Python Basics

Assignment_2

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

Ans:- The Boolean data type in Python represents a logical value, which can be either True or False.

To write a Boolean value in Python, you can simply use the True or False keyword. For example:

```
is_raining = True
is_sunny = False
```

These statements create two variables, is_raining and is_sunny, and assign them the values True and False, respectively.
Boolean values are often used in control statements to determine whether a certain condition is met. For example:

```
if is_raining:
    print('Bring an umbrella!')
else:
    print('No umbrella needed.')
```

This code will print 'Bring an umbrella!' if the is_raining variable is True, and 'No umbrella needed.' if it is False.

That's all you need to know about the Boolean data type in Python! It's a simple but powerful data type that is used in many different types of program.

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

Ans:- In Python, there are three different types of Boolean operators:

1 -> Logical AND (and): This operator returns True if both of the operands are True, and False otherwise. For example:

```
True and True  # True
True and False  # False
False and True  # False
False and False  # False
```

2 -> Logical OR (or): This operator returns True if at least one of the operands is True, and False otherwise. For example:

```
True or True  # True
True or False  # True
False or True  # True
False or False  # False
```

3 -> Logical NOT (not): This operator returns the opposite of the operand. If the operand is True, it returns False, and if the operand is False, it returns True. For example:

```
not True  # False
not False  # True
```

3. Make a list of each Boolean operator truth tables (i.e. every possible combination of Boolean values for the operator and what it evaluate).

Ans :- Logical AND (and):

Operand	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

4. What are the values of the following expressions?
(5 > 4) and (3 == 5)
not (5 > 4)
(5 > 4) or (3 == 5)
not ((5 > 4) or (3 == 5))
(True and True) and (True == False)
(not False) or (not True)

Ans :-

In [1]:

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

```
False
False
True
True
False
True
```

5. What are the six comparison operators?

Ans :-
1 Equal to (==): This operator returns True if the operands are equal, and False otherwise.
2 Not equal to (!=): This operator returns True if the operands are not equal, and False otherwise.

3 Greater than (>): This operator returns True if the left operand is greater than the right operand, and False otherwise.
4 Less than (<): This operator returns True if the left operand is less than the right operand, and False otherwise.
5 In Python, greater than or equal (>=): operator is a comparison operator that returns True if the left operand is greater than or equal to the right operand, and False otherwise.
6 In Python, less than or equal to (<=), the <= operator is a comparison operator that returns True if the left operand is less than or equal to the right operand, and False otherwise.

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 (==) is used to compare the values of two operands, while the assignment operator (=) is used to assign a value to a variable.

The equal to operator returns True if the operands are equal, and False otherwise. For example:

```
x = 5
y = 10
print(x == y)  # False

x = 10
y = 10
print(x == y)  # True
```

The assignment operator assigns a value to a variable. For example:

```
x = 5
y = 10
x = y
print(x)  # 10
```

It's important to remember that the equal to operator is used for comparison, while the assignment operator is used for assignment.
You should use the equal to operator when you want to compare two values, and the assignment operator when you want to assign a value to a variable.

Here's a simple example of when you might use the equal to operator:

```
if x == y:
    print('x and y are equal')
else:
    print('x and y are not equal')
```

This code will print 'x and y are equal' if x and y are equal, and 'x and y are not equal' otherwise.

7. Identify the three block in the code.

Ans :-

In [2]:

```
spam = 0
if spam == 10:
  print('eggs') # Block 1
  if spam > 5:
    print('bacon') # Block 2
  else:
    print('ham') # Block 3
  print('spam')
print('spam')
```

```
spam
```

```
1 The first block is the first if statement, which consists of the line if spam == 10:
and the line print('eggs').
This block will be executed if the condition spam == 10 is True.
2 The second block is the second if statement, which consists of the line if spam > 5:
and the line print('bacon').
This block will be executed if the condition spam > 5 is True.
3 The third block is the else clause, which consists of the line else: and the line
print('ham').
This block will be executed if the condition in the second if statement is False.

The code also contains two additional lines that are not part of any block: print('spam')
and print('spam').
These lines will be executed regardless of the values of spam or the results of the if
statements.
```

```
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 :-
```

In [3]:

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

```
Greetings!
```

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

Ans: - Ctrl + C
```

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

Ans: - The main difference between both the statements is that when break keyword comes,
it terminates the execution of the current loop and passes the control over the next loop
or main body, whereas when continue keyword is encountered, it skips the current
iteration and executes the very next iteration in the loop.

break: The break statement is used to exit a loop prematurely.
```

When a break statement is encountered inside a loop, the loop is terminated immediately, and the program continues with the next line of code after the loop.

continue: The continue statement is used to skip the rest of the current iteration of a loop and move on to the next one.
When a continue statement is encountered inside a loop,
the program skips the remaining statements in the current iteration and goes back to the top of the loop to start the next iteration.

Here's an example of how you might use break and continue in a loop:

```
for i in range(10):
    if i == 5:
        break
    print(i)
```

This code will print the numbers 0 through 4, and then exit the loop when it reaches the break statement at i == 5.

```
for i in range(10):
    if i % 2 == 0:
        continue
    print(i)
```

This code will print the odd numbers 1 through 9, skipping the even numbers when it reaches the continue statement at i % 2 == 0.

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

Ans:-
In Python, the range() function is used to generate a sequence of numbers. It takes three arguments: start, stop, and step.

start: The starting value of the sequence. This is optional, and the default value is 0.
stop: The ending value of the sequence (exclusive). This is the required argument.
step: The amount by which the values in the sequence should be incremented. This is optional, and the default value is 1.

Here's an example of how you might use range() in a for loop:

```
for i in range(10):
    print(i)
```
This code will print the numbers 0 through 9. The range() function generates a sequence of numbers from 0 to 9 (exclusive), with a step size of 1.
```
for i in range(0, 10):
    print(i)
```
This code will also print the numbers 0 through 9. Here, the range() function generates a sequence of numbers from 0 to 10 (exclusive), with a step size of 1.
```
for i in range(0, 10, 1):
    print(i)
```
This code will also print the numbers 0 through 9. Here, the range() function generates a sequence of numbers from 0 to 10 (exclusive), with a step size of 1.

In all three examples, the range() function generates a sequence of numbers from 0 to 9 (exclusive), with a step size of 1.
The only difference is in the way the arguments are specified.

I hope this helps! Let me know if you have any other questions.

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 :-

In [1]:

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

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

In [2]:

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

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

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

Ans :-

```
from spam import bacon
bacon()

or
spam.bacon()
```