1. Two values of Boolean are ‘True ’ and ‘False’. To write two Boolean values , you can assign them to variables or directly use them

   Using boolean value as an Assign :- val = True
   val1 =False
   print(val) # output :- True
   print(val1) # output :- False

   using Boolean value as an expression :- x = True and False
   y = not True
   print(x) # output :False

2. The three different types of boolean operators in Python are Logical AND(**and**), Logical OR(**or**) , Logical Not(**not**)

3. Truth table for each Boolean operator are as follows

   Logical AND(‘and’)

Operand 1	**Operand 2**	**Result**
False	False	False
False	True	False
True	False	False
True	true	True

Logical OR(‘or’)

Operand 1	**Operand 2**	**Result**
False	False	False
False	True	True
True	False	True
True	true	True

Logical NOT(‘not’)

Operand	**Result**
False	True
True	False

4. a) (5 > 4) and (3 == 5) :- It gives **'False'**
   b) not (5 > 4) :- It gives **False**
   c) (5 > 4) or (3 == 5) :- It gives **True**
   d) not ((5> 4) or (3 == 5)) :- It gives **False**
   e) (True and True) and (True == False) :- It gives **False**
   f) (not False) or (not True) :- It gives **True**

5. Six comparison operators are as following :-
   - Greater than ('**>**')
   - Less than (**'<**')
   - Equal to ('**==**')
   - Not equal to ('**!=**')
   - Greater than or equal to **('>=')**
   - Less than or equal to (**'<=**')

6. The equal-to operator ("**==**"), is used to compare two values for equality. It checks if the values on both sides of the operator are equal. This operator returns a Boolean value, either True or False, based on the result of the comparison while the assignment operator (**"="**), is used to assign a value to a variable. It assigns the value on the right side of the operator to the variable on the left side.

```
x = 5
y = 8
if x == y:
   print("x is equal to y")
else:
   print("x is not equal to y")
```

In this example, **"x=5"** we are using assignment operator, value **'5'** is assigned to variable **'x'** the condition **"x == y"** checks if the value of x is equal to the value of y. If they are equal, it prints "x is equal to y"

7. **Block 1:**

```
spam = 0
 if spam == 10:
   print('eggs')
```

Block 2:

```
if spam > 5:
   print('bacon')
else:
   print('ham')
```

Block 3:

```
print('spam)
      print('spam')
```

8. **PROGRAM 1** (‘**spam**’ value can be changed, if we change value to **‘2’** , it will print “**howdy**”)

```
spam = 1

if spam == 1:

  print("Hello")

elif spam == 2:

   print("Howdy")

else:

 print("Greetings!")
```

PROGRAM 2 (taking input from user for **‘spam’,** based on input from user it will produce desired output)

```
spam= str(input("Enter value of spam:  "))
if(spam== '1'):
   print("hello")
elif(spam=='2'):
   print("Howdy")
else:
   print("Greetings")
```

9. If your program is stuck in an endless loop in Python, you stop the execution by pressing ‘**Ctrl + C’** on your keyboard. It sends an interrupt signal (SIGINT) to the running program, causing it to terminate

10. When continue is executed, it causes the loop to skip the rest of the current iteration and move on to the next iteration. For example consider the following code :

```
for i in range(1, 6):
   if num == 3:
      continue
   print(i)
```

In this case, when i is 3, the continue statement is triggered, and the print(i) statement is skipped. The loop then continues with the next iteration, printing the remaining numbers
So output is :
1
2
4
5

When break is executed, it immediately terminates the loop, regardless of the remaining iterations.

```
for num in range(1, 6):
  if num == 4:
    break
  print(num)
```

In this case, when num is 4, the break statement is triggered, and the loop is exited. Therefore, only the numbers from 1 to 3 are printed.

So in conclusion the '**continue'** statement is used to skip the rest of the current iteration and move to the next iteration of the loop. On the other hand, the **'break'** statement is used to immediately terminate the loop, exiting the loop altogether.

11. **range(10):**

    This specifies a range that **starts from 0** (default start value) and goes up to, but not including, the specified **end value, ie. 10**
    The **step value is assumed to be 1** (default step value).
    This will generate a sequence of numbers from 0 to 9.

    range(0, 10):

    This **explicitly specifies the start value as** 0 and the **end value as 10 (not included).**
    The step value is implicitly assumed to be 1 (default step value).
    This will generate a sequence of numbers from 0 to 9, just like range(10).

    range(0, 10,1):

    This explicitly specifies the start value as 0, the end value as 10 (not included), and the step value is 1

    The step value determines the increment between each number
    Since the step value is 1, this will generate a sequence of numbers from 0 to 9, similar to the previous two cases.

    The difference between the range(10), range(0,10) and range(0,10,1) lies in the explicitness of the start value and step value in the latter two cases, while range(10) uses the default start value of 0 and step value of 1.

12. Printing numbers 1 to 10 using **for loop** :

    ```
    for i in range (1,11):
        print(i)
    ```

    Printing numbers 1 to 10 using **while loop** :

```
i=1
while(i<11)
print(i)
i=i+1
```

13. Firstly we have to import module named **spam** , then we can call function named **“bacon()”,** using dot(“**.**”) notation as shown below

```
Import spam

Spam.bacon()
```