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

The two values of the Boolean data type are "True" and "False". In Python, they are written with the capital letter "T" and "F", respectively.

2. What are the three different types of Boolean operators?

AND operator: The AND operator returns "True" if both operands are "True", and "False" otherwise. In Python, the AND operator is denoted by the symbol "and".

OR operator: The OR operator returns "True" if at least one operand is "True", and "False" otherwise. In Python, the OR operator is denoted by the symbol "or".

NOT operator: The NOT operator negates the value of its operand. If the operand is "True", then NOT returns "False", and vice versa. In Python, the NOT operator is denoted by the keyword "not".

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).

Here are the truth tables for each Boolean operator:

1. AND operator:

|  |  |  |
| --- | --- | --- |
| **Operand 1** | **Operand 2** | **Result** |
| True | True | True |
| True | False | False |
| False | True | False |
| False | False | False |

1. OR operator:

|  |  |  |
| --- | --- | --- |
| **Operand 1** | **Operand 2** | **Result** |
| True | True | True |
| True | False | True |
| False | True | True |
| False | False | False |

1. NOT operator:

|  |  |
| --- | --- |
| **Operand** | **Result** |
| True | False |
| False | True |

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)

Here are the values of the given expressions:

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

The expression **(5 > 4)** evaluates to **True**, and **(3 == 5)** evaluates too **False**.

Therefore, the expression as a whole evaluates too **False**.

not (5 > 4)

The expression **(5 > 4)** evaluates to **True**.

The **not** operator negates the value of the expression, so the overall expression evaluates too **False**.

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

The expression **(5 > 4)** evaluates to **True**, and **(3 == 5)** evaluates too **False**.

Or operator returns **True** if at least one operand is **True**, so the expression as a whole evaluates to **True**.

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

The expression **(5 > 4) or (3 == 5)** evaluates to **True** (as explained in the previous step).

The **not** operator negates the value of the expression, so the overall expression evaluates too **False**.

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

The expression **True == False** evaluates too **False**.

And operator returns **True** if both operands are **True**, so the expression as a whole evaluates too **False**.

(Not False) or (not True)

* + The expression **not False** evaluates to **True**, and **not True** evaluates too **False**.
  + Or operator returns **True** if at least one operand is **True**, so the expression as a whole evaluates to **True**.

5. What are the six comparison operators?

The six comparison operators in Python are:

**>:** Greater than. Returns **True** if the left operand is greater than the right operand, otherwise returns **False**.

**<:** Less than. Returns **True** if the left operand is less than the right operand, otherwise returns **False**.

**>=:** Greater than or equal to. Returns **True** if the left operand is greater than or equal to the right operand, otherwise returns **False**.

**<=:** Less than or equal to. Returns **True** if the left operand is less than or equal to the right operand, otherwise returns **False**.

**==:** Equal to. Returns **True** if the left operand is equal to the right operand, otherwise returns **False**.

**! =:** Not equal to. Returns **True** if the left operand is not equal to the right operand, otherwise returns **False**.

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

The equal to operator **==** is a comparison operator that is used to test whether two values or expressions are equal to each other, while the assignment operator **=** is used to assign a value to a variable.

the assignment operator **=** is used to assign a value to a variable. For example, **x = 3** assigns the value **3** to the variable **x**. The assignment operator does not compare two values, but rather sets the value of a variable to a given value.

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

if spam == 10:

print('eggs')

Block 2

if spam > 5:

print('bacon')

else:

print('ham')

Block3

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.

if spam == 1:

print('Hello')

elif spam == 2:

print('Howdy')

else:

print('Greetings!')

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

If a program is stuck in an endless loop and is not responding, you can try to interrupt the program by pressing the **Ctrl** + **C** keys on your keyboard. This will send a **Keyboard Interrupt** signal to the program, which will cause it to stop running and return control to the command prompt or terminal.

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

break and continue are two flow control statements in Python that are used to alter the behaviour of loops. The main difference between break and continue is that break is used to exit a loop, while continue is used to skip an iteration of a loop and continue to the next iteration.

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

Here are the differences between range (10), range (0, 10), and range (0, 10, 1):

* range (10): This generates a sequence of numbers from 0 up to (but not including) 10, with a step size of 1. This is equivalent to range (0, 10, 1).
* range (0, 10): This generates a sequence of numbers starting from 0 up to (but not including) 10, with a step size of 1. This is equivalent to range (10) and range (0, 10, 1).
* range (0, 10, 1): This generates a sequence of numbers starting from 0 up to (but not including) 10, with a step size of 1. This is equivalent to range (10) and range (0, 10).

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.

for i in range (1, 11):

print(i)

Output=== 1

2

3

4

5

6

7

8

9

10

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

If you have a function named **bacon ()** inside a module named **spam**, you can call it after importing **spam** by using the dot notation to reference the function.