

Week - 8:

1. Import numpy, Plotpy and Scipy and explore their functionalities.

NumPy – Introduction

NumPy is a Python package. It stands for 'Numerical Python'. It is a library consisting of multidimensional array objects and a collection of routines for processing of array.

Numeric, the ancestor of NumPy, was developed by Jim Hugunin. Another package Numarray was also developed, having some additional functionalities. In 2005, Travis Oliphant created NumPy package by incorporating the features of Numarray into Numeric package. There are many contributors to this open source project.

Operations using NumPy

Using NumPy, a developer can perform the following operations –

1. Mathematical and logical operations on arrays.
2. Fourier transforms and routines for shape manipulation.
3. Operations related to linear algebra. NumPy has in-built functions for linear algebra and random number generation.
4. install NumPy, run the following command.
5. Python setup.py install
6. To test whether NumPy module is properly installed, try to import it from Python prompt.
7. import numpy
8. If it is not installed, the following error message will be displayed.
9. Traceback (most recent call last):
10. File "<pyshell#0>", line 1, in <module>
11. import numpy
12. ImportError: No module named 'numpy'

Alternatively, NumPy package is imported using the following syntax –

Example 1

```
import numpy as np
import numpy as np
a = np.array([1,2,3])
print(a)
The output is as follows
-[1, 2, 3]
```

Example 2

```
# more than one dimensions
import numpy as np
a = np.array([[1, 2], [3, 4]])
print(a)
The output is as follows –
[[1, 2]
 [3, 4]]
```

Example 3

```
# minimum dimensions
import numpy as np
a = np.array([1, 2, 3,4,5], ndmin = 2)
print a
The output is as follows –
[[1, 2, 3, 4, 5]]
Example 4
# dtype parameter
import numpy as np
a = np.array([1, 2, 3], dtype = complex)
print a
The output is as follows –
[ 1.+0.j,  2.+0.j,  3.+0.j]
```

SciPy - Introduction

SciPy, a scientific library for Python is an open source, BSD-licensed library for mathematics, science and engineering. The SciPy library depends on NumPy, which provides convenient and fast N-dimensional array manipulation. The main reason for building the SciPy library is that, it should work with NumPy arrays. It provides many user-friendly and efficient numerical practices such as routines for numerical integration and optimization. This is an introductory tutorial, which covers the fundamentals of SciPy and describes how to deal with its various modules.

Example 1

```
pip install plotly "notebook>=5.3" "ipywidgets>=7.2"
```

Inside Jupyter notebook:

```
import plotly.graph_objs as go
fig = go.FigureWidget()
# Display an empty figure
fig
# Add a scatter chart
fig.add_scatter(y=[2, 1, 4, 3])
# Add a bar chart
fig.add_bar(y=[1, 4, 3, 2])
# Add a title
fig.layout.title = 'Hello FigureWidget'
```

See the Python documentation for more examples.

Plotpy - Introduction

plotly.py is an interactive, open-source, and browser-based graphing library for Python :sparkles:

Built on top of plotly.js, plotly.py is a high-level, declarative charting library. plotly.js ships with over 30 chart types, including scientific charts, 3D graphs, statistical charts, SVG maps, financial charts, and more.

plotly.py is MIT Licensed. Plotly graphs can be viewed in Jupyter notebooks, standalone HTML files, or hosted online on plot.ly.

Example 1(Scatter plot)

```
N = 100
x = np.random.rand(N)
y = np.random.rand(N)
colors = np.random.rand(N)
sz = np.random.rand(N)*30
fig = go.Figure()
fig.add_scatter(x=x,
                y=y,
                mode='markers',
                marker={'size': sz,
                      'color': colors,
                      'opacity': 0.6,
                      'colorscale': 'Viridis'
                      });
iplot(fig)
```

Example 2(Line Plot)

```
import plotly.plotly as py
import plotly.graph_objs as go
# Create random data with numpy
import numpy as np
N = 500
random_x = np.linspace(0, 1, N)
random_y = np.random.randn(N)
# Create a trace
trace = go.Scatter(
    x = random_x,
    y = random_y
)
data = [trace]
py.iplot(data, filename='basic-line')
```

Example 3(Bar plot)

```
import plotly.plotly as py
import plotly.graph_objs as go
data = [go.Bar(
    x=['giraffes', 'orangutans', 'monkeys'],
    y=[20, 14, 23]
)]
py.iplot(data, filename='basic-bar')
```

Example 4(Pie chart)

```
import plotly.plotly as py
import plotly.graph_objs as go
labels = ['Oxygen','Hydrogen','Carbon_Dioxide','Nitrogen']
values = [4500,2500,1053,500]
trace = go.Pie(labels=labels, values=values)
py.iplot([trace], filename='basic_pie_chart')
```

2. Install NumPy package with pip and explore it.

Introduction

To run the NumPy program, first, Numpy needs to be installed. Numpy is installed from Python's official website using **pip and conda** by running different commands on different operating systems. Many important packages are automatically installed within the Numpy library.

Steps for Installing NumPy

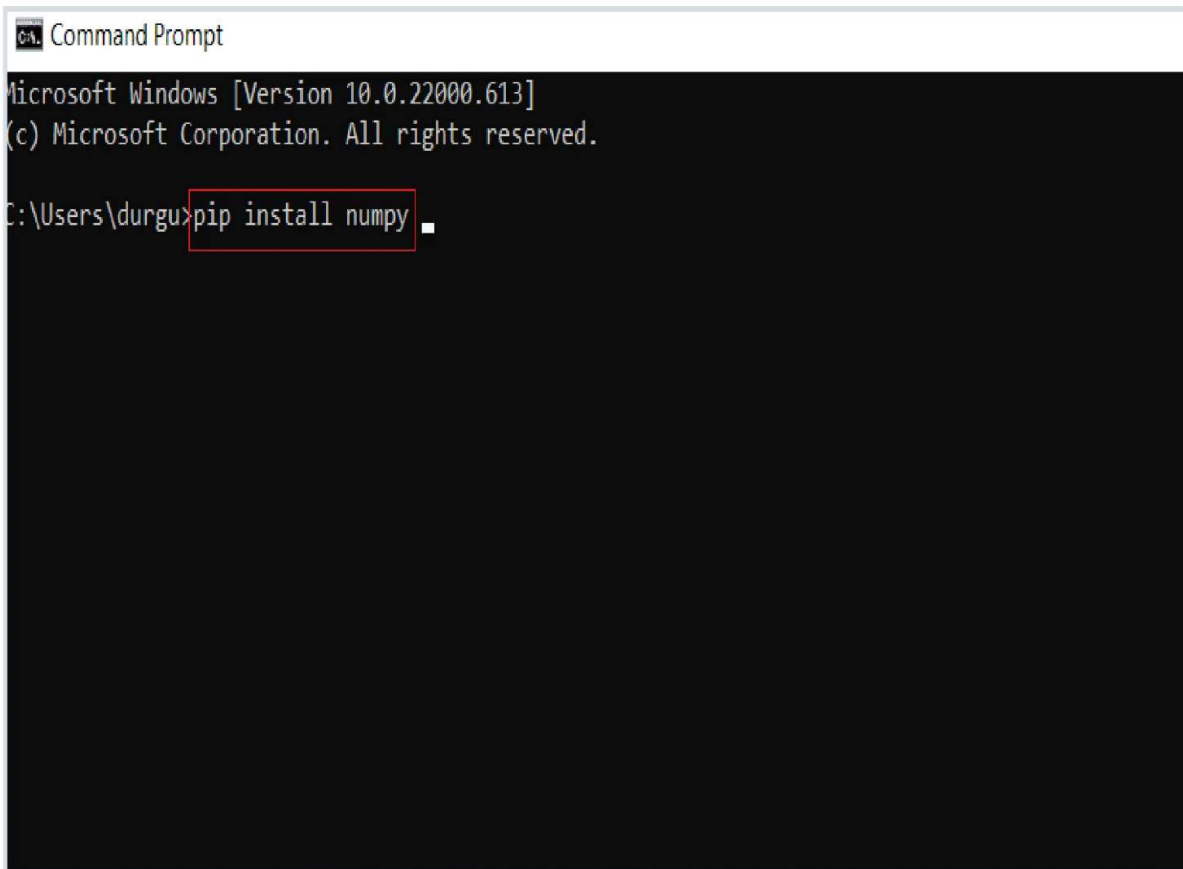
1. Download and Install Python.
 - Python should be installed before installing NumPy.
 - Download Python from [Python's official website](#) according to the operating system. After installing Python, Numpy is installed using different commands in different operating systems.

Installing NumPy on Windows Operating System

Steps for installing NumPy on Windows:

1. **Install NumPy** using the following PIP command in the command prompt terminal:

pip install numpy



```
Command Prompt
Microsoft Windows [Version 10.0.22000.613]
(c) Microsoft Corporation. All rights reserved.

C:\Users\durgu>pip install numpy
```

The installation will start automatically, and Numpy will be successfully installed with its latest version

```
Command Prompt
Microsoft Windows [Version 10.0.22000.613]
(c) Microsoft Corporation. All rights reserved.

C:\Users\durgu>pip install numpy
Collecting numpy
  Downloading numpy-1.22.3-cp310-cp310-win_amd64.whl (14.7 MB)
    ----- 14.7/14.7 MB 4.5 MB/s eta 0:00:00
Installing collected packages: numpy
Successfully installed numpy-1.22.3

C:\Users\durgu>
```

Verify NumPy Installation by typing the command given below in cmd:

pip show Numpy

It will show the location and numpy version installed.

```
Command Prompt
Microsoft Windows [Version 10.0.22000.613]
(c) Microsoft Corporation. All rights reserved.

C:\Users\durgu>pip show numpy
Name: numpy
Version: 1.22.3
Summary: NumPy is the fundamental package for array computing with Python.
Home-page: https://www.numpy.org
Author: Travis E. Oliphant et al.
Author-email:
License: BSD
Location: c:\users\durgu\appdata\local\programs\python\python310\lib\site-packages
Requires:
Required-by:

C:\Users\durgu>
```

Import NumPy Package

1. Create python Environment in cmd by typing: **Python**

```
Command Prompt - python
Microsoft Windows [Version 10.0.22000.613]
(c) Microsoft Corporation. All rights reserved.

C:\Users\durgu>python
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

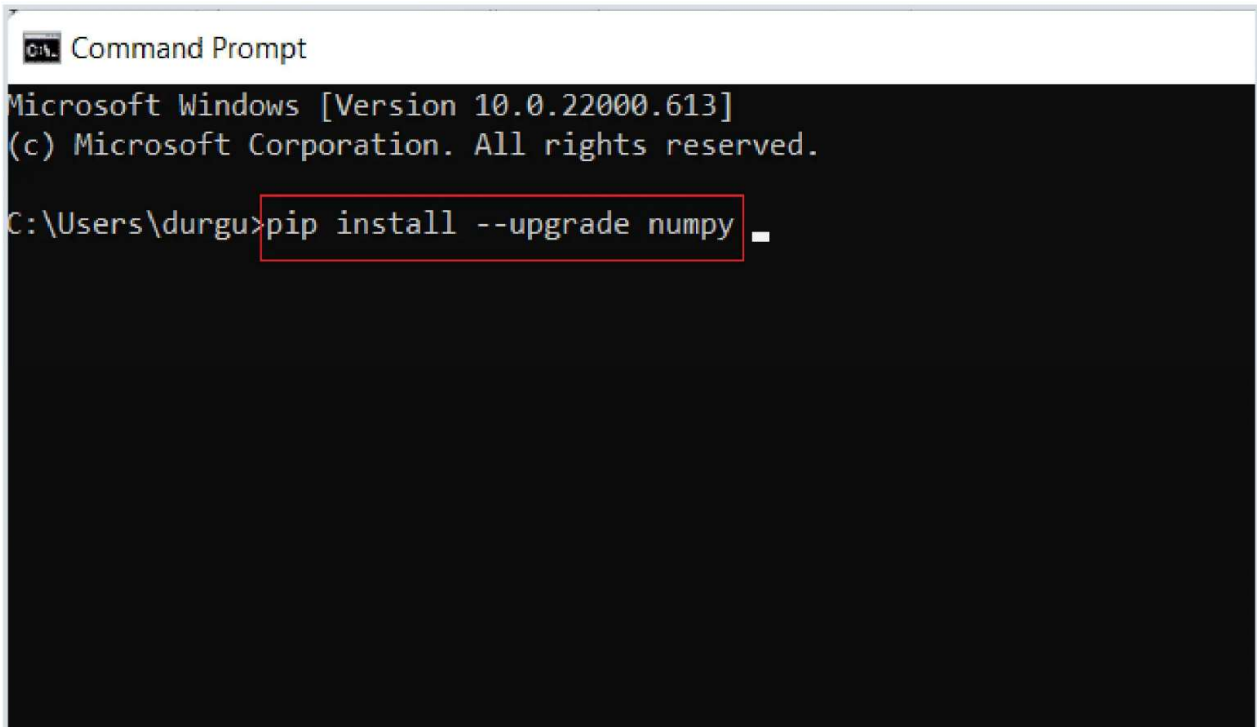
2. Type command **import numpy as np**

```
Command Prompt - python
Microsoft Windows [Version 10.0.22000.613]
(c) Microsoft Corporation. All rights reserved.

C:\Users\durgu>python
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy as np
>>> █
```

Upgrade NumPy by using the following command:

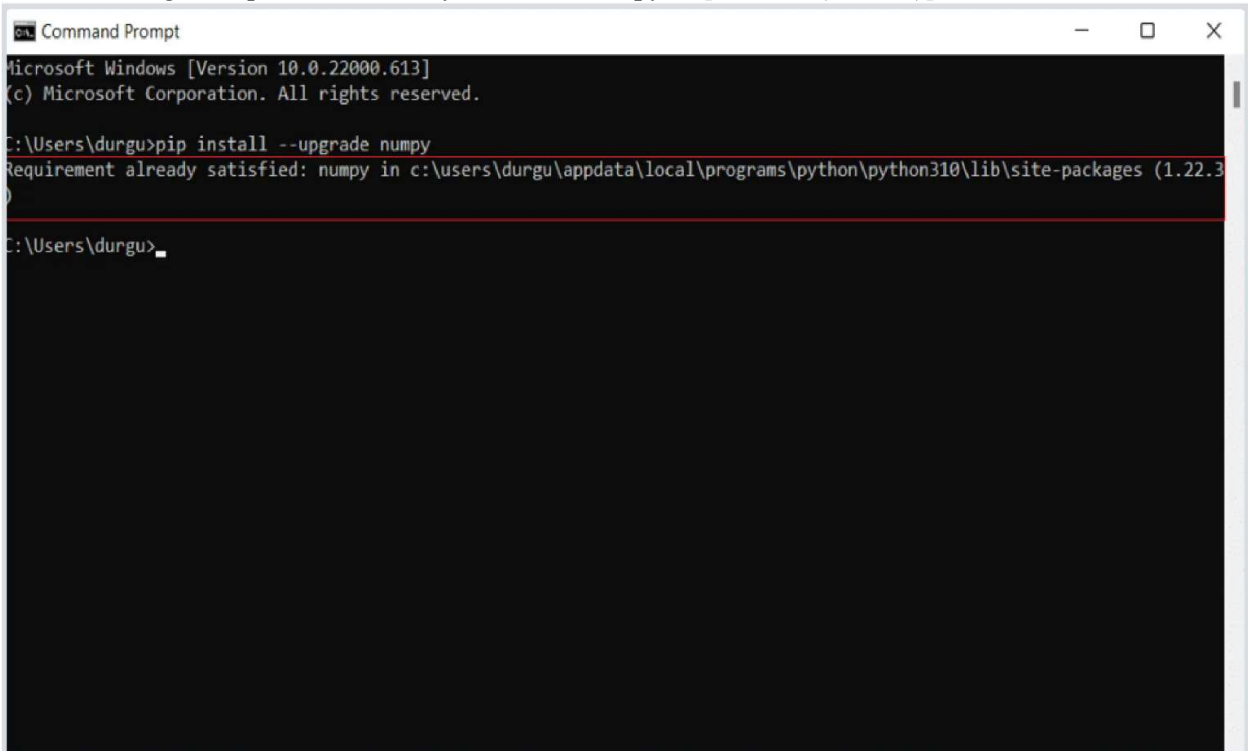
pip install --upgrade numpy



```
Command Prompt
Microsoft Windows [Version 10.0.22000.613]
(c) Microsoft Corporation. All rights reserved.

C:\Users\durgu>pip install --upgrade numpy
```

Shows message: Requirement already satisfied: numpy in [location (version)]



```
Command Prompt
Microsoft Windows [Version 10.0.22000.613]
(c) Microsoft Corporation. All rights reserved.

C:\Users\durgu>pip install --upgrade numpy
Requirement already satisfied: numpy in c:\users\durgu\appdata\local\programs\python\python310\lib\site-packages (1.22.3)

C:\Users\durgu>
```

By performing all the above steps, Numpy will be successfully installed.

3. Write a program to implement Digital Logic Gates – AND, OR, NOT, EX-OR

Logic gates are elementary building blocks for any digital circuits. It takes one or two inputs and produces output based on those inputs. Outputs may be high (1) or low (0). Logic gates are used for circuits that perform calculations, data storage, or show off object-oriented programming especially the power of inheritance.

Types of Logic Gates in Python

There are seven basic logic gates in [Python](#). These are the following:

- AND Gate
- OR Gate
- NOT Gate
- NAND Gate
- NOR Gate
- XOR Gate

1. AND Gate in Python

The AND gate gives an output of 1 if both the two inputs are 1, it gives 0 otherwise.

```
# working of AND gate
def AND (a, b):
    if a == 1 and b == 1:
        return True
    else:
        return False

# Driver code
if __name__ == '__main__':
    print(AND(1, 1))

    print("+-----+-----+")
    print(" | AND Truth Table | Result |")
    print(" A = False, B = False | A AND B =",AND(False,False)," | ")
    print(" A = False, B = True | A AND B =",AND(False,True)," | ")
    print(" A = True, B = False | A AND B =",AND(True,False)," | ")
    print(" A = True, B = True | A AND B =",AND(True,True)," | ")
```

2. OR Gate in Python

The OR gate gives an output of 1 if either of the two inputs are 1, it gives 0 otherwise.

```
# working of OR gate
def OR(a, b):
    if a == 1 or b == 1:
        return True
    else:
        return False

# Driver code
if __name__ == '__main__':
    print(OR(0, 0))
    print("+-----+-----+")
    print(" | OR Truth Table | Result |")
    print(" A = False, B = False | A OR B =", OR(False, False), " | ")
    print(" A = False, B = True | A OR B =", OR(False, True), " | ")
    print(" A = True, B = False | A OR B =", OR(True, False), " | ")
    print(" A = True, B = True | A OR B =", OR(True, True), " | ")
```

3. NOT Gate in Python

It acts as an inverter. It takes only one input. If the input is given as 1, it will invert the result as 0 and vice-versa.

```
# working of Not gate

def NOT(a):
    return not a

# Driver code
if __name__ == '__main__':
    print(NOT(0))

    print("+-----+-----+")
    print(" | NOT Truth Table | Result |")
    print(" A = False | A NOT =", NOT(False), " | ")
    print(" A = True, | A NOT =", NOT(True), " | ")
```

4. XOR Gate in Python

The XOR gate gives an output of 1 if either of the inputs is different, it gives 0 if they are the same.

```
# working of Xor gate
```

```
def XOR (a, b):
```

```
    if a != b:
```

```
        return 1
```

```
    else:
```

```
        return 0
```

```
# Driver code
```

```
if __name__ == '__main__':
```

```
    print(XOR(5, 5))
```

```
    print("+-----+-----+")
```

```
    print(" | XOR Truth Table | Result |")
```

```
    print(" A = False, B = False | A XOR B =",XOR(False,False)," | ")
```

```
    print(" A = False, B = True | A XOR B =",XOR(False,True)," | ")
```

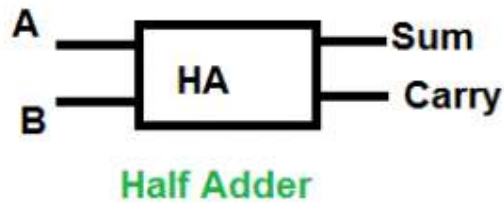
```
    print(" A = True, B = False | A XOR B =",XOR(True,False)," | ")
```

```
    print(" A = True, B = True | A XOR B =",XOR(True,True)," | ")
```

4. Write a program to implement Half Adder, Full Adder, and Parallel Adder

Half Adder

Given two inputs of Half Adder A, B. The task is to implement the Half Adder circuit and Print output i.e sum and carry of two inputs



Logical Expression:

$\text{Sum} = A \text{ XOR } B$

$\text{Carry} = A \text{ AND } B$

Truth Table:

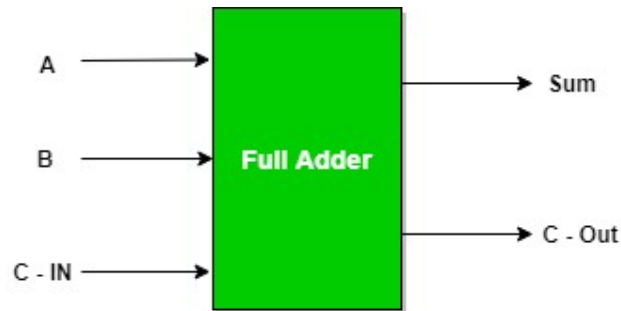
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

program to implement Half Adder

```
import numpy as np
def half_adder(A, B):
    Sum = np.bitwise_xor(A, B)
    Carry = np.bitwise_and(A, B)
    return Sum, Carry
# Driver code
A = 0
B = 1
Sum, Carry = half_adder(A, B)
print("Sum:", Sum)
print("Carry:", Carry)
```

Full Adder :

A Full Adder is a logical circuit that performs an addition operation on three one-bit binary numbers. The full adder produces a sum of the three inputs and carry value.



Logical Expression :

$$\text{SUM} = \text{C-IN} \text{ XOR } (\text{A XOR B})$$

$$\text{C-OUT} = \text{A B} + \text{B C-IN} + \text{A C-IN}$$

Truth Table :

Inputs			Outputs	
A	B	C - IN	Sum	C - Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

python program to implement full adder

```
# python program to implement full adder
# Function to print sum and C-Out
def getResult(A, B, C):

    # Calculating value of sum
    Sum = C ^ (A ^ B)
    C
    # Calculating value of C-Out
    C_Out = Bin&(not(A ^ B))| not(A)&B

    # printing the values
    print("Sum = ", Sum)
    print("C-Out = ", C_Out)

# Driver code
A = 0
B = 0
C = 1
# passing three inputs of fulladder as arguments to get result function
getResult(A, B, C)
```

Parallel Adder

A **Parallel Adder** is a digital circuit capable of finding the arithmetic **sum** of two binary numbers that is **greater than one bit** in length by operating on corresponding pairs of bits in parallel.

```
def binary_adder(a,b):
    an = len(a)
    bn = len(b)

    # Convert strings to list of bits -- very functional syntax here
    al = list(int(x,2) for x in list(a))
    bl = list(int(x,2) for x in list(b))

    # Pad smaller list with 0's
    dif = an - bn
    # more digits in a than b
    if dif > 0:
        for i in range(dif):
            bl.insert(0,0)
    else:
        for i in range(abs(dif)):
            al.insert(0,0)

    print(al)
    print(bl)

    result = []
    carry = 0
    # Iterate through list right to left, calling full_adder each time and
    # inserting the sum each time
    for i in range(len(al)-1,-1,-1):
        carry,sum = full_adder(carry,al[i],bl[i])
        result.insert(0,sum)
        print result
    result.insert(0,carry)

    return "".join(str(x) for x in result)

# Driver code
A = 0
B = 0
# passing two inputs of parallel adder as arguments to get result function
binary_adder(A,B)
```

5. Write a GUI program to create a window wizard having two text labels, two text fields and two buttons as Submit and Reset.

Our first GUI will be a window with a label and two buttons:

```
from tkinter import Tk, Label, Button
class MyFirstGUI:
    def __init__(self, master):
        self.master = master
        master.title("A simple GUI")
        self.label = Label(master, text="This is our first GUI!")
        self.label.pack()
        self.greet_button = Button(master, text="Greet", command=self.greet)
        self.greet_button.pack()
        self.close_button = Button(master, text="Close", command=master.quit)
        self.close_button.pack()
    def greet(self):
        print("Greetings!")
root = Tk()
my_gui = MyFirstGUI(root)
root.mainloop()
```

Example for the Button Class

The following script defines two buttons: one to quit the application and another one for the action, i.e. printing the text "Tkinter is easy to use!" on the terminal.

```
import tkinter as tk
def write_slogan():
    print("Tkinter is easy to use!")
root = tk.Tk()
frame = tk.Frame(root)
frame.pack()
button = tk.Button(frame,
                    text="QUIT",
                    fg="red",
                    command=quit)
button.pack(side=tk.LEFT)
slogan = tk.Button(frame,
                    text="Hello",
                    command=write_slogan)
slogan.pack(side=tk.LEFT)
root.mainloop()
```

The result of the previous example looks like this:

