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

**The Boolean data type represents a logical value that can have one of two states: true or false. These values are used to express the truth or falsity of a condition in programming.**

**In most programming languages, including Python, the two values of the Boolean data type are typically represented as:**

**True: Represents the true state or a condition that evaluates to true. It is written as the keyword True.**

**False: Represents the false state or a condition that evaluates to false. It is written as the keyword False.**

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

**Logical AND (represented as and): The logical AND operator returns True if both operands are True, and False otherwise. It evaluates to True only if all conditions are true.**

**EX: x = True**

**y = False**

**result = x and y**

**print(result)**

**Logical OR (represented as or): The logical OR operator returns True if at least one of the operands is True, and False otherwise. It evaluates to False only if all conditions are false.**

**EX: x = True**

**y = False**

**result = x or y**

**print(result)**

**Logical NOT (represented as not): The logical NOT operator negates the value of its operand. If the operand is True, the result is False, and if the operand is False, the result is True. It essentially flips the Boolean value.**

**EX: x = True**

**result = not x**

**print(result)**

**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 ).**

**Logical AND (and):**

**Operand 1 Operand 2 Result**

**False False False**

**False True False**

**True False False**

**True True True**

**Logical OR (or):**

**Operand 1 Operand 2 Result**

**False False False**

**False True True**

**True False True**

**True True True**

**Logical NOT (not):**

**Operand Result**

**False True**

**True False**

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

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

**The expression (5 > 4) evaluates to True because 5 is indeed greater than 4.**

**The expression (3 == 5) evaluates to False because 3 is not equal to 5.**

**The overall expression evaluates to False because both sub-expressions connected by and must be True for the entire expression to be True.**

**Therefore, the value of the expression is False.**

**not (5 > 4)**

**The expression (5 > 4) evaluates to True because 5 is greater than 4.**

**The not operator negates the value, so True becomes False.**

**Therefore, the value of the expression is False.**

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

**The expression (5 > 4) evaluates to True because 5 is greater than 4.**

**The expression (3 == 5) evaluates to False because 3 is not equal to 5.**

**The overall expression evaluates to True because at least one of the sub-expressions connected by or is True.**

**Therefore, the value of the expression is True.**

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

**The sub-expression (5 > 4) or (3 == 5) evaluates to True because 5 is greater than 4.**

**The not operator negates the value, so True becomes False.**

**Therefore, the value of the expression is False.**

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

**The sub-expression True and True evaluates to True because both operands are True.**

**The sub-expression True == False evaluates to False because True is not equal to False.**

**The overall expression evaluates to False because the second sub-expression is False.**

**Therefore, the value of the expression is False.**

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

**The sub-expression not False evaluates to True because not False is the same as True.**

**The sub-expression not True evaluates to False because not True is the same as False.**

**The overall expression evaluates to True because at least one of the sub-expressions**

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

**The six comparison operators are:**

**Equal to (==): Compares if two values are equal and returns True if they are equal, False otherwise.**

**Not equal to (!=): Compares if two values are not equal and returns True if they are not equal, False otherwise.**

**Greater than (>): Checks if the value on the left is greater than the value on the right and returns True if it is, False otherwise.**

**Less than (<): Checks if the value on the left is less than the value on the right and returns True if it is, False otherwise.**

**Greater than or equal to (>=): Checks if the value on the left is greater than or equal to the value on the right and returns True if it is, False otherwise.**

**Less than or equal to (<=): Checks if the value on the left is less than or equal to the value on the right and returns True if it is, False otherwise.**

**These comparison operators are used to compare values and evaluate conditions in programming. They are commonly used in conditional statements, loops, and expressions to make decisions based on the comparison results.**

**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 (==) operator:**

**The equal to operator (==) is a comparison operator used to check if two values are equal. It compares the values on both sides of the operator and returns True if they are equal, and False if they are not.**

**Example:**

**x = 5**

**y = 7**

**if x == y:**

**print("x is equal to y")**

**else:**

**print("x is not equal to y")**

**In this example, the == operator is used to compare the values of x and y. If they are equal, it will print "x is equal to y". Otherwise, it will print "x is not equal to y". The equal to operator is commonly used in conditional statements and logical operations to compare values.**

**Assignment (=) operator:**

**The assignment 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.**

**Example: x = 10**

**In this example, the = operator is used to assign the value 10 to the variable x. It does not compare the values; rather, it assigns the value to the variable.**

**When to use each:**

**You would use the equal to operator (==) when you want to compare two values for equality in a condition or expression. It is used to check if two values are the same.**

**On the other hand, you would use the assignment operator (=) when you want to assign a value to a variable, updating or initializing its 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')**

**OUTPUT:**

**Block 1:**

**if spam == 10:**

**print('eggs')**

**This block is an if statement that checks if the value of spam is equal to 10. If the condition is true, it executes the indented statement print('eggs')c**

**Block 2:**

**if spam > 5:**

**print('bacon')**

**This block is another if statement that checks if the value of spam is greater than 5. If the condition is true, it executes the indented statement print('bacon'). Note that this block is independent of the previous if statement.**

**Block 3:**

**else:**

**print('ham')**

**This block is associated with the else clause of the second if statement. It executes the indented statement print('ham') when the condition in the second if statement evaluates to false.**

**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 = 3**

**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?**

**Ctrl + C: This key combination is widely used across different operating systems and programming environments. Pressing Ctrl + C sends an interrupt signal to the running program, causing it to terminate. This combination is effective for stopping the execution of a program that is stuck in an endless loop.**

**Ctrl + Break: On some systems, such as Windows, you can use the Ctrl + Break key combination to interrupt the program's execution and break out of the endless loop.**

**Ctrl + \ (backslash): This key combination is used in certain environments to generate a "quit" signal, which can be used to forcefully terminate a program that is unresponsive or stuck in a loop.**

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

**break and continue are two control flow statements used in loops (such as for and while loops) to alter the execution of the loop. Here's how you can tell the difference between break and continue:**

**break statement:**

**The break statement is used to terminate the current loop and immediately exit the loop's block of code.**

**When a break statement is encountered within a loop, the loop is immediately terminated, and the program continues with the next statement after the loop.**

**It allows you to prematurely exit the loop based on a certain condition or event.**

**After encountering a break statement, the loop's remaining iterations are skipped, and control is transferred to the code following the loop.**

**Typically, break is used to exit a loop when a specific condition is met.**

**continue statement:**

**The continue statement is used to skip the rest of the current iteration of a loop and move on to the next iteration.**

**When a continue statement is encountered within a loop, the loop's block of code for the current iteration is immediately skipped, and the program proceeds with the next iteration of the loop.**

**It allows you to skip certain iterations of the loop based on a certain condition.**

**After encountering a continue statement, the loop does not terminate but proceeds with the next iteration, ignoring the remaining code in the current iteration.**

**Typically, continue is used to skip specific iterations or avoid executing certain parts of the loop's block under certain conditions.**

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

**In a for loop, the range() function is commonly used to generate a sequence of numbers. The range() function can be called with different arguments to specify the start, stop, and step values for the sequence. Here's the difference between range(10), range(0, 10), and range(0, 10, 1):**

**range(10):**

**This call to range() generates a sequence of numbers starting from 0 (default start value) up to, but not including, the specified stop value (10 in this case).**

**The step value defaults to 1, meaning the sequence increments by 1 with each iteration.**

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

**print(i)**

**Output:**

**0**

**1**

**2**

**3**

**4**

**5**

**6**

**7**

**8**

**9**

**range(0, 10):**

**This call to range() specifies both the start and stop values for the sequence.**

**The sequence starts from the specified start value (0 in this case) and goes up to, but not including, the specified stop value (10 in this case).**

**The step value defaults to 1, so the sequence increments by 1 with each iteration.**

**Example usage:**

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

**print(i)**

**Output:**

**0**

**1**

**2**

**3**

**4**

**5**

**6**

**7**

**8**

**9**

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

**This call to range() specifies both the start and stop values, as well as the step value for the sequence.**

**The sequence starts from the specified start value (0 in this case) and goes up to, but not including, the specified stop value (10 in this case).**

**The step value is explicitly set to 1, meaning the sequence increments by 1 with each iteration.**

**Specifying a step value of 1 does not change the behavior of the loop compared to the previous examples.**

**Example usage:**

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

**print(i)**

**Output:**

**0**

**1**

**2**

**3**

**4**

**5**

**6**

**7**

**8**

**9**

**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.**

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

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

**print(i)**

**Output:**

**1**

**2**

**3**

**4**

**5**

**6**

**7**

**8**

**9**

**10**

**An equivalent program that prints the numbers 1 to 10 using a while loop:**

**num = 1**

**while num <= 10:**

**print(num)**

**num += 1**

**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?**

**import spam**

**spam.bacon()**

**In this code, the import spam statement imports the spam module, making its functions and variables accessible in the current Python script. To call the bacon() function, you use the syntax spam.bacon(), where spam is the module name and bacon() is the function name.**

**By prefixing the function name with the module name, you explicitly specify that the function belongs to the spam module, allowing you to call it without any naming conflicts with other functions or variables in your script.**