Assignment-2

1. Data type: Which Determine the type of the data is called as data type. There are mainly Five types of data types in python. There are:

* Integer
* Float
* String
* Complex
* Boolean
* Boolean:

Boolean is a data type which is used is decision making statements. In such as conditional statements.

Boolean statements mainly possible two types of values. There are ‘True’ and ‘Farlse’. Here, True and False are must be in capital letters. Because, Python case sensitive language. So, we must use capital letters in Boolean values.

Boolean values are manly used to represent logical states such as True/False or Yes/No conditions. In integer the values of True and False are 0 and 1.

Examples:

>>>a=int(input("Enter the value of a:"))

>>>b=int(input("Enter the value of b:"))

>>>a>b

Output:

Enter the value of a:55

Enter the value of b:13

True

>>>a=int(input("Enter the value of a:"))

>>>b=int(input("Enter the value of b:"))

>>>a>b

Output:

Enter the value of a:5

Enter the value of b:65

False

2)

In Python, the three different types of Boolean operators are:

1. AND Operator :

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

Example:

>>>True and True

True

>>>True and False

False

>>>False and True

False

>>>False and False

False

1. OR Operator:

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

Example:

>>>True or True

True

>>>True or False

True

>>>False or True

True

>>>False or False

False

1. NOT Operator:

The `not` operator is a unary operator that negates the value of its operand. It returns `True` if the operand is `False`, and `False` if the operand is `True`.

Example:

>>>not True

False

>>>not False

True

These operators are essential for creating logical expressions and making decisions based on conditions in Python programs. They are frequently used in constructs like `if` statements, loops, and more complex logical operations.

3)

AND operator:

|  |  |  |
| --- | --- | --- |
| A | B | A and B |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

OR operator:

|  |  |  |
| --- | --- | --- |
| A | B | A or B |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

NOT operator:

|  |  |
| --- | --- |
| A | not A |
| 0 | 1 |
| 1 | 0 |

4)

>>>(5>4) and (3==5) #True and False

False

>>>not (5>4) # not True

False

>>>(5>4) or (3==5) #True or False

True

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

False

>>>(True and True) and (True == False) # True and False

False

>>>(not False) or (not True) #True or False

True

5)

The six comparison operators in Python are:

1.Equal to (==):

Checks if the left operand is equal to the right operand.

Example:

>>>print(5==5)

True

2. Not equal to (!=):

Checks if the left operand is not equal to the right operand.

Example:

>>>print(5!=8)

True

1. Greater than (>):

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

Example:

>>>print(8>10)

False

1. Less than (<):

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

Example:  
>>>print(70<5)

False

5.Greater than or equal to (>=):

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

Example:

>>>print(10>=5)

True

1. Less than or equal to (<=):

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

Example:

>>>print(40<=8)

False

These operators are used to compare values and make decisions based on the outcomes of these comparisons in Python programs.

6)

Equal to (==): The equal to (`==`) operator is used for

comparison, to check if two values are equal. It evaluates to `True` if the values on both sides of the operator are equal, and `False` otherwise.

Assignment operator(=): The assignment operator (`=`) is used to assign a value to a variable. It takes the value on the right-hand side and assigns it to the variable on the left-hand side.

>>># Assignment Operator

>>>x = 10 # This assigns the value 10 to the variable x

>>>print(x)

10

>>># Equal To Operator

>>>y = 5

>>>result = (y == 5) # This checks if y is equal to 5. Since it is, result will be True.

>>>print(result)’

True

In this above example x=10 assigns the value 10 to the variable x. This means that x now hold the value 10. `(y == 5)` is a comparison using the equal to operator. It checks if the value of `y` is equal to `5`. Since `y` is indeed equal to `5`, the result of the comparison is `True`.

7)

In the provided code

if spam > 5:

print('bacon')

else:

print('ham')

print('spam')

print('spam')

the three blocks can be identified as follows:

Block 1:

>>>spam = 0

This is a single line of code that assigns the value `0` to the variable `spam`. It's not a block itself, but it's a standalone statement.

Block 2:

>>>if spam == 10:

>>> print('eggs')

This is an if statement block. It checks if the value of `spam` is equal to `10`. If that condition is true, it will execute the indented code block, which in this case is `print('eggs')`.

Block 3:

>>>if spam > 5:

>>> print('bacon')

>>>else:

>>> print('ham')

>>> print('spam')

>>> print('spam')

This is an if-else statement block. It first checks if `spam` is greater than `5`. If that condition is true, it will execute the indented code block which is `print('bacon')`. If the condition is false, it will execute the else block, which contains multiple print statements.

8)

Here's the Python code that:

>>>spam = 1

>>> # You can change the value of spam to test different case

>>>if spam == 1:

>>> print('Hello')

>>>elif spam == 2:

>>> print('Howdy')

>>>else:

>>> print('Greetings!')

Explanation:

- We first assign a value to the variable `spam`. You can change this value to test different cases.

- The `if` statement checks whether `spam` is equal to `1`. If this condition is true, it will print `'Hello'`.

- If the first condition is not true, the `elif` statement checks whether `spam` is equal to `2`. If this condition is true, it will print `'Howdy'`.

- If none of the conditions above are met (i.e., if `spam` is anything other than `1` or `2`), it will execute the `else` block, which prints `'Greetings!'`.

This code will print the appropriate greeting based on the value stored in `spam`.

9)

If your program is stuck in an endless loop and you want to interrupt it, you can typically press the keyboard shortcut to manually terminate the program. The most common keyboard shortcut to stop a program from running is:

- Ctrl + C on Windows and Linux systems

- In Jupyter notebook we use interrupt the kernal

This combination sends an interrupt signal to the running program, causing it to terminate abruptly.

10)

|  |
| --- |
|  |
| Break | Continue |
| i) )The break statement is used to exit a loop. Before it reaches its normal termination condition. | i)The continue statement is used to force the loop to start the next iteration, skipping the rest of the code in the loop block |
| ii) When break is encountered, it immediately exits the loop and continues with the next statement after the loop. | ii) When continue is encountered, it immediately moves to the next iteration of the loop, without executing any of the code below it in the loop block. |
| iii) You might use break when you've found what you were looking for in a loop and you want to stop further iterations. | iii) You might use continue when you want to skip a specific iteration based on a certain condition without exiting the loop entirely. |
| iii) for i in range(5):  if i == 3:  break  print(i) | iii) for i in range(5):  if i == 2:  continue  print(i) |
| 0  1  2 | 0  1  2  3  4 |
|  |  |

11)

In a for loop, the expressions range(10), range(0, 10), and range(0, 10, 1) are functionally equivalent. They all generate a sequence of numbers from 0 up to10, incrementing by 1.

a. range(10):

- This generates a sequence of numbers starting from 0 up to 10.But here without including the last number 10.

- By default, if you provide a single argument to the range function, it assumes it's the end value, starting from 0, and increments by 1.

- So, range(10) is equivalent to range(0, 10, 1).

Example:

>>>for i in range(10):

>>> print(i)

Output:

0

1

2

3

4

5

6

7

8

9

b. range(0, 10):

- This explicitly specifies both the start and end values of the sequence.

- It starts from 0 and goes up to (but not including) 10, incrementing by 1.

Example:

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

>>> print(i)

Output :

0

1

2

3

4

5

6

7

8

9

c. range(0, 10, 1):

- This is explicitly specifying the start value (0), end value (10), and step size (1).

- The step size determines how the sequence progresses from one element to the next.

Example (:

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

>>> print(i)

Output :

0

1

2

3

4

5

6

7

8

9

```

using `range(10)`, `range(0, 10)`, or `range(0, 10, 1)` will produce the same sequence of numbers from 0 to 9.

12)

Using a For Loop:

>>># Using a for loop

>>>for i in range(1, 11):

>>> print(i)

Output:

1

2

3

4

5

6

7

8

9

10

Using a While Loop:

>>># Using a while loop

>>>i = 1

>>>while i <= 10:

>>> print(i)

>>> i += 1

Output:

1

2

3

4

5

6

7

8

9

10

Variable I is set to 1, and then use a loop to print the value of i and increment it by 1 in each iteration. The loop continues as long as i is less than or equal to 10.

13)

After importing the module spam, you would call the function bacon() using the following syntax:

import spam

# Call the function

spam.bacon()

This assumes that bacon() is a function defined inside the spam module. The spam. prefix specifies that you're referring to the bacon() function within the spam module.