Assignment No. 2

Submitted By: Sachin Dodake

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

`Ans-1`

- Python boolean type is one of the built-in data types provided by Python, which represents one of the two values i.e. `True` or `False`.
- It is used to represent the truth values of the expressions. For example, `1==1` is `True` whereas `2<1` is `False`.
- The Python Boolean type has only two possible values:

```
i)  True
ii)  False
```

- The first letter of these boolean data types must be a capital.

Few observations of Boolean Dat Types:

i) Most Values are True:

- Almost any value is evaluated to True if it has some sort of content.
- Any string is True, except empty strings.
- Any number is True, except 0.
- Any list, tuple, set, and dictionary are True, except empty ones.
- **Examples:**

```
i) bool("abc") -----------------------------------> True.
ii) bool(123) ------------------------------------> True.
iii) bool(["apple", "cherry", "banana"]) ---> True.
```

ii) Some Values are False:

- In fact, there are not many values that evaluate to False, except empty values, such as (), [], {}, "", the number 0, and the value None. And of course the value False evaluates to False.
- **Examples:**

```
i) bool(False) -----------------------------------> False.
ii) bool(None) -----------------------------------> False.
iii) bool(0) -----------------------------------> False.
iv) bool("") -----------------------------------> False.
v) bool(()) -----------------------------------> False.
vi) bool([]) -----------------------------------> False.
vii) bool({}) -----------------------------------> False.
```

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

`Ans-2`

- **Definition:** Boolean operators are those that take Boolean inputs and return Boolean results.
- Boolean operators are used in a boolean expression to return boolean values.
- Secondly, Boolean operators can compress multiple if-else boolean expressions into one single line of code.
- Lastly, there are three types of python boolean operators:

```
i) `and` operator (Conjunction);
ii) `or` operator (Disjunction);
iii) `not` operator (Negation).
```

`i) and-operator:`

- The `AND` boolean operator is similar to the `bitwise AND` operator where the operator analyzes the expressions written on both sides and returns the output.

| Operand-1| AND | Operand-2 | Result | | :---: | :---: | :---: |:---: | | True | and | True | True | | True | and | False | False | | False | and | False | True | | False | and | True | True |

`ii) or-operator:`

- The `OR` operator is similar to the `bitwise OR` operator. In the `bitwise OR`, we were focussing on either of the bit being 1. Here, we take into account if either of the expression is true or not. If at least one expression is true, consequently, the result is true.

| Operand-1| OR | Operand-2 | Result | | :---: | :---: | :---: |:---: | | True | or | True | True | | True | or | False | True | | False | or | True | True | | False | or | False | False |

`iii) not-operator:`

- The `NOT` operator reverses the result of the boolean expression that follows the operator. It is important to note that the `NOT` operator will only reverse the final result of the expression that immediately follows. Moreover, the `NOT` operator is denoted by the keyword `not`.

| NOT | Operand | Result | | :---: | :---: |:---: | | not | True | False | | not | False | True|

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-3`

- The truth table for various Boolean Operators are as below:

`i) AND Truth Table:`

- The truth table for boolean 'AND' is as below:

| Operand-1| AND | Operand-2 | Result | | :---: | :---: | :---: |:---: | | True | and | True | True | | True | and | False | False | | False | and | False | True | | False | and | True | True |

`ii) OR Truth Table:`

- The truth table for boolean 'OR' is as below:

| Operand-1| OR | Operand-2 | Result | | :---: | :---: | :---: |:---: | | True | or | True | True | | True | or | False | True | | False | or | True | True | | False | or | False | False |

`iii) NOT Truth Table:`

- The truth table for boolean 'NOT' is as below:

| NOT | Operand | Result | | :---: | :---: |:---: | | not | True | False | | not | False | True |

4. What are the values of the following expressions?

```
i ) (5 > 4) and (3 == 5)
ii) not (5 > 4)
iii) (5 > 4) or (3 == 5)
iv) not ((5 > 4) or (3 == 5))
v) (True and True) and (True == False)
vi) (not False) or (not True)
```

`Ans-4:`

i) (5 > 4) and (3 == 5) =====> `False`.
ii) not (5 > 4) =====> `False`.
iii) (5 > 4) or (3 == 5) =====> `True`
iv) not ((5 > 4) or (3 == 5)) =====> `False`
v) (True and True) and (True == False) =====> `False`
vi) (not False) or (not True) =====> `True`.

5. What are the six `comparison operators`?

- A comparison operator compares two values and returns a `boolean value`, either `True` or `False`.
- We can use these comparison operators to compare both `numbers` and `strings`.
- Python has six comparison operators, which are as follows:

```
i) less than (<),
ii) less than or equal to (<=),
iii) greater than (>),
iv) greater than or equal to (>=),
v) equal to (==),
vi) not equal to (!=).
```

`i) Less than operator (<):`

- The Less Than operator (<) compares two values and returns True if the value on the left is less than the value on the right. Otherwise, it returns False:
- Syntax:

```
left_value  < right_value
```

- Example:

```
i) 10 < 20 =====> True   #
ii) 30 < 20 =====> False  #
iii) apple < orange  =====>True   # True because the letter a in apple is before the letter o in orange
 iv) banana < apple =====> False   # False because the letter 'b' is after the letter 'a'
```

`ii) Less than or equal to operator (<=):`

- The less than or equal to operator compares two values and returns True if the left value is less than or equal to the right value. Otherwise, it returns False:
- Syntax:

```
left_value <= right_value
```

- Example:

```
i) 20 <= 20 =====> True
ii) 10 <= 20 =====> True
iii) 30 <= 30 =====> True
```

`iii) Greater than operator (>):`

- The greater than operator (>) compares two values and returns True if the left value is greater than the right value. Otherwise, it returns False
- Syntax:

```
left_value > right_value
```

- Example:

```
i) 20 > 10 =====> True
ii) 20 > 20 =====> False
iii) apple > orange =====> False
iv) orange > apple =====> True
```

`iv) Greater Than or Equal To operator (>=):`

- The greater than or equal to operator (>=) compares two values and returns True if the left value is greater than or equal to the right value. Otherwise, it returns False.
- Syntax:

```
left_value >= right_value
```

- Example:

```
i) 20 >= 10 =====> True
ii) 20 >= 20 =====> True
iii) 10 >= 20 =====> False
```

`v) Equal To operator (==):`

- The equal to operator (==) compares two values and returns True if the left value is equal to the right value. Otherwise, it returns False
- Syntax:

```
left_value == right_value
```

- Example:

```
i) 20 == 10 =====> False
ii) 20 == 20 =====> True
```

`vi) Not Equal To operator (!=):`

- The not equal to operator (!=) compares two values and returns True if the left value isn't equal to the right value. Otherwise, it returns False.
- Syntax:

```
left_value != right_value
```

- Example:

```
i) 20 != 20 =====> False
ii) 20 != 10 =====> True
```

6. How do you tell the difference between the `equal to (==)` and `assignment (=)` operators?Describe a condition and when you would use one.

`Ans-6`

`Equal To Operator (==):`

- Two consecutive equal marks is used to check whether 2 expressions give the same value .
- It is used for comparing two values. It returns 1 if both the values are equal otherwise returns 0.
- Constant term can be placed in the left hand side. Example: 1==1 is valid and returns 1.
- Example:

```
i) 1212 == 2323 =====> False
ii) 1111 == 2222 =====> False
```

`Assignment Operator (=):`

- A single equal mark is used to assign a value to a variable.
- It is used for assigning the value to a variable.
- Constant term cannot be placed on left hand side. Example: 1=x; is invalid.
- Example:

```
i) a = 55       # assign value
    b = 22       # assign the expression to the left operand
```

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-7`

- Indentation is used to define a `block of code` in python.
- The `Braces {}` are used to define a block of code in most programming languages, like C, C++, and Java. But this `indentation` makes python unique among all programming languages. This `indentation` highlights the `block of code`.
- In Python, indentation is done with whitespace. All statements with the same right-hand distance belong to the same code block. If a block needs to be more nested, it is indented to the right.
- Indentation is only used in most other programming languages to help make the code look nice. However, it is required in Python to indicate which block of code a statement belongs to.
- Few important points:

```
i) A block begins when the indentation increases.
ii) A block ends when the indentation decreases.
iii) Multiple blocks can be closed in the same decrease
```

- The blocks in the given code are mentioned below

In [73]:
```
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
```

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-8`

In [78]:
```
#  taking number from user
spam = eval(input("Enter the Number :  "))

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

```
Enter the Number :  1
Hello
```

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

`Ans-9`

- An endless loop occurs when a program keeps executing within one loop, never leaving it.
- To exit out of infinite loops on the command line, press `CTRL + C`.

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

`Ans-10`

`break:`

- A break statement in Python alters the flow of a loop by terminating it once a specified condition is met.
- The break statement in Python terminates the loop containing it.
- A break statement is used to terminate the loop whenever a particular condition is satisfied.
- The statement is there just next after the loop receives control of the program.
- The break statement will end the innermost loop if it is contained within a nested loop that is the loop inside the other loop.
- It is used to end the loop that it is enclosed in, such as a do-while, while, switch, and for statement.

In [79]:
```
# example of break statement

for num in range(0,10):
    if num == 5:
        break
    print(f'Iteration: {num}')
```

```
Iteration: 0
Iteration: 1
Iteration: 2
Iteration: 3
Iteration: 4
```

`continue:`

- The continue statement in Python is used to skip the remaining code inside a loop for the current iteration only.
- The continue statement is used to skip the remaining code inside a loop for the current iteration only.
- The continue statement skips the remaining lines of code, for the current iteration of the loop.
- In this case, the loop does not end, it continues with the next iteration.

In [80]:
```
# example of continue statement

for num in range(0,10):
    if num == 5:
        continue
    print(f'Iteration: {num}')
```

```
Iteration: 0
Iteration: 1
Iteration: 2
Iteration: 3
Iteration: 4
Iteration: 6
Iteration: 7
Iteration: 8
Iteration: 9
```

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

`Ans-11`

`range() :`

- The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and stops before a specified number.
- Syntax:

```
range(start, stop, step)
```

- Parameters:

```
        start - Optional. An integer number specifying at which position to start. Default is 0
        stop -  Required. An integer number specifying at which position to stop (not included).
        step - Optional. An integer number specifying the incrementation. Default is 1
```

`Expalination:`

- There is no difference between `range(10)`, `range(1,10)`, and `range(0,10,1)` because all these represents the ame output value.
- The output of all these are shown below:

In [81]:
```
# using python range(10) function
for i in range(10):
    print( i, end=" ")
```

```
0 1 2 3 4 5 6 7 8 9
```

In [82]:
```
# using python range(0,10) function
for i in range(0,10):
    print(i, end=" ")
```

```
0 1 2 3 4 5 6 7 8 9
```

In [83]:
```
# using python range(0,10,1) function
for i in range(0,10,1):
    print(i, end=" ")
```

```
0 1 2 3 4 5 6 7 8 9
```

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-12`

`Using for loop`

In [84]:
```
for num in range(1, 11):    #  looping i in range from 1 to 10
    print(num)                         # printing value of i
```

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

`Using while loop`

In [85]:
```
num = 1                           # defining variable num with value 1
while num<=10:        #  applying while loop with condition
    print(num)              # printing number
    num+=1                # increamenting number to get next number
```

```
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 13

```
In [ ]: # importing "spam" module
        import spam

        # calling bacon function from spam module
        spam.bacon()
```

```
In [ ]:
```