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

Ans:

The Boolean data type represents a logical value that can be either true or false. These two values are the only possible values for the Boolean data type.

In programming languages, the true value is often represented by the keyword "true," and the false value is represented by the keyword "false." The specific syntax for representing these values may vary slightly depending on the programming language you are using, but these are the commonly used representations.

Here are some examples of how you might write the Boolean values in python programming languages:

In Python:

True

False

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

Ans:The three types of Boolean operators are:

. AND Operator: The AND operator combines two Boolean expressions and returns true only if both expressions are true. It is typically represented by the symbol "&&" or the keyword "AND". For example:

- In Python: `expression1 and expression2`

OR Operator: The OR operator combines two Boolean expressions and returns true if either of the expressions is true. It is typically represented by the symbol "||" or the keyword "OR". For example:

- In Python: `expression1 or expression2`

. NOT Operator: The NOT operator negates the Boolean value of an expression. It returns true if the expression is false, and false if the expression is true. It is typically represented by the symbol "!" or the keyword "NOT". For example:

- In Python: `not expression`

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 Boolean operator:

1. AND Operator (&&):

| Expression 1 | Expression 2 | Result |

|--------------|--------------|--------|

| false | false | false |

| false | true | false |

| true | false | false |

| true | true | true |

2. OR Operator (||):

| Expression 1 | Expression 2 | Result |

|--------------|--------------|--------|

| false | false | false |

| false | true | true |

| true | false | true |

| true | true | true |

3. NOT Operator (!):

| Expression | Result |

|------------|--------|

| false | true |

| true | false |

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 given expressions are:

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

- (5 > 4) evaluates to True.

- (3 == 5) evaluates to False.

- The expression becomes True and False.

- The result is False.

ii.not (5 > 4):

- (5 > 4) evaluates to True.

- Applying the NOT operator negates the value, so the result is False.

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

- (5 > 4) evaluates to True.

- (3 == 5) evaluates to False.

- The expression becomes True or False.

- The result is True.

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

- (5 > 4) evaluates to True.

- (3 == 5) evaluates to False.

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

- Applying the NOT operator negates the value, so the result is False.

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

- True and True evaluates to True.

- True == False evaluates to False.

- The expression becomes True and False.

- The result is False.

vi.. (not False) or (not True):

- not False evaluates to True.

- not True evaluates to False.

- The expression becomes True or False.

- The result is True.

So, the values of the given expressions are as follows:

I.(5 > 4) and (3 == 5): False

ii. not (5 > 4): False

iii.. (5 > 4) or (3 == 5): True

iv.not ((5 > 4) or (3 == 5)): False

v. (True and True) and (True == False): False

vi. (not False) or (not True): True

5. What are the six comparison operators?

Ans:The six comparison operators are:

i. Equal to (==): This operator checks if two values are equal. If the values are equal, it returns true; otherwise, it returns false.

ii Not equal to (!=): This operator checks if two values are not equal. If the values are not equal, it returns true; otherwise, it returns false.

iii. Greater than (>): This operator checks if the left operand is greater than the right operand. If it is, it returns true; otherwise, it returns false.

vi. Less than (<): This operator checks if the left operand is less than the right operand. If it is, it returns true; otherwise, it returns false.

v. Greater than or equal to (>=): This operator checks if the left operand is greater than or equal to the right operand. If it is, it returns true; otherwise, it returns false.

vi. Less than or equal to (<=): This operator checks if the left operand is less than or equal to the right operand. If it is, it returns true; otherwise, it returns false.

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 (==) and assignment (=) operators are used for different purposes in programming, and they have distinct functionalities:

i Equal To (==) Operator:

The "equal to" operator (==) is a comparison operator used to check if two values are equal. It evaluates the operands on both sides of the operator and returns a Boolean value (true or false) based on whether the operands are equal or not. It does not change the values of the operands; it only compares them.

Example:

```python

x = 5

y = 10

if x == y:

print("x is equal to y")

else:

print("x is not equal to y")

```

In this example, the equal to operator (==) compares the values of x and y. Since x is not equal to y (5 is not equal to 10), it will print "x is not equal to y."

ii. Assignment (=) Operator:

The assignment operator (=) is used to assign a value to a variable. It takes the value on the right side of the operator and assigns it to the variable on the left side. It does not perform any comparison; it simply sets the value of the variable.

Example:

```python

x = 5

y = x

print(y) # Output: 5

```

In this example, the assignment operator (=) assigns the value of x (which is 5) to the variable y. As a result, the value of y becomes 5.

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 the given code, there are three blocks identified by indentation levels:

i. Block 1:

python

spam = 0

if spam == 10:

print('eggs')

```

This block consists of an if statement, which checks if the variable `spam` is equal to 10. However, since the value of `spam` is 0, the condition is not met, and the `print('eggs')` statement will not be executed.

ii. Block 2:

python

if spam > 5:

print('bacon')

else:

print('ham')

This block also contains an if statement. It checks if the variable `spam` is greater than 5. Since `spam` is 0 (not greater than 5), the condition in the first if statement fails, and the else block will be executed. Therefore, it will print 'ham'.

iii Block 3:

python

print('spam')

print('spam')

These two print statements are not indented and are not part of any conditional block. They will be executed regardless of the previous conditions. Therefore, 'spam' will be printed twice.

the code will output:

ham

spam

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:You can achieve this using an if-elif-else statement in Python. Here's the code to accomplish the task:

```python

spam = int(input("Enter a value for spam: "))

if spam == 1:

print("Hello")

elif spam == 2:

print("Howdy")

else:

print("Greetings!")

```

In this code, we use the `input()` function to get user input and store it in the variable `spam`. We convert the input to an integer using `int()` since `input()` returns a string by default.

The if-elif-else statement then checks the value of `spam` and executes the corresponding block of code based on the conditions:

- If `spam` is equal to 1, it prints "Hello."

- If `spam` is equal to 2, it prints "Howdy."

- For any other value of `spam`, it prints "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 want to interrupt or stop its execution, you can typically press the following key combinations depending on your operating system:

i. Windows/Linux:

- Press `Ctrl + C`. This sends a "SIGINT" (interrupt) signal to the running program, requesting it to terminate.

ii. macOS:

- Press `Command + .` (Command key and period key together). This sends an interrupt signal to the program, similar to `Ctrl + C` in Windows/Linux.

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

Ans:Both `break` and `continue` are control flow statements used in loops (such as `for` and `while`) in programming. They serve different purposes and have distinct effects on the loop's behavior:

i. `break`:

- When the `break` statement is encountered within a loop, it immediately terminates the loop's execution, and the program continues to execute the code following the loop.

- It is often used when a specific condition is met, and there is no need to continue the loop further. It allows you to exit the loop prematurely.

Example of `break`:

python

for i in range(1, 6):

if i == 3:

break

print(i)

Output:

1

2

In this example, the loop will terminate when `i` is equal to 3 because of the `break` statement.

ii. `continue`:

- When the `continue` statement is encountered within a loop, it immediately skips the rest of the current iteration and proceeds to the next iteration of the loop.

- It is often used when you want to skip certain iterations based on a condition, but you want the loop to continue running.

Example of `continue`:

`python

for i in range(1, 6):

if i == 3:

continue

print(i)

Output:

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: The difference between `range(10)`, `range(0, 10)`, and `range(0, 10, 1)` in a `for` loop:

i. `range(10)`:

- Generates a sequence of numbers from 0 to 9 (10 is excluded).

- Default starting value is 0, and default step size is 1.

ii. `range(0, 10)`:

- Same as `range(10)`, explicitly stating the starting value as 0.

- Generates a sequence of numbers from 0 to 9 (10 is excluded).

- Default step size is 1.

iii `range(0, 10, 1)`:

- Same as the previous cases, explicitly stating the starting value as 0 and step size as 1.

- Generates a sequence of numbers from 0 to 9 (10 is excluded) with a step size of 1.

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: two short Python programs—one using a `for` loop and the other using a `while` loop to print the numbers from 1 to 10:

i. Using a `for` loop:

```python

# Using a for loop

for i in range(1, 11):

print(i)

```

Output:

1

2

3

4

5

6

7

8

9

10

ii.Using a `while` loop:

python

# Using a while loop

count = 1

while count <= 10:

print(count)

count += 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?

Ans:If you have a function named `bacon()` inside a module named `spam`, you can call it after importing the `spam` module using the dot notation.

i. First, you need to import the `spam` module:

import spam

ii. Then, you can call the `bacon()` function using the dot notation:

spam.bacon()

```

For example, if the `bacon()` function in the `spam` module prints "I'm bacon!", the complete code would look like this:

spam.py (module):

def bacon():

print("I'm bacon!")

main.py:

import spam

spam.bacon()

Output:

I'm bacon!