***ASSIGNMENT\_2***

**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 possible values**: True** and **False**. They represent the logical concepts of true and false, respectively. These values are case-sensitive, so make sure to capitalize the first letter.

value1 = **True**

value2 = **False**

we can also use these Boolean values for logical operations and comparisons in Python.

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

***ANS:***

The three different types of Boolean operators in Python are:

**AND Operator**: The and operator returns True if both the operands are True. Otherwise, it returns False.

*Example:*

x = True

y = False

result = x **and** y

print(result) # Output: **False**

**OR Operator**: The or operator returns True if at least one of the operands is True. If both operands are False, it returns False.

*Example:*

x = True

y = False

result = x **or** y

print(result) # Output: **True**

**NOT Operator**: The not operator is a unary operator that negates the Boolean value of the operand. If the operand is True, not returns False, and vice versa.

*Example:*

x = True

result = not x

print(result) # Output: False

**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:***

**AND Operator (and):**

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

**OR Operator (or):**

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

**NOT Operator (not):**

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

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.

**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:***

1. (5 > 4) **and** (3 == 5)

(5 > 4) evaluates to **True**

(3 == 5) evaluates to **False**

True and False evaluates to **False**

So, the value of the expression is **False.**

1. **not (5 > 4)**

(5 > 4) evaluates to **True**

not True evaluates to **False**

So, the value of the expression is **False.**

1. (5 > 4) **or** (3 == 5)

(5 > 4) evaluates to **True**

(3 == 5) evaluates to **False**

True or False evaluates to **True**

So, the value of the expression is **True.**

1. **not** ((5 > 4) **or** (3 == 5))

(5 > 4) evaluates to **True**

(3 == 5) evaluates to **False**

True or False evaluates to **True**

not True evaluates to **False**

So, the value of the expression is **False**.

1. (True and True) **and** (True == False)

True and True evaluates to **True**

True == False evaluates to **False**

So, the value of the expression is **False.**

1. (not False) **or** (not True)

not False evaluates to **True**

not True evaluates to **False**

True or False evaluates to **True**,

So, the value of the expression is **True**

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

***ANS:***

The six comparison operators in Python are as follows:

**Equal to (==):** Checks if two values are equal and returns **True** if they are, and **False** otherwise.

**Not equal to (!=):** Checks if two values are not equal and returns **True** if they are not, and **False** if they are equal.

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

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

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

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

These comparison operators are used to compare values and expressions in Python and return Boolean values based on the result of the comparison.

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

***ANS:***

Equal to (==) operator: The double equals sign (==) is used for comparison to check if two values are equal. It returns True if the values are equal and False if they are not. It is used in conditional statements, such as if statements or loops, to evaluate conditions.

Example:

x = 5

y = 3

if x == y:

print("x and y are equal")

Assignment (=) operator: The single equals sign (=) is used for assignment, which assigns a value to a variable. It takes the value on the right and assigns it to the variable on the left.

Example:

x = 5 # Assigns the value 5 to the variable x

the equal to (==) operator is used to compare values, while the assignment (=) operator is used to assign values to variables.

To illustrate their difference in a condition, consider the following example:

x = 10

y = 5

if x == y: # Comparison using the equal to operator

print("x and y are equal")

else:

print("x and y are not equal")

x = y # Assignment of y's value to x

print(x) # Output: 5

In this example, the equal to operator (==) is used in the if statement to compare the values of x and y. The assignment operator (=) is then used to assign the value of y to x, resulting in x being assigned the value 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 provided code, there are three blocks based on indentation. Blocks in Python are defined by the indentation level and are used to group statements together. Let's identify the blocks in the code:

spam = 0

if spam == 10:

print('eggs')

**Block 1:** This block contains the assignment statement spam = 0 and an if statement. It is indented by four spaces, indicating that it belongs to the if statement. If the condition spam == 10 is true, it would execute the statement print('eggs').

if spam > 5:

print('bacon')

else:

print('ham')

**Block 2:** This block starts with an if statement and is indented by four spaces. It contains two nested print statements. If the condition spam > 5 is true, it would execute the statement print('bacon'). Otherwise, it would execute the else block and print ham.

print('spam')

print('spam')

Block 3: This block consists of two consecutive print statements. It is not indented and is executed unconditionally after the previous blocks.

**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:***

we can use an if-elif-else statement to achieve the desired output based on the value stored in the spam variable. Here's an example code snippet that accomplishes this:

spam = 1

if spam == 1:

print("Hello")

elif spam == 2:

print("Howdy")

else:

print("Greetings!")

In this code, if the value stored in spam is 1, it will print "Hello". If the value is 2, it will print "Howdy". Otherwise, if the value is anything else, it will print "Greetings!". You can modify the value of spam to test different scenarios and see the corresponding output.

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

***ANS:***

If program is stuck in an endless loop and need to interrupt its execution, we can typically press the following key combination based on our operating system:

Windows/Linux: Press Ctrl + C keys simultaneously. This sends an interrupt signal to the running program, causing it to terminate.

Mac: Press Command + C keys simultaneously. This sends a signal to stop the execution of the program.

By pressing these key combinations, you can forcefully stop the program and break out of the endless loop.

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

***ANS:***

In Python, break and continue are two different control flow statements used within loops to alter the execution flow. Here's how they differ:

**break statement**: The break statement is used to terminate the loop prematurely and immediately exit the loop. When encountered, the break statement will exit the current loop, regardless of whether the loop condition is still satisfied or not. After the break statement is executed, the program execution continues with the code immediately following the loop.

Example:

for i in range(5):

if i == 3:

break

print(i)

# Output: **0 1 2**

In this example, the loop is terminated when i becomes 3, and the program continues executing the code after the loop.

**continue statement:** The continue statement is used to skip the remaining code within the current iteration of the loop and move to the next iteration. When encountered, the continue statement jumps to the next iteration without executing the subsequent statements in the loop body. The loop continues executing from the beginning with the next iteration.

Example:

for i in range(5):

if i == 2:

continue

print(i)

# Output: 0 1 3 4

In this example, when i is equal to 2, the continue statement is triggered, skipping the print(i) statement. The loop then proceeds to the next iteration.

**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 range() function can be used to generate a sequence of numbers. Let's explore the differences between range(10), range(0, 10), and range(0, 10, 1):

**range(10):** This form of range() specifies the stop value, which is exclusive. It generates a sequence of numbers starting from 0 and up to (but not including) the specified stop value, which is 10 in this case.

Example:

for i in range(10):

print(i)

# Output: 0 1 2 3 4 5 6 7 8 9

In this case, the loop iterates from 0 to 9 (10 iterations) since the stop value is 10.

**range(0, 10):** This form of range() specifies both the start and stop values. It generates a sequence of numbers starting from the specified start value (0) and up to (but not including) the specified stop value (10).

Example:

for i in range(0, 10):

print(i)

# Output: 0 1 2 3 4 5 6 7 8 9

The output is the same as in the previous example because the start value defaults to 0 if not explicitly specified.

**range(0, 10, 1**): This form of range() specifies both the start and stop values, along with the step value. It generates a sequence of numbers starting from the specified start value (0), incrementing by the specified step value (1), and up to (but not including) the specified stop value (10).

Example:

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

print(i)

# Output: 0 1 2 3 4 5 6 7 8 9

The output is the same as in the previous examples because the step value defaults to 1 if not explicitly specified.

**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:***

***# Using a for loop***

for i in range(1, 11):

print(i)

**Output:**

1

2

3

4

5

6

7

8

9

10

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

***# Using a while loop***

i = 1

while i <= 10:

print(i)

i += 1

Output:

1

2

3

4

5

6

7

8

9

10

In both cases, the numbers from 1 to 10 are printed, with the for loop iterating over the range from 1 to 10, and the while loop executing as long as the value of i is less than or equal to 10.

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

***ANS:***

If we have a function named bacon() inside a module named spam, we can call it after importing the spam module by using the dot notation. Here's how we would do it:

import spam

spam.bacon()

In this example, the spam module is imported using the import statement. Then, you can call the bacon() function by prefixing it with the module name and a dot (spam. Bacon()). This notation specifies that the bacon() function is located within the spam module, allowing you to call it and execute its code.