Python Fundamentals Part I

Workbook

Table of Contents

1.Course Objectives

After completing this course, students will be able to:

- Learn how Python works and what it is good for
- Understand Python's place in the world of programming languages
- Learn to work with and manipulate strings in Python
- Learn to perform math operations with Python
- Learn to work with Python sequences: lists, arrays, dictionaries, and sets
- Learn to collect user input and output results
- Learn flow control processing in Python
- Learn to write to and read from files using Python
- Learn to write functions in Python
- Learn to handle exceptions in Python
- Learn to work with dates and times in Python

2.Course Content

Module 01: Python Introduction

I. What is Python
   a. Brief history
   b. Why Python
   c. Installation
   d. The first program "Hello world"

II. What are Variables and how are they used in programming
   a. Intro to Python Variables
   b. Process flow of variables
   c. Variables Declaration
   d. Naming Requirements
   e. Variable Casting
   f. Naming Conventions

III. What are Data types and how do they relate to variable
   a) Introduction to Python Data Types
   b) Mutable vs Immutable

IV. Understand what is Boolean Logic
   a) Introduction to Boolean Operators in Python
   b) Comparison Operators
   c) Binary Boolean Operators
   d) Not Operator
   e) Combination of Binary Boolean and Comparison Operators
   f) Conclusion

3.What is Python

i) Brief history

The programming language Python was conceived in the late 1980s, and its implementation was started in December 1989 by Guido van Rossum at CWI in the Netherlands as a successor to ABC capable of exception handling and interfacing with the Amoeba operating system. Van Rossum is Python's principal author, and his continuing central role in deciding the direction of Python is reflected in the title given to him by the Python community, Benevolent Dictator for Life (BDFL). (However, van Rossum stepped down as leader on July 12, 2018.). Python was named after the BBC TV show Monty Python's Flying Circus.

Python 2.0 was released on October 16, 2000, with many major new features, including a cycle-detecting garbage collector (in addition to reference counting) for memory management and support for Unicode. However, the most important change was to the development process itself, with a shift to a more transparent and community-backed process.

Python 3.0, a major, backward-incompatible release, was released on December 3, 2008, after a long period of testing. Many of its major features have also been backported to the backward-compatible, while by now unsupported, Python 2.6 and 2.7.

ii) Why Python

Every year I consider whether to continue using Python or whether to move on to a different language—perhaps one that's newer to the programming world. But I continue to focus on Python for many reasons:

i. Python is an incredibly efficient language
ii. Python's syntax will also help you write "clean" code
iii. Your code will be easy to read
iv. Easy to debug
v. Easy to extend and build upon

People use Python for many purposes:
I. To make games
II. Build web applications
III. Solve business problems
IV. And develop internal tools at all kinds of interesting companies.
V. Python is also used heavily in scientific fields for academic research and applied for work.

One of the most important reasons I continue to use Python is because of the Python community, which includes an incredibly diverse and welcoming group of people. Community is essential to programmers because programming isn't a solitary pursuit. Most of us, even the most experienced programmers, need to ask advice from others who have already solved similar problems. Having a well-connected and supportive community is critical in helping you solve problems, and the Python community is fully supportive of people like you who are learning Python as your first programming language. Python is a great language to learn, so let's get started!

iii) Installation

a) Setting up your Programming environment

Python differs slightly on different operating systems, so you'll need to keep a few considerations in mind. Here, we'll look at the two major versions of Python currently in use and outline the steps to set up Python on your system.

b) Python 2 and Python 3

Today, two versions of Python are available: Python 2 and the newer Python 3. Every programming language evolves as new ideas and technologies emerge, and the developers of Python have continually made the language more versatile and powerful. Most changes are incremental and hardly noticeable, but in some cases code written for Python 2 may not run properly on systems with Python 3 installed.

If both versions are installed on your system or if you need to install Python, use Python 3. If Python 2 is the only version on your system and you'd rather jump into writing code instead of installing Python, you can start with Python 2. But the sooner you upgrade to using Python 3 the better, so you'll be working with the most recent version 3.9*.

c) Hello World!

A long-held belief in the programming world has been that printing a *Hello world!* message to the screen as your first program in a new language will bring you luck.
In Python, you can write the Hello World program in one line:

d) Example 01

```
print("Hello world!")
```

Such a simple program serves a very real purpose. If it runs correctly on your system, any Python program you write should work as well. We'll look at writing this program with PyCharm in just a moment.

e) Installing Python and PyCharm

Follow the guide that was sent to you upon joining this course or access google classroom.

iv) The first program "Hello world"

Before you start make sure that the following prerequisites are met:

1. You are working with PyCharm Community
2. You have installed Python itself. If you're using macOS or Linux, your computer already has Python installed.

a) Creating a Python project

I. Let’s start our project: if you’re on the Welcome screen, click New Project. If you’ve already got a project open, choose File | New Project

II. Choose the project location. To do that, click the Browse button next to the Location field, and specify the directory for your project. (This is optional the default is still fine)

III. Also, deselect the Create a main.py welcome script checkbox because you will create a new Python file for this tutorial

IV. Python's best practice is to create a virtualenv for each project. To do that, expand the Python Interpreter: New Virtualenv Environment node and select a tool used to create a new virtual environment. Let's choose the Virtualenv tool, and specify the location and base interpreter used for the new virtual environment. Select the two checkboxes below if necessary

V. When configuring the base interpreter, you need to specify the path to the Python executable. If PyCharm detects no Python on your machine, it provides two options: to download the latest Python versions from python.org or to specify a path to the Python executable (in case of non-standard installation)

VI. Then click the Create button at the bottom of the New Project dialog

VII. If you’ve already got a project open, after clicking Create PyCharm will ask you whether to open a new project in the current window or a new one. Choose Open in the current window - this will close the current project,

b) Creating a Python file

I. Select the project root in the Project tool window, then select File | New ... from the main menu or press Alt+Insert.

II. Choose the option Python file from the popup, and then type the new filename “Helloworld” and press enter.

III. PyCharm creates a new Python file and opens it for editing.

c) Editing source code

I. Enter the following line
   a. print("Hello world!")

II. Right-click “Example 01.py” and select “Run” to execute the code

III. If code is executed correctly the words “Hello World!” should be printed in the terminal below.

Note: A video showing the above steps is accessible in the classroom.

4.What are Variables and how are they used in programming

i) Intro to Python Variables

A *Variable* is nothing but a programming element that is used for defining, storing, and performing operations on the input data.
Let's try using a variable in Example 02.py. Add a new line at the beginning of the file, and modify the second line

a) Example 02

```
message = "Hello world!"
print(message)
```

Run this program (Ctrl+Shift+F10 or Right-click the tab and select "Run") to see what happens.
You should see the same output you saw previously:

```
Hello world!
```

We've added a variable named ***message***. Every variable holds a value, which is the information associated with that variable. In this case, the value is the text ***"Hello world!"***
Adding a variable makes a little more work for the Python interpreter. When it processes the first line, it associates the text ***"Hello world!"*** with the variable ***message***. When it reaches the second line, it prints the value associated with the ***message*** to the screen.

Let's expand on this program by modifying Example 03.py to print a second ***message***.
Add a blank line to HelloWorld, and then add two new lines of code:

b) Example 03

```
message = "Hello world!"
print(message)

message = "Hello Python Fundamentals Course world!"
print(message)
```

Now when you run Example 03.py, you should see two lines of output:

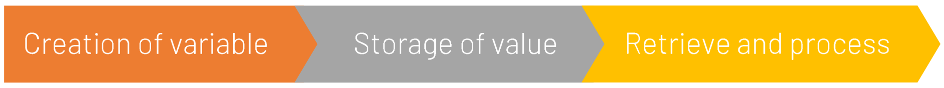
```
Hello world!
Hello Python Fundamentals Course world!
```

You can change the value of a variable in your program at any time, and Python will always keep track of its current value.

ii) Process flow of variables

Python variables are of four different types, and they are **Integer**, **Long Integer**, **Float**, and **String**. Integers are used for defining numeric values, Long Integers are used for defining integers with bigger lengths than a normal Integer, Floats are used for defining decimal values, and Strings are used for defining characters.

The process flow of variables can be defined as below:



Like every other programming language variables play a critical role in python too. let us discuss in detail python programming language variables.

iii) Variables Declaration

Like other programming, languages python does not expect a static variable declaration along with its type of the variable being handled. Python can determine the type of the variable just based on the type of value being stored in it.

Create a new file in PyCharm and enter follow Example 04 and see if you can get the same output.

a) Example 04:

```
ten = 10
twenty = 20
thirty = 30
forty = 40
fifty = 50

print(f"Total numeric value : {ten + twenty + thirty + forty + fifty} ")
```

Output

```
Total numeric value: 150
```

b) Example 05:

```
ten = " 10 "
twenty = " 20 "
thirty = " 30 "
forty = " 40 "
fifty = " 50 "
print(f"Total text value: {ten} + {twenty} + {thirty} + {forty} + {fifty}")
```

Output

```
Total text value: 10 + 20 + 30 + 40 + 50
```

Explanation

The above program shows the addition of values with a difference of ten up to fifty. each value is stored in a different variable. the significance is we can notice the process of operator overloading coming into play, in the first set the variables are stored with static numeric values whereas in the second set the numeric values are stored within double quotes which make them a string value, this leads the value to be an addition in the first set whereas in the second set it turns out to be a concatenation of the strings involved.

iv) Naming Requirements

1. Variables in Python are required to start with a letter or an underscore.
2. The rest of the name must consist of letters, numbers, or underscores
3. Names are case-sensitive

a) Example 06

```
Valid:
_cats = "meow"
abc123 = 1970

Invalid:
2cats = 1  # SyntaxError
hey@you = 1  # SyntaxError
```

b) Naming Conventions

a) Most variable names should be written in lower_snake_case. This means that all words should be lowercase, and words should be separated by an underscore.
b) CAPITAL_SNAKE_CASE usually refers to constants
c) UpperCamelCase usually refers to a class (more on that later)
d) Variables that start and end with two underscores (called "dunder" for double underscore) are intended to be private or are builtins to the language

c) Example 07

```
myVariable = 3  # Get outta here, this isn't JavaScript!
my_variable = 3  # much better
AVOGADRO_CONSTANT = 6.022140857 * 10 ** 23 #https://en.wikipedia.org/wiki/Avogadro_constant
__no_touchy__ = 3  # someone doesn't want you to mess with this!
```

v) Variable Casting

Variable casting is the process of converting a variable from one type to the other. for achieving this in python the casting functions are in place. the casting functions take the responsibility for the conversion of the variables from their actual type to the other format.

I. Type int(x) to convert x to a plain integer.
II. Type long(x) to convert x to a long integer.
III. Type float(x) to convert x to a floating-point number.

a) Example 08

```
ten = 10
twenty = 20
thirty = 30
forty = 40
fifty = 50

total = ten + twenty + thirty + forty + fifty
print(f"Total numeric value : {total}", f"Variable Type :{type(total)}")
```

Output

```
Total numeric value : 150 Variable Type :<class 'int'>
```

b) Example 09

```
ten = str(10)
twenty = str(20)
thirty = str(30)
forty = str(40)
fifty = str(50)

total = ten + twenty + thirty + forty + fifty
print(f"Total text value : {total}", f"Variable Type :{type(total)}")
```

Output

```
Total text value : 1020304050 Variable Type :<class 'str'>
```

c) Example 10

```
ten = float(10)
twenty = float(20)
thirty = float(30)
forty = float(40)
fifty = float(50)

total = ten + twenty + thirty + forty + fifty
print(f"Total numeric value : {total}", f"Variable Type :{type(total)}")
```

Output

```
Total numeric value : 150.0 Variable Type :<class 'float'>
```

The above program shows the addition of values with a difference of ten up to fifty. each value is stored in a different variable. here unlike the first program, the subsequent variables are typecast and the results of the casted values are printed along with their type. we can notice how the process of typecasting converts a variable of integer type to string and then to float.

vi) Conclusion

Like any other programming languages, the concept of variables plays a very significant role in python too, the classified number of functionalities and flexibility in coding them make variables more precise entities for access in the python programming language.

Exercise 01

Please submit screenshots with your answers and upload them to the classroom.
The due date of exercise is 11 Feb at noon solutions to the exercise will be shared at the session coming on the 13 Feb 2021 at 9 am

1. Declare a new variable with the string value "Yellow" and print the value to the console.

2. Declare a new variable called color and assign the value "red" to it. Then, print its value on the console.

3. Store the result of multiplying 2345 times 7323 in a variable called variables_are_cool. Now print the result in the console.

4. Set the values for my_var1 and my_var2 so the code prints 'Hello World' in the console.

5. What are Data types and how do they relate to variable

i) Introduction to Python Data Types

Data types are nothing but the different types of input data accepted by a programming language, for defining, declaring, storing, and performing mathematical & logical values/ operations. In Python, there are several data types used for dealing with the general operations on the input data given by the program developer. A few of the commonly used data types are **Numbers** for numeric values, **String** for single or series of characters, **Tuple** for a combination of different data types, **List** for a collection of values, etc.

a) Example 11

```
var1 = 20
var2 = 20.65
var3 = "Hello!, World"

print( type(var1) )
print( type(var2) )
print( type(var3) )
```

Output

```
<class 'int'>
<class 'float'>
<class 'str'>
```

b) The standard data types of python are given below :

I. Numbers: The Number data type is used to stores numeric values.
II. String: String data type is used to stores the sequence of characters.
III. Tuple: Tuple data type is used to stores a collection of different data types of elements and it is immutable.
IV. List: List data type is used to stores the collection of different data types of elements and it is mutable.
V. Set: Set data type is used to stores the collection of different data types of elements; it is mutable and store unique elements.
VI. Dictionary: Dictionary data type is used to stores a collection of different data types of elements in the form of key-value pairs, it is mutable and stores the unique key.

c) Numbers

When a number is assigned to a variable a Number class object is created.
Consider an example: var a = 100, var b = 200
var a and var b numbers are assigned and these are objects of number.
The Number can have 4 types of numeric data:

I. int : int stores integers eg a=100, b=25, c=526, etc.
II. long: long stores higher range of integers eg a=908090999L, b=-0x1990999L, etc.
III. float: float stores floating-point numbers eg a=25.6, b=45.90, c=1.290, etc.
IV. complex: complex stores numbers eg a=3 + 4j, b=2 + 3j, c=complex(4,6), etc.

d) String

The string can be defined as the sequence of characters represented in the quotation marks. In python, the string can be quoted by single, double, or triple quotes. In python, there are various inbuilt operators and functions available to easily work with the string data type. The following example shows the string handling with inbuilt operators and functions

e) Example 12

```
s = 'hello! how are you'

print (s[1])

print (s[2:6])

print(s*3)

s1 = 'hello world'
print (s + s1)
```

Output

```
e
llo!
hello! how are youhello! how are youhello! how are you
hello! how are youhello world
```

f) Tuple

Tuples also store the collection of the elements of different data types. A tuple is the same as the list, but a tuple is immutable (non-editable or cannot modify the size and elements value). To create a tuple uses the () simple parenthesis, within this bracket stores all the elements separated with the comma (,).

The following example shows the tuple handling:

g) Example 13

```
tp  = ("apple", "a", 100, 20.78)
print (tp[1])
print (tp[1:])
print (tp[:3])
print (tp)
print (tp + tp)
print (tp * 3)
print (type(tp))
tp[1] = ("banana")
print (tp)
```

Output

```
('apple', 'a', 100)
('apple', 'a', 100, 20.78)
('apple', 'a', 100, 20.78, 'apple', 'a', 100, 20.78)
('apple', 'a', 100, 20.78, 'apple', 'a', 100, 20.78, 'apple', 'a', 100, 20.78)
<class 'tuple'>

Traceback (most recent call last):
  File "C:\Users\Letla\PycharmProjects\python_fundamentals_part1\05 What are Data types and how do they relate to variable\Example 13 - Tuple.py", line 10, in <module>
    tp[1] = ("banana")
TypeError: 'tuple' object does not support item assignment
```

h) List

List stores a collection of different types of elements. The list is mutable (editable). It is the same as arrays in C, but the list stores elements of different data types. To create a list uses the [] square brackets, within this brackets stores all the elements separated with the comma (,). We can use index[i], slice [:] operators, concatenation operator (+), repetition operator (*), etc to works with the list same as with the strings.

The following example shows the list handling:

i) Example 14

```
ls  = ["apple", "a", 100, 20.78]
print (ls[1])
print (ls[1:])
print (ls[:3])
print (ls)
print (ls + ls)
print (ls * 3)
print (type(ls))
ls[1] = "banana"
print (ls)
```

Output

```
a
['a', 100, 20.78]
['apple', 'a', 100]
['apple', 'a', 100, 20.78]
['apple', 'a', 100, 20.78, 'apple', 'a', 100, 20.78]
['apple', 'a', 100, 20.78, 'apple', 'a', 100, 20.78, 'apple', 'a', 100, 20.78]
<class 'list'>
['apple', 'banana', 100, 20.78]
```

j) Set

Set also stores the collection of the elements of different data types. A Set is the same as the list and tuple, but the set is immutable (non-editable or cannot modify the size and elements value), un order, and stores only the unique elements. To create a set uses the {} curly brackets, within this brackets stores all the elements separated with the comma (,).

The following example shows the set handling:

k) Example 15

```
st  = {"apple", "banana", 100, 20.78}
print (st)
print (type(st))
print (st + st)
print (st * 2)
```

Output

```
Traceback (most recent call last):
  File "C:\Users\Letla\PycharmProjects\python_fundamentals_part1\05 What are Data
types and how do they relate to variable\Example 15 - Set.py", line 7, in <module>
    print (st + st) # set cannot support concatenation
TypeError: unsupported operand type(s) for +: 'set' and 'set'

{'apple', 'banana', 100, 20.78}
<class 'set'>
```

l) Dictionary

Dictionary is also stored in a collection of different data types elements in the form of key-value pairs. It is an ordered, mutable, and stores unique keys as a set. To create a set uses the {} curly brackets same as a set, within this, brackets stores all the elements (key-value pair) separated with the comma (,).

The following example shows the set handling

m) Example 16

```
dict = {"fruits":["apple", "banana"],'qty':100}
print("Fruits: ",dict['fruits'])
print("Quantity: ", dict['qty'])
print ("Dictionary: ",dict)
print ("Keys: ",dict.keys())
print ("values: ",dict.values())
print ("key value pairs: ",dict.items()
```

Output

```
Fruits:  ['apple', 'banana']
Quantity:  100
Dictionary:  {'fruits': ['apple', 'banana'], 'qty': 100}
Keys:  dict_keys(['fruits', 'qty'])
values:  dict_values([['apple', 'banana'], 100])
key value pairs:  dict_items([('fruits', ['apple', 'banana']), ('qty', 100)])
```

ii) Mutable vs Immutable

We have said that everything in Python is an object, yet there is an important distinction between objects. Some objects are mutable while some are immutable.

a) Immutable objects in Python

For some types in Python, once we have created instances of those types, they never change. They are immutable.

c) Example 17

int objects are immutable in Python.
What will happen if we try to change the value of an *int* object?

```
x = 24601
print(x)

x = 24602
print(x)
```

Output

```
24601
24602
```

d) Example 18

Well, it seems that we changed **x** successfully. This is exactly where many people get confused. What exactly happened under the hood here? Let's use **id** to further investigate:

```
x = 24601
print(x)
print(id(x))

x = 24602
print(x)
print(id(x))

Output
24601
1470416816

24602
1470416832
```

So we can see that by assigning **x** = 24602, we didn't change the value of the object that **x** had been bound to before. Rather, we created a new object and bound the name **x** to it.

Whenever we assign a new value to a name (in the above example - **x**) that is bound to an *int* object, we change the binding of that name to another object.

The same applies to ***tuples, strings*** (str objects), and ***bools*** as well. In other words, ***int*** (and other number types such as ***float***), ***tuple, bool***, and ***str*** objects are immutable.

e) Mutable objects in Python

Some types in Python can be modified after creation, and they are called mutable.
For example, we know that we can modify the contents of a list object:

f) Example 19

```
my_list = [1, 2, 3]
print(my_list)
my_list[0] = 'a new value'
print(my_list)

Output
[1, 2, 3]
['a new value', 2, 3]
```

Does that mean we created a new object when assigning a new value to the first element of my_list? Again, we can use id to check:

g) Example 20

```
my_list = [1, 2, 3]
print(my_list)
print(id(my_list))

my_list[0] = 'a new value'
print(my_list)
print(id(my_list))
```

Output

```
[1, 2, 3]
2749794763200

['a new value', 2, 3]
2749794763200
```

So our first assignment my_list = [1, 2, 3] created an object in the address 55834760, with the values of 1, 2, and 3.
We then modified the first element of this list object using my_list[0] = 'a new value', that is - without creating a new list object.

In addition to lists, other Python types that are mutable include sets and dicts.

Exercise 02

Please submit screenshots with your answers and upload them to the classroom.
The due date of exercise is 11 Feb at noon solutions to the exercise will be shared at the session coming on the 13 Feb 2021 at 9 am

1. create a variable and assign 100 to it
2. cast the variable created in the last question to a string and float
3. take the words and numbers and create a tuple -> Python, 200, fun, z
4. from this list -> example_list = ["apple", "a", 100, 20.78] print the second and third item only
5. create a set that contains all the 9 provencies
6. from this sample dictonary -> dict = {"fruits":["apple", "banana"],'qty':100} print the type to console

6. Understand what Boolean Logic is

I. Introduction to Boolean Operators in Python

The operators such as not, and, or that are used to perform logical operations in Python, with results of the operations involving them being returned in TRUE or FALSE. The not operator having the highest priority, followed by the and operator also the or operator being the lowest in the order of the priority, and that the **not** operator has a lower priority than non-Boolean operators. in Python programming language the **and** as well as **or** operator is known as the short-circuit operators, are also called Boolean operators

a) Boolean Values

The datatypes like Integer, Float, String, etc. can hold unlimited values, variables of type Boolean can have one of the two values which are either TRUE or FALSE. In Python as a programming language, True and False values are represented as a string without enclosing them in double or single inverted commas and they always start with the uppercase T and F.
Let's consider an example to understand more:

b) Example 21

```
bool_var = True
print(bool_var)
```

Output

```
True
```

In the above example, the variable named bool_var stores the Boolean value of True, and when you print it out on the terminal, it shows True as the value.
By default, the Boolean value True is True in Python and False is False in Python.

I. *true* with a lowercase T is treated as a variable and not as a Boolean value
II. We cannot assign any values or expressions to the Boolean Values True or False in Python

Any value for a numeric datatype but 0 and any value for a string datatype but an empty string when typecasted to Boolean gives True value otherwise, it treats it as False.

c) Example 22

```
a = 1
print(bool(a))

a = 0
print(bool(a))

a = "some string"
print(bool(a))
```

```
a = "" ""
print(bool(a))
```

Output

```
True
False
True
False
```

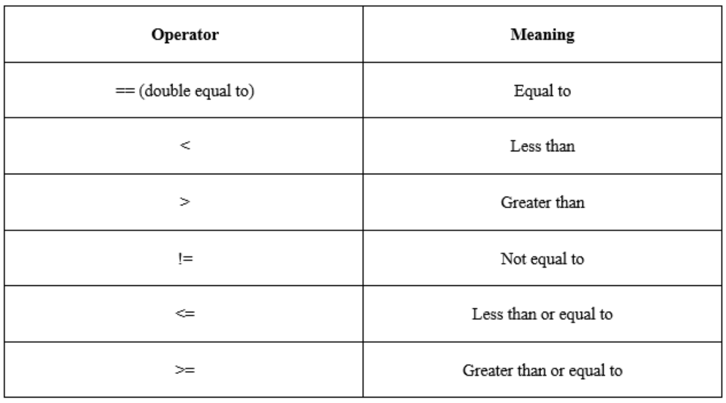
Now that we have understood the Boolean values and their behavior in the Python programming language, let us understand the Boolean Operators.

d) Boolean Operators in Python

Boolean Operators are the operators that operate on the Boolean values and if it is applied on a non-Boolean value then the value is first typecasted and then operated upon. These might also be regarded as the logical operators and the final result of the Boolean operation is a Boolean value, True or False.

ii) Comparison Operators

There are six comparison operators as described in the table below which evaluate the expression to a Boolean value.



Operator	**Meaning**
== (double equal to)	Equal to
<	Less than
>	Greater than
!=	Not equal to
<=	Less than or equal to
>=	Greater than or equal to

Now, let us consider an example each and see how they behave in Python Programming Language

a) Example 23

```
a = 1

print(a == 1)
print(a != 10)
print(a != 1)
print(a > 10)
print(a < 12)
print(a >= 1)
print(a <= 7)

Output
True
True
False
False
True
True
True
```

So, you can see that with the integer value of 1 assigned to the variable 'a' and compared it with many other integral values, we get different Boolean results depending on the scenario. The value of 'a' can also be compared with other variables in a similar fashion

iii) Binary Boolean Operators

These operators are the ones that operate on two values which are both Boolean. The 'and' operator and the 'or' operator are the two binary Boolean operators that operate on some logic to give the Boolean value again. The standard Truth table for these two logical binary Boolean operators is as follows.

The truth table for the 'and' operator. Even if one value is false then the whole expression is False.

Expression	**Evaluates to**
True and True	True
True and False	False
False and True	False
False and False	False

The truth table for the 'or operator. Even if one value is true then the whole expression is True.

Expression	Evaluates to
True or True	True
True or False	True
False or True	True
False or False	False

Now, let's see some examples in Python. In Python, these operators are used by the keywords 'and' and 'or' for the 'and' logic and the 'or' logic respectively

a) Example 24

```
a = True
b = False

print(a and b)

print(a or b)
```

Output

```
False
True
```

iv) Not Operator

The 'not' operator is the logical Boolean Operator that compliments the current Boolean value of the variable. That is, if the value is 'true' then the not operator will modify it to 'false' and vice versa. In Python, it is represented by the keyword 'not'.

Let's see the 'not' operator in action in Python

a) Example 25

```
a = True
print(not a)

print(not not not not a)
```

Output

```
False
True
```

So, this is the way the 'not' operator works in Python

v) Combination of Binary Boolean and Comparison Operators

Since the comparison operators evaluate to Boolean values and binary operators operate upon two Boolean values, we can have an expression that uses a combination of binary Boolean and comparison operators to get a Boolean resultant again.
Let’s consider a few examples and see how to exploit the feature.

a) Example 26

```
print((5 > 3) and (7 == 7))
```

Output

```
True
```

The first bracket evaluates True and second to True as well and the final expression will be True and True which is True.

We can also use the ‘not’ operator in this kind of expression. For example,

b) Example 27

```
print((7 > 3) and (9 != 8) and not False)
```

Output

```
True
```

In this example too, the final ‘not False’ evaluates to True, (9! = 8) evaluates to True and (7 > 3) also evaluates to True which gives us the final expression of (True and True and True) which results to be True.

Note – The expressions inside the brackets are evaluated on priority in Python. The priority of other operators goes like this. If the expression is filled with mathematical operators, the ‘and’ operators, the ‘or’ operators and the ‘not’ operators, then the mathematical operators are evaluated first followed by the ‘not’ operators, followed by the ‘and’ operators and at the end the ‘or’ operators.

vi) Conclusion

Boolean operators are one of the predominant logic that comes in handy while programming; especially while doing some decision making in the logic. Having a thorough knowledge of how they behave would make you an outstanding programmer. Happy coding!

Exercise 03

Will not be adding an exercise after this section due to its complexity but should you require one please ask so I can provide you with it.