Assignment 2

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

Ans: The two values of the Boolean data type are:

- True - Written as “True”
- False - Written as “False”

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

Ans: The three different types of Boolean operators are:

1. AND - Typically written as “and”
2. OR - Typically written as “or”
3. NOT - Typically written as “not”

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”

A	B	A and B
True	True	True
True	False	False
False	True	False
False	False	False

- Or operator “or”

A	B	A or B
True	True	True
True	False	True
False	True	True
False	False	False

- Not operator “not”

A	not A
True	False
False	True

4. What are the values of the following expressions?

- (5 > 4) and (3 == 5): ***False***
- not (5 > 4): ***False***
- (5 > 4) or (3 == 5): ***True***

- not ((5 > 4) or (3 == 5)): ***<u>False</u>***
- (True and True) and (True == False): ***<u>False</u>***
- (not False) or (not True): ***<u>True</u>***

5. What are the six comparison operators?

Ans: The six comparison operators in Python 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. How do you tell the difference between the equal to and assignment operators? Describe a condition and when you would use one.

Ans: In Python, the equal to operator (`==`) and the assignment operator (`=`) serve different purposes:

- Equal to Operator (`==`)
  - Purpose: Used to compare two values to see if they are equal.
  - Example: 5 == 5 evaluates to `True` because both sides of the operator are equal.
  - Use: Use `==` when I want to check if two values or variables are the same in a condition or comparison.
  - Ex:

    ```
    if x == 10:
        print("x is 10")
    ```

    In this example, the condition checks if the value of `x` is equal to `10`.
- Assignment Operator (`=`)
  - Purpose: Used to assign a value to a variable.
  - Example: x = 5 assigns the value `5` to the variable `x`.
  - Use: Use `=` when you need to set or change the value of a variable.
  - Ex: `x = 10`

    In this example, `10` is assigned to the variable `x`.
- Condition and Usage
  - Equal to (`==`):
    - Scenario: Checking if a user's input matches a specific value.
      Example:

      ```
      user_input = input("Enter your age: ")
      if user_input == "18":
          print("You are 18 years old.")
      ```

  - Assignment (`=`):

- Scenario: Setting or updating the value of a variable.
  Example:
  ```
  age = 18
  if age == 18:
          print("You are 18 years old.")
  ```

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:

- Block 1:
  ```
  if spam == 10:
          Print(“eggs”)
  ```
- Block 2:
  ```
  if spam > 5:
          print(“bacon”)
  else:
          print(“ham”)
  ```
- Block 3:
  ```
  print(“spam”)
  print(“spam”)
  ```

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:

```
spam = int(input("Enter a value for spam: "))
if spam == 1:
        print('Hello')
elif spam == 2:
        print('Howdy')
```

```
else:
    print('Greetings!')
```

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

Ans: Press "**Ctrl + C**" in the terminal or command prompt where the program is running. This sends a Keyboard Interrupt signal to the Python interpreter, which will stop the program.

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

Ans:

- **break**
  Exits the current loop entirely and transfers control to the code following the loop.
  Use break when you want to terminate the loop based on a condition.
  Ex:
  ```
  for i in range(10):
      if i == 5:
          break
      print(i)
  ```
- **continue**
  Skips the rest of the code inside the current iteration of the loop and proceeds to the next iteration.
  Use continue when you want to skip specific iterations of the loop based on a condition but still want to continue looping.
  Ex:
  ```
  for i in range(10):
      if i == 5:
          continue
      print(i)
  ```

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

Ans:

- range(10)
  Generates numbers from 0 up to, but not including, 10
- range(0, 10)
  Generates numbers from 0 up to, but not including, 10. This is the same as range(10) with an explicit start value of 0
- range(0, 10, 1)
  Generates numbers from 0 up to, but not including, 10, with a step of 1. This explicitly specifies the start value, stop value, and step size.

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:

- For loop:
  ```
  for i in range(1, 11):
  	print(i)
  ```
- While loop:
  ```
  i = 1
  while i <= 10:
  	print(i)
  	i += 1
  ```

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

Ans:

```
import spam
spam.bacon()
```