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

Ans: The two values of the Boolean data type are:

a) True: Represents a true or positive condition. In Python, you write it as `True`.

b) False: Represents a false or negative condition. In Python, you write it as `False`.

Boolean values are fundamental in programming as they are used in conditional statements, loops, and many other control structures to make decisions and control the flow of code execution.

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

Ans: The three different types of Boolean operators are:

a) AND operator: It returns True if both operands are True, and False otherwise. In Python, it is represented by the keyword `and`.

b) OR operator: It returns True if at least one of the operands is True, and False if both operands are False. In Python, it is represented by the keyword `or`.

c) NOT operator: It returns the opposite Boolean value of the operand. If the operand is True, the NOT operator returns False, and if the operand is False, it returns True. In Python, it is represented by the keyword `not`.

These Boolean operators are used to combine or manipulate Boolean values and are frequently used in conditional statements and logical expressions in Python programming.

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: Following are the truth tables for the three Boolean operators:

a) Operator (represented by `and`):

|  |  |  |
| --- | --- | --- |
| Operand 1 | Operand 2 | Result |
| True | True | True |
| True | False | False |
| False | True | False |
| False | False | False |

b) Operator (represented by `or`):

|  |  |  |
| --- | --- | --- |
| Operand 1 | Operand 2 | Result |
| True | True | True |
| True | False | True |
| False | True | True |
| False | False | False |

c) NOT Operator (represented by `not`):

|  |  |
| --- | --- |
| Operand | Result |
| True | False |
| False | True |

These truth tables illustrate the different outcomes based on the combinations of Boolean values when using each Boolean operator.

4. What are the values of the following expressions?

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

The expression `(5 > 4) and (3 == 5)` evaluates to `False`.

Breaking it down:- `(5 > 4)` evaluates to `True` because 5 is indeed greater than 4.

- `(3 == 5)` evaluates to `False` because 3 is not equal to 5.

ii) not (5 > 4)

The expression `not (5 > 4)` evaluates to `False`.

Breaking it down:

- `(5 > 4)` evaluates to `True` because 5 is indeed greater than 4.

- Applying the `not` operator to `True` results in the opposite value, which is `False`.

Therefore, the overall result of the expression is `False`.

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

The expression `(5 > 4) or (3 == 5)` evaluates to `True`.

Breaking it down:- `(5 > 4)` evaluates to `True` because 5 is indeed greater than 4.

- `(3 == 5)` evaluates to `False` because 3 is not equal to 5.

Using the `or` operator, the expression returns `True` because for the `or` operator to yield `True`, at least one of the operands must be `True`. In this case, one of the operands is `True`, so the overall result is `True`.

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

The expression `not ((5 > 4) or (3 == 5))` evaluates to `False`.

Breaking it down:- `(5 > 4)` evaluates to `True` because 5 is indeed greater than 4.

- `(3 == 5)` evaluates to `False` because 3 is not equal to 5.

- `(5 > 4) or (3 == 5)` evaluates to `True` because at least one of the operands is `True`.

Applying the `not` operator to `True` results in the opposite value, which is `False`.

Therefore, the overall result of the expression is `False`.

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

The expression `(True and True) and (True == False)` evaluates to `False`.

Breaking it down:- `(True and True)` evaluates to `True` because both operands are `True`, and the `and` operator requires both operands to be `True` for the result to be `True`.

- `(True == False)` evaluates to `False` because `True` is not equal to `False`.

Applying the `and` operator to `True` and `False` results in `False`.

Therefore, the overall result of the expression is `False`.

vi) (not False) or (not True)

The expression `(not False) or (not True)` evaluates to `True`.

Breaking it down:

- `not False` evaluates to `True` because the `not` operator negates the Boolean value of `False`.

- `not True` evaluates to `False` because the `not` operator negates the Boolean value of `True`.

Using the `or` operator, the expression returns `True` because at least one of the operands is `True`.

Therefore, the overall result of the expression is `True`.

5. What are the six comparison operators?

Ans: The six comparison operators in Python are:

a. Equal to: `==` - Checks if two values are equal.

b. Not equal to: `!=` - Checks if two values are not equal.

c. Greater than: `>` - Checks if the left operand is greater than the right operand.

d. Less than: `<` - Checks if the left operand is less than the right operand.

e. Greater than or equal to: `>=` - Checks if the left operand is greater than or equal to the right operand.

f. 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 to compare two values and check if they are equal, whereas the assignment operator (`=`) is used to assign a value to a variable.To differentiate between the equal to and assignment operators, you can consider their purposes and usage within a statement or context.

Equal to operator (`==`):

The equal to operator is used in conditional statements or expressions to compare values. It returns `True` if the values on both sides of the operator are equal and `False` otherwise. For example:

x = 5

if x == 5:

print("x is equal to 5")

In this case, the equal to operator (`==`) compares the value of `x` to `5` and returns `True`, allowing the code within the `if` statement to execute.

Assignment operator (`=`): The assignment operator is used to assign a value to a variable. It assigns the value on the right side of the operator to the variable on the left side. For example: x = 10

In this case, the assignment operator (`=`) assigns the value `10` to the variable `x`. It does not compare the values; instead, it assigns the value. In summary, the equal to operator (`==`) is used for comparison, while the assignment operator (`=`) is used to assign values to variables.

7. Identify the three blocks in this code:

i)spam = 0

In the given code, there is only one block, which consists of a single line:

spam = 0

This line assigns the value `0` to the variable `spam`. There are no other blocks or sections of code present in this snippet.

ii)if spam == 10:

The line `if spam == 10:` is the beginning of a conditional statement block. However, in the given code snippet, there is no code provided for what should be executed if the condition `spam == 10` evaluates to `True`. A complete conditional statement typically includes an indented block of code that follows the condition. For example:

if spam == 10:

("Spam is equal to 10")

# Additional code to be executed if the condition is True

In this case, the indented block of code following the `if` statement would represent the actions or instructions to be executed if the condition `spam == 10` evaluates to `True`.

iii) print('eggs')

The line `print('eggs')` is not part of any specific block in the code provided. It is a standalone line of code that will print the string "eggs" to the console when executed. The absence of any conditional statement or loop in the given code snippet means that this line will be executed regardless of any conditions or iterations.

iv) if spam > 5:

The line `print('eggs')` is not part of any specific block in the code provided. It is a standalone line of code that will print the string "eggs" to the console when executed. The absence of any conditional statement or loop in the given code snippet means that this line will be executed regardless of any conditions or iterations.

v) print('bacon'):

The line `print('bacon')` is also a standalone line of code that will print the string "bacon" to the console when executed. It’s not part of any specific block in the code provided. It will be executed independently of any conditional statements or loops.

else:

vi) print('ham'): The line `print('ham')` is also a standalone line of code, similar to the previous example. It will print the string "ham" to the console when executed. Since it is not part of any specific block or conditional statement in the given code snippet, it will be executed independently and will print "ham" to the console whenever this line of code is encountered during program execution.

vii) print('spam'):

The line `print('spam')` is also a standalone line of code that will print the string "bacon" to the console when executed. It’s not part of any specific block in the code provided. It will be executed independently of any conditional statements or loops.

print('spam')

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: Here's the code that fulfills the requirement

spam = 1

if spam == 1:

print("Hello")

elif spa m == 2:

print("Howdy")

else:

print("Greetings!")

In this code, the value of the variable `spam` is checked using conditional statements (`if`, `elif`, `else`). If `spam` is equal to `1`, it will print "Hello". If `spam` is equal to `2`, it will print "Howdy". If the value of `spam` is anything else, it will print "Greetings!".

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

Ans: If your program is stuck in an endless loop and you want to stop its execution, you can typically press the following keys depending on your operating system and development environment:

1. \*\*Ctrl + C\*\*: This is a common key combination used to interrupt the execution of a program. It sends a termination signal to the program, causing it to stop running.
2. B) \*\*Ctrl + Break\*\*: On some systems, including Windows, pressing Ctrl + Break can also be used to break out of an infinite loop or stop the execution of a program.
3. \*\*Ctrl + D\*\*: This key combination is specific to Unix-like systems (e.g., Linux or macOS) and is used to send an "End of File" (EOF) signal, which can be used to terminate the program.

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

Ans: In Python, "break" and "continue" are both control flow statements used within loops, but they serve different purposes.

a) Break: The "break" statement is used to exit or terminate the current loop entirely. When encountered, the program immediately exits the loop, and execution continues with the code following the loop. Break is often used to prematurely stop a loop based on a certain condition. For example:

for i in range(1, 10):

if i == 5:

break

print(i)

In this case, when the value of `i` becomes 5, the `break` statement is executed, and the loop is terminated. The output will be: 1, 2, 3, 4.

b) Continue: The "continue" statement is used to skip the remaining code within the loop for the current iteration and move on to the next iteration. When encountered, the program immediately goes to the next iteration of the loop, skipping any code that follows the continue statement within that iteration. Continue is often used to skip certain iterations based on specific conditions. For example:

for i in range(1, 6):

if i == 3:

continue

print(i)

In this case, when the value of `i` is 3, the `continue` statement is executed, and the remaining code within that iteration (the `print(i)` statement) is skipped. The loop continues with the next iteration. The output will be: 1, 2, 4, 5.

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, `range(10)`, `range(0, 10)`, and `range(0, 10, 1)` are different ways of specifying the range of values that the loop will iterate over. However, they essentially produce the same result.

a)- `range(10)`: This creates a sequence of numbers starting from 0 (default start value) up to, but not including, 10 (specified stop value). The step value is implicitly assumed to be 1. So, `range(10)` is equivalent to `range(0, 10, 1)`.

b) `range(0, 10)`: This explicitly specifies the start value (0) and stop value (10), but the step value is again implicitly assumed to be 1. So, `range(0, 10)` is equivalent to `range(0, 10, 1)`.

c)`range(0, 10, 1)`: This explicitly specifies the start value (0), stop value (10), and step value (1). It creates a sequence of numbers starting from 0, incrementing by 1 at each step, and stopping just before reaching 10.

In summary, all three forms (`range(10)`, `range(0, 10)`, and `range(0, 10, 1)`) create a sequence of numbers from 0 to 9 (excluding 10) and can be used interchangeably in a for loop.

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

for i in range(1, 11):

print(i)

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

i = 1

while i <= 10:

print(i)

i += 1

Both programs achieve the same result of printing the numbers 1 to 10. The for loop iterates over the range from 1 to 10 (inclusive), while the while loop continues until the value of `i` reaches 11, printing the current value of `i` and incrementing it by 1 in each iteration.

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` that contains a function named `bacon()`, you can call the `bacon()` function using the dot notation. Here's how you would do it:

python

import spam

spam.bacon()

In this example, `spam` is the module name, and `bacon()` is the function name. By using `spam.bacon()`, you specify that you want to call the `bacon()` function from the `spam` module.