Quick Start to Python for Networking and SysAdmin

LEARNING VOYAGE

Table of Contents

Table of Contents

1. Python Scripting Overview

Python Scripting Overview

This lesson will cover the following topics:

- Python scripting &Installing and using Python and various tools
- Variables, numbers, and strings
- Python supported data structures and how to use all of these concepts in a script
- Decision making; that is, the if statement &Looping statements; that is, the for and while loops
- Functions & Modules

Technical requirements

- Two versions, 3.7.2 and 2.7.15, are available at python.org/downloads/.
- In this course we'll work with version 3.7 for code examples and package installing.

Why Python?

- Python has a wide range of libraries for open source data analysis tools, web frameworks, testing, and so on.

- Python is a programming language that can be used on different platforms (Windows, Mac, Linux, and embedded Linux H/W platforms, such as Raspberry Pi).

- It's used to develop desktop as well as web applications & Developers can write programs with fewer lines if they use Python.

Python syntax compared to other programming languages

- The code written in Python is highly readable because it's similar to the English language.

- To complete a command, Python uses new lines & Python has a great feature: indentation.

- Using indentations, we can define the scope for decision-making statements, loops such as for and while loops, functions, and classes.

Installation on the Linux platform

- Most Linux distributions have Python 2 in their default installations.
- Some of them also have Python 3 included.
- To install python3 on Debian-based Linux, run the following command in the Terminal:

```
sudo apt install python3
```

- To install python3 on centos, run the following command in the Terminal:

```
sudo yum install python3
```

Installation on the Windows platform

Installing and using pip to install packages

- In Linux, install pip as follows:

sudo apt install python-pip --- This will install pip for python 2.
sudo apt install python3-pip --- This will install pip for python 3.

- In Windows, install pip as follows:

python -m pip install pip

Installation on Mac

- To install python3, first we must have brew installed on our system.
- To install brew on your system, run the following command:

```
/usr/bin/ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/ma
ster/install)"
```

- By running the preceding command. brew will get installed, Now we will install python3 using brew:

```
brew install python3
```

Installing Jupyter notecourse

- Install Jupyter using pip:

pip install jupyter

- In Linux, pip install jupyter will install Jupyter for python 2.
- If you want to install jupyter for python 3, run the following command:

pip3 install jupyter

Installing and using the virtual environment

- First check whether pip is installed or not, We are going to install pip for python3:

```
sudo apt install python3-pip
```

- Install the virtual environment using pip3:

```
sudo pip3 install virtualenv
```

- Now we will create the virtual environment, You can give it any name; I have called it pythonenv:

```
virtualenv pythonenv
```

Installing and using the virtual environment

- Activate your virtual environment:

```
source venv/bin/activate
```

- After your work is done, you can deactivate virtualenv by using following command:

```
deactivate
```

Python interpreter

- Python is an interpreted language; It has an interactive console called the Python interpreter or Python shell.
- This shell provides a way to execute your program line by line without creating a script.
- You can access all of Python's built-in functions and libraries, installed modules, and command history in the Python interactive console.
- This console gives you the opportunity to to explore Python & You're able to paste code into scripts when you are ready.

Starting the interactive console

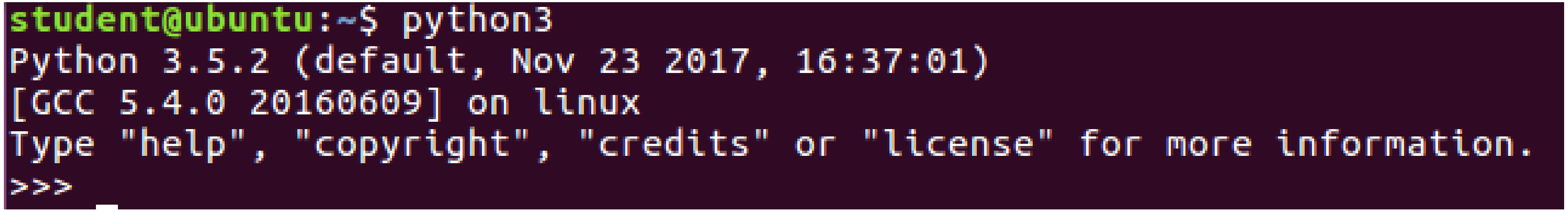
- We can access Python's interactive console from any computer that has Python already installed.

- Run the following command to start Python's interactive console:

```
$ python
```

Writing scripts with the Python interactive console

- The Python interactive console starts from >>> prefix.
- This console will accept the Python commands, which you'll write after >>> prefix.
- Refer to the following screenshot:



```
student@ubuntu:~$ python3
Python 3.5.2 (default, Nov 23 2017, 16:37:01)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Writing scripts with the Python interactive console

- Now, we will see how to assign values to the variable, as in the following example:

```
>>> name = John
```

- Here, we've assigned a character value of John to the name variable.
- We pressed Enter and received a new line with >>> prefix:

```
>>> name = John
```

Writing scripts with the Python interactive console

- Now, we will see an example of assigning values to variables and then we will perform a math operation to get the values:

```
>>> num1 = 5000
>>> num2 = 3500
>>> num3 = num1 + num2
>>> print (num3)
8500
>>> num4 = num3 - 2575
>>> print (num4)
5925
>>>
```

Writing scripts with the Python interactive console

- Next, we subtracted one variable from the result variable, and the output will get stored in the fourth variable.
- Then, we printed the result on to the Terminal.
- So this tells us that we can also use the Python interpreter as a calculator:

```
>>> 509 / 22
23.136363636363637
>>>
```

Multiple lines

- To come out of these lines, you have to press the Enter key twice.
- Now we will look at the following example:

```
>>> val1 = 2500
>>> val2 = 2400
>>> if val1 > val2:
... print("val1 is greater than val2")
... else:
... print("val2 is greater than val1")
...
val1 is greater than val2
>>>
```

Multiple lines

- Remember, statements in if and else blocks are indented.
- If you don't use indentation, you will get the following error:

```
>>> if val1 > val2:
... print("val1 is greater than val2")
File "<stdin>", line 2
print("val1 is greater than val2")
^
IndentationError: expected an indented block
>>>
```

Importing modules through the Python interpreter

- If you are importing any module, then the Python interpreter checks if that module is available or not.
- You can do this by using the import statement.
- If that module is available, then you will see the >>> prefix after pressing the Enter key, This indicates that the execution was successful.
- If that module doesn't exist, the Python interpreter will show an error:

```
>>> import time
>>>
```

Importing modules through the Python interpreter

- After importing the time module, we get the >>> prefix.
- This means that the module exists and this command gets executed successfully:

```
>>> import matplotlib
```

- If the module doesn't exist, then you will get Traceback error:

```
File "<stdin>", line 1, in <module>
ImportError: No module named 'matplotlib'
```

Importing modules through the Python interpreter

- To solve this error, we will have to install matplotlib and then again try to import matplotlib.

- After installing matplotlib, you should be able to import the module, as follows:

```
>>> import matplotlib
>>>
```

Exiting the Python console

We can come out of the Python console in two ways:

- The keyboard shortcut: Ctrl + D
- Using the quit() or exit() functions

The keyboard shortcut

- The keyboard shortcut, Ctrl + D, will give you the following code:

```
>>> val1 = 5000
>>> val2 = 2500
>>>
>>> val3 = val1 - val2
>>> print (val3)
2500
>>>
student@ubuntu:~$
```

Using the quit() or exit() functions

- quit() will take you out of Python's interactive console.
- It will also take you to the original Terminal you were previously in:

```
>>> Lion = 'Simba'
>>> quit()
student@ubuntu$
```

Indentation and tabs

- We use indentation to indicate the block of code in Python programs.
- To indent a block of code, you can use spaces or tabs.
- Refer to the following example:

```
if val1 > val2:
    print ("val1 is greater than val2")
print("This part is not indented")
```

Variables

- In Python, the value of a variable may change during the program execution, as well as the type.
- In the following line of code, we assign the value 100 to a variable:

```
n = 100
```

Here are assigning 100 to the variable n. Now, we are going to increase the value of n by 1:

```
>>> n = n + 1
>>> print(n)
101
>>>
```

Variables

- The following is an example of a type of variable that can change during execution:

```
a = 50 # data type is implicitly set to integer
a = 50 + 9.50 # data type is changed to float
a = "Seventy" # and now it will be a string
```

Variables

- Python takes care of the representation for the different data types; that is, each type of value gets stored in different memory locations.
- A variable will be a name to which we're going to assign a value:

```
>>> msg = 'And now for something completely different'
>>> a = 20
>>> pi = 3.1415926535897932
```

Variables

- The type of a variable is the type of the value it refers to.
- Look at the following code:

```
>>> type(msg)
<type 'str'>
>>> type(a)
<type 'int'>
>>> type(pi)
<type 'float'>
```

Creating and assigning values to variables

- In Python, variables don't need to be declared explicitly to reserve memory space.
- So, the declaration is done automatically whenever you assign a value to the variable.
- In Python, the equal sign = is used to assign values to variables & consider the following example:

Refer to the file 1_1.txt

Creating and assigning values to variables

- Multiple assignments for the same value can be done as follows:

```
x = y = z = 1
```

- In the preceding example, we created three variables and assigned an integer value 1 to them, and all of these three variables will be assigned to the same memory location.
- In Python, we can assign multiple values to multiple variables in a single line:

```
x, y, z = 10, 'John', 80
```

Numbers

- The Python interpreter can also act as a calculator.
- You just have to type an expression and it will return the value.
- Parentheses ( ) are used to do the grouping, as shown in the following example:

```
>>> 5 + 5
10
>>> 100 - 5*5
75
>>> (100 - 5*5) / 15
5.0
>>> 8 / 5
1.6
```

Numbers

- Consider the following example:

```
>>> 14/3
4.666666666666667
>>>
>>> 14//3
4
>>>
>>> 14%3
2
>>> 4*3+2
14
>>>
```

Numbers

- To calculate powers, Python has the ** operator, as shown in the following example:

```
>>> 8**3
512
>>> 5**7
78125
>>>
```

Numbers

- The equal sign (=) is used for assigning a value to a variable:

```
>>> m = 50
>>> n = 8 * 8
>>> m * n
3200
```

- If a variable does not have any value and we still try to use it, then the interpreter will show an error:

```
>>> k
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'k' is not defined
>>>
```

- If the operators have mixed types of operands, then the value we get will be of a floating point:

```
>>> 5 * 4.75 - 1
22.75
```

- In the Python interactive console, _ contains the last printed expression value, as shown in the following example:

```
>>> a = 18.5/100
>>> b = 150.50
>>> a * b
27.8425
>>> b + _
178.3425
>>> round(_, 2)
178.34
>>>
```

Numbers

- We can create number objects just by assigning a value to them, as shown in the following example:

```
num1 = 50
num2 = 25
```

- The del statement is used to delete single or multiple variables. Refer to the following example:

```
del num
del num_a, num_b
```

Strings

Like numbers, strings are also one of the data structures in Python. Python can manipulate strings. Strings can be expressed as follows:

- Enclosed in single quotes ('...')
- Enclosed in double quotes ("...")

See the following example:

```
>>> 'Hello Python'
'Hello Python'
>>> "Hello Python"
'Hello Python'
```

Strings

- A string is a set of characters.
- We can access the characters one at a time, as follows:

```
>>> city = 'delhi'
>>> letter = city[1]
>>> letter = city[-3]
```

Strings

- It starts from 0. So, in the preceding example, when you will execute letter = city[1], you will get the following output:

city d e l h i

index 0 1 2 3 4

-5 -4 -3 -2 -1

Output:

e

l

Concatenation (+) and repetition (*)

- Next, comes concatenation and repetition.
- Refer to the following code:

```
>>> 3 * 'hi' + 'hello'
'hihihihello'
```

- We can automatically concatenate two strings just by writing them next to each other.
- These two strings must be enclosed between quotes, as shown here:

```
>>> 'he' 'llo'
'hello'
```

- This feature is really helpful when you have long strings and you want to break them.
- Here is an example:

```
>>> str = ('Several strings'
... 'joining them together.')
>>> str
'Several strings joining them together.'
```

String slicing

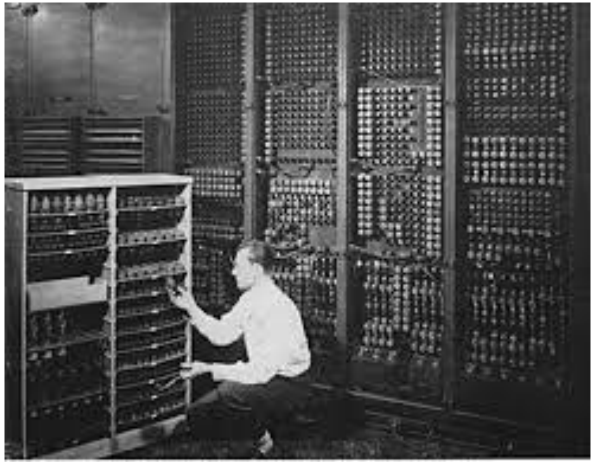
Consider a string, str = "Programming":

```
>>> str[0:2]
'Pr'
>>> str[2:5]
'ogr'
```

String slicing

- Now, the default of an omitted first index is zero, as in the example:

```
>>> str[:2] + str[2:]
'Python'
>>> str[:4] + str[4:]
'Python'
>>> str[:2]
'Py'
>>> str[4:]
'on'
>>> str[-2:]
'on'
```

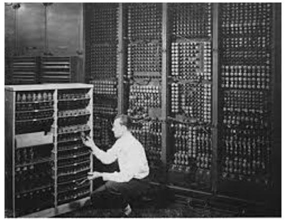


Accessing values in strings

```
#!/usr/bin/python3
str1 = 'Hello Python!'
str2 = "Object Oriented Programming"
print ("str1[0]: ", str1[0])
print ("str2[1:5]: ", str2[1:5])
```

Output:

```
str1[0]: H
str2[1:5]: bjec
```



Updating strings

- We can update a string by reassigning a new value to the specified index.
- Refer to the following example:

```
#!/usr/bin/python3
str1 = 'Hello Python!'
print ("Updated String: - ", str1 [:6] + 'John')
```

Output:

```
Updated String: - Hello John
```

Escape characters

Notations	Hex characters	Description
a	0x07	Bell or alert
b	0x08	Backspace
cx		Control-x
n	0x0a	Newline
C-x		Control-x
e	0x1b	Escape
f	0x0c	Form feed
s	0x20	Space
M-C-x		Meta-control-x
x		Character x
nnn		Octal notation, where n is in the range 0.7
r	0x0d	Carriage return
xnn		Hexadecimal notation, where n is in the range 0.9, a.f, or A.F
t	0x09	Tab
v	0x0b	Vertical tab

Special string operators

Operator	Description	Example
+	Concatenation: adds values on either side of the operator	a + b will give HelloWorld
[]	Slice: gives the character from the given index	a[7] will give r
[:]	Range slice: gives the characters from the given range	a[1:4] will give ell
*	Repetition: creates new strings, concatenating multiple copies of the same string	a*2 will give HelloHello
not in	Membership: returns true if a character does not exist in the given string	Z not in a will give 1
in	Membership: returns true if a character exists in the given string	H in a will give 1
%	Format: performs string formatting	

% string formatting operator

- % is a string formatting operator in Python. Refer to the following example:

```
#!/usr/bin/python3
print ("Hello this is %s and my age is %d !" % ('John', 25))
```

Output:

```
Hello this is John and my age is 25 !
```

S.No.	Format symbol and conversion
1	%c – character
2	%s – string conversion via str() prior to formatting
3	%i – signed decimal integer
4	%d – signed decimal integer
5	%u – unsigned decimal integer
6	%o – octal integer
7	%x – hexadecimal integer (lowercase letters)
8	%X – hexadecimal integer (uppercase letters)
9	%e – exponential notation (with lowercase e)
10	%E – exponential notation (with uppercase E)
11	%f – floating point real number

Triple quotes in Python

- Python's triple quotes functionality for strings is used to span multiple lines, including newlines and tabs.
- The syntax for triple quotes consists of three consecutive single or double quotes, Refer to the following code:

Refer to the file 1_2.txt

- It produces the following output. Note the tabs and newlines:

Refer to the file 1_3.txt

Strings are immutable

- Strings are immutable, meaning we can't change the values.
- Refer to the given example:

```
>>> welcome = 'Hello, John!'
>>> welcome[0] = 'Y'
TypeError: 'str' object does not support item assignment
```

Strings are immutable

- As the strings are immutable; we cannot change an existing string.
- But we can create a new string that will be different from the original:

```
>>> str1 = 'Hello John'
>>> new_str = 'Welcome' + str1[5:]
>>> print(str1)
Hello John
>>> print(new_str)
Welcome John
>>>
```

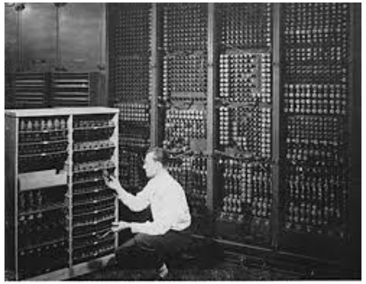
Understanding lists

- Python supports a data structure called list, which is a mutable and ordered sequence of elements.
- Each element in that list is called as item, Lists are defined by inserting values between square brackets [ ].
- Each element of list is given a number, which we call as a position or index.
- The index starts from zero; that is, the first index is zero, the second index is 1, and so on.

Understanding lists

- The following is the code for creating a list:

```
l = list()
numbers = [10, 20, 30, 40]
animals = ['Dog', 'Tiger', 'Lion']
list1 = ['John', 5.5, 500, [110, 450]]
```



Understanding lists

- As you might expect, you can assign list values to variables:

```
>>> cities = ['Mumbai', 'Pune', 'Chennai']
>>> numbers_list = [75, 857]
>>> empty_list = []
>>> print (cities, numbers_list, empty_list)
['Mumbai', 'Pune', 'Chennai'] [75, 857] []
```

Accessing values in lists

```
#!/usr/bin/python3
cities = ['Mumbai', 'Bangalore', 'Chennai', 'Pune']
numbers = [1, 2, 3, 4, 5, 6, 7 ]
print (cities[0])
print (numbers[1:5])
```

Output:

```
Mumbai
[2, 3, 4, 5]
```

Updating lists

```
#!/usr/bin/python3
cities = ['Mumbai', 'Bangalore', 'Chennai', 'Pune']
print ("Original Value: ", cities[3])
cities[3] = 'Delhi'
print ("New value: ", cities[3])
```

Output:

```
Original Value: Pune
New value: Delhi
```

Deleting list elements

```
#!/usr/bin/python3
cities = ['Mumbai', 'Bangalore', 'Chennai', 'Pune']
print ("Before deleting: ", cities)
del cities[2]
print ("After deleting: ", cities)
```

Output:

```
Before deleting: ['Mumbai', 'Bangalore', 'Chennai', 'Pune']
After deleting: ['Mumbai', 'Bangalore', 'Pune']
```

Basic list operations

There are five basic list operations:

- Concatenation
- Repetition
- Length
- Membership
- Iteration

Basic list operations

Description	Expression	Result
Concatenation	[30, 50, 60] + ['Hello', 75, 66]	[30,50,60,'Hello',75,66]
Membership	45 in [45,58,99,65]	True
Iteration	for x in [45,58,99] : print (x,end = ' ')	45 58 99
Repetition	['Python'] * 3	['python', 'python', 'python']
Length	len([45, 58, 99, 65])	4

List operations

- The + operator concatenates lists:

```
>>> a = [30, 50, 60]
>>> b = ['Hello', 75, 66 ]
>>> c = a + b
>>> print c
[30,50,60,'Hello',75,66]
```

List operations

- Similarly, the * operator repeats a list a given number of times:

```
>>> [0] * 4
[0, 0, 0, 0]
>>> ['Python'] * 3
['python', 'python', 'python']
```

Indexing, slicing, and matrices

- Now, we will create a list named cities and we will see the index operations:
- cities = ['Mumbai', 'Bangalore', 'Chennai', 'Pune']

Description	**Expression**	**Results**
Index start at zero	cities[2]	'Chennai'
Slicing: getting sections	cities[1:]	['Bangalore', 'Chennai', 'Pune']
Negative: count from the right	cities[-3]	'Bangalore'

Tuples

- Python's tuple data structure is immutable, meaning we cannot change the elements of the tuples.
- Basically, a tuple is a sequence of values that are separated by commas and are enclosed in parentheses ( ).
- Like lists, tuples are an ordered sequence of elements:

```
>>> t1 = 'h', 'e', 'l', 'l', 'o'
```

- Tuples are enclosed in parentheses ( ):

```
>>> t1 = ('h', 'e', 'l', 'l', 'o')
```

Tuples

- You can also create a tuple with a single element.
- You just have to put a final comma in the tuple:

```
>>> t1 = 'h',
>>> type(t1)
<type 'tuple'>
```

- A value in parentheses is not a tuple:

```
>>> t1 = ('a')
>>> type(t1)
<type 'str'>
```

Tuples

- We can create an empty tuple using the tuple() function:

```
>>> t1 = tuple()
>>> print (t1)
()
```

- If the argument is a sequence (string, list, or tuple), the result is a tuple with the elements of the sequence:

```
>>> t = tuple('mumbai')
>>> print t
('m', 'u', 'm', 'b', 'a', 'i')
```

Tuples

- Tuples have values between parentheses ( ) separated by commas:

```
>>> t = ('a', 'b', 'c', 'd', 'e')
>>> print t[0]
'a’
```

- The slice operator selects a range of elements.

```
>>> print t[1:3]
('b', 'c')
```

Accessing values in tuples

```
#!/usr/bin/python3
cities = ('Mumbai', 'Bangalore', 'Chennai', 'Pune')
numbers = (1, 2, 3, 4, 5, 6, 7)
print (cities[3])
print (numbers[1:6])
```

Output:

```
Pune
(2, 3, 4, 5)
```

Updating tuples

```
#!/usr/bin/python3
cities = ('Mumbai', 'Bangalore', 'Chennai', 'Pune')
numbers = (1,2,3,4,5,6,7)
tuple1 = cities + numbers
print(tuple1)
```

Output:

```
('Mumbai', 'Bangalore', 'Chennai', 'Pune', 1, 2, 3, 4, 5, 6, 7)
```

Deleting tuple elements

```
#!/usr/bin/python3
cities = ('Mumbai', 'Bangalore', 'Chennai', 'Pune')
print ("Before deleting: ", cities)
del cities
print ("After deleting: ", cities)
```

Output:

```
Before deleting: ('Mumbai', 'Bangalore', 'Chennai', 'Pune')
Traceback (most recent call last):
File "01.py", line 5, in <module>
print ("After deleting: ", cities)
NameError: name 'cities' is not defined
```

Basic tuple operations

Like lists, there are five basic tuple operations:

- Concatenation
- Repetition
- Length
- Membership
- Iteration

Basic tuple operations

Description	Expression	Results
Iteration	for x in (45,58,99) : print (x,end = ' ')	45 58 99
Repetition	('Python') * 3	('python', 'python', 'python')
Length	len(45, 58, 99, 65)	4
Concatenation	(30, 50, 60) + ('Hello', 75, 66)	(30,50,60,'Hello',75,66)
Membership	45 in (45,58,99,65)	True

Indexing, slicing, and matrices

- Now, we will create a tuple named cities and perform some index operations:
- cities = ('Mumbai', 'Bangalore', 'Chennai', 'Pune')

Description	Expression	Results
Index starts at zero	cities[2]	'Chennai'
Slicing: getting sections	cities[1:]	('Bangalore', 'Chennai', 'Pune')
Negative: count from the right	cities[-3]	'Bangalore'

max() and min()

- Using the max() and min() functions, we can find the highest and lowest values from the tuple.
- These functions allow us to find out information about quantitative data.
- Let's look at an example:

```
>>> numbers = (50, 80,98, 110.5, 75, 150.58)
>>> print(max(numbers))
150.58
>>>
```

max() and min()

- Using max(), we will get the highest value in our tuple. Similarly, we can use the min() function:

```
>>> numbers = (50, 80,98, 110.5, 75, 150.58)
>>> print(min(numbers))
50
>>>
```

- So, here we are getting the minimum value.

Sets

- A set is an unordered collection of elements with no duplicates, The basic use of a set is to check membership testing and eliminate duplicate entries.
- These set objects support mathematical operations, such as union, intersection, difference, and symmetric difference.
- We can create a set using curly braces or the set() function, If you want create an empty set, then use set(), not {}.
- Here is a brief demonstration:

Refer to the file 1_4.txt

Sets

- Set comprehensions are also supported in Python. Refer to the following code:

```
>>> a = {x for x in 'abracadabra' if x not in 'abc'}
>>> a
{'r', 'd'}
```

Dictionaries

- A dictionary is a data type in Python, which consists of key value pairs and is enclosed in curly braces {}.

- Dictionaries are unordered and indexed by keys, where each key must be unique.

- These keys must be immutable type.

- Tuples can be used as keys if they contain only strings, numbers, or tuples.

```
>>> student = {'Name':'John', 'Age':25}
>>> student['Address'] = 'Mumbai'
>>> student
student = {'Name':'John', 'Age':25, 'Address':'Mumbai'}
>>> student['Age']
25
>>> del student['Address']
>>> student
student = {'Name':'John', 'Age':25}
>>> list(student.keys())
['Name', 'Age']
>>> sorted(student.keys())
['Age', 'Name']
>>> 'Name' in student
True
>>> 'Age' not in student
False
```

Dictionaries

- Arbitrary key and value expressions along with dictionary comprehensions are used to create dictionaries:

```
>>> {x: x**2 for x in (4, 6, 8)}
{4: 16, 6: 36, 8: 64}
```

- When the keys are simple strings, it is sometimes easier to specify pairs using keyword arguments:

```
>>> dict(John=25, Nick=27, Jack=28)
{'Nick': 27, 'John': 25, 'Jack': 28}
```

Parsing command-line arguments

Command-line arguments in Python

- We can start a program with additional arguments, in the command line.
- Python programs can start with command-line arguments.
- Let's look at an example:

```
$ python program_name.py img.jpg
```

- Here, program_name.py and img.jpg are arguments.

Command-line arguments in Python

Module	**Use**	**Python version**
optparse	Deprecated	< 2.7
sys	All arguments in sys.argv (basic)	All
argparse	Building a command-line interface	> = 2.3
fire	Automatically generating **Command-Line Interfaces** (**CLIs**)	All
docopt	Creating CLIs interfaces	> = 2.5

Sys.argv

```
01.py
import sys
print('Number of arguments:', len(sys.argv))
print('Argument list:', str(sys.argv))
```

Output:

```
Python3 01.py img
Number of arguments 2
Arguments list: ['01.py', 'img']
```

Decision making

Python if statement syntax

The following is the syntax for the if statement:

```
if test_expression:
    statement(s)
```

Decision making

```
a = 10
if a > 0:
    print(a, "is a positive number.")
print("This statement is always printed.")

a = -10
if a > 0:
    print(a, "is a positive number.")
```

Output:

```
10 is a positive number.
This statement is always printed.
```

Python if...else statement syntax

- In this section, we are going to learn about the if..else statement.
- The else block will get executed only when the if condition is false. Refer to the following syntax:

```
if test expression:
    if block
else:
    else block
```

Python if...else statement syntax

- Indentation is used to separate the blocks. Refer to the following example:

```
a = 10
if a > 0:
    print("Positive number")
else:
    print("Negative number")
```

```
Output:
Positive number
```

Python if...elif...else statement

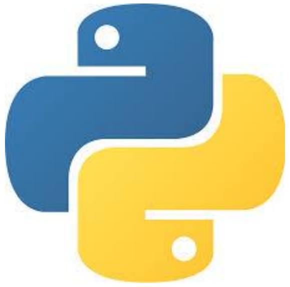
- The elif statement checks multiple statements for a true value.
- Whenever the value evaluates to true, that code block gets executed, Refer to the following syntax:

```
if test expression:
    if block statements
elif test expression:
    elif block statements
else:
    else block statements
```

Python if...elif...else statement

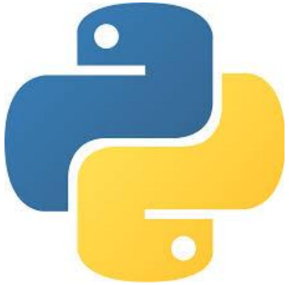
- The elif statement checks multiple statements for a true value.
- Whenever the value evaluates to true, that code block gets executed, Refer to the following syntax:

```
if test expression:
    if block statements
elif test expression:
    elif block statements
else:
    else block statements
```



- Only one block among the several if...elif...else blocks is executed according to the condition.
- The if block can have only one else block. But it can have multiple elif blocks, Let's take a look at an example:

```
a = 10
if a > 50:
 print("a is greater than 50")
elif a == 10:
 print("a is equal to 10")
else:
 print("a is negative")

Output:
a is equal to 10
```



Loops

To handle all of the looping requirements in your script, Python supports two loops:

- for loop
- while loop

Now, we are going to learn about for loop and while loop.

for loop

- for loop iterates over each item of the sequence or any other iterable object and it will execute the statements in the for block each time.

- Refer to the following syntax:

```
for i in sequence:
    for loop body
```

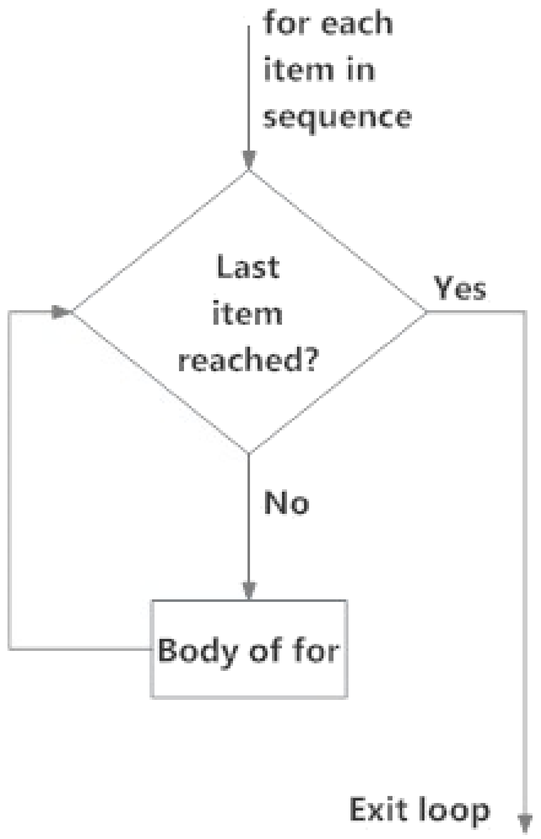


Fig: operation of for loop

- Refer to the following example:

```
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]
sum = 0
for i in numbers:
 sum = sum + i
 print("The sum is", sum)
```

Output:

```
The sum is 6
The sum is 11
The sum is 14
The sum is 22
The sum is 26
The sum is 28
The sum is 33
The sum is 37
The sum is 48
```

The range() function

- The Python range() function will generate a sequence of numbers.
- For example, range(10) will generate numbers from 0 to 9 (10 numbers).
- We can also define the start, stop, and step size as parameters and range() will be as follows:

range(start, stop, step size).

Step size defaults to 1 if not provided.

For loop example using range() function:

The range() function

- Let's take a look at an example:

```
for i in range(5):
 print("The number is", i)
```

Output:

```
The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
```

while loop

- while is a looping statement that will iterate over a block of code until the entered test expression is true.

- We use this loop when we don't know how many times the iterations will go on. Refer to the following syntax:

```
while test_expression:
    while body statements
```

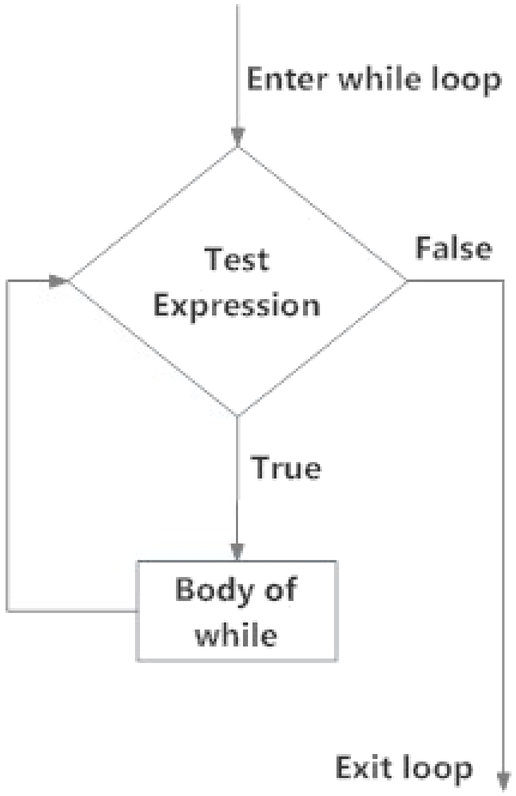


Fig: operation of while loop

```
a = 10
sum = 0
i = 1
while i <= a:
    sum = sum + i
    i = i + 1
    print("The sum is", sum)
```

Output:

```
The sum is 1
The sum is 3
The sum is 6
The sum is 10
The sum is 15
The sum is 21
The sum is 28
The sum is 36
The sum is 45
The sum is 55
```

Iterators

- In Python, an iterator is an object that can be iterated upon.
- It is an object that will return data, one element at a time.
- Python's iterator object implements two methods, __iter__() and __next__().
- Mostly, iterators are implemented within loops, generators, and comprehensions.

Refer to the file 1_5.txt

Generators

How to create a generator in Python?

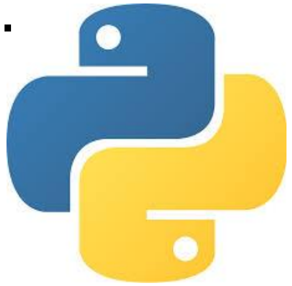
- Creating a generator is easy in Python.
- You can create a generator just by defining a function with a yield statement instead of a return statement.
- If a function contains at least one yield statement, it becomes a generator function. yield and return statements will return some value from a function.
- Here is an example:

Refer to the file 1_6.txt

Functions

- A function is a set of statements that perform a specific task.
- Using functions helps in breaking our program into smaller parts.
- Programs will be more organized if we use functions as it avoids repetition and makes code reusable.
- Look at the following syntax:

```
def function_name(parameters):
    statement(s)
```



Functions

- Refer to the following example:

```
def welcome(name):
    print("Hello " + name + ", Welcome to Python
Programming !")

welcome("John")
```

Output:

```
Hello John, Welcome to Python Programming !
```

The return statement

- The return statement is used to exit a function. Refer to the following syntax:

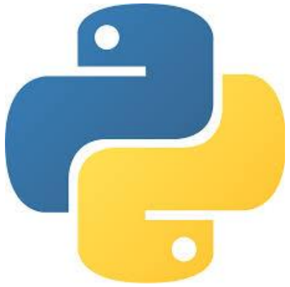
```
return [expression_list]
```

The return statement

```
def return_value(a):
    if a >= 0:
        return a
    else:
        return -a
print(return_value(2))
print(return_value(-4))
```

Output:

```
2
4
```



Lambda functions

- Lambda functions are used along with built-in functions, such as filter(), and map().
- The filter() function returns a list of elements and has only one iterable as input.
- The following shows an example using filter():

```
numbers = [10, 25, 54, 86, 89, 11, 33, 22]
new_numbers = list(filter(lambda x: (x%2 == 0) , numbers))
print(new_numbers)
```

Output:

```
[10, 54, 86, 22]
```

Lambda functions

- The map() function returns a list of results after applying the specified function.
- Now, let's look at an example using map():

```
my_list = [1, 5, 4, 6, 8, 11, 3, 12]
new_list = list(map(lambda x: x * 2 , my_list))
print(new_list)
```

Output:

```
[2, 10, 8, 12, 16, 22, 6, 24]
```

- Here, the map() function is taking a lambda function and a list.

Modules

- Modules are just files that contain Python statements and definitions.
- A file that contains Python code (for example, sample.py) is called a module and its module name would be sample.
- Using modules, we can break larger programs into small and organized ones.
- An important feature of a module is re-usability.

Modules

- Let's create a module and import it, We will create two scripts: sample.py and add.py.
- We will import a sample module in our add.py.
- Now, save the following code as sample.py.
- Let's take a look with the following example:

```
sample.py
def addition(num1, num2):
    result = num1 + num2
    return result
```

Importing modules

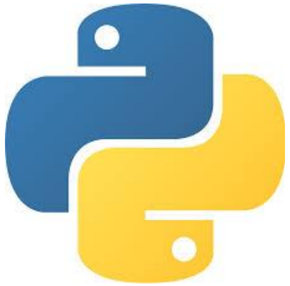
- Now, after creating a module, we will learn how to import that module.
- In the previous example, we created a sample module.
- Now we will import the sample module in add.py script:

add.py

```
import sample
sum = sample.addition(10, 20)
print(sum)
```

Output:

```
30
```



Summary

- In this lesson, we've given an overview of the Python scripting language.
- We have learned about how to install Python and various tools.
- We also learned about the Python interpreter and how to use it.
- We learned about Python-supported data types, variables, numbers and strings, decision-making statements, and looping statements in Python.

2. Debugging and Profiling Python Scripts

Debugging and Profiling Python Scripts

In this lesson, you'll learn about the following:

- Python debugging techniques
- Error handling (exception handling)
- Debugger tools
- Debugging basic program crashes
- Profiling and timing programs
- Making programs run faster

What is debugging?

- Debugging is a process that resolves the issues that occur in your code and prevent your software from running properly.

- In Python, debugging is very easy, The Python debugger sets conditional breakpoints and debugs the source code one line at a time.

- We'll debug our Python scripts using a pdb module that's present in the Python standard library.

Error handling (exception handling)

- An exception is an error that occurs during program execution.
- Whenever any error occurs, Python generates an exception that will be handled using a try…except block.
- Some exceptions can't be handled by programs so they result in error messages, Now, we are going to see some exception examples.
- In your Terminal, start the python3 interactive console and we will see some exception examples:

Refer to the file 2_1.txt

Error handling (exception handling)

- Now we are going to see a try…except block that handles an exception.
- In the try block, we will write a code that may generate an exception.
- In the except block, we will write a solution for that exception.
- The syntax for try…except is as follows:

```
try:
        statement(s)
except:
        statement(s)
```

Error handling (exception handling)

- A try block can have multiple except statements.
- We can handle specific exceptions also by entering the exception name after the except keyword.
- The syntax for handling a specific exception is as follows:

```
try:
        statement(s)
except exception_name:
        statement(s)
```

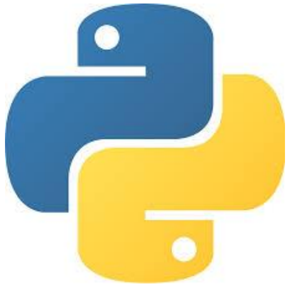
Error handling (exception handling)

```
a = 35
b = 57
try:
        c = a + b
        print("The value of c is: ", c)
        d = b / 0
        print("The value of d is: ", d)

except:
        print("Division by zero is not possible")

print("Out of try...except block")
```

Error handling (exception handling)

- Run the script as follows and you will get the following output:

```
student@ubuntu:~$ python3 exception_example.py
The value of c is:  92
Division by zero is not possible
Out of try...except block
```



Debuggers tools

There are many debugging tools supported in Python:

- winpdb
- pydev
- pydb
- pdb
- gdb
- pyDebug

```
layers: [ArtLayers]
text: Hello
AdjustmentReference: …
AnchorPosition: [Obje…
AntiAlias: [Object] A…
app: [Application Ado…
AutoKernType: [Object…
```

```
function addTextLayer(text) {
    var layers = app.activeDocument.artLayers;
    var layer = layers.add();
    layer.kind = LayerKind.TEXT;

    var textItem = layer.textItem;
    textItem.kind = TextType.PARAGRAPHTEXT;
    textItem.size = 24;
    textItem.position = [0, 0];
    textItem.contents = text;
    textItem.width = new UnitValue('500 pixels');
    textItem.height = new UnitValue('500 pixels');
}
```

The pdb debugger

Now we will learn about how we can use the pdb debugger. There are three ways to use this debugger:

- Within an interpreter
- From a command line
- Within a Python script

```
function addTextLayer(text) {
    var layers = app.activeDocument.artLayers;
    var layer = layers.add();
    layer.kind = LayerKind.TEXT;

    var textItem = layer.textItem;
    textItem.kind = TextType.PARAGRAPHTEXT;
    textItem.size = 24;
    textItem.position = [0, 0];
    textItem.contents = text;
    textItem.width = new UnitValue('500 pixels');
    textItem.height = new UnitValue('500 pixels');
}
```

The pdb debugger

- We are going to create a pdb_example.py script and add the following content in that script:

```
class Student:
        def __init__(self, std):
                self.count = std

        def print_std(self):
                for i in range(self.count):
                        print(i)
                return
if __name__ == '__main__':
        Student(5).print_std()
```

```
function addTextLayer(text) {
    var layers = app.activeDocument.artLayers;
    var layer = layers.add();
    layer.kind = LayerKind.TEXT;

    var textItem = layer.textItem;
    textItem.kind = TextType.PARAGRAPHTEXT;
    textItem.size = 24;
    textItem.position = [0, 0];
    textItem.contents = text;
    textItem.width = new UnitValue('500 pixels');
    textItem.height = new UnitValue('500 pixels');
}
```

Within an interpreter

- To start the debugger from the Python interactive console, we are using run() or runeval().
- Start your python3 interactive console.
- Run the following command to start the console:

```
$ python3
```

Within an interpreter

```
student@ubuntu:~$ python3
Python 3.5.2 (default, Nov 23 2017, 16:37:01)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> import pdb_example
>>> import pdb
>>> pdb.run('pdb_example.Student(5).print_std()')
> <string>(1)<module>()
(Pdb)
```

```
student@ubuntu:~$ python3
Python 3.5.2 (default, Nov 23 2017, 16:37:01)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> import pdb_example
>>> import pdb
>>> pdb.run('pdb_example.Student(5).print_std()')
> <string>(1)<module>()
(Pdb) continue
0
1
2
3
4
>>>
```

```
function addTextLayer(text) {
    var layers = app.activeDocument.artLayers;
    var layer = layers.add();
    layer.kind = LayerKind.TEXT;

    var textItem = layer.textItem;
    textItem.kind = TextType.PARAGRAPHTEXT;
    textItem.size = 24;
    textItem.position = [0, 0];
    textItem.contents = text;
    textItem.width = new UnitValue('500 pixels');
    textItem.height = new UnitValue('500 pixels');
}
```

From a command line

- The simplest and most straightforward way to run a debugger is from a command line.

- Our program will act as input to the debugger.

- You can use the debugger from command line as follows:

```
$ python3 -m pdb pdb_example.py
```

```
student@ubuntu:~$ python3 -m pdb pdb_example.py
> /home/student/pdb_example.py(1)<module>()
-> class Student:
(Pdb) continue
0
1
2
3
4
The program finished and will be restarted
> /home/student/pdb_example.py(1)<module>()
-> class Student:
(Pdb)
```

Within a Python script

- The previous two techniques will start the debugger at the beginning of a Python program.
- But this third technique is best for long-running processes.
- To start the debugger within a script, use set_trace().
- Now, modify your pdb_example.py file as follows:

Refer to the file 2_2.txt

Within a Python script

- Now, run the program as follows:

```
student@ubuntu:~$ python3 pdb_example.py
> /home/student/pdb_example.py(10)print_std()
-> print(i)
(Pdb) continue
0
> /home/student/pdb_example.py(9)print_std()
-> pdb.set_trace()
(Pdb)
```

Debugging basic program crashes

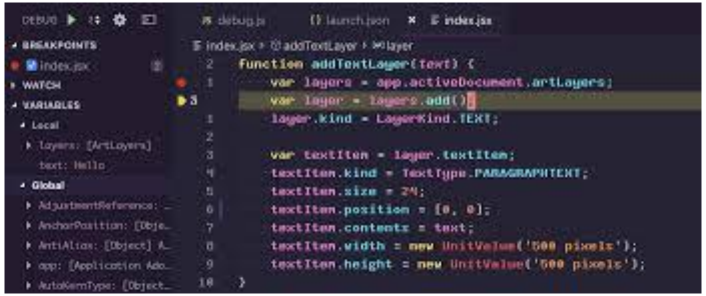
- Now, we will create a script named trace_example.py and write the following content in the script:

```
class Student:
	def __init__(self, std):
		self.count = std

	def go(self):
		for i in range(self.count):
			print(i)
		return
if __name__ == '__main__':
	Student(5).go()
```

Debugging basic program crashes

- The output will be as follows:

Refer to the file 2_3.txt



```
function addTextLayer(text) {
    var layers = app.activeDocument.artLayers;
    var layer = layers.add();
    layer.kind = LayerKind.TEXT;

    var textItem = layer.textItem;
    textItem.kind = TextType.PARAGRAPHTEXT;
    textItem.size = 24;
    textItem.position = [0, 0];
    textItem.contents = text;
    textItem.width = new UnitValue('500 pixels');
    textItem.height = new UnitValue('500 pixels');
}
```

Profiling and timing programs

The cProfile module

- Now, we will write a cprof_example.py script and write the following code in it:

```
mul_value = 0
def mul_numbers( num1, num2 ):
        mul_value = num1 * num2;
        print ("Local Value: ", mul_value)
        return mul_value
mul_numbers( 58, 77 )
print ("Global Value: ", mul_value)
```

- Run the program and you will see the output as follows:

```
student@ubuntu:~$ python3 -m cProfile cprof_example.py
Local Value:  4466
Global Value:  0
         6 function calls in 0.000 seconds
   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.000    0.000 cprof_example.py:1(<module>)
        1    0.000    0.000    0.000    0.000 cprof_example.py:2(mul_numbers)
        1    0.000    0.000    0.000    0.000 {built-in method builtins.exec}
        2    0.000    0.000    0.000    0.000 {built-in method builtins.print}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler'
objects}
```

timeit

- We are going to write a script to time a piece of code.
- Create a timeit_example.py script and write the following content into it:

```
import timeit
prg_setup = "from math import sqrt"
prg_code = '''
def timeit_example():
        list1 = []
        for x in range(50):
                list1.append(sqrt(x))
'''
# timeit statement
print(timeit.timeit(setup = prg_setup, stmt = prg_code, number = 10000))
```

Making programs run faster

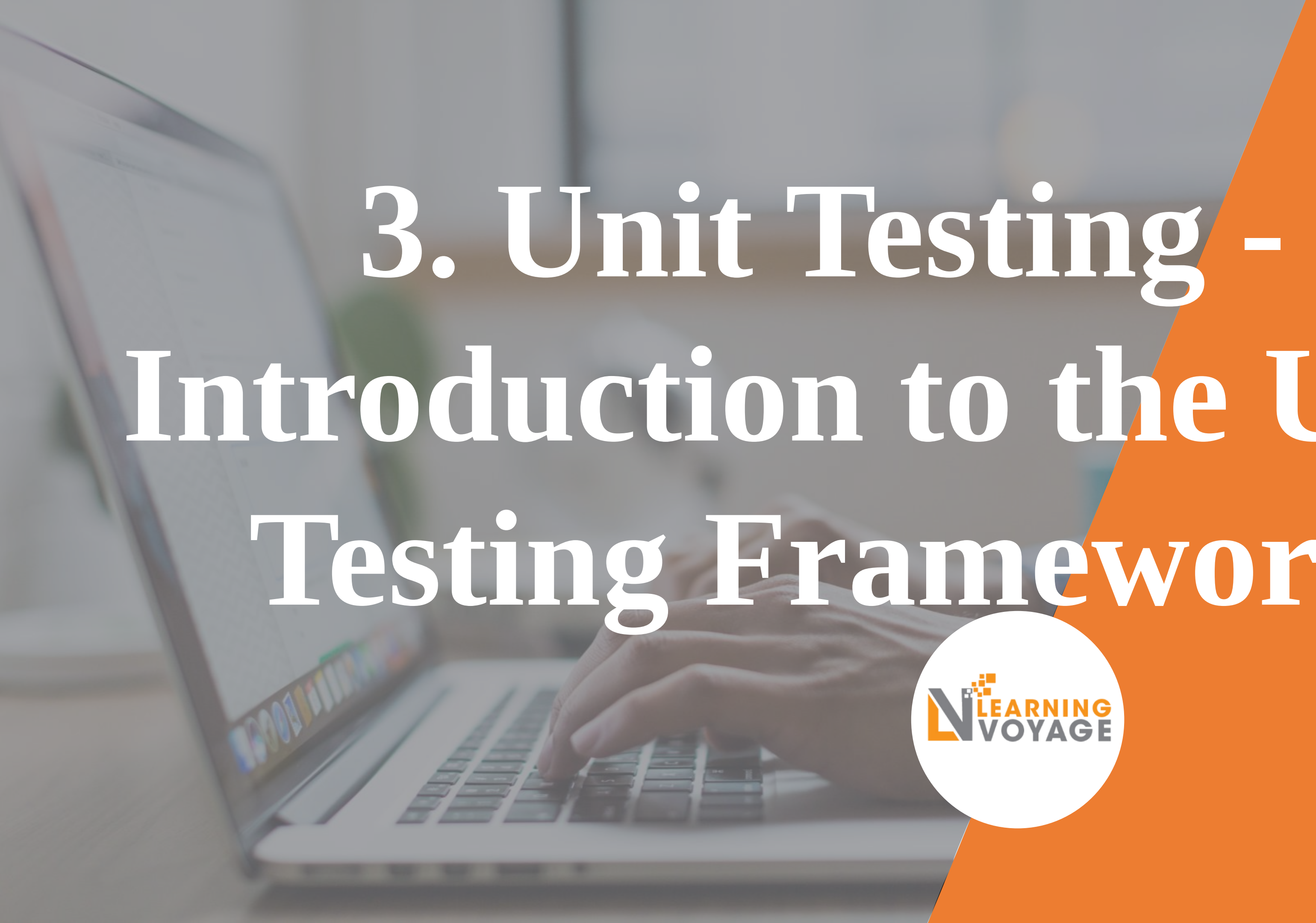
There are various ways to make your Python programs run faster, such as the following:

- Profile your code so you can identify the bottlenecks
- Use built-in functions and libraries so the interpreter doesn't need to execute loops
- Avoid using globals as Python is very slow in accessing global variables
- Use existing packages

Summary

- In this lesson, we learned about the importance of debugging and profiling programs.
- We learned what the different techniques available for debugging are.
- We learned about the pdb Python debugger and how to handle exceptions.
- We learned about how to use the cProfile and timeit modules of Python while profiling and timing our scripts.

3. Unit Testing - Introduction to the Unit Testing Framework



LEARNING VOYAGE

Unit Testing

In this lesson, you will learn about the following topics:

- Introduction to unit testing framework
- Creating unit testing tasks

What is unittest?

unittest supports four main concepts, which are listed here:

- test fixture: This includes preparation and cleanup activities for performing one or more tests
- test case: This includes your individual unit test. By using the TestCase base class of unittest, we can create new test cases
- test suite: This includes a collection of test cases, test suites, or both. This is for executing test cases together
- test runner: This includes arranging the test executions and giving output to the users

Creating unit tests

- First, create a script named arithmetic.py and write the following code in it:

```
# In this script, we are going to create a 4 functions: add_numbers,
sub_numbers, mul_numbers, div_numbers.
def add_numbers(x, y):
    return x + y
def sub_numbers(x, y):
    return x - y
def mul_numbers(x, y):
    return x * y
def div_numbers(x, y):
    return (x / y)
```

Creating unit tests

- In the preceding script, we created four functions: add_numbers, sub_numbers, mul_numbers, and div_numbers.
- Now, we are going to write test cases for these functions.
- First, we will learn how we can write test cases for the add_numbers function.
- Create a test_addition.py script and write the following code in it:

Refer to the file 3_1.txt

Creating unit tests

- Now, we will run our test_addition.pytest script and we will see what result we get after running this script.
- Run the script as follows and you will get the following output:

```
student@ubuntu:~$ python3 test_addition.py
...
----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

Creating unit tests

- Whenever you run your test script, you have three possible test results:

Result	Description
OK	Successful
FAIL	Test failed– raises an AssertionError exception
ERROR	Raises an exception other than AssertionError

Methods used in unit testing

- You can use the unittest module from the command line as well.
- So, you can run the previous test script as follows:

```
student@ubuntu:~$ python3 -m unittest test_addition.py
...
----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

Methods used in unit testing

- Create a if_example.py script and write the following code in it:

```
def check_if():
    a = int(input("Enter a number \n"))
    if (a == 100):
        print("a is equal to 100")

    else:
        print("a is not equal to 100")

    return a
```

Methods used in unit testing

- Now, create a test_if.pytest script and write following code in it:

```
import if_example
import unittest

class Test_if(unittest.TestCase):
    def test_if(self):
        result = if_example.check_if()
        self.assertEqual(result, 100)

if __name__ == '__main__':
    unittest.main()
```

Methods used in unit testing

- Run the test script as follows:

```
student@ubuntu:~/Desktop$ python3 -m unittest test_if.py
Enter a number
100
a is equal to 100
.
----------------------------------------------------------------------
Ran 1 test in 1.912s

OK
```

Methods used in unit testing

- We run the script for a successful test result.
- Now, we will enter some value other than 100 and we must get a FAILED result.
- Run the script as follows:

Refer to the file 3_2.txt

Summary

- In this lesson, we have learned about the unittest, which is Python's unit testing framework.
- We also learned about how to create test cases and methods used in unit testing.
- In the next lesson, we are going to learn how to automate the regular administrative activities of the system administrator.

4. Automating Regular Administrative Activities

Automating Regular Administrative Activities

In this lesson, we will cover the following topics:

- Accepting input by redirection, pipe, and input files
- Handling passwords at runtime in scripts
- Executing external commands and getting their output
- Prompting for a password during runtime and validation
- Reading configuration files

Accepting input by redirection, pipe, and input files

Input by redirection

- stdin and stdout are objects created by the os module. We're going to write a script in which we will use stdin and stdout.

- Create a script called redirection.py and write the following code in it:

Refer to the file 3_3.txt

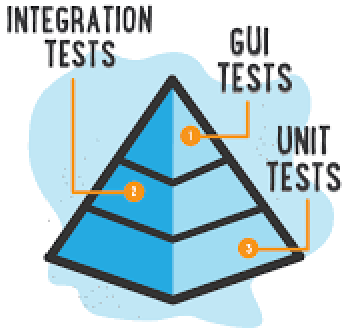
Automating Regular Administrative Activities

- Run the preceding program as follows:

```
$ python3 redirection.py
```

- We will receive the following output:

```
Output:
Enter a string: hello
Enter another string: python
Entered strings are:  'hello' and 'python'
```



Input by pipe

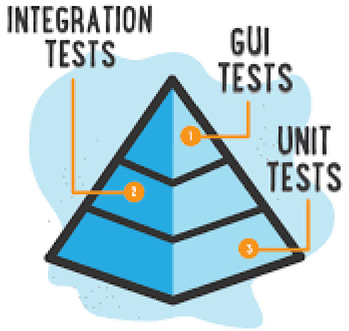
- Now, we are going to see how we can accept an input using pipe.
- For that, first we'll write a simple script that returns a floor division.
- Create a script called accept_by_pipe.py and write the following code in it:

```
import sys

for n in sys.stdin:
    print ( int(n.strip())//2 )
```

Input by pipe

- Run the script and you will get the following output:

```
$ echo 15 | python3 accept_by_pipe.py
Output:
7
```



Input by pipe

- The output will be printed on a new line for each input, as we have written \n between the input value.
- To enable this interpretation of a backslash, we passed the -e flag:

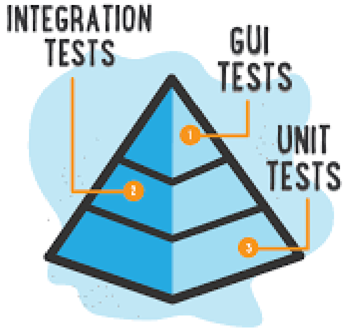
```
$ echo -e '15\n45\n20' | python3 accept_by_pipe.py
Output:
7
22
10
```

Input by input file

- We are going to look at an example for this.
- But first, we are going to create a simple text file called sample.txt and we'll write the following code in it:

- Sample.txt:

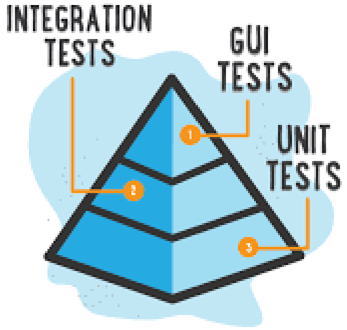
```
Hello World
Hello Python
```



Input by input file

- Now, create a script called accept_by_input_file.py and write the following code in it:

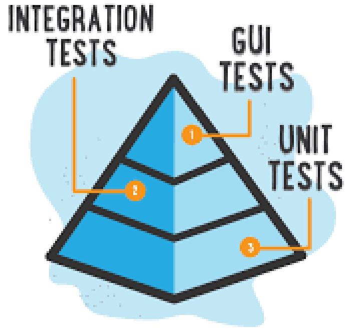
```
i = open('sample.txt','r')
o = open('sample_output.txt','w')

a = i.read()
o.write(a)
```



Input by input file

- Run the program and you will get the following output:

```
$ python3 accept_by_input_file.py
$ cat sample_output.txt
Hello World
Hello Python
```



Handling passwords at runtime in scripts

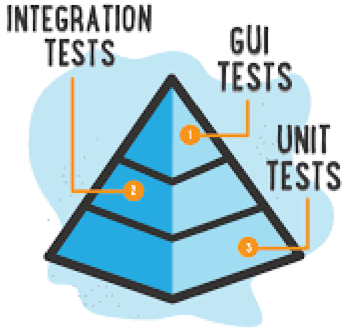
- In this section, we will look at a simple example for handling passwords in script.
- We will create a script called handling_password.py and write the following content in it:

Refer to the file 3_4.txt

Handling passwords at runtime in scripts

- Run the preceding script and you will receive the following output:

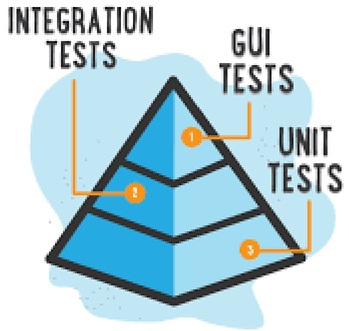
```
$ python3 handling_password.py

Output:
Successful connection 192.168.2.106
```



Handling passwords at runtime in scripts

- In the preceding script, we used the paramiko module.
- The paramiko module is a Python implementation of ssh that provides client-server functionality.
- Install paramiko as follows:

```
pip3 install paramiko
```



Executing external commands and getting their output

- We are going to see how we can execute external commands and get their output in Python by using the subprocess module.
- We will create a script called execute_external_commands.py and write the following code in it:

```
import subprocess
subprocess.call(["touch", "sample.txt"])
subprocess.call(["ls"])
print("Sample file created")
subprocess.call(["rm", "sample.txt"])
```

Executing external commands and getting their output

```
subprocess.call(["ls"])
print("Sample file deleted")
```

- Run the program and you will get the following output:

Refer to the file 3_5.txt

Capturing output using the subprocess module

- In this section, we are going to learn about how we can capture output.
- We will pass PIPE for the stdout argument to capture the output.
- Write a script called capture_output.py and write the following code in it:

```
import subprocess
res = subprocess.run(['ls', '-1'], stdout=subprocess.PIPE,)
print('returncode:', res.returncode)
print(' {} bytes in stdout:\n{}'.format(len(res.stdout),
res.stdout.decode('utf-8')))
```

Capturing output using the subprocess module

- Execute the script as follows:

```
student@ubuntu:~$ python3 capture_output.py
```

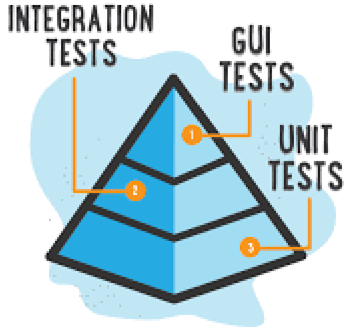
- On execution, we will receive the following output:

Refer to the file 3_6.txt

Prompting for passwords during runtime and validation

- Create a script called no_prompt.py and write the following code in it:

```
import getpass
try:
	p = getpass.getpass()
except Exception as error:
	print('ERROR', error)
else:
	print('Password entered:', p)
```

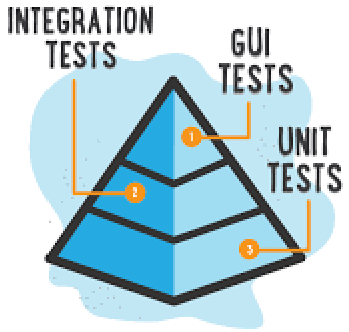


Prompting for passwords during runtime and validation

- In this script, a prompt is not provided for the user. So, by default, it is set to the Password prompt.

- Run the script as follows:

```
$ python3 no_prompt.py
Output :
Password:
Password entered: abcd
```



Prompting for passwords during runtime and validation

- We will provide a prompt for entering a password.
- So, create a script callled with_prompt.py and write the following code in it:

```
import getpass
try:
	p = getpass.getpass("Enter your password: ")
except Exception as error:
	print('ERROR', error)
else:
	print('Password entered:', p)
```

Prompting for passwords during runtime and validation

- Now, we have written a script that provides a prompt for a password.
- Run the program as follows:

```
$ python3 with_prompt.py
Output:
Enter your password:
Password entered: abcd
```

Prompting for passwords during runtime and validation

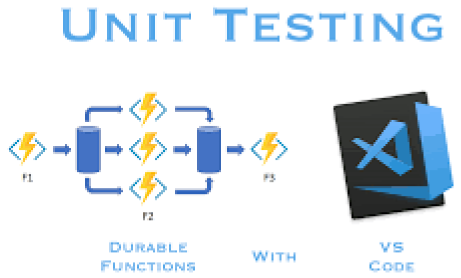
- Write a script called getpass_example.py and write the following code in it:

```
import getpass
passwd = getpass.getpass(prompt='Enter your password: ')
if passwd.lower() == '#pythonworld':
        print('Welcome!!')
else:
        print('The password entered is incorrect!!')
```

Prompting for passwords during runtime and validation

- Run the program as follows (here we are entering a correct password, that is, #pythonworld):

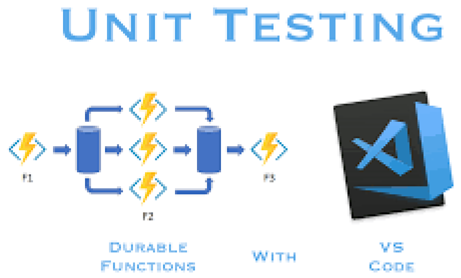
```
$ python3 getpass_example.py
Output:
Enter your password:
Welcome!!
```



Prompting for passwords during runtime and validation

- Now, we will enter a wrong password and will check what message we receive:

```
$ python3 getpass_example.py
Output:
Enter your password:
The password entered is incorrect!!
```



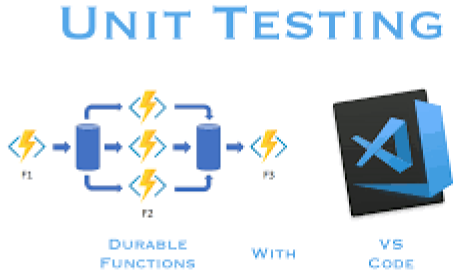
- Create a script called password_prompt_again.py and write the following code in it:

```
import getpass
user_name = getpass.getuser()
print ("User Name : %s" % user_name)
while True:
        passwd = getpass.getpass("Enter your Password : ")
        if passwd == '#pythonworld':
                print ("Welcome!!!")
                break
        else:
                print ("The password you entered is incorrect.")
```

Prompting for passwords during runtime and validation

- Run the program and you will get the following output:

```
student@ubuntu:~$ python3
password_prompt_again.py
User Name : student
Enter your Password :
The password you entered is incorrect.
Enter your Password :
Welcome!!!
```



Reading configuration files

- To read a configuration file, configparser has the read() method.
- Now, we will write a simple script named read_config_file.py.
- Before that, create a .ini file named read_simple.ini and write the following content in it: read_simple.ini

```
[bug_tracker]
url =https://timesofindia.indiatimes.com/
```

Reading configuration files

- Create read_config_file.py and enter the following content in it:

```
from configparser import ConfigParser
p = ConfigParser()
p.read('read_simple.ini')
print(p.get('bug_tracker', 'url'))
```

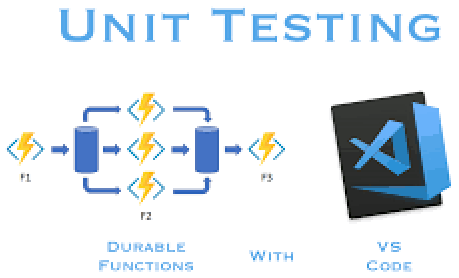
Reading configuration files

- Run read_config_file.py and you will get the following output:

```
$ python3 read_config_file.py
```

Output:

```
https://timesofindia.indiatimes.com/
```



Reading configuration files

```
from configparser import ConfigParser
import glob

p = ConfigParser()
files = ['hello.ini', 'bye.ini', 'read_simple.ini', 'welcome.ini']
files_found = p.read(files)
files_missing = set(files) - set(files_found)
print('Files found:  ', sorted(files_found))
print('Files missing:  ', sorted(files_missing))
```

Reading configuration files

- Run the preceding script and you will get the following output:

```
$ python3 read_many_config_file.py

Output
Files found:   ['read_simple.ini']
Files missing:   ['bye.ini', 'hello.ini', 'welcome.ini']
```

Adding logging and warning code to scripts

- Now, we are going to see a simple logging example.
- We will write a script called logging_example.py and write the following code in it:

```
import logging
LOG_FILENAME = 'log.txt'
logging.basicConfig(filename=LOG_FILENAME,
level=logging.DEBUG,)
logging.debug('This message should go to the log file')
with open(LOG_FILENAME, 'rt') as f:
        prg = f.read()
print('FILE:')
print(prg)
```

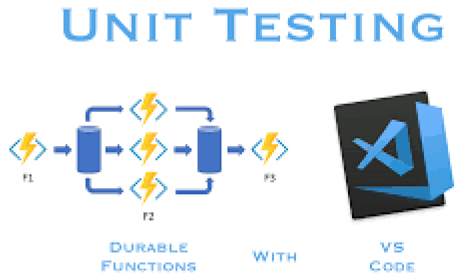
Adding logging and warning code to scripts

- Run the program as follows::

```
$ python3 logging_example.py
```

Output:

```
FILE:
DEBUG:root:This message should go to the log file
```



Adding logging and warning code to scripts

- Check hello.py and you see the debug message printed in that script:

```
$ cat log.txt
```

Output:

```
DEBUG:root:This message should go to the log file
```

Adding logging and warning code to scripts

- Now, we will write a script called logging_warnings_codes.py and write the following code in it:

```
import logging
import warnings
logging.basicConfig(level=logging.INFO,)
warnings.warn('This warning is not sent to the logs')
logging.captureWarnings(True)
warnings.warn('This warning is sent to the logs')
```

Adding logging and warning code to scripts

- Run the script as follows:

```
$ python3 logging_warnings_codes.py

Output:
logging_warnings_codes.py:6: UserWarning: This warning is not sent to the logs
    warnings.warn('This warning is not sent to the logs')
WARNING:py.warnings:logging_warnings_codes.py:10: UserWarning: This warning is sent to the logs
    warnings.warn('This warning is sent to the logs')
```

Generating warnings

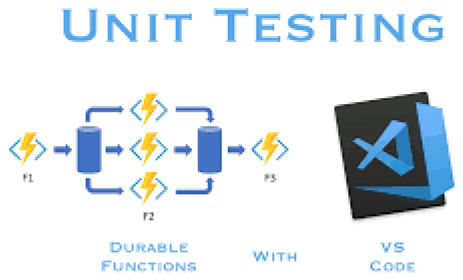
- warn() is used to generate the warnings.
- Now, we will see a simple example of generating warnings.
- Write a script called generate_warnings.py and write a following code in it:

```
import warnings
warnings.simplefilter('error', UserWarning)
print('Before')
warnings.warn('Write your warning message here')
print('After')
```

Generating warnings

- Run the script as follows:

```
$ python3 generate_warnings.py
```



Output:

Before:

```
Traceback (most recent call last):
  File "generate_warnings.py", line 6, in <module>
    warnings.warn('Write your warning message here')
UserWarning: Write your warning message here
```

Putting limits on CPU and memory usage

- In this section, we will learn about how we can limit CPU and memory usage.
- First, we will write a script for putting a limit on CPU usage.
- Create a script called put_cpu_limit.py and write the following code in it:

Refer to the file 4_1.txt

Putting limits on CPU and memory usage

- Run the preceding script as follows:

```
$ python3 put_cpu_limit.py

Output:
Soft limit starts as  : -1
Soft limit changed to : 10
Starting: Thu Sep  6 16:13:20 2018
EXPIRED : Thu Sep  6 16:13:31 2018
(time ran out)
```

Launching webbrowser

- In this section, we will learn about the webbrowser module of Python.
- This module has functions to open URLs in browser applications. We will see a simple example.
- Create a script called open_web.py and write the following code in it:

```
import webbrowser
webbrowser.open('https://timesofindia.indiatimes.com/world')
```

Launching webbrowser

- Run the script as follows:

```
$ python3 open_web.py
```

Output:
Url mentioned in open() will be opened in your browser.
webbrowser – Command line interface

Launching webbrowser

- You can also use the webbrowser module of Python through the command line and can use all of it.

- To use webbrowser through the command line, run the following command:

```
$ python3 -m webbrowser -n https://www.google.com/
```

Using the os module for handling directory and files

Creating and deleting the directory

- In this section, we are going to create a script where we will see what functions we can use for working with the directories on your filesystem, which will include creating, listing, and removing the content.
- Create a script called os_dir_example.py and write the following code in it:

Refer to the file 4_2.txt

Using the os module for handling directory and files

- Run the script as follows:

```
$ python3 os_dir_example.py
```

Output:

```
Creating abcd
Creating abcd/sample_example.txt
Cleaning up
```

Examining the content of a filesystem

- In this section, we will list all of the content of a directory using listdir().

- Create a script called list_dir.py and write the following code in it:

```
import os
import sys
print(sorted(os.listdir(sys.argv[1])))
```

Examining the content of a filesystem

- Run the script as follows:

```
$ python3 list_dir.py /home/student/

['.ICEauthority', '.bash_history', '.bash_logout', '.bashrc',
'.cache', '.config', '.gnupg', '.local', '.mozilla',
'.pam_environment', '.profile', '.python_history', '.ssh',
'.sudo_as_admin_successful', '.viminfo', '1.sh', '1.sh.x',
'1.sh.x.c', 'Desktop', 'Documents', 'Downloads', 'Music',
'Pictures', 'Public', 'Templates', 'Videos', 'examples.desktop'
```

Making backups (with rsync)

- This is the most important work system administrators have to do. In this section, we will learn about making backups using rsync.
- The rsync command is used for copying files and directories locally, as well as remotely, and performing data backups using rsync.
- For that, we are going write a script called take_backup.py and write the following code in it:

Refer to the file 4_3.txt

Making backups (with rsync)

- Run the script as follows:

```
student@ubuntu:~/work$ python3 take_backup.py
Output :
Enter directory to backup
/home/student/work
/home/student/work saved.
Where to backup?
/home/student/Desktop
Doing the backup now!
Do you want to Continue? yes/no
yes
```

Summary

- In this lesson, we learned about how we can automate regular administration tasks.
- We learned about accepting input by various techniques, prompting for passwords at runtime, executing external commands, reading configuration files, adding warnings in your script, launching webbrowser through the script as well as the command line, using the os module to handle files and directories, and making backups.

5. Handling Files, Directories, and Data

Handling Files, Directories, and Data

In this lesson, you will learn about the following:

- Using the os module to work with directories
- Copying, moving, renaming, and deleting data
- Working with paths, directories, and files
- Comparing data
- Merging data
- Pattern matching files and directories
- Metadata: data about data

Using the os module to work with directories

Get the working directory

- Start the python3 console and enter the following commands to get the directory name:

```
$ python3
Python 3.6.5 (default, Apr  1 2018, 05:46:30)
[GCC 7.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import os
>>> os.getcwd()
'/home/student'
```

Changing the directory

- Using the os module, we can change the current working directory.
- For that, the os module has the chdir() function, for example:

```
>>> os.chdir('/home/student/work')
>>> print(os.getcwd())
/home/student/work
>>>
```

Listing files and directories

- Listing the directory contents is easy in Python.
- We are going to use the os module that has a function named listdir(), which will return the names of files and directories from your working directory:

```
>>> os.listdir()
['Public', 'python_learning', '.ICEauthority', '.python_history', 'work',
'.bashrc', 'Pictures', '.gnupg', '.cache', '.bash_logout',
'.sudo_as_admin_successful', '.bash_history', '.config', '.viminfo',
'Desktop', 'Documents', 'examples.desktop', 'Videos', '.ssh',
'Templates', '.profile', 'dir', '.pam_environment', 'Downloads', '.local',
'.dbus', 'Music', '.mozilla']
>>>
```

Renaming a directory

- The os module in Python has a rename() function that helps in changing the name of the directory:

```
>>> os.rename('work', 'work1')
>>> os.listdir()
['Public', 'work1', 'python_learning', '.ICEauthority', '.python_history',
'.bashrc', 'Pictures', '.gnupg', '.cache', '.bash_logout',
'.sudo_as_admin_successful', '.bash_history', '.config', '.viminfo',
'Desktop', 'Documents', 'examples.desktop', 'Videos', '.ssh',
'Templates', '.profile', 'dir', '.pam_environment', 'Downloads', '.local',
'.dbus', 'Music', '.mozilla']
>>
```

Copying, moving, renaming, and deleting data

- We will be learning about the four basic operations that system administrators perform on data, which are copy, move, rename, and delete.

- Python has a built-in module called shutil, which can perform these tasks.

- Using the shutil module, we can perform high-level operations on the data as well.

Copying the data

- In this section, we will see how we can copy files using the shutil module.
- For that, first, we will create a hello.py file and write some text in it.
- hello.py:

```
print ("")
print ("Hello World\n")
print ("Hello Python\n")
```

Copying the data

- Now, we will write the code for copying into the shutil_copy_example.py script. Write the following content in it:

```
import shutil
import os
shutil.copy('hello.py', 'welcome.py')
print("Copy Successful")
```

- Run the script as follows:

```
$ python3 shutil_copy_example.py
```

```
Output:
Copy Successful
```

Moving the data

- Here, we will see how we can move the data. We will use shutil.move() for this purpose. shutil.move(source, destination) will move the file from source to destination.
- Now, we will create a shutil_move_example.py script and write the following content in it:

```
import shutil
shutil.move('/home/student/sample.txt',
'/home/student/Desktop/.')
```

- Run the script as follows:

```
$ python3 shutil_move_example.py
```

Renaming data

- In the previous section, we learned how we can use shutil.move() to move files from source to destination.
- Using shutil.move(), files can be renamed.
- Create a shutil_rename_example.py script and write the following content in it:

```
import shutil
shutil.move('hello.py', 'hello_renamed.py')
```

- Run the script as follows:

```
$ python3 shutil_rename_example.py
Output:
```

Deleting data

- Now, create a os_remove_file_directory.py script and write the following content in it:

```
import os
os.remove('sample.txt')
print("File removed successfully")
os.rmdir('work1')
print("Directory removed successfully")
```

Deleting data

- Run the script as follows:

```
$ python3 os_remove_file_directory.py
```

```
Output:
File removed successfully
Directory removed successfully
```

Copying the data

- In this section, we will see how we can copy files using the shutil module.
- For that, first, we will create a hello.py file and write some text in it.

hello.py:

```
print ("")
print ("Hello World\n")
print ("Hello Python\n")
```

Working with paths

- Start the python3 console:

```
student@ubuntu:~$ python3
Python 3.6.6 (default, Sep 12 2018, 18:26:19)
[GCC 8.0.1 20180414 (experimental) [trunk revision
259383]] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>
```

Working with paths

- os.path.absname(path): Returns the absolute version of your pathname.

```
>>> import os
>>> os.path.abspath('sample.txt')
'/home/student/work/sample.txt’
```

- os.path.dirname(path): Returns the directory name of your path.

```
>>> os.path.dirname('/home/student/work/sample.txt')
'/home/student/work'
```

Working with paths

- os.path.basename(path): Returns the base name of your path.

```
>>> os.path.basename('/home/student/work/sample.txt')
'sample.txt’
```

- os.path.exists(path): Returns True if path refers to the existing path.

```
>>> os.path.exists('/home/student/work/sample.txt')
True
```

Working with paths

- os.path.getsize(path): Returns the size of the entered path in bytes.

```
>>> os.path.getsize('/home/student/work/sample.txt')
39
```

- os.path.isfile(path): Checks whether the entered path is an existing file or not.
- Returns True if it is a file.

```
>>> os.path.isfile('/home/student/work/sample.txt')
True
```

Working with paths

- os.path.isdir(path): Checks whether the entered path is an existing directory or not.

- Returns True if it is a directory.

```
>>> os.path.isdir('/home/student/work/sample.txt')
False
```

Comparing data

- Before starting with the example, make sure you have pandas installed on your system.
- You can install pandas as follows:

```
pip3 install pandas      --- For Python3

or

pip install pandas       --- For python2
```

Comparing data

- Create the student1.csv file content as follows:

```
Id,Name,Gender,Age,Address
101,John,Male,20,New York
102,Mary,Female,18,London
103,Aditya,Male,22,Mumbai
104,Leo,Male,22,Chicago
105,Sam,Male,21,Paris
106,Tina,Female,23,Sydney
```

Comparing data

- Create the student2.csv file content as follows:

```
Id,Name,Gender,Age,Address
101,John,Male,21,New York
102,Mary,Female,20,London
103,Aditya,Male,22,Mumbai
104,Leo,Male,23,Chicago
105,Sam,Male,21,Paris
106,Tina,Female,23,Sydney
```

Comparing data

```
import pandas as pd
df1 = pd.read_csv("student1.csv")
df2 = pd.read_csv("student2.csv")
s1 = set([ tuple(values) for values in df1.values.tolist()])
s2 = set([ tuple(values) for values in df2.values.tolist()])
s1.symmetric_difference(s2)
print (pd.DataFrame(list(s1.difference(s2))), '\n')
print (pd.DataFrame(list(s2.difference(s1))), '\n')
```

Comparing data

- Run the script as follows:

```
$ python3 compare_data.py
Output:
     0     1       2   3        4
0  102  Mary  Female  18   London
1  104   Leo    Male  22  Chicago
2  101  John    Male  20 New York

     0     1       2   3        4
0  101  John    Male  21 New York
1  104   Leo    Male  23  Chicago
2  102  Mary  Female  20   London
```

Merging data

- Now, create a merge_data.py script and write the following code in it:

```
import pandas as pd
df1 = pd.read_csv("student1.csv")
df2 = pd.read_csv("student2.csv")
result = pd.concat([df1, df2])
print(result)
```

```
$ python3 merge_data.py

Output:
    Id    Name  Gender  Age   Address
0  101    John    Male   20  New York
1  102    Mary  Female   18    London
2  103  Aditya    Male   22    Mumbai
3  104     Leo    Male   22   Chicago
4  105     Sam    Male   21     Paris
5  106    Tina  Female   23    Sydney
0  101    John    Male   21  New York
1  102    Mary  Female   20    London
2  103  Aditya    Male   22    Mumbai
3  104     Leo    Male   23   Chicago
4  105     Sam    Male   21     Paris
5  106    Tina  Female   23    Sydney
```

Pattern matching files and directories

- Now, we will look at an example.
- First, create a pattern_match.pyscript and write the following content in it:

```
import glob
file_match = glob.glob('*.txt')
print(file_match)
file_match = glob.glob('[0-9].txt')
print(file_match)
file_match = glob.glob('**/*.txt', recursive=True)
print(file_match)
file_match = glob.glob('**/', recursive=True)
print(file_match)
```

Pattern matching files and directories

- Run the script as follows:

```
$ python3 pattern_match.py

Output:
['file1.txt', 'filea.txt', 'fileb.txt', 'file2.txt', '2.txt', '1.txt',
'file.txt']
['2.txt', '1.txt']
['file1.txt', 'filea.txt', 'fileb.txt', 'file2.txt', '2.txt', '1.txt',
'file.txt', 'dir1/3.txt', 'dir1/4.txt']
['dir1/']
```

Metadata: data about data

- First, we have to install the pyPdf module, as follows:

```
pip install pyPdf
```

Metadata: data about data

```
import pyPdf
def main():
        file_name = '/home/student/sample_pdf.pdf'
        pdfFile = pyPdf.PdfFileReader(file(file_name,'rb'))
        pdf_data = pdfFile.getDocumentInfo()
        print ("----Metadata of the file----")
        for md in pdf_data:
                print (md+ ":" +pdf_data[md])
if __name__ == '__main__':
        main()
```

Metadata: data about data

- Run the script as follows:

```
student@ubuntu:~$ python metadata_example.py
----Metadata of the file----
/Producer:Acrobat Distiller Command 3.0 for SunOS
4.1.3 and later (SPARC)
/CreationDate:D:19980930143358
```

Compressing and restoring

- For that, we are going to write a compress_a_directory.py script and write the following content in it:

```
import shutil
shutil.make_archive('work', 'zip', 'work/’)
```

- Run the script as follows:

```
$ python3 compress_a_directory.py
```

Compressing and restoring

- Now, to restore the data from the compressed file, we are going to use the unpack_archive() function from the shutil module.
- Create an unzip_a_directory.py script and write the following content in it:

```
import shutil
shutil.unpack_archive('work1.zip’)
```

- Run the script as follows:

```
$ python3 unzip_a_directory.py
```

Using the tarfile module to create TAR archives

- Now, create a tarfile_example.py script and write the following content in it:

```
import tarfile
tar_file = tarfile.open("work.tar.gz", "w:gz")
for name in ["welcome.py", "hello.py", "hello.txt",
"sample.txt", "sample1.txt"]:
		tar_file.add(name)
tar_file.close()
```

Using the tarfile module to create TAR archives

- Run the script as follows:

```
$ python3 tarfile_example.py
```

- Now, check your present working directory; you will see work.tar.gz has been created.

Using a tarfile module to examine the contents of TAR files

- Create a examine_tar_file_content.py script and write the following content in it:

```
import tarfile
tar_file = tarfile.open("work.tar.gz", "r:gz")
print(tar_file.getnames())
```

Using a tarfile module to examine the contents of TAR files

- Run the script as follows:

```
$ python3 examine_tar_file_content.py

Output:
['welcome.py', 'hello.py', 'hello.txt', 'sample.txt',
'sample1.txt']
```

Summary

- In this lesson, we learned about Python scripts for handling files and directories.
- We also learned how to use the os module to work with directories.
- We learned how to copy, move, rename, and delete files and directories.
- We also learned about the pandas module in Python, which is used in comparing and merging data.

6. File Archiving, Encrypting, and Decrypting

File Archiving, Encrypting, and Decrypting

In this lesson, we will cover the following topics:

- Creating and unpacking archives
- Tar archives
- ZIP creation
- File encryption and decryption

Creating and unpacking archives(Creating archives)

- Now, we are going to write a script called shutil_make_archive.py and write the following content in it:

```
import tarfile
import shutil
import sys

shutil.make_archive(
            'work_sample', 'gztar',
            root_dir='..',
            base_dir='work',
)
print('Archive contents:')
with tarfile.open('work_sample.tar.gz', 'r') as t_file:
  for names in t_file.getnames():
    print(names)
```

Creating and unpacking archives

- Run the program and you'll get the following output:

```
$ python3 shutil_make_archive.py
Archive contents:
work
work/bye.py
work/shutil_make_archive.py
work/welcome.py
work/hello.py
```

Unpacking archives

- Now, create a script called shutil_unpack_archive.py and write the following code in it:

```
import pathlib
import shutil
import sys
import tempfile
with tempfile.TemporaryDirectory() as d:
 shutil.unpack_archive('work_sample.tar.gz',
extract_dir='/home/student/work',)
 prefix_len = len(d) + 1
 for extracted in pathlib.Path(d).rglob('*'):
 print(str(extracted)[prefix_len:])
```

Unpacking archives

- Run the script as follows:

```
student@ubuntu:~/work$ python3
shutil_unpack_archive.py
```

- Now, check your work/ directory and you will find the work/ folder in it, which will have the extracted files.

Tar archives

- Create a script called check_archive_file.py and write the following content in it:

```
import tarfile

for f_name in ['hello.py', 'work.tar.gz', 'welcome.py',
'nofile.tar', 'sample.tar.xz']:
  try:
    print('{:} {}'.format(f_name, tarfile.is_tarfile(f_name)))
  except IOError as err:
    print('{:} {}'.format(f_name, err))
```

Tar archives

- Run the script and you will get the following output:

```
student@ubuntu:~/work$ python3 check_archive_file.py
hello.py          False
work.tar.gz        True
welcome.py         False
nofile.tar          [Errno 2] No such file or directory:
'nofile.tar'
sample.tar.xz   True
```

Tar archives

- Now, we are going to add a new file into our already created archived file.
- Create a script called add_to_archive.py and write the following code in it:

Refer to the file 6_1.txt

- Run the script and you will get the following output:

Refer to the file 6_2.txt

Tar archives

- Create a script called read_metadata.py and write the following content in it:

```
import tarfile
import time
with tarfile.open('work.tar', 'r') as t:
        for file_info in t.getmembers():
                print(file_info.name)
                print("Size   :", file_info.size, 'bytes')
                print("Type   :", file_info.type)
                print()
```

Tar archives

- Run the script and you will get the following output:

Refer to the file 6_3.txt

Tar archives

- Now, we will extract the contents from an archive using the extractall() function.
- For that, create a script called extract_contents.py and write the following code in it:

```
import tarfile
import os
os.mkdir('work')
with tarfile.open('work.tar', 'r') as t:
        t.extractall('work')
print(os.listdir('work'))
```

Tar archives

- Run the script and you will get the following output:

```
student@ubuntu:~/work$ python3 extract_contents.py
```

- Check your current working directory,and you will find the work/ directory.

- Navigate to that directory and you can find your extracted files.

ZIP creation

- First, we will learn how to create a zip file using the make_archive() function of the shutil module.
- Create a script called make_zip_file.py and write the following code in it:

```
import shutil
shutil.make_archive('work', 'zip', 'work’)
```

- Run the script as follows:

```
student@ubuntu:~$ python3 make_zip_file.py
```

ZIP creation

- Create a script called check_zip_file.py and write the following content in it:

```
import zipfile
for f_name in ['hello.py', 'work.zip', 'welcome.py',
'sample.txt', 'test.zip']:
            try:
                        print('{:}          {}'.format(f_name,
zipfile.is_zipfile(f_name)))
            except IOError as err:
                        print('{:}          {}'.format(f_name, err))
```

ZIP creation

- Run the script as follows:

```
student@ubuntu:~$ python3 check_zip_file.py
Output :
hello.py          False
work.zip          True
welcome.py        False
sample.txt        False
test.zip          True
```

ZIP creation

- Now, we will see how we can read the metadata from an archived ZIP file using the zipfile module of Python.
- Create a script called read_metadata.py and write the following content in it:

Refer to the file 6_4.txt

- Execute the script as follows:

Refer to the file 6_5.txt

File encryption and decryption

- Install pyAesCrypt as follows:

```
pip3 install pyAesCrypt
```

- Create a script called file_encrypt.py and write the following code in it:

Refer to the file 6_6.txt

File encryption and decryption

- Run the script as follows:

```
student@ubuntu:~/work$ python3 file_encrypt.py
Output :
```

- Please check your current working directory.

- You will find the sample.txt.aes encrypted file in it.

File encryption and decryption

```
import pyAesCrypt
from os import stat, remove
bufferSize = 64 * 1024
password = "#Training"
encFileSize = stat("sample.txt.aes").st_size
with open("sample.txt.aes", "rb") as fIn:
 with open("sampleout.txt", "wb") as fOut:
 try:
 pyAesCrypt.decryptStream(fIn, fOut, password, bufferSize,
encFileSize)
 except ValueError:
 remove("sampleout.txt")
```

File encryption and decryption

- Run the script as follows:

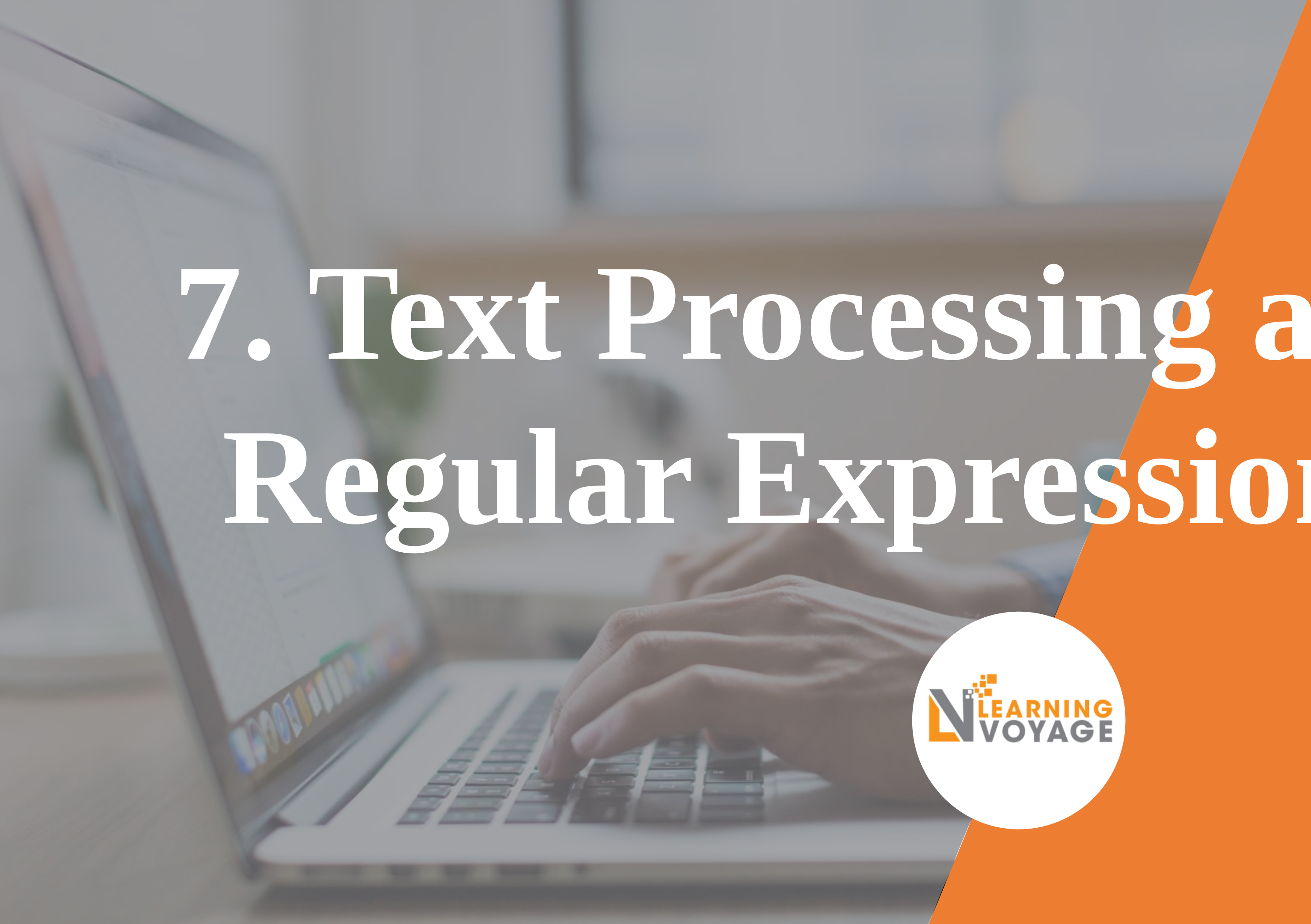
```
student@ubuntu:~/work$ python3 file_decrypt.py
```

- Now, check your current working directory.

- A file named sampleout.txt will be created. That's your decrypted file.

Summary

- In this lesson, we learned about creating and extracting archived files.
- Archiving plays an important role in managing files, directories, and data.
- It also stores the files and directories into a single file.
- We learned in detail about the tarfile and zipfile Python modules that enable you to create, extract, and test archive files.



7. Text Processing and Regular Expressions

Text Processing and Regular Expressions

In this lesson, we will cover the following topics:

- Text wrapping
- Regular expressions
- Unicode strings

Text wrapping

The wrap() function

The wrap() function is used to wrap an entire paragraph in to a single string. The output will be a list of output lines & The syntax is textwrap.wrap(text, width):

- text: Text to wrap.
- width: Maximum length allowed of a wrapped line. The default value is 70.

Text wrapping

- Now, we will see an example of wrap().
- Create a wrap_example.py script and write the following content in it:

```
import textwrap

sample_string = '''Python is an interpreted high-level programming
language for general-purpose programming. Created by Guido van
Rossum and first released in 1991, Python has a design philosophy that
emphasizes code readability, notably using significant whitespace.'''

w = textwrap.wrap(text=sample_string, width=30)
print(w)
```

Text wrapping

- Run the script and you will get the output as follows:

```
student@ubuntu:~/work$ python3 wrap_example.py
['Python is an interpreted high-', 'level programming
language for', 'general-purpose programming.', 'Created
by Guido van Rossum', 'and first released in', '1991,
Python has a design', 'philosophy that emphasizes',
'code readability,  notably', 'using significant
whitespace.']
```

Text wrapping

The fill() function

- The fill() function works similarly to textwrap.wrap, except it returns the data joined into a single, newline-separated string.
- This function wraps the input in text and returns a single string containing the wrapped text.
- The syntax for this function is:

textwrap.fill(text, width)

Text wrapping

- Now, we will see an example of fill(). Create a fill_example.py script and write the following content in it:

```
import textwrap

sample_string = '''Python is an interpreted high-level
programming language.'''

w = textwrap.fill(text=sample_string, width=50)
print(w)
```

Text wrapping

- Run the script and you will get the output as follows:

```
student@ubuntu:~/work$ python3 fill_example.py
Python is an interpreted high-level programming
language.
```

The dedent() function

- The dedent() is another function of the textwrap module.
- This function removes the common leading whitespaces from every line of your text.
- The syntax for this function is as follows:

```
textwrap.dedent(text)
```

The dedent() function

```
import textwrap

str1 = '''
          Hello Python World \tThis is Python 101
          Scripting language\n
          Python is an interpreted high-level programming language for
general-purpose programming.
          '''
print("Original: \n", str1)
print()

t = textwrap.dedent(str1)
print("Dedented: \n", t)
```

Text wrapping

- Run the script and you will get the output as follows:

```
student@ubuntu:~/work$ python3 dedent_example.py

Hello Python World   This is Python 101
Scripting language

Python is an interpreted high-level programming
language for general-purpose programming.
```

The indent() function

- The indent() function is used to add the specified prefix to the beginning of the selected lines in your text.

- The syntax for this function is:

```
textwrap.indent(text, prefix)
```

The indent() function

- Create a indent_example.py script and write the following content in it:

```
import textwrap

str1 = "Python is an interpreted high-level programming language
for general-purpose programming. Created by Guido van Rossum
and first released in 1991, \n\nPython has a design philosophy that
emphasizes code readability, notably using significant whitespace."

w = textwrap.fill(str1, width=30)
i = textwrap.indent(w, '*')
print(i)
```

The indent() function

- Run the script and you will get the output as follows:

```
student@ubuntu:~/work$ python3 indent_example.py
*Python is an interpreted high-
*level programming language for
*general-purpose programming.
*Created by Guido van Rossum
*and first released in 1991,
*Python has a design philosophy
*that emphasizes code
*readability, notably using
*significant whitespace.
```

The shorten() function

- This function of the textwrapmodule is used truncate the text to fit in the specified width.
- For example, if you want to create a summary or preview, use the shorten() function.
- Using shorten(), all the whitespaces in your text will get standardized into a single space & The syntax for this function is:

```
textwrap.shorten(text, width)
```

The shorten() function

- Now we will see an example of shorten().
- Create a shorten_example.py script and write the following content in it:

```
import textwrap

str1 = "Python is an interpreted high-level programming language
for general-purpose programming. Created by Guido van Rossum
and first released in 1991, \n\nPython has a design philosophy that
emphasizes code readability, notably using significant whitespace."

s = textwrap.shorten(str1, width=50)
print(s)
```

The shorten() function

- Run the script and you will get the output as follows:

```
student@ubuntu:~/work$ python3 shorten_example.py
Python is an interpreted high-level [...]
```

Regular expressions

In Python, a regular expression is denoted as re and can be imported through the re module. Regular expressions support four things:

- Identifiers
- Modifiers
- Whitespace characters
- Flags

Regular expressions

Identifier	Description
\w	Matches alphanumeric characters, including underscore (_)
\W	Matches non-alphanumeric characters, excluding underscore (_)
\d	Matches a digit
\D	Matches a non-digit
\s	Matches a space
\S	Matches anything but a space
.	Matches a period (.)
\b	Matches any character except a new line

Regular expressions

Modifier	**Description**
^	Matches start of the string
$	Matches end of the string
?	Matches 0 or 1
*	Matches 0 or more
+	Matches 1 or more
|	Matches either or x/y
[ ]	Matches range
{x}	Amount of preceding code

Regular expressions

Character	Description
\s	Space
\t	Tab
\n	New line
\e	Escape
\f	Form feed
\r	Return

Regular expressions

- The following table lists the flags, and there's a description for each one:

Flag	**Description**
re.IGNORECASE	Case-insensitive matching
re.DOTALL	Matches any character including new lines
re.MULTILINE	Multiline matching
Re.ASCII	Makes escape match only on ASCII characters

The match() function

- The match() function is a function of the re module.
- The match object has two methods:

group(num): Returns an entire match
groups(): Return all matching subgroups in tuple

- The syntax for this function is as follows:

re.match(pattern, string)

The match() function

- Now, we are going see an example of re.match().
- Create a re_match.pyscript and write the following content in it:

```
import re

str_line = "This is python tutorial. Do you enjoy learning python ?"
obj = re.match(r'(.*) enjoy (.*?) .*', str_line)
if obj:
		print(obj.groups())
```

The match() function

- Run the script and you will get the output as follows:

```
student@ubuntu:~/work$ python3 re_match.py
('This is python tutorial. Do you', 'learning')
```

The search() function

- The search() function of the re module will search through a string.
- None if no match found & The match object has two methods:

group(num): Returns an entire match
groups(): Returns all matching subgroups in tuple

- The syntax for this function is as follows:

re.search(pattern, string)

The search() function

- Create a re_search.pyscript and write following content in it:

```
import re

pattern = ['programming', 'hello']
str_line = 'Python programming is fun'
for p in pattern:
	print("Searching for %s in %s" % (p, str_line))
	if re.search(p, str_line):
		print("Match found")
	else:
		print("No match found")
```

The search() function

- Run the script and you will get the output as follows:

```
student@ubuntu:~/work$ python3 re_search.py
Searching for programming in Python programming is
fun
Match found
Searching for hello in Python programming is fun
No match found
```

The findall() function

- This is one of the methods of the match object & the findall() method finds all the matches and then returns them as a list of strings.
- Each element of the list represents as a match & this method searches for the pattern without overlapping.
- Create a re_findall_example.pyscript and write the following content in it:

Refer to the file 7_1.txt

The search() function

- Run the script and you will get the output as follows:

```
student@ubuntu:~/work$ python3
re_findall_example.py
['Red', 'Red']
['per', 'peck', 'peppers', 'peppers', 'per']
['picked', 'pickled', 'pickled', 'pick']
['Hello', 'hello', 'HELLO']
```

The sub() function

- This is one of the most important functions of the re module.
- The sub() is used for replacing the re pattern with the specified replacement.
- It will replace all the occurrences of the re pattern with the replacement string & the syntax is as follows:

```
re.sub(pattern, repl_str, string, count=0)
```

The sub() function

- Now we are going to create a re_sub.pyscript and write the following content in it:

```
import re

str_line = 'Peter Piper picked a peck of pickled peppers. How many pickled
peppers did Peter Piper pick?'

print("Original: ", str_line)
p = re.sub('Peter', 'Mary', str_line)
print("Replaced: ", p)

p = re.sub('Peter', 'Mary', str_line, count=1)
print("Replacing only one occurrence of Peter… ")
print("Replaced: ", p)
```

The sub() function

- Run the script and you will get the output as follows:

```
student@ubuntu:~/work$ python3 re_sub.py
Original:  Peter Piper picked a peck of pickled peppers. How
many pickled peppers did Peter Piper pick?
Replaced:  Mary Piper picked a peck of pickled peppers.
How many pickled peppers did Mary Piper pick?
Replacing only one occurrence of Peter...
Replaced:  Mary Piper picked a peck of pickled peppers.
How many pickled peppers did Peter Piper pick?
```

The sub() function

- Let's look at an the example of subn().
- Create a re_subn.pyscript and write the following content in it:

Refer to the file 7_2.txt

The sub() function

- Run the script and you will get the output as follows:

```
student@ubuntu:~/work$ python3 re_subn.py
str1:-
Original:  Sky is blue. Sky is beautiful.
Replaced:  ('Sky is blue. Sky is stunning.', 1)

str_line:-
Original:  Peter Piper picked a peck of pickled peppers. How many
pickled peppers did Peter Piper pick?
Replaced:  ('Mary Piper picked a peck of pickled peppers. How
many pickled peppers did Mary Piper pick?', 2)
```

Unicode strings

- Python handles Unicode strings in a very easy way.
- The string type actually holds Unicode strings, not a sequence of bytes.
- Start the python3 console in your system and start writing the following:

Refer to the file 7_3.txt

Unicode code point

- In this section, we are going to learn about the unicode code point.
- Python has a powerful built-in function named ord() to get a Unicode code point from a given character.
- So, let's see an example of getting a Unicode code point from a character, as shown in the following code:

Refer to the file 7_4.txt

Encoding

- The transformation from Unicode code point to byte string is known as encoding.
- So, let's see an example of how to encode Unicode code point, as shown in following code:

```
>>> str = u'Office'
>>> enc_str = type(str.encode('utf-8'))
>>> enc_str
<class 'bytes'>
```

Decoding

- The transformation from a byte string to a Unicode code point is known as decoding.
- So, let's see an example of how to decode a byte string to get a Unicode code point as shown in following code:

```
>>> str = bytes('Office', encoding='utf-8')
>>> dec_str = str.decode('utf-8')
>>> dec_str
'Office'
```

Avoiding UnicodeDecodeError

- UnicodeDecodeError occurs whenever byte strings cannot decode to Unicode code points.

- To avoid this exception, we can pass replace, backslashreplace, or ignore to the error argument in decode the as shown here:

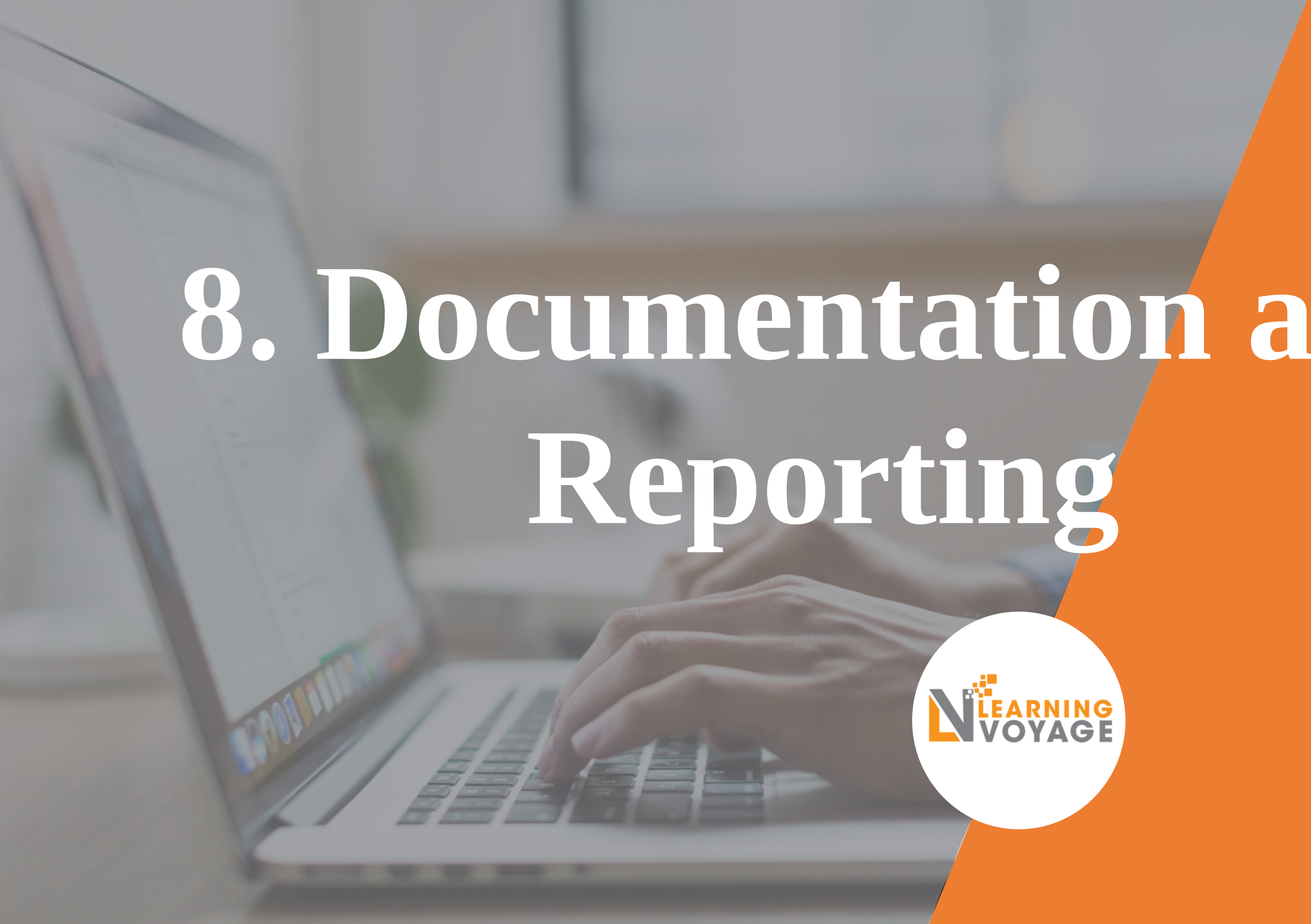
Refer to the file 7_5.txt

Summary

- In this lesson, we learned about regular expressions, using which we can define the rules for a set of strings that we want to match.

- We learned about the four functions of the re module: match(), search(), findall(), and sub().

- We learned about the textwrap module, which is used for formatting and wrapping plain text.



8. Documentation and Reporting

Documentation and Reporting

In this lesson, you will learn about the following:

- Standard input and output
- Information formatting
- Sending emails

Standard input and output

- Now, we will see an example of stdin and stdout.
- For that purpose, create a script,stdin_stdout_example.py, and write the following content in it:

```
import sys
print("Enter number1: ")
a = int(sys.stdin.readline())
print("Enter number2: ")
b = int(sys.stdin.readline())
c = a + b
sys.stdout.write("Result: %d " % c)
```

Standard input and output

- Run the script and you will get the output as follows:

```
student@ubuntu:~/work$ python3
stdin_stdout_example.py
Enter number1:
10
Enter number2:
20
Result: 30
```

Standard input and output

- Syntax:

```
input(prompt)
```

- The input() function returns a string value.
- If you want a number value, simply write the 'int keyword before input(). You can do this as follows:

```
int(input(prompt))
```

Standard input and output

- Similarly, you can write float for float values, Now, we will look at an example.

- Create a input_example.py script and write the following code in it:

Refer to the file 8_1.txt

Standard input and output

- Run the script and you will get the output as follows:

```
student@ubuntu:~/work$ python3 input_example.py
Output:
Enter a string: Hello
Entered string is:  Hello
Enter the value of a: 10
Enter the value of b: 20
Value of c is:  30
Enter num 1: 10.50
Enter num 2: 2.0
Value of num 3 is:  5.25
```

Standard input and output

- We will look at a simple example for the print() function.
- Create a print_example.py script and write the following content in it:

Refer to the file 8_2.txt

Standard input and output

- Run the script and you will get the output as follows:

```
student@ubuntu:~/work$ python3 print_example.py
Hello Python
80
The value of c is:  1.6666666666666667
```

Information formatting

- Create a format_example.pyscript and write the following content in it:

```
# Using single formatter
print("{}, My name is John".format("Hi"))
str1 = "This is John. I am learning {} scripting language."
print(str1.format("Python"))
print("Hi, My name is Sara and I am {} years old !!".format(26))
# Using multiple formatters
str2 = "This is Mary {}. I work at {} Resource department. I am {} years old !!"
print(str2.format("Jacobs", "Human", 30))
print("Hello {}, Nice to meet you. I am {}.".format("Emily", "Jennifer"))
```

Information formatting

- Run the script as follows:

```
student@ubuntu:~/work$ python3 format_example.py
Output:
Hi, My name is John
This is John. I am learning Python scripting language.
Hi, My name is Sara and I am 26 years old !!
This is Mary Jacobs. I work at Human Resource
department. I am 30 years old !!
Hello Emily, Nice to meet you. I am Jennifer.
```

Information formatting

- Now, we will look at an example.
- Create a string_formatting.pyscript and write the following content in it:

Refer to the file 8_3.txt

Information formatting

- Run the script and you will get the output as follows:

```
student@ubuntu:~/work$ python3 string_formatting.py
The values of a and b are 10 30
The value of c is 40
My name is John
The value of z is 351.750000
Normal: Hello, I am Mary !!
Right aligned: Hello, I am      Mary !!
Left aligned: Hello, I am Mary      !!
The truncated string is Exam
Student details: Name:John Address:New York
```

Sending email

- We are going to look at an example.
- In this example, we will send an email containing a simple text from Gmail to the recipients.
- Create a send_email.pyscript and write the following content in it:

Refer to the file 8_4.txt

Sending email

- Run the script as follows:

```
student@ubuntu:~/work$ python3 send_text.py
```

- Output:

```
Password:
Mail sent successfully
```

Sending email

- Now, we will look at one more example of sending an email with an attachment.
- In this example, we are going to send an image to the recipient.
- We are going to send this mail via Gmail.
- Create a send_email_attachment.pyscript and write the following content in it:

Refer to the file 8_5.txt

Sending email

- Run the script as follows:

```
student@ubuntu:~/work$ python3
send_email_attachment.py
```

- Output:

```
Password:
Email with attachment sent successfully!!
```

Summary

- In this lesson, we learned about standard input and output.
- We learned how stdin and stdout act as keyboard input and user's Terminal respectively.
- We also learned about input() and print() functions.
- In addition to this, we learned about sending an email from Gmail to the receivers.

9. Working with Various Files

Working with Various Files

In this lesson, the following topics will be covered:

- Working with PDF files
- Working with Excel files
- Working with CSV files
- Working with txt files

Working with PDF files

- We must install this module first. To install PyPDF2, run the following command in your Terminal:

```
pip3 install PyPDF2
```

- Now, we are going to look at some of the operations to work on PDF files, such as reading a PDF, getting the number of pages, extracting text, and rotating PDF pages.

Reading a PDF document & getting the number of pages

- Right now, I have the test.pdf file present in my system so I will use this file throughout this section.
- Enter your PDF filename in place of test.pdf. Create a script called read_pdf.py and write the following content in it:

```
import PyPDF2

with open('test.pdf', 'rb') as pdf:
    read_pdf= PyPDF2.PdfFileReader(pdf)
    print("Number of pages in pdf : ", read_pdf.numPages)
```

Reading a PDF document & getting the number of pages

- Run the script and you will get the following output:

```
student@ubuntu:~/work$ python3 read_pdf.py
```

- Following is the output:

```
Number of pages in pdf :  20
```

Extracting text

- To extract the pages of the pdf file, the PyPDF2 module has the extractText() method.
- Create a script called extract_text.py and write the following content in it:

```
import PyPDF2
with open('test.pdf', 'rb') as pdf:
    read_pdf = PyPDF2.PdfFileReader(pdf)
    pdf_page = read_pdf.getPage(1)
    pdf_content = pdf_page.extractText()
    print(pdf_content)
```

Extracting text

- Run the script and you will get the following output:

```
student@ubuntu:~/work$ python3 extract_text.py
```

- Following is the output:

Refer to the file 9_1.txt

Rotating PDF pages

- In this section, we are going to see how to rotate PDF pages.
- For that, we will use the rotate.Clockwise() method of a PDF object.
- Create a script called rotate_pdf.py and write the following content in it:

Refer to the file 9_2.txt

Rotating PDF pages

- Run the script and you will get the following output:

```
student@ubuntu:~/work$ python3 rotate_pdf.py
```

- Following is the output:

```
pdf successfully rotated
```

Working with Excel files

Using the xlrd module

- First, we have to install the xlrd module.
- Run the following command in your Terminal to install the xlrd module:

```
pip3 install xlrd
```

Working with Excel files

Reading an Excel file

- Create a script called read_excel.py and write the following content in it:

```
import xlrd

excel_file = (r"/home/student/sample.xlsx")
course_obj = xlrd.open_workcourse(excel_file)
excel_sheet = course_obj.sheet_by_index(0)
result = excel_sheet.cell_value(0, 1)
print(result)
```

Working with Excel files

- Run the script and you will get the following output:

```
student@ubuntu:~$ python3 read_excel.py
```

- Following is the output:

```
First Name
```

Extracting the names of columns

- Create a script called extract_column_names.py and write the following content in it:

```
import xlrd

excel_file = ("/home/student/work/sample.xlsx")
course_obj = xlrd.open_workcourse(excel_file)
excel_sheet = course_obj.sheet_by_index(0)
excel_sheet.cell_value(0, 0)
for i in range(excel_sheet.ncols):
		print(excel_sheet.cell_value(0, i))
```

Working with Excel files

- Run the script and you will get the following output:

```
student@ubuntu:~/work$ python3
extract_column_names.py
```

- Following is the output:

```
Id
First Name
Last Name
Gender
Age
Country
```

Using pandas

- Before proceeding to read Excel files using Pandas, first we have to install the pandas module.

- We can install pandas using the following command:

```
pip3 install pandas
```

Reading an Excel file

- Create a script called rd_excel_pandas.py and write the following content in it:

```
import pandas as pd

excel_file = 'sample.xlsx'
df = pd.read_excel(excel_file)
print(df.head())
```

Working with Excel files

- Run the preceding script and you will get the following output:

```
student@ubuntu:~/test$ python3 rd_excel_pandas.py
```

- Following is the output:

```
OrderDate     Region  ...   Unit Cost     Total
0  2014-01-09   Central  ...     125.00       250.00
1   6/17/15     Central     ...  125.00      625.00
2  2015-10-09   Central     ...     1.29         9.03
3  11/17/15     Central   ...      4.99       54.89
4  10/31/15     Central   ...      1.29       18.06
```

Reading specific columns in an Excel file

- Now, let’s look at an example to read specific columns in an Excel file.
- Create a script called rd_excel_pandas1.py and write the following content in it:

```
import pandas as pd

excel_file = 'sample.xlsx'
cols = [1, 2, 3]
df = pd.read_excel(excel_file , sheet_names='sheet1',
usecols=cols)
print(df.head())
```

Reading specific columns in an Excel file

- Run the preceding script and you will get the following output:

```
student@ubuntu:~/test$ python3 rd_excel_pandas1.py
```

- Following is the output:

```
Region      Rep    Item
0  Central    Smith    Desk
1  Central   Kivell    Desk
2  Central     Gill  Pencil
3  Central  Jardine  Binder
4  Central  Andrews  Pencil
```

Using openpyxl

- openpyxl is a Python library that's used to read and write xlsx, xlsm, xltx, and xltm files.

- First, we have to install openpyxl.

- Run the following command:

```
pip3 install openpyxl
```

Creating a new Excel file

```
from openpyxl import Workcourse

course_obj = Workcourse()
excel_sheet = course_obj.active
excel_sheet['A1'] = 'Name'
excel_sheet['A2'] = 'student'
excel_sheet['B1'] = 'age'
excel_sheet['B2'] = '24'

course_obj.save("test.xlsx")
print("Excel created successfully")
```

Working with Excel files

- Run the script and you will get the following output:

```
student@ubuntu:~/work$ python3 create_excel.py
```

- Following is the output:

```
Excel created successfully
```

Appending values

- In this section, we are going to append values in Excel, For that, we are going to use the append() method.
- We can add a group of values at the bottom of the current sheet in which we want to put the values.
- Create a script called append_values.py and write the following content in it:

Refer to the file 9_3.txt

Working with Excel files

- Run the script and you will get the following output:

```
student@ubuntu:~/work$ python3 append_values.py
```

- Following is the output:

```
values are successfully appended
```

Reading multiple cells

- Create a script called read_multiple.py and write the following content in it:

```
import openpyxl

course_obj = openpyxl.load_workcourse('sample.xlsx')
excel_sheet = course_obj.active
cells = excel_sheet['A1': 'C6']
for c1, c2, c3 in cells:
        print("{0:6} {1:6} {2:6}".format(c1.value, c2.value, 
c3.value))
```

Working with Excel files

- Run the script and you will get the following output:

```
student@ubuntu:~/work$ python3 read_multiple.py
```

- Following is the output:

```
Id      First Name Last Name
   101 John   Smith
   102 Mary   Williams
   103 Rakesh Sharma
   104 Amit   Roy
   105 Sandra Ace
```

Working with CSV files

- The CSV format stands for Comma Separated Values & the commas are used to separate the fields in a record.
- These are commonly used for importing and exporting the format for spreadsheets and databases.
- A CSV file is a plain text file that uses a specific type of structuring to arrange tabular data.
- Python has the built-in csv module that allows Python to parse these types of files

Reading a CSV file

- Create a script called csv_read.py and write the following content in it:

```
import csv

csv_file = open('test.csv', 'r')
with csv_file:
    read_csv = csv.reader(csv_file)
    for row in read_csv:
        print(row)
```

Working with CSV files

- Run the script and you will get the following output:

```
student@ubuntu:~$ python3 csv_read.py
```

- Following is the output:

Refer to the file 9_4.txt

Writing into a CSV file

- To write data in a csv file, we use the csv.writer module.
- In this section, we will store some data into the Python list and then put that data into the csv file.
- Create a script called csv_write.py and write the following content in it:

Refer to the file 9_5.txt

Working with CSV files

- Run the script and you will get the following output:

```
student@ubuntu:~$ python3 csv_write.py
```

- Following is the output:

```
[['Name', 'Sport'], ['Andres Iniesta', 'Football'], ['AB de Villiers', 'Cricket'], ['Virat Kohli', 'Cricket'], ['Lionel Messi', 'Football']]
```

Working with txt files

- A plain text file is used to store data that represents only characters or strings and doesn't consider any structured metadata.
- In Python, there's no need to import any external library to read and write text files.
- Python provides an built-in function to create, open, close, and write and read text files.
- To do the operations, there are different access modes to govern the type of operation possible in an opened file.

The open() function

- This function is used to open a file and does not require any external module to be imported.

- The syntax is as follows:

Name_of_file_object = open("Name of file","Access_Mode")

The open() function

- For the preceding syntax, the file must be in the same directory that our Python program resides in.
- If the file is not in the same directory, then we also have to define the file path while opening the file.
- The syntax for such a condition is shown here:

```
Name_of_file_object = open("/home/……/Name of file","Access_Mode")
```

File opening

- The file is in the same directory as the append mode:

```
text_file = open("test.txt","a")
```

- If the file is not in the same directory, we have to define the path in the append mode:

```
text_file = open("/home/…../test.txt","a")
```

The close() function

- The syntax is as follows:

```
Name_of_file_object.close()
```

- The following code syntax can be use to simply open and close a file:

```
#Opening and closing a file test.txt:
text_file = open("test.txt","a")
text_file.close()
```

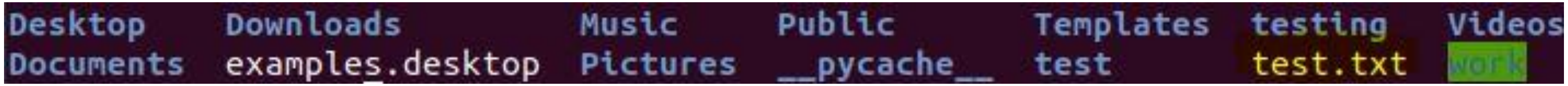
Writing a text file

- Create a script called text_write.py and write the following content in it:

```
text_file = open("test.txt", "w")
text_file.write("Monday\nTuesday\nWednesday\nThursday\nFriday\nSaturday\n")
text_file.close()
```

Writing a text file

- Run the preceding script and you will get the output as follows:



```
Desktop    Downloads         Music     Public       Templates  testing   Videos
Documents  examples.desktop  Pictures  __pycache__  test       test.txt  work
```

Reading a text file

- To read the data from this file, we use the read() method of the file handle object.
- Create a script called text_read.py and write the following content in it:

```
text_file = open("test.txt", "r")
data = text_file.read()
print(data)
text_file.close()
```

Reading a text file

- Following is the output:

```
student@ubuntu:~$ python3 text_read.py
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
```

Summary

- In this lesson, we learned about various files.
- We learned about PDF, Excel, CSV, and text files.
- We used Python modules to perform some operations on these types of files.

10. Basic Networking - Socket Programming

Basic Networking - Socket Programming

In this lesson, you will learn about the following:

- Sockets
- The http package
- The ftplib module
- The urllib package

Sockets

- Let's see how to create a socket and the socket class has a socket function, with two arguments: address_family and socket type.

- The following is the syntax:

```
import socket
 s = socket.socket(address_family, socket type)
```

Sockets

- Sockets can be configured as server and client sockets.
- When both TCP/IP sockets are connected, communication will be bi-directional.
- Now we are going explore an example of client-server communication.
- We will create two scripts: server.py and client.py and the server.py script is as follows:

Refer to the file 10_1.txt

Sockets

- Now we will write a script for the client.

- The client.py script is as follows:

Refer to the file 10_2.txt

Sockets

Terminal 1:python3 server.py	**Terminal 2:**python3 client.py
student@ubuntu:~/work$ python3 server.py Connection from: ('127.0.0.1', 35120) from connected user: Hello from client -> Hello from server !	student@ubuntu:~/work$ python3 client.py -> Hello from client Received from server: Hello from server ! ->

The http package

- http.client: This is a low-level HTTP protocol client
- http.server: This contains basic HTTP server classes
- http.cookies: This is used for implementing state management with cookies
- http.cookiejar: This module provides cookie persistence

The http.client module

- First, we are going explore an example of making an http connection.
- For that, create a make_connection.py script and write the following content in it:

```
import http.client

con_obj = http.client.HTTPConnection('Enter_URL_name',
80, timeout=20)
print(con_obj)
```

The http.client module

- Run the script and you will get the output as follows:

```
student@ubuntu:~/work$ python3 make_connection.py
<http.client.HTTPConnection object at
0x7f2c365dd898>
```

The http.client module

```
import http.client

con_obj = http.client.HTTPSConnection("www.imdb.com")
con_obj.request("GET", "/")
response = con_obj.getresponse()

print("Status: {}".format(response.status))

headers_list = response.getheaders()
print("Headers: {}".format(headers_list))

con_obj.close()
```

The http.client module

- The output should be as follows:

Refer to the file 10_3.txt

The http.client module

- Run the script as follows:

```
student@ubuntu:~/work$ python3 post_example.py
```

- You should get the following output:

Refer to the file 10_4.txt

The http.server module

- First, navigate to your desired directory and run the following command:

```
student@ubuntu:~/Desktop$ python3 -m http.server 9000
```

- Now open your browser and write localhost:9000 in your address bar and press Enter.
- You will get the output following:

Refer to the file 10_5.txt

The ftplib module

- ftplib is a module in Python that provides all the functionality needed to perform various actions over the FTP protocol. ftplib contains the FTP client class, as well as some helper functions.
- Using this module, we can easily connect to an FTP server to retrieve multiple files and process them.
- By importing the ftplib module, we can use all the functionality it provides.

Downloading files

- In this section, we are going to learn about downloading files from another machine using ftplib.
- For that, create a get_ftp_files.py script and write the following content in it:

Refer to the file 10_6.txt

Downloading files

- Run the script as follows:

```
student@ubuntu:~/work$ python3 get_ftp_files.py
```

- You should get the following output:

```
Downloading...hello
Downloading...hello.c
Downloading...sample.txt
Downloading...strip_hello
Downloading...test.py
```

Getting a welcome message using getwelcome():

- Now we will see an example of getwelcome().
- Create a get_welcome_msg.py script and write the following content in it:

```
from ftplib import FTP
ftp = FTP('your-ftp-domain-or-ip')
ftp.login('your-username','your-password')
welcome_msg = ftp.getwelcome()
print(welcome_msg)
ftp.close()
```

Getting a welcome message using getwelcome():

- Run the script as follows:

```
student@ubuntu:~/work$ python3 get_welcome_msg.py
220 (vsFTPd 3.0.3)
```

Sending commands to the server using the sendcmd()

- In this section, we are going to learn about the sendcmd() function.
- We can use the sendcmd() function to send a simple string command to the server to get the String response.
- The client can send FTP commands such as STAT, PWD, RETR, and STOR.
- The ftplib module has multiple methods that can wrap these commands.

Sending commands to the server using the sendcmd()

- Create a send_command.py script and write the following content in it:

Refer to the file 10_7.txt

The http.client module

- Run the script as follows:

```
student@ubuntu:~/work$ python3 send_command.py
```

- You will get the following output:

Refer to the file 10_8.txt

The urllib package

urllib has a few different modules, which are listed here:

- urllib.request: This is used for opening and reading URLs.
- urllib.error: This contains exceptions raised by urllib.request.
- urllib.parse: This is used for parsing URLs.
- urllib.robotparser: This is used for parsing robots.txt files.

The urllib package

- Create aurl_requests_example.py script and write the following content in it:

```
import urllib.request
x = urllib.request.urlopen('https://www.imdb.com/')
print(x.read())
```

- Run the script as follows:

```
student@ubuntu:~/work$ python3 url_requests_example.py
```

The urllib package

- Here is the output:

Refer to the file 10_9.txt

Python urllib response headers

- Create a url_response_header.py script and write the following content in it:

```
import urllib.request

x = urllib.request.urlopen('https://www.imdb.com/')
print(x.info())
```

The urllib package

- Run the script as follows:

```
student@ubuntu:~/work$ python3 url_response_header.py
```

- Here is the output:

Refer to the file 10_10.txt

Summary

- In this lesson, we learned about sockets, which are used for bi-directional client-server communication.
- We learned about three internet modules: http, ftplib, and urllib.
- The http package has modules for the client and server: http.client and http.server respectively.
- Using ftplib, we downloaded files from another machine.

11. Handling Emails Using Python Scripting

Handling Emails Using Python Scripting

In this lesson, you'll learn about the following:

- Email message format
- Adding HTML and multimedia content
- POP3 and IMAP servers

Email message format

In this section, we're going to learn about the email message format. Email messages consist of three primary components:

- The receiver's email address
- The sender's email address
- The message

Email message format

- Now, we're going to see a simple example of sending a plain text email from your Gmail address, in which you'll learn about writing an email message and sending it.

- Now, create a script, write_email_message.py, and write the following content in it:

Refer to the file 11_1.txt

Email message format

- Run the script and you'll get the following output:

```
student@ubuntu:~/work/lesson_11$ python3
write_email_message.py
Output:
Password:
Mail sent successfully
```

Adding HTML and multimedia content

- First, we'll see how we can add HTML content.
- For that, create a script, add_html_content.py, and write the following content in it:

Refer to the file 11_2.txt

Adding HTML and multimedia content

- Run the script and you'll get the following output:

```
student@ubuntu:~/work/lesson_11$ python3
add_html_content.py
Output:
Password:
Mail sent successfully !!
```

Adding HTML and multimedia content

- Now, we'll see how we can add an attachment and send it through a Python script.

- For that, create a script, add_attachment.py, and write the following content in it:

Refer to the file 11_3.txt

Adding HTML and multimedia content

- Run the script and you'll get the output as follows:

```
student@ubuntu:~/work/lesson_11$ python3
add_attachment.py
Output:
Password:
Attachment sent successfully !!
```

POP3 and IMAP servers

Receiving email using the poplib library

The POP3 protocol works on two ports:

- Port 110: The default non-encrypted port
- Port 995: The encrypted port

POP3 and IMAP servers

- Now, we'll see some examples.
- First, we'll see an example where we get a number of emails.
- For that, create a script, number_of_emails.py, and write the following content in it:

Refer to the file 11_4.txt

POP3 and IMAP servers

- Run the script, as follows:

```
student@ubuntu:~$ python3 number_of_emails.py
```

POP3 and IMAP servers

- Now, we're going to write a script to get the latest email.
- For that, create a script, latest_email.py, and write the following content in it:

Refer to the file 11_5.txt

- Run the script, as follows:

```
student@ubuntu:~$ python3 latest_email.py
```

POP3 and IMAP servers

- Now, we're going to write a script to get all of the emails.
- For that, create a script, all_emails.py, and write the following content in it:

Refer to the file 11_6.txt

- Run the script, as follows:

```
student@ubuntu:~$ python3 latest_email.py
```

POP3 and IMAP servers

Receiving email using the imaplib library

The IMAP protocol works on two ports:

- Port 143: The default non-encrypted port
- Port 993: The encrypted port

POP3 and IMAP servers

- Now, we're going to see an example using the imaplib library.
- Create a script, imap_email.py, and write the following content in it:

- Refer to the file 11_7.txt

- Run the script, as follows:

```
student@ubuntu:~$ python3 imap_email.py
```

Summary

- In this lesson, we learned about how to write an email message in a Python script.
- We also learned about the Python smtplib module, which is used for sending and receiving emails via Python scripts.
- We also learned about how to receive emails through POP3 and IMAP protocols.
- Python supplies the poplib and imaplib libraries with which we can perform tasks.

12. Remote Monitoring of Hosts Over Telnet and SSH

Remote Monitoring of Hosts Over Telnet and SSH

In this lesson, we will cover the following topics:

- The telnetlib() module
- The subprocess.Popen() module
- SSH using fabric module
- SSH using Paramiko library
- SSH using Netmiko library

The telnetlib() module

- Telnet uses TCP on the default port number 23.
- To use Telnet, make sure it is installed on your system & If not, run the following command to install it:

```
$ sudo apt-get install telnetd
```

- To run Telnet using the simple Terminal, you just have to enter the following command:

```
$ telnet ip_address_of_your_remote_server
```

The telnetlib() module

- Python has the telnetlib module to perform Telnet functions through Python scripts.
- Before telnetting your remote device or router, make sure they are configured properly and, if not, you can do basic configuration by using the following command in the router's Terminal:

Refer to the file 12_1.txt

The telnetlib() module

- Now, let's see the example of Telnetting a remote device.
- For that, create a telnet_example.pyscript and write following content in it:

Refer to the file 12_2.txt

The telnetlib() module

- Run the script and you will get the output as follows:

Refer to the file 12_3.txt

SSH

- User who accesses a remote server or device must install an SSH client.
- Install SSH by running the following command in your Terminal:

```
$ sudo apt install ssh
```

SSH

- Also, on a remote server where the user wants to communicate, an SSH server must be installed and running.
- SSH uses the TCP protocol and works on port number 22 by default.
- You can run the ssh command through the Terminal as follows:

```
$ ssh host_name@host_ip_address
```

The subprocess.Popen() module

- Now, let's see some useful arguments of subprocess.Popen():

```
class subprocess.Popen(args, bufsize=0, executable=None,
stdin=None, stdout=None,
                                        stderr=None, preexec_fn=None,
close_fds=False, shell=False,
                                        cwd=None, env=None,
universal_newlines=False,
                startupinfo=None, creationflags=0)
```

The subprocess.Popen() module

- Now, we are going to see an example of subprocess.Popen().
- For that, create a ssh_using_sub.py script and write the following content in it:

Refer to the file 12_4.txt

The subprocess.Popen() module

- Run the script and you will get the output as follows:

```
student@ubuntu:~$ python3 ssh_using_sub.py
Output :
student@192.168.0.106's password:
[b'Desktop\n', b'Documents\n', b'Downloads\n',
b'examples.desktop\n', b'Music\n', b'Pictures\n', b'Public\n',
b'sample.py\n', b'spark\n', b'spark-2.3.1-bin-hadoop2.7\n',
b'spark-2.3.1-bin-hadoop2.7.tgz\n', b'ssh\n', b'Templates\n',
b'test_folder\n', b'test.txt\n', b'Untitled1.ipynb\n',
b'Untitled.ipynb\n', b'Videos\n', b'work\n']
```

SSH using fabric module

- To use fabric module, first you have to install it using the following command:

```
$ pip3 install fabric3
```

- Now, we will see an example, Create a fabfile.pyscript and write the following content in it:

Refer to the file 12_5.txt

SSH using fabric module

- Run the script and you will get the output as follows:

```
student@ubuntu:~$ fab dir
Output:
[student@192.168.0.106] Executing task 'dir'
[student@192.168.0.106] run: mkdir fabric

Done.
Disconnecting from 192.168.0.106... done.
```

SSH using the Paramiko library

- Before using Paramiko, make sure you have installed it properly on your system.

- If it is not installed, you can install it by running the following command in your Terminal:

```
$ sudo pip3 install paramiko
```

SSH using the Paramiko library

- Before we are going to do SSH to your remote device or multi-layer router, make sure they are configured properly and, if not, you can do basic configuration by using the following command in a multi-layer router Terminal:

Refer to the file 12_6.txt

SSH using the Paramiko library

- Now, create a pmiko.pyscript and write the following content in it:

Refer to the file 12_7.txt

SSH using the Paramiko library

- Run the script and you will get the output as follows:

```
student@ubuntu:~$ python3 pmiko.py
Output:
SSH connection is successfuly established with
192.168.0.70
Creating VLAN 2
Creating VLAN 3
Creating VLAN 4
Creating VLAN 5
```

SSH using the Netmiko library

- Now, let's see an example.
- Create a nmiko.pyscript and write the following code in it:

Refer to the file 12_8.txt

SSH using the Netmiko library

- Run the script and you will get the output as follows:

Refer to the file 12_9.txt

Summary

- In this lesson, you learned about Telnet and SSH.
- You also learned the different Python modules such as telnetlib, subprocess, fabric, Netmiko, and Paramiko, using which we perform Telnet and SSH.
- SSH uses the public key encryption for security purposes and is more secure than Telnet.

13. Working with Apache and Other Log Files

Working with Apache and Other Log Files

In this lesson, you will learn the following:

- Parsing complex log files
- The need for exceptions
- Tricks for parsing different files
- Error log
- Access log
- Parsing other log files

Parsing complex log files

- let's create a read_apache_log.pyscript and write the following content in it:

```
def read_apache_log(logfile):
        with open(logfile) as f:
                log_obj = f.read()
                print(log_obj)

if __name__ == '__main__':
        read_apache_log("access.log")
```

Parsing complex log files

- Run the script and you will get the output as follows:

Refer to the file 13_1.txt

Parsing complex log files

- After reading log entries in the access.log file, now we are going to parse the IP addresses from the log file.

- For that, create a parse_ip_address.pyscript and write the following content in it:

Refer to the file 13_2.txt

Parsing complex log files

- Run the script and you will get the output as follows:

Refer to the file 13_3.txt

The need for exceptions

Analyzing exceptions

- Consider the following example:

```
f = open('logfile', 'r')
print(f.read())
f.close()
```

The need for exceptions

- After running the program, you get the output as follows:

```
Traceback (most recent call last):
  File "sample.py", line 1, in <module>
    f = open('logfile', 'r')
FileNotFoundError: [Errno 2] No such file or directory: 'logfile'
```

The need for exceptions

- Consider the following example:

```
try:
    f = open('logfile', 'r')
    print(f.read())
    f.close()
except:
    print("file not found. Please check whether the file is
present in your directory or not.")
```

The need for exceptions

- After running the program, you get the output as follows:

```
file not found. Please check whether the file is present
in your directory or not.
```

Tricks for parsing different files

- Remember the log files can be either plain text or compressed.
- All the Log files have a .log extension for a plain text file and log.bz2 for a bzip2 file.
- You should process the set of files based on their name.
- All the parsing of log files must be combined into a single report.

Error log

In this section, we are going to learn about the error log. The related directives for the error log are as follows:

- ErrorLog
- LogLevel

Error log

- The error log file is not customizable. The entries in the error log that deals with the requests will have corresponding entries in the access log.
- You should always monitor the error log for the problems during testing.
- On Unix systems, you can run the following command to accomplish this:

```
$ tail -f error_log
```

Access log

- Storing the information in the access log means starting log management.
- The next step will be analyzing the information that helps us get useful statistics.
- The Apache httpd has various versions, and these versions have used some other modules and directives to control access logging.
- You can configure the format of the access log & this format is specified using a format string.

Common log format

- In this section, we are going to learn about common log format.
- The following syntax shows the configuration for the access log:

```
LogFormat "%h %l %u %t \"%r\" %>s %b" nick_name
        CustomLog logs/access_log nick_name
```

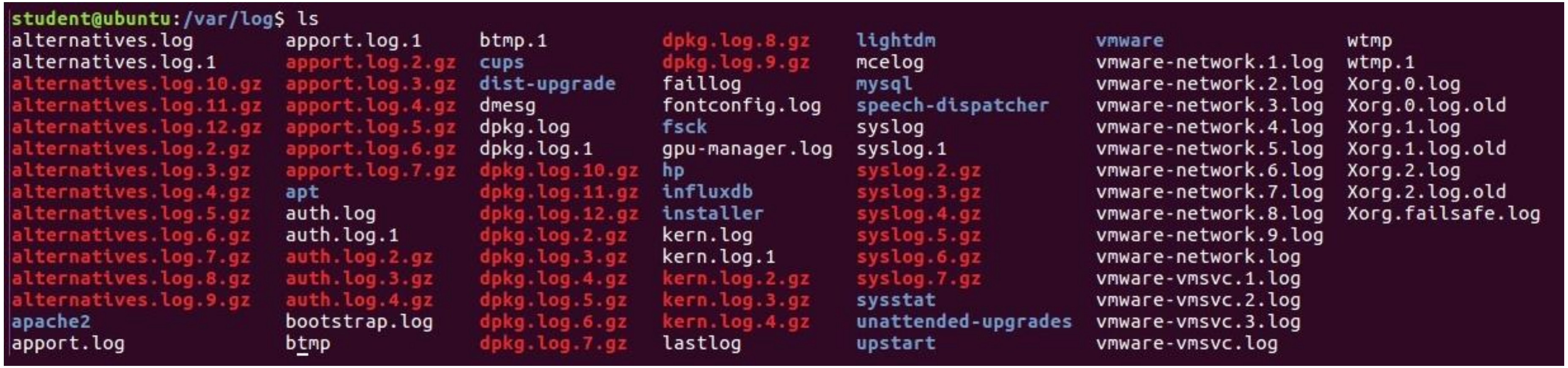
Common log format

- %t: This term is used to detect the time at which the processing request of server is finished.

- The format is as follows:

[day/month/year:hour:minute:second zone]

Parsing other log files



```
student@ubuntu:/var/log$ ls
alternatives.log        apport.log.1     btmp.1          dpkg.log.8.gz     lightdm              vmware                wtmp
alternatives.log.1      apport.log.2.gz  cups            dpkg.log.9.gz     mcelog               vmware-network.1.log  wtmp.1
alternatives.log.10.gz  apport.log.3.gz  dist-upgrade    faillog           mysql                vmware-network.2.log  Xorg.0.log
alternatives.log.11.gz  apport.log.4.gz  dmesg           fontconfig.log    speech-dispatcher    vmware-network.3.log  Xorg.0.log.old
alternatives.log.12.gz  apport.log.5.gz  dpkg.log        fsck              syslog               vmware-network.4.log  Xorg.1.log
alternatives.log.2.gz   apport.log.6.gz  dpkg.log.1      gpu-manager.log   syslog.1             vmware-network.5.log  Xorg.1.log.old
alternatives.log.3.gz   apport.log.7.gz  dpkg.log.10.gz  hp                syslog.2.gz          vmware-network.6.log  Xorg.2.log
alternatives.log.4.gz   apt              dpkg.log.11.gz  influxdb          syslog.3.gz          vmware-network.7.log  Xorg.2.log.old
alternatives.log.5.gz   auth.log         dpkg.log.12.gz  installer         syslog.4.gz          vmware-network.8.log  Xorg.failsafe.log
alternatives.log.6.gz   auth.log.1       dpkg.log.2.gz   kern.log          syslog.5.gz          vmware-network.9.log
alternatives.log.7.gz   auth.log.2.gz    dpkg.log.3.gz   kern.log.1        syslog.6.gz          vmware-network.log
alternatives.log.8.gz   auth.log.3.gz    dpkg.log.4.gz   kern.log.2.gz     syslog.7.gz          vmware-vmsvc.1.log
alternatives.log.9.gz   auth.log.4.gz    dpkg.log.5.gz   kern.log.3.gz     sysstat              vmware-vmsvc.2.log
apache2                 bootstrap.log    dpkg.log.6.gz   kern.log.4.gz     unattended-upgrades  vmware-vmsvc.3.log
apport.log              btmp             dpkg.log.7.gz   lastlog           upstart              vmware-vmsvc.log
```

Parsing other log files

- Let's see an example for parsing one of the log files from before.
- Create a simple_log.pyscript and write the following content in it:

```
f=open('/var/log/kern.log','r')

lines = f.readlines()
for line in lines:
        kern_log = line.split()
        print(kern_log)
f.close()
```

The need for exceptions

- Run the script and you will get the output as follows:
- Refer to the file 13_4.txt

The need for exceptions

- let's see an example of such a condition.
- Create a simple_log1.py script and put the following script in it:

```
f=open('/var/log/kern.log','r')

lines = f.readlines()
for line in lines:
        kern_log = line.split()[1:3]
        print(kern_log)
```

- Run the script and you will get the following output:

```
student@ubuntu:~$ python3 simple_log1.py
Output:
['26', '14:37:20']
['26', '14:37:20']
['26', '14:37:32']
['26', '14:39:38']
['26', '14:39:38']
['26', '14:39:38']
['26', '14:39:38']
['26', '14:39:38']
['26', '14:39:38']
['26', '14:39:38']
['26', '14:39:38']
['26', '14:39:38']
```

Summary

- In this lesson, you learned about how to work with different types of log files.

- You also learned about parsing complex log files and the need for exceptions while handling these files.

- The tricks for parsing log files will help in performing the parsing smoothly.

- You also learned about ErrorLog and AccessLog.

14. SOAP and REST API Communication

SOAP and REST API Communication

In this lesson, you will learn the following:

- What is SOAP?
- Using libraries for SOAP
- What is a RESTful API?
- Using standard libraries for RESTful API
- Working with JSON data

What is SOAP?

- SOAP is Simple Object Access Protocol. SOAP is the standard communication protocol system that permits processes to use different operating systems.
- These communicate via HTTP and XML. It is a web services technology.
- SOAP APIs are mainly designed for tasks such as creating, updating, deleting, and recovering data.

Using libraries for SOAP

There are various libraries used for SOAP listed here:

- SOAPpy
- Zeep
- Ladon
- suds-jurko
- pysimplesoap

Using libraries for SOAP

- To use the functionality of Zeep, you need to install it first.
- Run the following command in your Terminal to install Zeep:

```
$ pip3 install Zeep
```

Using libraries for SOAP

- Now, we are going to see an example. Create a soap_example.pyscript and write the following code in it:

```
import zeep

w =
'http://www.soapclient.com/xml/soapresponder.wsdl'
c = zeep.Client(wsdl=w)
print(c.service.Method1('Hello', 'World'))
```

Using libraries for SOAP

- Run the script and you will get the following output:

```
student@ubuntu:~$ python3 soap_example.py
Output :
Your input parameters are Hello and World
```

What is a RESTful API?

- REST stands for Representational State Transfer, RESTful API has an approach to communication used in the development of web services.
- It is a style of a web service that works as a channel of communication between different systems over the internet.
- It is an application interface and is used to GET, PUT, POST, and DELETE data using HTTP requests.

Using standard libraries for RESTful APIs

- First, you must install the requests library as follows:

```
$ pip3 install requests
```

- Now, we will see an example.
- Create a rest_get_example.pyscript and write the following content in it:

```
import requests
req_obj =
requests.get('https://www.imdb.com/news/top?ref_=nv_tp_nw')
print(req_obj)
```

Using standard libraries for RESTful APIs

- Run the script and you will get the output as follows:

```
student@ubuntu:~/work$ python3 rest_get_example.py
Output:
<Response [200]>
```

Using standard libraries for RESTful APIs

```
import requests
import json

url_name = 'http://httpbin.org/post'
data = {"Name" : "John"}
data_json = json.dumps(data)
headers = {'Content-type': 'application/json'}
response = requests.post(url_name, data=data_json,
headers=headers)
print(response)
```

Using standard libraries for RESTful APIs

- Run the script and you will get the following output:

```
student@ubuntu:~/work$ python3 rest_post_example.py
Output:
<Response [200]>
```

Working with JSON data

- json.dump(obj, fileObj): This function will serialize an object as a JSON-formatted stream.
- json.dumps(obj): This function will serialize an object as a JSON formatted string.
- json.load(JSONfile): This function will deserialize a JSON file as a Python object.
- json.loads(JSONfile): This function will deserializes a string-type JSON file to a Python object.

Working with JSON data

It also has two classes for encoding and decoding listed here:

- **JSONEncoder:** Used to convert Python objects into JSON format.

- **JSONDecoder:** Used to convert a JSON formatted file into a Python object.

Working with JSON data

- For that, create a script json_to_python.py and write the following code in it:

```
import json

j_obj =  '{ "Name":"Harry", "Age":26,
"Department":"HR"}'
p_obj = json.loads(j_obj)
print(p_obj["Name"])
print(p_obj["Department"])
```

Working with JSON data

- Run the script and you will get the output as follows:

```
student@ubuntu:~/work$ python3 json_to_python.py
Output:
Harry
HR
```

Working with JSON data

- Now, we are going to see how to convert Python to JSON.
- For that, create a python_to_json.pyscript and write the following code in it:

```
import json

emp_dict1 =  '{ "Name":"Harry", "Age":26, "Department":"HR"}'
json_obj = json.dumps(emp_dict1)
print(json_obj)
```

Working with JSON data

- Run the script and you will get the following output:

```
student@ubuntu:~/work$ python3 python_to_json.py
Output:
"{ \"Name\":\"Harry\", \"Age\":26, \"Department\":\"HR\"}"
```

Working with JSON data

- Now, we are going to see how to convert Python objects of various types into the JSON string.
- For that, create a python_object_to_json.pyscript and write the following content in it:

Refer to the file 14_1.txt

- Run the script and you will get the following output:

Refer to the file 14_2.txt

Working with JSON data

Python	**JSON**
dict	Object
list	Array
tuple	Array
str	String
int	Number
float	Number
True	true
False	false
None	null

Summary

- In this lesson, you learned about SOAP APIs and RESTful APIs.
- You learned about the zeep Python library for SOAP APIs and the requests library for REST APIs.
- You also learned to work with JSON data, for instance, converting JSON to Python and vice versa.

15. Web Scraping - Extracting Useful Data from Websites

Extracting Useful Data from Websites

In this lesson, we will cover the following topics:

- What is web scraping?
- Data extraction
- Extracting information from Wikipedia

What is web scraping?

- Web scraping is the technique used to extract information from websites.
- This technique is used to transform unstructured data into structured data.
- The use of web scraping is to extract the data from the websites.

Data extraction

- First, we must install these two libraries.
- Run the following commands to install the requests and beautifulsoup libraries:

```
$ pip3 install requests
$ pip3 install beautifulsoup4
```

The requests library

- The use of the requests library is to use HTTP within our Python script in human-readable format.
- We can download the pages using the requests library in Python.
- The requests library has different types of requests., here, we are going to learn about the GET request.
- The GET request is used to retrieve information from a web server and the GET request downloads the HTML content of a specified web page.

The beautifulsoup library

- beautifulsoup is a library in Python, used for web scraping.

- It has simple methods for searching, navigating, and modifying.

- It is simply a toolkit used for extracting the data you needed from a web page.

The beautifulsoup library

- For that purpose, create a parse_web_page.py script and write the following content in it:

```
import requests
from bs4 import BeautifulSoup

page_result =
requests.get('https://www.imdb.com/news/top?ref_=nv_nw_tp')
parse_obj = BeautifulSoup(page_result.content, 'html.parser')

print(parse_obj)
```

The beautifulsoup library

- Run the script and you will get the output as follows:
- Refer to the file 15_1.txt

The beautifulsoup library

```
import requests
from bs4 import BeautifulSoup

page_result =
requests.get('https://www.imdb.com/news/top?ref_=nv_nw_tp')
parse_obj = BeautifulSoup(page_result.content, 'html.parser')

top_news = parse_obj.find(class_='news-article__content')
print(top_news)
```

The beautifulsoup library

- Run the script and you'll get the output as follows:
- Refer to the file 15_2.txt

The beautifulsoup library

```
import requests
from bs4 import BeautifulSoup

page_result =
requests.get('https://www.imdb.com/news/top?ref_=nv_nw_tp')
parse_obj = BeautifulSoup(page_result.content, 'html.parser')

top_news = parse_obj.find(class_='news-article__content')
top_news_a_content = top_news.find_all('a')
print(top_news_a_content)
```

The beautifulsoup library

- Run the script and you'll get the output as follows:
- Refer to the file 15_3.txt

Extracting information from Wikipedia

```
import requests
from bs4 import BeautifulSoup

page_result =
requests.get('https://en.wikipedia.org/wiki/Portal:History')
parse_obj = BeautifulSoup(page_result.content, 'html.parser')

h_obj = parse_obj.find(class_='hlist noprint')
h_obj_a_content = h_obj.find_all('a')

print(h_obj)
print(h_obj_a_content)
```

Extracting information from Wikipedia

- Run the script and you will get the following output:

Refer to the file 15_4.txt

Summary

- In this lesson, you learned about what web scraping is.
- We learned about two libraries that are used in extracting the data from a web page.
- We also extracted information from Wikipedia.

16. Statistics Gathering and Reporting

Statistics Gathering and Reporting

In this lesson, we will cover the following topics:

- NumPY module
- Pandas module
- Data visualization

NumPY module

- NumPY is a Python module that provides efficient operations on arrays.

- Install NumPY by running the following command in your Terminal:

```
$ pip3 install numpy
```

NumPY module

- We are going to use this numpy library to do operations on a numpy array.
- Now we are going to see how to create numpy arrays.
- For that, create a script called simple_array.py and write following code in it:

```
import numpy as np
my_list1 = [1,2,3,4]
my_array1 = np.array(my_list1)
print(my_list11, type(my_list1))
print(my_array1, type(my_array1))
```

NumPY module

- Run the script and you will get the following output:

```
student@ubuntu:~$ python3 simple_array.py
```

- The output is as follows:

```
[1, 2, 3, 4] <class 'list'>
[1 2 3 4] <class 'numpy.ndarray'>
```

NumPY module

```
import numpy as np

my_list1 = [1,2,3,4]
my_list2 = [11,22,33,44]

my_lists = [my_list1, my_list2]
my_array = np.array(my_lists)
print(my_lists, type(my_lists))
print(my_array, type(my_array))
```

NumPY module

- Run the script and you will get the following output:

```
student@ubuntu:~$ python3 mult_dim_array.py
```

- The output is as follows:

```
[[1, 2, 3, 4], [11, 22, 33, 44]] <class 'list'>
[[ 1 2 3 4]
 [11 22 33 44]] <class 'numpy.ndarray'>
```

NumPY module

```
import numpy as np

my_list1 = [1,2,3,4]
my_list2 = [11,22,33,44]

my_lists = [my_list1,my_list2]
my_array = np.array(my_lists)
print(my_array)

size = my_array.shape
print(size)

data_type = my_array.dtype
print(data_type)
```

NumPY module

- Run the script and you will get the following output:

```
student@ubuntu:~$ python3 size_and_dtype.py
```

- The output is as follows:

```
[[ 1  2  3  4]
 [11 22 33 44]]
(2, 4)
int64
```

NumPY module

- First, we will make an array with all zeros using the np.zeros() function, as shown here:

```
student@ubuntu:~$ python3
Python 3.6.7 (default, Oct 22 2018, 11:32:17)
[GCC 8.2.0] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>> import numpy as np
>>> np.zeros(5)
array([0., 0., 0., 0., 0.])
>>>
```

NumPY module

- After making the array with all zeros, we are going to make the array with all 1's using the np.ones() function of numpy, as shown here:

```
>>> np.ones((5,5))
array([[1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.]])
>>>
```

NumPY module

- Now, we are going to make an empty array using the np.empty() function of numpy, as shown here:

```
>>> np.empty([2,2])
array([[6.86506982e-317,  0.00000000e+000],
       [6.89930557e-310,  2.49398949e-306]])
>>>
```

NumPY module

- Now, let's see how to make an identity array using the np.eye() function, which results in the array with its diagonal value 1, as shown next:

```
>>> np.eye(5)
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
>>>
```

NumPY module

- Now, we are going to see the range function, which is used to create an array using the np.arange() function of numpy, as shown here:

```
>>> np.arange(10)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>>
```

Using arrays and scalars

- In this section, we are going to look at various arithmetic operations on arrays using numpy.
- For that, first we will create a multidimensional array, as follows:

Refer to the file 16_1.txt

Using arrays and scalars

- Now, let's look at some arithmetic operations on arrays.
- First, we will study the multiplication of arrays, as shown here:

```
>>> arr*arr
array([[16, 25, 36],
       [49, 64, 81]])
>>>
```

Using arrays and scalars

- Now, we are going to look at a subtraction operation on an array, as shown here:

```
>>> arr-arr
array([[0, 0, 0],
       [0, 0, 0]])
>>>
```

Using arrays and scalars

- Now we are going to look at arithmetic operations on arrays with scalars.
- Let's look at some operations:

```
>>> 1 / arr
array([[0.25          , 0.2       , 0.16666667],
       [0.14285714 , 0.125     , 0.11111111]])
>>>
```

Using arrays and scalars

- Now we will look at the exponential operation on the numpy array, as shown here:

```
>>> arr ** 3
array([[ 64, 125, 216],
       [343, 512, 729]])
>>>
```

Array indexing

- To perform indexing operations on the array, first we create a new numpy array and for that we are going to use the range() function to create the array, as shown here:

Refer to the file 16_2.txt

Array indexing

- Now, we are going to perform a different indexing operation on array arr.
- First, let's get the value in the array at a particular index:

```
>>> arr[7]
7
>>>
```

Array indexing

- After getting the value at a particular index, we are going to get values in a range.
- Let's look at the following example:

```
>>> arr[2:10]
array([2, 3, 4, 5, 6, 7, 8, 9])
>>> arr[2:10:2]
array([2, 4, 6, 8])
>>>
```

Array indexing

- We can also get values in the array from the index until the end, as show in the following example:

```
>>> arr[5:]
array([ 5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])
>>>
```

Array indexing

- Now we are going to look at slicing of the numpy array.
- In slicing, we actually take some part of our original array and store it in a specified array name.
- Let's look at an example:

```
>>> arr_slice = arr[0:8]
>>> arr_slice
array([0, 1, 2, 3, 4, 5, 6, 7])
>>>
```

Array indexing

- We can also give updated values to the slice of the array.
- Let's look at an an example:

```
>>> arr_slice[:] = 29
>>> arr_slice
array([29, 29, 29, 29, 29, 29, 29, 29])
>>>
```

Array indexing

- Let's see the result after giving values to the slice of the array and the effect on our original array:

```
>>> arr
array([29, 29, 29, 29, 29, 29, 29, 29,  8,  9, 10, 11, 12,
13, 14, 15])
>>>
```

Array indexing

- The changes applied onto the copy of the array do not affect the original array.
- So let's look at an example of copying an array:

```
>>> cpying_arr = arr.copy()
>>> cpying_arr
array([29, 29, 29, 29, 29, 29, 29, 29,  8,  9, 10, 11, 12,
13, 14, 15])
>>>
```

Indexing a 2D array

- A 2D array is an array of arrays, So, let's look at an example of a 2D array:

```
>>> td_array = np.array(([5,6,7],[8,9,10],[11,12,13]))
>>> td_array
array([[  5,   6,    7],
       [  8,   9,  10],
       [11, 12,  13]])
>>>
```

Indexing a 2D array

- Now we are also going to fetch the values in td_array through indexing.
- Let's look at an example to access values through indexing:

```
>>> td_array[1]
array([ 8,  9, 10])
>>>
```

Indexing a 2D array

- In such a type of indexing, when we access the value, we get the whole array.
- Instead of getting the whole array, we can also get access to particular value, Let's look at an example:

```
>>> td_array[1,0]
8
>>>
```

Indexing a 2D array

- We can also set up the two-dimensional array in a different way.
- First, set our 2D array with increased length.
- Let's set the length to 10. So, for that, we create a sample array with all zeros in it and, after that, we are going to put values in it, Let's look at an example:

Refer to the file 16_2.txt

Indexing a 2D array

- Now let's do some fancy indexing on it, as shown in the following example:

```
>>> td_array[[1,3,5,7]]
array([[1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [3., 3., 3., 3., 3., 3., 3., 3., 3., 3.],
       [5., 5., 5., 5., 5., 5., 5., 5., 5., 5.],
       [7., 7., 7., 7., 7., 7., 7., 7., 7., 7.]])
>>>
```

Universal array functions

- Create a script called sqrt_array.py and write the following content in it:

```
import numpy as np

array = np.arange(16)
print("The Array is : ",array)
Square_root = np.sqrt(array)
print("Square root of given array is : ", Square_root)
```

Universal array functions

- Run the script and you will get the following output:

```
student@ubuntu:~/work$ python3 sqrt_array.py
```

- The output is as follows:

```
The Array is : [ 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15]
Square root of given array is : [0. 1. 1.41421356
1.73205081 2. 2.23606798
 2.44948974 2.64575131 2.82842712 3. 3.16227766
3.31662479
 3.46410162 3.60555128 3.74165739 3.87298335]
```

Universal array functions

- Create a script called expo_array.py and write the following content in it:

```
import numpy as np

array = np.arange(16)
print("The Array is : ",array)
exp = np.exp(array)
print("exponential of given array is : ", exp)
```

Universal array functions

- Run the script and you will get the following output:

```
student@ubuntu:~/work$ python3 expo_array.py
```

- The output is as follows:

```
The Array is :  [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15]
exponential of given array is :  [1.00000000e+00 2.71828183e+00 7.38905610e+00 2.00855369e+01
 5.45981500e+01 1.48413159e+02 4.03428793e+02 1.09663316e+03
 2.98095799e+03 8.10308393e+03 2.20264658e+04 5.98741417e+04
 1.62754791e+05 4.42413392e+05 1.20260428e+06 3.26901737e+06]
```

Pandas module

- To use the functionality of this module, you must import it first.
- First, install the following packages that we need in our examples by running the following commands:

```
$ pip3 install pandas
$ pip3 install matplotlib
```

Series

- First, we will look at an example of series without declaring an index.
- For that, create a script called series_without_index.py and write the following content in it:

```
import pandas as pd
import numpy as np

s_data = pd.Series([10, 20, 30, 40], name = 'numbers')
```

Series

- Run the script and you will get the following output:

```
student@ubuntu:~/work$ python3
series_without_index.py
```

- The output is as follows :

```
0 10
1 20
2 30
3 40
Name: numbers, dtype: int64
```

Series

```
import pandas as pd
import numpy as np

s_data = pd.Series([10, 20, 30, 40], index = ['a', 'b', 'c',
'd'], name = 'numbers')
print(s_data)
print()
print("The data at index 2 is: ", s_data[2])
print("The data from range 1 to 3 are:\n", s_data[1:3])
```

Series

```
student@ubuntu:~/work$ python3 series_with_index.py
a    10
b    20
c    30
d    40
Name: numbers, dtype: int64

The data at index 2 is:  30
The data from range 1 to 3 are:
 b    20
c    30
Name: numbers, dtype: int64
```

DataFrames

- If you don't have a csv file in your system, create a file named employee.csv, as follows:

```
Id, Name, Department, Country
101, John, Finance, US
102, Mary, HR, Australia
103, Geeta, IT, India
104, Rahul, Marketing, India
105, Tom, Sales, Russia
```

DataFrames

- Now, we are going to read this csv file into a DataFrame.
- For that, create a script called read_csv_dataframe.py and write the following content in it:

```
import pandas as pd
file_name = 'employee.csv'
df = pd.read_csv(file_name)
print(df)
print()
print(df.head(3))
print()
print(df.tail(1))
```

DataFrames

- Run the script and you will get the following output:

Refer to the file 16_3.txt

Data visualization

Data visualization is the term that describes the efforts in understanding the significance of data, placing it in a visual manner. In this section, we are going to look at the following data visualization techniques:

- Matplotlib
- Plotly

Data visualization

Matplotlib

- To use matplotlib in your Python program, first we have to install matplotlib.
- Run the following command in your Terminal to install matplotlib:

```
$ pip3 install matplotlib
```

Data visualization

- Now, you have to install one more package, tkinter, for graphical representations.

- Install it using the following command:

```
$ sudo apt install python3-tk
```

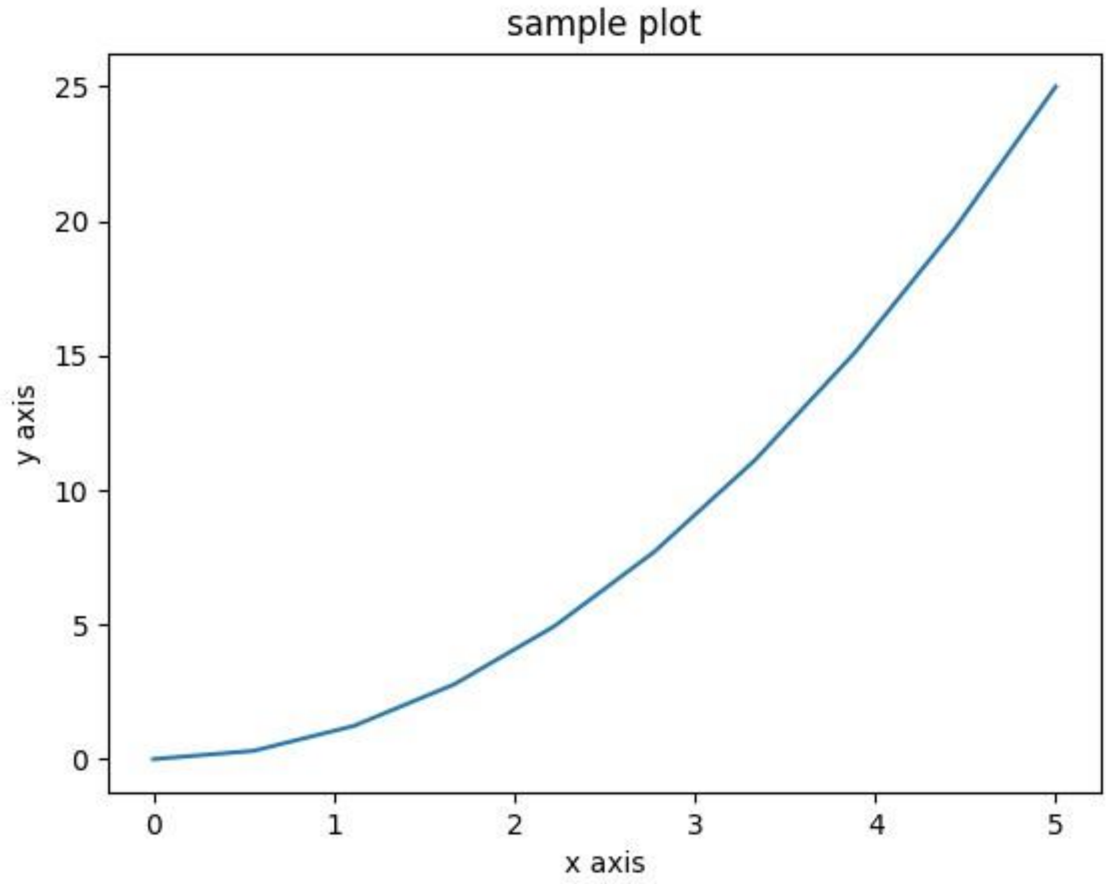
Data visualization

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 5, 10)
y = x**2
plt.plot(x,y)
plt.title("sample plot")
plt.xlabel("x axis")
plt.ylabel("y axis")
plt.show()
```

Data visualization

- Run the script and you will get the following output:

```
student@ubuntu:~/work$ python3 simple_plot.py
```


sample plot
y axis
x axis
0
5
10
15
20
25
0
1
2
3
4
5

Data visualization

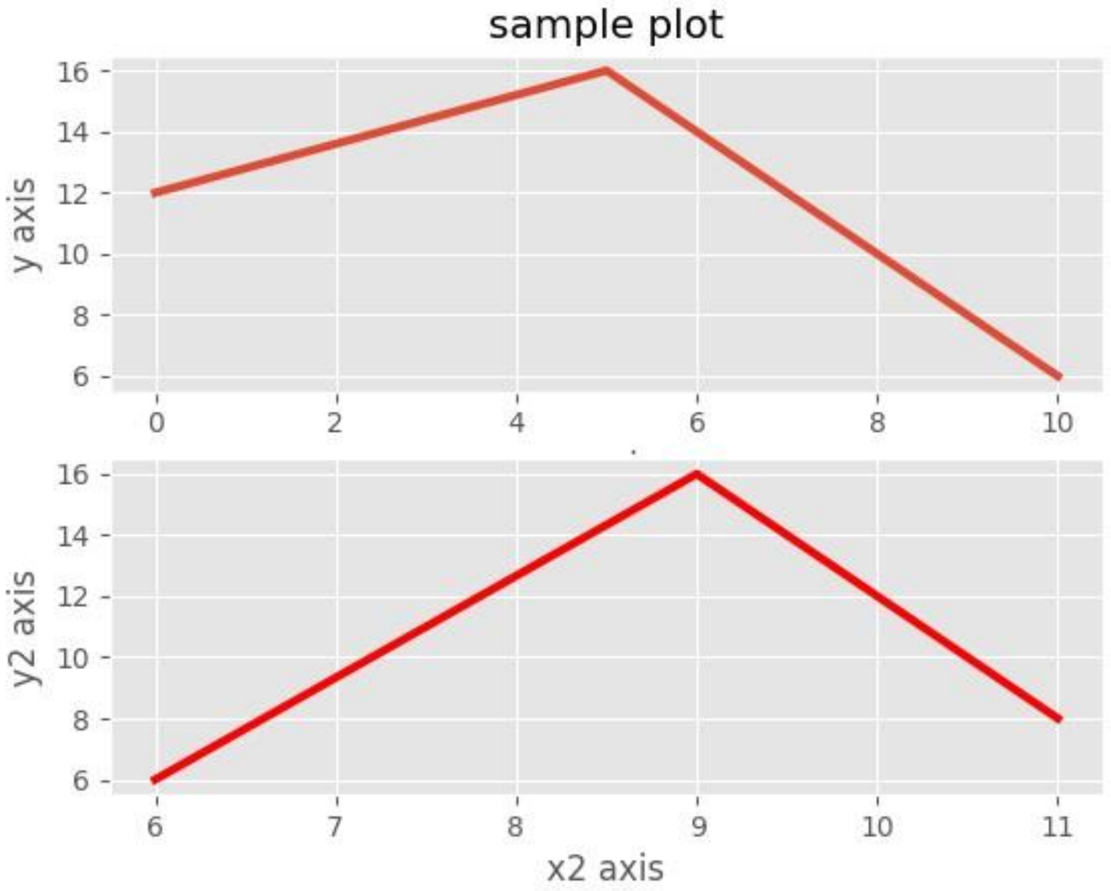
- Now, create a script called simple_plot2.py and write the following content in it:

Refer to the file 16_4.txt

Data visualization

- Run the script and you will get the following output:

```
student@ubuntu:~/work$ python3 simple_plot2.py
```


sample plot
y axis
y2 axis
x2 axis
16
14
12
10
8
6
0
2
4
6
8
10
7
9
11

Data visualization

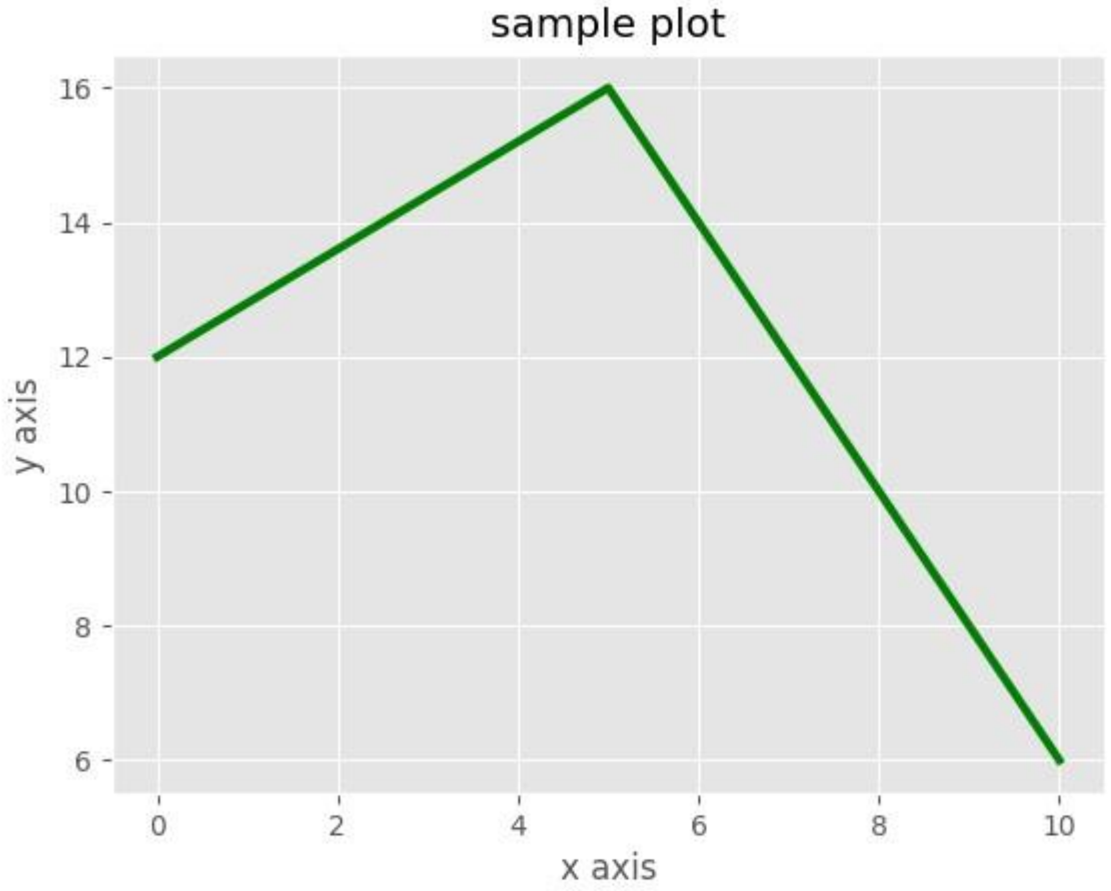
- We'll save the preceding figure in a file named my_sample_plot.jpg.
- Now, we will look at an example.
- For that, create a script called simple_plot3.py and write the following content in it:

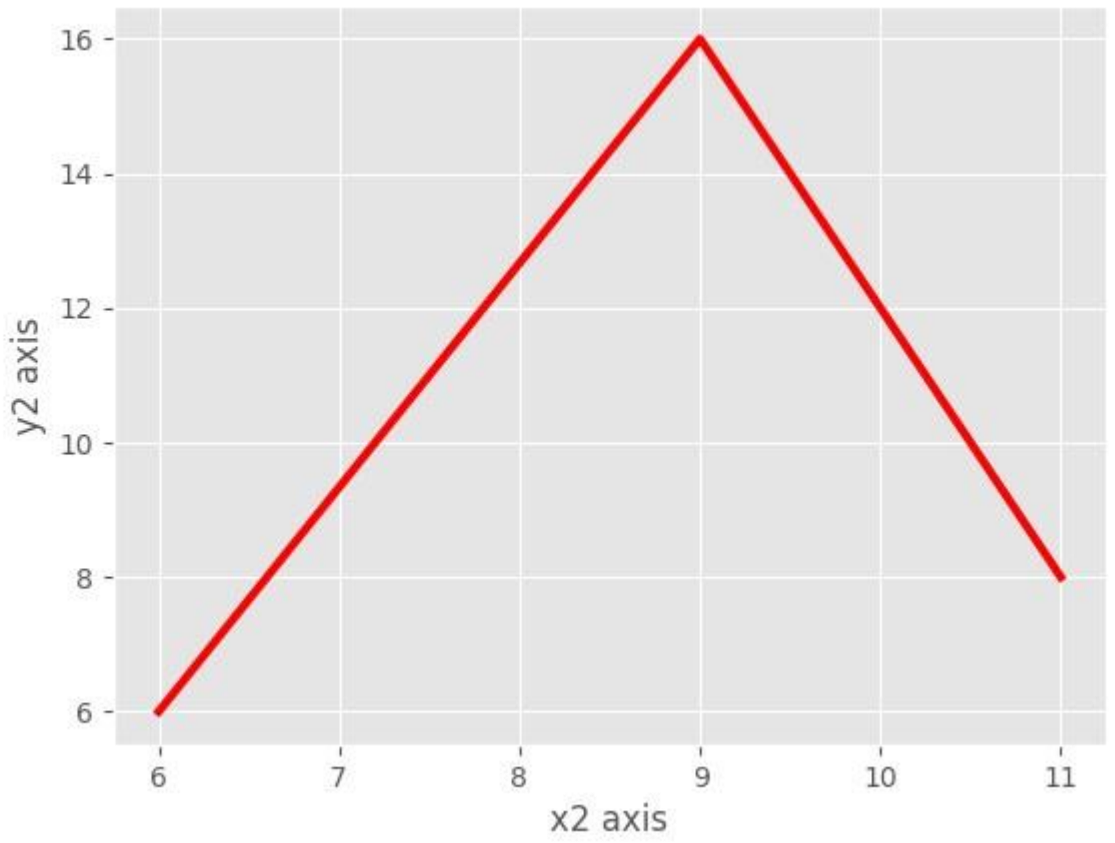
Refer to the file 16_5.txt

Data visualization

- Run the script and you will get the following output:

```
student@ubuntu:~/work$ python3 simple_plot3.py
```


sample plot
y axis
x axis
16
14
12
10
8
6
0
2
4
6
8
10


16
14
12
10
8
6
y2 axis
6
7
8
9
10
11
x2 axis

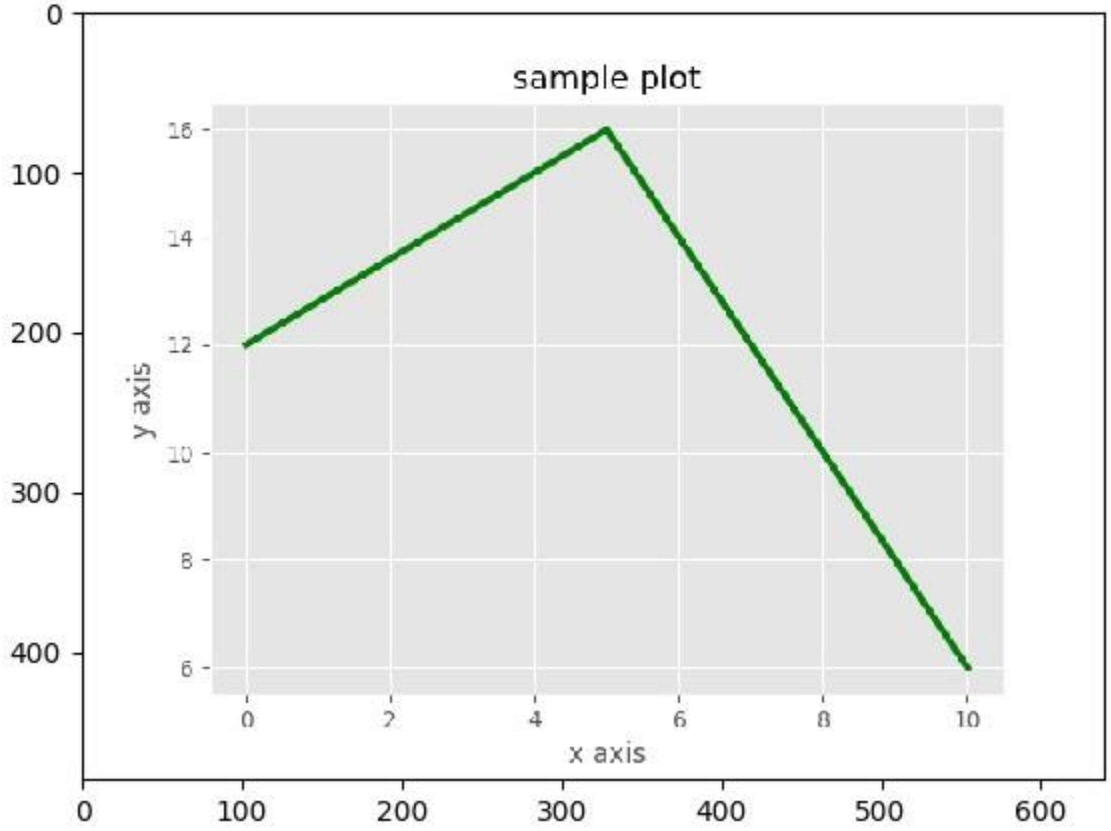
Data visualization

- Now, we will look at an example to open saved figures.
- For that, create a script called open_image.py and write the following content in it:

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
plt.imshow(mpimg.imread('my_sample_plot1.jpg'))
plt.show()
```

- Run the script and you will get the following output:

```
student@ubuntu:~/work$ python3 open_image.py
```


sample plot
y axis
x axis
16
14
12
10
8
6
0
2
4
6
8
10
0
100
200
300
400
500
600

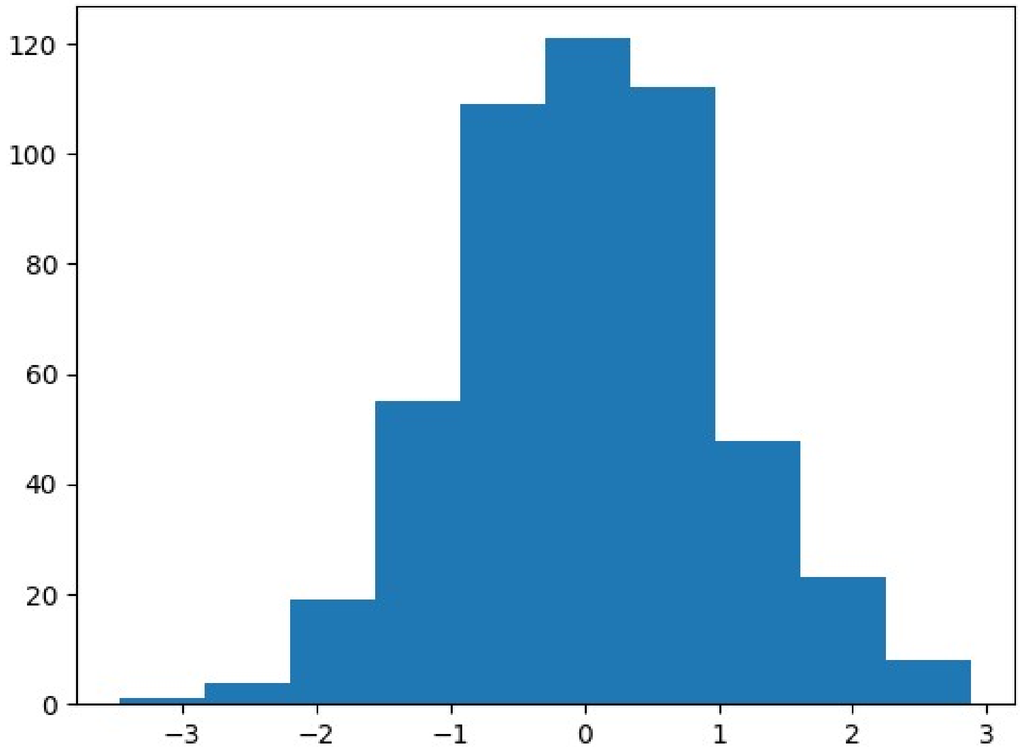
Histograms

```
import matplotlib.pyplot as plt
import numpy as np

x = np.random.randn(500)
plt.hist(x)
plt.show()
```

- Run the script and you will get the following output:

```
student@ubuntu:~/work$ python3
histogram_example.py
```

Histograms



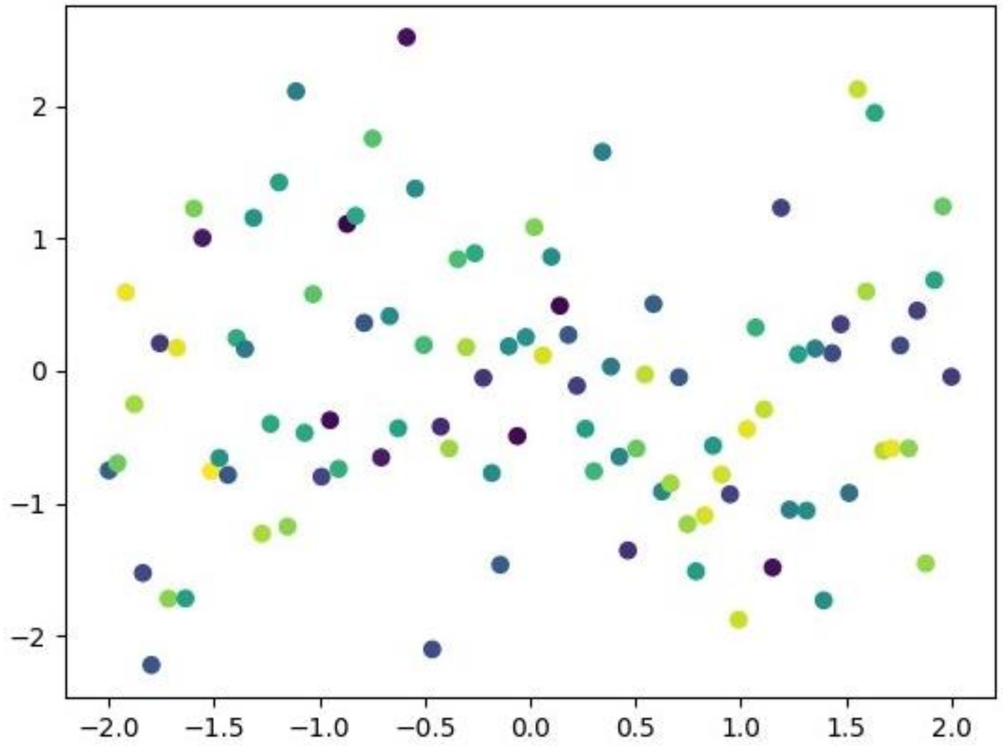
Scatter plots

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-2,2,100)
y = np.random.randn(100)
colors = np.random.rand(100)
plt.scatter(x,y,c=colors)
plt.show()
```

- Run the script and you will get the following output:

```
student@ubuntu:~/work$ python3 scatterplot_example.py
```

Histograms



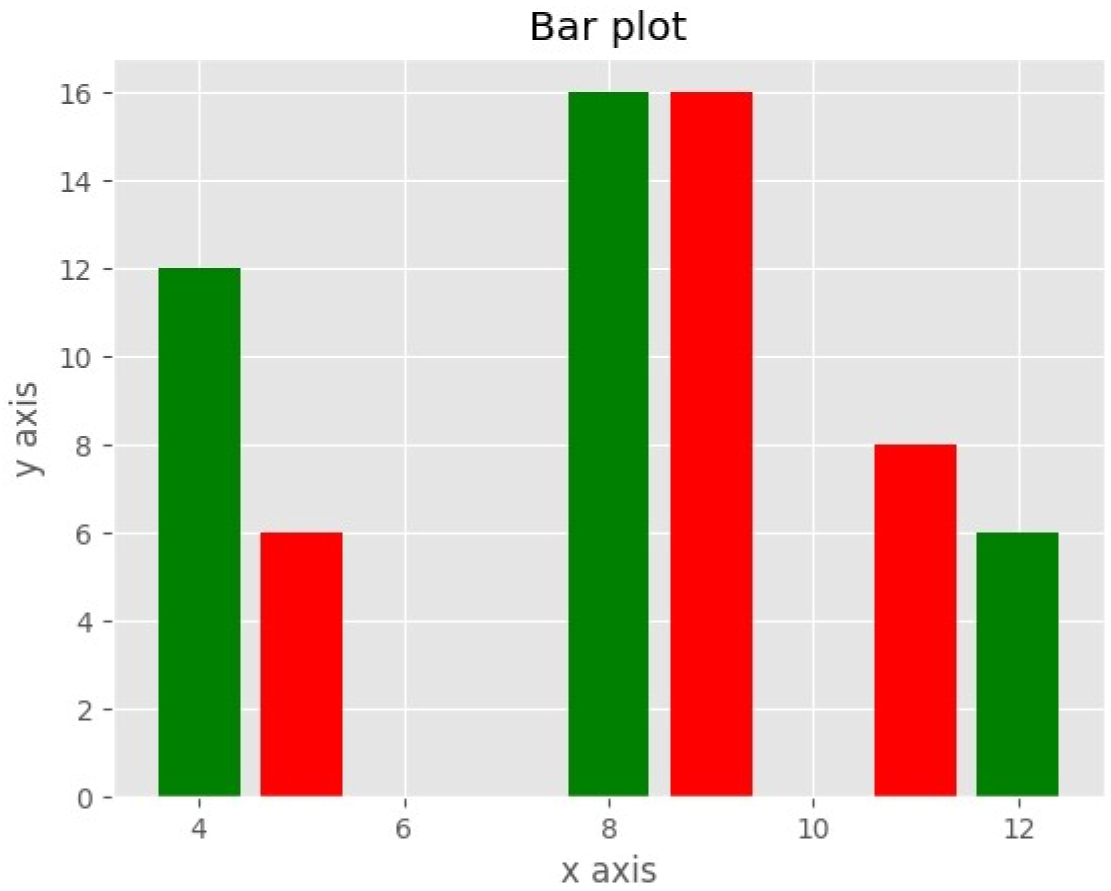
Bar charts

- A bar chart is a chart that represents your data in rectangular bars.
- You can plot them vertically or horizontally.
- Create a script called bar_chart.py and write the following content in it:

Refer to the file 16_6.txt

- Run the script and you will get the following output:

```
student@ubuntu:~/work$ python3 bar_chart.py
```


Bar plot
y axis
x axis
0
2
4
6
8
10
12
14
16
4
6
8
10
12

Plotly

- To install plotly, run the following command in your Terminal:

```
$ pip3 install plotly
```

- We can use plotly online as well as offline.
- For online usage, you need to have a plotly account and after that you need to set up your credentials in Python:

```
plotly.tools.set_credentials_file(username='Username',
api_key='APIkey')
```

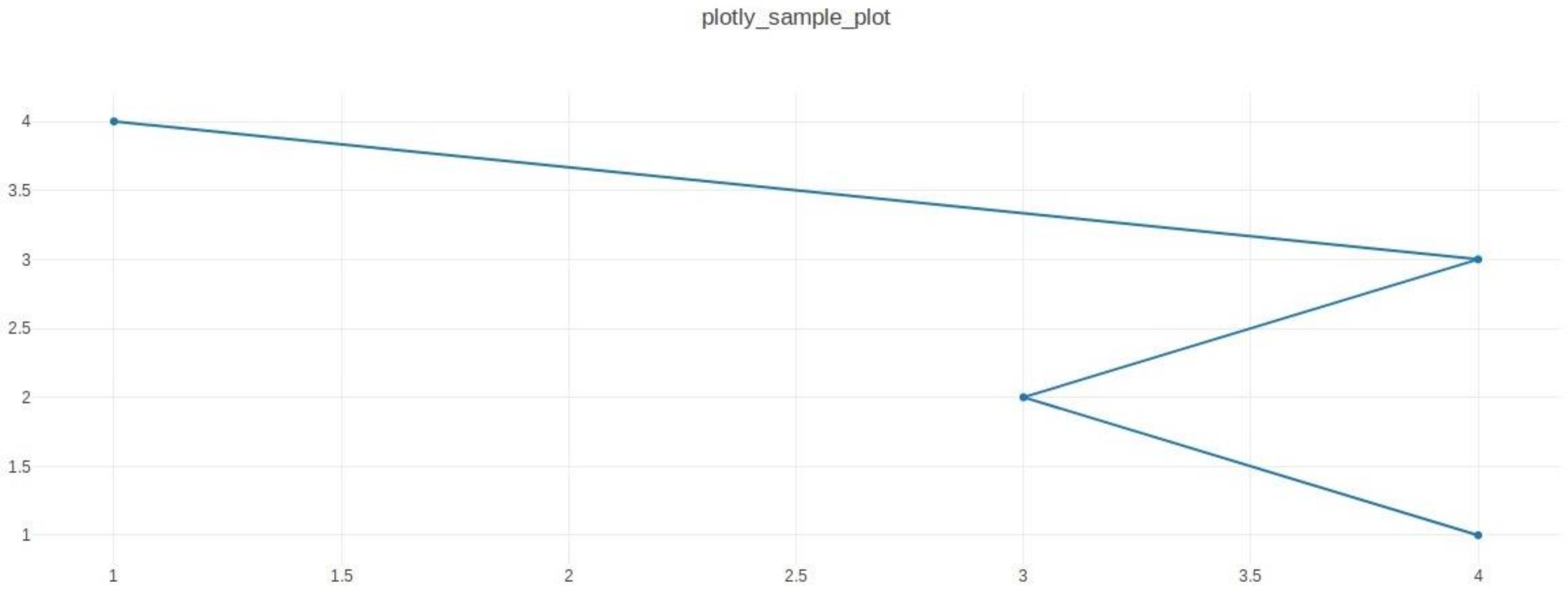
Plotly

```
import plotly
from plotly.graph_objs import Scatter, Layout

plotly.offline.plot({
    "data": [Scatter(x=[1, 4, 3, 4], y=[4, 3, 2, 1])],
    "layout": Layout(title="plotly_sample_plot")
})
```

- Run the preceding script as sample_plotly.py. You will get the following output:

```
student@ubuntu:~/work$ python3 sample_plotly.py
```

Plotly



Scatter plots

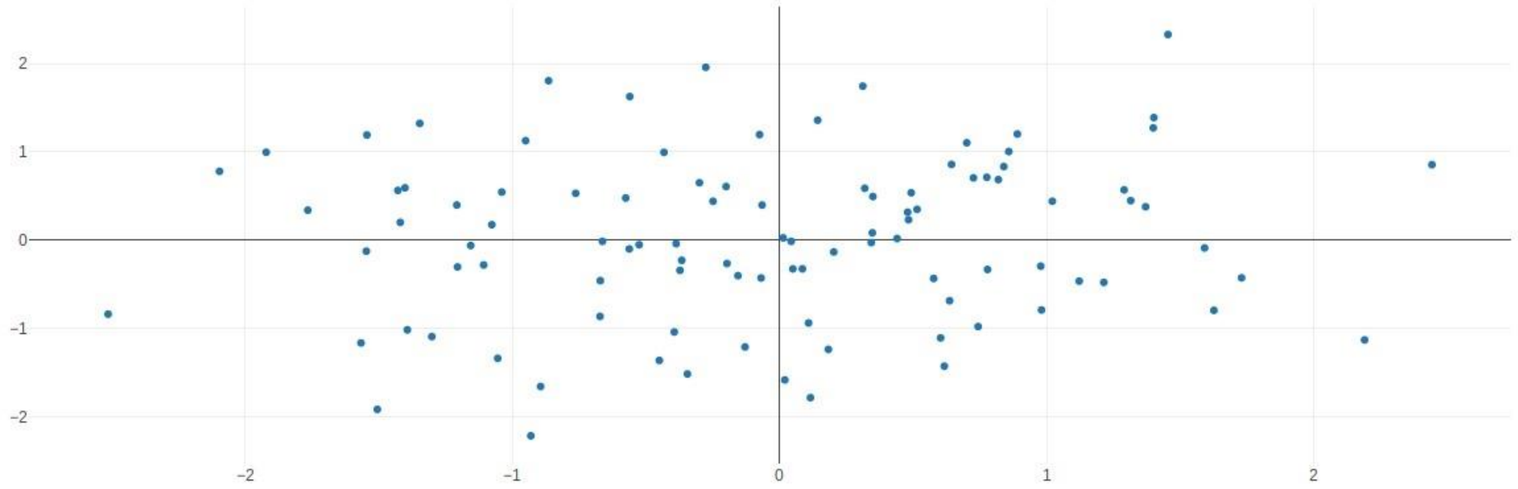
```
import plotly
import plotly.graph_objs as go
import numpy as np

x_axis = np.random.randn(100)
y_axis = np.random.randn(100)

trace = go.Scatter(x=x_axis, y=y_axis, mode = 'markers')
data_set = [trace]
plotly.offline.plot(data_set, filename='scatter_plot.html')
```

Scatter plots

- Run the script and you will get the following output:

```
student@ubuntu:~/work$ python3 scatter_plot_plotly.py
```

- The output is as follows:



Line scatter plots

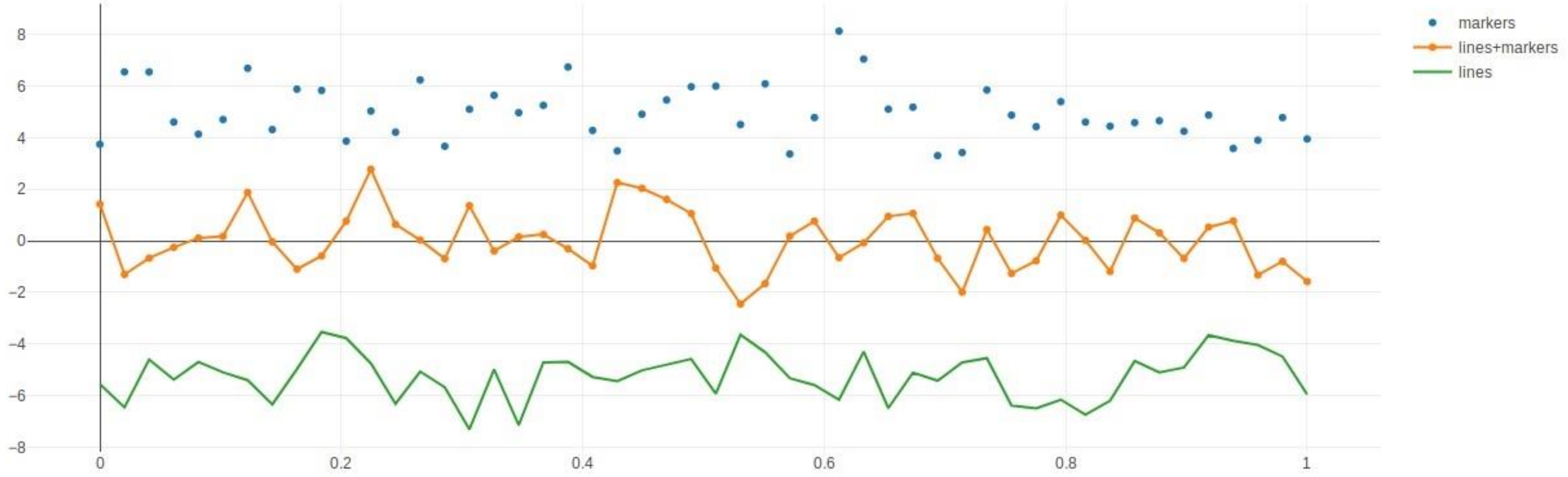
- We can also create some more informative plots, such as a line scatter plot.
- Let's look at an example. Create a script called line_scatter_plot.py and write the following content in it:

Refer to the file 16_7.txt

- Run the script and you will get the following output:

```
student@ubuntu:~/work$ python3 line_scatter_plot.py
```

Line scatter plots



Box plots

- The box plot is usually informative and also helpful, especially when you have too much to show with very less data.
- Let's look at an example, Create a script called plotly_box_plot.py and write the following content in it:

Refer to the file 16_8.txt

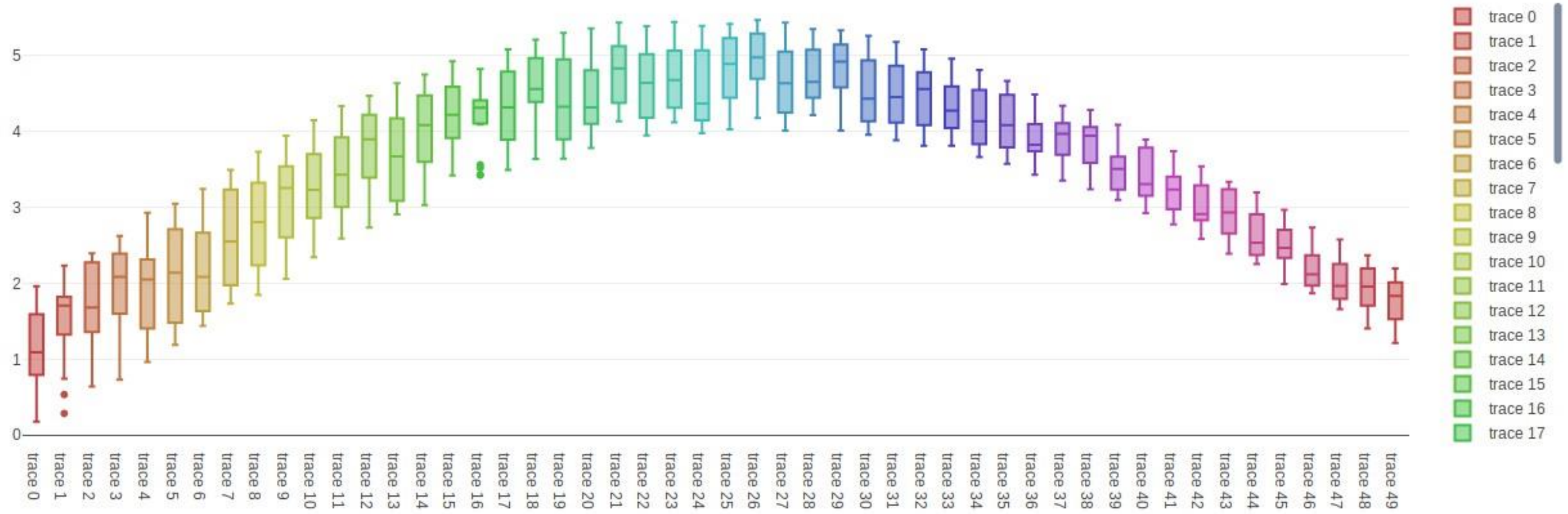
- Run the script and you will get the following output:

```
student@ubuntu:~/work$ python3 plotly_box_plot.py
```

- Run the script and you will get the following output:

student@ubuntu:~/work$ python3 plotly_box_plot.py

- The output is as follows:



Contour plots

- Let's look at an example of a contour plot.
- Create a script called contour_plotly.py and write the following content in it:

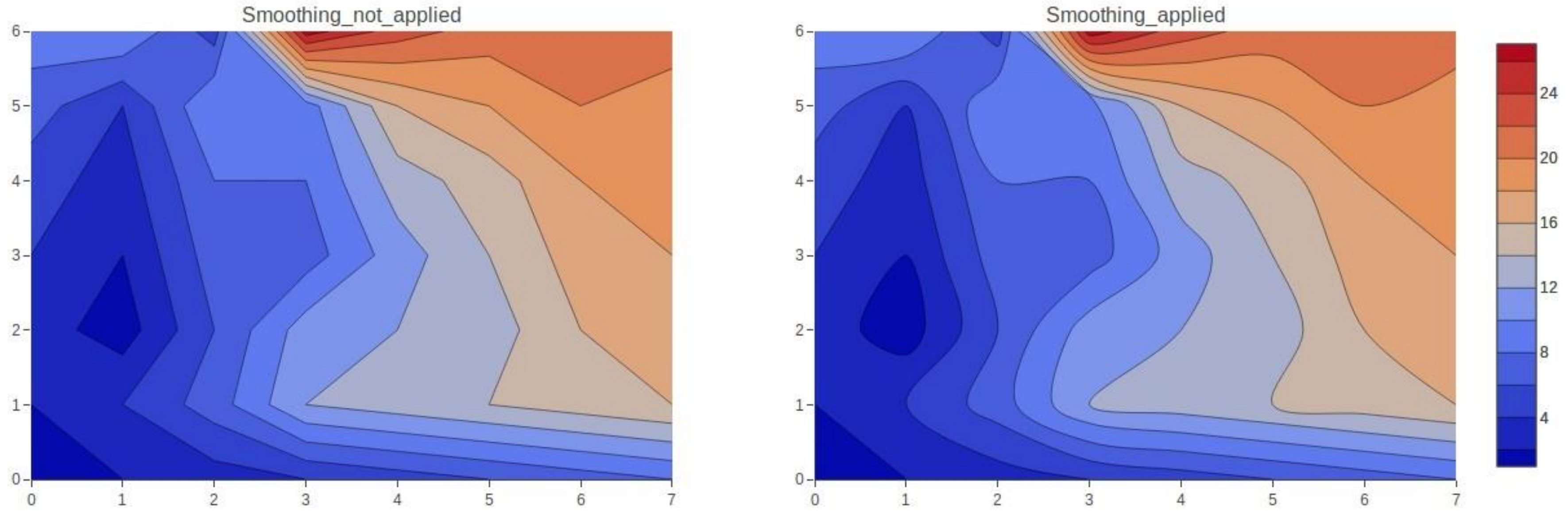
Refer to the file 16_9.txt

- Run the script and you will get the following output:

```
student@ubuntu:~/work$ python3 contour_plotly.py
This is the format of your plot grid:
[ (1,1) x1,y1 ]  [ (1,2) x2,y2 ]
```

Contour plots

- The output is as follows:



Summary

- In this lesson, we learned about the NumPY and Pandas modules, as well as data visualization techniques.
- In the NumPY module section, we learned about indexing and slicing the array and the universal array function.
- In the pandas module section, we learned about Series and DataFrames.

17. MySQL and SQLite Database Administrations

MySQL and SQLite Database Administrations

In this lesson, you will learn the following:

- MySQL database administration
- SQLite database administration

MySQL database administration

- Let's learn how to install MySQL and a Python mysqldb package.
- For this, run the following command in your Terminal:

```
$ sudo apt install mysql-server
```

MySQL database administration

- This command installs the MySQL server and various other packages.
- While installing the package, we are prompted to enter a password for the MySQL root account:
- The following code is used for checking for the mysqldb package to install:

```
$ apt-cache search MySQLdb
```

MySQL database administration

- And the following is for installing the Python interface for MySQL:

```
$ sudo apt-get install python3-mysqldb
```

- Now, we will check if mysql is installed properly or not.
- For this, run the following command in Terminal:

```
student@ubuntu:~$ sudo mysql -u root -p
```

MySQL database administration

- Once the command runs, you will get the following output:

Refer to the file 16_10.txt

MySQL database administration

By running sudo mysql -u root -p, you will get the mysql console. There are some commands used for listing databases and tables, and using the database to store our work. We will see them one by one:

- This is for listing all the databases:

```
show databases;
```

- And this is for using the database:

```
use database_name;
```

MySQL database administration

- The following code is used for listing all the tables:

```
show tables;
```

MySQL database administration

- In this section, we are going to create a database named test and we will use this database throughout this section:

Refer to the file 16_11.txt

MySQL database administration

- Now, we are going to create a user and grant the privileges to that user. Run the following commands:

```
mysql> create user 'test_user'@'localhost' identified by 'test123';
Query OK, 0 rows affected (0.06 sec)

mysql> grant all on test.* to 'test_user'@'localhost';
Query OK, 0 rows affected (0.02 sec)

mysql>
```

Getting a database version

```
import MySQLdb as mdb
import sys

con_obj = mdb.connect('localhost', 'test_user', 'test123', 'test')
cur_obj = con_obj.cursor()
cur_obj.execute("SELECT VERSION()")
version = cur_obj.fetchone()
print ("Database version: %s " % version)

con_obj.close()
```

Getting a database version

- Run the script and you will get the following output:

```
student@ubuntu:~/work/mysql_testing$ python3
get_database_version.py
Output:
Database version: 5.7.24-0ubuntu0.18.04.1
```

Creating a table and inserting data

- Now, we are going to create a table and we will insert some data into it.
- For that, create a create_insert_data.pyscript and write the following content in it:

Refer to the file 16_12.txt

Creating a table and inserting data

- Run the script and you will get the following output:

```
student@ubuntu:~/work/mysql_testing$ python3
create_insert_data.py

Output:
Table Created !!
Data inserted Successfully !!
```

Creating a table and inserting data

- To check whether your table is created successfully or not, open your mysql console and run the following commands:

Refer to the file 16_13.txt

Retrieving the data

```
import MySQLdb as mdb

con_obj = mdb.connect('localhost', 'test_user', 'test123', 'test')
with con_obj:
        cur_obj = con_obj.cursor()
        cur_obj.execute("SELECT * FROM courses")
        records = cur_obj.fetchall()
        for r in records:
                print(r)
```

Retrieving the data

- Run the script and you will get the output as follows:

```
student@ubuntu:~/work/mysql_testing$ python3
retrieve_data.py
Output:
(1, 'Harry Potter')
(2, 'Lord of the rings')
(3, 'Murder on the Orient Express')
(4, 'The adventures of Sherlock Holmes')
(5, 'Death on the Nile')
```

Updating the data

```
import MySQLdb as mdb

con_obj = mdb.connect('localhost', 'test_user', 'test123', 'test')
cur_obj = con_obj.cursor()
cur_obj.execute("UPDATE courses SET Name = 'Fantastic Beasts'
WHERE Id = 1")
try:
    con_obj.commit()
except:
    con_obj.rollback()
```

Updating the data

- Run the script as follows:

```
student@ubuntu:~/work/mysql_testing$ python3 update_data.py
```

- Now, to check if your record is updated or not, run retrieve_data.py as follows:

```
student@ubuntu:~/work/mysql_testing$ python3 retrieve_data.py

Output:
(1, 'Fantastic Beasts')
(2, 'Lord of the rings')
(3, 'Murder on the Orient Express')
(4, 'The adventures of Sherlock Holmes')
(5, 'Death on the Nile')
```

Deleting the data

```
import MySQLdb as mdb

con_obj = mdb.connect('localhost', 'test_user', 'test123', 'test')
cur_obj = con_obj.cursor()
cur_obj.execute("DELETE FROM courses WHERE Id = 5");
try:
		con_obj.commit()
except:
		con_obj.rollback()
```

Deleting the data

- Run the script as follows:

```
student@ubuntu:~/work/mysql_testing$ python3 delete_data.py
```

- Now, to check whether your record is deleted or not, run the retrieve_data.py script as follows:

```
student@ubuntu:~/work/mysql_testing$ python3 retrieve_data.py
```

Output:

```
(1, 'Fantastic Beasts')
(2, 'Lord of the rings')
(3, 'Murder on the Orient Express')
(4, 'The adventures of Sherlock Holmes')
```

SQLite database administration

- Now, we will install SQLite first.
- Run the following command in your Terminal:

```
$ sudo apt install sqlite3
```

SQLite database administration

- Now, first, we will see how to create a database in SQLite.
- To create a database, you simply have to write the command in your Terminal as follows:

```
$ sqlite3 test.db
```

- After running this command, you will get the sqlite console opened in your Terminal as follows:

```
student@ubuntu:~$ sqlite3 test.db
SQLite version 3.22.0 2018-01-22 18:45:57
Enter ".help" for usage hints.
sqlite>
```

Connecting to the database

- Create a connect_database.pyscript and write the following content in it:

```
import sqlite3

con_obj = sqlite3.connect('test.db')
print ("Database connected successfully !!")
```

Connecting to the database

- Run the script and you will get the following output:

```
student@ubuntu:~/work $ python3
connect_database.py

Output:
Database connected successfully !!
```

Creating a table

- Now, we are going to create a table in our database. For that, we will create a create_table.pyscript and write the following content in it:

```
import sqlite3

con_obj = sqlite3.connect("test.db")
with con_obj:
        cur_obj = con_obj.cursor()
        cur_obj.execute("""CREATE TABLE courses(title text,
author text)""")

print ("Table created")
```

Creating a table

- Run the script and you will get the output as follows:

```
student@ubuntu:~/work $ python3 create_table.py
```

```
Output:
Table created
```

Inserting the data

- Now, we will insert the data into our table.

- For that, we will create a insert_data.pyscript and write the following content in it:

Refer to the file 16_14.txt

Connecting to the database

- Run the script and you will get the following output:

```
student@ubuntu:~/work$ python3 insert_data.py

Output:
Data inserted Successfully !!
```

Retrieving the data

```
import sqlite3

con_obj = sqlite3.connect('test.db')
cur_obj = con_obj.execute("SELECT title, author from
courses")
for row in cur_obj:
		print ("Title = ", row[0])
		print ("Author = ", row[1], "\n")

con_obj.close()
```

Retrieving the data

- Run the script and you will get the output as follows:

Refer to the file 16_15.txt

Retrieving the data

- You can also retrieve the data in the sqlite3 console.
- For that, start the SQLite console first and then retrieve the data as follows:

Refer to the file 16_16.txt

Updating the data

- We can update the data from our table using the update statement.
- Now, we are going to see an example of updating data.
- For that, create a update_data.pyscript and write the following content in it:

Refer to the file 16_17.txt

Updating the data

- Run the script and you will get the following output:

```
student@ubuntu:~/work $ python3 update_data.py
```

```
Output:
Data updated Successfully !!
```

Updating the data

- Now, to check that the data is actually updated or not, you can run retrieve_data.py, or else you can go to the SQLite console and run select * from courses;.

- You will get the updated output as follows:

Refer to the file 16_18.txt

Deleting the data

- Now, we will see an example of deleting data from a table.

- We are going to do this using the delete statement.

- Create a delete_data.pyscript and write the following content in it:

Refer to the file 16_19.txt

Updating the data

- Run the script and you will get the following output:

```
student@ubuntu:~/work $ python3 delete_data.py
```

Output:

```
Data deleted successfully !!
```

Updating the data

- In the preceding example, we deleted a record from a table, We used the delete SQL statement.
- Now, to check whether the data is deleted successfully or not, run retrieve_data.py or start the SQLite console, as follows:

Refer to the file 16_20.txt

Updating the data

- You can see the record whose author was john smith is deleted:

Checking on SQLite console:

Output:

```
student@ubuntu:~/work$ sqlite3 test.db
SQLite version 3.22.0 2018-01-22 18:45:57
Enter ".help" for usage hints.
sqlite>
sqlite> select * from courses;
Pride and Prejudice|Jane Austen
The Lord of the Rings|J. R. R. Tolkien
Murder on the Orient Express|Agatha Christie
A Study in Scarlet|Arthur Conan Doyle
sqlite>
```

Summary

- In this lesson, we learned about MySQL as well as SQLite database administration.

- We created databases and tables, We then inserted a few records in tables.

- Using the select statement, we retrieved the records.

THANK YOU ☺

LEARNING VOYAGE