**Python Assignment 2**

Q1. 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 **True** and **False**. These values are written as **True** and **False** respectively. They are used to represent logical values in programming, such as in conditional statements and loops. It's important to note that in Python, the first letter of each value is capitalized, and they are not enclosed in quotes.

Q2. What are the three different types of Boolean operators?

Ans. The three different types of Boolean operators are:

* Logical AND (&&):- The logical **AND** operator returns true if both operands are true; otherwise, it returns false. It evaluates to true only if all conditions are true.
* Logical OR (||):- The logical **OR** operator returns true if at least one of the operands is true; otherwise, it returns false. It evaluates to true if any of the conditions are true.
* Logical NOT (!):- The logical **NOT** operator is a unary operator that negates the value of its operand. It returns true if the operand is false, and false if the operand is true. It essentially reverses the logical state of a Boolean expression.

These operators are commonly used in programming to combine or manipulate Boolean values and expressions. They allow you to create more complex conditional statements and perform logical operations based on the truth or falsity of certain conditions.

Q3. 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 the three main Boolean operators:

**AND:-**

A B A AND B

0 0 0

0 1 0

1 0 0

1 1 1

**OR:-**

A B A OR B

0 0 0

0 1 1

1 0 1

1 1 1

**NOT:-**

A NOT A

0 1

1 0

In the truth tables, "A" and "B" represent the input Boolean values, and **"AND," "OR,"** and **"NOT"** represent the Boolean operators. The output values in the truth tables represent the result of applying the corresponding operator to the input values.

Q4. 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. Here are the values of the expressions:

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

The expression (5 > 4) evaluates to True

The expression (3 == 5) evaluates to False

True and False evaluates to False not (5 > 4)

The expression (5 > 4) evaluates to True

not True evaluates to False (5 > 4) or (3 == 5)

The expression (5 > 4) evaluates to True

The expression (3 == 5) evaluates to False

True or False evaluates to True

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

The expression (5 > 4) evaluates to True

The expression (3 == 5) evaluates to False

True or False evaluates to True

not True evaluates to False

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

True and True evaluates to True

True == False evaluates to False

True and False evaluates to False

(not False) or (not True)

not False evaluates to True

not True evaluates to False

True or False evaluates to True

Q5. What are the six comparison operators?

Ans. In programming, comparison operators are used to compare two values and return a Boolean value (True or False) based on the comparison. There are six comparison operators:

Greater than (>): This operator returns True if the left operand is greater than the right operand, otherwise it returns False.

Less than (<): This operator returns True if the left operand is less than the right operand, otherwise it returns False.

Greater than or equal to (>=): This operator returns True if the left operand is greater than or equal to the right operand, otherwise it returns False.

Less than or equal to (<=): This operator returns True if the left operand is less than or equal to the right operand, otherwise it returns False.

Equal to (==): This operator returns True if the left operand is equal to the right operand, otherwise it returns False.

Not equal to (!=): This operator returns True if the left operand is not equal to the right operand, otherwise it returns False.

For example, in Python:

x = 5

y = 3

print(x > y) # True

print(x < y) # False

print(x >= y) # True

print(x <= y) # False

print(x == y) # False

print(x != y) # True

Q6. How do you tell the difference between the equal to and assignment operators? Describe a condition and when you would use one.

Ans. In programming, the equal to operator == is used for comparison, while the assignment operator = is used to assign a value to a variable. The equal to operator is used to compare two values to see if they are equal, while the assignment operator is used to store a value in a variable.

For example, if we have a variable x and we want to assign the value 5 to it, we use the assignment operator like this:

x = 5

On the other hand, if we want to compare the value of x with another value, say 10, we use the equal to operator like this:

x = 5

if x == 10:

print("x is equal to 10")

else:

print("x is not equal to 10")

In this example, we first assign the value 5 to the variable x using the assignment operator. We then use the equal to operator to check if the value of x is equal to 10. Since the value of x is not equal to 10, the output of the program is "x is not equal to 10".

In summary, the equal to operator == is used for comparison, while the assignment operator = is used to assign a value to a variable. We use the equal to operator when we want to compare two values, while we use the assignment operator when we want to store a value in a variable.

Q7. 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, blocks of code are defined by their indentation level. Each block begins with a line that ends with a colon (:), and all subsequent lines indented to the same level belong to the same block.

Using this rule, the three blocks in the given code are:

The first block, consisting of a single line:

spam = 0

This line assigns the value 0 to the variable spam.

The second block, consisting of two lines:

if spam == 10:

print('eggs')

This block is an if statement that checks if the value of spam is equal to 10. If it is, the print('eggs') statement will be executed. However, since the value of spam is not 10, this block is skipped.

The third block, consisting of four lines:

if spam > 5:

print('bacon')

else:

print('ham')

print('spam')

print('spam')

This block is also an if statement that checks if the value of spam is greater than 5. If it is, the print('bacon') statement will be executed. If it is not, the print('ham') statement will be executed. Regardless of which statement is executed, the next two lines (print('spam') and print('spam')) are executed because they are not indented to the same level as the if statement.

Q8. 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 an example code in Python that checks the value of the variable spam and prints the corresponding message:

spam = 3 # change the value of spam to test the different conditions

if spam == 1:

print("Hello")

elif spam == 2:

print("Howdy")

else:

print("Greetings!")

In this code, the value of spam is first set to 3 (you can change it to test the different conditions). The code then checks the value of spam using the if-elif-else statement.

If spam is equal to 1, the code prints "Hello". If spam is equal to 2, the code prints "Howdy". If spam is anything else, the code prints "Greetings!".

Q9. If your programme is stuck in an endless loop, what keys you’ll press?

Ans. If your program is stuck in an endless loop, you can use the following key combinations to interrupt or halt the program execution:

**Ctrl+C** :- This is the most commonly used key combination to stop a running program on a command-line interface (CLI). It sends an interrupt signal to the program, causing it to terminate abruptly.

**Ctrl+Z** :- This key combination is used to suspend the running program and return to the command prompt. However, it doesn't terminate the program; you need to use other commands to terminate the program.

**\*Ctrl+\*** :- This key combination sends a SIGQUIT signal to the program, causing it to terminate with a core dump. This is a more forceful way to terminate a program than **Ctrl+C**.

It's important to note that using these key combinations to stop a program can result in data loss or corruption, especially if the program was in the middle of writing to a file or a database. Therefore, it's always better to design your program to handle errors gracefully and provide a way to exit the program without losing data.

Q10. How can you tell the difference between break and continue?

Ans. Both break and continue are control statements in programming languages like Python that allow you to alter the flow of execution in a loop.

Here's how you can tell the difference between break and continue:

Break statement: When a break statement is executed inside a loop (e.g. for loop, while loop), it immediately terminates the loop and control is transferred to the next statement after the loop. This means that the loop is completely stopped, and any remaining iterations are skipped. The break statement is often used to exit a loop early if a certain condition is met.

Here's an example code that uses break:

for i in range(1, 11):

if i == 5:

break # exit the loop when i is equal to 5

print(i)

Output:

Copy code

1

2

3

4

In this example, the loop iterates over the range of numbers from 1 to 10. When the value of i is equal to 5, the break statement is executed, causing the loop to terminate immediately. As a result, the remaining numbers (6 to 10) are not printed.

Continue statement: When a continue statement is executed inside a loop, it skips the current iteration of the loop and continues with the next iteration. This means that the loop doesn't terminate, but some iterations are skipped. The continue statement is often used to skip certain iterations if a certain condition is met.

Here's an example code that uses continue:

for i in range(1, 11):

if i % 2 == 0:

continue # skip even numbers

print(i)

Output:

Copy code

1

3

5

7

9

In this example, the loop iterates over the range of numbers from 1 to 10. When the value of i is even, the continue statement is executed, causing the loop to skip that iteration and continue with the next iteration. As a result, only the odd numbers (1, 3, 5, 7, 9) are printed.

In summary, the break statement is used to completely exit a loop, while the continue statement is used to skip certain iterations of a loop.

Q11. In a for loop, what is the difference between range (10), range (0, 10), and range (0, 10, 1)?

Ans. In Python, range() is a built-in function that generates a sequence of numbers. It's commonly used in for loops to iterate over a range of values.

Here's the difference between range(10), range(0, 10), and range(0, 10, 1) in a for loop:

range(10) generates a sequence of numbers starting from 0 and ending at 9 (i.e., the last number is not included). This is the default behavior of range().

For example, the following code iterates over a sequence of numbers generated by range(10):

for i in range(10):

print(i)

Output:

0

1

2

3

4

5

6

7

8

9

range(0, 10) generates a sequence of numbers starting from 0 and ending at 9 (i.e., the last number is not included). This is equivalent to range(10).

For example, the following code iterates over a sequence of numbers generated by range(0, 10):

for i in range(0, 10):

print(i)

Output:

0

1

2

3

4

5

6

7

8

9

range(0, 10, 1) generates a sequence of numbers starting from 0 and ending at 9 (i.e., the last number is not included), with a step size of 1. This is equivalent to range(10).

For example, the following code iterates over a sequence of numbers generated by range(0, 10, 1):

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

print(i)

Output:

0

1

2

3

4

5

6

7

8

9

In summary, range(10), range(0, 10), and range(0, 10, 1) are equivalent and generate the same sequence of numbers. The difference is only in the way the arguments are specified. The default start value is 0, and the default step value is 1.

Q12. 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)

Output:

1

2

3

4

5

6

7

8

9

10

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

Output:

1

2

3

4

5

6

7

8

9

10

Both programs produce the same output, but they use different types of loops to achieve the same result.

Q13. If you had a function named bacon() inside a module named spam, how would you call it after importing spam?

Ans. If you had a function named bacon() inside a module named spam, you could call it after importing spam by using the dot notation to access the function:

import spam

spam.bacon()

This assumes that the bacon() function is defined in the spam module and is accessible from outside the module (i.e., it's not a private function). If bacon() is a private function (i.e., its name starts with an underscore), it's not intended to be accessed from outside the module, and attempting to do so may result in an Attribute Error.