1- LXML Module

LXML is a Python library used for processing XML and HTML documents. It provides a fast and efficient way to parse and manipulate XML/HTML data.

LXML is a powerful library for working with XML and HTML data in Python.

1.1- Parsing XML

In [2]:
```
# This example demonstrates how to parse an XML document using LXML
from lxml import etree

xml_data = '''
<root>
    <element>Value 1</element>
    <element>Value 2</element>
</root>
'''

root = etree.fromstring(xml_data)
for element in root.findall('element'):
    print(element.text)
```

```
Value 1
Value 2
```

1.2- Parsing HTML

LXML can also be used to parse HTML documents.

In [3]:
```
from lxml import html

html_data = '''
<html>
    <body>
        <h1>Heading</h1>
        <p>Paragraph 1</p>
        <p>Paragraph 2</p>
    </body>
</html>
'''

tree = html.fromstring(html_data)
headings = tree.xpath('//h1/text()')
paragraphs = tree.xpath('//p/text()')

print("Headings:", headings)
print("Paragraphs:", paragraphs)
```

```
Headings: ['Heading']
Paragraphs: ['Paragraph 1', 'Paragraph 2']
```

1.3- Modifying XML

LXML can be used to modify existing XML data.

In [4]:

```
from lxml import etree

xml_data = '''
<root>
    <element>Value 1</element>
    <element>Value 2</element>
</root>
'''

root = etree.fromstring(xml_data)
for element in root.findall('element'):
    element.text = element.text.upper()

modified_xml = etree.tostring(root)
print(modified_xml.decode())
```

```
<root>
    <element>VALUE 1</element>
    <element>VALUE 2</element>
</root>
```

In [5]:

```python
# Example how to extract code from url and retrieve information
import requests
import xml.etree.ElementTree as ET

# URL of the XML file
url = "https://www.w3schools.com/xml/simple.xml"

# Send a GET request to fetch the XML content
response = requests.get(url)
xml_content = response.content

# Parse the XML content
tree = ET.fromstring(xml_content)

# Create a dictionary to store food information
food_info = {}

# Iterate through food elements
for food in tree.findall('food'):
    name = food.find('name').text
    price = food.find('price').text
    calories = food.find('calories').text
    description = food.find('description').text

    food_info[name] = {
        'price': price,
        'calories': calories,
        'description': description
    }

# Print the extracted information
for food_name, info in food_info.items():
    print(f"Food: {food_name}")
    print(f"Price: {info['price']}")
    print(f"Calories: {info['calories']}")
    print(f"Description: {info['description']}")
    print('-' * 20)
```

```
Food: Belgian Waffles
Price: $5.95
Calories: 650
Description: Two of our famous Belgian Waffles with plenty of real maple syrup
--------------------
Food: Strawberry Belgian Waffles
Price: $7.95
Calories: 900
Description: Light Belgian waffles covered with strawberries and whipped cream
--------------------
Food: Berry-Berry Belgian Waffles
Price: $8.95
Calories: 900
Description: Light Belgian waffles covered with an assortment of fresh berries and whipped cream
--------------------
Food: French Toast
Price: $4.50
Calories: 600
Description: Thick slices made from our homemade sourdough bread
--------------------
Food: Homestyle Breakfast
Price: $6.95
Calories: 950
Description: Two eggs, bacon or sausage, toast, and our ever-popular hash browns
--------------------
```

2- ConfigParser Module

The configparser module in Python is used for working with configuration files. It allows you to read and write configuration settings in an organized manner. This is particularly useful for managing application settings, parameters, and options

3- Threading

Threading in Python allows you to execute multiple threads (smaller units of a program) concurrently within a single process. It is particularly useful for tasks that can be performed independently and simultaneously, such as I/O-bound operations.

3.1-Basic Threading

In [7]:
```python
import threading

def print_numbers():
    for i in range(1, 6):
        print("Number:", i)

def print_letters():
    for letter in 'abcde':
        print("Letter:", letter)

# Create two threads
thread1 = threading.Thread(target=print_numbers)
thread2 = threading.Thread(target=print_letters)

# Start the threads
thread1.start()
thread2.start()

# Wait for both threads to finish
thread1.join()
thread2.join()

print("Both threads have finished")
```

```
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
Letter: a
Letter: b
Letter: c
Letter: d
Letter: e
Both threads have finished
```

3.2-Threading with Function Arguments

In [8]:
```python
import threading

def print_numbers(start, end):
    for i in range(start, end):
        print("Number:", i)

def print_letters():
    for letter in 'abcde':
        print("Letter:", letter)

# Create two threads with function arguments
thread1 = threading.Thread(target=print_numbers, args=(1, 6))
thread2 = threading.Thread(target=print_letters)

# Start the threads
thread1.start()
thread2.start()

# Wait for both threads to finish
thread1.join()
thread2.join()

print("Both threads have finished")
```

```
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
Letter: a
Letter: b
Letter: c
Letter: d
Letter: e
Both threads have finished
```

3.3-Threading with Lock

In [9]:
```python
import threading

counter = 0
counter_lock = threading.Lock()

def increment_counter():
    global counter
    with counter_lock:
        for _ in range(100000):
            counter += 1

# Create multiple threads to increment the counter
threads = []
for _ in range(5):
    thread = threading.Thread(target=increment_counter)
    thread.start()
    threads.append(thread)

# Wait for all threads to finish
for thread in threads:
    thread.join()

print("Counter value:", counter)
```

```
Counter value: 500000
```

4- Numpy

NumPy is a Python library used for numerical computations, particularly for working with arrays and matrices of numeric data. It provides a wide range of mathematical functions to operate on these arrays efficiently.

NumPy is a powerful library for numerical computations and array operations in Python.

In [10]:

```python
import numpy as np

# Create an array
arr1 = np.array([1, 2, 3, 4, 5])

# Perform operations on the array
mean = np.mean(arr1)
sum = np.sum(arr1)
max_value = np.max(arr1)

# Print the results
print("Array:", arr1)
print("Mean:", mean)
print("Sum:", sum)
print("Max Value:", max_value)
```

```
Array: [1 2 3 4 5]
Mean: 3.0
Sum: 15
Max Value: 5
```

In [11]:

```python
import numpy as np

# Creating a random sequence
random_array = np.random.rand(5)
print("Random Array:", random_array)

# Creating a patterned sequence
patterned_array = np.arange(0, 10, 2)
print("Patterned Array:", patterned_array)

# Creating a 1D array
arr1d = np.array([1, 2, 3, 4, 5])
print("1D Array:", arr1d)

# Creating a 2D array
arr2d = np.array([[1, 2, 3], [4, 5, 6]])
print("2D Array:")
print(arr2d)

# Finding minimum and maximum values
min_value = np.min(arr1d)
max_value = np.max(arr1d)
print("Minimum Value:", min_value)
print("Maximum Value:", max_value)

# Finding indices of minimum and maximum values
min_index = np.argmin(arr1d)
max_index = np.argmax(arr1d)
print("Index of Minimum Value:", min_index)
print("Index of Maximum Value:", max_index)

# Creating identity matrix
identity_matrix = np.eye(3)
print("Identity Matrix:")
print(identity_matrix)

# Creating diagonal matrix
diagonal_matrix = np.diag([1, 2, 3])
print("Diagonal Matrix:")
print(diagonal_matrix)

# Transpose of a matrix
transpose_matrix = np.transpose(arr2d)
```

```python
print("Transpose Matrix:")
print(transpose_matrix)

# Dot product of matrices
matrix_a = np.array([[1, 2], [3, 4]])
matrix_b = np.array([[5, 6], [7, 8]])
dot_product = np.dot(matrix_a, matrix_b)
print("Dot Product:")
print(dot_product)
```

```
Random Array: [0.40998461 0.18012539 0.5737706  0.60926968 0.97293499]
Patterned Array: [0 2 4 6 8]
1D Array: [1 2 3 4 5]
2D Array:
[[1 2 3]
 [4 5 6]]
Minimum Value: 1
Maximum Value: 5
Index of Minimum Value: 0
Index of Maximum Value: 4
Identity Matrix:
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
Diagonal Matrix:
[[1 0 0]
 [0 2 0]
 [0 0 3]]
Transpose Matrix:
[[1 4]
 [2 5]
 [3 6]]
Dot Product:
[[19 22]
 [43 50]]
```

4.1- Linear Algebra

Within numpy there is a sub-module linalg, short for linear algebra The numpy.linalg module in NumPy provides functions for performing various linear algebra operations on arrays, such as matrix multiplication, eigenvalue decomposition, singular value decomposition, and more.

In [12]:

```python
import numpy as np

# Create a sample 2x2 matrix
matrix = np.array([[2, -1], [1, 3]])

# Determinant of a matrix
determinant = np.linalg.det(matrix)
print("Determinant:", determinant)

# Inverse of a matrix
inverse_matrix = np.linalg.inv(matrix)
print("Inverse Matrix:")
print(inverse_matrix)

# Cholesky factorization of a matrix
A = np.array([[4, 12, -16], [12, 37, -43], [-16, -43, 98]])
cholesky_matrix = np.linalg.cholesky(A)
print("Cholesky Matrix:")
print(cholesky_matrix)

# Eigenvalues and eigenvectors of a matrix
eigenvalues, eigenvectors = np.linalg.eig(matrix)
print("Eigenvalues:", eigenvalues)
print("Eigenvectors:")
print(eigenvectors)

# Rank of a matrix
rank = np.linalg.matrix_rank(matrix)
print("Rank of Matrix:", rank)
```

```
Determinant: 7.000000000000001
Inverse Matrix:
[[ 0.42857143  0.14285714]
 [-0.14285714  0.28571429]]
Cholesky Matrix:
[[ 2.  0.  0.]
 [ 6.  1.  0.]
 [-8.  5.  3.]]
Eigenvalues: [2.5+0.8660254j 2.5-0.8660254j]
Eigenvectors:
[[ 0.70710678+0.j          0.70710678-0.j        ]
 [-0.35355339-0.61237244j -0.35355339+0.61237244j]]
Rank of Matrix: 2
```

In [ ]: