Assignment – 2

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

The Boolean datatype in Python has two values i.e., True and False. These values are used to indicate the truthfulness or falsity of a condition or an expression. These values must always be capitalized with an uppercase i.e., ‘T’ and ‘F’. This will be used in the following ways:

* x = 5

y = 10

is\_greater = x > y

if is\_greater == True:

print("x is greater than y.")

else:

print("x is not greater than y.")

* name = "Alice"

is\_admin = False

if is\_admin:

print(f"Welcome, {name} Admin")

else:

print(f"Hello, {name}!")

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

The following are the three different types of Boolean operators in python.

* **Logical AND(and):** The ‘and’ operator returns ‘True’ if both conditions are ‘True’ otherwise, it returns ‘False’. It evaluates the second condition only if the first condition is ‘True’.

x = 5

y = 10

result = (x < y) and (y > 0)

print(result) # Output: True

* **Logical OR(or):** The ‘or’ operator returns ‘True’ if at least one of the conditions is ‘True’ otherwise, it returns ‘False’. This operator only evaluates the second condition if the first condition is ‘False’.

x = 5

y = 10

result = (x > y) or (y > 0)

print(result) # Output: True

* **Logical NOT(not):** The ‘not’ operator is a unary operator that returns the opposite Boolean value of the condition. If it is true, it returns false and vice-versa.

x = 5

y = 10

result = not (x > y)

print(result) # Output: True

1. **Make a list of each Boolean operator's truth tables (i.e. every possible combination of Boolean values for the operator and what it evaluates ).**

These are the following truth tables of the Boolean operators in Python.

* **Logical AND (and):**

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

* **Logical OR (or):**

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

* **Logical NOT (not):**

|  |  |
| --- | --- |
| **Condition - 1** | **Result** |
| True | False |
| False | True |

1. **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)**

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

(5 > 4) evaluates to True whereas (3 == 5) evaluates to False. Therefore, True and False results in False.

* not (5 > 4):

not (5 > 4) evaluates to False because (5 > 4) results in True, and the not operator inverts the Boolean value.

* (5 > 4) or (3 == 5):

(5 > 4) evaluates to True whereas (3 == 5) evaluates to False. Therefore, True or False results in True.

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

not ((5 > 4) or (3 == 5)) evaluates to False because (5 > 4) or (3 == 5) results in True, and the not operator inverts the Boolean value.

* (True and True) and (True == False):

(True and True) evaluates to True whereas (True == False) evaluates to False. Therefore, True and False results in False.

* (not False) or (not True):

(not False) evaluates to True whereas (not True) evaluates to False. Therefore, True or False results in True.

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

The six comparison operators in Python are used to compare values and expressions. They return Boolean values (True or False) based on the comparison result. Here are the six comparison operators:

* **Equal to (==):** Checks if the values on both sides of the operator are equal.
* **Not equal to (!=):** Checks if the values on both sides of the operator are not equal.
* **Greater than (>):** Checks if the value on the left side is greater than the value on the right side.
* **Less than (<):** Checks if the value on the left side is less than the value on the right side.
* **Greater than or equal to (>=):** Checks if the value on the left side is greater than or equal to the value on the right side.
* **Less than or equal to (<=):** Checks if the value on the left side is less than or equal to the value on the right side.

The following Python code snippet demonstrates the functionality of the above 6 Python comparison operators.

x = 5

y = 10

print(x == y) # Output: False

print(x != y) # Output: True

print(x > y) # Output: False

print(x < y) # Output: True

print(x >= y) # Output: False

print(x <= y) # Output: True

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

In Python, the ‘equal to’ operator(==) and the ‘assignment’ operator(=) are two different operators with two distinct purposes. Let's understand the functionality of these operators.

* **Equal to operator (==):**

This equal-to operator is used for comparison. It checks whether the two values are equal or not. If they are equal, it returns a Boolean result based on the comparison.

**Syntax**: value1 == value2

* **Assignment Operator (=):**

This assignment operator is used to assign a value to a variable. It takes the value on the right side and assigns it to the variable on the left side.

**Syntax:** Variable = value

The following code snippet will give the functionality of the above 2 operators.

player1\_score = 100

player2\_score = 95

if player1\_score == player2\_score:

print("Congratulations! players scored same”)

else:

print("Players have different scores.")

# Output: Players have different scores.

In this example, we use the equal to operator (==) to compare the values of player1\_score and player2\_score. If both scores are equal, the condition inside the if statement is True, and the program will print the congratulatory message. Otherwise, if the scores are different, the condition will be False, and the program will print the message indicating that the players have different scores.

1. **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')**

In the provided code snippet, the indentation determines the blocks. Python uses indentation to define the scope of code blocks. The blocks are separated based on their indentation level. Here are the three blocks in the code:

Block 1: spam = 0

This is a simple statement that assigns the value 0 to the variable spam.

Block 2:

if spam == 10:

print('eggs')

if spam > 5:

print('bacon')

else:

print('ham')

This is a series of conditional statements. It consists of three parts:

if spam == 10: This is the first if statement that checks if the value of spam is equal to 10. If the condition is True, it will execute the following indented code block.

if spam > 5: This is the second if statement that checks if the value of spam is greater than 5. If the condition is True, it will execute the following indented code block.

else: This is the else statement that executes the indented code block if none of the preceding if statements' conditions are True.

Block 3:

print('spam')

print('spam')

This is a block of code that contains two print statements. They will be executed unconditionally after the previous conditional block because they are not indented under any conditional statement.

**Note:**

It's important to note that the correct indentation is crucial in Python, as it determines the structure and flow of the code. Indentation errors can lead to unexpected behaviour or syntax errors.

1. **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.**

spam = int(input(“Enter the value”))

If spam == 1:

print(“Hello”)

elif spam == 2:

print(“Howdy”)

else:

print(“Greetings!”)

In the above code snippet, we are taking user input which stores that value in a variable spam. If spam is equal to 1, it prints Hello. if spam is equal to 2, it prints Howdy otherwise Greetings!

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

If your program is stuck in an endless loop, you can forcefully stop it by pressing the following keys:

* **Ctrl + C**: This key combination is used to send an interrupt signal to the running program in most terminals and IDEs. Pressing this key combination will force the program to stop immediately, breaking out of the endless loop.
* **Ctrl + Break:** On some platforms or IDEs, the key combination may be Ctrl + Break instead of Ctrl + C. This also sends an interrupt signal and forcefully stops the running program.
* **Ctrl + Z:** On Windows platforms, you can try pressing Ctrl + Z to suspend the program. This might not stop the program immediately but may allow you to regain control and terminate it using other methods.

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

In Python, break and continue are two different control flow statements used within loops.

* **break:** It is used to terminate the loop when a certain condition is met. Once the break statement is encountered, the loop immediately exits, and the program continues to execute the code after the loop.

for i in range(5):

if i == 3:

break

print(i)

# Output: 0 1 2

* **continue:** It is used to skip the current iteration of the loop and move to the next iteration, without executing the remaining code in the loop block for the current iteration.

for i in range(5):

if i == 2:

continue

print(i)

# Output: 0 1 3 4

In conclusion, break is used to exit the loop entirely, while continue is used to skip the current iteration and proceed with the next iteration. Both break and continue are useful tools for controlling the flow of loops in Python.

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

In a for loop, range(10), range(0, 10), and range(0, 10, 1) produce the same sequence of numbers. The difference lies in their syntax and arguments, which can affect the behaviour of the loop.

* **range(10):** The range(10) function generates a sequence of numbers from 0 (inclusive) up to 10 (exclusive) with a step of 1 (default step).

for i in range(10):

print(i)

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

* **range(0, 10):** The range(0, 10) function also generates a sequence of numbers from 0 (inclusive) up to 10 (exclusive) with a step of 1 (default step). The start value (0 in this case) is explicitly specified as the first argument.

for i in range(0, 10):

print(i)

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

* **range(0, 10, 1):** The range(0, 10, 1) function generates a sequence of numbers from 0 (inclusive) up to 10 (exclusive) with a step of 1. The start value (0 in this case) and the step value (1 in this case) are explicitly specified as the first and second arguments, respectively.

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

print(i)

# Output: 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.**

**Using a for loop:**

for i in range(1, 11):

print(i)

#output:

1

2

3

4

5

6

7

8

9

10

**Using a while loop:**

num = 1

while num <= 10:

print(num)

num += 1

#output:

1

2

3

4

5

6

7

8

9

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, you can call it after importing the spam module using the dot notation.

import spam # Import the 'spam' module

# Call the 'bacon()' function from the 'spam' module

spam.bacon()

In this code, we first import the spam module using the import statement. Then, we call the bacon() function using the dot notation, where spam is the module name and bacon() is the function inside that module.

If the function bacon() is defined inside a module named spam, the function call spam.bacon() will execute the code inside the bacon() function from the imported spam module.