**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**. These values represent the two possible states of truth or falsehood in logic. In programming, Boolean values are often used for logical operations and comparisons.

Example:

a = 10

b = 5

print(a > b) # Output: True

print(a < b) # Output: False

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

**Ans**: Boolean operators are used in search engines and databases to connect and define the relationship between keywords or search terms. The three main types of Boolean operators are:

1. **AND:** This operator narrows down search results by requiring that both terms or conditions are present in the results. It is used to combine search terms to find results that include all of the specified terms. For example, "cats AND dogs" will return results that contain both the word "cats" and the word "dogs".
2. **OR:** This operator broadens search results by allowing either one or both of the terms or conditions to be present in the results. It is used to expand search results to include any of the specified terms. For example, "cats OR dogs" will return results that contain either the word "cats", the word "dogs", or both.
3. **NOT:** This operator excludes specific terms or conditions from the search results. It is used to narrow down search results by excluding certain terms or conditions. For example, "cats NOT dogs" will return results that contain the word "cats" but exclude any results that also contain the word "dogs".

**3. Make a list of each Boolean operators truth tables (i.e. every possible combination of Boolean values for the operator and what it evaluate).**

**Ans:**

1. Ans: AND Operator Truth Table:

|  |  |  |
| --- | --- | --- |
| A | B | A AND B |
| False | False | False |
| False | True | False |
| True | False | False |
| True | True | True |

2. OR Operator Truth Table:

|  |  |  |
| --- | --- | --- |
| A | B | A OR B |
| False | False | False |
| False | True | True |
| True | False | True |
| True | True | True |

1. NOT Operator Truth Table:

|  |  |
| --- | --- |
| A | NOT A |
| False | True |
| True | False |

These truth tables show the result of each Boolean operator for every possible combination of input values (True or False).

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

**Ans:**

1. (5 > 4) and (3 == 5)
   * (5 > 4) evaluates to True
   * (3 == 5) evaluates to False
   * True and False evaluates to False
2. not (5 > 4)
   * (5 > 4) evaluates to True
   * not True evaluates to False
3. (5 > 4) or (3 == 5)
   * (5 > 4) evaluates to True
   * (3 == 5) evaluates to False
   * True or False evaluates to True
4. not ((5 > 4) or (3 == 5))
   * (5 > 4) evaluates to True
   * (3 == 5) evaluates to False
   * True or False evaluates to True
   * not True evaluates to False
5. (True and True) and (True == False)
   * True and True evaluates to True
   * True == False evaluates to False
   * True and False evaluates to False
6. (not False) or (not True)
   * not False evaluates to True
   * not True evaluates to False
   * True or False evaluates to True

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

**Ans:** The six comparison operators in Python are:

1. Equal to (==): Checks if two values are equal.
2. Not equal to (!=): Checks if two values are not equal.
3. Greater than (>): Checks if the left operand is greater than the right operand.
4. Less than (<): Checks if the left operand is less than the right operand.
5. Greater than or equal to (>=): Checks if the left operand is greater than or equal to the right operand.
6. 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.**

**Ans:** In Python, the equal to operator (==) is used for comparison, to check if two values are equal. On the other hand, the assignment operator (=) is used to assign a value to a variable.

Here's how you can differentiate between them:

* Equal to operator (==): This operator is used in conditional statements or expressions to compare two values and return True if they are equal, and False otherwise.
* For example:

x = 5

y = 10

if x == y: print("x is equal to y")

else: print("x is not equal to y")

* Assignment operator (=): This operator is used to assign a value to a variable. It takes the value on the right and assigns it to the variable on the left.
* For example:

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

One condition where you would use the equal to operator (==) is when you need to compare twovalues to make a decision in your program. For example, checking if a user input matches a predefined value, or if a calculated result matches an expected outcome.

On the other hand, you would use the assignment operator (=) when you need to store a value in a variable or update its value. This is fundamental for manipulating data and keeping track of different values within your program.

**7. Identify the three blocks in this code:**

**Ans:** So, the three blocks in the code are:

1. The block that sets the value of **spam** to **0**.
2. The block associated with the first **if** statement, although it doesn't execute because the condition is not met (**spam == 10**).
3. The block associated with the second **if** statement and the subsequent **else** clause. This block prints either 'bacon' or 'ham' based on the value of **spam**.

Top of Form

**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 = int(input("Enter the value of spam: ")) # Assuming the value of spam is taken as input

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?**

**ANS:** If a program is stuck in an endless loop and you need to interrupt its execution, you can typically use the following key combination based on the environment:

1. **Ctrl + C**: This key combination is commonly used across various operating systems and environments to interrupt the execution of a program. Pressing Ctrl + C sends a SIGINT signal to the running program, causing it to terminate.

In most cases, pressing Ctrl + C will halt the execution of the program stuck in the endless loop. However, there might be cases where this doesn't work due to system or program-specific reasons. In such cases, you might need to use other methods specific to the environment or operating system.

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

**ANS:**

1. **break statement**:
   * When encountered within a loop (for loop or while loop), the **break** statement immediately terminates the loop's execution, regardless of whether the loop's condition has been met or not.
   * After **break** is executed, the program continues to execute the code immediately after the loop.
   * **break** is typically used to exit a loop prematurely based on certain conditions.

Example:

for i in range(5): if i == 3: break print(i)

Output:

0 1 2

1. **continue statement**:
   * When encountered within a loop, the **continue** statement skips the remaining code inside the loop for the current iteration and moves on to the next iteration of the loop.
   * It does not terminate the loop entirely; rather, it continues with the next iteration.
   * **continue** is typically used to skip certain iterations of a loop based on certain conditions, without terminating the loop.

Example:

for i in range(5): if i == 2: continue print(i)

Output:

0 1 3 4   
 In summary, **break** is used to exit the loop entirely, while **continue** is used to skip the current iteration and proceed with 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 Python, **range()** is a built-in function used to generate a sequence of numbers. Let's break down the differences between **range(10)**, **range(0, 10)**, and **range(0, 10, 1)** in a for loop:

1. **range(10)**:
   * This generates a sequence of numbers from 0 up to (but not including) 10.
   * It starts from 0 by default.
   * It implicitly specifies the start value as 0 and the step value as 1.
   * This is equivalent to **range(0, 10, 1)**.

Example:

for i in range(10): print(i)

Output:

0 1 2 3 4 5 6 7 8 9

1. **range(0, 10)**:
   * This generates a sequence of numbers from 0 up to (but not including) 10.
   * It explicitly specifies the start value as 0.
   * It implicitly specifies the step value as 1.

Example:

for i in range(0, 10): print(i)

Output:

0 1 2 3 4 5 6 7 8 9

1. **range(0, 10, 1)**:
   * This generates a sequence of numbers from 0 up to (but not including) 10.
   * It explicitly specifies the start value as 0.
   * It explicitly specifies the step value as 1.

Example:

for i in range(0, 10, 1): print(i)

Output:

0 1 2 3 4 5 6 7 8 9

In summary, all three forms of **range()** in the context of a for loop produce the same sequence of numbers from 0 to 9, inclusive. The difference lies in the explicitness of specifying the start value and the step value.

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

**ANS:**

Here's a short program that prints the numbers 1 to 10 using a for loop:

# Using a for loop

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

And here's the equivalent program using a while loop:

# 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?**

**ANS:** Both programs will produce the same output, printing the numbers 1 to 10 inclusive. The for loop iterates over the range of numbers from 1 to 10, while the while loop iterates as long as the variable **i** is less than or equal to 10, incrementing **i** by 1 in each iteration.

If you have a function named **bacon()** inside a module named **spam**, and you've imported the **spam** module into your Python script or interactive session, you would call the **bacon()** function using dot notation, like this:

**import spam**

# Call the bacon() function from the spam module

**spam.bacon()**

This assumes that the **bacon()** function is a top-level function within the **spam** module. If **bacon()** is nested within another scope inside the **spam** module, you would call it accordingly, following the hierarchy of the module's structure.