1.What are the two values of the Boolean data type? How do you write them?

Ans:- The Boolean data type has two values: True and False. In Python, you write them exactly like that, with the first letter capitalized. Here's an example:
is_active = True

is_inactive = False

2. What are the three different types of Boolean operators?
Ans: - Boolean operators are AND, OR, and NOT. AND returns True if both operands are true. OR returns True if at least one operand is true. NOT inverts the truth value.

3. Make a list of each Boolean operator's truth tables (i.e. every possible combination of Boolean values for the operator and what it evaluates).
Ans:-
AND Operator (&&):-

A	**B**	**A && B**
T	T	T
T	F	F
F	T	F
F	F	F

OR Operator (||):-

A	**B**	**A || B**
T	T	T
T	F	T
F	T	T

A	B	A || B
F	F	F

NOT Operator (!)

A	!A
T	F
F	T

In these tables, T stands for true and F stands for false. The third column represents the result of the operation. For example, in the AND operator table, A && B is true only when both A and B are true. Otherwise, it's false. Similarly, for the OR operator, A || B is true if either A or B is true. For the NOT operator, !A is the opposite of A.

4. What are the values of the following expressions?

(5 > 4) and (3 == 5)

not (5 > 4)

(5 > 4) or (3 == 5)

not ((5 > 4) or (3 == 5))

(True and True) and (True == False)

(not False) or (not True)

Ans:-

(5 > 4) and (3 == 5) evaluates to False because while 5 > 4 is True, 3 == 5 is False, and True and False results in False.

1. not (5 > 4) evaluates to False because 5 > 4 is True, and not True is False.

2. (5 > 4) or (3 == 5) evaluates to True because 5 > 4 is True, and for an OR operation, if any one of the operands is True, the result is True.
3. not ((5 > 4) or (3 == 5)) evaluates to False. The inner expression (5 > 4) or (3 == 5) is True, so not True gives us False.
4. (True and True) and (True == False) evaluates to False. The expression (True and True) is True, but (True == False) is False. So, we have True and False, which results in False.
5. (not False) or (not True) evaluates to True. The expression (not False) is True, and (not True) is False. So, we have True or False, which results in True.

5. What are the six comparison operators?

Ans:-
The six comparison operators in many programming languages are:

1. == (Equal to): Checks if the values of two operands are equal or not; if yes, then the condition becomes true.
2. != (Not equal to): Checks if the values of two operands are equal or not; if the values are not equal, then the condition becomes true.
3. > (Greater than): Checks if the value of the left operand is greater than the value of the right operand; if yes, then the condition becomes true.
4. < (Less than): Checks if the value of the left operand is less than the value of the right operand; if yes, then the condition becomes true.
5. >= (Greater than or equal to): Checks if the value of the left operand is greater than or equal to the value of the right operand; if yes, then the condition becomes true.
6. <= (Less than or equal to): Checks if the value of the left operand is less than or equal to the value of the right operand; if yes, then the condition becomes true.

These operators are used in conditional statements for comparing values and making decisions based on those comparisons. They return a Boolean result after the comparison: either `True` or `False`.

6. How do you tell the difference between the equal to and assignment operators?Describe a condition and when you would use one.

Ans:- The equal to and assignment operators are used for different purposes in programming:

The equal to operator (==) is a comparison operator. It's used to compare two values for equality. If the values are equal, the operator returns True; otherwise, it returns False. For example, in the expression if (a == b), we're checking if a is equal to b.

The assignment operator (=) is used to assign a value to a variable. For instance, in the statement a = 5, we're assigning the value 5 to the variable a.

7. Identify the three blocks in this code:

```
spam = 0

if spam == 10:

print('eggs')

if spam > 5:

print('bacon')

else:

print('ham')

print('spam')

print('spam')
```

Ans:-

```
# Block 1

spam = 0

# Block 2

if spam == 10:

   print('eggs')

# Block 3

if spam > 5:

   print('bacon')

else:

   print('ham')

   print('spam')

   print('spam')
```

8. Write code that prints Hello if 1 is stored in spam, prints Howdy if 2 is stored in spam, and prints Greetings! if anything else is stored in spam.

Ans:-

```
if spam==1:
        print('Hello')
    if spam==2:
        print('Howdy')
    else:
        print('Greetings!')
```

9.If your programme is stuck in an endless loop, what keys you'll press?

Ans:- Ctrl + C

10. How can you tell the difference between break and continue?

Ans:- break terminates the loop, while continue skips to the next iteration. Both are used to control the flow of loops.

11. In a for loop, what is the difference between range(10), range(0, 10), and range(0, 10, 1)?

Ans:- In Python, `range(10)`, `range(0, 10)`, and `range(0, 10, 1)` are all equivalent. They all generate a sequence of numbers from 0 to 9 (inclusive of 0, exclusive of 10). Here's what each argument means:

- `range(stop)`: Generates numbers from 0 up to but not including `stop`.
- `range(start, stop)`: Generates numbers from `start` up to but not including `stop`.
- `range(start, stop, step)`: Generates numbers from `start` up to but not including `stop`, incrementing by `step`.

So in all three cases, you get the sequence [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]. The difference is just in how you specify the arguments. If you don't provide a start or step, they default to 0 and 1 respectively.

12. Write a short program that prints the numbers 1 to 10 using a for loop. Then write an equivalent program that prints the numbers 1 to 10 using a while loop.

Ans:
(a) for i in range(1,10):

```
    Print(i)
```

(b)
```
i = 1
while i <= 10:
    print(i)
    i += 1
```

13. If you had a function named bacon() inside a module named spam, how would you call it after importing spam?

Ans:- import spam

```
    spam.bacon()
```