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

**Solution 1.**

The Boolean data type represents a binary value that can be either true or false. In most programming languages, including Python, the two values of the Boolean data type are typically written as True and False. Note that the capitalization is important, as these values are treated as reserved keywords in Python.Here's an example of using Boolean values in Python:

is\_raining = True

is\_sunny = False

if is\_raining:

print("It is raining.")

else:

print("It is not raining.")

if is\_sunny:

print("It is sunny.")

else:

print("It is not sunny.")

#Output

It is raining.

It is not sunny.

It's important to note that Boolean values are not limited to just True and False. In some programming languages, other representations such as 1 for true and 0 for false may be used. However, in Python, the standard Boolean values are True and False.

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

**Solution 2.**

The three different types of Boolean operators are:

1. **AND operator (&& or and):**

The AND operator returns true if both operands are true; otherwise, it returns false. It evaluates to true only if all the conditions it connects are true.

2. **OR operator (|| or or):**

The OR operator returns true if at least one of the operands is true; otherwise, it returns false. It evaluates to true if any of the conditions it connects is true.

3. **NOT operator (! or not):**

The NOT operator is a unary operator that returns the opposite of the operand's value. If the operand is true, the NOT operator returns false, and vice versa.

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 ).

**Solution 3.**

**AND Operator (&& or and):**

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

**OR Operator (|| or or):**

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

**NOT Operator (! or not):**

|  |  |
| --- | --- |
| **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)

**Solution 4.**

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

**Result: False**

Explanation: The expression (5 > 4) evaluates to True, and (3 == 5) evaluates to False. The and operator returns False if any of the operands is False, so the overall expression evaluates to False.

**not (5 > 4)**

**Result: False**

Explanation: The expression (5 > 4) evaluates to True, and the not operator negates the value, resulting in False.

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

**Result: True**

Explanation: The expression (5 > 4) evaluates to True, and (3 == 5) evaluates to False. The or operator returns True if at least one of the operands is True, so the overall expression evaluates to True.

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

**Result: False**

Explanation: The inner expression (5 > 4) or (3 == 5) evaluates to True since (5 > 4) is True. The not operator negates the value, resulting in False.

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

**Result: False**

Explanation: The expression (True and True) evaluates to True, but (True == False) evaluates to False. The and operator returns False if any of the operands is False, so the overall expression evaluates to False.

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

**Result: True**

Explanation: The expression (not False) evaluates to True, and (not True) evaluates to False. The or operator returns True if at least one of the operands is True, so the overall expression evaluates to True.

5. What are the six comparison operators?

**Solution 5.**

The six comparison operators in Python are:

* Equal to (==): Checks if the left operand is equal to the right operand.
* Not equal to (!=): Checks if the left operand is not equal to the right operand.
* Greater than (>): Checks if the left operand is greater than the right operand.
* Less than (<): Checks if the left operand is less than the right operand.
* Greater than or equal to (>=): Checks if the left operand is greater than or equal to the right operand.
* Less than or equal to (<=): Checks if the left operand is less than or equal to the right operand.

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

**Solution 6.**

**Equal to operator (==):**

The equal to operator is used to compare two values for equality. It returns True if the values are equal, and False otherwise. It is a comparison operator and does not modify the values being compared.

**Assignment operator (=):**

The assignment operator is used to assign a value to a variable. It assigns the value on the right-hand side to the variable on the left-hand side. It is an assignment operator and does not perform a comparison.

**Example:**

Let's consider a condition where we want to check if a variable x is equal to 10:

x = 10 # Assignment operator assigns the value 10 to the variable x

if x == 10: # Equal to operator compares the value of x with 10

print("x is equal to 10")

else:

print("x is not equal to 10")

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')

**Solution 7.**

The code provided has incorrect indentation, which can lead to syntax errors. If we will give the right indentation then it will execute else statement as the value of spam is neither equal to 10 nor it is greater than 5

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.

**Solution 8:**

spam = 0

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?

**Solution 9.**

**Ctrl + C** (or **Command + C** on macOS)

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

**Solution 10.**

**break statement:**

The break statement is used to terminate the current loop prematurely. When encountered, it immediately exits the loop, regardless of the loop's condition. The program continues to execute from the next statement after the loop. Break is often used to exit a loop early when a certain condition is met

**continue statement:**

The continue statement is used to skip the remaining code within a loop iteration and move on to the next iteration. When encountered, it jumps back to the beginning of the loop, skipping any code below the continue statement. The loop continues with the next iteration based on the loop's condition.

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

**Solution 11.**

**range(10):**

This form of range() generates a sequence of numbers starting from 0 and ending at (but not including) the specified value, which is 10 in this case. It increments the sequence by the default step size of 1.

**range(0, 10):**

This form of range() specifies both the start and end values for the sequence. It generates a sequence of numbers starting from the given start value (0) and ending at (but not including) the specified end value (10). The sequence increments by the default step size of 1.

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

This form of range() includes both the start value, end value, and the step size. It generates a sequence of numbers starting from the given start value (0), ending at (but not including) the specified end value (10), and incrementing by the specified step size (1). In this case, the step size is explicitly defined as 1, although it is the default value and could be omitted.

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.

**Solution 12.**

**Using for loop**

for i in range(1, 11):

print(i)

**Using 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?

**Solution 13.**

import spam

spam.bacon()