"Python For Scientific and Numeric Computing"

Special Assignment Report

Submitted in Partial Fulfillment of the
Requirements for completion of

Course on

3EC3112 Embedded Systems Programming

By

Manav Chotalia
(21MECE05)

Department of Electronics and Communication Engineering,
Institute of Technology,
Nirma University,
Ahmedabad 382 481

October 2021

CERTIFICATE

This is to certify that the Comprehensive Evaluation Report entitled “Python for Scientific and Numeric Computing” submitted by Mr. Manav Chotalia as per the academic records (21MECE05) towards the partial fulfillment of for completion of Course on 3EC3112 Embedded Systems Programming is the record of work carried out by him/her individually.

Date:
Faculty Coordinator:

Undertaking for Originality of the Work

I, Manav Chotalia, 21MECE05, give undertaking that the Comprehensive Evaluation Report entitled "Python for Scientific and Numeric Computing" submitted by me, towards the partial fulfillment of for completion of Course on 3EC3112 Embedded Systems Programming, is the original work carried out by me and I give assurance that no attempt of plagiarism has been made. I understand that in the event of any similarity found subsequently with any other published work or any report elsewhere; it will result in severe disciplinary action.

____

Signature of the Student

Date: _______________

Place: _______________

Abstract

Python is an extremely usable, high-level programming language that is now a standard in Computer Science. This Project focus on numerical and scientific computing with python. It is open source, completely standardized across different platforms, immensely flexible, and easy to use and learn. Programs written in Python are highly readable and often much shorter than comparable programs written in other languages like C or Fortran. Moreover, Python comes pre-loaded with standard modules that provide a huge array of functions and algorithms,[2] for tasks like parsing text data, manipulating and finding files on disk, reading/writing compressed files, and downloading data from web servers. Python is also capable of all of the complex techniques that advanced programmers expect, like object orientation. NumPy, SciPy, Matplotlib libraries are explained in details with their syntax, use cases and detailed review. Finally using all explained libraries little project is created.

INDEX

LIST OF FIGURES

Chapter 1

Introduction

1.1 History of Python

Python was conceived in the late 1980s by **Guido van Rossum** at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to ABC programming language, which was inspired by SETL, capable of exception handling and interfacing with the Amoeba operating system. Its implementation began in December 1989. [7]Van Rossum shouldered sole responsibility for the project, as the lead developer, until 12 July 2018, when he announced his "permanent vacation" from his responsibilities as Python's "Benevolent Dictator for Life", a title the Python community bestowed upon him to reflect his long-term commitment as the project's chief decision-maker. In January 2019, active Python core developers elected a five-member "Steering Council" to lead the project.[7]

Python 2.0 was released on 16 October 2000, with many major new features, including a cycle-detecting garbage collector and support for Unicode.[7]

Python 3.0 was released on 3 December 2008. It was a major revision of the language that is not completely backward-compatible. Many of its major features were backported to Python 2.6.x and 2.7.x version series. Releases of Python 3 include the 2to3 utility, which automates the translation of Python 2 code to Python 3.[7]

Python 2.7's end-of-life date was initially set at 2015 then postponed to 2020 out of concern that a large body of existing code could not easily be forward-ported to Python 3. No more security patches or other improvements will be released for it. With Python 2's end-of-life, only Python 3.6.x and later are supported.[7]

1.2 Characteristics of Python

Python is a modern, general-purpose, object-oriented, high-level programming language.

General characteristics of Python:

- clean and simple language: Easy-to-read and intuitive code, easy-to-learn minimalistic syntax,
- maintainability scales well with size of projects.
- expressive language: Fewer lines of code, fewer bugs, easier to maintain.

Technical details:

- dynamically typed: No need to declare the type of variables, function arguments or return types.
- automatic memory management: No need to explicitly allocate and deallocate memory for variables
- and data arrays. No memory leak bugs.
- interpreted: No need to compile the code. The Python interpreter reads and executes the python
- code directly.

Advantages:

- The main advantage is ease of programming, minimizing the time required to develop, debug and maintain the code.
- Well-designed language that encourages many good programming practices.
- Modular and object-oriented programming, good system for packaging and re-use of code.
- This often results in more transparent, maintainable and bug-free code.
- Documentation tightly integrated with the code.
- A large standard library, and a large collection of add-on packages.

Disadvantages:

- Since Python is an interpreted and dynamically typed programming language, the execution of python code can be slow compared to compiled statically typed programming languages, such as C and Fortran.
- Somewhat decentralized, with different environment, packages and documentation spread out at different places. Can make it harder to get started.

1.3 Python for Scientific and Numeric Computing

Python has a strong position in scientific computing because of large community of users, easy to use, help and documentation. It has extensive ecosystem of scientific libraries and environments like NumPy (http://numpy.scipy.org) for Numerical Python, Pandas (https://pandas.pydata.org/) for Data science, SciPy (http://www.scipy.org) Scientific Python, Matplotlib (http://www.matplotlib.org) graphics library. It has great performance due to close integration with time-tested and highly optimized codes written in C and Fortran.[3]

Python has good support for Parallel processing with processes and threads, Inter process communication (MPI), GPU computing with OpenCL and CUDA, readily available and suitable for use on high-performance computing clusters and No license costs, no unnecessary use of research budget.

Further in the report, given detailed description about installing Python, keywords or tokens of python, Operators of python, Data types and flow control of python. Also, same details will discuss about different libraries NumPy, SciPy, Pandas, Matplotlib. And at last project done using discussed things.

Chapter 2

Python Programming

2.1 Installing Python

- At the very first for installing Python in Windows go to official site of Python. https://www.python.org/downloads/windows/[1]

Fig 2.1 Download Page for Python

- In the Download page go to latest release of python like Python 3.7.4.

- Then select x64 or x86 file based on your computer configuration. And download installer.

Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64	9b00c8cf6d9ec0b9abe83184a40729a2	7504391	SIG
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64	a702b4b0ad76debdb3043a583e563400	26680368	SIG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64	28cb1c608bbd73ae8e53a3bd351b4bd2	1362904	SIG
Windows x86 embeddable zip file	Windows		9fab3b81f8841879fda94133574139d8	6741626	SIG
Windows x86 executable installer	Windows		33cc602942a54446a3d6451476394789	25663848	SIG
Windows x86 web-based installer	Windows		1b670cfa5d317df82c30983ea371d87c	1324608	SIG

Fig 2.2 Installer page for windows

- Install downloaded .exe file.

Fig 2.3 Installer dialog box

- Add Python 3.7 to PATH otherwise you will have to do it explicitly. It will start installing python on windows.

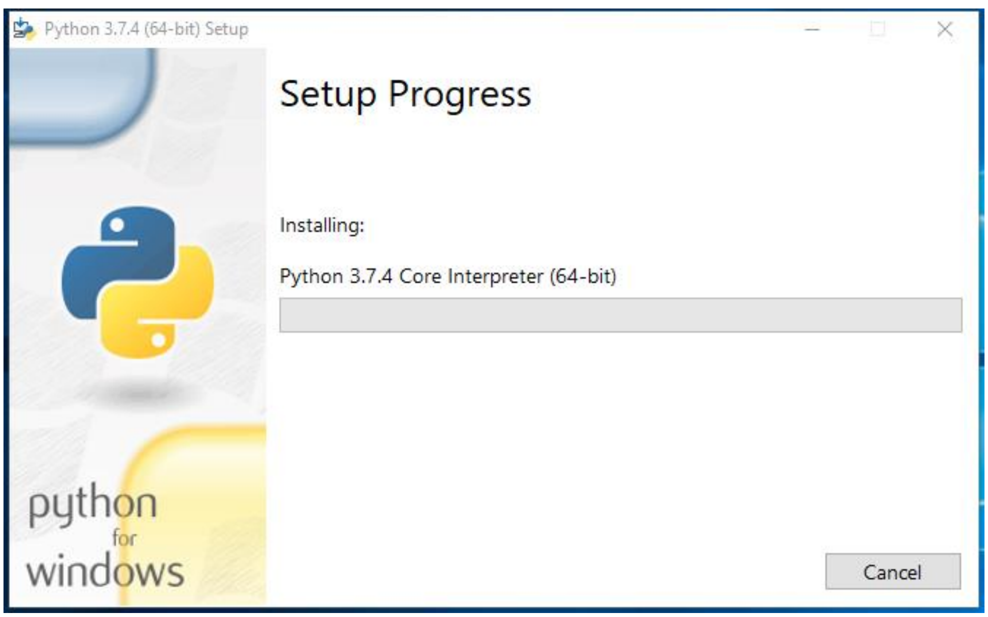

Fig 2.4 Installing Preview

- After installation is complete click on **Close**.

- Now go to windows and CMD and type Python.

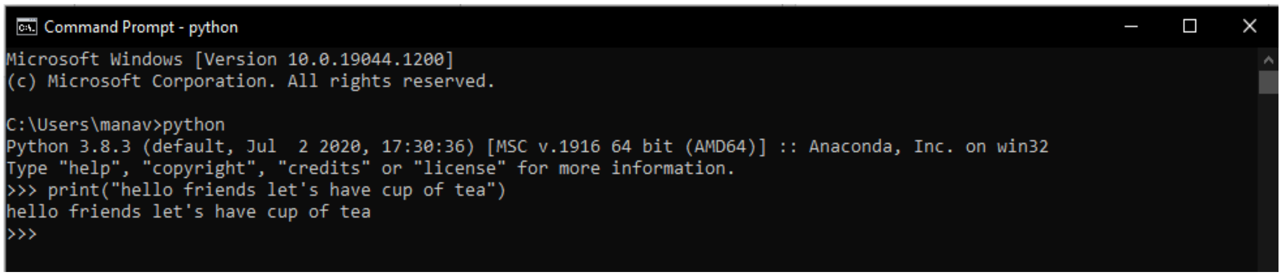

Fig 2.5 Installed python view from command prompt

You can also install **Anaconda, Spider, Atom and Google Collab** for python IDE

- Also, there are different methods for installing python in Linux, Mac OS, Unix, Android, iOS and many more systems.
- Where Google Collab is totally Cloud based IDE where every program runs in google cloud platform and use utilize TensorFlow framework.

2.2 Keywords of Python

There are **33 keywords** in python. List is given below.

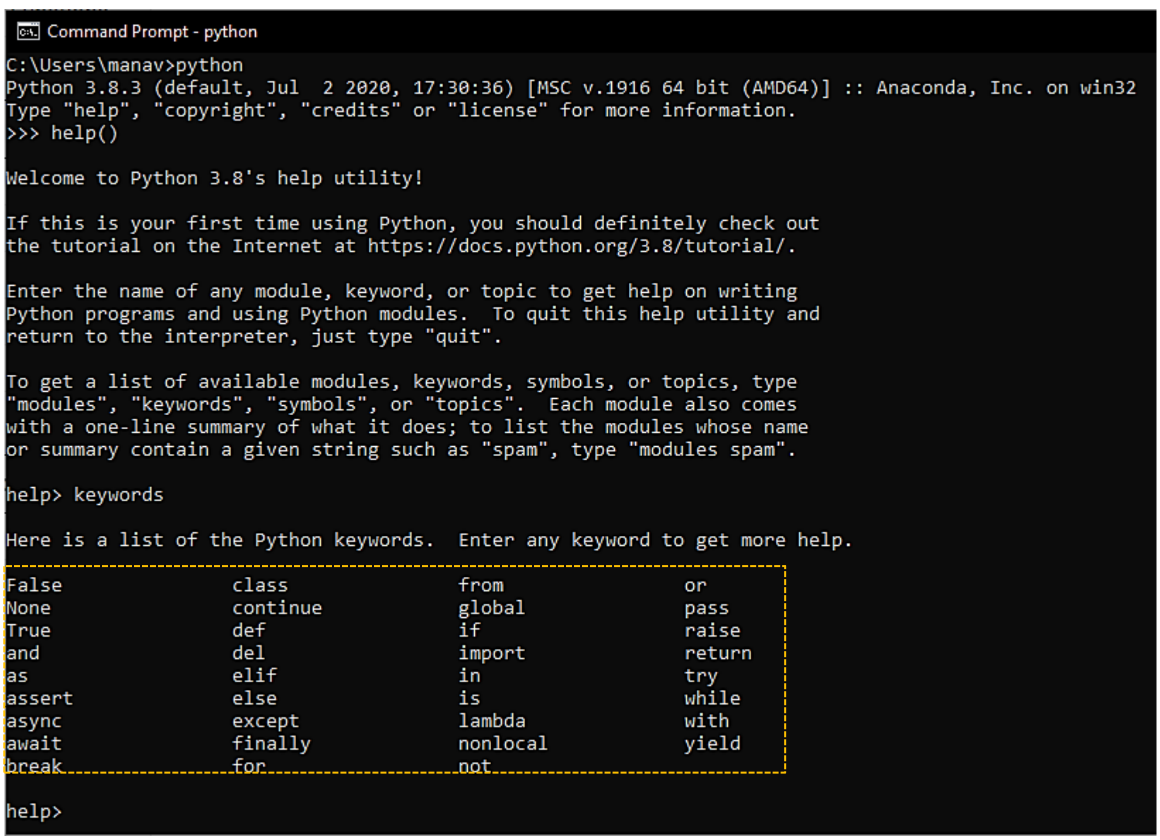

Fig 2.6 Keywords of Python

2.3 Operators of Python

There are **Seven** types of operators in Python

Fig 2.7 Operators of Python

- Arithmetic Operator
    - + (Addition)
    - – (Subtraction)
    - * (Multiplication)
    - / (Division)
    - ** (Exponentiation)
    - // (Floor division)
    - % (Modulus)
- Relational Operator
    - > (Greater than)
    - < (Less than)
    - == (Equal to)
    - != (Not equal to)
    - >= (Greater than or equal to)
    - <= (Less than or equal to)
- Assignment Operator
    - = (Assign)
    - += (Add and assign)
    - -= (Subtract and assign)
    - *= (Multiply and assign)
    - /= (Divide and assign)
    - %= (Modulus and assign)
    - **= (Exponentiation and assign)
    - //= (Floor-divide and assign)

- Logical Operator
  - and (Logical and)
  - or (Logical or)
  - not (Logical not)
- Bitwise Operator
  - & (Bitwise and)
  - | (Bitwise or)
  - ^ (Bitwise xor)
  - ~ (Bitwise 1's complement)
  - << (Bitwise left-shift)
  - >> (Bitwise right-shift)
- Membership Operator
  - In
  - Not in
- Identity Operator
  - Is
  - Is not

2.4 Datatypes of Python

There are 5 types of Data types in python

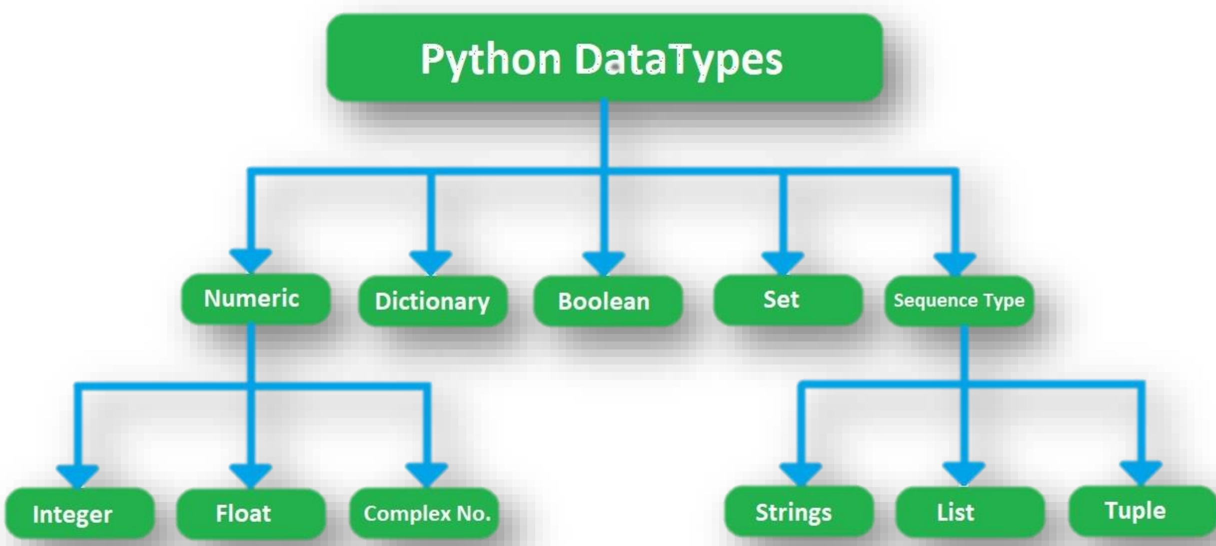

Fig 2.8 Datatypes of Python

Declaration of every Datatypes:

- Numeric Datatype
  - Integer: x = int(50)
  - Float: x = float(1.5)
  - Complex: x = complex(2+4j)
- Dictionary Datatype:

  x = dict(name="John", age=36)
- Boolean Datatype:

  x = bool(1)
- Sequence Datatype
  - String:

    x = str("Hello World')
  - List:

    x = list(("apple", "banana", "cherry"))
  - Tuple:

    x = tuple(("apple", "banana", "cherry"))

2.5 Control Flow of Python

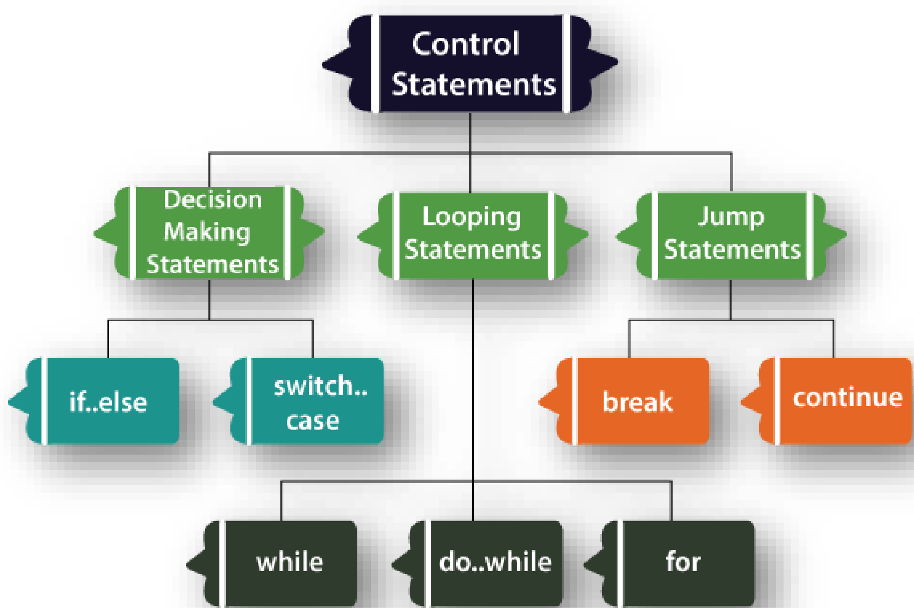

Fig 2.9 Control Flow of Python

- If … else statement:

```
if expression:
    statement(s)
elif expression:
    statement(s)
elif expression:
    statement(s)
...
else:
    statement(s)
```

- Switch ... case statement:

```
def week(i):
        switcher={
                0:'Sunday',
                1:'Monday',
                2:'Tuesday',
                3:'Wednesday',
                4:'Thursday',
                5:'Friday',
                6:'Saturday'
             }
         return switcher.get(i,"Invalid day of week")
```

- While loop:

```
while expression:
    statement(s)
```

- For loop:

```
for target in iterable:
    statement(s)
```

- Break statement:

```
while True:       # this loop can never terminate naturally
    x = get_next( )
    y = preprocess(x)
    if not keep_looping(x, y): break
    process(x, y)
```

- Continue statement:

```
for x in some_container:
    if not seems_ok(x): continue
    lowbound, highbound = bounds_to_test( )
    if x<lowbound or x>=highbound: continue
    if final_check(x):
        do_processing(x)
```

Chapter 3

NumPy

3.1 Introduction

The NumPy package (module) is used in almost all numerical computation using Python. It is a package that provide high-performance vector, matrix and higher-dimensional data structures for Python. It is implemented in C and Fortran so when calculations are vectorized (formulated with vectors and matrices), performance is very good.[3]

3.2 Creating NumPy arrays

There are many ways to initialize new NumPy arrays, for example from

- a Python list or tuples:

  v = array([1,2,3,4])

- using functions that are dedicated to generating NumPy arrays, such as arange, linspace, etc.

  x = arange(0, 10, 1)

  x = linspace(0, 10, 25)

  x, y = mgrid[0:5, 0:5]

- reading data from files like Comma-separated values (CSV), Numpy's native file format etc.

3.3 Features of NumPy

- Manipulating arrays
  - Indexing
  - Index slicing
  - Fancy indexing
- Functions for extracting data from arrays and creating arrays
  - Where: indices = where(mask)
  - Diag: diag(A)
  - Take: v2.take(row_indices)
  - Choose: which = [1, 0, 1, 0] choices = [[-2,-2,-2,-2], [5,5,5,5]]

    choose(which, choices)

- Linear algebra
    - Scalar-array operations
    - Element-wise array-array operations
    - Matrix algebra
    - Array/Matrix transformations
    - Matrix computations
        - Inverse
        - Determinant
    - Data processing
        - Mean
        - standard deviations and variance
        - min and max
        - sum, prod, and trace
    - Computations on subsets of arrays
    - Calculations with higher-dimensional data
- Reshaping, resizing and stacking arrays
- Adding a new dimension: newaxis
- Stacking and repeating arrays
    - tile and repeat
    - concatenate
    - hstack and vstack
- Copy and “deep copy"
- Iterating over array elements
- Vectorizing functions
- Using arrays in conditions
- Type casting

Chapter 4

SciPy

4.1 Introduction

The SciPy framework builds on top of the low-level NumPy framework for multi-dimensional arrays, and provides a large number of higher-level scientific algorithms[3].

To access the SciPy package in a Python program, we start by importing everything from the SciPy module. from scipy import *

4.2 Features of SciPy

- Supports Special functions
- Preform Integration of different types
    - Numerical integration: quadrature
- Solve Ordinary differential equations (ODEs)
- Solve Fourier transform
- Perform Linear Algebra. like,
    - Linear equation systems
    - Eigenvalues and eigenvectors
    - Matrix operations
    - Sparse matrices
- Perform Optimization. Like,
    - Finding a minima
    - Finding a solution to a function
- Perform Interpolation
- Find statistics

Chapter 5

Pandas

5.1 Introduction

Pandas is a newer package built on top of NumPy and pandas objects are valid arguments to most NumPy functions: [3]

- fast and efficient **Series (1-dimensional) and Data Frame (2-dimensional) heterogeneous objects** for data manipulation with integrated indexing
- tools for **reading and writing data from different formats**: CSV and text files, Microsoft Excel, SQL databases, HDF5...
- intelligent **label-based slicing**
- **time series-functionality**
- integrated **handling of missing data**
- For calling Pandas Library `import pandas as pd`

5.2 Features of Pandas

- **delim_whitespace:** Boolean, default False. Specifies whether or not whitespace (e.g. ' ' or ' ') will be used as the sep.
- **parse_dates**: boolean or list of ints or names or list of lists or dict, default False boolean. dict, e.g. {'foo' : [1, 3]} -> parse columns 1, 3 as date and call result 'foo'
- **index_col**: int or sequence or False, default None. Column to use as the row labels of the DataFrame.
- **skiprows:** list-like or integer, default None. Line numbers to skip (0-indexed) or number of lines to skip (int) at the start of the file
- **header**: int or list of ints, default 'infer'. Row number(s) to use as the column names, and the start of the data. Default behavior is as if set to 0 if no names passed, otherwise None.

Chapter 6

Matplotlib

6.1 Introduction

Matplotlib is an excellent 2D and 3D graphics library for generating scientific figures. Some of the many advantages of this library include: [3]

- Easy to get started
- Support for LATEX formatted labels and texts
- Great control of every element in a figure, including figure size and DPI.
- High-quality output in many formats, including PNG, PDF, SVG, EPS, and PGF.
- GUI for interactively exploring figures and support for headless generation of figure files (useful for batch jobs).

6.2 Features of Matplotlib

- MATLAB-like API
- The matplotlib object-oriented API
    - Figure size, aspect ratio and DPI
    - Saving figures
    - Legends, labels and titles
    - Formatting text: LaTeX, fontsize, font family
    - Setting colors, linewidths, linetypes
    - Control over axis appearance
    - Placement of ticks and custom tick labels
    - Axis number and axis label spacing
    - Axis grid
    - Axis spines
    - Twin axes
    - Axes where x and y is zero
    - Other 2D plot styles
    - Text annotation
    - Figures with multiple subplots and insets
    - Colormap and contour figures

- 3D figures
    - Surface plots
    - Wire-frame plot
    - Coutour plots with projections
    - Change the view angle
  - Animations
  - Backends
    - Generating SVG with the svg backend
    - The IPython notebook inline backend
    - Interactive backend

Chapter 7

SymPy

7.1 Introduction

There are two notable Computer Algebra Systems (CAS) for Python:

- **SymPy** - A python module that can be used in any Python program, or in an IPython session, that provides powerful CAS features. [3]
- **Sage** - Sage is a full-featured and very powerful CAS environment that aims to provide an open-source system that competes with Mathematica and Maple. Sage is not a regular Python module, but rather a CAS environment that uses Python as its programming language.

7.2 Features of SymPy

- Supports Different Symbolic variables
  - Complex numbers
  - Rational numbers
- Perform Numerical evaluation
- Supports Algebraic manipulations. Like,
  - Expand and factor
  - Simplify
  - apart and together
- Solve calculus
  - Differentiation
  - Integration
    - Sums and products
- Define limits
- Supports series
- Performs Linear algebra
  - Matrices
- Solving equations

Chapter 8

Project “Generate 3D model of bones from CT scan images”

8.1 Description

In this project taken CT scan file of patient and convert into 3D model in STL file format. For that use CT scan images which are in Dicom file format. First I had to understand Dicom File format.

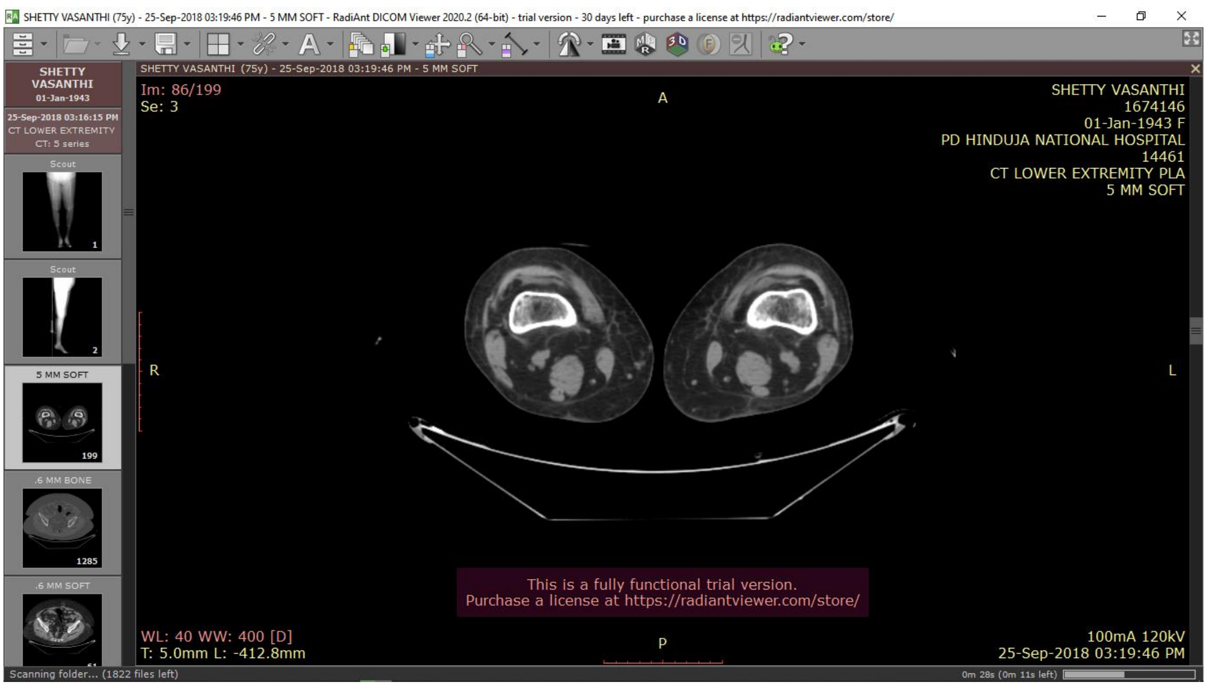

Fig 8.1 Radiant Dicom Viewer Software

For that I Used **Radiant Dicom Viewer** Software from which I could understand files and how it works. After that I knew that it’s a series of image capture in some distance. And every image contains single slice of scan part.

Therefore, I had to do some image processing to extract Bones from Images which contains specific pixel values in image.

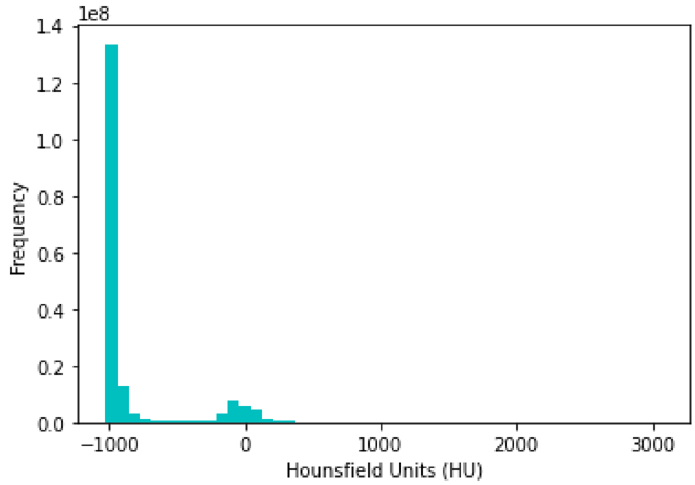

Fig 8.2 Histrogram of Images

From **Histrogram mapping** I extracted bone data. After Extracting data from every image, I had to convert 3d Model.

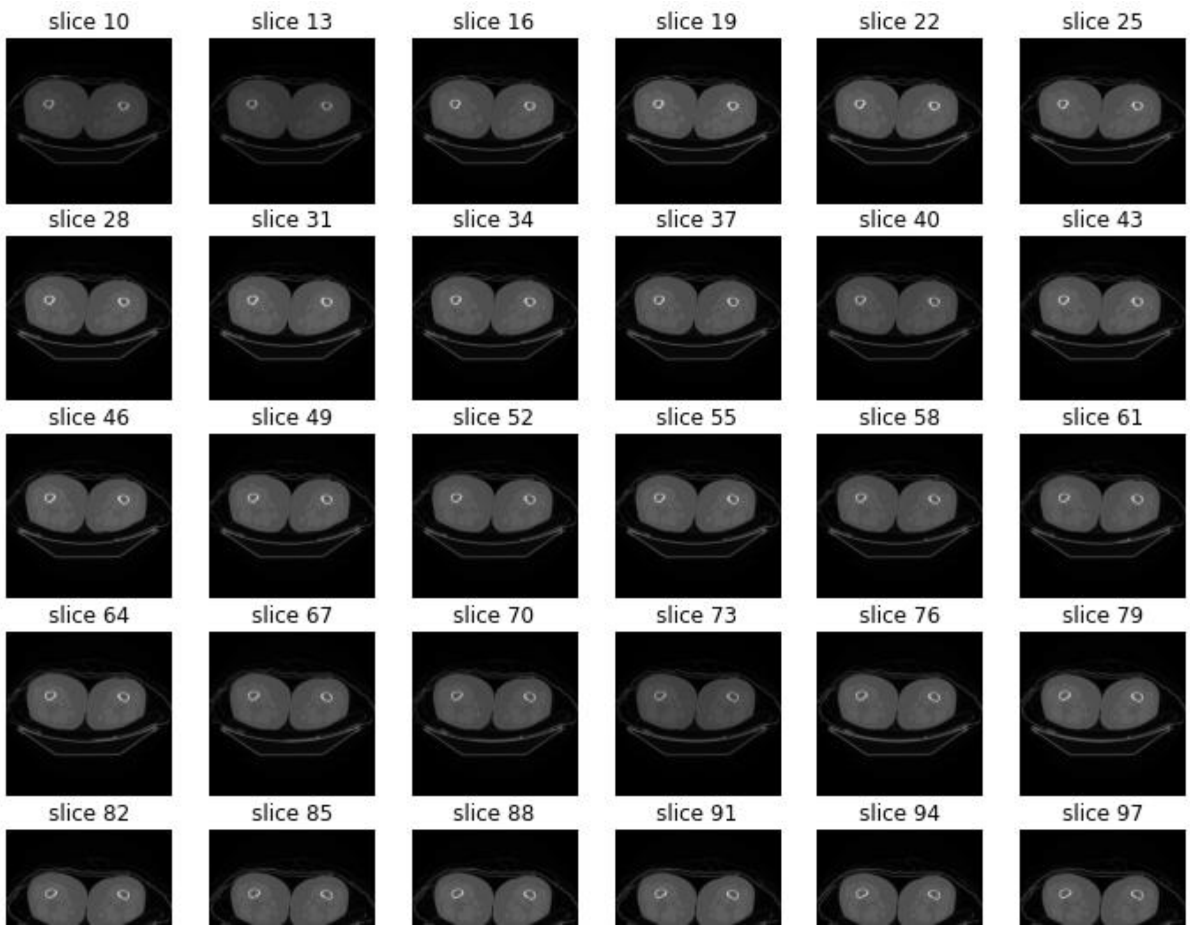

Fig 8.3 Sliced images of mapped data

Therefore, first need to find spacing between images means at which distance every image is taken or what is the distance between 2 images.

From that resample all data and created a **mesh** to visualize in 3d plot. From **Mesh** I created vertices and faces for STL file format and export the STL File.

8.2 Output

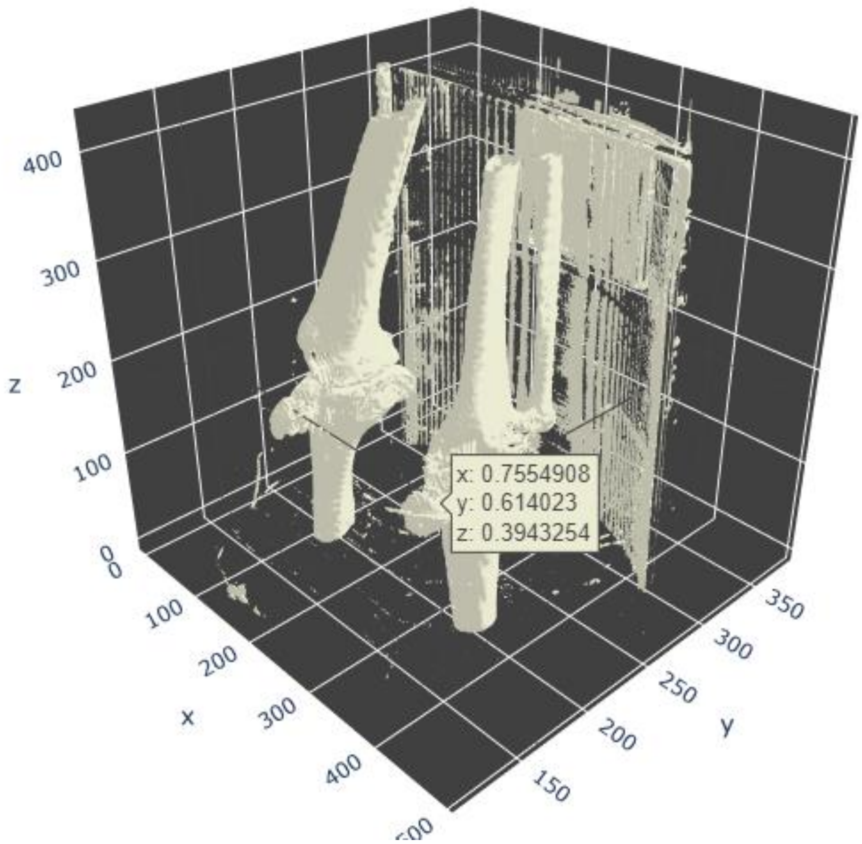

Fig 8.4 3D plot of processed image via meshing

Chapter 9

Conclusion

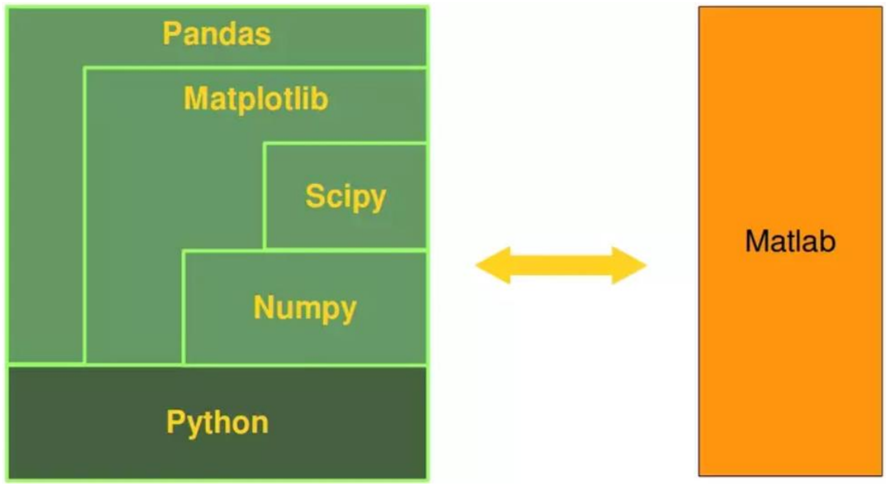

Fig 9.1 Comparison between MATLAB and Python

MATLAB is most preferred language for Scientific and numerical computation. But only drawback for using is platform dependency. Where Python can run independent of platform. Python is a general-purpose language. Nevertheless, Python is also - in combination with its specialized modules, like NumPy, SciPy, Matplotlib, Pandas and so, - an ideal programming language for solving numerical problems. Furthermore, the community of Python is a lot larger and faster growing than the one from R.

In this research comparison of NumPy, SciPy, Matplotlib, Pandas, SymPy is given. From comparison it is clear that NumPy is use for mainly for numerical computation and multi array manipulation. Where SciPy is for more scientific research and do derivation and integration kind of work. Where Pandas is for work with data science and data manipulation with different frameworks. Matplotlib is for plotting 2D or 3D plots and visualization of data. It has same facility like MATLAB. Where SymPy is to solve Different Symbolic variables and ODE. Which is complete accumulation for Scientific and numerical calculation.

At the and in hands-on project worked on title “Generate 3D model of bones from CT scan images”. Taken CT scan file of patient and convert into 3D model in STL file format.

References

[1] S. Nath, "Scientific computing using python," *Iitk.ac.in*. [Online]. Available: https://www.cse.iitk.ac.in/users/swaprava/courses/python/python.pdf. [Accessed: 09-Nov-2021].

[2] B. Adibhatla, "Top 5 Python libraries and packages for numeric and scientific applications," *Analyticsindiamag.com*, 30-Nov-2018. [Online]. Available: https://analyticsindiamag.com/top-5-python-libraries-and-packages-for-numeric-and-scientific-applications/. [Accessed: 09-Nov-2021].

[3] v0 1., "Numerical and scientific computing in python," *Www.bu.edu*. [Online]. Available: https://www.bu.edu/tech/files/2020/02/Numerical-and-Scientific-Computing-in-Python-v0.1.2.pdf. [Accessed: 09-Nov-2021].

[4] "Numpy, Matplotlib & Scipy Tutorial," *Python-course.eu*. [Online]. Available: https://www.python-course.eu/numerical_programming_with_python.php. [Accessed: 09-Nov-2021].

[5] *Scipy.org*. [Online]. Available: https://www.scipy.org/about.html. [Accessed: 09-Nov-2021].

[6] "1. Getting started with Python for science — Scipy lecture notes," *Scipy-lectures.org*. [Online]. Available: https://scipy-lectures.org/intro/. [Accessed: 09-Nov-2021].

[7] Wikipedia contributors, "Python (programming language)," *Wikipedia, The Free Encyclopedia*, 09-Nov-2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Python_(programming_language)&oldid=1054309711. [Accessed: 09-Nov-2021].

[8] "5 roles for Python in real-time embedded systems," *Beningo.com*, 10-Oct-2019. [Online]. Available: https://www.beningo.com/5-roles-for-python-in-real-time-embedded-systems/. [Accessed: 09-Nov-2021].

Appendix A

Project Code

```
!lscpu
!pip install plotly==4.11.0
import os
from google.colab import drive
drive.mount('/content/drive')
os.chdir('/content/drive/My Drive/Dicom_Leg')
!ls
!pip install pydicom
!pip install chart_studio
# common packages
import numpy as np
import os
import copy
from math import *
import matplotlib.pyplot as plt
from functools import reduce
# reading in dicom files
import pydicom
# skimage image processing packages
from skimage import measure, morphology
from skimage.morphology import ball, binary_closing
from skimage.measure import label, regionprops
# scipy linear algebra functions
from scipy.linalg import norm
import scipy.ndimage
# ipywidgets for some interactive plots
from ipywidgets.widgets import *
import ipywidgets as widgets
# plotly 3D interactive graphs
import plotly
from plotly.graph_objs import *
import chart_studio.plotly as py
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
from plotly.tools import FigureFactory as FF
from plotly.offline import download_plotlyjs, init_notebook_mode, plot,
 iplot
def load_scan(path):
  slices = []
  for root, dirs, files in os.walk(path):
```

```
  path = root.split(os.sep)
  for file in files:
    slices.append(pydicom.dcmread('/'.join(path) + '/' + file))

 slices = [s for s in slices if 'SliceLocation' in s]
 slices.sort(key = lambda x: int(x.InstanceNumber))
 try:
  slice_thickness = np.abs(slices[0].ImagePositionPatient[2] - slices[1].ImagePositionPatient[2])
 except:
  slice_thickness = np.abs(slices[0].SliceLocation - slices[1].SliceLocation)
 for s in slices:
   s.SliceThickness = slice_thickness
 return slices
def get_pixels_hu(scans):
 print(len(scans))
 image = np.stack([s.pixel_array for s in scans if len(s.pixel_array) == 512])
 image = image.astype(np.int16)
 # Set outside-of-scan pixels to 0
 # The intercept is usually -1024, so air is approximately 0
 image[image == -2000] = 0

 # Convert to Hounsfield units (HU)
 intercept = scans[0].RescaleIntercept
 slope = scans[0].RescaleSlope

 if slope != 1:
   image = slope * image.astype(np.float64)
   image = image.astype(np.int16)

 image += np.int16(intercept)

 return np.array(image, dtype=np.int16)
# set path and load files
path = './A/A/B'
path1 = './A/A/A'
path2= './A/A/C'

print('DEBUG : Loading Files.')
patient_dicom = load_scan(path)
print('DEBUG : Files Loaded.')

from collections import Counter
```

```python
Counter([len(s.pixel_array) for s in patient_dicom])
patient_pixels = get_pixels_hu(patient_dicom)
file_used= patient_pixels
imgs_to_process = file_used.astype(np.float64)

plt.hist(imgs_to_process.flatten(), bins=50, color='c')
plt.xlabel("Hounsfield Units (HU)")
plt.ylabel("Frequency")
plt.show()
id = 0
imgs_to_process = patient_pixels

def sample_stack(stack, rows=6, cols=6, start_with=10, show_every=3):
    fig,ax = plt.subplots(rows,cols,figsize=[12,12])
    for i in range(rows*cols):
        ind = start_with + i*show_every
        ax[int(i/rows),int(i % rows)].set_title('slice %d' % ind)
        ax[int(i/rows),int(i % rows)].imshow(stack[ind],cmap='gray')
        ax[int(i/rows),int(i % rows)].axis('off')
    plt.show()

sample_stack(imgs_to_process)
print(f"Slice Thickness:{patient_dicom[0].SliceThickness}")
print(f"Pixel Spacing (row, col): ({patient_dicom[0].PixelSpacing[0]}, {patient_dicom[0].PixelSpacing[1]}). ")
id = 0
imgs_to_process = patient_pixels
def resample(image, scan, new_spacing=[1,1,1]):
  # Determine current pixel spacing
  spacing = map(float, ([scan[0].SliceThickness] + list(scan[0].PixelSpacing)))
  spacing = np.array(list(spacing))

  resize_factor = spacing / new_spacing
  new_real_shape = image.shape * resize_factor
  new_shape = np.round(new_real_shape)
  real_resize_factor = new_shape / image.shape
  new_spacing = spacing / real_resize_factor

  image = scipy.ndimage.interpolation.zoom(image, real_resize_factor)

  return image, new_spacing

print("Shape before resampling\t", imgs_to_process.shape)
```

```
imgs_after_resamp, spacing = resample(imgs_to_process, patient_dicom, [1,1,1])
#imgs_after_resamp = imgs_to_process
print("Shape after resampling\t", imgs_after_resamp.shape)
def make_mesh(image, threshold=226, step_size=1):

    print("Transposing surface")
    p = image.transpose(2,1,0)

    print("Calculating surface")
    verts, faces, norm, val = measure.marching_cubes_lewiner(p, threshold, step_size=step_size, allow_degenerate=True)
    return verts, faces
import plotly.graph_objects as go
def plotly_3d(verts, faces):
    x,y,z = zip(*verts)

    print("Drawing")

    # Make the colormap single color since the axes are positional not intensity.
#    colormap=['rgb(255,105,180)','rgb(255,255,51)','rgb(0,191,255)']
    colormap=['rgb(236, 236, 212)','rgb(236, 236, 212)']

    fig = FF.create_trisurf(x=x,
                        y=y,
                        z=z,
                        plot_edges=False,
                        colormap=colormap,
                        simplices=faces,
                        backgroundcolor='rgb(64, 64, 64)',
                        title="Interactive Visualization")
    iplot(fig)

    #fig = go.Figure(data=go.Isosurface(
    #    x=x,
    #    y=y,
    #    z=z,
    #    value=faces,
    #    isomin=10,
    #    isomax=50,
    #    surface_count=5, # number of isosurfaces, 2 by default: only min and max
```

```
    #    colorbar_nticks=5, # colorbar ticks correspond to isosurface values
    #    caps=dict(x_show=False, y_show=False)
    #    ))
    #fig.show()

def plt_3d(verts, faces):
    print("Drawing")
    x,y,z = zip(*verts)
    fig = plt.figure(figsize=(10, 10))
    ax = fig.add_subplot(111, projection='3d')

    # Fancy indexing: `verts[faces]` to generate a collection of triangles
    mesh = Poly3DCollection(verts[faces], linewidths=0.05, alpha=1)
    face_color = [1, 1, 0.9]
    mesh.set_facecolor(face_color)
    ax.add_collection3d(mesh)

    ax.set_xlim(0, max(x))
    ax.set_ylim(0, max(y))
    ax.set_zlim(0, max(z))
    ax.set_fc((0.7, 0.7, 0.7))
    plt.show()
v, f = make_mesh(imgs_after_resamp,226,2)
plotly_3d(v, f)
pip install numpy-stl
import numpy as np
from stl import mesh
body = mesh.Mesh(np.zeros(f.shape[0], dtype=mesh.Mesh.dtype))
for i, fc in enumerate(f):
    for j in range(3):
        body.vectors[i][j] = v[fc[j],:]
body.save('body.stl')
import csv
file = open('f.csv','w')
writer = csv.writer(file, delimiter=',')
writer.writerows(f)
file.close()
```