① Explain the math operators in python from highest to lowest presedence with an example for each.

>

operator	Description	Example
**	Exponentiation (Raise to the power)	2**3 = 8
~ + -	Complement unary plus and minus	
* / % //	Multipy, divide, modulo and floor division	3*5 = 15 21/7 = 3 22%8 = 6 21//8 = 2
+ -	Addition and Subtraction	8+8 = 16 8-2 = 4
>> <<	Right and left bitwise Shift	
&	Bitwise And	
^ |	Bitwise Exclusive 'OR' and regular 'OR'	
<= <> >=	Comparison operators	
<> == !=	Equality operators	
= %= /= //= -= += *= **=	Assignment operators	

(2) what are Comparison and boolean operators, list all in python. Explain use of these operators with Examples.

> Comparison operators also called as relational operators, Compare two values and Evaluate down to a single Boolean value.

operator	Name		Example	output
==	Equal	x=5 . y=3	print(x==y)	false
!=	Not Equal		print(x!=y)	true
>	Greater than		print(x>y)	true
<	lesser than		print(x<y)	false
>=	Greater than or Equal to		print(x>=y)	true
<=	lesser than or Equal to		print(x<=y)	false

Boolean operators

The three Boolean operators (and, or and not) are used to compare Boolean values. Like comparison operators, they Evaluate these Expressions down to a Boolean value.

Binary Boolean operators

The 'and' & 'or' operators always take two Boolean values. So they are considered binary operators. The 'and' operator Evaluates an Expression to True if both Boolean values are True; otherwise, it evaluates to false

```
>>> True and true
    True
```

On the other hand, the 'or' operator Evaluates an Expression to True if Either of the two Boolean values is true. If Both are false, it Evaluates to false >>> false or true True.

unlike 'and', 'or' the not operator operates on only one Boolean value. This makes it a unary operator. The not operator simply evaluates to the opposite Boolean value.

```
>>> ~~T~~ not True
    False
>>> not not true
    True
```

(5) Explain elif, for, while, break and continue in python with Example.

> The elif statement is an "else if" statement that always follows an if or another elif statement.

It provides another condition that is checked only if all of the previous conditions were false.

In code, an elif statement always contains :-

- The ~~elif~~ elif keyword
- The condition
- A colon
- starting on the next line, an indented block of code.

Eg!:
```
if name == 'Alice':
        print('Hi', Alice')

    elif age<12:
        print('you are not Alice, kiddo')
```

while!:

You can make a block of code execute over and over again using a while statement. The code in a while clause will be executed as long as the while statement's condition is True.

In code, a while statement always consists of the following

- The while keyword
- A condition
- A colon
- Starting on the next line, an indented block of code.

Eg:
```
spam =0
while spam <5 :
    print ('Hello world')
    Spam =Spam +1
```

Break:-

if the execution reaches a break statement, it immidiately exits the while loop's clause.

Eg:
```
while True:
    print ('Please type your name:')
    name = input( )
    if name == 'your name':
        break
    print ('Thank you')
```

Continue:-

when the program execution reaches a continue statement, the program execution immediately jumps back to the start of the loop and re evaluates the loop's condition.

Eg:
```
while true:
    print ('who are you')
    name = input()
```

Scanned with CamScanner

```
Eg  name != 'Joe':
        Continue
    Print ('Hello, Joe. Password?')
     Password = input()
     if password == 'Swordfish':
          break
     print ('Access granted')
```

For Loop:-

If you want to execute a block of code only a certain number of times, you can do this with a for loop statement and range() function.

for statement looks something like for i in range(5):
and includes the following:

- The for keyword.
- A variable name
- The in keyword
- A Call to the range() method with up to three integers passed to it
- A Colon
- Starting on the next line, an indented block of code.

```
Eg:- for i in range(5):
        print('Jimmy Five times (' + str(i) + ')')
```

④ what is Exception handling , write a python program to handle zero division exception.

> Errors can be handled with try and except statements. The code that could potentially have an error is put in a try clause.
The program execution moves to the start of a following except clause if an error happens.

```
def spam (divide By):
    try:
        return 42/ divideBy
    except Zero DivisionError:
        print(' Error: Invalid argument')

    print (spam(2))
    Print (spam (4))
    print (spam (0))
    print (spam (1))
```

~~[illegible]~~ o/p: 21.0
```
11.5
Error: Invalid argument
42.0
```

⑤ Create a function to print a blank of tic tac toe board.

> tictac = {'low-L':' ','low-M':' ','low-R':' ',
'mid-L':' ','mid-M':' ','mid-R':' ',
'top-L':' ','top-M':' ','top-R':' '}

```
def print_board():
    print (tictac ['top-L'] + '|' + tictac['top-M'] + '|'+
                 tictac [ 'top-R' ]).
    print ('-+-+-')

    print (tictac ['mid-L'] + '|' + tictac ['mid-M'] + '|'
             + tictac ['mid-R'])
    print ('-+-+-')
    print (tictac ['low-L'] + '|' + tictac[ 'low-M' ]+ '|'
             + tictac['low-R'])

print_board()
```

Output:

```
 |  |
-+--+-
 |  |

-+ - +-
 |  |
```

(6) with Example Explain join(), split() and string() in Python.

> The join() method is useful when you have a list of strings that need to be joined together into a single string value.

The join() method is called on a string, gets passed a list of strings, and returns a string.

Eg:
```
>>> .join (['My', 'name', 'is', 'simon'])
'My name is Simon'.
```

Scanned with CamScanner

The split () method does the opposite: it's called on a string value and returns a list of strings.

```
>>> 'my name is Simon'. split()
    ['my', 'name', 'is', 'simon']
```

(7) what is dictionary, how is it different from list. write a python program to count occurances of characters in string and print the count.

> The Dictionary Data type like a list, a dictionary is a mutable collection of many values.
Indexes for dictionaries can use many different data type, not just integers. Indexes for dictionaries are called keys, and a key with its associated value is called a key - value pair.
List is an ordered sequence of objects, whereas dictionaries are unordered sets. List can be accessed by positions.

To print the number of occurances of characters in a string

```
my_string = 'Hello, how are you? buddy'
my_char = 'o'
print (my_string. count (my_char))
```

O/P: 3

(8) Write a program to concatenate and compare a string. Read the strings from user.

7

Concatenate:-

```
print ('Enter your name:')
 name = input()
print (' Enter your age :')
  ~~age = input()~~  age = int(input())
  ~~age = str (intp~~
    print ('you are'+ name +'and your age is'+ age)
```

Output:-

```
Enter your name: Bob
Enter your age: 28
You are Bob and your age is 28
```

Compare:-

```
print (' Enter first string:)
 str_1 = input()
print (' Enter Second String:' )
 str_2 = input()

 If str_1 == str_2 :
  print ('strings are Equal')
 Else:
  print (' strings are not Equal')
```

output:-

```
~~Enter string~~
Enter first string: John
Enter second String : John
```

'Strings are Equal'

output 2:

```
Enter ~~String~~ first String: John
Enter second String: Alice
'Strings are not Equal'
```