Kakunje Software™

engineering ideas

Python Library/Packages

PYTHON INBUILT LIBRARIES

Python has a vast collection of **“inbuilt libraries”** that simplify development by providing pre-written functions for various tasks.

Here are some commonly used libraries

1. General purpose libraries
2. Mathematical libraries
3. Data handling libraries
4. Date and Time Handling libraries
5. Networking and Web Scrapping
6. Concurrency and Parallel Processing
7. File handling and Compression
8. Regular expressions

GENERAL PURPOSE LIBRARIES

os Module

The os module in Python provides functions to interact with the **operating system,** such as **file handling, directory manipulation, process management,** and **environment variables.**

Example

#Check OS type

```
import os
print(os.name)  # Get the name of the OS (e.g., 'nt' for Windows, 'posix' for Linux/Mac)
```

#Get System Environment Variables

```
import os
print(os.environ)  # Get all environment variables
print(os.environ.get("HOME"))  # Get the value of a specific variable
```

#Get Current Working Directory

```
import os
print(os.getcwd())  # Prints the current working directory
```

#Check if a file exists

```
import os
print(os.path.exists("example.txt"))  # Check if the file exists
```

#Rename a File

```
import os
os.rename("old_name.txt", "new_name.txt")  # Rename file
```

#Delete a File

```
import os
os.remove("example.txt")  # Delete file
```

sys Module

The sys module provides access to **system-specific parameters and functions**, such as **interacting with the interpreter, handling command-line arguments, modifying the Python path, and exiting programs**.

Example

#Get Python Version

```
import sys
print(sys.version)  # Full Python version details
print(sys.version_info)  # Version as a tuple (major, minor, micro)
```

#Get Platform Name

```
import sys
print(sys.platform)  # Returns OS name ('win32' for Windows, 'linux' for Linux, 'darwin' for macOS)
```

#Command-Line Arguments

Command-line arguments are stored in sys.argv, where:

- sys.argv[0] is the script name
- sys.argv[1:] are the additional arguments

```
import sys
print("Script Name:", sys.argv[0])  # Name of the script
print("Arguments:", sys.argv[1:])  # List of arguments passed
```

#Exit the Program

```
import sys
print("Before exit")
sys.exit()  # Terminates the program
print("This will not be printed")  # This line is not executed
```

#Standard Output

```
import sys
sys.stdout.write("Hello, World!\n")  # Print without using print()
```

#Input from User

```
import sys
print("Enter your name:")
name = sys.stdin.readline().strip()  # Read input from the user
print("Hello,", name)
```

shutil Module

The shutil module provides high-level **file operations**, including **copying, moving, archiving, and deleting files & directories**. It is useful for automating file management tasks.

Example

#Copy a File

```
import shutil
shutil.copy("source.txt", "destination.txt")  # Copies the file
```

#Copy a Directory

```
import shutil
shutil.copytree("source_folder", "destination_folder")  # Copies directory
```

#Move a File

```
import shutil
shutil.move("source.txt", "new_folder/source.txt")  # Moves file
```

#Remove a File

```
import os
os.remove("file.txt")  # Deletes file
```

#Remove an Empty Directory

```
import os
os.rmdir("empty_folder")  # Removes only if empty
```

#Remove a Directory with Contents

```
import shutil
shutil.rmtree("folder_to_delete")  # Deletes the entire folder
```

#Check Disk Space

```
import shutil
total, used, free = shutil.disk_usage("/")
print(f"Total: {total // (1024**3)} GB")  # Total space
print(f"Used: {used // (1024**3)} GB")    # Used space
print(f"Free: {free // (1024**3)} GB")    # Free space
```

MATHEMATICAL LIBRARIES

math Module

The math module provides **mathematical functions** such as **trigonometry, logarithms, exponentials, factorials, rounding, and constants.**

Example

#Square root

```
import math
print(math.sqrt(25))  # Output: 5.0
```

#Power Function

```
import math
print(math.pow(2, 3))  # Output: 8.0 (2^3)
```

#Absolute Value

```
import math
print(math.fabs(-10))  # Output: 10.0
```

#Round Down

```
import math
print(math.floor(4.7))  # Output: 4
```

#Round Up

```
import math
print(math.ceil(4.2))  # Output: 5
```

#Round to Nearest Integer

```
print(round(4.5))  # Output: 4
print(round(4.6))  # Output: 5
```

#Trigonometric Functions

```
import math
print(math.sin(math.radians(30)))  # Convert degrees to radians before using
print(math.cos(math.radians(60)))
print(math.tan(math.radians(45)))
```

#Natural Log

```
import math
print(math.log(10))  # Natural log (base e)
```

#Factorial
```
import math
print(math.factorial(5))  # Output: 120 (5!)
```

#Combinations
```
import math
print(math.comb(5, 2))  # Output: 10 (5 choose 2)
```

#Permutations
```
import math
print(math.perm(5, 2))  # Output: 20
```

#Constants
```
import math
print(math.pi)  # Output: 3.141592653589793
print(math.e)   # Output: 2.718281828459045
```

random Module

The random module is used to **generate random numbers**, **shuffle sequences**, and **pick random elements**. It is widely used in gaming, simulations, security, and data science.

Example

#Generate a Random Float between 0 and 1

```
import random
print(random.random())  # Output: Random float between 0.0 and 1.0
```

#Generate a Random Integer

```
import random
print(random.randint(1, 10))  # Output: Random integer between 1 and 10 (inclusive)
```

#Generate a Random Float in a Range

```
import random
print(random.uniform(5, 10))  # Output: Random float between 5 and 10
```

#Generate a Random Number Within a range

```
import random
print(random.randrange(1, 10, 2))  # Output: Random odd number (1, 3, 5, 7, 9)
```

#Pick a Random element from a List

```
import random
colors = ["Red", "Blue", "Green", "Yellow"]
print(random.choice(colors))  # Output: Random color from the list
```

#Pick Multiple Random elements from a List

```
import random
fruits = ["Apple", "Banana", "Cherry", "Mango"]
print(random.choices(fruits, k=2))  # Output: Random selection of 2 elements
```

#Shuffle a List

```
import random
cards = ["Ace", "King", "Queen", "Jack"]
random.shuffle(cards)
print(cards)  # Output: Shuffled list
```

statistics Module

The statistics module provides a set of functions for **statistical operations**. It allows you to calculate **mean, median, variance, standard deviation**, and other useful statistical metrics.

Example

#Mean: The mean (average) is the sum of all values divided by the number of values.

```
import statistics
data = [1, 2, 3, 4, 5]
print(statistics.mean(data))  # Output: 3.0 (Mean of the list)
```

#Median: The median is the middle value of a dataset when sorted.

```
import statistics
data = [1, 2, 3, 4, 5]
print(statistics.median(data))  # Output: 3.0 (Middle value)
```

#Mode: The mode is the most frequently occurring value in a dataset.

```
import statistics
data = [1, 2, 2, 3, 4]
print(statistics.mode(data))  # Output: 2 (Most frequent value)
```

#Variance: Variance measures the spread of data points from the mean.

```
import statistics
data = [1, 2, 3, 4, 5]
print(statistics.variance(data))  # Output: 2.5 (Variance)
```

#Standard Deviation: Standard deviation is the square root of variance and shows how much data deviates from the mean.

```
import statistics
data = [1, 2, 3, 4, 5]
print(statistics.stdev(data))  # Output: 1.58 (Standard deviation)
```

#Correlation: Calculates the Pearson correlation coefficient, a measure of the linear relationship between two variables.

```
import statistics
data1 = [1, 2, 3, 4, 5]
data2 = [5, 4, 3, 2, 1]
print(statistics.correlation(data1, data2))  # Output: -1.0 (perfect negative correlation)
```

DATA HANDLING LIBRARIES

json Module

The json module provides a way to **encode (serialize)** and **decode (deserialize)** JSON data. JSON (JavaScript Object Notation) is a lightweight data-interchange format that's widely used for transmitting data between a server and a client.

Example

#Convert a Python Dictionary to JSON

```
import json
data = {"name": "Alice", "age": 25, "is_student": True}
json_data = json.dumps(data)
print(json_data)  # Output: {"name": "Alice", "age": 25, "is_student": true}
```

#Convert a Python List to JSON

```
import json
data = [1, 2, 3, 4, 5]
json_data = json.dumps(data)
print(json_data)  # Output: [1, 2, 3, 4, 5]
```

#Parse JSON String into Python Dictionary

```
import json
json_data = '{"name": "Alice", "age": 25, "is_student": true}'
data = json.loads(json_data)
print(data)  # Output: {'name': 'Alice', 'age': 25, 'is_student': True}
```

#Parse JSON String into Python List

```
import json
json_data = '[1, 2, 3, 4, 5]'
data = json.loads(json_data)
print(data)  # Output: [1, 2, 3, 4, 5]
```

#Read JSON Data from a File:

```
import json
# Open and read a JSON file
with open('data.json', 'r') as file:
    data = json.load(file)
    print(data)  # Python object created from the JSON file
```

#Write JSON Data to a File

```
import json
data = {"name": "Alice", "age": 25, "is_student": True}
# Write to a JSON file
with open('data.json', 'w') as file:
    json.dump(data, file, indent=4)
```

#Pretty Print JSON String: Use the **indent** parameter in json.dumps() to pretty-print JSON with indentation.

```
import json
data = {"name": "Alice", "age": 25, "is_student": True}
json_data = json.dumps(data, indent=4)
print(json_data)
#Output
{
    "name": "Alice",
    "age": 25,
    "is_student": true
}
```

csv Module

The csv module in Python provides **functionality for reading from and writing to CSV (Comma-Separated Values) files**. CSV files are widely used for **data exchange** because they store tabular data in a simple text format, making it easy to export and import data across different programs.

Example

```
#Read a CSV File
import csv
# Open the CSV file
with open('data.csv', mode='r') as file:
    csv_reader = csv.reader(file)
    for row in csv_reader:
        print(row)  # Each row is a list
```

#Writing To CSV File

```
import csv
# Data to be written
data = [
    ['Name', 'Age', 'City'],
    ['Alice', 25, 'New York'],
    ['Bob', 30, 'Los Angeles'],
    ['Charlie', 35, 'Chicago']
]

# Open the CSV file
with open('output.csv', mode='w', newline='') as file:
    csv_writer = csv.writer(file)
    for row in data:
        csv_writer.writerow(row)  # Write each row
```

#Write Data with Header: csv.DictWriter write data to a CSV file using dictionaries where the keys are the column headers.

```
import csv
# Data to be written
data = [
    {'Name': 'Alice', 'Age': 25, 'City': 'New York'},
    {'Name': 'Bob', 'Age': 30, 'City': 'Los Angeles'},
    {'Name': 'Charlie', 'Age': 35, 'City': 'Chicago'}
]

# Open the CSV file
with open('output_dict.csv', mode='w', newline='') as file:
    fieldnames = ['Name', 'Age', 'City']
    csv_writer = csv.DictWriter(file, fieldnames=fieldnames)
    csv_writer.writeheader()  # Write header
    csv_writer.writerows(data)  # Write rows
```

DATE AND TIME HANDLING

dateTime Module

The datetime module in Python provides classes for manipulating **dates** and **times**. It allows you to work with **time zones**, **timedeltas**, and perform operations like **formatting** and **parsing** date and time strings.

Example

#Get Current Date

```
import datetime
current_date = datetime.date.today()
print(current_date)  # Output: Current date (e.g., 2025-01-31)
```

#Get Current Date and Time

```
import datetime
current_datetime = datetime.datetime.now()
print(current_datetime)  # Output: Current date and time (e.g., 2025-01-31 17:30:00)
```

#Create a Specific Date

```
import datetime
specific_date = datetime.date(2025, 12, 25)
print(specific_date)  # Output: 2025-12-25
```

#Get Year, Month, Day from a Date

```
import datetime
current_date = datetime.date.today()
print(current_date.year)  # Get year
print(current_date.month)  # Get month
print(current_date.day)  # Get day
```

#Create a Specific Time

```
import datetime
specific_time = datetime.time(14, 30, 0)  # 14:30:00
print(specific_time)  # Output: 14:30:00
```

#Get Hour, Minute, Second from a Time

```
import datetime
current_time = datetime.datetime.now().time()
print(current_time.hour)   # Get hour
print(current_time.minute) # Get minute
print(current_time.second) # Get second
```

#Format Date/Time to String: You can convert a datetime object to a formatted string using **strftime()**.

- %Y – Year(4 digits)
- %m – Month (2 digits)
- %d – Day of the month (2 digits)
- %H – Hour (24-hour format)
- %M – Minue
- %S – Second

```
import datetime
current_datetime = datetime.datetime.now()
formatted_date = current_datetime.strftime("%Y-%m-%d %H:%M:%S")
print(formatted_date)  # Output: 2025-01-31 17:30:00 (current time)
```

time Module

The time module provides functions to work with **timestamps, delays, and performance measurement**. It is useful for **measuring execution time**, **sleeping a program**, and working with **epoch timestamps**.

Example

#Get Current Timestamp

```
import time
timestamp = time.time()
print(timestamp)  # Output: 1706775000.123456 (Current timestamp)
```

#Get Readable Current time

```
import time
print(time.ctime())  # Output: Fri Jan 31 17:30:00 2025
```

#Pause Execution

```
import time
print("Start")
time.sleep(3)  # Sleep for 3 seconds
print("End")
```

#Get Local Time

```
import time
local_time = time.localtime()
print(local_time)  # Output: time.struct_time(tm_year=2025, tm_mon=1, tm_mday=31,
tm_hour=17, tm_min=30, tm_sec=0, ...)
```

#Convert Time to String

```
import time
formatted_time = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
print(formatted_time)  # Output: 2025-01-31 17:30:00
```

Networking and Web Scraping

urllib Module

The urlibmodule is a **built-in** Python library for handling **URLs, HTTP requests, and web scraping**. It is useful for **fetching web pages, downloading files, and handling query parameters**.

Example

```
#Open a URL
import urllib.request
response = urllib.request.urlopen("https://www.example.com")
print(response.status)  # Output: 200 (HTTP Status Code)
print(response.read().decode("utf-8"))  # Output: Page content
```

#GET Request with Query Parameters

```
import urllib.request
import urllib.parse

params = urllib.parse.urlencode({"search": "Python", "page": 1})
url = f"https://www.example.com/search?{params}"

response = urllib.request.urlopen(url)
print(response.read().decode("utf-8"))  # Fetch search results
```

#POST Request

```
import urllib.request
import urllib.parse

url = "https://httpbin.org/post"
data = urllib.parse.urlencode({"username": "user", "password": "pass"}).encode()
req = urllib.request.Request(url, data=data, method="POST")

response = urllib.request.urlopen(req)
print(response.read().decode("utf-8"))
```

#Parse a URL

```
import urllib.parse
url = "https://www.example.com/search?q=python&page=2"
parsed_url = urllib.parse.urlparse(url)

print(parsed_url.scheme)  # Output: https
print(parsed_url.netloc)  # Output: www.example.com
print(parsed_url.path)    # Output: /search
print(parsed_url.query)   # Output: q=python&page=2
```

#Build a URL

```
import urllib.parse
url_parts = ("https", "www.example.com", "/search", "", "q=python&page=2", "")
full_url = urllib.parse.urlunparse(url_parts)
print(full_url)  # Output: https://www.example.com/search?q=python&page=2
```

#Handle HTTP Errors

```
import urllib.request
import urllib.error

try:
    response = urllib.request.urlopen("https://invalid-url.com")
except urllib.error.HTTPError as e:
    print("HTTP Error:", e.code, e.reason)
except urllib.error.URLError as e:
    print("URL Error:", e.reason)
```

#Download a File from a URL

```
import urllib.request
urllib.request.urlretrieve("https://www.example.com/image.jpg", "image.jpg")
print("Download complete!")
```

Socket Module

The socket module provides a low-level interface for network communication. It is used to implement both **client-server communication** and **network-based applications** (e.g., HTTP servers, chat applications). Socket supports **TCP/IP** and **UDP** protocols, allowing Python programs to communicate over the network.

Example

```
#Creating a TCP Socket
import socket

# Create a TCP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Define server address and port
host = "localhost"
port = 12345

# Connect to the server
sock.connect((host, port))
```

#Creating a UDP Socket

```
import socket

# Create a UDP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Define server address and port
host = "localhost"
port = 12345

# Send data to the server
message = b"Hello, Server!"
sock.sendto(message, (host, port))
```

#TCP Server Example (Listening for Connections)

```
import socket
# Create a TCP socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Define server address and port
host = "localhost"
port = 12345
# Bind the socket to the address and port
server_socket.bind((host, port))
# Listen for incoming connections (max 1 connection in the queue)
server_socket.listen(1)
print(f"Server listening on {host}:{port}...")
# Accept a connection
client_socket, client_address = server_socket.accept()
print(f"Connection from {client_address}")
# Receive data from the client
data = client_socket.recv(1024)
print(f"Received from client: {data.decode()}")
# Send a response to the client
client_socket.send(b"Hello, Client!")
# Close the connection
client_socket.close()
```

#UDP Server Example (Listening for Messages)

```
import socket

# Create a UDP socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Define server address and port
host = "localhost"
port = 12345

# Bind the socket to the address and port
server_socket.bind((host, port))

print(f"Server listening on {host}:{port}...")

# Listen for incoming messages
message, client_address = server_socket.recvfrom(1024)
print(f"Received message from {client_address}: {message.decode()}")

# Send a response to the client
server_socket.sendto(b"Hello, Client!", client_address)
```

#TCP Client Example (Sending Data to Server)

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

# Define server address and port
host = "localhost"
port = 12345

# Connect to the server
client_socket.connect((host, port))

# Send data to the server
client_socket.send(b"Hello, Server!")

# Receive response from the server
response = client_socket.recv(1024)
print(f"Received from server: {response.decode()}")

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

#UDP Client Example (Sending Data to Server)

```
import socket

# Create a UDP socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Define server address and port
host = "localhost"
port = 12345

# Send data to the server
message = b"Hello, Server!"
client_socket.sendto(message, (host, port))

# Receive response from the server
response, server_address = client_socket.recvfrom(1024)
print(f"Received from server: {response.decode()}")

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

#Closing the Socket

```
# Close the socket after use
sock.close()
```

#Error Handling

```
try:
    # Create a socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # Connect to the server
    sock.connect(('localhost', 12345))
    # Send and receive data
    sock.send(b'Hello, Server!')
    response = sock.recv(1024)
    print(response.decode())

except socket.error as e:
    print(f"Socket error: {e}")

finally:
    sock.close()  # Ensure the socket is closed
```

FILE HANDLING AND COMPRESSION

zipfile Module

The zipfile module in Python provides functions to **create, read, write, and extract ZIP archives**. It is useful for **compressing files, extracting archives, and managing large datasets**.

Example

#Creating and ZIP File and Add Files

```
import zipfile

with zipfile.ZipFile("my_archive.zip", "w") as zipf:
    zipf.write("file1.txt")  # Add a file
    zipf.write("file2.txt")  # Add another file
```

#Add Files with Compression

```
import zipfile

with zipfile.ZipFile("compressed.zip", "w", compression=zipfile.ZIP_DEFLATED) as zipf:
    zipf.write("file1.txt")
    zipf.write("file2.txt")
```

#Extract All Files

```
import zipfile
with zipfile.ZipFile("my_archive.zip", "r") as zipf:
    zipf.extractall("output_folder")  # Extract all files to a folder
```

#Extract a specific File

```
import zipfile
with zipfile.ZipFile("my_archive.zip", "r") as zipf:
    zipf.extract("file1.txt", "output_folder")  # Extract only 'file1.txt’
```

#List Files Inside a ZIP

```
import zipfile
with zipfile.ZipFile("my_archive.zip", "r") as zipf:
    print(zipf.namelist())  # Output: ['file1.txt', 'file2.txt’]
```

#Get File Details

```
import zipfile
with zipfile.ZipFile("my_archive.zip", "r") as zipf:
    info = zipf.getinfo("file1.txt")
    print(info.filename)  # Output: file1.txt
    print(info.file_size)  # Output: File size in bytes
```

tarfile Module

The tarfile module allows you **create, read, write, and extract TAR archieves** (.tar, .tar.gz, .tar.bz2, .tar.xz). It is useful for **compressing large datasets, backups, and file distribution.**

Example

#Create a TAR File

```
import tarfile
with tarfile.open("my_archive.tar", "w") as tar:
    tar.add("file1.txt")  # Add a file
    tar.add("file2.txt")  # Add another file
```

#Create a Compressed TAR File

```
import tarfile
with tarfile.open("compressed.tar.gz", "w:gz") as tar:
    tar.add("file1.txt")
    tar.add("file2.txt")
```

#Extract All Files

```
import tarfile
with tarfile.open("my_archive.tar", "r") as tar:
    tar.extractall("output_folder")  # Extract all files to a folder
```

#Extract a Specific File

```
import tarfile
with tarfile.open("my_archive.tar", "r") as tar:
    tar.extract("file1.txt", "output_folder")  # Extract only 'file1.txt’
```

#List File inside a TAR

```
import tarfile
with tarfile.open("my_archive.tar", "r") as tar:
    print(tar.getnames())  # Output: ['file1.txt', 'file2.txt’]
```

#Get File Details

```
import tarfile
with tarfile.open("my_archive.tar", "r") as tar:
    info = tar.getmember("file1.txt")
    print(info.name)  # Output: file1.txt
    print(info.size)  # Output: File size in bytes
```

#Open and Extract a .tar.gz File

```
import tarfile
with tarfile.open("compressed.tar.gz", "r:gz") as tar:
    tar.extractall("output_folder")
```

#Create a .tar.gz Archieve

```
import tarfile
with tarfile.open("new_archive.tar.gz", "w:gz") as tar:
    tar.add("file1.txt")
    tar.add("file2.txt")
```

CONCURRENCY AND PARALLEL PROCESSING

threading Module

The threading module provides a way to run multiple tasks **concurrently** in the same program using **threads**. It is useful for **parallel execution, handling background tasks, and improving performance** in I/O-bound operations.

Example

```
#Creating a Thread
import threading

def print_numbers():
    for i in range(5):
        print(f"Number: {i}")

t = threading.Thread(target=print_numbers) # Create a thread
t.start()  # Start the thread
t.join()   # Wait for the thread to complete
print("Main thread finished!")
```

#Running Multiple Threads

```
import threading

def print_numbers():
    for i in range(3):
        print(f"Number: {i} from {threading.current_thread().name}")

def print_letters():
    for char in "ABC":
        print(f"Letter: {char} from {threading.current_thread().name}")

# Create multiple threads
t1 = threading.Thread(target=print_numbers, name="Thread-1")
t2 = threading.Thread(target=print_letters, name="Thread-2")
# Start threads
t1.start()
t2.start()
# Wait for both threads to finish
t1.join()
t2.join()
print("Main thread finished!")
```

#Pass Arguments To Threads

```
import threading
def greet(name):
    print(f"Hello, {name}!")
# Create a thread with arguments
t = threading.Thread(target=greet, args=("Alice",))
t.start()
t.join()
```

#Daemon Threads (Background Tasks) : Daemon threads run **in the background** and automatically stop when the main program exits.

```
import threading
import time
def background_task():
    while True:
        print("Running background task...")
        time.sleep(2)
# Create a daemon thread
t = threading.Thread(target=background_task, daemon=True)
t.start()
time.sleep(5)
print("Main thread exiting...")  # The daemon thread stops when the main thread exits
```

#Thread Locking: When multiple threads **access shared resources**, data corruption may occur. Use **locks** to avoid conflicts.

```
import threading

counter = 0
lock = threading.Lock()

def increment():
    global counter
    for _ in range(1000000):
        with lock:  # Ensures only one thread modifies `counter` at a time
            counter += 1

t1 = threading.Thread(target=increment)
t2 = threading.Thread(target=increment)
t1.start()
t2.start()
t1.join()
t2.join()
print("Final Counter Value:", counter)  # Expected output: 2000000
```

multiprocessing Module

The multiprocessing nodule allows Python programs to run **multiple processes in parallel**, utilizing **multiple CPU cores** for better performance. It is useful for **CPU-bound tasks**, such as **data processing, computations, and parallel execution**.

Example

```
#Creating Multiple Processes
import multiprocessing

def print_numbers():
    for i in range(5):
        print(f"Number: {i}")

p = multiprocessing.Process(target=print_numbers)    # Create a process
p.start()  # Start the process
p.join() # Wait for the process to complete
print("Main process finished!")
```

#Running Multiple Processes

```
import multiprocessing

def task(name):
    print(f"Task {name} is running")

# Create multiple processes
processes = []
for i in range(3):
    p = multiprocessing.Process(target=task, args=(i,))
    processes.append(p)
    p.start()

# Wait for all processes to finish
for p in processes:
    p.join()

print("All tasks completed!")
```

#Using Pool for Parallel Processing: Instead of manually creating processes, **Pool** can automatically manage multiple processes.

```
import multiprocessing

def square(n):
    return n * n

if __name__ == "__main__":  # Required for Windows
    with multiprocessing.Pool(processes=4) as pool:  # Create a pool of 4 processes
        results = pool.map(square, [1, 2, 3, 4, 5])
    print(results)  # Output: [1, 4, 9, 16, 25]
```

#Using Queues for Inter-Process Communication

```
import multiprocessing

def worker(q):
    q.put("Hello from process!")

if __name__ == "__main__":
    q = multiprocessing.Queue()
    p = multiprocessing.Process(target=worker, args=(q,))
    p.start()
    p.join()

    print(q.get())  # Output: "Hello from process!"
```

REGULAR EXPRESSIONS

re Module

The re module provides support for **regular expressions (regex)**, allowing you to search, match, and manipulate text efficiently.

Example

#Searching for patterns

```
import re
text = "Hello, world!"
match = re.match(r"Hello", text)  # Check if 'Hello' is at the start
print(match)  # Output: <re.Match object> (if found) or None
```

#Find a pattern anywhere in a String

```
import re
text = "Hello, world!"
search = re.search(r"world", text)
print(search)  # Output: <re.Match object> (if found) or None
```

#Replace Text

```
import re
text = "I love Python! Python is awesome!"
new_text = re.sub(r"Python", "Java", text)  # Replace 'Python' with 'Java'
print(new_text)  # Output: I love Java! Java is awesome!
```

#Split a String

```
import re
text = "apple, banana; mango|grape"
words = re.split(r",|;|\|", text)  # Split at comma, semicolon, or pipe
print(words)  # Output: ['apple', 'banana', 'mango', 'grape']
```

PYTHON EXTERNAL LIBRARIES

Python has several **“external libraries”** that extend its functionality for various domains. These libraries are not built into Python and must be installed using **pip**.

Here are some external libraries:

1. Data science and Analysis
2. Data Visualization
3. Web Development
4. Web scrapping
5. Automation and scripting
6. Networking
7. Game development

PIP INSTALLATION: pip is the package manager for Python that allows you to install and manage libraries like NumPy, Pandas, Matplotlib, etc.
Command: python -m ensurepip --default-pip
Check version using command: pip –version

PYTHON LIBRARY/PACKAGES INSTALLATION USING GITHUB

- Open a terminal or command prompt add Python IDLE path.
- To clone the official Python repository or a specific fork using command
  git clone https://github.com/python/cpython.git
- Open the cloned folder in visual studio.
- Open the PCbuild directory.
- Build the solution file (Pcbuild.sln) in Release mode.
- Verify installation using python3.x –version
  Replace python3.x with the version you installed e.g.,python3.13

DATA SCIENCE AND ANALYSIS

Numpy Module

Numpy is the core library for scientific and numerical computing in Python. It provides high performance multi-dimensional array object and tools for working with arrays.

NumPy Installation

pip install numpy

Import NumPy

Once NumPy is installed, import it in your applications by adding the import keyword.

Syntax:

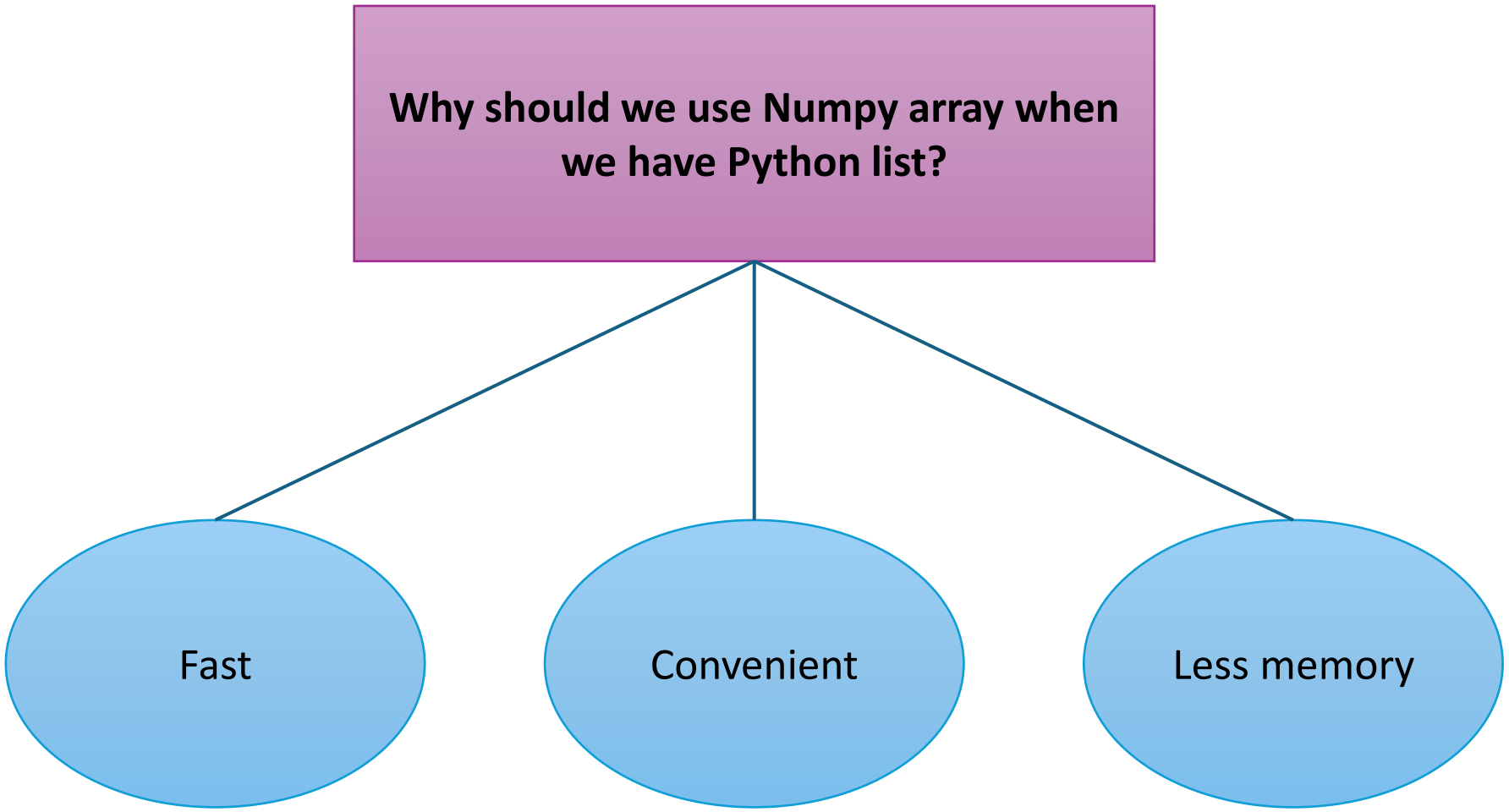
```
import numpy
```

NumPy as np

NumPy is usually imported under the np alias.

Syntax:

```
import numpy as np
```


Why should we use Numpy array when we have Python list?
Fast
Convenient
Less memory

Creating Arrays

1) **Using Lists or Tuples:** We can create a NumPy array from a Python list or tuple using the numpy.array() function.

Example 1:

```
import numpy as np
# Creating a 1Dimension array from a list
arr1d = np.array([1, 2, 3, 4, 5]) print(arr1d)
print(arr1d)                          #[1 2 3 4 5]
```

Example 2:

```
import numpy as np
# Creating a 2Dimension array from a list of lists
arr2d = np.array([[1, 2, 3], [4, 5, 6]])
print(arr2d)            #[ [1 2 3]
                        [4 5 6] ]
```

Example 3:

```
import numpy as np
#Creating array from a tuple
arr=np.array((1,2,3,4,5))
print(arr)            #[1 2 3 4 5]
```

Using NumPy Functions:

NumPy provides functions to create arrays of specific shapes and values.

np.zeros():The np.zeros() function in NumPy is used to create an array filled with zeros.

Example:

```
import numpy as np
# Creating a 2x3 array filled with zeros
zeros_arr = np.zeros((2, 3))  #2 rows and 3 columns
print(zeros_arr)
```

Output:

```
[[0. 0. 0.]
 [0. 0. 0.]]
```

np.ones():The np.ones() function in NumPy is used to create an array filled with ones.

Example:

```
import numpy as np
# Creating a 3x2 array filled with ones
ones_arr = np.ones((3, 2)) #3 rows and 2 columns
print(ones_arr)
```

Output:

```
[[1. 1.]
[1. 1.]
[1. 1.]]
```

np.arange(): The np.arange()function in NumPy is used to create an array containing evenly spaced values within a specified range.

Example:

```
import numpy as np
# Creating an array containing values from 3 to 9 with a step size of 2
arr = np.arange(3, 10, 2)
print(arr)
```

Output:

```
[3 5 7 9]
```

np.random.rand(): The np.random.rand() function in NumPy is used to generate an array of specified shape filled with random numbers.

Example:

```
import numpy as np
rand_arr = np.random.rand(2, 3)
print(rand_arr)
```

Output:

```
[ [0.29865018    0.73081801      0.73348864]
              [0.57138207     0.85962445      0.12015773] ]
```

Access Array Elements(Array Indexing)

Access 1-D arrays:

An array element can be accessed by referring to its index number.

The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.

Example:

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr[1])                          #2
```

Access 2-D arrays:

To access elements from 2-D arrays we can use comma separated integers representing the dimension and the index of the element.

Example:

```
import numpy as np
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print(arr)
print(arr[0, 1]) #print 2nd element from 1st row
```

Output:

```
[[ 1 2 3 4 5]
[ 6 7 8 9 10]]
2
```

Access 3-D arrays:

To access elements from 3-D arrays we can use comma separated integers representing the dimensions and the index of the element.

Example:

```
# Access the third element of the second array of the first array
import numpy as np
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print(arr)
print('Third element of the second array of the first array is', arr[0, 1, 2])
```

Output:

```
[[[ 1  2  3]
  [ 4  5  6]]
[[ 7  8  9]
  [10 11 12]]]
Third element of the second array of the first array is 6
```

Array Slicing

Example 1

```
#Slice elements from index 1 to index 5
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5])                                    #[2 3 4 5]
```

Example 2

```
# Slice from the index 3 from the end to index 1 from the end
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[-3:-1])                          #[5 6]
```

Example 3

```
#Return the odd numbers from the array
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[::2])        #Output:[1 3 5 7]
```

Example 4

```
# From both elements, return index 2
import numpy as np
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[0:2, 2])        #Output:[3 8]
```

NumPy Data Types

Data types in NumPy and the characters used to represent them are:
i - integer
b - Boolean
u - unsigned integer
f - float
c - complex float
m - time delta
M - datetime
O - object
S - string
U - Unicode string
V - fixed chunk of memory for other type (void)

Checking the Data Type of an Array

The NumPy array object has a property called dtype that returns the data type of the array.

Example:

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr.dtype)
```

Output:

int64

Creating Arrays with a Defined Data Type

It allows you to specify the exact type of elements that the array will contain.

Example:

```
#Create an array with data type string
import numpy as np
arr = np.array([1, 2, 3, 4], dtype='S')
print(arr)          # [b'1' b'2' b'3' b'4'] # b refers to the byte string
```

Array Shape:

The shape of an array refers to its dimensions, specified as a tuple of integers representing the size of each dimension.

Example:

```
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6]])
print(arr.shape)                                    #(2, 3)
```

Array Reshaping

Reshaping means changing the shape of an array. The shape of an array is the number of elements in each dimension.

By reshaping we can add or remove dimensions or change number of elements in each dimension.

Example 1

```
#Converting from 1-D to 2-D with 4 arrays each with 3 elements
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
new_arr = arr.reshape(4, 3)
print(new_arr)
```

Output:

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```

Example 2

#Converting from 1-D to 3-D with 2 arrays that contains 3 arrays, each with 2 elements

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
new_arr = arr.reshape(2, 3, 2)
print(new_arr)
```

Output:

```
[[[ 1 2]
  [ 3 4]
  [ 5 6]]
[[ 7 8]
  [ 9 10]
  [ 11 12]]]
```

Example 3

#Converting from 1-D with 8 elements to a 2-D array with 3 elements in each dimension

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
new_arr = arr.reshape(3, 3)
print(new_arr)
```

Output:

Value Error: cannot reshape array of size 8 into shape (3,3)

Joining NumPy Arrays

Joining means putting contents of two or more arrays in a single array. In NumPy, you can join or concatenate arrays along different axes.

numpy.concatenate(): This function concatenates arrays along a specified axis. It takes a sequence of arrays and the axis along which the arrays will be joined.

Example 1:

```
import numpy as np
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5, 6]])
result = np.concatenate((arr1, arr2), axis=0)  # Concatenate along the rows
print(result)
```

Output:

```
[[1 2]
 [3 4]
 [5 6]]
```

Example 2

```
import numpy as np
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5, 6]])
result = np.concatenate((arr1, arr2))
print(result)
```

Output:

```
[[1 2]
 [3 4]
 [5 6]
 [7 8]]
```

Example 3

```
import numpy as np
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5], [6]])  # Reshaping arr2 to have 2 rows
result = np.concatenate((arr1, arr2), axis=1)  # Concatenate along the columns
print(result)
```

Output:

```
[[1 2 5]
 [3 4 6]]
```

Splitting NumPy Arrays

Splitting is reverse operation of Joining. We can split arrays along different axes using various functions. The most common functions for splitting arrays are **numpy.split(), numpy.vsplit(),** and **numpy.hsplit()**.

Numpy.split():

This functions splits an array into multiple sub-arrays along a specified axis. It takes the array to be split, the number of sections to split it into, and the axis along which to split.

Example

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
result = np.split(arr, 2)       #split arr into 2 equal sized sub-arrays
print(result)
```

Output

```
[array([1, 2, 3, 4]), array([5, 6, 7, 8])]
```

Example 2

```
#Split the 2-D array into three 2-D arrays along rows axis =1
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
result = np.split(arr, 3, axis = 1)
print(result)
```

Output

```
[array([[1],
        [4]]), array([[2],
        [5]]), array([[3],
        [6]])]
```

Numpy.vsplit():

This function splits an array vertically(row-wise) into multiple sub-arrays. It takes the array to be split and the number of rows along which to split.

Example

```
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])
result=np.vsplit(arr, 2)#split arr into 2 equal sized sub-arrays along rows
print(result)
```

Output

```
[array([[1, 2, 3],
        [4, 5, 6]]), array([[7, 8, 9],
        [10, 11, 12]])]
```

Numpy.hsplit():

This function splits an array horizontally(column-wise) into multiple sub-arrays. It takes the array to be split and the number of columns along which to split.

Example

```
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])
result=np.hsplit(arr, 2)#split arr into 2 equal sized sub-arrays along columns
print(result)
```

Output

```
[array([[1, 2],
        [5, 6],
        [9, 10]]), array([[3, 4],
        [7, 8],
        [11, 12]])]
```

Searching Arrays

where(): Searching arrays in NumPy involves in finding elements that meet certain conditions or criteria within an array. To search an array, use the **where()** method.

Example

```
#Find the indexes where the value is 4
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 4, 4])
x = np.where(arr == 4)
print(x)
```

Output

```
(array([3, 5, 6], dtype = int64),) #return a tuple
```

Example

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
x = np.where(arr % 2 == 0)
print(x)
```

Output

```
(array([1, 3, 5, 7]),)
```

Sorting Arrays

Sorting means putting elements in an ordered sequence. Ordered sequence is any sequence that has an order corresponding to elements, like numeric or alphabetical, ascending or descending.

Example

```
import numpy as np
arr = np.array([3, 2, 0, 1])
print(np.sort(arr))
```

Output

```
[0 1 2 3]
```

Example 2

```
#sorting a 2-D array
import numpy as np
arr = np.array([[3, 2, 4], [5, 0, 1]])
print(np.sort(arr))
```

Output

```
[[2 3 4]
 [0 1 5]]
```

Pandas Module

Pandas is a tool for data processing which helps in data analysis. It provides functions and methods to efficiently manipulate large datasets.

PANDAS INSTALLATION

pip install pandas

Import Pandas as pd

Pandas is usually imported under the pd alias.

Syntax: import pandas as pd

Series: A one-dimensional labeled array capable of holding any data type (e.g., integers, strings, floating-point numbers, Python objects). It's similar to a list or a one-dimensional array but with additional features like labeled indexing.

DataFrame: A two-dimensional labeled data structure with columns of potentially different types. It's similar to a spreadsheet or SQL table, and it consists of rows and columns. Each column in a DataFrame is a Series.

Pandas Series

A Pandas Series is like a column in a table. It is a one-dimensional array holding data of any type.

Example 1:

```
import pandas as pd
a = [1, 7, 2]
myvar = pd.Series(a)
print(myvar)
```

Output:

```
0    1
1    7
2    2
dtype: int64
```

Example 2:

```
#To return the first value of the series
import pandas as pd
a = [1, 7, 2]
myvar = pd.Series(a)
print(myvar[0])                #Output: 1
```

Create Labels

With the index argument, you can name your own labels.

Example 1

```
import pandas as pd
a = [1, 7, 2]
myvar = pd.Series(a, index = [“x”, “y”, “z”])
print(myvar)
```

Output

```
x  1
y  7
z  2
dtype: int64
```

Example 2

```
#access an item by referring to the label
import pandas as pd
a = [1, 7, 2]
myvar = pd.Series(a, index = [“x”, “y”, “z”])
print(myvar[“y”])              #7
```

Example 3

```
#To select only some of the items in the dictionary
 import pandas as pd
calories = {“day1”: 365, “day2”: 342, “day3”: 387}
myvar = pd.Series(calories, index = [“day1”, “day2”])
print(myvar)
```

Output

```
day1                365
day2                380
dtype:      int64
```

Pandas DataFrames

A Pandas DataFrame is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns.

Example

```
import pandas as pd
data = {“calories”: [345, 380, 390],
                “duration”: [50, 40, 45]}
df = pd.DataFrame(data)
print(df)
```

Output

```
        calories  duration
0        345                50
1        380                40
2        390                45
```

Locate Row

Pandas use the **loc** attribute to return one or more specified row(s).

Example 1

```
import pandas as pd
data = {"calories": [365, 380, 390],"duration": [50, 40, 45] }
#load data into a DataFrame object:
df = pd.DataFrame(data)
print(df.loc[0])
```

Output

```
calories  365
duration 50
Name: 0, dtype: int64
```

Example:

```
import pandas as pd
data = {"calories": [365, 380, 390],"duration": [50, 40, 45] }
#load data into a DataFrame object:
df = pd.DataFrame(data)
print(df.loc[[0, 1]])
```

Output:

```
     calories  duration
0       365  50
1       380       40
```

Load Files Into a DataFrame

If the data sets are stored in a file, Pandas can load them into a DataFrame.

Example

```
import pandas as pd
df = pd.read_csv('data.csv')
print(df)
```

Information About the Data

The DataFrame object has a method called **info()**, which gives the information about the data set. It also tells how many Non-Null values are present in each column.

Syntax

```
print(df.info())
```

Viewing the Data

head(): The head() method returns the headers and a specified number of rows, starting from the top.

Example

```
#print the first 10 rows of the DataFrame
import pandas as pd
df = pd.read_csv('data.csv')
print(df.head(10))
```

tail(): The tail() method returns the headers and a specified number of rows, starting from the bottom.

Example

```
#print the last 10 rows of the DataFrame
print(df.tail(10))
```

sample(): The sample() function is used to randomly sample data from a DataFrame or Series.

Example

```
#print the 10 sample data
print(df.sample(10))
```

describe(): The describe() method in pandas is used to generate descriptive statistics of a DataFrame or Series.

Example

```
print(df.describe())
```

columns: You can get all the column names present in your dataset.

Example

```
print(df.columns)
```

dtypes: Gives the data type of each columns.

Example

print(df.dtypes)

shape: Returns the number of rows and columns in a DataFrame.

Example

print(df.shape)

df[‘column_name’] OR df.column_name: Returns the column whichever we want

Example

print(df[‘age’]) / print(df.age)

df[[‘column_1’, ‘column_2’, ‘column_n’]]: Returns the multiple column whichever we require.

Example

print(df[[‘age’, ‘name’, ‘Id’, ‘address’]]

df[‘column_name’].unique(): It displays the unique values of that specific column of a dataframe.

Example

print(df[‘gender’].unique())

Df[‘column_name’].value_counts(): It is used to count the unique values in a specific column of a DataFrame.

Example

Print(df[‘gender’].value_counts())

rename: It is used to rename columns in a pandas DataFrame.

Example

```
#renaming the columns
df.rename(columns = {'bmi': 'BMI'}, inplace = True)
```

isna(): It is used to detect the missing values or null values in a DataFrame or Series.

Example

```
#find missing values from the data set.
null = df.isna().sum()
print("Number of null values in the dataset are:", null)
```

Data Cleaning:

Data cleaning in Pandas means fixing the bad data in your data set.

Bad data could be:

- Empty cells
- Data in wrong format
- Wrong data
- Duplicates

Empty Cells:

Empty cells can potentially give you a wrong result when you analyse data.

Remove Rows: One way to deal with empty cell is to remove rows that contain empty cells. This is usually Ok, if data sets are very big and removing a few rows will not have a big impact on the result. **dropna()** method is used.

Example

```
import pandas as pd
df = pd.read_csv('data.csv')
new_df = df.dropna()
print(new_df)
```

Replace Empty Values in the entire DataFrame: Another way of dealing with empty cells is to insert a new value instead. The **fillna()** method allows us to replace empty cells with a value.

Example

```
#Replace the null value with the value 100

import pandas as pd

df = pd.read_csv('data.csv')

df.fillna(100, inplace = True)

print(df)
```

Replace Only For Specified Columns: To only replace empty values for one column, specify the column name for the DataFrame.

Example

```
#Replace null values in the "abc" column with the value 100
import pandas as pd
df = pd.read_csv('data.csv')
df["abd"].fillna(100, inplace = True)
print(df)
```

bfill()/ffill(): Replace null values using bfill or ffill
Backfill: The bfill method fills missing values using the next valid observation.
Forwardfill: The ffill method fills missing values using the previous valid observation.

Example

```
#fill missing values using back fill
df_bfill = df.bfill()
#fill missing values using the forward fill
df_ffill = df.ffill()
```

Replace Using Mean, Median, or Mode: A common way to replace empty cells, is to calculate the mean, median, or mode value of the column. Pandas uses the mean(), median() and mode() methods to calculate the respective values for a specified column.

Example

```
#calculate he MEAN and replace any empty values with it (Datatype: Integer or Float)
import pandas as pd
df = pd.read_csv('data.csv')
x = df["abc"].mean()
df["abc"].fillna(x, inplace = True)
print(df)
```

Example 2

```
#Calculate the MEDIAN and replace any empty values with it. (Datatype: Integer or Float)
import pandas as pd
df = pd.read_csv('data.csv')
x = df["abc"].median()
df["abc"].fillna(x, inplace = True)
print(df)
```

Example 3

```
#Calculate the MODE and replace any empty values with it. (Datatype: Integer, Float, String, Character)
import pandas as pd
df = pd.read_csv('data.csv')
x = df["abc"].mode()[0]      #Gets the first mode value
df["abc"].fillna(x, inplace =True)
print(df)
```

Removing Rows: Another way to handling wrong data is to remove the rows that contains wrong data.

Example

```
#Delete rows where “Duration” is higher than 120
import pandas as pd
df = pd.read_csv(‘data.csv’)
for x in df.index:
        if df.loc[x, “Duration”] > 120:
                df.drop(x, inplace = True)
print(df)
```

Duplicates:

Discovering Duplicates: Duplicate rows are rows that have been registered more than one time. To discover duplicates, we can use the **duplicated()** method. The duplicated() method returns a Boolean values for each row.

Example

```
import pandas as pd
df = pd.read_csv(‘data.csv’)
print(df.duplicated())
```

Output

```
# Returns True for every row that is a duplicate, otherwise False.
```

Removing Duplicates: To remove duplicates, the **drop_duplicates()** method is used.

Example

```
import pandas as pd
df = pd.read_csv('data.csv')
df.drop_duplicates(inplace = True)
print(df.to_string())
```

Save the File:

```
#To save the file
df.to_csv('saved.csv')
```

SciPy Module

SciPy (Scientific Python) is a powerful library built on **NumPy**, providing advanced functions for **scientific computing, optimization, integration, statistics, signal processing, and linear algebra**. It is widely used in **machine learning, engineering, physics, and data science**.

INSTALLATION

pip install scipy

#Solving Linear Equations

```
import numpy as np
from scipy.linalg import solve

A = np.array([[3, 1], [1, 2]])
b = np.array([9, 8])
x = solve(A, b)
print(x)  # Output: Solution to Ax = b
```

#Find minimum of a Function

```
from scipy.optimize import minimize

# Define a function to minimize
def f(x):
    return x**2 + 3*x + 2

result = minimize(f, x0=0)  # Start search at x0=0
print(result.x)  # Output: Minimum value of x
```

#Calculate Mean, Median, Mode

```
from scipy import stats

data = [1, 2, 3, 4, 5, 5, 6, 7, 8, 9]
print(stats.mode(data))  # Output: Mode (most frequent value)
print(stats.describe(data))  # Output: Summary statistics (mean, variance, etc.)
```

#Compute Definite Integral

```
from scipy.integrate import quad

# Function to integrate
def f(x):
    return x**2

integral, error = quad(f, 0, 2)  # Compute ∫ x^2 dx from 0 to 2
print(integral)  # Output: 2.666...
```

#Solve Differential Equations

```
import numpy as np
from scipy.integrate import odeint

# Define dy/dt = -y
def model(y, t):
    return -y

y0 = 5  # Initial condition
t = np.linspace(0, 10, 100)  # Time points
solution = odeint(model, y0, t)
print(solution)  # Output: Values of y over time
```

#Interpolate Missing Values

```
import numpy as np
from scipy.interpolate import interp1d

x = np.array([1, 2, 4, 5])
y = np.array([10, 20, 40, 50])

interpolator = interp1d(x, y, kind="linear")
print(interpolator(3))  # Output: Interpolated value at x=3
```

DATA VISUALIZATION

Matplotlib Module

- It is an open source drawing library which supports rich drawing types.
- It is used to draw 2D and 3D graphics.
- You can understand your data easily by visualizing it with the help of matplotlib.
- You can generate plots, histograms, bar charts and many other charts with just a few lines of code.

MATPLOTLIB INSTALLATION

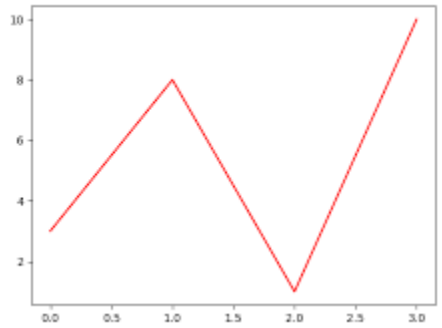
pip install matplotlib

Import Matplotlib

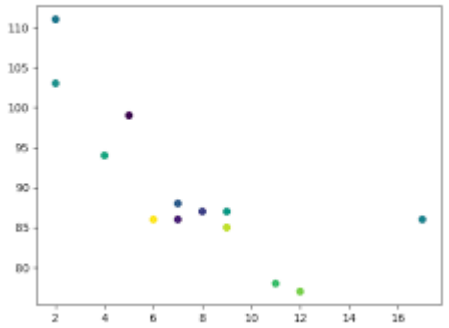
Syntax:

import matplotlib.pyplot as plt

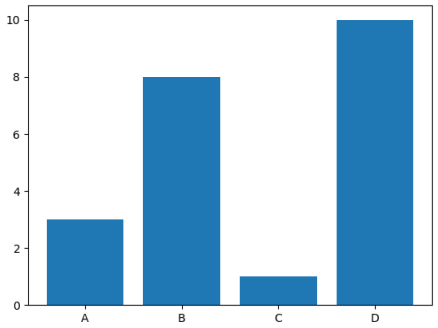
Types of Plots



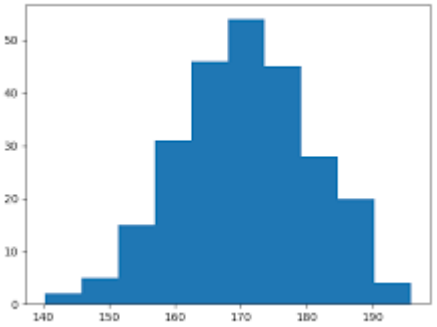
Line Plot



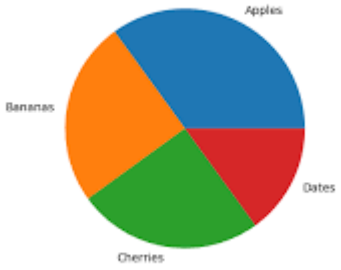
Scatter Plot



Bar Chart



Histogram



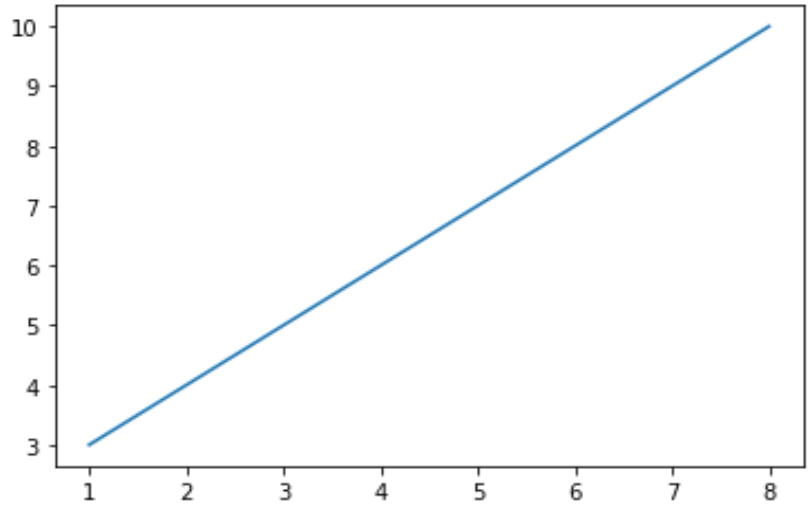
Pie Chart

Matplotlib Plotting

Plotting x and y points: The **plot()** function is used to draw points (markers) in a diagram. By default, the plot() function draws a line from point to point. The function takes parameters for specifying points in the diagram. Parameter 1 is an array containing the points on the **x-axis**. Parameter 2 is an array containing the points on the **y-axis**.

Example

```
#Draw a line in a diagram from position (1, 3) to position (8, 10)
import matplotlib.pyplot as plt
import numpy as np
xpoints = np.array([1, 8])
ypoints = np.array([3,10])
plt.plot(xpoints, ypoints)
plt.show()
```

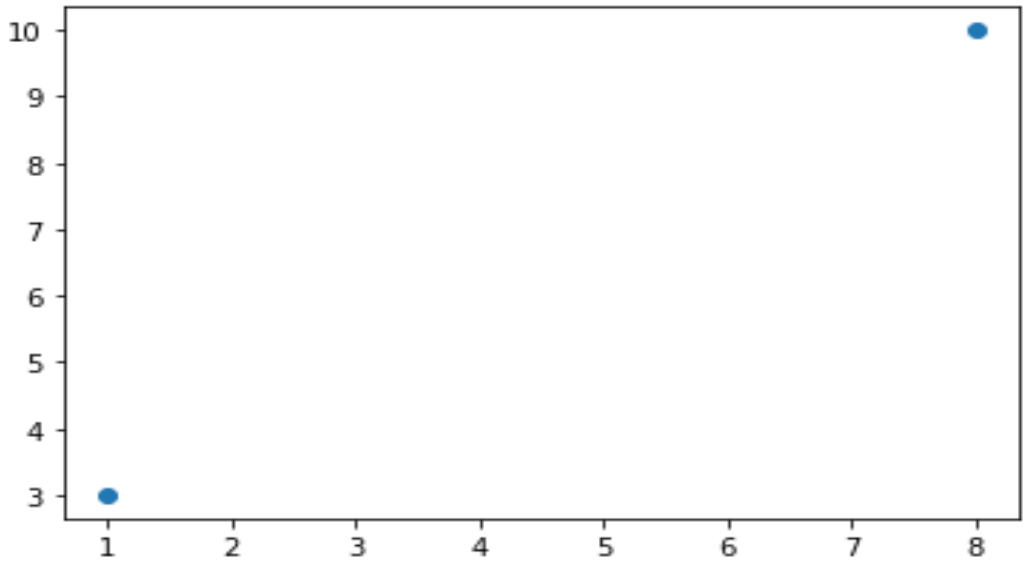


Plotting without Line: To plot only the markers, you can use shortcut string notation parameter **'o'**, which means 'rings'.

Example

#Draw two points in the diagram, one at position (1,3) and one in position (8,10)

```
import matplotlib.pyplot as plt
import numpy as np
xpoints = np.array([1, 8])
ypoints = np.array([3,10])
plt.plot(xpoints, ypoints, 'o')
plt.show()
```

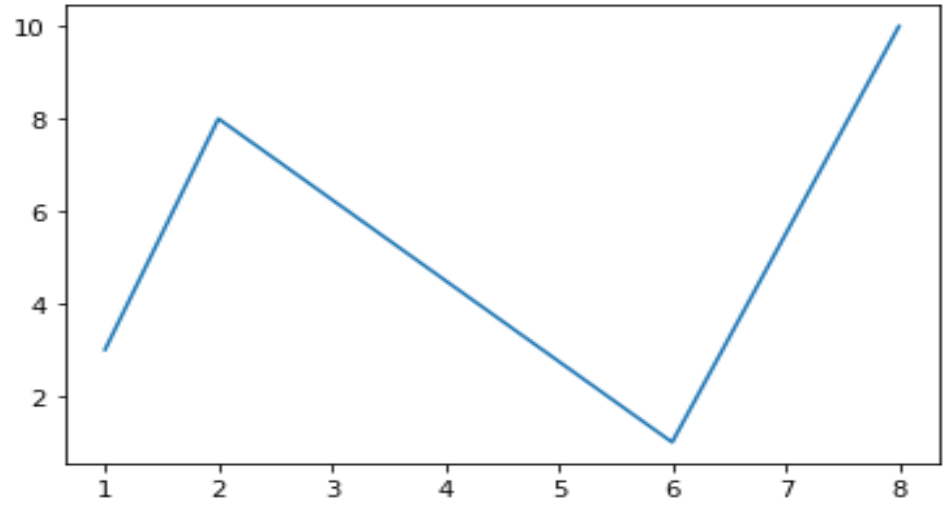


Multiple Points: You can plot as many points as you like, just make sure you have the same number of points in both axis.

Example

#Draw a line in a diagram from position (1, 3) to (2, 8) then to (6, 1) and finally to position (8, 10)

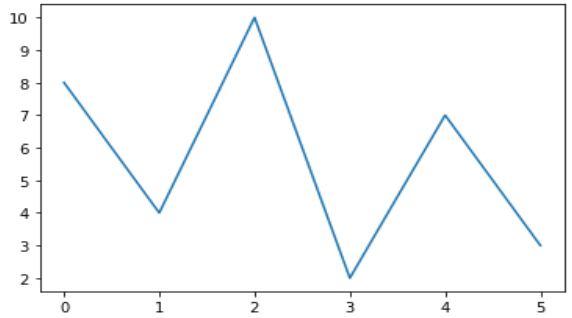
```
import matplotlib.pyplot as plt
import numpy as np
xpoints = np.array([1, 2, 6, 8])
ypoints = np.array([3, 8, 1, 10])
plt.plot(xpoints, ypoints)
plt.show()
```



Default X-Points: If we do not specify the points on the x-axis, they will get the default values 0, 1, 2, 3, etc., depending on the length of the y-points.

Example

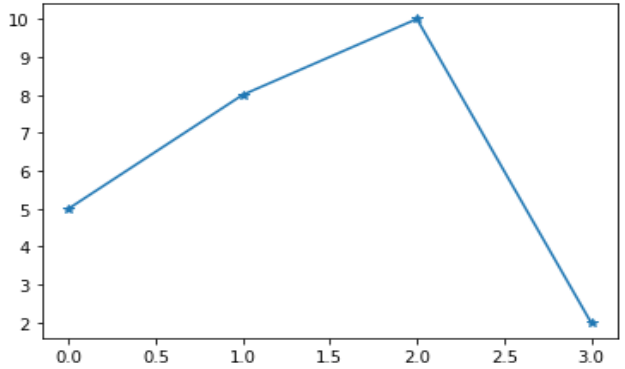
```
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([8, 4, 10, 2, 7, 3])
plt.plot(ypoints)
plt.show()
```



Markers: You can use the keyword argument **marker** to emphasize each point with a specified marker.

Example

```
#Mark each point with a circle
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([5, 8, 10, 2])
plt.plot(ypoints, marker = '*')
plt.show()
```



Marker Reference:

There are different types of markers

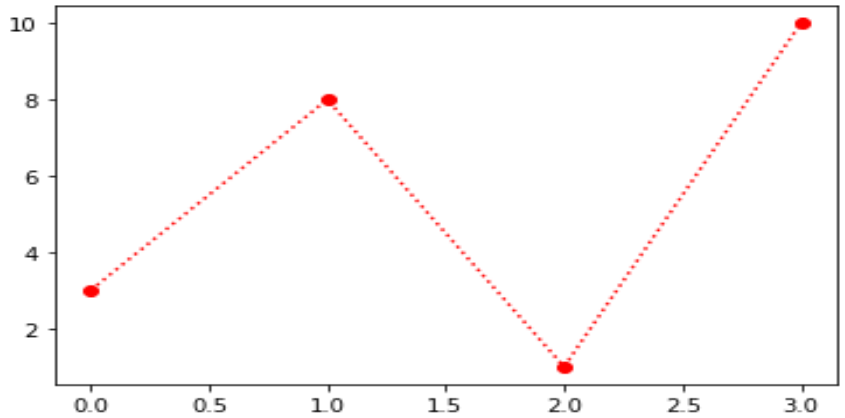
Marker	Description	Marker	Description
o	Circle	H	Hexagon
*	Star	h	Hexagon
.	Point	v	Triangle Down
,	Pixel	^	Triangle Up
x	X	<	Triangle Left
X	X (filled)	>	Triangle Right
+	Plus	1	Tri Down
P	Plus (filled)	2	Tri Up
S	Square	3	Tri Left
D	Diamond	4	Tri Right
d	Diamond (thin)	|	Vline
p	Pentagon	_	Hline

Format Strings: A format string is a compact and flexible way to specify the style of a plot or data points within a plot. It consists of one or more characters that represent the color, marker, and line style of the plot. This parameter is also called **“fmt”.**

Syntax: fmt = marker | line | color

Example

```
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, ‘o:r’)
plt.show()
```



Line Reference

Line Syntax	Description
-	Solid Line
:	Dotted Line
--	Dashed Line
-.	Dashed/ dotted Line

Color Reference

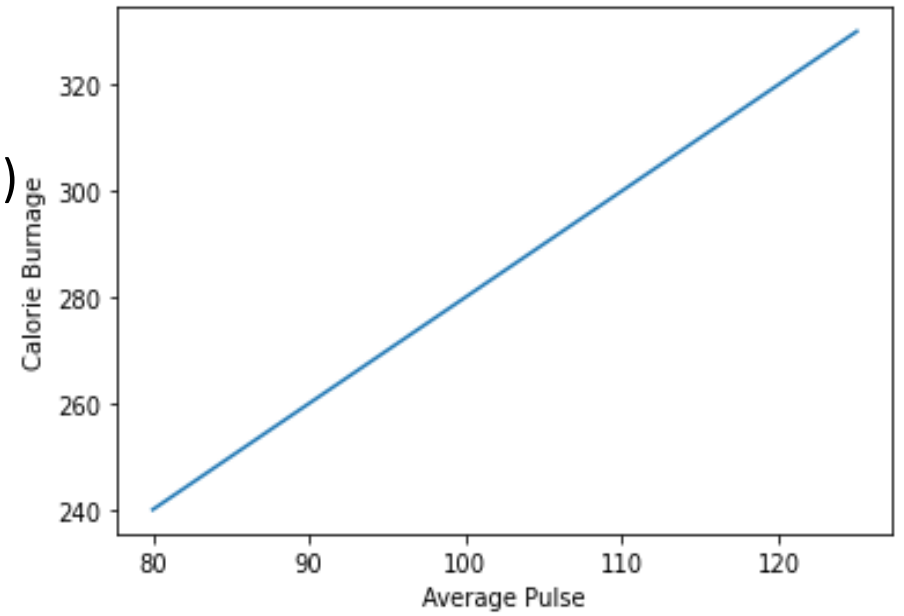
Color Syntax	Description
r	Red
g	Green
b	Blue
c	Cyan
m	Magenta
y	Yellow
k	Black
w	White

Create Labels for a Plot

With Pyplot, you can use the **xlabel()** and **ylabel()** functions to set a label for the x-axis and y-axis.

Example

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])
plt.plot(x, y)
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")
plt.show()
```



Create a Title for a Plot

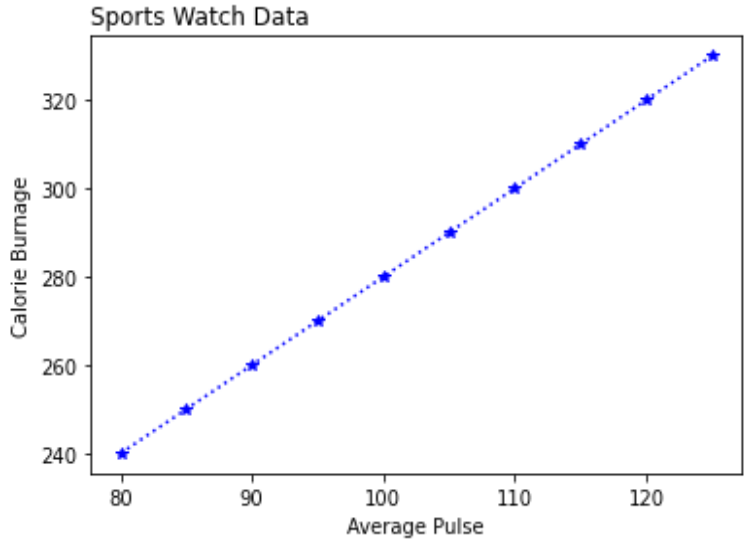
With Pyplot, you can use the **title()** function to set a title for the plot.

Position the Title

You can use the **loc** parameter in title() to position the title. Legal values are: ‘left’, ‘right’, and ‘center’. Default value is ‘center’.

Example

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])
plt.plot(x, y)
plt.title(“Sports Watch Data”, loc = ‘left’)
plt.xlabel(“Average Pulse”)
plt.ylabel(“Calorie Burnage”)
plt.show()
```



Matplotlib Scatter

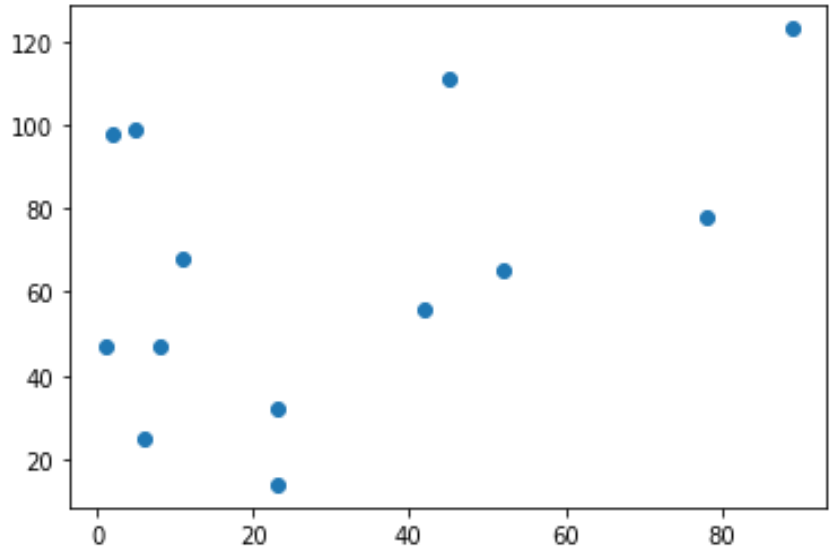
Creating Scatter Plots

With Pyplot, you can use the **scatter()** function to draw a scatter plot.

The **scatter()** function plots one dot for each observation. It needs two arrays of the same length, one for the values of the x-axis, and one for values on the y-axis.

Example

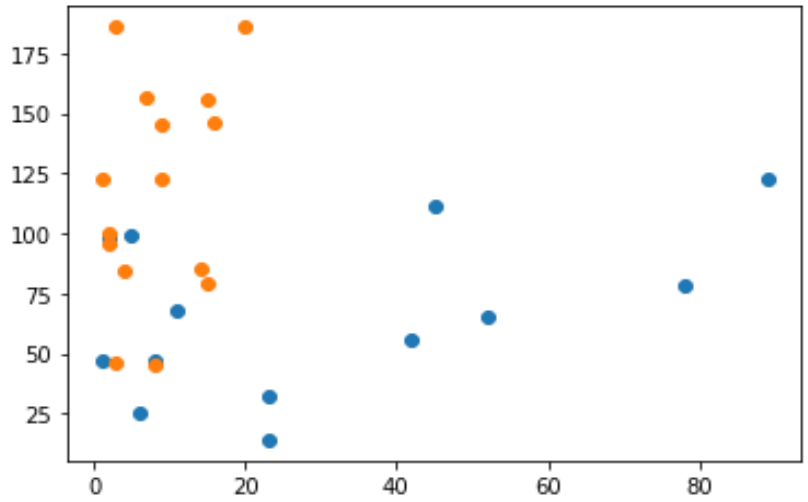
```
import matplotlib.pyplot as plt
import numpy as np
x = np.array([5, 6, 78, 45, 42, 23, 1, 52, 89, 23, 11, 2, 8])
y = np.array([99, 25, 78, 111, 56, 32, 47, 65, 123, 14, 68, 98, 47])
plt.scatter(x, y)
plt.show()
```



Compare Plots

Example

#Draw two plots on the same figure.
import matplotlib.pyplot as plt
import numpy as np
#Day one observation
x = np.array([5, 6, 78, 45, 42, 23, 1, 52, 89, 23, 11, 2, 8])
y = np.array([99, 25, 78, 111, 56, 32, 47, 65, 123, 14, 68, 98, 47])
plt.scatter(x, y)
#Day two observation
x = np.array([2, 4, 8, 9, 15, 3, 1, 9, 14, 2, 7, 16, 20, 3, 15])
y = np.array([100, 84, 45, 123, 156, 186, 123, 145, 85, 96, 157, 146, 186, 46, 79])
plt.scatter(x, y)
plt.show()

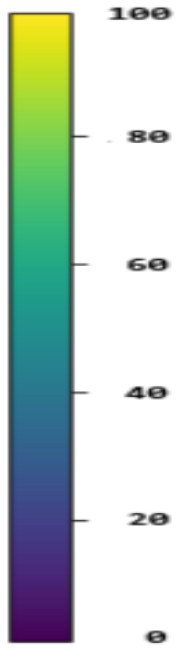


ColorMap

The Matplotlib module has a number of available colormaps.

A colormap is like a list of colors, where each color has a value that ranges from 0 to 100.

This colormap is called ‘viridis’ and as you can see it ranges from 0, which is a purple color, up to 100, which is a yellow color.



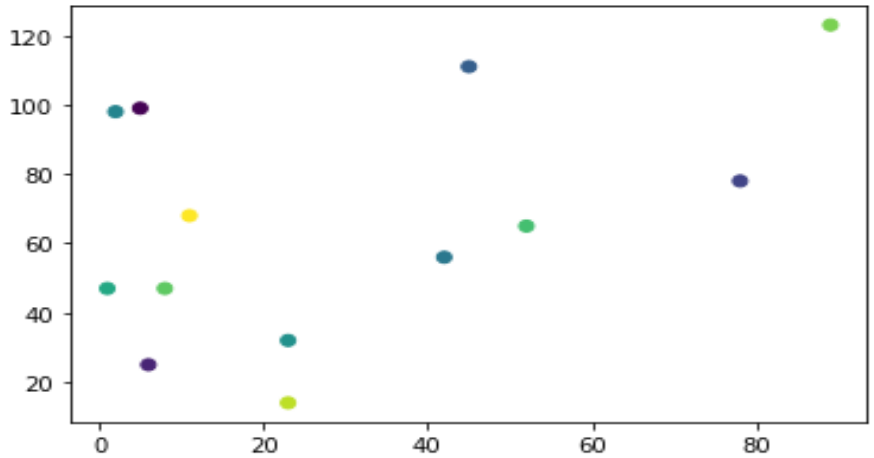
How to use the ColorMap

You can specify the colormap with the keyword argument **cmap** with the value of the colormap, in this case **'viridis'** which is one of the built-in colormaps available in Matplotlib.

In addition you have to create an array with values (from 0 to 100), one value for each point in the scatter plot.

Example

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array([5, 6, 78, 45, 42, 23, 1, 52, 89, 23, 11, 2, 8])
y = np.array([99, 25, 78, 111, 56, 32, 47, 65, 123, 14, 68, 98, 47])
colors = np.array([0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 45, 75])
plt.scatter(x, y, c = colors, cmap = 'viridis')
plt.show()
```



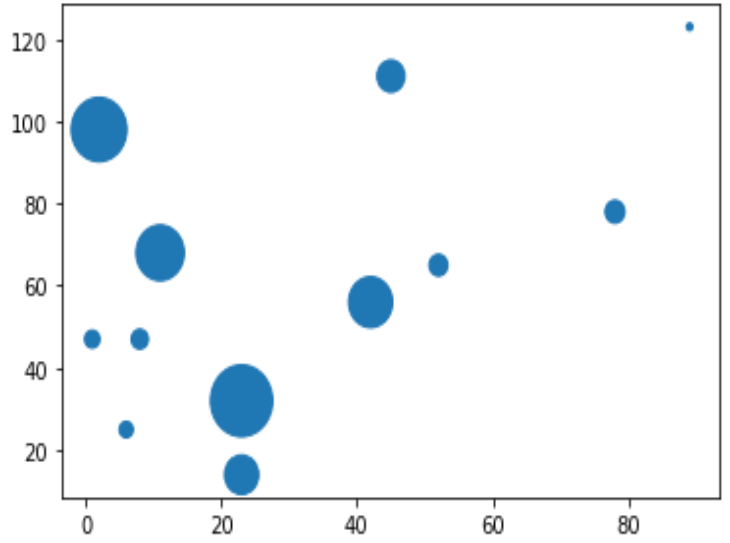
Size

You can change the size of the dots with the **s** argument.

Just like colors, make sure the array for sizes has the same length as the arrays for the x-axis and y-axis.

Example

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array([5, 6, 78, 45, 42, 23, 1, 52, 89, 23, 11, 2, 8])
y = np.array([99, 25, 78, 111, 56, 32, 47, 65, 123, 14, 68, 98, 47])
sizes = np.array([20, 50, 100, 200, 500, 1000, 60, 90, 10, 300, 600, 800, 75])
plt.scatter(x, y, s = sizes)
plt.show()
```



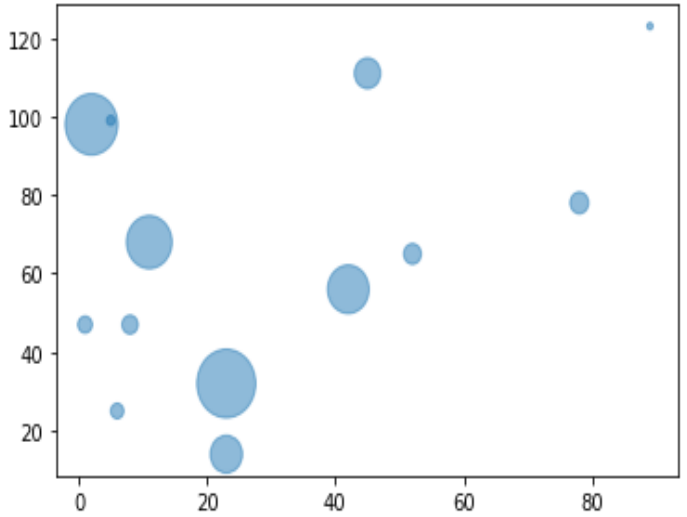
Alpha

You can adjust the transparency of the dots with the **alpha** argument.

Just like colors, make sure the array for sizes has the same length as the arrays for the x-axis and y-axis.

Example

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array([5, 6, 78, 45, 42, 23, 1, 52, 89, 23, 11, 2, 8])
y = np.array([99, 25, 78, 111, 56, 32, 47, 65, 123, 14, 68, 98, 47])
sizes = np.array([20, 50, 100, 200, 500, 1000, 60, 90, 10, 300, 600, 800, 75])
plt.scatter(x, y, s = sizes, alpha = 0.5)
plt.show()
```

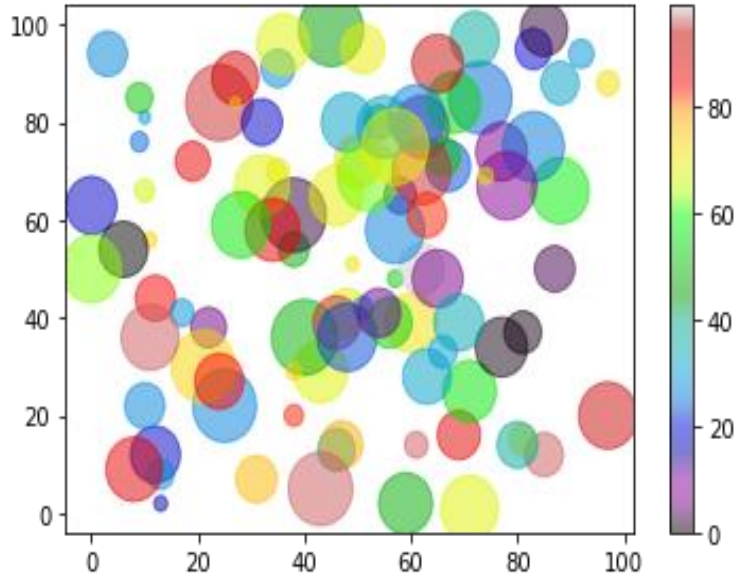


Combine Color, Size and Alpha

Example

#Create random arrays with 100 values for x-points, y-points, colors and sizes.

```
import matplotlib.pyplot as plt
import numpy as np
x = np.random.randint(100, size = (100))
y = np.random.randint(100, size =(100))
colors = np.random.randint(100, size = (100))
sizes = 10 * np.random.randint(100, size=(100))
plt.scatter(x, y, c = colors, s = sizes, alpha = 0.5, cmap = 'nipy_spectral')
plt.colorbar()
Plt.show()
```



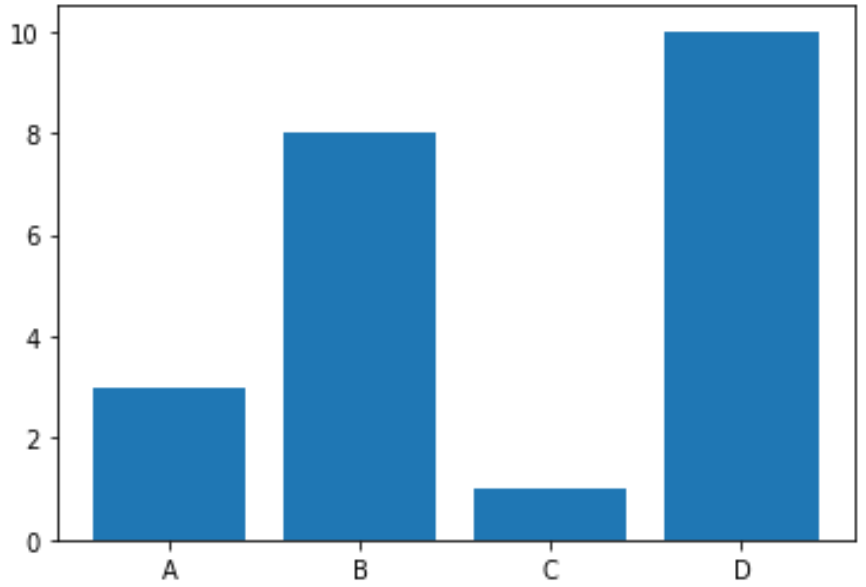
Matplotlib Bar Charts

Creating Bars

With Pyplot, you can use the **bar()** function to draw bar graphs.

Example

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])
plt.bar(x,y)
plt.show()
```

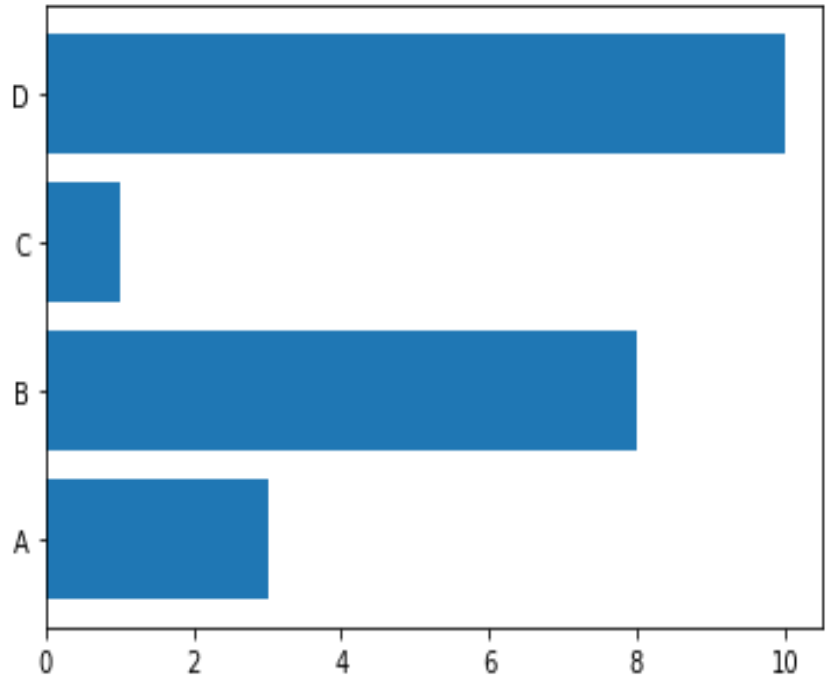


Horizontal Bars

If you want the bars to be displayed horizontally instead of vertically, use the **barh()** function

Example

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])
plt.barh(x,y)
plt.show()
```

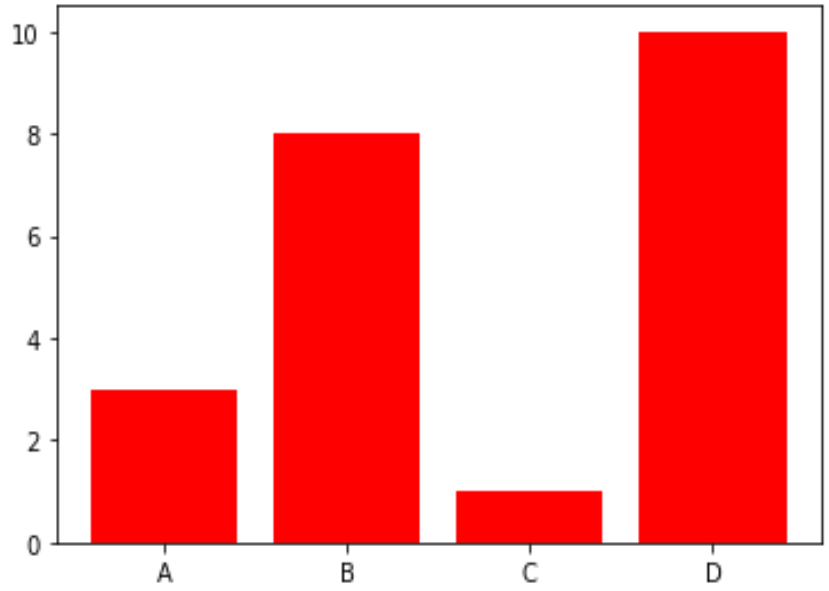


Bar Color

The **bar()** and **barh()** takes the keyword argument **color** to set the color of the bars.

Example

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])
plt.bar(x,y, color = "red")
plt.show()
```

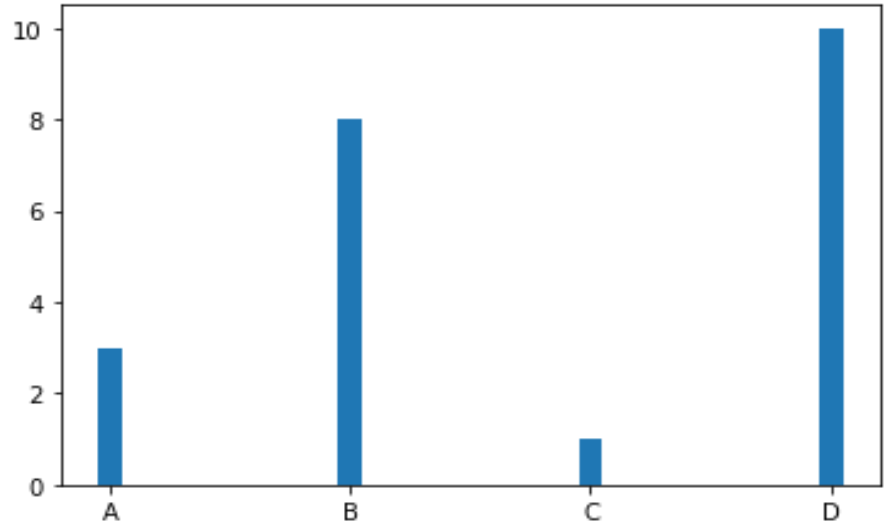


Bar Width

The **bar()** takes the keyword argument **width** to set the width of the bars.

Example

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])
plt.bar(x,y, width = 0.1)
plt.show()
```

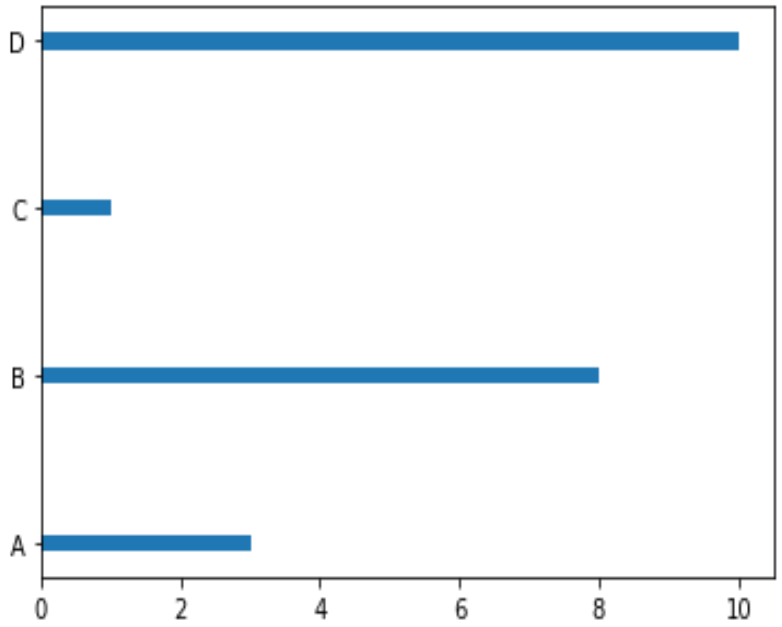


Bar Height

The **barh()** takes the keyword argument **height** to set the height of the bars.

Example

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])
plt.barh(x,y, height = 0.1)
plt.show()
```



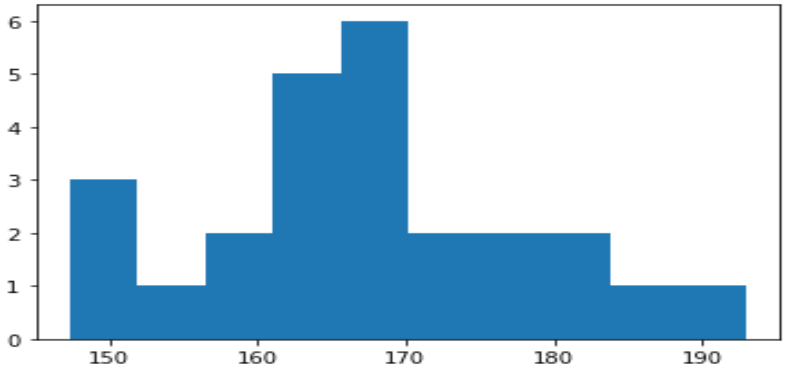
Matplotlib Histograms

A histogram is a graph showing frequency distributions.
It is a graph showing the number of observations within each given interval.

Create Histogram
In Matplotlib, we use the **hist()** function to create histograms.
The **hist()** function will use an array of numbers to create a histogram, the array is sent into the function as an argument.

Example

```
import matplotlib.pyplot as plt
import numpy as np
x = np.random.normal(170, 10, 250)
plt.hist(x)
plt.show()
```



Matplotlib Pie Charts

Creating Pie Charts

With Pyplot, you can use the **pie()** function to draw pie charts.

Example

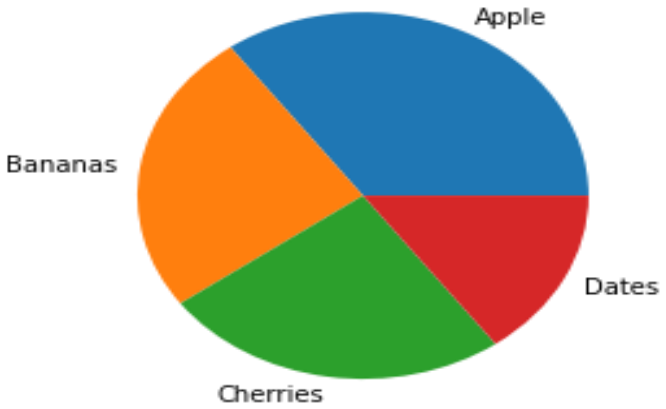
```
import matplotlib.pyplot as plt
import numpy as np
y = np.array([35, 25, 25, 15])
plt.pie(y)
plt.show()
```

Labels

Add labels to the pie chart with the **labels** parameter. The labels parameter must be an array with one label for each wedge.

Example

```
import matplotlib.pyplot as plt
import numpy as np
y = np.array([35, 25, 25, 15])
mylables = [“Apple”, “Bananas”, “Cherries”, “Dates”]
plt.pie(y, labels = mylabels)
plt.show()
```

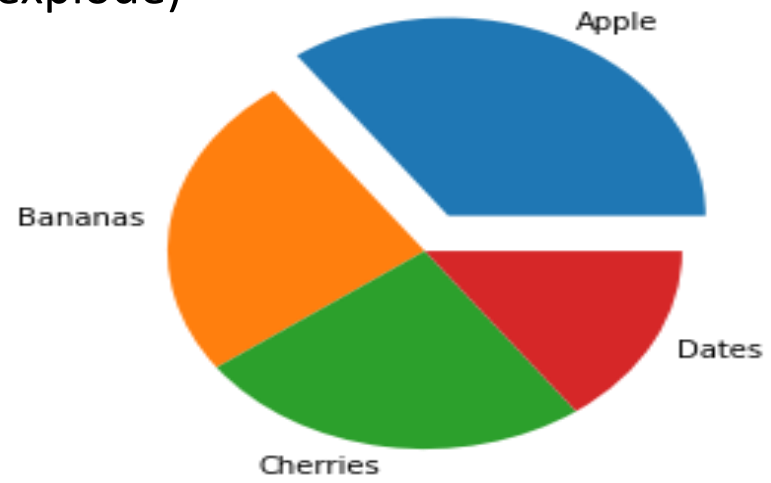


Explode

The **explode** parameter allows you to make one of the wedges to stand out. The explode parameter, if specified, and not **None,** must be an array with one value for each wedge. Each value represents how far from the center each wedge is displayed.

Example

```
#Pull the “Apples” wedge 0.2 from the center of the pie
import matplotlib.pyplot as plt
import numpy as np
y = np.array([35, 25, 25, 15])
mylables = [“Apple”, “Bananas”, “Cherries”, “Dates”]
myexplode = [0.2, 0, 0, 0]
plt.pie(y, labels = mylabels, explode = myexplode)
plt.show()
```

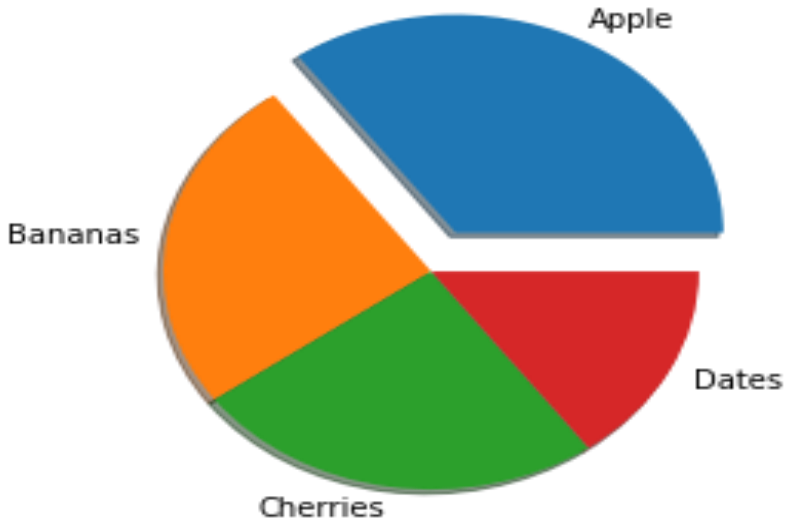


Shadow

Add a shadow to the pie chart by setting the **shadows** parameter to **True**

Example

```
import matplotlib.pyplot as plt
import numpy as np
y = np.array([35, 25, 25, 15])
mylables = [“Apple”, “Bananas”, “Cherries”, “Dates”]
myexplode = [0.2, 0, 0, 0]
plt.pie(y, labels = mylabels, explode = myexplode, shadow = True)
plt.show()
```

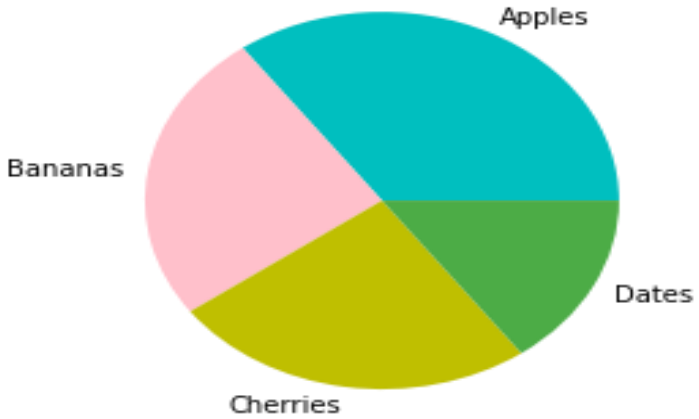


Colors

You can set the color of each wedge with the **colors** parameter. The colors parameter, if specified, must be an array with one value for each wedge.

Example

```
#Specify a new color for each wedge
import matplotlib.pyplot as plt
import numpy as np
y = np.array([35, 25, 25, 15])
mylabels = [“Apples”, “Bananas”, “Cherries”, “Dates”]
mycolors = [“c”, “pink”, “y”, “#4CAC46”]
plt.pie(y, labels = mylabels, colors = mycolors)
plt.show()
```

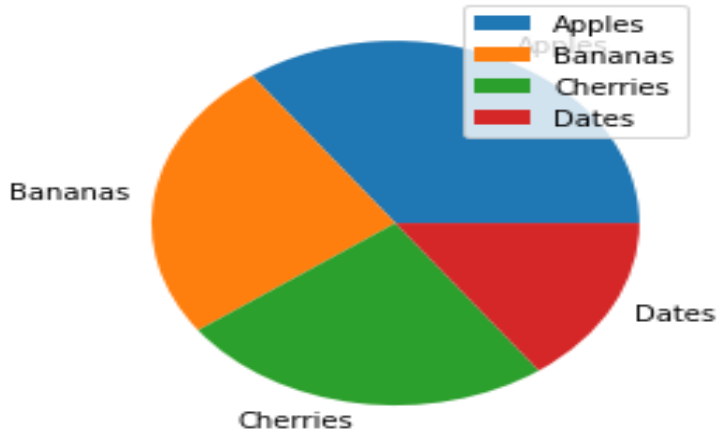


Legend

To add a list of explanation for each wedge, use the **legend()** function.

Example

```
import matplotlib.pyplot as plt
import numpy as np
y = np.array([35, 25, 25, 15])
mylabels = [“Apples”, “Bananas”, “Cherries”, “Dates”]
plt.pie(y, labels = mylabels)
plt.legend()
plt.show()
```

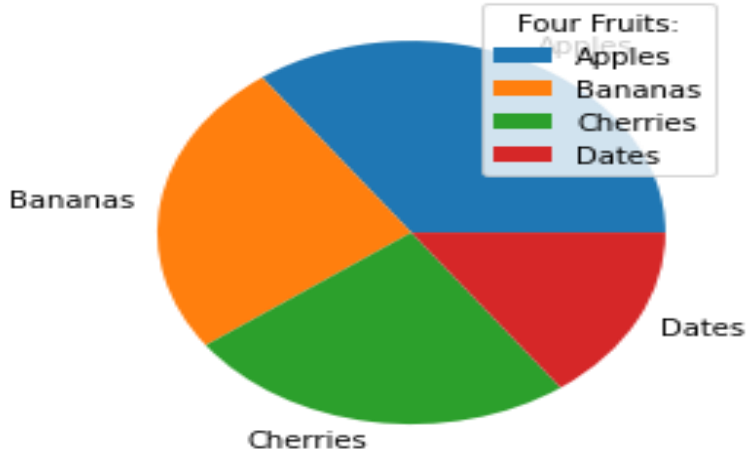


Legend with Header

To add a header to the legend, add the **title** parameter to the legend function.

Example

```
import matplotlib.pyplot as plt
import numpy as np
y = np.array([35, 25, 25, 15])
mylabels = [“Apples”, “Bananas”, “Cherries”, “Dates”]
plt.pie(y, labels = mylabels)
plt.legend(title = “Four Fruits:”)
plt.show()
```



Seaborn Module

Seaborn is a data visualization library in Python. It helps to create interesting and interactive plots to draw meaningful information from data.

SEABORN INSTALLATION

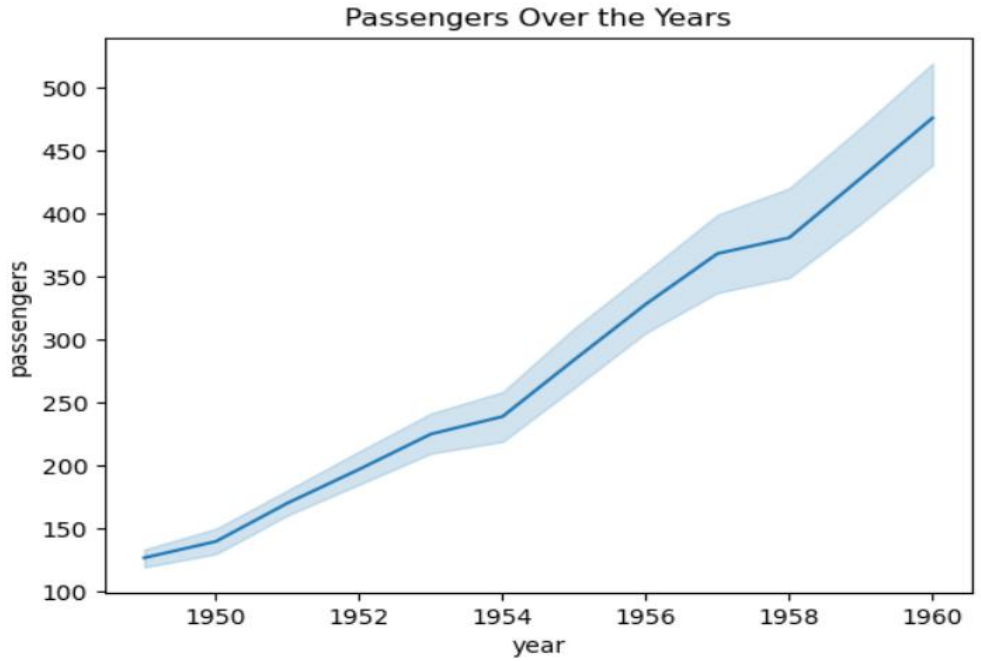
pip install seaborn

Import Seaborn

import seaborn as sns

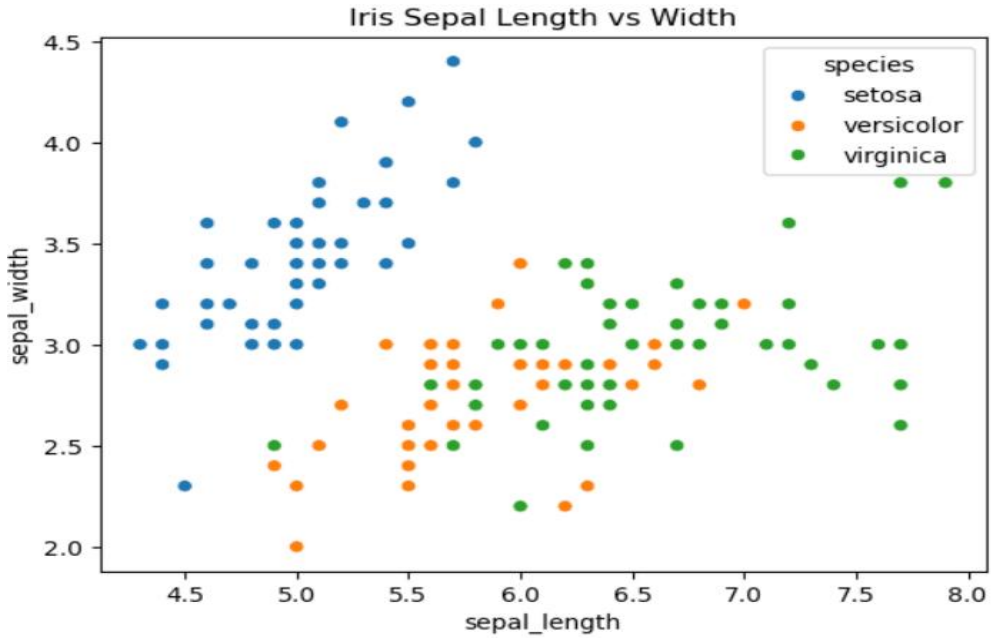
Line plot

```
import seaborn as sns
import matplotlib.pyplot as plt
# Load sample dataset
data = sns.load_dataset("flights")
# Create line plot
sns.lineplot(x="year", y="passengers", data=data)
plt.title("Passengers Over the Years")
plt.show()
```



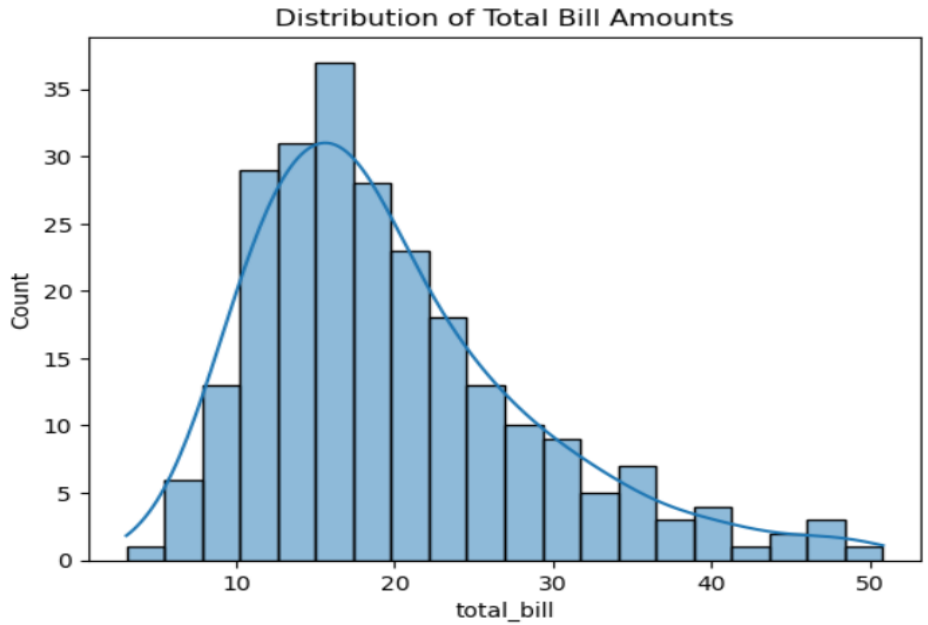
Scatter Plot

```
import seaborn as sns
import matplotlib.pyplot as plt
# Load dataset
data = sns.load_dataset("iris")
# Create scatter plot
sns.scatterplot(x="sepal_length", y="sepal_width", hue="species", data=data)
plt.title("Iris Sepal Length vs Width")
plt.show()
```



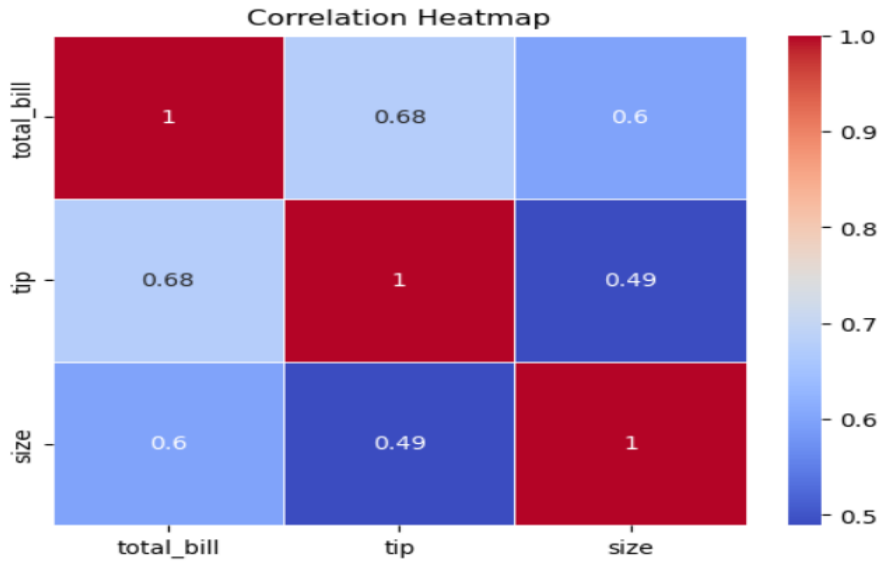
Histogram

```
import seaborn as sns
import matplotlib.pyplot as plt
# Load dataset
data = sns.load_dataset("tips")
# Create histogram
sns.histplot(data["total_bill"], bins=20, kde=True)
plt.title("Distribution of Total Bill Amounts")
plt.show()
```



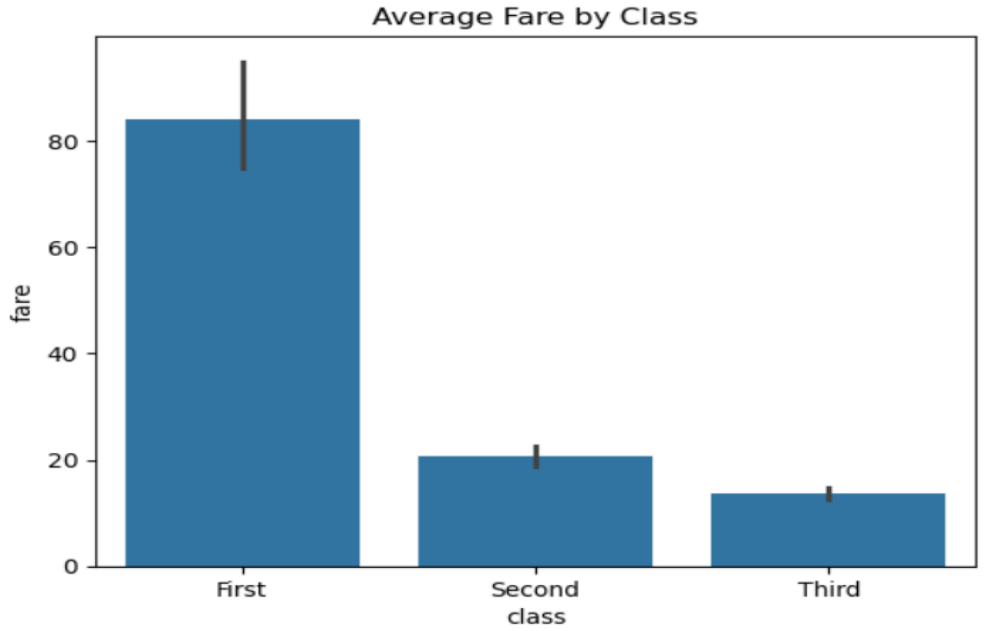
Heatmap (Correlation Matrix)

```
import seaborn as sns
import matplotlib.pyplot as plt
# Load dataset
data = sns.load_dataset("tips")
# Convert categorical columns to numeric (optional, only if needed)
data_numeric = data.select_dtypes(include=["number"])  # Select only numeric columns
# Compute correlation matrix
corr = data_numeric.corr()
# Create heatmap
sns.heatmap(corr, annot=True, cmap="coolwarm", linewidths=0.5)
plt.title("Correlation Heatmap")
plt.show()
```



Bar Plot

```
import seaborn as sns
import matplotlib.pyplot as plt
# Load dataset
data = sns.load_dataset("titanic")
# Create bar plot
sns.barplot(x="class", y="fare", data=data)
plt.title("Average Fare by Class")
plt.show()
```



WEB DEVELOPMENT

Flask Module

Flask is a **lightweight web framework** for building web applications in Python. It is designed to be **simple, easy to use, and flexible**, making it ideal for both beginners and experienced developers. Flask is **micro**, meaning it doesn't impose dependencies or large structure, but you can extend it with libraries for additional features.

INSTALLATION

pip install flask

#Create your First Flask App

```
from flask import Flask

app = Flask(__name__)

@app.route("/")  # Define route for the home page
def hello_world():
    return "Hello, World!"  # Return a response to the client

if __name__ == "__main__":
    app.run(debug=True)  # Run the app in debug mode
```

#Define More Routes

```
from flask import Flask

app = Flask(__name__)

@app.route("/")  # Home page
def home():
    return "Welcome to the Home Page"

@app.route("/about")  # About page
def about():
    return "About Us"

if __name__ == "__main__":
    app.run(debug=True)
```

#Create a Form (POST and GET Methods)

```
from flask import Flask, render_template, request

app = Flask(__name__)

@app.route("/", methods=["GET", "POST"])
def home():
    if request.method == "POST":
        name = request.form.get("name")  # Get data from the form
        return f"Hello, {name}!"  # Display the user's name
    return render_template("form.html")  # Show the form on GET request

if __name__ == "__main__":
    app.run(debug=True)
```

#Redirect to another Route

```
from flask import Flask, redirect, url_for

app = Flask(__name__)

@app.route("/")
def home():
    return redirect(url_for("about"))  # Redirect to the 'about' route

@app.route("/about")
def about():
    return "This is the About Page"

if __name__ == "__main__":
    app.run(debug=True)
```

Django Module

Django is a **high-level web framework** for Python that encourages **rapid development** and follows the **"Don't repeat yourself" (DRY)** principle. It is used for building **robust, scalable, and secure web applications**. Django includes **built-in tools** for handling user authentication, database models, routing, and much more.

INSTALLATION

pip install django

#creating a Django Project

```
django-admin startproject myproject  # Creates a new project directory
cd myproject
python manage.py runserver  # Run the server
```

#Creating an App

```
python manage.py startapp myapp  # Create a new app directory
```

#Create Views: Views in Django handle HTTP requests and return HTTP responses.

```
# myapp/views.py
from django.http import HttpResponse

def home(request):
    return HttpResponse("Hello, World!")
```

#Configure URLs: URLs map web addresses to views in Django.

```
# myproject/urls.py
from django.contrib import admin
from django.urls import path
from myapp import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.home),  # Home page route
]
```

WEB SCRAPING

BeautifulSoup Module

BeautifulSoup is a **library used for web scraping** and **parsing HTML and XML documents**. It provides simple methods for navigating the document tree, searching for elements, and extracting data.

INSTALLATION

pip install beautifulsoup4

#Parsing HTML Content

```
from bs4 import BeautifulSoup

html_doc = """
<html>
   <head><title>Test Page</title></head>
   <body>
      <h1>Welcome to BeautifulSoup</h1>
      <p class="content">This is a sample HTML document for scraping.</p>
      <p class="content">Another paragraph.</p>
   </body>
</html>
"""

soup = BeautifulSoup(html_doc, "html.parser")
print(soup.prettify())  # Pretty-print the HTML structure
```

#Accessing HTML Elements

```
# Get the title tag
title_tag = soup.title
print(title_tag)  # Output: <title>Test Page</title>

# Get the text inside the title tag
print(title_tag.string)  # Output: Test Page
```

#Accessing Tags and Attributes

```
# Get the first <p> tag with class 'content'
p_tag = soup.find('p', class_='content')
print(p_tag)  # Output: <p class="content">This is a sample HTML document for scraping.</p>

# Get the text inside the <p> tag
print(p_tag.get_text())  # Output: This is a sample HTML document for scraping.

# Get an attribute of the tag (e.g., class)
print(p_tag['class'])  # Output: ['content']
```

#Find all Occurrences of a Tag

```
# Find all <p> tags with class 'content'
p_tags = soup.find_all('p', class_='content')
for tag in p_tags:
    print(tag.get_text())  # Output: text of each <p> tag
```

#Find a Tage by ID

```
# Find the tag with a specific id
soup = BeautifulSoup("""
<html>
    <body>
        <div id="main-content">
            <h2>Title</h2>
            <p>Paragraph content</p>
        </div>
    </body>
</html>
""", "html.parser")

main_content = soup.find(id="main-content")
print(main_content.get_text())  # Output: TitleParagraph content
```

#Modify a Tag’s Content

```
# Modify the content inside a <h1> tag
h1_tag = soup.h1
h1_tag.string = "New Title"
print(soup.h1)  # Output: <h1>New Title</h1>
```

#Add a New Tag

```
# Add a new <p> tag to the body
new_tag = soup.new_tag("p")
new_tag.string = "This is a new paragraph."
soup.body.append(new_tag)

print(soup.body)  # Output: <body>... <p>This is a new paragraph.</p></body>
```

#working with CSS Selectors

```
html_doc = """
<html>
   <body>
      <div class="content">
         <h2>Header 1</h2>
         <p>Paragraph 1</p>
      </div>
      <div class="content">
         <h2>Header 2</h2>
         <p>Paragraph 2</p>
      </div>
   </body>
</html>
"""

soup = BeautifulSoup(html_doc, "html.parser")
# Find all div elements with class 'content'
content_divs = soup.select("div.content")
for div in content_divs:
   print(div.get_text())
```

AUTOMATION & SCRIPTING

Selenium Module

Selenium is a powerful tool for automating web browsers. It allows you to simulate user interactions with a web page, such as clicking buttons, entering text in forms, and scraping dynamic content from websites.

INSTALLATION

pip install selenium

#Interacting with Web Elements

```
from selenium import webdriver
from selenium.webdriver.common.by import By

driver = webdriver.Chrome(executable_path="path/to/chromedriver")
driver.get("https://www.example.com")

# Find element by ID
element = driver.find_element(By.ID, "element_id")

# Find element by Class Name
element = driver.find_element(By.CLASS_NAME, "class_name")

# Find element by Name
element = driver.find_element(By.NAME, "element_name")

# Do something with the element (e.g., click, send keys)
element.click()

driver.quit()
```

#Enter Text in Input Fields

```
input_field = driver.find_element(By.NAME, "username")
input_field.send_keys("myusername")  # Simulate typing in the input field
```

#Click a Button

```
button = driver.find_element(By.ID, "submit_button")
button.click()  # Simulate clicking a button
```

#Submit a Form

```
form = driver.find_element(By.NAME, "login_form")
form.submit()  # Submit the form
```

#Accept or Dismiss Alerts

```
# Accept an alert
alert = driver.switch_to.alert
alert.accept()

# Dismiss an alert
alert.dismiss()
```

#Getting Alert Text

```
alert = driver.switch_to.alert
alert_text = alert.text
print(alert_text)  # Output: Text of the alert
```

#Taking Screenshots

```
driver = webdriver.Chrome(executable_path="path/to/chromedriver")
driver.get("https://www.example.com")

# Take a screenshot and save it as a PNG file
driver.save_screenshot("screenshot.png")

driver.quit()
```

#Closing the Browser

```
driver.quit()  # Close the browser window and end the session
```

PyAutoGUI Module

PyAutoGUI is a **GUI automation** library for Python, which allows you to programmatically control the **mouse**, **keyboard**, and **screenshots**. It is useful for automating repetitive tasks such as **testing applications, form filling, clicking buttons**, and **taking screenshots**.

INSTALLATION

```
pip install pyautogui
```

#Move the Mouse

```
import pyautogui

# Move the mouse to (x=100, y=200)
pyautogui.moveTo(100, 200)

# Move the mouse to (x=500, y=500) over 2 seconds
pyautogui.moveTo(500, 500, duration=2)
```

#Clicking with the Mouse

```
# Click at the current position
pyautogui.click()

# Click at specific coordinates
pyautogui.click(200, 300)

# Double click at the current position
pyautogui.doubleClick()

# Right-click at the current position
pyautogui.rightClick()
```

#Drag the Mouse

```
# Move the mouse to (100, 100) and then drag it to (200, 200)
pyautogui.moveTo(100, 100)
pyautogui.dragTo(200, 200, duration=1)
```

#Typing Text

```
# Type the text "Hello, world!"
pyautogui.write("Hello, world!")

# Type with a delay between each character
pyautogui.write("Hello, world!", interval=0.2)
```

#Pressing Keys

```
# Press the Enter key
pyautogui.press("enter")

# Press multiple keys (e.g., Shift + A)
pyautogui.hotkey("shift", "a")

# Simulate pressing Ctrl + C
pyautogui.hotkey("ctrl", "c")
```

#Taking Screenshots

```
# Take a screenshot of the entire screen
screenshot = pyautogui.screenshot()

# Save the screenshot as a PNG file
screenshot.save("screenshot.png")

# Capture a region (left=100, top=100, width=300, height=400)
screenshot = pyautogui.screenshot(region=(100, 100, 300, 400))
screenshot.save("partial_screenshot.png")
```

#Automating the Task of Opening an Application: For example, you can automate opening a calculator on Windows by following these steps:

```
import pyautogui
import time

# Open Start Menu (Press the Windows key)
pyautogui.hotkey('win', 'r')

# Type "calc" to open the calculator
pyautogui.write("calc")
pyautogui.press("enter")

# Wait for the calculator to open
time.sleep(1)

# Perform some operations (simulating keypresses)
pyautogui.write("123+456")
pyautogui.press("enter")
```

NETWORKING

Requests Module

The requests module is a **simple and elegant HTTP library** for sending HTTP requests in Python. It is designed to be easy to use for **sending HTTP requests**, handling responses, working with APIs, and interacting with web servers.

INSTALLATION
pip install requests

#Basic GET Request

```
import requests

# Send a GET request to a URL
response = requests.get('https://jsonplaceholder.typicode.com/posts/1')

# Print the status code and the response content
print(response.status_code)  # Output: 200 (OK)
print(response.text)  # Output: Response body (JSON data in this case)
```

#GET Request with query Parameter

```
import requests

# Sending GET request with query parameters
params = {'userId': 1}
response = requests.get('https://jsonplaceholder.typicode.com/posts', params=params)

# Print the response content
print(response.json())  # Output: List of posts for user with ID 1
```

#Basic POST Request

```
# Data to send in the POST request
data = {'title': 'foo', 'body': 'bar', 'userId': 1}

# Sending a POST request
response = requests.post('https://jsonplaceholder.typicode.com/posts', json=data)

# Print the status code and the response content
print(response.status_code)  # Output: 201 (Created)
print(response.json())  # Output: The newly created post data
```

#Handling HTTP Headers

```
import requests

headers = {'User-Agent': 'my-app', 'Authorization': 'Bearer YOUR_TOKEN'}
response = requests.get('https://jsonplaceholder.typicode.com/posts/1', headers=headers)

# Print the response status and content
print(response.status_code)
print(response.text)
```

#Handling Redirects

```
import requests

# Disable automatic redirects
response = requests.get('http://github.com', allow_redirects=False)

print(response.status_code)  # Output: 301 (Redirect)
print(response.headers['Location'])  # Output: URL to which it's redirecting
```

#Authentication

```
import requests
from requests.auth import HTTPBasicAuth

# Sending a GET request with Basic Authentication
response = requests.get('https://httpbin.org/basic-auth/user/pass', auth=HTTPBasicAuth('user',
'pass'))

# Print the response content
print(response.status_code)  # Output: 200 if authentication is successful
```

GAME DEVELOPMENT

PyGame Module

PyGame is a cross-platform **game development library** for creating **2D games** and multimedia applications. It provides modules for handling **graphics**, **sound**, **events**, and more, making it an ideal choice for building games and simulations.

INSTALLATION

pip install pygame

#Initialize PyGame and Create a Window

```
import pygame

# Initialize Pygame
pygame.init()

# Set up display window
screen = pygame.display.set_mode((800, 600))  # Window size: 800x600
# Set window title
pygame.display.set_caption("My First Game")

# Main game loop
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:  # Check if the user closes the window
            running = False

# Quit Pygame
pygame.quit()
```

#Handle Key Presses

```
# Set up display window
screen = pygame.display.set_mode((800, 600))
pygame.display.set_caption("Event Handling")

# Main game loop
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        elif event.type == pygame.KEYDOWN:  # Check if any key is pressed
            if event.key == pygame.K_LEFT:  # Left arrow key
                print("Left arrow key pressed!")
            elif event.key == pygame.K_RIGHT:  # Right arrow key
                print("Right arrow key pressed!")

pygame.quit()
```

#Draw Shapes (Rectangle)

```
# Set up display window
screen = pygame.display.set_mode((800, 600))
pygame.display.set_caption("Drawing Shapes")

# Colors
RED = (255, 0, 0)

# Main game loop
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
    # Fill screen with black color
    screen.fill((0, 0, 0))
    # Draw a rectangle (x, y, width, height)
    pygame.draw.rect(screen, RED, (100, 100, 200, 150))
    pygame.display.update()  # Update the screen

pygame.quit()
```

#Render Text

```
# Set up display window
screen = pygame.display.set_mode((800, 600))
pygame.display.set_caption("Text Rendering")
# Set font for rendering text
font = pygame.font.Font(None, 36)  # Default font, size 36
text = font.render("Hello, PyGame!", True, (255, 255, 255))  # White text
# Main game loop
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
    # Fill the screen with black color
    screen.fill((0, 0, 0))
    # Draw text on screen at (x=100, y=100)
    screen.blit(text, (100, 100))
    pygame.display.update()  # Update the screen

pygame.quit()
```

#Playing a Sound

```
# Set up display window
screen = pygame.display.set_mode((800, 600))
pygame.display.set_caption("Playing Sound")

# Load a sound file (WAV format)
sound = pygame.mixer.Sound('sound.wav')

# Play the sound
sound.play()

# Main game loop
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

pygame.quit()
```

#Playing Background Music

```
# Set up display window
screen = pygame.display.set_mode((800, 600))
pygame.display.set_caption("Playing Music")

# Load background music (MP3 format)
pygame.mixer.music.load('background.mp3')

# Play music in a loop (-1 means infinite loop)
pygame.mixer.music.play(-1)

# Main game loop
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

pygame.quit()
```

IMAGE PROCESSING MODULES IN PYTHON

Pillow Module

Pillow (formerly **PIL - Python Imaging Library**) is a **lightweight and easy-to-use image processing library** in Python. It allows you to **open, edit, and save images**, perform **image transformations, filtering, and conversions**.

INSTALLATION
pip install pillow

Example
#Open an Image

```
from PIL import Image

# Open an image file
image = Image.open("example.jpg")

# Show the image
image.show()
```

#Check Image Format, Mode and Size

```
print(image.format)  # Output: JPEG, PNG, etc.
print(image.size)    # Output: (Width, Height)
print(image.mode)    # Output: RGB, RGBA, L (grayscale)
```

#Resize an Image

```
resized_image = image.resize((200, 200))
resized_image.show()
```

#Crop an Image

```
cropped_image = image.crop((50, 50, 300, 300))  # (left, top, right, bottom)
cropped_image.show()
```

#Rotate an Image

```
rotated_image = image.rotate(90)  # Rotate by 90 degrees
rotated_image.show()
```

#Convert Image to GrayScale

```
gray_image = image.convert("L")
gray_image.show()
```

#Convert Image to Black and White

```
bw_image = image.convert("1")  # Convert to black & white
bw_image.show()
```

#Save an Image in a Different Format

```
gray_image.save("output.png")  # Convert and save as PNG
```

#Apply a Blur Filter

```
from PIL import ImageFilter

blurred_image = image.filter(ImageFilter.BLUR)
blurred_image.show()
```

#Apply Sharpening

```
sharpened_image = image.filter(ImageFilter.SHARPEN)
sharpened_image.show()
```

#Apply Edge Detection

```
edges = image.filter(ImageFilter.FIND_EDGES)
edges.show()
```

#Generate Thumbnail

```
thumbnail = image.copy()
thumbnail.thumbnail((100, 100))
thumbnail.show()
```

#Save image in Different Formats

```
image.save("output.png")  # Save as PNG
image.save("output.bmp")  # Save as BMP
```

#Compress Image while Saving

```
image.save("compressed.jpg", quality=50)  # Reduce quality to 50%
```

OpenCV Module

OpenCV is a **powerful computer vision and image processing library** used for **image recognition, object detection, video processing, and real-time applications** like face detection and motion tracking.

INSTALLATION

pip install opencv-python

For additional functionalities like video encoding

pip install opencv-python-headless

Example

```
#Read and Display an Image
import cv2
# Read an image
image = cv2.imread("example.jpg")
# Show the image
cv2.imshow("Image", image)
# Wait for a key press and close the window
cv2.waitKey(0)
cv2.destroyAllWindows()
```

#Save an Image

```
cv2.imwrite("output.jpg", image)
```

#Resize an Image

```
resized_image = cv2.resize(image, (300, 300))  # Resize to 300x300
cv2.imshow("Resized Image", resized_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

#Crop an Image

```
cropped_image = image[50:300, 100:400]  # Crop region (y1:y2, x1:x2)
cv2.imshow("Cropped Image", cropped_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

#Convert Image To GrayScale

```
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
cv2.imshow("Grayscale Image", gray_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

#Convert Image to HSV (Hue, Saturation, Value)

```
hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
cv2.imshow("HSV Image", hsv_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

#Apply Gaussian Blur

```
blurred = cv2.GaussianBlur(image, (15, 15), 0)
cv2.imshow("Blurred Image", blurred)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

#Apply Edge Detection (Canny)

```
edges = cv2.Canny(image, 100, 200)  # Thresholds: 100 and 200
cv2.imshow("Edge Detection", edges)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
#Open a Video File
cap = cv2.VideoCapture("video.mp4")

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    cv2.imshow("Video", frame)
    if cv2.waitKey(25) & 0xFF == ord("q"):  # Press 'q' to exit
        break

cap.release()
cv2.destroyAllWindows()
```

#Capture Video From Webcam

```
cap = cv2.VideoCapture(0)  # 0 for default webcam

while True:
    ret, frame = cap.read()
    cv2.imshow("Webcam", frame)
    if cv2.waitKey(1) & 0xFF == ord("q"):  # Press 'q' to exit
        break

cap.release()
cv2.destroyAllWindows()
```

#Object Tracking in Real Time by Color

```
cap = cv2.VideoCapture(0)
while True:
    ret, frame = cap.read()
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    # Define color range for detection (blue in this case)
    lower_blue = (100, 150, 50)
    upper_blue = (140, 255, 255)

    # Create a mask
    mask = cv2.inRange(hsv, lower_blue, upper_blue)
    result = cv2.bitwise_and(frame, frame, mask=mask)
    cv2.imshow("Original", frame)
    cv2.imshow("Filtered", result)

    if cv2.waitKey(1) & 0xFF == ord("q"):
        break

cap.release()
cv2.destroyAllWindows()
```

SPEECH PROCESSING MODULE

librosa Module

Librosa is a powerful library for **audio processing, feature extraction, and music analysis**. It is widely used for **speech processing, sound classification, and music information retrieval (MIR)**.

INSTALLATION
pip install Librosa

Example
#Load an Audio File

```
# Load an audio file
audio_file = "example.wav"
y, sr = librosa.load(audio_file, sr=None)  # sr=None keeps original sampling rate

print(f"Audio Duration: {librosa.get_duration(y=y, sr=sr)} seconds")
print(f"Sampling Rate: {sr} Hz")
```

#Play an Audio File

```
import sounddevice as sd

# Play the audio
sd.play(y, sr)

# Wait until the audio is finished playing
sd.wait()
```

#Plot a Waveform

```
plt.figure(figsize=(10, 4))
librosa.display.waveshow(y, sr=sr)
plt.title("Waveform")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.show()
```

#Plot the Spectrogram

```
# Compute the Short-Time Fourier Transform (STFT)
D = librosa.amplitude_to_db(np.abs(librosa.stft(y)), ref=np.max)

plt.figure(figsize=(10, 4))
librosa.display.specshow(D, sr=sr, x_axis="time", y_axis="log")
plt.colorbar(format="%+2.0f dB")
plt.title("Spectrogram")
plt.show()
```

#Extract Mel-Frequency Cepstral Coefficients(MFCCs)

```
# Extract MFCCs
mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=13)

# Display MFCCs
plt.figure(figsize=(10, 4))
librosa.display.specshow(mfccs, sr=sr, x_axis="time")
plt.colorbar()
plt.title("MFCCs")
plt.show()
```

#Extract Chroma Features (For music Analysis)

```
# Compute chroma features
chroma = librosa.feature.chroma_stft(y=y, sr=sr)
plt.figure(figsize=(10, 4))
librosa.display.specshow(chroma, sr=sr, x_axis="time", y_axis="chroma")
plt.colorbar()
plt.title("Chroma Features")
plt.show()
```

#Beat tracking (Tempo Estimation)

```
# Estimate tempo and beats
tempo, beats = librosa.beat.beat_track(y=y, sr=sr)
print(f"Estimated Tempo: {tempo} BPM")
# Convert beats to time values
beat_times = librosa.frames_to_time(beats, sr=sr)
# Plot beats on waveform
plt.figure(figsize=(10, 4))
librosa.display.waveshow(y, sr=sr)
plt.vlines(beat_times, ymin=-1, ymax=1, color="r", linestyle="--", label="Beats")
plt.legend()
plt.title("Beat Tracking")
plt.show()
```

#Extracting Zero-Crossing Rate (ZCR)

```
# Compute zero-crossing rate
zcr = librosa.feature.zero_crossing_rate(y)

# Plot ZCR
plt.figure(figsize=(10, 4))
plt.plot(zcr[0], label="Zero-Crossing Rate")
plt.xlabel("Frames")
plt.ylabel("Rate")
plt.title("Zero-Crossing Rate")
plt.legend()
plt.show()
```

#Save Processed Audio

```
import soundfile as sf

# Save audio
sf.write("processed_audio.wav", y, sr)
print("Processed audio saved!")
```

#Noise Reduction (Remove Background Noise)

```
import noisereduce as nr

# Reduce noise
y_denoised = nr.reduce_noise(y=y, sr=sr, prop_decrease=0.8)

# Save the noise-reduced audio
sf.write("denoised_audio.wav", y_denoised, sr)
```

TEXT PROCESSING MODULE

NLTK (Natural Language Toolkit)

NLTK is a popular Python library for **Natural Language Processing.** It provides tools for text preprocessing, tokenization, stemming, lemmatization, stop word removal and more.

NLTK INSTALLATION

pip install nltk

Tokenization

Tokenization is the process of splitting text into smaller units, called tokens. These tokens can be words or sentences and are essential for text preprocessing in NLP.

Types of Tokenization

- **Word Tokenization** – Splitting text into individual words.
- **Sentence Tokenization** – Splitting text into sentences.

Example for Word Tokenization

```
import nltk
from nltk.tokenize import word_tokenize
text = "Tokenization is an important step in NLP. Let's understand it!"
nltk.download('punkt')  # Download tokenizer models
# Tokenizing words
words = word_tokenize(text)
print("Word Tokens:", words)
```

Output

Word Tokens: [‘Tokenization’, ‘is’, ‘an’, ‘important’, ‘step’, ‘in’, ‘NLP’, ‘.’, ‘Let’, “’s”, ‘understand’, ‘it’, ‘!’]

Example for Sentence Tokenization

```
from nltk.tokenize import sent_tokenize
text = "Tokenization is an important step in NLP. Let's understand it!"
# Tokenizing sentences
sentences = sent_tokenize(text)
print("Sentence Tokens:", sentences)
```

Output

Sentence Tokens: [‘Tokenization is an important step in NLP.’, “Let’s understand it!”]

Stemming

Stemming is the process of reducing words to their root or base form by removing suffixes. It helps in text normalization by treating words like "running," "runs," and "runner" as the same word: "run.“

Example

```
import nltk
from nltk.stem import PorterStemmer
# Initialize stemmer
stemmer = PorterStemmer()
# Sample words
words = ["running", "runs", "runner", "studies", "studying", "happiness", "happily"]
# Apply stemming
stemmed_words = [stemmer.stem(word) for word in words]
print("Original Words:", words)
print("Stemmed Words:", stemmed_words)
```

Output

```
Original Words: ['running', 'runs', 'runner', 'studies', 'studying', 'happiness', 'happily’]
Stemmed Words: ['run', 'run', 'runner', 'studi', 'studi', 'happi', 'happili']
```

Lemmatization

Lemmatization is the process of reducing a word to its root or dictionary form (lemma) while ensuring that the transformed word remains meaningful.

Unlike stemming, which may cut words too aggressively, lemmatization produces real words.

Example

```
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
# Download necessary resources
nltk.download('wordnet')
nltk.download('omw-1.4')
lemmatizer = WordNetLemmatizer()
words = ["running", "flies", "better", "geese", "studies", "happily"]
# Apply lemmatization
lemmatized_words = [lemmatizer.lemmatize(word) for word in words]
print("Original Words:", words)
print("Lemmatized Words:", lemmatized_words)
```

Output

Original Words: [‘running’, ‘flies’, ‘better’, ‘geese’, ‘studies’, ‘happily’]

Lemmatized Words: [‘running’, ‘fly’, ‘better’, ‘goose’, ‘study’, ‘happily’]

Parts of Speech (POS)

Parts of Speech (POS) are categories of words that define their role in a sentence. These categories help us understand the structure and meaning of sentences. Each word in a sentence belongs to one or more parts of speech.

Common Parts of Speech

- Noun (N)
- Pronoun(PRP)
- Verb (V)
- Adjective (JJ)
- Adverb (RB)
- Preposition (IN)
- Conjunction (CC)
- Interjection (UH)
- Article (DT)

Example

```
import nltk
from nltk.tokenize import word_tokenize
from nltk import pos_tag
# Sample text
text = "The quick brown fox jumps over the lazy dog."
# Tokenize and POS tagging
tokens = word_tokenize(text)
tagged_tokens = pos_tag(tokens)
print("POS Tagged Tokens:", tagged_tokens)
```

Output

```
POS Tagged Tokens: [('The', 'DT'), ('quick', 'JJ'), ('brown', 'NN'), ('fox', 'NN'), ('jumps', 'VBZ'), ('over',
'IN'), ('the', 'DT'), ('lazy', 'JJ'), ('dog', 'NN'), ('.', '.')]
```

Stopword Removal

Stopwords are common words (like "the", "is", "in", "and") that are generally removed from text data during text preprocessing because they don’t add meaningful information in many text mining and NLP tasks. Removing stopwords helps in reducing the size of the data, making it more efficient for processing.

Example

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
nltk.download('stopwords')
text = "This is an example sentence to demonstrate stopword removal."
# Tokenize the text into words
words = word_tokenize(text)
stop_words = set(stopwords.words('english'))
# Filter out stopwords
filtered_words = [word for word in words if word.lower() not in stop_words]
print("Original Words:", words)
print("Filtered Words:", filtered_words)
```

Output

```
Original Words: ['This', 'is', 'an', 'example', 'sentence', 'to', 'demonstrate', 'stopword', 'removal', '.’]
Filtered Words: ['example', 'sentence', 'demonstrate', 'stopword', 'removal', '.']
```