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

**Ans:** The Boolean data type in Python has two values: True and False. These values represent the two possible states of truth or logical conditions. They are used extensively in control structures like if-statements and loops to make decisions based on conditions.

To write these Boolean values in Python, simply use their respective keywords: **True** and **False**.

Note that the capitalization matters; they must be capitalized exactly as shown.

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

**Ans:** In Python, there are three main types of Boolean operators used to perform logical operations on Boolean values or expressions. These operators are used to combine, negate, or compare conditions. The three types of Boolean operators are:

**Logical AND (and):**

The and operator returns True if both of its operands are True, and False otherwise.

**Logical OR (or):**

The or operator returns True if at least one of its operands is True, and False if both operands are False.

**Logical NOT (not):**

The not operator is used to negate a Boolean value. It returns the opposite of the Boolean value of its operand.

These Boolean operators are commonly used to build complex conditions, make decisions, and control the flow of your program based on different logical 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:** Here are the truth tables for each of the three main Boolean operators: and, or, and not. In these tables, True is represented as T and False is represented as F.

1. Logical AND (and) Truth Table:

|  |  |  |
| --- | --- | --- |
| Operand 1 | Operand 2 | Result |
| T | T | T |
| T | F | F |
| F | T | F |
| F | F | F |

1. Logical OR (or) Truth Table:

|  |  |  |
| --- | --- | --- |
| Operand 1 | Operand 2 | Result |
| T | T | T |
| T | F | T |
| F | T | T |
| F | F | F |

1. Logical NOT (not) Truth Table:

|  |  |
| --- | --- |
| Operand | Result |
| T | F |
| F | T |

These truth tables show the possible combinations of Boolean values for each operator and the resulting Boolean value when the operator is applied to those operands.

**Top of Form**

**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:** The values of the following expressions

Expressions:

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

Values:

False

Expressions:

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

Values:

True

Expressions:

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

Values:

False

Expressions:

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

Values:

False

Expressions:

(not False)or (not True)

Values:

True

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

**Ans:** In Python, there are six comparison operators used to compare values and expressions. These operators return Boolean values (True or False) based on the comparison result. Here are the six comparison operators:

1. **Equal to (==):** This operator checks if two values are equal.

Example: x == y

1. **Not equal to (!=):** This operator checks if two values are not equal.

Example: x != y

1. **Greater than (>):** This operator checks if the left operand is greater than the right operand.

Example: x > y

1. **Less than (<):** This operator checks if the left operand is less than the right operand.

Example: x < y

1. **Greater than or equal to (>=):** This operator checks if the left operand is greater than or equal to the right operand.

Example: x >= y

1. **Less than or equal to (<=):** This operator checks if the left operand is less than or equal to the right operand.

Example: x <= y

These comparison operators are used to evaluate conditions and make decisions based on the relationships between values or expressions.

**Top of Form**

**6. How do you tell the difference between the equal to and assignment operators? Describe a**

**condition and when you would use one.**

**Ans:** The equal to operator (==) and the assignment operator (=) serve different purposes in Python, and they are used in different contexts.

1. Equal to Operator (==): The equal to operator is used to compare two values to determine if they are equal. It returns True if the values are equal and False otherwise.
2. Assignment Operator (=): The assignment operator is used to assign a value to a variable. It doesn't compare values for equality; rather, it assigns the value on the right side of the operator to the variable on the left side.

Example:

a=5

b=10

if (a==b):

print(" a and b are equal")

else:

print(" a and b are not equal")

a and b are not equal

**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:** In Python, code blocks are defined by their indentation. Blocks are groups of code statements that are executed together under certain conditions. In the given code, there are three blocks: Here's a breakdown of the blocks:

1. **Block 1:** This block contains the first if statement. However, since spam is not equal to 10, the code inside this block will not be executed.
2. **Block 2:** This block contains the second if statement. If spam is greater than 5, the code inside this block will be executed. In the given code, the value of spam is 0, so this block will not be executed.
3. **Block 3:** This block contains the else part of the second if statement. Since the if condition in Block 2 is not met, the else block will be executed, and "ham" will be printed.

The lines print('spam') are not in any specific block and will be executed after all the conditions and blocks.

Remember that in Python, the indentation is crucial for determining the scope of blocks and the flow of your code.

**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:**The code is follows:

spam="How u doin"

if (spam==1):

print("Hello")

elif (spam==2):

print("Howdy")

else:

print("Greetings!")

Greetings!

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

**Ans:** If my Python program is stuck in an endless loop and I need to stop its execution, I would press Ctrl + C on my keyboard. This sends a keyboard interrupt signal to the running program, typically causing it to stop its execution. This method allows me to break out of the loop or terminate the program that is stuck. Just keep in mind that using a keyboard interrupt might not give the program a chance to perform necessary cleanup operations, so it's best to design code to avoid such situations.

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

**Ans:** In Python, both break and continue are control flow statements used within loops to alter the flow of the program. However, they serve different purposes and have distinct effects:

1. **break Statement:** The break statement is used to exit the current loop immediately, regardless of whether the loop's condition is still satisfied or not. It terminates the loop's execution entirely and continues with the next statement after the loop.

Example:

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

break

print(i)

In this example, the loop prints numbers from 0 to 2, but when i becomes 3, the break statement is encountered, causing the loop to exit.

1. **continue Statement:** The continue statement is used to skip the rest of the current iteration of the loop and proceed to the next iteration. It doesn't terminate the loop; rather, it jumps directly to the next iteration.

Example:

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

In this example, the loop prints numbers from 0 to 4, but when i is 3, the continue statement is encountered, causing the loop to skip the print statement for that iteration and move to the next iteration.

In summary, the key difference between break and continue is that break terminates the loop completely, while continue skips the current iteration and continues with the next iteration of the loop.

**Top of Form**

**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 expressions range(10), range(0, 10), and range(0, 10, 1) essentially have the same effect, but they differ in their syntax and the arguments they accept.

1. **range(10):** This form of range generates a sequence of numbers starting from 0 up to (but not including) the specified value (10 in this case). By default, range starts at 0 and has a step size of 1.
2. **range(0, 10):** This form of range explicitly specifies the start and end values of the sequence. It generates numbers from the start value (0) up to (but not including) the end value (10), with a default step size of 1.
3. **range(0, 10, 1):** This form of range specifies the start, end, and step values for the sequence. It generates numbers from the start value (0) up to (but not including) the end value (10), incrementing by the specified step value (1).

In all three cases, the loops will produce the same output, printing numbers from 0 to 9. The differences lie in how you specify the start, end, and step values when using the range() function.

**Top of Form**

**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:** Programs are given bellow

**Using for loop:**

for i in range(1,11):

print(i)

1

2

3

4

5

6

7

8

9

10

**Using while loop:**

i=1

while i<=10:

print(i)

i+=1

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?**

**Ans:** If I have a function named bacon() inside a module named spam, and you've imported the spam module, you can call the bacon() function using the following syntax:

python code

import spam spam.bacon()

Here, you're using the module name (spam) followed by a dot (.) to access and call the function bacon() that resides within the spam module. This syntax ensures that you're calling the bacon() function from the spam module after importing it.