6.1) Sorting letter of string



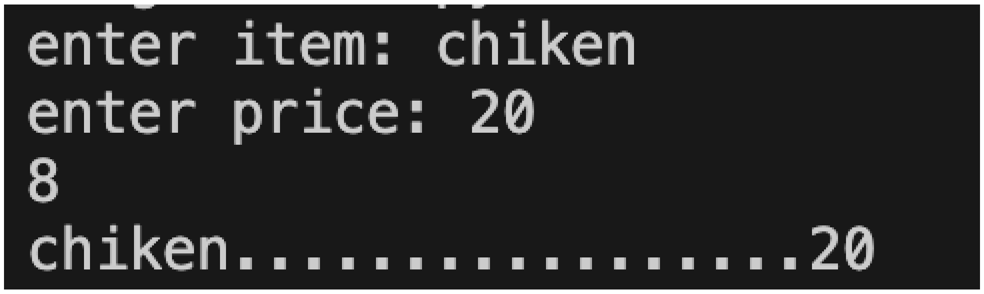
```
s="chandravarma"
a=sorted(s)
print(a)
b="".join(a)
print(b)
#output
# ['a', 'a', 'a', 'a', 'c', 'd', 'h', 'm', 'n', 'r', 'r', 'v']
# aaaacdhmnrrv
```

6.2) Displaying Data in give format



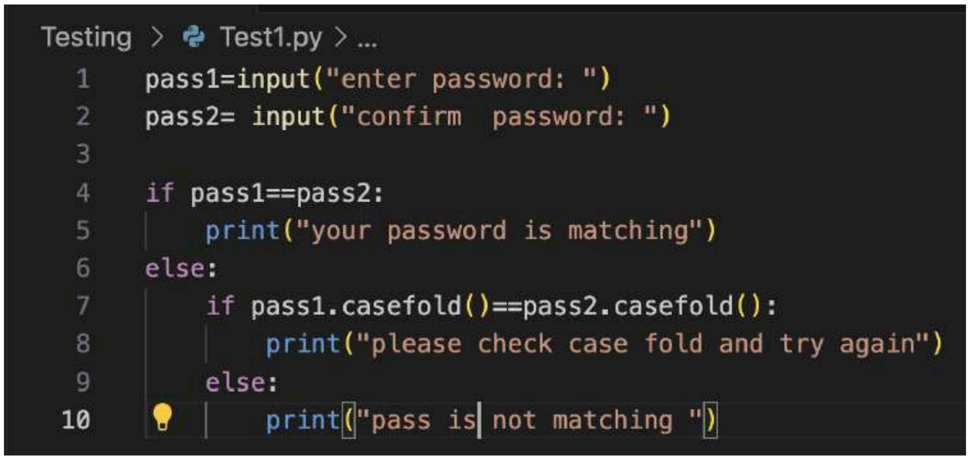
```
item=input("enter item: ")
price= input("enter price: ")
total_len=len(item)+len(price)
print(total_len)

dots="."*(25-total_len)
print(item+dots+price)
```

Output:



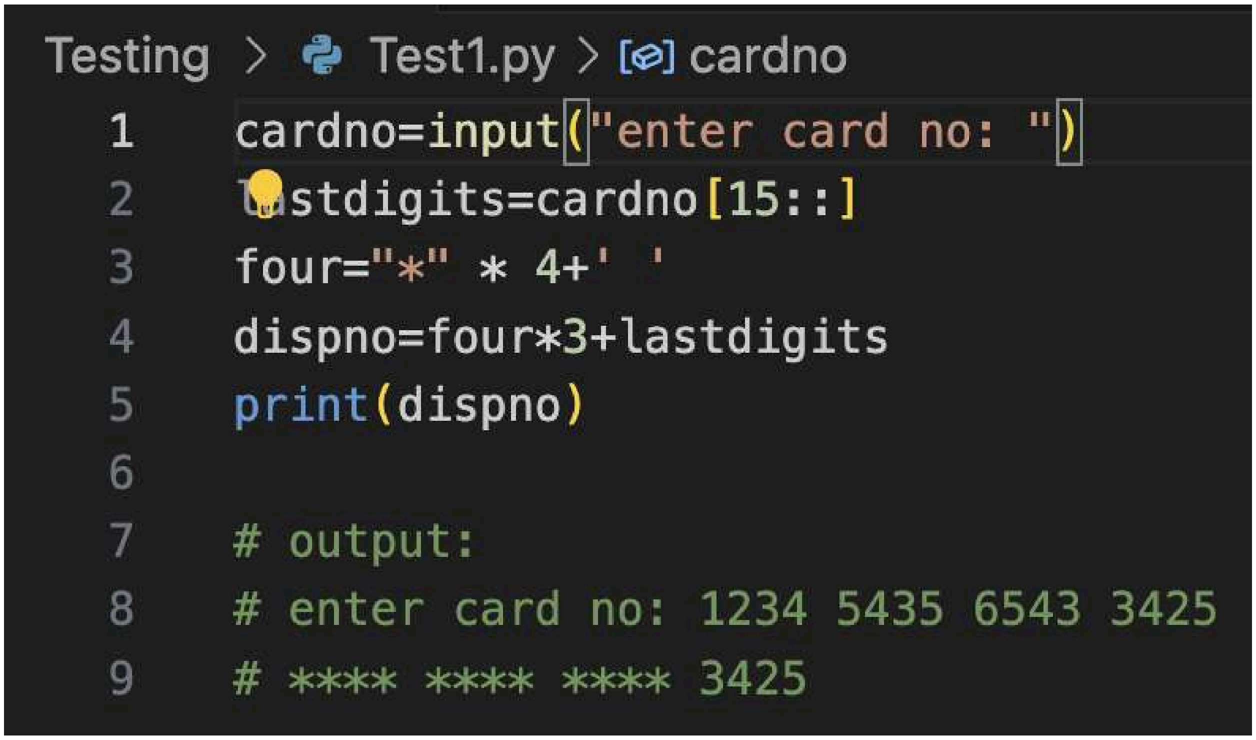
```
enter item: chiken
enter price: 20
8
chiken...................20
```

6.3) Check if the password and confirm password are same


Testing > Test1.py > ...

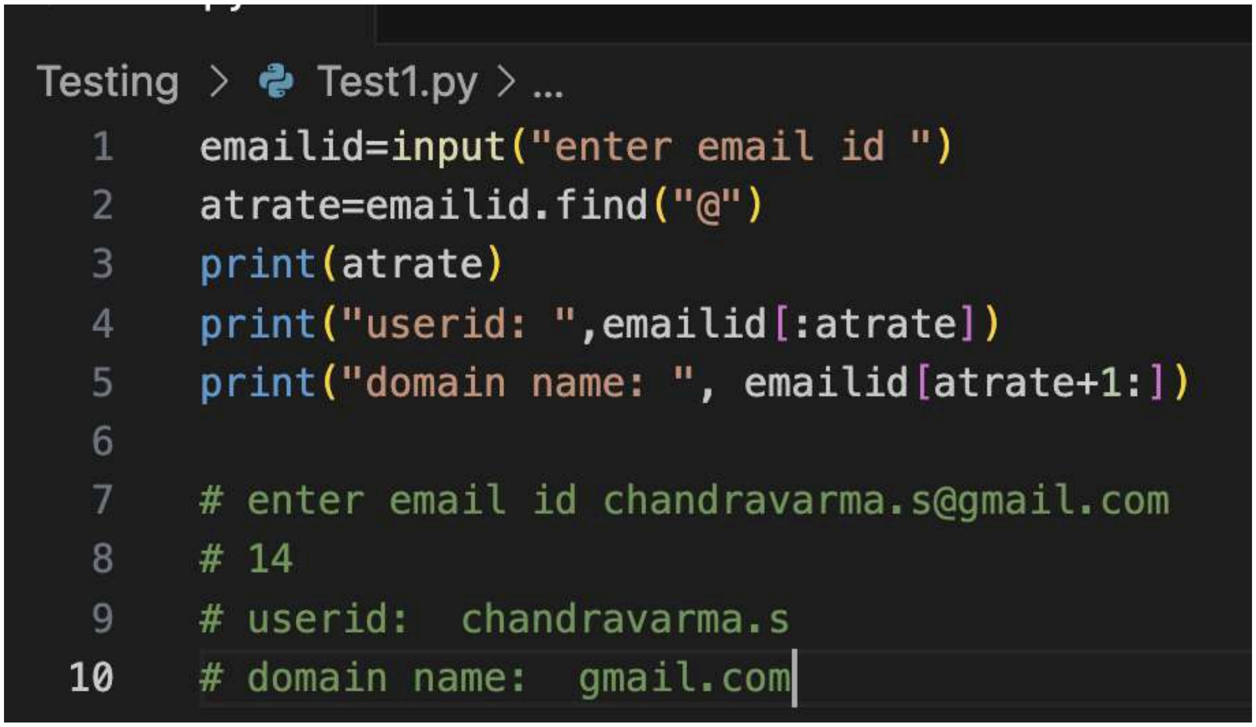
```python
pass1=input("enter password: ")
pass2= input("confirm  password: ")

if pass1==pass2:
    print("your password is matching")
else:
    if pass1.casefold()==pass2.casefold():
        print("please check case fold and try again")
    else:
        print("pass is not matching ")
```

6.4) Display credit card number


Testing > Test1.py > cardno

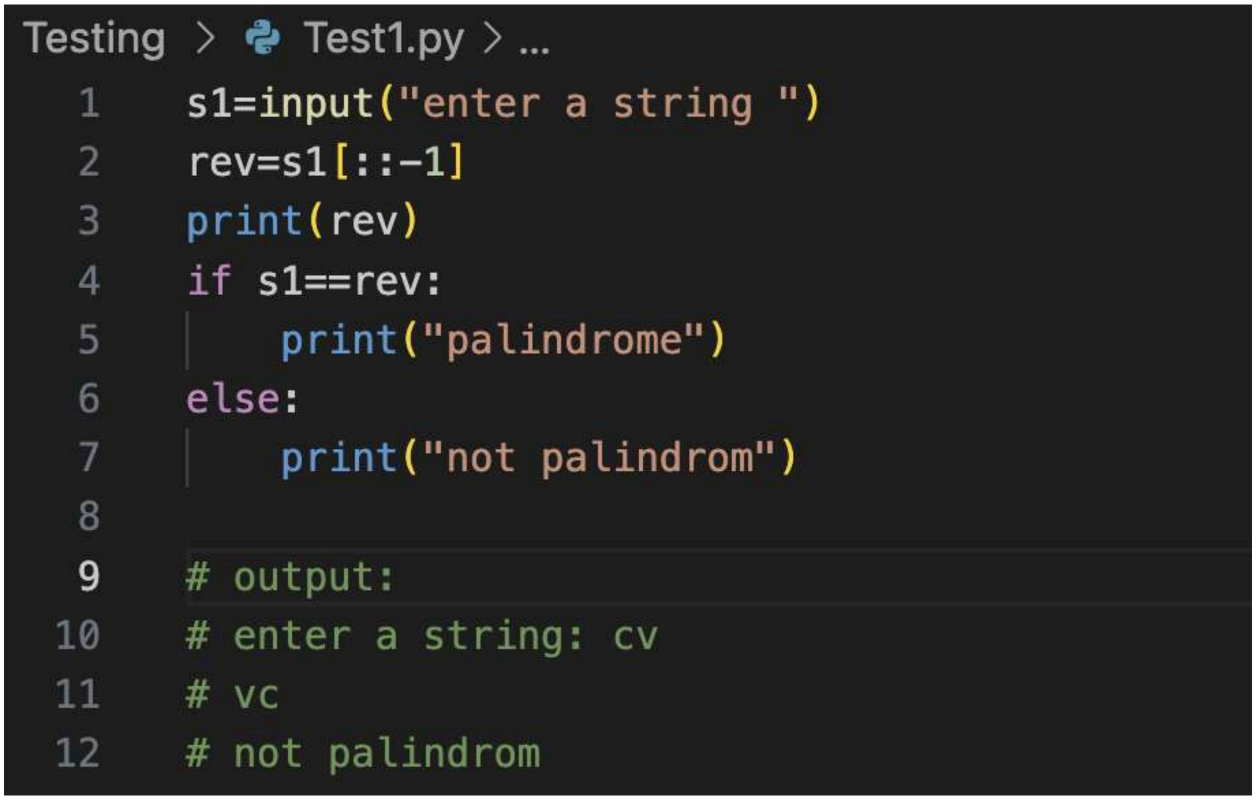
```python
cardno=input("enter card no: ")
lastdigits=cardno[15::]
four="*" * 4+' '
dispno=four*3+lastdigits
print(dispno)

# output:
# enter card no: 1234 5435 6543 3425
# **** **** **** 3425
```

6.4) find user id and domain name from email id


Testing > Test1.py > ...

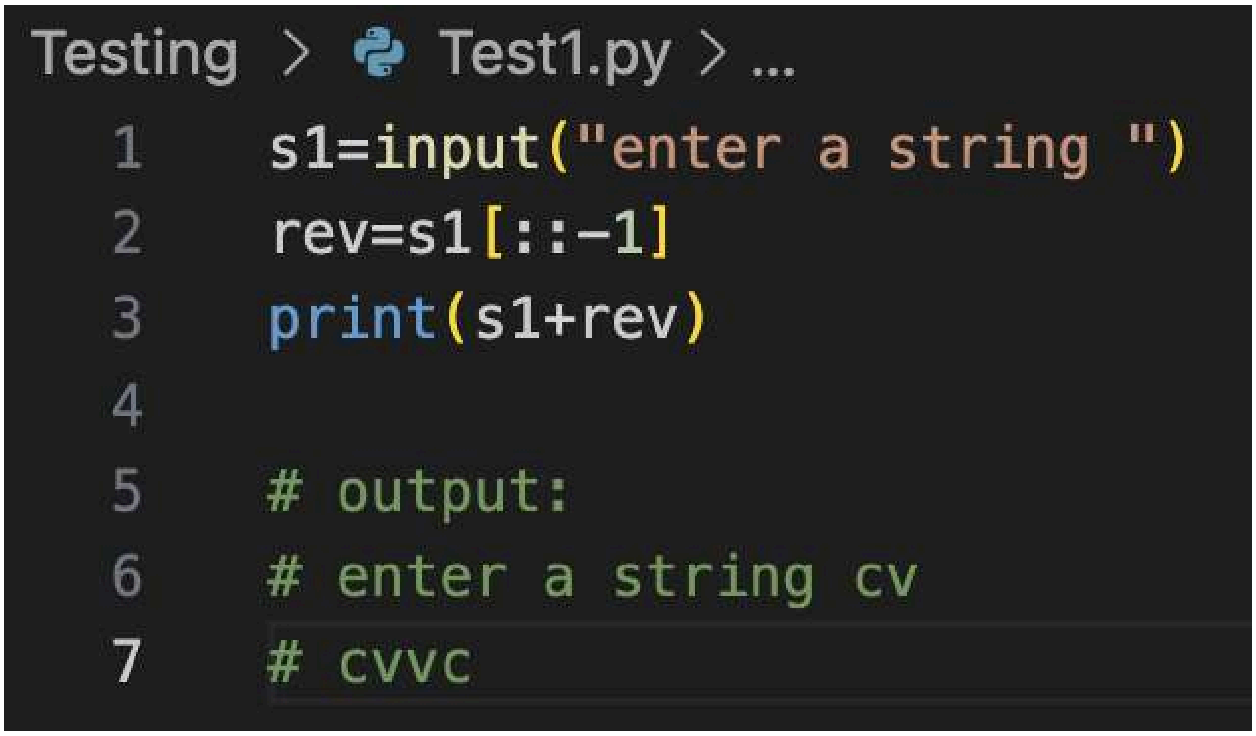
```python
emailid=input("enter email id ")
atrate=emailid.find("@")
print(atrate)
print("userid: ",emailid[:atrate])
print("domain name: ", emailid[atrate+1:])

# enter email id chandravarma.s@gmail.com
# 14
# userid:  chandravarma.s
# domain name:  gmail.com
```

6.4) checking string is palindrome or not


Testing > Test1.py > ...

```python
s1=input("enter a string ")
rev=s1[::-1]
print(rev)
if s1==rev:
    print("palindrome")
else:
    print("not palindrom")

# output:
# enter a string: cv
# vc
# not palindrom
```

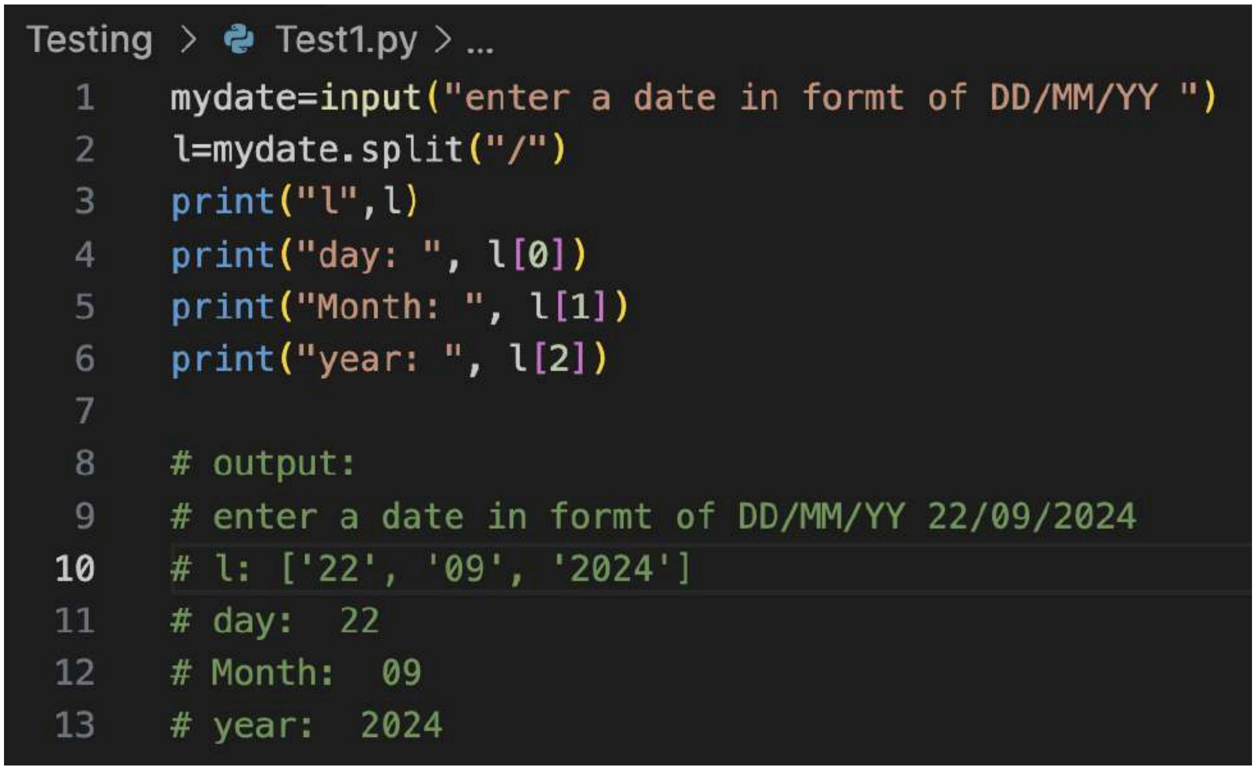
6.4) convert string to palindrome



Testing > Test1.py > ...

```python
s1=input("enter a string ")
rev=s1[::-1]
print(s1+rev)

# output:
# enter a string cv
# cvvc
```

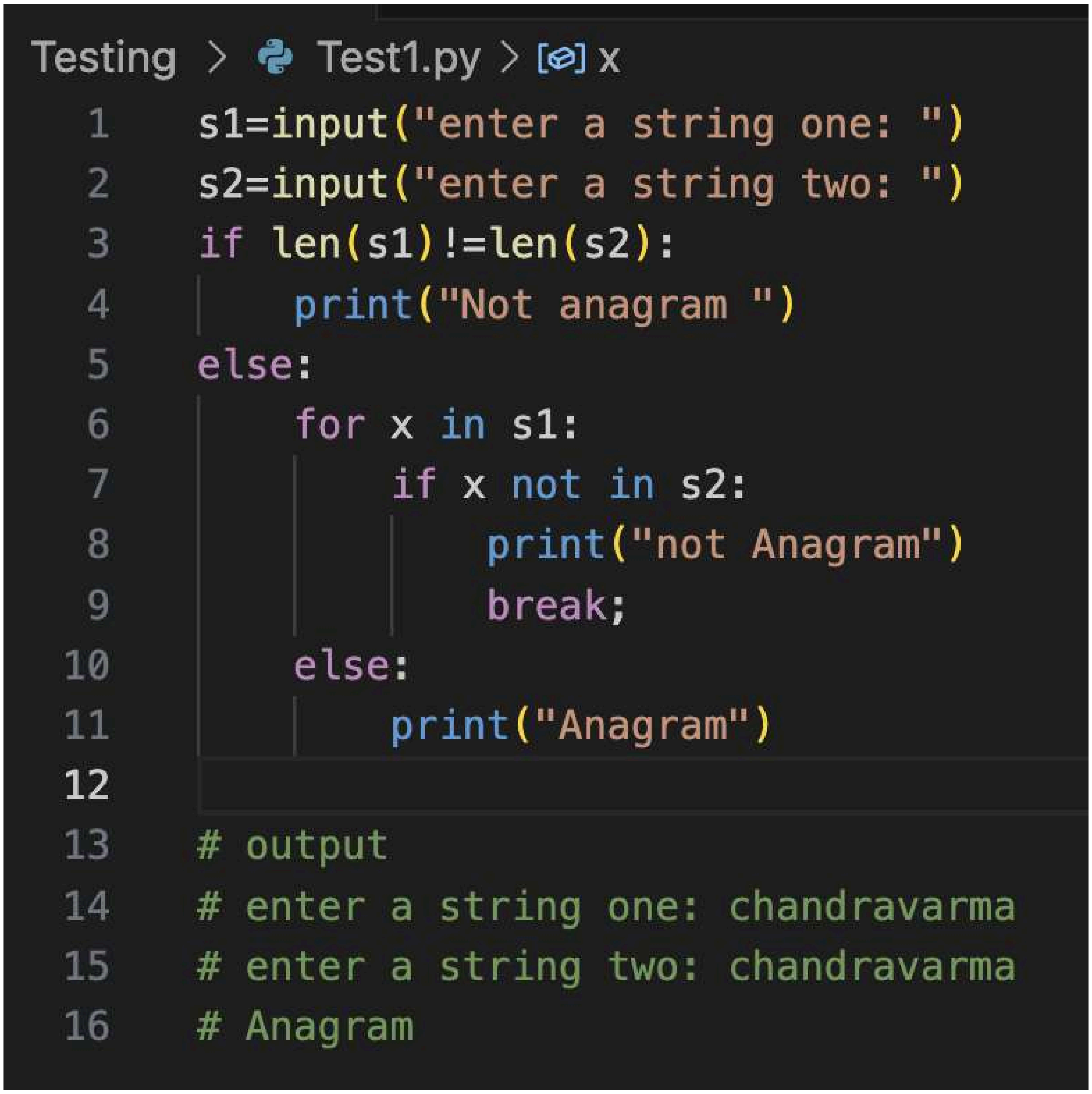
6.4) find date month year from given date



Testing > Test1.py > ...

```python
mydate=input("enter a date in formt of DD/MM/YY ")
l=mydate.split("/")
print("l",l)
print("day: ", l[0])
print("Month: ", l[1])
print("year: ", l[2])

# output:
# enter a date in formt of DD/MM/YY 22/09/2024
# l: ['22', '09', '2024']
# day:  22
# Month:  09
# year:  2024
```

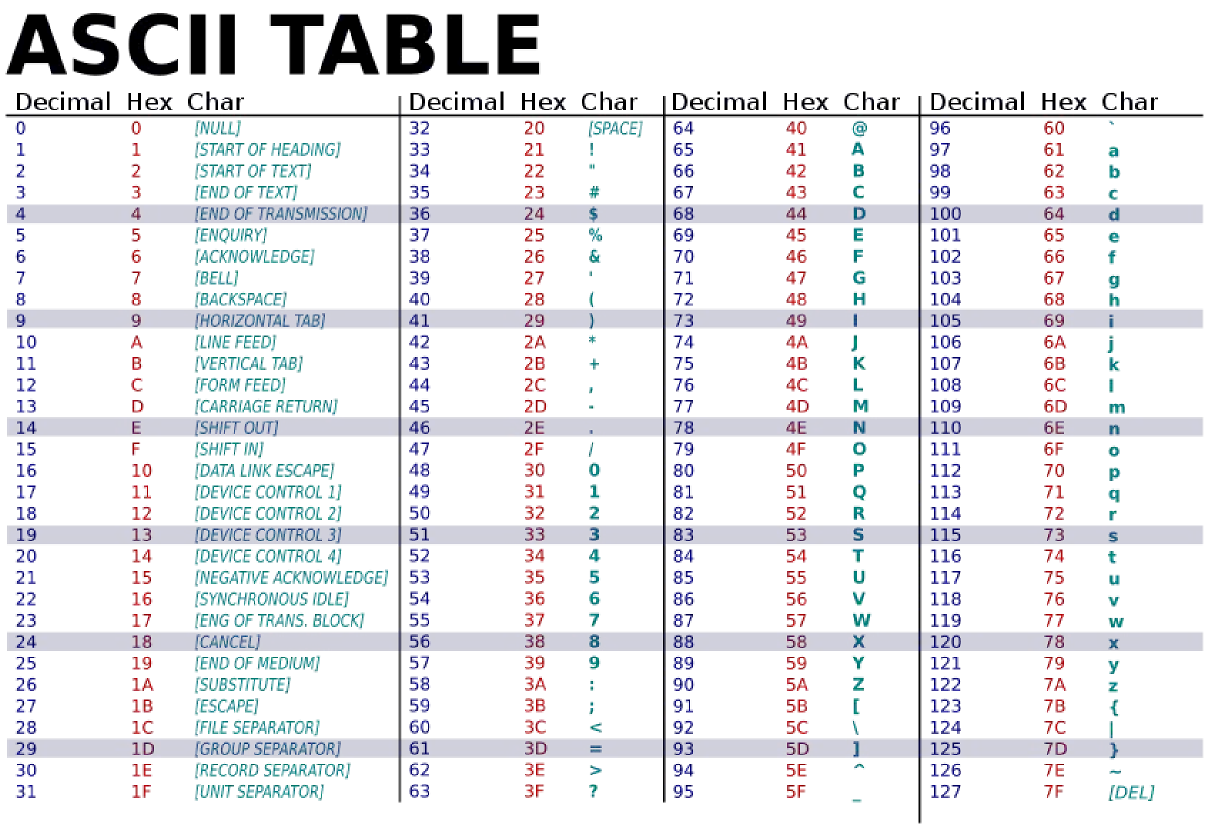
6.4) find string is anagram or not



```python
s1=input("enter a string one: ")
s2=input("enter a string two: ")
if len(s1)!=len(s2):
    print("Not anagram ")
else:
    for x in s1:
        if x not in s2:
            print("not Anagram")
            break;
    else:
        print("Anagram")

# output
# enter a string one: chandravarma
# enter a string two: chandravarma
# Anagram
```

Formatted Printing

1) ASCII Code vs Unicode

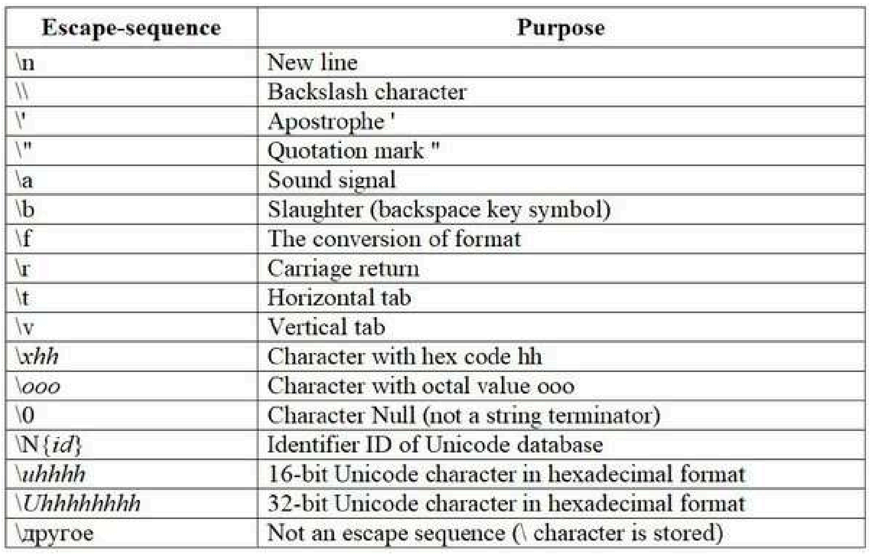


ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

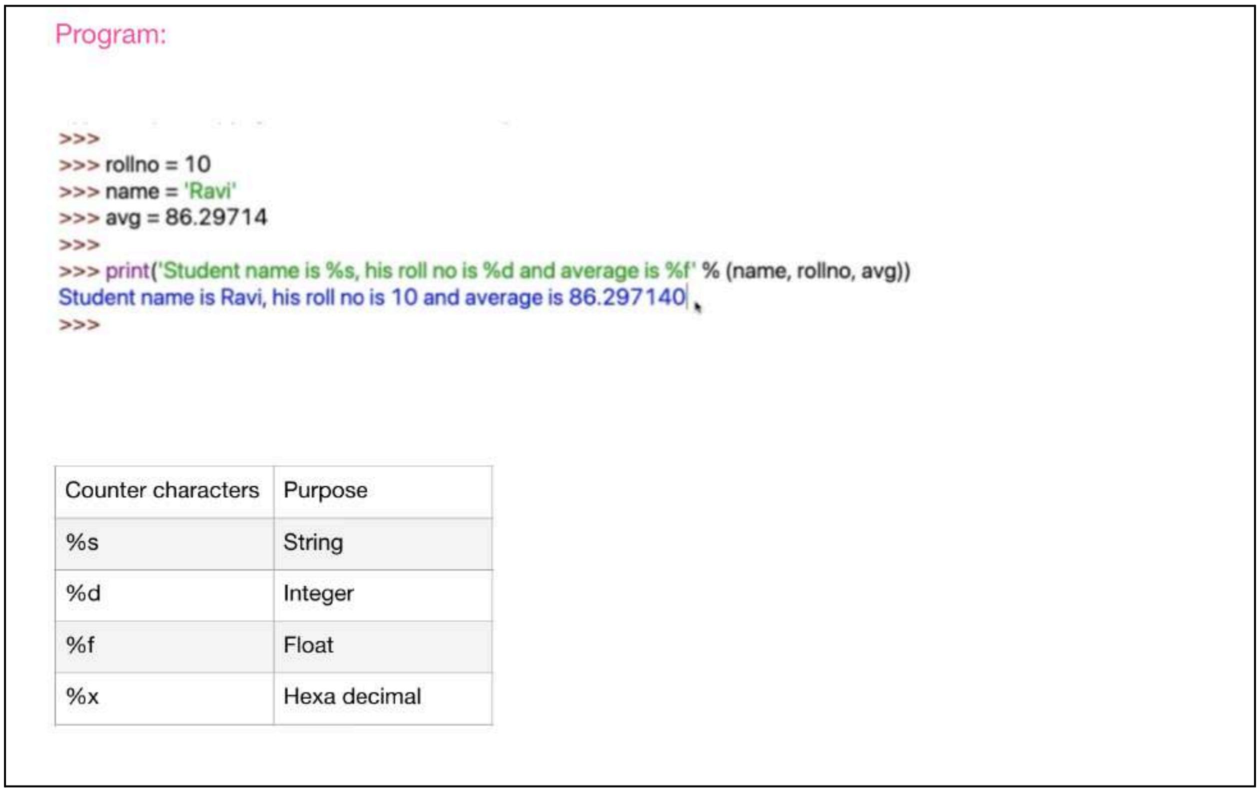
2) print Functions: use below sequence in print function

2.1 Escape Sequence



Escape-sequence	Purpose
\n	New line
\\	Backslash character
\'	Apostrophe '
\"	Quotation mark "
\a	Sound signal
\b	Slaughter (backspace key symbol)
\f	The conversion of format
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\x*hh*	Character with hex code hh
ooo	Character with octal value ooo
\0	Character Null (not a string terminator)
\N{*id*}	Identifier ID of Unicode database
\u*hhhh*	16-bit Unicode character in hexadecimal format
\U*hhhhhhhh*	32-bit Unicode character in hexadecimal format
\другое	Not an escape sequence (\ character is stored)

3) C-Style Printing



Program:

```
>>>
>>> rollno = 10
>>> name = 'Ravi'
>>> avg = 86.29714
>>>
>>> print('Student name is %s, his roll no is %d and average is %f' % (name, rollno, avg))
Student name is Ravi, his roll no is 10 and average is 86.297140
>>>
```

Counter characters	Purpose
%s	String
%d	Integer
%f	Float
%x	Hexa decimal