Unit 1: Python Libraries

Introduction to Python

Python: Overview

Python is a **dynamically typed**, **interpreted** programming language
- Created by Guido van Rossum in 1991
- Maintained by the Python Software Foundation

Design philosophy: simple, readable code

Python syntax differs from R, Java, C/C++, MATLAB
- whitespace delimited
- limited use of brackets, semicolons, etc

Python: Overview

Python is a **dynamically typed**, **interpreted** programming language
- Created by Guido van Rossum in 1991
- Maintained by the Python Software Foundation

Design philosophy: simple, readable code

Python syntax differs from R, Java, C/C++, MATLAB
- whitespace delimited
- limited use of brackets, semicolons, etc

In many languages, when you declare a variable, you must specify the variable's **type** (e.g., int, double, Boolean, string). Python does not require this.

Python: Overview

Python is a **dynamically typed** **interpreted** programming language
- Created by Guido van Rossum in 1991
- Maintained by the Python Software Foundation

Design philosophy: simple, readable code

Python syntax differs from R, Java, C/C++, MATLAB
- whitespace delimited
- limited use of brackets, semicolons, etc

Some languages (e.g., C/C++ and Java) are **compiled**: we write code, from which we get a runnable program via **compilation**. In contrast, Python is **interpreted**: A program, called the **interpreter**, runs our code directly, line by line.

Compiled vs interpreted languages: compiled languages are (generally) faster than interpreted languages, typically at the cost of being more complicated.

Image credit: https://www.python.org/community/logos/

Data

- Data is raw information, and
- analysis of data is the systematic process of **interpreting** and **transforming** that data into **meaningful** insights.

Data Analysis

- Data analysis involves **inspecting**, **cleansing**, **transforming**, and **modeling** data to extract **useful information,** draw **conclusions**, and support **decision-making**.
- Typically focuses on interpreting **existing data** to find **patterns**, **trends**, **relationships** (within the dataset), and **insights**.
- **Common tools** include Excel, SQL, R, Python (with libraries like Pandas and Matplotlib), and statistical software.
- Provides descriptive statistics, visualizations, and reports that summarize the data.

Data Science

- Data science is a **multidisciplinary** field that uses scientific methods, processes, algorithms, and systems to **extract** knowledge and insights from structured and unstructured data.
- Encompasses **data analysis**, **predictive** modeling, **machine learning**, and **big data** technologies. It often involves **developing** new algorithms and **models** to make data-driven **decisions** or **predictions**.
- Includes tools and languages like Python, R, SQL, Hadoop, Spark, TensorFlow, and more. Data scientists also use advanced statistical techniques and machine learning algorithms.
- Builds predictive models, and automated systems, and provides deeper insights that can drive strategic decisions and innovations.

Data Analytics

- Data analytics refers to the process of examining datasets to draw conclusions about the information they contain, often with the help of specialized systems and software.
- It can **overlap** with both data analysis and data science but is often more focused on specific business applications.

Comparison

- **Data Analysis**: Understanding and interpreting **historical data**. (less complex, reporting, trend analysis, business intelligence)
- **Data Science**: Building models and algorithms for future predictions and automated decision-making. (more complex, product development, **personalized recommendation**, AI applicaition)
- **Data Analytics**: Applying data insights to specific business contexts and problems. (market analysis, customer insights, strategic planning)

Python

- Python – interpreted language (Python interpreter runs program by executing one statement at a time)
- Is dynamically typed language
- Install – pip install python
- Standard interactive python interpreter – is invoked on command line with **python** command

```
C:\Users\SONAM>python
Python 3.12.4 (tags/v3.12.4:8e8a4ba, Jun  6 2024, 19:30:16) [MSC v.1940 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- Python script file is saved as **.py** files

Introduction to iPython and Jupyter Notebook

Ipython

- Interactive command-line terminal for python
- is the component in the standard scientific Python toolset that ties everything together.
- provides a robust and productive environment for interactive and exploratory computing.
- is an enhanced Python shell designed to accelerate the writing, testing, and debugging of Python code.
- also provides A Mathematica-like HTML notebook for connecting to IPython through a web browser
- IPython is designed for both interactive computing and software development work.
- installation –pip install ipython

Introduction to iPython and Jupyter Notebook

Ipython Basics

- Ipython command is used to launch

```
C:\Users\SONAM>ipython
Python 3.12.4 (tags/v3.12.4:8e8a4ba, Jun  6 2024, 19:30:16) [MSC v.1940 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.26.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: a=5

In [2]: a
Out[2]: 5

In [3]:
```

Introduction to iPython and Jupyter Notebook

Jupyter notebook – web-based

- Use to work with larger blocks of code
- Is a type of interactive document for code, text, data visualization
- **Install** from cmd in window – pip install notebook
- Uses command – jupyter notebook
- File - **.ipynb**

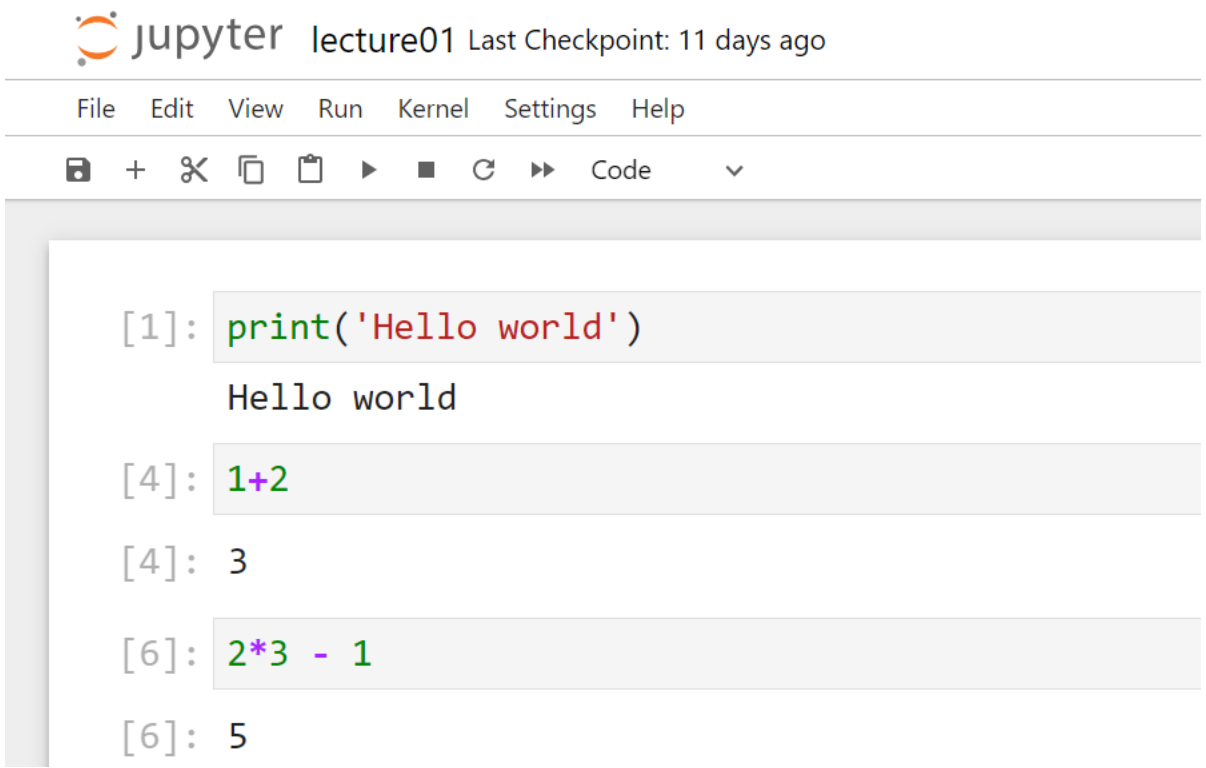
Introduction to iPython and Jupyter Notebook

Jupyter notebook

```
C:\Users\SONAM>jupyter notebook
[I 2024-08-16 14:58:45.312 ServerApp] Extension package jupyter_lsp took 0.2671s to import
[I 2024-08-16 14:58:46.032 ServerApp] Extension package jupyter_server_terminals took 0.7263s to import
[I 2024-08-16 14:58:47.794 ServerApp] jupyter_lsp | extension was successfully linked.
[I 2024-08-16 14:58:47.794 ServerApp] jupyter_server_terminals | extension was successfully linked.
[I 2024-08-16 14:58:47.810 ServerApp] jupyterlab | extension was successfully linked.
[I 2024-08-16 14:58:47.820 ServerApp] notebook | extension was successfully linked.
[I 2024-08-16 14:58:48.375 ServerApp] notebook_shim | extension was successfully linked.
[I 2024-08-16 14:58:48.689 ServerApp] notebook_shim | extension was successfully loaded.
[I 2024-08-16 14:58:48.704 ServerApp] jupyter_lsp | extension was successfully loaded.
[I 2024-08-16 14:58:48.704 ServerApp] jupyter_server_terminals | extension was successfully loaded.
[I 2024-08-16 14:58:48.723 LabApp] JupyterLab extension loaded from E:\SONAM\Python\Lib\site-packages\jupy
[I 2024-08-16 14:58:48.723 LabApp] JupyterLab application directory is E:\SONAM\Python\share\jupyter\lab
[I 2024-08-16 14:58:48.723 LabApp] Extension Manager is 'pypi'.
```

Introduction to iPython and Jupyter Notebook

Jupyter notebook



Essential Python Libraries

- **NumPy**
- Short for **Numerical Python**, is the foundational package for scientific computing in Python.
- For numerical data, NumPy arrays are a much more efficient way of storing and manipulating data than the other **built-in** Python **data structures**.
- many other python libraries are built on NumPy

Link: http://www.numpy.org/

Essential Python Libraries

- **Pandas**
- adds data structures and tools designed to work with table-like data – DataFrame (a twodimensional tabular, column-oriented data structure with both row and column labels)
- provides tools for data manipulation: reshaping, merging, sorting, slicing, aggregation etc.
- allows handling missing data

Link: http://pandas.pydata.org/

Essential Python Libraries

- **Matplotlib**
- matplotlib is the most popular Python library for producing plots and other 2D data visualizations.
- It is well-suited for creating plots suitable for publication.
- integrates well with IPython, thus providing a comfortable interactive environment for plotting and exploring data.
- line plots, scatter plots, barcharts, histograms, pie charts etc.

Link: https://matplotlib.org/

Essential Python Libraries

- **SciPy**
- SciPy is a **collection** of packages addressing a number of different standard problem domains in scientific computing.
- collection of algorithms for **linear algebra**, **differential equations**, numerical integration, optimization, statistics and more
- built on NumPy
- Together, NumPy and SciPy form a reasonably complete and mature computational foundation for many traditional scientific computing applications.

Link: https://www.scipy.org/scipylib/

Essential Python Libraries

- **SciKit-Learn**:
- provides machine learning algorithms: classification, regression, clustering, model validation etc.

- built on NumPy, SciPy and matplotlib

Link: http://scikit-learn.org/

Essential Python Libraries

- **Statsmodels**
- Is a statistical analysis packages, implemented number of regression analysis models
- Compared to scikit-learn, statsmodels contains algorithms for classical (primary frequents) statistics and econometrics.
- Is focused more on statistical inference, providing uncertainity estimates and p-values for parameters, scikit-learn by contrast, is more prediction -focused

Link: https://www.scipy.org/scipylib/

Exercise

```
print('Hello world’)

print("Hello world!")

1

1+2

2*3
```

Exercise

2*3

2*3-1

2**7

6/3

8//3

Exercise

8%3

type(42)

type(2.7178)

type("bird")

Exercise

```
approx_pi = 3.141592
type(approx_pi)


pi_int = int(approx_pi)
type(pi_int)

pi_int
```

Exercise

```
int_from_str = int('8675309’)
type(int_from_str)

int_from_str

float_from_int = float(42)
type(float_from_int)
```

Exercise

goat_int = int('goat’)

answer = 2*does_not_exist

'one' * 'two’

'cat' + 'dog’

'goat'*3

Exercise

```
import math
rt2 = math.sqrt(2)
print(rt2)

a=2b=3
math.pow(a,b)
```

Exercise

```
def print_wittgenstein():
        print('Die Welt ist Alles')
        print('was der Fall ist.')
```

Exercise

```
def print_wittgenstein(bread):
        print(bread)
        print('Die Welt ist Alles')
        print('was der Fall ist.')
```

Python Internet modules

- Python offers several modules that make working with the internet and handling network-related tasks easier
- provides **two levels** of access to network programming.
- These are –
  - **Low-Level** Access: At the low level, you can access the basic socket support of the operating system. You can implement client and server for both connection-oriented and connectionless protocols.
  - **High-Level** Access: At the high level allows to implement protocols like HTTP, FTP, etc.

Python Internet modules

- **requests**
- The requests module is a simple and easy-to-use library for making HTTP requests.
- It allows you to send HTTP requests with methods such as GET and POST.
- **http client**
- The http.client module is part of Python's standard library and can be used for making HTTP requests.

Python Internet modules

- **urllib**
- The urllib module is a part of Python's standard library and provides a set of functions and classes for **URL handling** and **web requests**.
- **BeautifulSoup** (for web scraping)
- While not strictly for internet connections, BeautifulSoup is often used in conjunction with requests or urllib to parse HTML and extract data from web pages.

Python Internet modules

- **smtplib** (for sending emails)
- The smtplib module defines an SMTP client session object that can be used to send mail to any Internet machine with an SMTP or ESMTP listener daemon.
- **Socket**
- The socket module provides access to the **low-level** networking interface, enabling the **creation** of network **connections** and handling **data transfer**.

Basic Concept

- **Sockets**: A socket is an **endpoint** in a network communication.
- **Server**: A server listens for incoming connections.
- **Client**: A client initiates a connection to the server.
- **IP Address**: A unique address that identifies a device on a network.
- **Port**: An endpoint for network communication on a device.

Sockets library

- Python's socket module provides a way to communicate over the network using standard protocols like TCP/IP. It's a fundamental module for low-level network programming.
- **TCP and UDP**
- TCP (Transmission Control Protocol): **Reliable**, **connection-oriented** communication.
- UDP (User Datagram Protocol): **Unreliable**, **connectionless** communication.

Sockets Module/Library

- The primary socket API functions and methods in this module are:
  - socket()
  - .bind()
  - .listen()
  - .accept()
  - .connect()
  - .connect_ex()
  - .send()
  - .recv()
  - .close()

Creating TCP Server

we have to include the socket module

```
# import the socket library
import socket
```

```
#reserve port on server
port = 40674

# server address
server_address = 'localhost'
```

Creating TCP Server

```
# create a socket object - Create a TCP/IP socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
# bind to port
server_socket.bind((server_address, port))
# print ("socket binded to %s" %(port))
```

Creating TCP Server

```
# put the socket into listening mode
server_socket.listen()
print ("socket is listening")
```

Creating TCP Server

```
# loop connection with client until interrupted
while True:
    #establish the connection with client
    client_socket, client_address=server_socket.accept()
    print ('Got connection from', client_address )

    # Receive data from client
    data = client_socket.recv(1024)
    print(f"Received: {data.decode()}")

    # close the connection with the client
    client_socket.close()
```

Creating TCP Client

```
# import the socket library
import socket

# Define server address and port
server_port   = 40674
server_address = 'localhost'

# Create a TCP/IP socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Creating TCP Client

```
# Create a TCP/IP socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
# Connect to the server
client_socket.connect((server_address, server_port))
```

Creating TCP Client

```
# Send and receive data
message = "Hello, Server!"
client_socket.sendall(message.encode())
```

```
# Close the connection
client_socket.close()
```

TCP Client

```
# Send and receive data
message = "Hello, Server!"
client_socket.sendall(message.encode())
```

TCP Server

```
# send message to client
client_socket.send(b'Thank you for connecting')
```