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

The Boolean data type has two values:

True: Represents a true or affirmative condition. In Python, it is written as **True**.

False: Represents a false or negative condition. In Python, it is written as **False.**

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

1. **AND Operator (&& or "and" in Python):**
2. **OR Operator (|| or "or" in Python):**
3. **NOT Operator (! or "not" in Python):**

**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 evaluate ).**

Truth Tables for each of the three primary Boolean operators: AND, OR, and NOT.

**AND Operator Truth Table** (&& or "and" in Python)\*\*:

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

**OR Operator Truth Table (|| or "or" in Python)\*\*:**

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

**NOT Operator Truth Table (! or "not" in Python)\*\*:**

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

These truth tables show the result of applying each Boolean operator to different combinations of Boolean values (True and False). They define how each operator behaves based on the inputs and provide a reference for understanding how these operators work in logical expressions.

**4. What are the values of the following expressions?**

1. (5 > 4) and (3 == 5)

(5 > 4) is **True**

(3 == 5) is **False**

So 5 > 4) and (3 == 5) evaluates to **True** and **False**, which is **False**.

1. not (5 > 4)

(5 > 4) is **True**

‘not’ operator negates the **True** value to **False**.

not (5 > 4) is **False**.

1. (5 > 4) or (3 == 5)

(5 > 4) is **True**

(3 == 5) is **False**

When you use the ‘or’ operator, at least one condition must be True for the result to be True.

(5 > 4) or (3 == 5) is **True**

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

(5 > 4) is **True**

(3 == 5) is **False**

(5 > 4) or (3 == 5) is **True**

not ((5 > 4) or (3 == 5)) is **False**

1. (True and True) and (True == False)

(True and True) is **True**

(True == False) is **False**

(True and True) and (True == False) is **False**

1. (not False) or (not True)

(not False) is **True**

(not True) is **False**

(not False) or (not True) is **True**

**5 What are the six comparison operators?**

1. **Equal to (==):**
2. **Not equal to (!=):**
3. **Greater than (>):**
4. **Less than (<):**
5. **Greater than or equal to (>=):**
6. **Less than or equal to (<=):**

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

**Equal To (Comparison) Operator (==)**:

* The equal to operator is used to compare two values or expressions to determine if they are equal.
* It returns **True** if the values are equal and **False** if they are not.
* It is primarily used in conditional statements and expressions to make decisions based on equality comparisons.

**Example:**

x = 5

y = 10

if x == y:

print("x is equal to y")

else:

print("x is not equal to y")

**Assignment Operator (=)**:

* The assignment operator is used to assign a value to a variable. It does not compare values for equality.
* It takes the value on its right-hand side and assigns it to the variable on its left-hand side.

**Example:**

x = 5 # Assign the value 5 to the variable x

**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')

**:-**

**1. Block 1:**

spam = 0

This block contains a single statement that assigns the value 0 to the variable `spam`. It is not indented, so it's not part of any conditional block.

**2. Block 2:**

if spam == 10:

print('eggs')

This block starts with the `if` statement and contains two indented lines. It is the first conditional block. If the condition `spam == 10` is true, it will execute the `print('eggs')` statement.

**3. Block 3:**

if spam > 5:

print('bacon')

else:

print('ham')

This block is the second conditional block. It begins with an `if` statement and contains two indented lines. If the condition `spam > 5` is true, it will execute the `print('bacon')` statement; otherwise, it will execute the `print('ham')` statement.

**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.**

spam = int(input("Enter a value for 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?**

Pressing this key combination (Ctrl and C keys simultaneously) will typically interrupt the execution of a running program in most command-line interfaces or terminals. It sends a "Keyboard Interrupt" signal to the program, causing it to terminate.

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

The **break** statement is used to exit or terminate the current loop prematurely. It is often used when a specific condition is met, and you want to exit the loop early without completing all iterations.

The **continue** statement is used to skip the rest of the current iteration of a loop and move on to the next iteration. When **continue** is encountered within a loop, the current iteration is aborted, and program control goes back to the beginning of the loop for the next iteration.

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

**range(10)**:

* This expression creates a range object that starts at 0 by default (if no starting value is specified) and goes up to (but doesn't include) the specified value, which is 10 in this case.
* It effectively generates the numbers 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9.

**range(0, 10):**

* This expression explicitly specifies both the start and end values for the range.
* it generates the same sequence of numbers as **range(10)**—from 0 to 9.

**range(0, 10, 1)**:

* This expression is similar to **range(0, 10)** but includes an optional third argument, which is the step value. In this case, the step value is 1, which means that it increments the sequence by 1 for each iteration.
* It also generates the numbers from 0 to 9, with a step of 1, so it's equivalent to the previous two expressions.

**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.**

**# Using a for loop**

for i in range(1, 11):

print(i)

**# Using a while loop**

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?**

import spam **# Import the spam module**

**# Call the bacon() function from the spam module**

spam.bacon()