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

The Boolean data type has two possible values: True and False.

In Python, these values are written exactly as shown: True and False. Note that the first letter of each value is capitalized, as Python is case-sensitive. These values are keywords in Python and represent the logical truth values of true and false, respectively.

Boolean values are commonly used for making logical comparisons, controlling program flow with conditional statements, and storing the results of logical operations.

For example, you can assign a Boolean value to a variable like this:

is\_raining = True

Or use them directly in conditional statements:

if is\_raining:

print("Bring an umbrella.")

else:

print("Enjoy the sunny day!")

In this example, the variable is\_raining is assigned the value True. The conditional statement checks if is\_raining is True, and based on the result, it executes the corresponding block of code.

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

The three different types of Boolean operators are:

Logical AND (and):

The logical AND operator returns True if both of its operands are True, and False otherwise. It is represented by the keyword and. For example:

x = True

y = False

result = x and y

print(result) # Output: False

Logical OR (or):

The logical OR operator returns True if at least one of its operands is True, and False otherwise. It is represented by the keyword or. For example:

x = True

y = False

result = x or y

print(result) # Output: True

Logical NOT (not):

The logical NOT operator negates the value of its operand. If the operand is True, it returns False, and if the operand is False, it returns True. It is represented by the keyword not. For example:

x = True

result = not x

print(result) # Output: False

These Boolean operators are commonly used for combining and manipulating Boolean values, making logical decisions, and controlling the flow of a program based on logical conditions.

1. **Boolean operator’s truth tables:**

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 |

These truth tables represent all possible combinations of Boolean values for each operator and their resulting evaluations.

1. **Value of following expressions**

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

not (5 > 4) --> False

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

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

(True and True) and (True == False) --> False

(not False) or (not True) --> True

1. **Six comparison operators in Python are:**

Equal to (==):

The equal to operator (==) compares if two values are equal and returns True if they are, and False otherwise. For example, 5 == 5 evaluates to True, while 5 == 6 evaluates to False.

Not equal to (!=):

The not equal to operator (!=) compares if two values are not equal and returns True if they are not equal, and False if they are equal. For example, 5 != 6 evaluates to True, while 5 != 5 evaluates to False.

Greater than (>):

The greater than operator (>) compares if the left operand is greater than the right operand and returns True if it is, and False otherwise. For example, 5 > 3 evaluates to True, while 5 > 7 evaluates to False.

Less than (<):

The less than operator (<) compares if the left operand is less than the right operand and returns True if it is, and False otherwise. For example, 5 < 7 evaluates to True, while 5 < 3 evaluates to False.

Greater than or equal to (>=):

The greater than or equal to operator (>=) compares if the left operand is greater than or equal to the right operand and returns True if it is, and False otherwise. For example, 5 >= 5 evaluates to True, while 5 >= 7 evaluates to False.

Less than or equal to (<=):

The less than or equal to operator (<=) compares if the left operand is less than or equal to the right operand and returns True if it is, and False otherwise. For example, 5 <= 7 evaluates to True, while 5 <= 3 evaluates to False.

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

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

To distinguish between the equal to (==) and assignment (=) operators, you can look at the context in which they are used. The equal to operator (==) is used for comparison, whereas the assignment operator (=) is used to assign a value to a variable.

Here's an example that demonstrates the difference:

x = 5 # Here, the assignment operator (=) is used to assign the value 5 to the variable x.

if x == 5:

print("x is equal to 5.")

here, the assignment operator (=) is used to assign the value 5 to the variable x. On the other hand, the equal to operator (==) is used within the conditional statement (if x == 5) to compare if the value of x is equal to 5. If the condition is True, it executes the indented block of code.

You would use the equal to operator (==) when you want to compare two values or variables to check if they are equal. This is commonly used in conditional statements, loops, and other situations where you need to make decisions based on equality.

On the other hand, you would use the assignment operator (=) when you want to assign a value to a variable, update the value of a variable, or initialize a variable with a specific value.

1. **Identified the three blocks**

Block 1:

if spam == 10:

print('eggs')

Block 2:

if spam > 5:

print('bacon')

Block 3:

else:

print('ham')

print('spam')

print('spam')

1. **Write code in python 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 = # Assign a value to the variable spam

if spam == 1:

print("Hello")

elif spam == 2:

print("Howdy")

else:

print("Greetings!")

(replace the comment # Assign a value to the variable spam with an actual value, such as spam = 1, spam = 2, or any other value, to see the corresponding message being printed based on the value stored in spam)

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

Ctrl + C (Control + C) on Windows or Linux

Command + . (Command + Period) on macOS

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

In Python, the break and continue statements are used to control the flow of execution within loops (such as for and while loops). Here's how you can differentiate between break and continue:

break statement:

The break statement is used to exit or terminate the innermost loop immediately when a certain condition is met.

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

After breaking out of the loop, the program will not re-enter the loop and will continue executing the code following the loop.

The break statement is often used to prematurely exit a loop when a specific condition is satisfied.

continue statement:

The continue statement is used to skip the current iteration of a loop and move to the next iteration without executing the remaining code within the loop for that iteration.

When the continue statement is encountered within a loop, it immediately jumps to the next iteration of the loop, bypassing the remaining statements for the current iteration.

After encountering the continue statement, the loop continues with the next iteration, if any, rather than executing the remaining statements for the current iteration.

The continue statement is often used to skip certain iterations based on specific conditions, allowing the loop to proceed with the next iteration.

In summary, break is used to terminate the loop entirely and continue with the next statement after the loop, while continue is used to skip the current iteration and proceed with the next iteration of the loop.

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

In a for loop, the expressions range(10), range(0, 10), and range(0, 10, 1) are equivalent and produce the same sequence of numbers from 0 to 9. However, there are slight differences in how they are defined:

range(10):

This expression specifies a range that starts from 0 (by default) and ends at the number one less than the given value (10 in this case).

The step value is not explicitly mentioned, so it defaults to 1.

The range produced by range(10) is [0, 1, 2, 3, 4, 5, 6, 7, 8, 9].

range(0, 10):

This expression explicitly specifies the start and end values for the range.

The range starts at 0 and ends at the number one less than the given end value (10 in this case).

Again, the step value is not explicitly mentioned, so it defaults to 1.

The range produced by range(0, 10) is the same as range(10): [0, 1, 2, 3, 4, 5, 6, 7, 8, 9].

range(0, 10, 1):

This expression explicitly specifies the start, end, and step values for the range.

The range starts at 0, ends at the number one less than the given end value (10 in this case), and increments by the specified step value (1 in this case).

Since the step value is 1, it produces a sequence of consecutive numbers from the start to the end.

The range produced by range(0, 10, 1) is the same as range(10): [0, 1, 2, 3, 4, 5, 6, 7, 8, 9].

1. **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 in Python that prints the numbers 1 to 10 using a for loop:

# Using a for loop

for num in range(1, 11):

print(num)

And here's an equivalent program that prints the numbers 1 to 10 using a while loop:

# Using a while loop

num = 1

while num <= 10:

print(num)

num += 1

Both programs will produce the same output, which is the numbers 1 to 10 printed on separate lines. The for loop iterates over the range from 1 to 10 (inclusive) and prints each number. The while loop initializes a variable num with 1, then repeatedly prints the value of num and increments it by 1 until it reaches 10.

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

**importing spam?**

If you have a function named bacon() inside a module named spam, and you want to call that function after importing the spam module, you would use the following syntax:

import spam

spam.bacon()

In this code, import spam imports the spam module into your Python program, making all its functions, variables, and other contents accessible using the spam namespace.

To call the bacon() function from the spam module, you would use spam.bacon(), where spam is the module name, and bacon() is the function name. This syntax specifies the module name followed by a dot (.) and then the function name.

By using spam.bacon(), you are explicitly referencing the bacon() function within the spam module, ensuring that the correct function is called.