# Assignment 2:

## Q1

The Boolean data type in Python has two possible values: True and False. These values represent the two states of Boolean logic: true and false.

In Python, we write these Boolean values as True and False (Capital T and F).

## Q2

The three different types of Boolean operators in Python are:

**Logical AND (and):**

The logical AND operator returns True if both operands are True, and False otherwise.

**Logical OR (or):**

The logical OR operator returns True if at least one of the operands is True, and False if both operands are False.

**Logical NOT (not):**

The logical NOT operator negates the value of the operand. If the operand is True, the NOT operator returns False, and vice versa.

## Q3

Here are the truth tables for each Boolean operator:

Logical AND (and):

Operand 1 Operand 2 Result

True True True

True False False

False True False

False False False

Logical OR (or):

Operand 1 Operand 2 Result

True True True

True False True

False True True

False False False

Logical NOT (not):

Operand Result

True False

False True

## Q4

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

(5 > 4) evaluates to True.

(3 == 5) evaluates to False.

The expression becomes True and False, which evaluates to False.

**not (5 > 4):**

(5 > 4) evaluates to True.

The not operator negates the value, so True becomes False.

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

(5 > 4) evaluates to True.

(3 == 5) evaluates to False.

The expression becomes True or False, which evaluates to True.

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

not((5 > 4)) evaluates to False.

(3 == 5) evaluates to False.

The expression becomes not (True or False), which evaluates to False.

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

(True and True) evaluates to True.

(True == False) evaluates to False.

The expression becomes True and False, which evaluates to False.

**(not False) or (not True):**

not False evaluates to True.

not True evaluates to False.

The expression becomes True or False, which evaluates to True.

The values of the given expressions are:

False

False

True

False

False

True

## Q5

The six comparison operators in Python are:

**Equal to (==):**

Checks if two operands are equal.

Example: 5 == 5 evaluates to True.

**Not equal to (!=):**

Checks if two operands are not equal.

Example: 5 != 3 evaluates to True.

**Greater than (>):**

Checks if the left operand is greater than the right operand.

Example: 5 > 3 evaluates to True.

**Less than (<):**

Checks if the left operand is less than the right operand.

Example: 3 < 5 evaluates to True.

**Greater than or equal to (>=):**

Checks if the left operand is greater than or equal to the right operand.

Example: 5 >= 5 evaluates to True.

**Less than or equal to (<=):**

Checks if the left operand is less than or equal to the right operand.

Example: 3 <= 5 evaluates to True.

## Q6

The equal to (==) operator and the assignment (=) operator serve different purposes in Python and can be distinguished as :

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

The equal to operator (==) is used for comparison.

It checks if the values on both sides of the operator are equal.

It returns True if the values are equal and False otherwise.

**Assignment (=) operator:**

The assignment operator (=) is used for variable assignment.

It assigns a value on the right side to a variable on the left side.

When we want to compare two values for equality, we use the equal to (==) operator. For example, in an if statement, we might use if x == 5: to check if x is equal to 5 before executing a block of code based on that condition.

When we want to assign a value to a variable, we use the assignment (=) operator. For example, we might use x = 5 to assign the value 5 to the variable x.

## Q7

In the given code, the three blocks can be identified based on the indentation levels. In Python, indentation is used to define blocks of code. Here's the code with the blocks:

spam = 0

# Block 1

if spam == 10:

print('eggs')

# Block 2

if spam > 5:

print('bacon')

else:

# Block 3

print('ham')

print('spam')

print('spam')

## Q8

spam = 1

if spam == 1:

print("Hello")

elif spam == 2:

print("Howdy")

else:

print("Greetings!")

## Q9

Ctrl + C

Pressing Ctrl + C , causes it to terminate. This key combination is commonly used to break out of an infinite loop or stop the execution of a program that is not responding.

## Q10

In Python, break and continue are control flow statements used within loops to alter the flow of execution. They serve different purposes:

**break:**

When encountered within a loop (such as a for or while loop), the break statement immediately terminates the loop.

It allows we to exit the loop prematurely, regardless of any remaining iterations.

After encountering a break, the program execution continues from the next statement after the loop.

**continue:**

When encountered within a loop, the continue statement skips the remaining code inside the loop for the current iteration and moves to the next iteration.

It allows we to skip specific iterations and continue with the next iteration of the loop.

After encountering continue, the program execution jumps to the beginning of the loop for the next iteration.

## Q11

**range(10):**

In this case, range(10) generates a sequence of numbers starting from 0 (default) and ending at 10 (exclusive) with a default step of 1.

The start value is not explicitly mentioned, so it defaults to 0.

The stop value is 10, which means the sequence will include numbers up to 9.

The step value is not explicitly mentioned, so it defaults to 1.

**range(0, 10):**

In this case, range(0, 10) generates a sequence of numbers starting from 0 (inclusive) and ending at 10 (exclusive) with a step of 1.

The start value is explicitly mentioned as 0.

The stop value is 10, which means the sequence will include numbers up to 9.

The step value is not explicitly mentioned, so it defaults to 1.

**range(0, 10, 1):**

In this case, range(0, 10, 1) generates a sequence of numbers starting from 0 (inclusive) and ending at 10 (exclusive) with a step of 1.

The start value is explicitly mentioned as 0.

The stop value is 10, which means the sequence will include numbers up to 9.

The step value is explicitly mentioned as 1.

## Q12

**#** **Using a for loop**

for i in range(1, 11):

print(i)

**# Using a while loop**

i = 1

while i <= 10:

print(i)

i += 1

## Q13

After importing the module named spam, we can call the function named bacon() using the following syntax:

**import spam**

**spam.bacon()**