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

**Ans.** The Boolean data type represents a logical value and has two possible values: true and false. These values are used to indicate the truth or falsehood of a statement or condition in programming.

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

**Ans.** The three different types of Boolean operators are:

Logical AND (&&): The logical AND operator returns true if both of its operands are true, and false otherwise. It is denoted by the double ampersand symbol (&&). For example, in the expression A && B, both A and B must evaluate to true for the entire expression to be true.

Logical OR (||): The logical OR operator returns true if at least one of its operands is true, and false otherwise. It is denoted by the double pipe symbol (||). For example, in the expression A || B, if either A or B (or both) evaluate to true, the entire expression is true.

Logical NOT (!): The logical NOT operator is a unary operator that negates the value of its operand. It returns true if the operand is false, and false if the operand is true. It is denoted by the exclamation mark symbol (!). For example, in the expression! A, if A is true, then! A evaluates to false.

These Boolean operators are commonly used in programming to perform logical operations and make decisions based on the truth values of conditions.

**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 ).**

**Ans.**

|  |  |  |
| --- | --- | --- |
| Logical AND (&&): | |  |
| **Operand 1** | **Operand 2** | **Result** |
| TRUE | TRUE | TRUE |
| TRUE | FALSE | FALSE |
| FALSE | TRUE | FALSE |
| FALSE | FALSE | FALSE |
| Logical OR (||): | |  |
|  | |  |
| **Operand 1** | **Operand 2** | **Result** |
| TRUE | TRUE | TRUE |
| TRUE | FALSE | TRUE |
| FALSE | TRUE | TRUE |
| FALSE | FALSE | FALSE |
| Logical NOT (!): | |  |
| **Operand** | **Result** |  |
| TRUE | FALSE |  |
| FALSE | TRUE |  |
| These truth tables show the result of each Boolean operator for every possible combination of Boolean values (true and false) for its operands. | | |

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

5 > 4 is true

3 == 5 is false

true and false is false

not (5 > 4)

2. 5 > 4 is true

not true is false

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

3. 5 > 4 is true

3 == 5 is false

true or false is true

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

4. 5 > 4 is true

3 == 5 is false

true or false is true

not true is false

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

5. True and True is true

True == False is false

true and false is false

(not False) or (not True)

6. not False is true

not True is false

true or false is true

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

2. not (5 > 4) is false

3. (5 > 4) or (3 == 5) is true

4. not ((5 > 4) or (3 == 5)) is false

5. (True and True) and (True == False) is false

6. (not False) or (not True) is true

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

**Ans.** The six comparison operators are:

1. Equal to (==): This operator checks if the operands on both sides are equal and returns true if they are, and false otherwise.
2. Not equal to (!=): This operator checks if the operands on both sides are not equal and returns true if they are not, and false if they are equal.
3. Greater than (>): This operator checks if the left operand is greater than the right operand and returns true if it is, and false otherwise.
4. Less than (<): This operator checks if the left operand is less than the right operand and returns true if it is, and false otherwise.
5. Greater than or equal to (>=): This operator checks if the left operand is greater than or equal to the right operand and returns true if it is, and false otherwise.
6. Less than or equal to (<=): This operator checks if the left operand is less than or equal to the right operand and returns true if it is, and false otherwise.

These comparison operators are commonly used to compare values and expressions in programming to determine the truth or falsehood of a condition.

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

**Ans**. To differentiate between the equal to (==) operator and the assignment (=) operator, consider the following:

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

The equal to operator (==) is used for comparison and checks if the values on both sides of the operator are equal. It returns true if the values are equal, and false if they are not. This operator is commonly used in conditional statements and expressions to compare values.

**Assignment (=) operator:**

The assignment operator (=) is used to assign a value to a variable. It assigns the value on the right-hand side of the operator to the variable on the left-hand side. It does not perform a comparison but instead sets the variable to a specific value.

Here's an example to illustrate the difference:

***x = 5 # The assignment operator assigns the value 5 to the variable x***

***if x == 5: # The equal to operator checks if the value of x is equal to 5***

***print("x is equal to 5.")***

In this example, the assignment operator (=) is used to assign the value 5 to the variable x. Later, the equal to operator (==) is used in the if statement to compare the value of x with 5. If x is equal to 5, it prints "x is equal to 5."

So, the assignment operator (=) is used when you want to assign a value to a variable, while the equal to operator (==) is used when you want to compare values in conditions or expressions.

**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')

This block initializes the variable spam to 0 and then checks if spam is equal to 10. However, since spam is not equal to 10, the indented code block under the if statement (print('eggs')) is not executed.

Block 2:

if spam > 5:

print('bacon')

else:

print('ham')

This block checks if spam is greater than 5. Since spam is 0, which is not greater than 5, the indented code block under the if statement (print('bacon')) is not executed. Instead, the code block under the else statement (print('ham')) is executed, and 'ham' is printed.

Block 3:

print('spam')

print('spam')

This block consists of two print statements that are not part of any conditional statement. They are executed regardless of the previous conditions. Therefore, 'spam' is printed twice.

So, the three blocks in the code are:

Block 1:

* Contains the variable assignment **and** an **if** statement.

Block 2:

* Contains an **if-else** statement.

Block 3:

* Consists of print statements executed unconditionally.

**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**. spam = input("Enter a value 1 or 2: ")

if spam == "1":

print("Hello")

elif spam == "2":

print("Howdy")

else:

print("Greeting!")

In this code, the value stored in spam is checked using if-elif-else statements. If the value is equal to

1, it prints "Hello". If the value is equal to 2, it prints "Howdy". For any other value, the else block is executed, and it prints "Greetings!".

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

**Ans**. If a program is stuck in an endless loop and you need to stop its execution, you can typically press the following key combinations depending on the operating system:

Press Ctrl + C: This sends an interrupt signal to the running program, causing it to terminate.

These key combinations provide a way to forcefully stop the execution of a program that is stuck in an infinite loop or not responding. However, it's important to note that abruptly stopping a program using these key combinations may cause unexpected behaviour, so it's always preferable to write programs that do not enter infinite loops or have proper exit conditions.

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

**Ans**. Both "break" and "continue" are control flow statements used in loops, but they serve different purposes:

**Break statement:**

The "break" statement is used to exit or terminate the current loop prematurely. When encountered within a loop, the "break" statement immediately terminates the loop's execution, and the program continues with the next statement following the loop. It effectively breaks out of the loop, regardless of the loop's condition.

Here's an example of using the "break" statement within a "for" loop in Python:

for i in range(1, 6):

if i == 4:

break

print(i)

In this example, when the loop variable "i" becomes 4, the "break" statement is executed, and the loop is terminated. The output will be:

1

2

3

**Continue statement:**

The "continue" statement is used to skip the current iteration of a loop and continue with the next iteration. When encountered within a loop, the "continue" statement stops the current iteration and moves on to the next iteration, without executing the remaining code in the loop for that specific iteration.

Here's an example of using the "continue" statement within a "for" loop in Python:

for i in range(1, 6):

if i == 3:

continue

print(i)

In this example, when the loop variable "i" becomes 3, the "continue" statement is executed, and the loop skips the remaining code for that iteration and moves on to the next iteration. The output will be:

1

2

4

5

To summarize, the "break" statement is used to terminate the entire loop, while the "continue" statement is used to skip the current iteration and move on to the next iteration of the loop.

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

**Ans**. In a for loop, the differences between range (10), range (0, 10), and range (0, 10, 1) are as follows:

range (10):

This specifies a range that starts from 0 (default start value) and ends at 10 (exclusive). It increments by 1 (default step value).

In this case, the range will generate values from 0 to 9.

It is equivalent to range (0, 10, 1).

range (0, 10):

This specifies a range that starts from 0 (inclusive) and ends at 10 (exclusive). It increments by 1 (default step value).

In this case, the range will generate values from 0 to 9.

It is equivalent to range (10) and range (0, 10, 1).

range (0, 10, 1):

This specifies a range that starts from 0 (inclusive) and ends at 10 (exclusive). It increments by 1.

In this case, the range will generate values from 0 to 9.

It is equivalent to range (10) and range (0, 10).

In summary, all three forms range (10), range (0, 10), and range (0, 10, 1) generate the same sequence of numbers from 0 to 9. The start value of 0 and the step value of 1 are the default values if not explicitly specified.

**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**. # Using a for loop

***for i in range(1, 11):***

***print(i)***

And here's the code using a while loop:

# Using a while loop

***i = 1***

***while i <= 10:***

***print(i)***

***i += 1***

Both programs will output the numbers 1 to 10, inclusive. The for loop iterates over the range from 1 to 10, and in each iteration, the current value of i is printed. The while loop uses a variable i that is initially set to 1. It continues to execute the block of code within the loop as long as i is less than or equal to 10. Inside the loop, i is printed, and then it is incremented by 1. This process continues until i reaches 11, which terminates the loop.

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

**Ans**. After importing the module named "spam," you can call the function named "bacon" using the following syntax:

python

Copy code

import spam

spam.bacon()

In this example, the module "spam" is imported using the import statement. To call the function "bacon" from the "spam" module, you prefix the function name with the module name, separated by a dot (.). Hence, spam.bacon() calls the "bacon" function defined within the "spam" module.