Example Packages, Libraries

INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI

Modules and Packages

- Packages like math() are part of standard library of python, which has many packages: https://docs.python.org/3/library/
- We have seen external package: request, which we have to install before importing
- Python provides a rich ecosystem of externally available packages for thousands of domains / problem areas
- Provides a convenient way of publishing a package and making it available through PyPI (Python Packaging Index)
- This extensibility and easy use of 3rd party packages makes python ecosystem very rich, versatile, and constantly evolving

The Python ‘os’ Module

Introduction and Overview

INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI

What is the “os” module?

- If your program wants to work with file system - we only know how to read or write files
- In a program how can we determine what files are there in the current directory, or what is the current directory, create a directory, etc ?
- The “os” module is a built-in Python library that provides functions for interacting with the file system of the operating system.
- It allows programs to perform tasks such as reading and writing files, navigating the file system, and executing shell commands

Examples of “os” module Functions

- As always, to use we have to import os by `import os` and then we can call functions provided by os
-
- “os.getcwd()”: Returns the current working directory.
- “os.listdir()”: Lists the contents of a directory
- “os.mkdir()”: Creates a new directory
- “os.rename()”: Renames a file or directory
- “os.rmdir()”: Removes an empty directory

Examples

Using getcwd():

```
import os
cwd = os.getcwd()
print(cwd)
```

Output:
“/home/user/documents” (or the current working directory on your system)

Using listdir():

```
files = os.listdir(directory)
print(files)
for file in files:
        f = open(file)
        s = f.readline()
        print(s)
```

Output:
```
[“file1.txt”,“file2.pdf”, … “file3.docx”]
```

First line of each file (assuming txt) # works only if no special file or directory

Examples

The “os.mkdir()” function is used to create a new directory.
Example code:

```
import os
new_dir =
'/home/user/documents/new_dir'
os.mkdir(new_dir)
```

- This creates a new directory called “new dir” inside the “documents” directory
- If the directory already exists, “os.mkdir()” will raise a “FileExistsError”

The “os.rename()” function is used to rename a file or directory.

Example code:

```
import os
old_name =
'/home/user/documents/old_name.txt'
new_name =
'/home/user/documents/new_name.txt'
os.rename(old_name, new_name)
```

- This renames the file old_name.txt to new_name.txt.
- If the new name already exists, it will be overwritten.

Other functions in os module

os.name() # gets the name of the OS

os.mkdir() # makes a directory in the current one

os.chdir(path) #changes the current directory

os.remove(path) #removes the file given by path

Has file reading functions also:

```
fd = os.open('file.txt', os.O_RDONLY)
data = os.read(fd, 10) # first 10 bytes
print(data.decode())  # as a string
os.close(fd)
```

For writing:

```
fd = os.open('file.txt', os.O_WRONLY)
data = b'Hello, world!'  # bytes object
os.write(fd, data)
os.close(fd)
```

Quiz

Which of the following is/are correct ?

(a) If a directory already exists, os.mkdir() raises an error if we create a new directory by the same name.

(b) If a directory already exists, os.mkdir() will overwrite it if we create a new directory by the same name.

(c) os.remove(“dir”) is used to remove non-empty directory “dir”.

(d) os.close(fd) closes the file represented by file descriptor fd.

Quiz – Solution

Which of the following is/are correct ?

(a) If a directory already exists, os.mkdir() raises an error if we create a new directory by the same name. ✅

(b) If a directory already exists, os.mkdir() will overwrite it if we create a new directory by the same name. ❌

(c) os.remove(“dir”) is used to remove non-empty directory “dir”.❌

(d) os.close(fd) closes the file represented by file descriptor fd. ✅

Answer: (a), (d)

(a) - It raises a FileExistsError

(c) - It is used only to remove empty directories

(d) - It is used to close the file

Datetime Module

Introduction

- It is a module in the standard library that provides classes for manipulating dates and times.
- The module includes several classes for working with dates and times, including:
  - datetime: This class provides a way to represent a specific point in time, including the date and time.
  - timedelta: This class represents a duration, which can be used to perform arithmetic operations with datetime objects.
  - date: This class provides a way to represent a specific date, without a specific time.
  - time: This class provides a way to represent a specific time, without a specific date.

date Class

- We need to import the date class from the datetime module.
- A date object represents a date (year, month and day).
    - Can also get the year, month, day from the date object using d.year, d.month, d.day respectively.
- We can create a date object containing the current date by using the class method named today().

```
from datetime import date
```

```
d = date(2023, 1, 17)
print(d)
```

```
Output: 2023-01-17
```

```
todays_date = date.today()
print("Today's date =", todays_date)
d = todays_date
print(f'{d.day}-{d.month}-{d.year}')
```

```
Output: Today's date = 2022-12-27
Output: 23-1-2023
```

datetime Class

- We need to import the datetime class from the datetime module.
- A datetime object represents information from both date and time objects.
    - Can also get the year, month, day from the datetime object using dt.year, dt.month, dt.day respectively.
    - Can get the current timestamp using dt.timestamp()
    - Can also access individual date and time objects of a datetime object using dt.date() and dt.time()
- We can create a datetime object containing the current datetime by using the class method named now().
- combine() method can be used to combine date and time objects into a datetime object
- We can also create datetime objects from a timestamp similar to date class.

```
from datetime import datetime
```

```
dt1 = datetime(2023, 1, 17)
dt2 = datetime(2023, 1, 17, 23, 55,
59, 342380)
print(f"dt1 = {dt1}\ndt2 = {dt2}")
```

```
Output: dt1 = 2023-01-17 00:00:00
        dt2 = 2023-01-17 23:55:59.342380
```

```
dt = datetime.now()
print("dt = ", dt )
print(f'{dt.year}, {dt.month},
{dt.hour}, {dt.minute}, {dt.second}')
```

```
Output:dt = 2023-01-17 12:29:22.715898
2023, 01, 17, 12, 29, 22
```

Some more Functionality

There are other operations available from datetime

Read yourself

Some extra slides given for you

time Class

- We need to import the time class from the datetime module.
- A time object instantiated from the time class represents the local time.
    - Can also get the hour, minute, second, microsecond detail from the time object using t.hour, t.minute, t.second, t.microsecond respectively.
    - Ranges : 0 <= hour < 24;
      0 <= minute < 60; 0 <= second < 60; 0 <= microsecond < 1000000

```
from datetime import time
```

```
a = time()
b = time(11, 34, 56)
c = time(hour = 11, minute = 34,
second = 56)
d = time(11, 34, 56, 234566)
print(f"a={a}\nb={b}\nc={c}\nd={d}")
```

```
Output:  a = 00:00:00
         b = 11:34:56
         c = 11:34:56
         d = 11:34:56.234566
```

timedelta Class

- We need to import the timedelta class from the datetime module.
- A timedelta object represents the difference between two dates or times.
  - Can get the hours, seconds, microseconds using td.hours, td.seconds, td.microseconds respectively.
- The difference between two date, two time or datetime objects is an object of timedelta class.
- We can add, subtract two timedelta objects. Operations like multiply divide are also supported on the same.

```
from datetime import timedelta
```

```
td = timedelta(days=4,hours=11,
minutes=4,seconds=54)
print(td)
```

```
Output: 4 days, 11:04:54
```

```
t1 = date(year=2018,month=7,day=12)
t2 = date(year=2017,month=12,day=23)
t3 = t1 - t2
print(t3, type(t3))
```

```
Output: 201 days, 0:00:00 <class
'datetime.timedelta'>
```

```
t1 = timedelta(weeks=2,days=1,hours=1,seconds=33)
t2 = timedelta(days=4,hours=11,minutes=4,seconds=54)
print(f"sum={t1+t2}\ndiff={t1-t2}\nmul={t2*2}\ndiv={
t1/3}")
```

```
Output: sum=19 days, 12:05:27
        diff=10 days, 13:55:39
        mul=8 days, 22:09:48
        div=5 days, 0:20:11
```

Formatting of date and time

- The way date and time is represented may be different in different places, organizations etc. E.g. formats like dd/mm/yyyy and mm/dd/yyyy are used at different places.
- strftime() and strptime() methods are used to handle this.
- The strftime() method creates a formatted string from a given date, datetime or time object.
    - %Y, %m, %d, %H etc. are format codes. T
    - It takes one or more format codes and returns a formatted string.
- The strptime() method creates a datetime object from a given string (representing date and time)
    - It takes two arguments:
        - a string representing date and time
        - format code equivalent to the first argument

```
now = datetime.now()
t = now.strftime("%H:%M:%S")
print("Time:", t)
dt1 = now.strftime("%m/%d/%Y, %H:%M:%S")
print("dt1:", dt1)
dt2 = now.strftime("%d/%m/%Y, %H:%M:%S")
print("dt2:", dt2)
```

```
Output: Time: 12:59:40
        dt1: 01/17/2023, 12:59:40
        dt2: 17/01/2023, 12:59:40
```

```
date_string = "25 December, 2022"
date_object = datetime.strptime(date_string,
"%d %B, %Y")
print("date_object =", date_object)
```

```
Output: date_object = 2022-12-25 00:00:00
```

Quiz(Multiple option correct)

Consider the following code :

```
from datetime import datetime
dt = datetime.now()
```

The code represents an object ‘dt’ of the datetime class from the datetime module.How can we get the month from the given datetime object dt ?

(a) dt.month

(b) dt.month()

(c) dt.date().month

(d) dt.date().month()

Quiz – Solution

Consider the following code :

```
from datetime import datetime
dt = datetime.now()
```

The code represents an object ‘dt’ of the datetime class from the datetime module.How can we get the month from the given datetime object dt ?

(a) dt.month ✅

(b) dt.month() ❌

(c) dt.date().month ✅

(d) dt.date().month() ❌

Answer : (a), (c)

Matplotlib Module

Introduction

- Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.
- Most of the Matplotlib utilities lies under the pyplot submodule.
- Matplotlib can be installed via pip by running the command:

```
pip install matplotlib
```

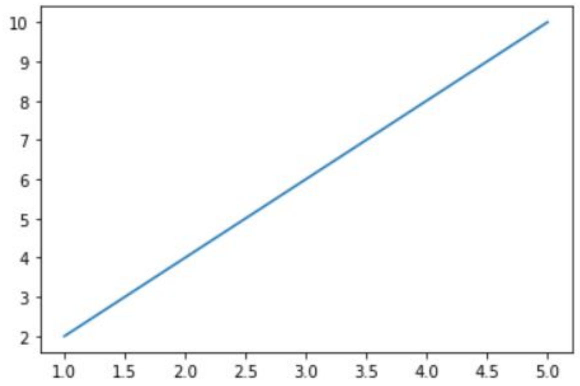
- To import the module into a Python script, use the following import statement

```
import matplotlib.pyplot as plt
```

Basic Plotting

```
import matplotlib.pyplot as plt

# Create some data
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

# Create a basic line plot
plt.plot(x, y)

# Show the plot
plt.show()
```

- This example shows how to create a basic line plot using the plot() function from the pyplot module.
- The plot() function takes in two lists of data, one for the x-axis and one for the y-axis, and creates a plot internally
- The show() function is used to display the plot (must call it)



Customization

Add titles to axis

Add title to the plot

Set limits to the axis values

Add markers in the plot

Choose color of the plot

Have multiple plots on one - legend for each

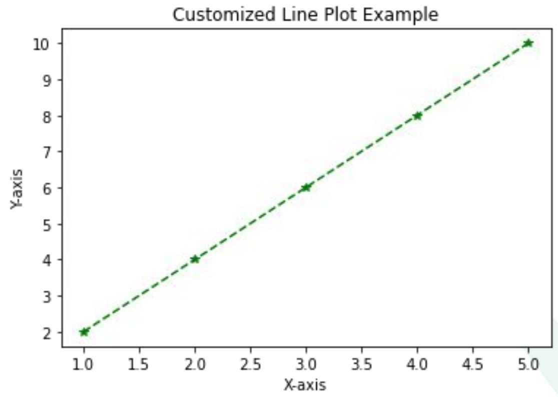
Different plot types:

- Line plot
- Scatter plot
- Bar plot
- Pie Chart

Customizing Plots – example

```
import matplotlib.pyplot as plt

# Create some data
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

# Create a line plot with custom colors and
marker
plt.plot(x, y, '--', color="green", marker='*')

# Add a title and labels for the x and y axis
plt.title("Customized Line Plot Example")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")

# plt.xticks(x)

# Show the plot
plt.show()
```

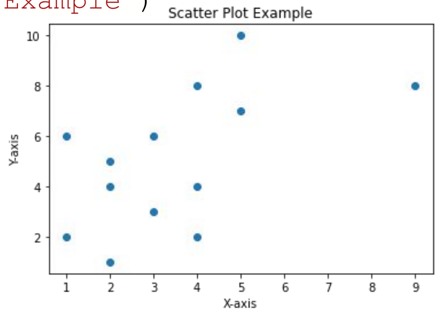
- This example shows how to customize the appearance of a plot.
- The plot() function takes in additional arguments to control the color and marker style of the line.
- The g-- argument makes the line green and dashed.
- The o argument makes the marker circles



Different Types of Plots

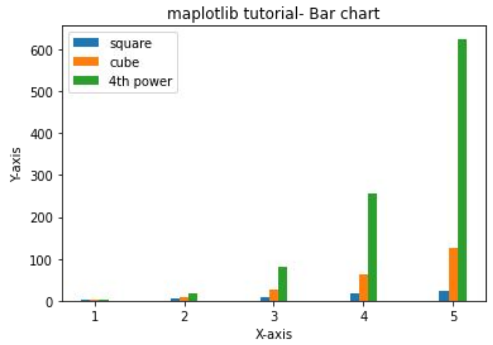
Scatter Plot

```
import matplotlib.pyplot as plt

# Create some data
x = [1, 2, 3, 4, 5, 2, 4, 3, 4, 2, 1, 5,9]
y = [2, 4, 6, 8, 10, 1, 2, 3, 4, 5, 6, 7, 8 ]

# Create a scatter plot
plt.scatter(x, y)

# Add a title and labels for the x and y axis
plt.title("Scatter Plot Example")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")

# Show the plot
plt.show()
```



Bar Plot

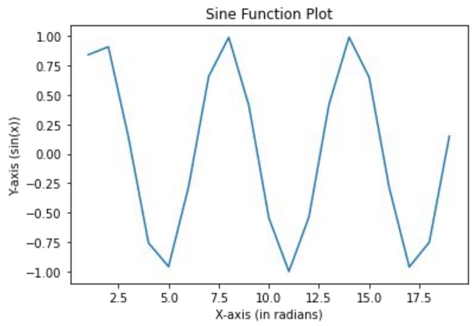
```
import numpy as np
import matplotlib.pyplot as plt

x=[1,2,3,4,5]
y1=[i**2 for i in x]
y2=[i**3 for i in x]
y3=[i**4 for i in x]

# plt.bar(x, y3, width=0.1)
# plt.bar(x, y2, width=0.1)
# plt.bar(x, y1, wi dth=0.1)


plt.bar([i-0.1 for i in x], y1, width=0.1, label="square")
plt.bar([i for i in x], y2, width=0.1, label="cube")
plt.bar([i+0.1 for i in x],y3, width=0.1, label="4th power")

# plt.ylim(0, 100)

plt.xlabel("X-axis")
plt.ylabel("Y-axis")

plt.legend()

plt.title("matplotlib tutorial- Bar chart" )
```



Plotting a function

- We have seen that for creating a plot we just need two lists, one for the x-axis and other for the y-axis
- For plotting a function func(), we just need to create a list containing the func(xi) for every xi in x

```
import matplotlib.pyplot as plt
import numpy as np
import math

# Create some data
x=[i for i in range(1,20)]
y=[math.sin(i) for i in x]

# Create a line plot
plt.plot(x, y)

# Add a title and labels for the x and y axis
plt.title("Sine Function Plot")
plt.xlabel("X-axis (in radians)")
plt.ylabel("Y-axis (sin(x))")

# plt.xticks(x)

# Show the plot
plt.show()
```



Quiz

How can you specify the color of a line plotted using Matplotlib?

a) Using the color parameter in the plot() function

b) Using the color parameter in the show() function

c) Using the color parameter in the display() function

d) Using the color parameter in the visualize() function

Quiz(Solution)

How can you specify the color of a line plotted using Matplotlib?

a) Using the color parameter in the plot() function

b) Using the color parameter in the show() function

c) Using the color parameter in the display() function

d) Using the color parameter in the visualize() function

a) Using the color parameter in the plot() function can be used to specify the color of a line plotted using Matplotlib.

numpy , pandas modules

Introduction and Overview

INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI

NumPy Module

Numpy Arrays

- Numpy arrays are similar to Python lists but they have several advantages.
- NumPy arrays have a fixed size and homogeneous type, while Python lists have a dynamic size and can store elements of different types.
- NumPy arrays are more memory efficient and faster than Python lists when performing element-wise operations.
- NumPy is particularly useful when working with large arrays and matrices of numerical data, or when performing mathematical operations on arrays.

Creating NumPy Arrays

There are several ways to create numpy arrays, such as:

- Using the arange function, which creates an array with a range of values: np.arange(0, 10, 1)
- Using the linspace function, which creates an array with a specified number of evenly spaced values: np.linspace(0, 10, 5)
- Using the zeros and ones functions, which create arrays filled with zeros or ones: np.zeros((3, 3)), np.ones((3, 3))

```
import numpy as np
#using python lists
my_list=[1,2,3]
my_np_arr=np.array(my_list)
print(my_np_arr)
print()

# Using arange function
a = np.arange(0, 10, 1)
print(a)
print()

# Using linspace function
b = np.linspace(0, 10, 5)
print(b)
print()

# Using zeros and ones function
c = np.zeros((3, 3))
d = np.ones((3, 3))
print(c)
print()
print(d)
```

Basic Properties of NumPy Arrays

- ndim

  ndim represents the number of dimensions of the numpy array.

- shape

  shape is a tuple of integers representing the size of the numpy array in each dimension

- size

  size is the total number of elements in the numpy array

- dtype

  dtype determines the data type of the elements of the numpy array

Array Indexing and Slicing

- Array indexing and slicing in numpy is similar to Python lists.
  - To access a specific element, use the square brackets: a[3]
  - To slice an array, use the colon operator: a[start:stop:step]
- Slicing and indexing can also be used to access sub-arrays and modify elements.
- Indexing and slicing can be used on both 1D and 2D arrays

```
import numpy as np

a = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])

# Indexing
print(a[3])

# Slicing
print(a[1:5])
print(a[::2])

# 2D arrays
b = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(b[1, 2])
print(b[:, 1:2])
```

Array Manipulation

Numpy provides functions for manipulating arrays, such as reshaping, transposing, and stacking.

- The reshape function can be used to change the shape of an array.
- The transpose function can be used to change the order of the axes: np.transpose(a)
- The concatenate function can be used to join multiple arrays along a specific axis: np.concatenate((a, b), axis=0)

```
import numpy as np

a = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])
b = np.array([10, 11, 12, 13, 14, 15, 16,
17, 18])

# Reshape
c = np.reshape(a, (3, 3))
print(c)

# Transpose
d = np.transpose(c)
print(d)

# Concatenate
e = np.concatenate((a, b), axis=0)
print(e)
```

Array Statistics

- Numpy provides several functions for calculating statistics on arrays, such as mean, median, standard deviation, and others
- These functions can be used on both 1D and 2D arrays
- Functions like min, max, argmin, argmax, sum are also available in numpy

```
import numpy as np


a = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])
# Mean
print(np.mean(a))
# Median
print(np.median(a))
# Standard deviation
print(np.std(a))
# Min
print(np.min(a))
# Max
print(np.max(a))
```

Array Operations

Numpy supports many mathematical operations that can be performed on arrays, such as:

- Addition, subtraction, multiplication, and division
- Exponentiation and square root
- Trigonometric functions
- Mathematical operations can be performed element-wise on arrays
- Broadcasting is the numpy's term for performing an operation on two arrays with different shapes
- scalar operations can also be performed on arrays

```python
import numpy as np
a = np.array([1, 2, 3, 4, 5])
b = np.array([6, 7, 8, 9, 10])

# Addition
print(a + b)

# Subtraction
print(a - b)

# Multiplication
print(a * b)

# Division
print(a / b)

# Broadcasting
print(a + 2)
```

Pandas

Introduction

- It is a powerful and widely used open-source library for data manipulation and analysis in Python.
- It provides data structures and data manipulation functions to handle and analyze large datasets in a flexible and efficient way.
- Two main data structures:
  - **Series** : One-dimensional array-like object that can hold any data type, similar to a column in a spreadsheet.
  - **DataFrame** : Two-dimensional table of data with rows and columns, similar to a spreadsheet.
- Provides functions of data manipulation & analysis like : reading & writing data in various formats (CSV, Excel, JSON), handling missing values, filterring, grouping and aggregation of data, visualizations etc.

Pandas DataFrame

- It can be created through a list
- By default, the rows are indexed using integer indices; columns are also indexed by default using integer indices
  - Can specify column names
  - Named indexing of rows also supported
- Dataframes can also be created using key/value type of objects (like dictionaries).
  - The “key” forms the column names.

```
import pandas as pd


a = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
df1 = pd.DataFrame(a)
print(df1)


d
f2 = pd.DataFrame(a, columns=["col1", "col2", "col3"])
print(df2)
df3 = pd.DataFrame(a, index=["row1", "row2", "row3"])
print(df3)


dict1 = {
  "age": [15, 22, 36],
  "country": ["India", "Russia", "China"],
  "BMI": [19.6, 22.52, 27.1],
  "gender": "F"
}
df4 = pd.DataFrame(dict1)
print(df4)
```

Pandas Series

- It can be created through a list
- Series without labels are indexed using element indices itself.
- We can also create labels to index the values.
- Series can also be created using key/value type of objects (like dictionaries)

```
import pandas as pd
#using list
a = [1, 2, 3, 4]
s1 = pd.Series(a)
print(s1)

#without labels, values accessed using index
print(s1[0], s1[2]) #values at index 0 & 2

#creating labels
s2 = pd.Series(a, index=["a", "b", "c", "d"])
print(s2)
print(s2["a"], s2["c"])

#using dictionary
dict1 = {"x": 10, "y": 20, "z":30}
s3 = pd.Series(dict1)
print(s3)
```

Dataframe attributes

- ndim: Represents the number of dimensions of the dataframe i.e. 2.
- shape: A tuple of integers representing the size of the dataframe in each dimension
- size: Size is the total number of elements in the dataframe
- columns: Used to fetch the label values for columns present in a particular data frame.
- dtypes: Display the data type for each column of a particular dataframe.
- T: Transpose operation - change the rows into columns and columns into rows.

Other operations

- **loc :** Access a group of rows and columns by label. Can also access a particular cell.
- **head:** This function returns the first n rows. (default 5).
  tail: Return the last n rows. (default 5)
- **describe:** Returns description of data. For numerical data it provides information like count (non-empty values), mean, std dev, max, min etc.
- **drop**: To remove specific row/column.

```
import pandas as pd

dict1 = {
   "age": [15, 22, 36],
   "country": ["India", "Russia", "China"],
   "BMI": [19.6, 22.52, 27.1],
   "gender": "F"
}
df = pd.DataFrame(dict1)

print(df.loc[0]) #access row 0
print(df.loc[:, "country"]) #access a column
print(df.loc[0, "age"]) #access particular cell

print(df.head(2)) #first 2 rows
print(df.tail(2)) #last 2 rows

print(df.describe())

print(df.drop([0], axis="index")) #drop 1st row
print(df.drop(["age"], axis="columns")) #drop "age" column
```