Built-in Modules

Built-in modules in Python are pre-existing libraries that provide a set of functions, classes, and constants designed to perform specific tasks. These modules are available in the Python standard library and cover a wide range of functionalities, making them readily accessible for developers without the need for external installations.

1- Datetime

The datetime module in Python provides classes for manipulating dates and times. It allows you to work with dates, times, time intervals, and perform various operations such as formatting, arithmetic, and comparisons.

In [1]:
```
# example
import datetime

current_datetime = datetime.datetime.now()
print("Current datetime:", current_datetime)

# Output: Current datetime: 2023-08-15 12:34:56.789012
```

```
Current datetime: 2023-08-15 10:49:50.803025
```

In [4]:
```
from datetime import datetime
my_dt = datetime.now()
my_dt.year # prints current year, 2020
# prints current month, 3
```

Out[4]:
```
2023
```

In [5]:
```
my_dt.month
```

Out[5]:
```
8
```

1.1- strftime

strftime in the datetime module is a method used to format a datetime object into a string representation based on a specified format string.

You can use a special method called strftime to access particular parts of the datetime object by passing special format specifiers.

```
In [7]: now = datetime.now() # current date and time
        year = now.strftime("%Y")
        year

Out[7]: '2023'
```

```
In [8]: month = now.strftime("%m")
        month

Out[8]: '08'
```

```
In [9]: day = now.strftime("%d")
        print("day: ", day)

        day:  15
```

```
In [10]: time = now.strftime("%H:%M:%S")
         print("time: ", time)

         time:  10:54:15
```

```
In [12]: date_time = now.strftime("%m/%d/%Y, %H:%M:%S")
         print("date and time:", date_time)

         date and time: 08/15/2023, 10:54:15
```

1.2- strptime()

Function to parse a string into a datetime object. strptime() is a method in the datetime module used to parse a string representing a date and time according to a specified format and convert it into a datetime object. It's commonly used to convert user-provided date strings into a structured datetime representation that can be manipulated and processed programmatically.

```
In [14]: # example
         date_string = '21 June, 2018'
         type(date_string)

Out[14]: str
```

```
In [16]: date_object = datetime.strptime(date_string, "%d %B, %Y")
         date_object

Out[16]: datetime.datetime(2018, 6, 21, 0, 0)
```

```
In [17]: type(date_object)

Out[17]: datetime.datetime
```

```
In [18]: # example
         from datetime import datetime

         date_string = "2023-08-15"
         formatted_date = datetime.strptime(date_string, "%Y-%m-%d")
         print(formatted_date)

         2023-08-15 00:00:00
```

2- OS Module

The os module in Python provides functions for interacting with the operating system, including file and directory operations.

In [19]:
```
# example
import os

current_directory = os.getcwd()
print("Current directory:", current_directory)
```

```
Current directory: C:\Users\EURO TEC COMPUTERS
```

In [20]:
```
# example how to make directory
import os

new_directory = "my_new_directory"
os.makedirs(new_directory)

# A directory named "my_new_directory" will be created in the current working directory.
```

3- Introspection (Inspect) Module

The inspect module in Python provides functions to access and retrieve information about live objects, such as modules, classes, functions, and methods. It's commonly used for introspection, which involves examining the attributes and structure of these objects programmatically.

In [21]:
```
# example
import inspect

def greet(name):
    return f"Hello, {name}!"

source_code = inspect.getsource(greet)
print(source_code)
```

```
def greet(name):
    return f"Hello, {name}!"
```

In [22]:
```python
# example
import inspect

def calculate(a, b, operation="add"):
    if operation == "add":
        return a + b
    elif operation == "subtract":
        return a - b

argspec = inspect.getfullargspec(calculate)
print("Function arguments:", argspec.args)
print("Default values:", argspec.defaults)
```

```
Function arguments: ['a', 'b', 'operation']
Default values: ('add',)
```

3.1- Functions

inspect.isfunction(object): Check if an object is a function.

inspect.getsource(object): Get the source code of a function.

3.2- Methods

inspect.ismethod(object): Check if an object is a method.

3.3- Classes

inspect.isclass(object): Check if an object is a class. inspect.getmembers(object, inspect.isclass): Get the classes defined in a module or object.

3.4- Modules

inspect.ismodule(object): Check if an object is a module.

In [23]:
```python
import inspect

def greet(name):
    return f"Hello, {name}!"

class MyClass:
    def my_method(self):
        return "This is a method."

def main():
    print("Is 'greet' a function?", inspect.isfunction(greet))
    print("Is 'my_method' a method?", inspect.ismethod(MyClass.my_method))
    print("Is 'MyClass' a class?", inspect.isclass(MyClass))
    print("Is 'inspect' a module?", inspect.ismodule(inspect))

    module_classes = inspect.getmembers(inspect, inspect.isclass)
    print("Classes in 'inspect' module:", [cls[0] for cls in module_classes])

if __name__ == "__main__":
    main()
```

```
Is 'greet' a function? True
Is 'my_method' a method? False
Is 'MyClass' a class? True
Is 'inspect' a module? True
Classes in 'inspect' module: ['ArgInfo', 'ArgSpec', 'Arguments', 'Attribute', 'BlockFinder', 'BoundArgument
s', 'ClassFoundException', 'ClosureVars', 'EndOfBlock', 'FrameInfo', 'FullArgSpec', 'OrderedDict', 'Paramete
r', 'Signature', 'Traceback', '_ClassFinder', '_ClassMethodWrapper', '_MethodWrapper', '_ParameterKind', '_W
rapperDescriptor', '_empty', '_void', 'attrgetter']
```

3.5- function or method signature

A function or method signature is like a blueprint that describes how a function or method should be called. It includes the names of parameters the function accepts, their data types, and whether they have default values.

In [24]:
```python
# example
import inspect

def greet(name: str, greeting: str = "Hello") -> str:
    return f"{greeting}, {name}!"

# Get the signature of the 'greet' function
signature = inspect.signature(greet)

# Print the parameters and their details
for parameter_name, parameter in signature.parameters.items():
    print(f"Parameter: {parameter_name}")
    print(f"   Default Value: {parameter.default}")
    print(f"   Annotation: {parameter.annotation}")
    print(f"   Kind: {parameter.kind}")
```

```
Parameter: name
   Default Value: <class 'inspect._empty'>
   Annotation: <class 'str'>
   Kind: POSITIONAL_OR_KEYWORD
Parameter: greeting
   Default Value: Hello
   Annotation: <class 'str'>
   Kind: POSITIONAL_OR_KEYWORD
```

4- Sqlite Module

The sqlite module in Python provides a lightweight, built-in way to interact with SQLite databases. It allows you to create, query, and modify SQLite databases.

In [1]:
```python
import sqlite3

# Connect to a database (creates a new one if not exists)
conn = sqlite3.connect('mydatabase.db')

# Create a cursor
cursor = conn.cursor()

# Create a table
cursor.execute('''CREATE TABLE IF NOT EXISTS users
                  (id INTEGER PRIMARY KEY, name TEXT, age INTEGER)''')

# Insert data
cursor.execute("INSERT INTO users (name, age) VALUES (?, ?)", ('Uzair', 30))
cursor.execute("INSERT INTO users (name, age) VALUES (?, ?)", ('Umair', 25))

# Commit changes
conn.commit()

# Query data
cursor.execute("SELECT * FROM users")
rows = cursor.fetchall()

for row in rows:
    print(row)

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

```
(1, 'Uzair', 30)
(2, 'Umair', 25)
```

Cursor

Why create cursor?

In SQLite and many other database systems, a cursor is used to interact with the database and execute SQL commands. It serves as a handle that allows you to navigate and manipulate the data within the database

In [2]:
```python
import sqlite3

# Connect to an existing database
conn = sqlite3.connect('mydatabase.db')

# Create a cursor
cursor = conn.cursor()

# Execute a SELECT query
cursor.execute('''SELECT * FROM users''')

# Fetch all the rows returned by the query
rows = cursor.fetchall()

# Print the fetched rows
for row in rows:
    print(row)

# Close the cursor and the connection
cursor.close()
conn.close()
```

```
(1, 'Uzair', 30)
(2, 'Umair', 25)
```

5- CSV Module

The csv module in Python provides functionality to work with Comma-Separated Values (CSV) files, which are commonly used for storing tabular data. It allows you to read from and write to CSV files easily.

5.1- reading from a CSV

In [4]:
```python
import csv

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

    # Iterate through each row in the CSV file
    for row in csvreader:
        print(', '.join(row))  # Print each row as a comma-separated strin#
# tit is   just example
```

```
6.4, 2.8, 5.6, 2.2, Iris-virginica
6.3, 2.8, 5.1, 1.5, Iris-virginica
6.1, 2.6, 5.6, 1.4, Iris-virginica
7.7, 3.0, 6.1, 2.3, Iris-virginica
6.3, 3.4, 5.6, 2.4, Iris-virginica
6.4, 3.1, 5.5, 1.8, Iris-virginica
6.0, 3.0, 4.8, 1.8, Iris-virginica
6.9, 3.1, 5.4, 2.1, Iris-virginica
6.7, 3.1, 5.6, 2.4, Iris-virginica
6.9, 3.1, 5.1, 2.3, Iris-virginica
5.8, 2.7, 5.1, 1.9, Iris-virginica
6.8, 3.2, 5.9, 2.3, Iris-virginica
6.7, 3.3, 5.7, 2.5, Iris-virginica
6.7, 3.0, 5.2, 2.3, Iris-virginica
6.3, 2.5, 5.0, 1.9, Iris-virginica
6.5, 3.0, 5.2, 2.0, Iris-virginica
6.2, 3.4, 5.4, 2.3, Iris-virginica
5.9, 3.0, 5.1, 1.8, Iris-virginica
```

5.2- writing to a CSV file

In [7]:
```python
import csv

# Data to be written to the CSV file
data = [
    ['Name', 'Age', 'Country'],
    ['Uzair', 28, 'pakistan'],
    ['faisal', 32, 'Canada'],
    ['zeeshan', 23, 'USA']
]

# Open the CSV file for writing
with open('output.csv', 'w', newline='') as csvfile:
    # Create a CSV writer object
    csvwriter = csv.writer(csvfile)

    # Write data to the CSV file
    csvwriter.writerows(data)
```

6- The Subprocess Module

The subprocess module in Python provides a way to spawn new processes, connect to their input/output/error pipes, and obtain their return codes. It is a powerful module for working with external processes, running system commands, and managing their interactions from within your Python script

In [17]:
```python


import subprocess

# Run the 'ls' command and capture its output
result = subprocess.run(['ls', '-l'], stdout=subprocess.PIPE, text=True)

# Print the captured output
print(result.stdout)
# just example for understanding no such dfile exist in my current directory 'ls', 'l'
# Note: specify the file
```

```
In [ ]: import subprocess

        # Run the 'ls' command and capture its return code
        result = subprocess.run(['ls', '-l'], stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True)

        # Check the return code and print output if an error occurred
        if result.returncode != 0:
            print("Error:", result.stderr)
        else:
            print(result.stdout)
        # just example for understanding no such dfile exist in my current directory 'ls', 'l'
        # Note: specify the file
```

```
In [20]: # Subprocess module exercise
         import subprocess
         import os

         # Replace 'chrome.exe' with the appropriate browser executable name if needed
         browser_executable = 'chrome.exe'

         try:
             # Find the path to the browser executable using the 'where' command on Windows
             browser_path = subprocess.check_output(['where', browser_executable], text=True).strip()

             # Launch the browser using the subprocess module
             subprocess.run([browser_path])

         except subprocess.CalledProcessError:
             print(f"Error: {browser_executable} not found or could not be opened.")
         except FileNotFoundError:
             print(f"Error: {browser_executable} not found in system PATH.")
```

```
Error: chrome.exe not found or could not be opened.
```

7 -The requests module

The requests module in Python is a powerful library used for making HTTP requests to web services and retrieving data from web pages or APIs. It simplifies the process of working with HTTP and provides a more user-friendly interface compared to the built-in urllib module.

In [21]:
```python
import requests

# URL of the web page you want to retrieve
url = 'https://github.com/'

# Make a GET request to the URL
response = requests.get(url)

# Check if the request was successful (status code 200)
if response.status_code == 200:
    # Print the content of the web page
    print(response.text)
else:
    print(f"Request failed with status code: {response.status_code}")
```

```





<!DOCTYPE html>
<html lang="en"   data-a11y-animated-images="system" data-a11y-link-underlines="false">
  <head>
    <meta charset="utf-8">
  <link rel="dns-prefetch" href="https://github.githubassets.com">
  <link rel="dns-prefetch" href="https://avatars.githubusercontent.com">
  <link rel="dns-prefetch" href="https://github-cloud.s3.amazonaws.com">
  <link rel="dns-prefetch" href="https://user-images.githubusercontent.com/">
  <link rel="preconnect" href="https://github.githubassets.com" crossorigin>
  <link rel="preconnect" href="https://avatars.githubusercontent.com">
```

```
In [23]: # example
         import requests
         x = requests.get("https://github.com/")
         print(x)

         print(x.status_code)
         20
```

```
<Response [200]>
200
```

```
Out[23]: 20
```

```
In [24]: print(x.text)
         print(x.headers)
         x = requests.head("https://github.com/")
         print(x.hearders)
         print(x.headers['Content-Type'])
```

```




<!DOCTYPE html>
<html lang="en"   data-a11y-animated-images="system" data-a11y-link-underlines="false">
  <head>
    <meta charset="utf-8">
  <link rel="dns-prefetch" href="https://github.githubassets.com">
  <link rel="dns-prefetch" href="https://avatars.githubusercontent.com">
  <link rel="dns-prefetch" href="https://github-cloud.s3.amazonaws.com">
  <link rel="dns-prefetch" href="https://user-images.githubusercontent.com/">
  <link rel="preconnect" href="https://github.githubassets.com" crossorigin>
  <link rel="preconnect" href="https://avatars.githubusercontent.com">



  <link crossorigin="anonymous" media="all" rel="stylesheet" href="https://github.githubassets.com/assets/l
```

The output of above code is different according to the url request of specific website

In [27]:
```
# example
import requests

# URLs to inspect
urls = [
    'http://www.google.com',
    'https://api.github.com/users/(create_your_profile)'
]

for url in urls:
    response = requests.get(url)

    # Print URL and headers
    print(f"Headers for URL: {url}")
    print(response.headers)
    print("=" * 50)  # Separator between URLs
```

```
Headers for URL: http://www.google.com (http://www.google.com)
{'Date': 'Wed, 16 Aug 2023 08:01:22 GMT', 'Expires': '-1', 'Cache-Control': 'private, max-age=0', 'Content-T
ype': 'text/html; charset=ISO-8859-1', 'Content-Security-Policy-Report-Only': "object-src 'none';base-uri 's
elf';script-src 'nonce-cg1cLyapQb9hJ_P_5qCufQ' 'strict-dynamic' 'report-sample' 'unsafe-eval' 'unsafe-inlin
e' https: http:;report-uri (http:;report-uri) https://csp.withgoogle.com/csp/gws/other-hp", (https://csp.wit
hgoogle.com/csp/gws/other-hp",) 'P3P': 'CP="This is not a P3P policy! See g.co/p3phelp for more info."', 'Co
ntent-Encoding': 'gzip', 'Server': 'gws', 'Content-Length': '8235', 'X-XSS-Protection': '0', 'X-Frame-Option
s': 'SAMEORIGIN', 'Set-Cookie': '1P_JAR=2023-08-16-08; expires=Fri, 15-Sep-2023 08:01:22 GMT; path=/; domain
=.google.com; Secure, AEC=Ad49MVE-wa_AbrgxSV2eotdXBXDXqzOxuL1mTLCuMuaZhm-Qb0Qmp2jKf8U; expires=Mon, 12-Feb-2
024 08:01:22 GMT; path=/; domain=.google.com; Secure; HttpOnly; SameSite=lax, NID=511=dN67_sWbEh8zrTZIeCxS4Z
VSYujk7jy7ddO0CXpaRAHu3rSPk9mI5qqUpGttitwcSmG_6x3sIsSShfx9HE-eZYJu_2Nq1HL6Hl63djtELWYq5FzSGHsFU9NjSoQ9aoWgkR
CHD09UsUU3Qt7Lupa4Pm-3Y226LpAjgS3xKNC6-JY; expires=Thu, 15-Feb-2024 08:01:22 GMT; path=/; domain=.google.co
m; HttpOnly'}
==================================================
Headers for URL: https://api.github.com/users/(create_your_profile) (https://api.github.com/users/(create_yo
ur_profile))
{'Server': 'GitHub.com', 'Date': 'Wed, 16 Aug 2023 08:01:23 GMT', 'Content-Type': 'application/json; charset
=utf-8', 'X-GitHub-Media-Type': 'github.v3; format=json', 'x-github-api-version-selected': '2022-11-28', 'Ac
cess-Control-Expose-Headers': 'ETag, Link, Location, Retry-After, X-GitHub-OTP, X-RateLimit-Limit, X-RateLim
it-Remaining, X-RateLimit-Used, X-RateLimit-Resource, X-RateLimit-Reset, X-OAuth-Scopes, X-Accepted-OAuth-Sc
opes, X-Poll-Interval, X-GitHub-Media-Type, X-GitHub-SSO, X-GitHub-Request-Id, Deprecation, Sunset', 'Access
-Control-Allow-Origin': '*', 'Strict-Transport-Security': 'max-age=31536000; includeSubdomains; preload', 'X
-Frame-Options': 'deny', 'X-Content-Type-Options': 'nosniff', 'X-XSS-Protection': '0', 'Referrer-Policy': 'o
rigin-when-cross-origin, strict-origin-when-cross-origin', 'Content-Security-Policy': "default-src 'none'",
'Vary': 'Accept-Encoding, Accept, X-Requested-With', 'Content-Encoding': 'gzip', 'X-RateLimit-Limit': '60',
'X-RateLimit-Remaining': '59', 'X-RateLimit-Reset': '1692176483', 'X-RateLimit-Resource': 'core', 'X-RateLim
it-Used': '1', 'Content-Length': '105', 'X-GitHub-Request-Id': '9121:3BEF73:4A77A1:53B656:64DC8253'}
==================================================
```

```
In [ ]:
```