Assignment-2:

1. Boolean Values and Writing:

- There are two Boolean values in Python:
  - `True`: Represents truth or a positive condition.
  - `False`: Represents falsity or a negative condition.

2. Boolean Operators:

- Python has three primary Boolean operators:
  - `and`: Performs a logical AND operation. Both operands must be True for the result to be True.
  - `or`: Performs a logical OR operation. At least one operand must be True for the result to be True.
  - `not`: Performs a logical NOT operation. Inverts the truth value of the operand.

3. Boolean Operator Truth Tables:

Operator	Operand 1	Operand 2	Result
`and`	True	True	True
	True	False	False
	False	True	False
	False	False	False
`or`	True	True	True
	True	False	True
	False	True	True
	False	False	False
`not`	True	-	False
	False	-	True

4. Boolean Expression Values:

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

5. Comparison Operators:

- Python provides six comparison operators:
  - `==`: Equal to
  - `!=`: Not equal to
  - `<`: Less than
  - `>`: Greater than
  - `<=`: Less than or equal to
  - `>=`: Greater than or equal to

6. Equal To vs. Assignment:

- `==`: Checks for equality between values (e.g., `x == 5` is True if x holds 5).
- `=`: Assigns a value to a variable (e.g., `x = 5` stores 5 in the variable x).

Conditions:

- Conditions are expressions that evaluate to True or False, often used in `if` statements to control program flow based on certain criteria.
- Example: `if x > 10: print("x is greater than 10")`. This statement checks if x is greater than 10, and if True, it executes the print statement.

7. Code Blocks:

- The code has three blocks:
  1. Assigns 0 to `spam`.
  2. `if spam == 10` block (not executed because spam is 0).
  3. `else` block (executed because the `if` condition is False). Prints "ham".
  4. Prints "spam" twice (these are outside any conditional block).

8.

```
if spam == 1:
    print("Hello")
elif spam == 2:
    print("Howdy")
else:
    print("Greetings!")
```

9. Escaping an Endless Loop:

- If your program is stuck in an endless loop, on most systems, you can press `Ctrl+C` to interrupt it.

10. Break vs. Continue:

- `break`: Exits the current loop completely
  (e.g.,
  ```
  for i in range(10):
        if i == 5: break;
              print(i)
  ```
  will print numbers from 0 to 4 and then exit).
- `continue`: Skips the current iteration of the loop and moves to the next one
  (e.g.,
  ```
  for i in range(10):
         if i % 2 == 0:
          continue;
               print(i)
  ```
  will print only odd numbers).

11. Difference between range(10), range(0, 10), and range(0, 10, 1) in a for loop:

- `range(10)`: This creates a sequence of numbers starting from 0 (inclusive) and going up to, but not including, 10. It's a shorthand for `range(0, 10)`.
- `range(0, 10)`: This explicitly defines the starting point (0) and the stopping point (10, non-inclusive). It's equivalent to `range(10)`.
- `range(0, 10, 1)`: This defines the start (0), stop (10), and step (1). Since the step is 1 by default in `range`, this is also the same as the previous two. Specifying 1 as the step is redundant here.

12. Printing numbers 1 to 10 with for and while loops:

- **for loop:**

```
for i in range(1, 11):
    print(i)
```

- **while loop:**

```
i = 1
while i <= 10:
 print(i)
  i += 1
```

13. Calling bacon() from spam module:

```
import spam

spam.bacon()
```