In the name of Allah

Hello Everybody

Let’s get started!

Library

A library, also known as a module or package, is a collection of pre-written code and functions that can be imported and used in your Python programs.

Import

Before you can use a library, you need to ensure it's installed on your system. Many libraries are not part of the Python standard library, so you may need to install them separately. After installing the library, you need to import it into your Python script or program using the import statement.

```
pip install numpy
```

```
import numpy
# or
import numpy as np  # Using an alias
(common for libraries with long names)
```

Built-in Libraries

Python has a vast ecosystem of simple and common libraries that cover a wide range of tasks

math

The math module provides a wide range of mathematical functions, including basic arithmetic, trigonometry, and more advanced mathematical operations.

```
import math
```

random

The random module provides functions for generating random numbers, making it useful for tasks like simulation, games, and cryptography.

```
import random
```

Example 1

Generate a random integer between 1 and 100 (inclusive)

```
import random

random_integer = random.randint(1, 100)
print(f"Random integer: {random_integer}")
```

Example 2

Generate a random floating-point number between 0 and 1

```
import random

random_float = random.random()
print(f"Random float: {random_float}")
```

Example 3

Generate a random floating-point number within a specified range

```
import random

random_range = random.uniform(2.0, 5.0)
print(f"Random float within range:
{random_range}")
```

Example 4

Shuffle a list randomly

```
import random

my_list = [1, 2, 3, 4, 5]
random.shuffle(my_list)
print(f"Shuffled list: {my_list}")
```

Example 5

Randomly choose an item from a list

```
import random

choices = ['apple', 'banana', 'cherry',
'date', 'fig']
random_choice = random.choice(choices)
print(f"Random choice: {random_choice}")
```

Example 6

Randomly sample multiple items from a list without replacement

```
import random

random_sample = random.sample(choices, 2)
# Choose 2 items
print(f"Random sample: {random_sample}")
```

datetime

The datetime module is a built-in library for working with dates and times. It provides classes and functions for parsing, formatting, and performing calculations with dates and times.

```
import datetime
```

Example 1

Get the current date and time

```
import datetime

current_datetime = datetime.datetime.now()
print("Current date and time:",
current_datetime)
```

Example 2

Get the current date

```
import datetime

current_date = datetime.date.today()
print("Current date:", current_date)
```

Example 3

Create a specific date

```
import datetime

specific_date = datetime.date(2023, 9, 15)
print("Specific date:", specific_date)
```

Example 4

Format a date as a string

```
import datetime

specific_date = datetime.date(2023, 9, 15)
formatted_date =
specific_date.strftime("%Y-%m-%d")
print("Formatted date:", formatted_date)
```

Example 5

Parse a date from a string

```
import datetime

date_string = "2023-09-15"
parsed_date =
datetime.datetime.strptime(date_string,
"%Y-%m-%d").date()
print("Parsed date:", parsed_date)
```

Example 6

Calculate the difference between two dates

```
import datetime

delta = specific_date - current_date
print("Time difference:", delta)
```

Example 7

Add or subtract days from a date

```
import datetime

new_date = specific_date +
datetime.timedelta(days=7)
print("New date after adding 7 days:",
new_date)
```

os

The os module provides a way to interact with the operating system, including functions for file and directory manipulation, environment variables, and more.

```
import os
```

Example 1

List files and directories in the current directory

```
import os

directory_contents = os.listdir()
print("Directory contents:",
directory_contents)
```

Example 2

Check if a file or directory exists

```
import os

file_exists = os.path.exists("example.txt")
print("Does example.txt exist?",
file_exists)
```

Example 3

Create a new directory

```
import os

new_directory = "my_new_directory"
os.mkdir(new_directory)
print("Created directory:", new_directory)
```

json

The json module is used for encoding and decoding JSON (JavaScript Object Notation) data. It's commonly used for working with data in JSON format, which is a popular data interchange format.

```
import json
```

Example 1

Serialize and Deserialize JSON

```
import json

# Create a Python dictionary
data = {
    "name": "John Doe",
    "age": 30,
    "city": "New York"
}
# Serialize (encode) the dictionary to JSON
json_data = json.dumps(data)
print("Serialized JSON data: ")
print(json_data)

# Deserialize (decode) JSON data to a
Python dictionary
decoded_data = json.loads(json_data)
print("\nDecoded Python dictionary: ")
print(decoded_data)
```

Example 2

Working with JSON files

```
import json

# Write JSON data to a file
with open("data.json", "w") as json_file:
    json.dump(data, json_file)

# Read JSON data from a file
with open("data.json", "r") as json_file:
    loaded_data = json.load(json_file)
print("\nLoaded data from JSON file:")
print(loaded_data)
```

CSV

The csv module allows you to work with CSV (Comma-Separated Values) files, making it easy to read and write structured data stored in text files.

```
import csv
```

Example 1

Reading from a CSV File

```
import csv

# Open the CSV file for reading
with open('data.csv', mode='r') as
csv_file:
    # Create a CSV reader object
    csv_reader = csv.reader(csv_file)

    # Iterate through each row in the CSV
file
    for row in csv_reader:
        name, age, location = row
        print(f"Name: {name}, Age: {age},
Location: {location}")
```

Example 2

Writing to a CSV File

```
import csv
# Data to write to the CSV file
data_to_write = [
    ['Eve', 28, 'San Francisco'],
    ['Frank', 35, 'Seattle']
]
# Open a new CSV file for writing
with open('new_data.csv', mode='w',
newline='') as csv_file:
    # Create a CSV writer object
    csv_writer = csv.writer(csv_file)
    # Write the header row
    csv_writer.writerow(['Name', 'Age',
'Location'])
    # Write the data rows
    csv_writer.writerows(data_to_write)
```

Example 3

Reading from a CSV File Using DictReader

```
import csv

# Open the CSV file for reading
with open('data_dictwriter.csv', mode='r')
as csv_file:
    # Create a DictReader object
    csv_reader = csv.DictReader(csv_file)

    # Iterate through each row in the CSV
file as a dictionary
    for row in csv_reader:
        name = row['Name']
        age = int(row['Age'])
        location = row['Location']
        print(f"Name: {name}, Age: {age},
Location: {location}")
```

Example 4

Writing to a CSV File Using DictWriter

```
import csv
# Data to write to the CSV file
data_to_write = [
    {'Name': 'Alice', 'Age': 25, 'Location':
'New York'}
]
# Specify the fieldnames (column names)
fieldnames = ['Name', 'Age', 'Location']
# Open a new CSV file for writing
with open('data_dictwriter.csv', mode='w',
newline='') as csv_file:
    # Create a DictWriter object
    csv_writer = csv.DictWriter(csv_file,
fieldnames=fieldnames)
    # Write the header row
    csv_writer.writeheader()
    # Write the data rows
    csv_writer.writerows(data_to_write)
```

That’s all for today!