Python Main Data Types

boolean = True / False

integer = 10

float = 10.01

string = “123abc”

None: absence of value.

list = [value1, value2, ...]

dict = { key1:value1, key2:value2, ...}

tuple: Ordered, immutable collection (e.g., (1, 2, 3))

range: Sequence of numbers (e.g., range(0, 10))

set: Unordered, unique elements (e.g., {1, 2, 3})

Built-in Functions

print(x, sep='y') → Prints x objects separated by y

input(s) → Prints s and waits for user input

len(x) → Returns the length of x (string, list, or dict)

min(L) → Returns the minimum value in a list

max(L) → Returns the maximum value in a list

sum(L) → Returns the sum of the values in a list

range(n1, n2, n) → Returns a sequence of numbers from n1 to n2 in steps of n

abs(n) → Returns the absolute value of n

round(n1, n) → Rounds n1 to n decimal places

type(x) → Returns the type of x (e.g., string, float)

Type casting

str(x) → Converts x to a string

list(x) → Converts x to a list

int(x) → Converts x to an integer

float(x) → Converts x to a float

help(s) → Displays help documentation for s

map(function, L) → Applies function to each value in list

Conditional Statements if, elif, else:

```
if condition:
    # Code to execute if condition is True
elif other_condition:
    # Code for another condition
else:
    # Code if none of the conditions are True
```

try, except, finally:

```
try:
    # Code to try
except Exception as e:
    # Handle the exception
finally:
    # Code that always executes
```

Loops

For Loop:

```
for item in iterable:
    # Code to execute for each item
```

While Loop:

```
while condition:
    # Code to execute while condition is True
```

Loop Control Statements

break → Exits the loop prematurely

continue → Skips the current iteration and continues with the next

pass → Does nothing (placeholder statement)

Functions

Defining a Function:

```
def funct_name(params):
    # Code to execute
    return value
```

Lambda Functions:

```
square = lambda x: x**2
print(square(5))  # Output: 25
```

Modules

Importing a Module:

```
import module_name
```

Importing Specific Functions:

```
from module_name import function_name
```

Using a Function from a Module:

```
module_name.function_name()
```

Reading and Writing Files

Writing to a File:

```
with open("file.txt", "w") as file:
    file.write("Hello, File!")
```

Reading from a File:

```
with open("file.txt", "r") as file:
    content = file.read()
    print(content)
```

Common String Operations

Creating Strings

Single Quotes:

```
s = 'Hello'
```

Double Quotes:

```
s = "Hello"
```

Multiline Strings:

```
s = """This is
a multiline
string."""
```

Raw Strings:

```
s = r"\n"  # Escape sequences are ignored
```

Accessing Strings

Indexing:

```
s[0]  # First character
```

Slicing:

```
s[1:4]  # Characters from index 1 to 3
```

Negative Indexing:

```
s[-1]  # Last character
```

String Methods

Case Conversion:

```
s.upper()     # Convert to uppercase
s.lower()     # Convert to lowercase
s.capitalize() # Capitalize first letter
s.title()     # Title case
s.swapcase()  # Swap case
```

Trimming:

```
s.strip()  # Remove leading and trailing whitespace
s.lstrip()  # Remove leading whitespace
s.rstrip()  # Remove trailing whitespace
```

Finding & Replacing:

```
s.find("substring")   # Index of first occurrence
s.rfind("substring")   # Index of last occurrence
s.replace("old", "new")  # Replace all occurrences
```

Testing:

```
s.isalpha()    # True if all characters are alphabetic
s.isdigit()    # True if all characters are digits
s.isalnum()    # True if all characters are alphanumeric
s.isspace()    # True if all characters are whitespace
s.startswith("prefix")  # True if string starts with prefix
s.endswith("suffix")   # True if string ends with suffix
```

String Operations

Concatenation:

```
s1 + s2  # Combine strings
```

Repetition:

```
s * 3  # Repeat string 3 times
```

Membership Test:

```
"sub" in s  # Check if substring exists
```

Splitting and Joining

Splitting Strings:

```
s = "a,b,c"
s.split(",")  # ['a', 'b', 'c']
```

Joining Strings:

```
lst = ["a", "b", "c"]
"-".join(lst)  # 'a-b-c'
```

String Formatting

Using f-Strings:

```
name = "Alice"
age = 25
print(f"{name} is {age} years old")
```

Reversing Strings

```
s = "Hello"
reversed_s = s[::-1]  # 'olleH'
```

Regular Expressions

. : Matches any character except a newline.

\b : word boundary

\d : Matches any digit

\d{3}: exact three digit num

[abc] : Matches any character in the set (e.g., a, b, or c).

[A-Z]: an uppercase letter

[a-z]+: one or more lowercase letters following

[0-9] : Matches any digit.

[aeiouAEIOU]: any vowel (uppercase or lowercase)

\w : Matches any word character

\w+: one or more word characters (letters, digits, or underscores)

\\Z: Matches the end of the string.

\\A: Matches the beginning of the string.

(x|y): Matches either x or y.

Search for Pattern:

```
import re
s = "Alice is 25 years old"
match = re.search(r"\d+", s)
print(match.group() if match else "No match")
```

Find All Matches:

```
matches = re.findall(r"[A-Z][a-z]+", "Alice and Bob")
print(matches)  # ['Alice', 'Bob']
```

Common List Operations

Creating Lists

Empty List: L = []

List with Elements: L = [1, 2, 3, "Hello"]

Using range: L = list(range(5)) → [0, 1, 2, 3, 4]

Accessing Elements

Indexing: L[0] → First element

Slicing: L[1:3] → Elements at index 1 to 2

Negative Indexing: L[-1] → Last element

Adding Elements

L.append(x) → Add x to the end of the list

L.insert(i, x) → Insert x at index i

L.extend([x, y]) → Add multiple elements

Removing Elements

L.pop(i) → Remove and return element at index i

L.remove(x) → Remove first occurrence of x

del L[i] → Delete element at index i

L.clear() → Remove all elements

List Methods

L.index(x) → Index of first occurrence of x

L.count(x) → Count occurrences of x
L.sort() → Sort list in ascending order
L.reverse() → Reverse list order
L.copy() → Create a shallow copy of the list

List Operations

Concatenation: L1 + L2 → Combine two lists
Repetition: L * 3 → Repeat list 3 times
Membership Test: x in L → Check if x exists in the list

List Comprehension

Example 1: [x**2 for x in range(5)] → [0, 1, 4, 9, 16]
Example 2: [x for x in L if x % 2 == 0] → Filter even numbers

Iteration:

```
L = [1, 2, 3, 4]
for x in L:
   print(x)
```

Nested Lists

Creating: L = [[1, 2], [3, 4]]
Accessing Elements: L[0][1] → 2

Pandas Basics

1. Importing Pandas

```
import pandas as pd
```

2. Creating DataFrames

From a Dictionary:

```
data = {"Name": ["Alice", "Bob", "Charlie"], "Age": [25, 30, 35]}
df = pd.DataFrame(data)
print(df)
```

From a List of Lists:

```
data = [["Alice", 25], ["Bob", 30], ["Charlie", 35]]
df = pd.DataFrame(data, columns=["Name", "Age"])
print(df)
```

3. Reading and Writing Data

CSV:

```
df = pd.read_csv("file.csv")  # Reading
df.to_csv("output.csv", index=False)  # Writing
```

Excel:

```
df = pd.read_excel("file.xlsx")  # Reading
df.to_excel("output.xlsx", index=False)  # Writing
```

4. Inspecting Data

df.head(n) → View the first n rows (default is 5)
df.tail(n) → View the last n rows
df.info() → Overview of DataFrame
df.describe() → Summary statistics for numerical columns
df.shape → Dimensions of DataFrame (rows, columns)
df.columns → Column names

5. Selecting Data

By Columns:

```
df["Name"]  # Select single column
df[["Name", "Age"]]  # Select multiple columns
```

By Rows (Indexing):

```
df.iloc[0]  # Select first row by index
df.loc[0]  # Select first row by label
```

By Condition:

```
df[df["Age"] > 30]  # Filter rows where Age > 30
```

6. Adding and Removing Data

Remove duplicate rows:

```
df.drop_duplicates()
```

Rename columns:

```
df.rename(columns={"old_name": "new_name"})
```

Apply a function element-wise:

```
df.apply(func)
```

Add a New Column:

```
df["City"] = ["New York", "Los Angeles", "Chicago"]
```

Delete a Column:

```
df.drop(columns=["City"], inplace=True)
```

Add a Row:

```
df.loc[len(df)] = ["David", 40]
```

Delete a Row:

```
df.drop(index=0, inplace=True)  # Delete first row
```

7. Sorting Data

Sort by Column:

```
df.sort_values("Age", ascending=True, inplace=True)
```

Sort by Index:

```
df.sort_index(inplace=True)
```

8. Aggregating Data

Basic Statistics:

```
df["Age"].mean()  # Average
df["Age"].sum()  # Sum
df["Age"].min()  # Minimum
df["Age"].max()  # Maximu
```

Group By:

```
df.groupby("City")["Age"].mean()  # Average age by City
```

9. Handling Missing Data

Check for Missing Values:

```
df.isnull()  # Returns a DataFrame of True/False
df.isnull().sum()  # Count missing values per column
```

Fill Missing Values:

```
df.fillna(0, inplace=True)
```

Drop Missing Values:

```
df.dropna(inplace=True)
```

10. Exporting Processed Data

Save the cleaned and processed DataFrame:

```
df.to_csv("processed_data.csv", index=False)
```

Matplotlib Basics

1. Importing Matplotlib

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

2. Basic Line Plot

```
x = np.linspace(0, 10, 100)
y = np.sin(x)
plt.plot(x, y, label='Sine Wave')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Basic Line Plot')
plt.legend()
plt.show()
```

3. Scatter Plot

```
x = np.random.rand(50)
y = np.random.rand(50)
plt.scatter(x, y, color='red', marker='o')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Scatter Plot')
plt.show()
```

4. Bar Plot

```
categories = ['A', 'B', 'C', 'D']
values = [3, 7, 8, 5]
plt.bar(categories, values, color='blue')
plt.xlabel('Categories')
plt.ylabel('Values')
plt.title('Bar Plot')
plt.show()
```

5. Histogram

```
data = np.random.randn(1000)
plt.hist(data, bins=30, color='green', edgecolor='black')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Histogram')
plt.show()
```

6. Pie Chart

```
sizes = [15, 30, 45, 10]
labels = ['Category A', 'Category B', 'Category C', 'Category D']
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140)
plt.title('Pie Chart')
plt.show()
```

7. Subplots

```
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)
plt.subplot(2, 1, 1)  # (rows, cols, index)
plt.plot(x, y1, label='Sine', color='blue')
plt.title('Sine Wave')
plt.subplot(2, 1, 2)
plt.plot(x, y2, label='Cosine', color='orange')
plt.title('Cosine Wave')
plt.tight_layout()
plt.show()
```

8. Customization Options

Change Line Style and Width:

```
plt.plot(x, y, linestyle='--', linewidth=2)
```

Change Marker Style:

```
plt.plot(x, y, marker='o', markersize=5)
```

Set Gridlines:

```
plt.grid(True)
```

Set Axis Limits:

```
plt.xlim(0, 5)
plt.ylim(-1, 1)
```

9. Saving the Plot

```
plt.savefig('plot.png', dpi=300, bbox_inches='tight')
```

Extras:

pip: Python package manager. install packages

Python Packages

Requests
A simple library for making HTTP requests.

Beautiful Soup
A library for web scraping and parsing HTML/XML.

Selenium
A library for browser automation.

PRAW (Python Reddit API Wrapper)
A library for interacting with Reddit's API.