Python Assignment – 2

1. Ans:

   The Boolean data type represents two values: true and false. These values are used to represent binary conditions in computer programming and logic.

   In many programming languages, the true value is typically represented as the keyword "true", and the false value is represented as the keyword "false".

   For example, in Python:

   X=True
   Y=False

2. **Ans:**

   a. **AND Operator (&&):** This operator returns true if both of the operands are true. In most programming languages, it's represented as &&.
   b. **OR Operator (||):** This operator returns true if at least one of the operands is true. It's typically represented as ||.
   c. **NOT Operator (!):** This operator is used to negate a Boolean value, meaning it returns the opposite of the operand's value. It's typically represented as !.

3. **Ans:**

   AND Operator (&&):

Operand 1	Operand 2	Result
T	T	T
T	F	F
F	T	F
F	F	F

   OR Operator (||):

Operand 1	Operand 2	Result
T	T	T
T	F	T
F	T	T
F	F	F

NOT Operator (!):

Operand	Result
T	F
F	T

4. Ans:

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

5 > 4 is true because 5 is greater than 4.
3 == 5 is false because 3 is not equal to 5.
Therefore, the expression becomes true and false, which evaluates to false.

2. not (5 > 4):
5 > 4 is true.
Not negates the value, so it becomes not true, which evaluates to false.

3. (5 > 4) or (3 == 5):
5 > 4 is true.
3 == 5 is false.
Therefore, the expression becomes true or false, which evaluates to true.

4. not ((5 > 4) or (3 == 5)):
(5 > 4) or (3 == 5) is true or false, which is true.
not negates the value, so it becomes not true, which evaluates to false.

5. (True and True) and (True == False):
The first part, True and True, is true.
The second part, True == False, is false.
Therefore, the expression becomes true and false, which evaluates to false.

6. (not False) or (not True):
not False is true.
not True is false.
Therefore, the expression becomes true or false, which evaluates to true.

5. Ans:

the six comparison operators are:

1. `==` (Equal to)
2. `! =` (Not equal to)
3. `>` (Greater than)
4. `<` (Less than)
5. `>=` (Greater than or equal to)
6. `<=` (Less than or equal to)

6. Ans:

we can differentiate between the equal to (`==`) and assignment (`=`) operators by considering their context and usage:

1. Equal To Operator (==):

The equal to operator is used for comparison.

It checks if two values or expressions are equal.

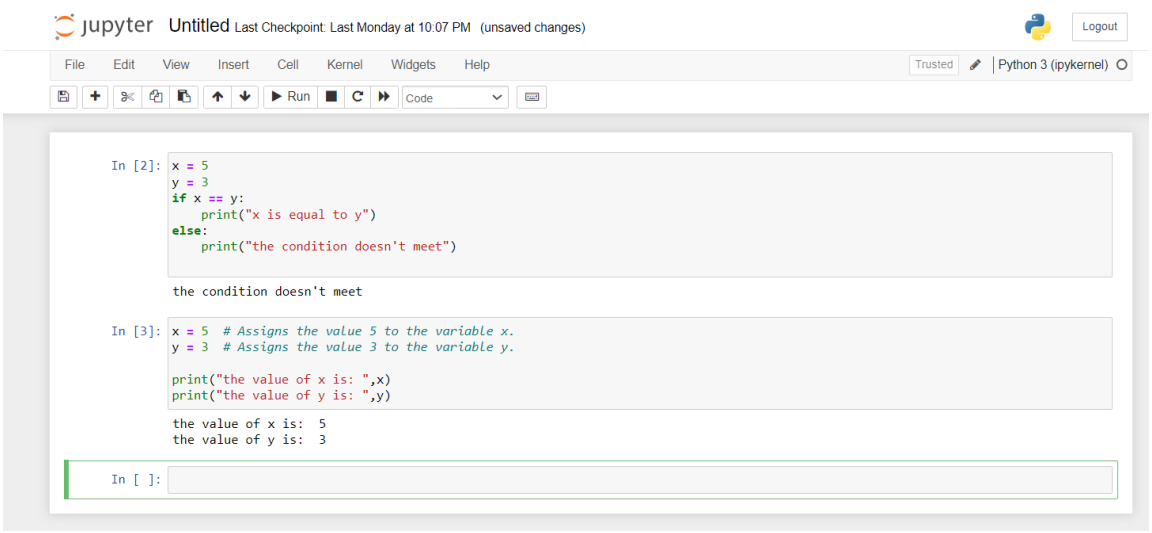
It returns a Boolean value (`true` if the values are equal, `false` otherwise).

2. Assignment Operator (=):

The assignment operator is used for assignment.

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

It does not perform a comparison; it performs an action.



7. Ans:

The three blocks of the programs are

1. spam = 0
   if spam == 10:
   print('eggs')

2. if spam > 5:
   print('bacon')
   else:
   print('ham')

3. print('spam')
   print('spam')

8. Ans:

```
In [6]: spam = int(input("Enter a number: "))
        if spam == 1:
            print("Hello")
        elif spam == 2:
            print("Howdy")
        else:
            print("Greetings!")

        Enter a number: 1
        Hello

In [ ]:
```

9. Ans:

We can usually press Ctrl + C in the command prompt or terminal where our program is running. This key combination sends an interrupt signal to the running process and often stops it.

10. Ans:

break:

The break statement is used to exit a loop prematurely.
When a break statement is encountered within a loop, it immediately terminates the loop, and the program continues with the next line of code after the loop.
It is often used to exit a loop when a certain condition is met or to stop processing further iterations.

continue:

The continue statement is used to skip the current iteration of a loop and move to the next iteration.
When a continue statement is encountered within a loop, it immediately stops the current iteration and goes to the next iteration of the loop.
It is often used when we want to skip certain elements or conditions in a loop but continue looping.

```
In [7]: for i in range(1, 6):
            if i == 3:
                break
            print(i)

1
2

In [8]: for i in range(1, 6):
            if i == 3:
                continue
            print(i)

1
2
4
5

In [ ]:
```

11. Ans:

In a `for` loop, the expressions `range(10)`, `range(0, 10)`, and `range(0, 10, 1)` are equivalent, and they will all generate the same sequence of numbers. The difference between them is mainly in the way they are written and the default values they assume:

1. range(10):

    This expression generates a sequence of numbers starting from 0 (inclusive) up to 9, but not including 10 by increments of 1 (the default increment).

    It is a concise way to generate numbers from 0 to 9.

2. range(0, 10):

    This expression explicitly specifies the start and stop values.

    It generates a sequence of numbers starting from 0 (inclusive) up to 9, but not including 10 by increments of 1 (the default increment).

3. range(0, 10, 1)

    This expression explicitly specifies the start, stop, and step values.

    It generates a sequence of numbers starting from 0 (inclusive) up to 9, but not including 10 by increments of 1 (explicitly specified).

12. Ans:

```
In [9]: for i in range(1, 11):
            print(i) #print 1-10 using for loop
```

```
1
2
3
4
5
6
7
8
9
10
```

```
In [10]: i = 1
         while i <= 10:
             print(i)
             i += 1
         #print 1-10 using while loop
```

```
1
2
3
4
5
6
7
8
9
10
```

13. Ans:

```
import spam

spam.bacon()
```