**Assignment 2**

Q1. What are the two values of the Boolean data type? How do you write them?

The two values of the Boolean data type in Python are True and False.

To write True in Python, simply type True. Similarly, to write False in Python, type False. Both True and False are reserved keywords in Python and they are case-sensitive, meaning that true and false will not be recognized as Boolean values.

Here is an example of assigning Boolean values to variables in Python:

x = True

y = False

print(x) # output: True

print(y) # output: False

In Python, Boolean values are commonly used for controlling the flow of a program through conditional statements, such as if statements and while loops.

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

The three different types of Boolean operators are:

AND operator: This operator returns True only if both the operands are True. Otherwise, it returns False. In Python, the AND operator is represented by the keyword and.

OR operator: This operator returns True if at least one of the operands is True. Otherwise, it returns False. In Python, the OR operator is represented by the keyword or.

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

Here's an example of how you can use these Boolean operators in Python:

x = 5

y = 10

z = 15

# Using AND operator

if x < y and y < z:

print("y is between x and z")

# Using OR operator

if x > y or y > z:

print("At least one of the conditions is true")

# Using NOT operator

if not(x > y):

print("x is not greater than y")

In this example, the AND operator is used to check if y is between x and z. The OR operator is used to check if at least one of the conditions is true. Finally, the NOT operator is used to check if x is not greater than y.

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 ).

**AND operator**

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

**OR operator**

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

**NOT operator**

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

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)

(5 > 4) and (3 == 5) evaluates to False. The expression (5 > 4) is True and (3 == 5) is False. Since and requires both operands to be True for the result to be True, the entire expression evaluates to False.

not (5 > 4) evaluates to False. The expression (5 > 4) is True. The not operator returns the opposite Boolean value of its operand, so not (5 > 4) is False.

(5 > 4) or (3 == 5) evaluates to True. The expression (5 > 4) is True. The or operator returns True if at least one of the operands is True, so the entire expression evaluates to True.

not ((5 > 4) or (3 == 5)) evaluates to False. The expression (5 > 4) or (3 == 5) evaluates to True because (5 > 4) is True. The not operator returns the opposite Boolean value of its operand, so not ((5 > 4) or (3 == 5)) is False.

(True and True) and (True == False) evaluates to False. The expression (True and True) is True. However, (True == False) is False, so the entire expression evaluates to False.

(not False) or (not True) evaluates to True. The expression not False is True. The expression not True is False. The or operator returns True if at least one of the operands is True, so the entire expression evaluates to True.

Q5. What are the six comparison operators?

> Greater than

< Less than

>= Greater than or equal to

<= Less than or equal to

== Equal to

!= Not equal to

Q6. 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 is denoted by == and is used to compare the equality of two values. On the other hand, the assignment operator is denoted by = and is used to assign a value to a variable.

To differentiate between the two operators, remember that the == operator is used to compare values, while the = operator is used to assign values.

Here's an example to illustrate the difference:

x = 5 # assignment operator assigns the value 5 to the variable x

y = 10 # assignment operator assigns the value 10 to the variable y

if x == y:

print("x and y are equal")

else:

print("x and y are not equal")

In this example, we first use the assignment operator to assign the values 5 and 10 to the variables x and y, respectively. Then we use the equal to operator to compare the values of x and y. Since x is not equal to y, the else block will execute, and the program will output "x and y are not equal".

We would use the equal to operator when we need to compare the equality of two values, for example, in an if statement or a loop condition. We would use the assignment operator when we need to assign a value to a variable, for example, when initializing a variable or updating its value.

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')

spam = 0 # Block 1: This is a single statement block that assigns the value 0 to the variable 'spam'.

if spam == 10: # Block 2: This is a multi-statement block that contains an 'if' statement.

print('eggs') # This is the first statement of Block 2. It will only execute if spam == 10.

if spam > 5: # This is a second multi-statement block that contains another 'if' statement.

print('bacon') # This is the first statement of Block 3. It will execute if spam > 5.

else: # This 'else' statement is part of Block 3.

print('ham')

print('spam') # These two statements are also part of Block 3.

print('spam') # This statement is not indented, so it is not part of any block.

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.

spam = 3 # set the value of spam

if spam == 1:

print('Hello')

elif spam == 2:

print('Howdy')

else:

print('Greetings!')

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

If your program is stuck in an endless loop and you need to stop it, you can typically press the Ctrl and C keys simultaneously to send an interrupt signal (often called a "keyboard interrupt" or "SIGINT") to the program. This should cause the program to terminate and return control back to the shell or command prompt.

On some platforms, such as Windows, you may need to press Ctrl and Break keys instead of Ctrl and C. Additionally, if the program is running in an IDE or graphical interface, you may need to consult the documentation for your specific environment to determine how to interrupt the program.

It's worth noting that interrupting a program may not always be a clean or safe way to terminate it, and there may be some risk of data loss or other unintended consequences. If possible, it's generally best to try to design your programs so that they do not get stuck in endless loops, or to use debugging tools and techniques to identify and fix the underlying issue.

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

In Python, break and continue are two control flow statements that can be used to modify the behavior of loops such as for and while loops. The main difference between them is that break is used to exit a loop entirely, while continue is used to skip over one iteration of a loop.

Here's how they work:

break: When executed inside a loop, the break statement immediately terminates the loop and control is transferred to the statement immediately following the loop. This means that the loop will not continue executing any further iterations.

Example usage of break:

for i in range(10):

if i == 5:

break

print(i)

In this example, the loop will iterate from 0 to 4, and when i is equal to 5, the break statement will be executed and the loop will be terminated.

continue: When executed inside a loop, the continue statement skips over the rest of the current iteration and moves on to the next iteration. This means that any code following the continue statement within the current iteration will be skipped, but the loop will continue to execute with the next iteration.

Example usage of continue:

for i in range(10):

if i == 5:

continue

print(i)

In this example, the loop will iterate from 0 to 9, and when i is equal to 5, the continue statement will be executed and the rest of the current iteration (i.e., the print statement) will be skipped. The loop will then continue with the next iteration.

In summary, break is used to immediately terminate a loop, while continue is used to skip over the current iteration and continue with the next iteration of a loop.

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

In Python, the range() function is used to generate a sequence of numbers that can be used in a loop. The range() function can be called with different arguments to generate different sequences. Here are the differences between range(10), range(0, 10), and range(0, 10, 1) in a for loop:

range(10): This generates a sequence of numbers from 0 to 9 (i.e., 10 numbers in total). This is the most common way to use range().

Example:

for i in range(10):

print(i)

This will print the numbers 0 to 9.

range(0, 10): This generates a sequence of numbers from 0 to 9 (i.e., 10 numbers in total), just like range(10). The only difference is that you explicitly specify the starting value (0) in this case.

Example:

for i in range(0, 10):

print(i)

This will also print the numbers 0 to 9.

range(0, 10, 1): This generates a sequence of numbers from 0 to 9 (i.e., 10 numbers in total), just like range(10) and range(0, 10). The difference here is that you explicitly specify the starting value (0) and the step size (1) in this case. The step size determines the difference between each pair of adjacent numbers in the sequence.

Example:

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

print(i)

This will also print the numbers 0 to 9.

In summary, all three of these expressions generate the same sequence of numbers from 0 to 9, and can be used interchangeably in most cases. The only difference is in how you specify the arguments to the range() function.

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.

Here's a short Python 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 will produce the same output:

1

2

3

4

5

6

7

8

9

10

Q13. 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 spam using the following syntax:

import spam

spam.bacon()

Here, we import the spam module using the import statement. This makes all the functions and variables defined in the spam module available to our program. We can then call the bacon() function using the syntax spam.bacon(). This tells Python to look for the bacon() function inside the spam module and call it.

Alternatively, you can import only the bacon() function from the spam module using the following syntax:

from spam import bacon

bacon()

Here, we use the from ... import statement to import only the bacon() function from the spam module. This makes the bacon() function available directly in our program, so we can call it using the syntax bacon() without having to prefix it with the spam module name.