Welcome to Python Programming!

Teaching Assistant of Introduction to Numerical Analysis Class

Class Date: 2024-2025

Instructor: Dr. Mina Zarei

Class Link: Telegram group.

Exercise class time: **Monday at 9:30 P002 Class** (determined by student vote)

Session 1 (Install Python in VS Code)

Follow the instructions in the GitHub link.

link: www.github.com/AliSeif96/Introduction-to-Numerical-Analysis/).

---

session 2 (Basic training)

1. Variables and Data Types
2. Taking Output and Input from the User
3. Conditionals (if, elif, else)
4. Loops
5. Functions

---

session 3 (Basic training 2)

6. Lists
7. Arrays
8. Matrices
9. Using `NumPy`

6. Lists

Lists are used to store multiple items in a single variable.

In [55]:
```
# Create a list of fruits, display it, add a new fruit to the list, and display the updated list.
fruits = ["apple", "banana", "cherry"]
print("\nFruits:", fruits)
```
```
Fruits: ['apple', 'banana', 'cherry']
```

In [56]:
```
# list()
fruits = list(("apple", "banana", "cherry")) # note the double round-brackets
print("\nFruits:", fruits)
```
```
Fruits: ['apple', 'banana', 'cherry']
```

In [57]:
```
# Allow Duplicates
fruits = ["apple", "banana", "cherry", "apple"] #Since lists are indexed, lists can have items with the same value:
print("\nFruits:", fruits)
```
```
Fruits: ['apple', 'banana', 'cherry', 'apple']
```

In [58]:
```
# List Length
fruits = ["apple", "banana", "cherry"]
print("\n",fruits)
print("Length of fruits:",len(fruits)) #To determine how many items a list has, use the len() function:
```
```
 ['apple', 'banana', 'cherry']
Length of fruits: 3
```

In [59]:
```
# List Items - Data Types
# List items can be of any data type:
list1 = ["apple", "banana", "cherry"]
list2 = [1, 5, 7, 9, 3]
list3 = [True, False, False]
print("\nlist1:",list1,"\nlist2:",list2,"\nlist3:",list3)
```
```
list1: ['apple', 'banana', 'cherry']
list2: [1, 5, 7, 9, 3]
list3: [True, False, False]
```

In [60]:
```
#A list can contain different data types:
list4 = ["Ali", 34, True, 40, "male"]
print("\ntype of list4:",list4)
```
```
type of list4: ['Ali', 34, True, 40, 'male']
```

In [61]:
```
# type()
fruits = ["apple", "banana", "cherry"]
print("\n",fruits)
print("type of mylist:",type(fruits))#From Python's perspective, lists are defined as objects with the data type 'list':
```
```
 ['apple', 'banana', 'cherry']
type of mylist: <class 'list'>
```

In [62]:
```
# Access Items
# List items are indexed and you can access them by referring to the index number:
fruits = ["apple", "banana", "cherry"]
print("\n",fruits)
print(f"Fruit {2}:",fruits[1])
```
```
 ['apple', 'banana', 'cherry']
Fruit 2: banana
```

In [63]:
```
# append
fruits = ["apple", "banana", "cherry"]
print("\n",fruits)
fruits.append("orange")  # Add an item to the list
print("Fruits after adding orange:", fruits)
```
```
 ['apple', 'banana', 'cherry']
Fruits after adding orange: ['apple', 'banana', 'cherry', 'orange']
```

In [64]:
```
# insert
fruits = ["apple", "banana", "cherry"]
fruits.insert(2, "watermelon")
print("Fruits after insert watermelon in 2 element:",fruits)
```
```
Fruits after insert watermelon in 2 element: ['apple', 'banana', 'watermelon', 'cherry']
```

In [65]:
```
# Remove Specified Index
fruits = ["apple", "banana", "cherry"]
fruits.pop(1)
print("Fruits after Remove element 1:",fruits)
```
```
Fruits after Remove element 1: ['apple', 'cherry']
```

In [66]:
```
# Negative Indexing
# Negative indexing means start from the end
# -1 refers to the last item, -2 refers to the second last item etc.
fruits = ["apple", "banana", "cherry"]
print("\n",fruits)
print(f"Last of Fruit:",fruits[-1])
```
```
 ['apple', 'banana', 'cherry']
Last of Fruit: cherry
```

In [67]:
```
# Range of Indexes
# @Note: The search will start at index 2 (included) and end at index 5 (not included).
# @Remember that the first item has index 0.
fruits = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print("\n",fruits)
print(fruits[2:5])
```
```
 ['apple', 'banana', 'cherry', 'orange', 'kiwi', 'melon', 'mango']
['cherry', 'orange', 'kiwi']
```

In [68]:
```
# Range of Negative Indexes
fruits = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print("\n",fruits)
print(fruits[-4:-1]) # This example returns the items from "orange" (-4) to, but NOT including "mango" (-1):
```
```
 ['apple', 'banana', 'cherry', 'orange', 'kiwi', 'melon', 'mango']
['orange', 'kiwi', 'melon']
```

In [69]:
```
# Sort List Alphanumerically
# List objects have a sort() method that will sort the list alphanumerically, ascending, by default:
fruits = ["orange", "mango", "kiwi", "pineapple", "banana"]
print("\n",fruits)
fruits.sort()
print("Sorted fruits:",fruits)

thislist = [100, 50, 65, 82, 23]
print("\n",thislist)
thislist.sort()
print("Sorted list:",thislist) #thislist.sort(reverse = True) #Sort Descending
```
```
 ['orange', 'mango', 'kiwi', 'pineapple', 'banana']
Sorted fruits: ['banana', 'kiwi', 'mango', 'orange', 'pineapple']

 [100, 50, 65, 82, 23]
Sorted list: [23, 50, 65, 82, 100]
```

In [70]:
```
# Clear the List
# The del keyword can also delete the list completely.
fruits = ["apple", "banana", "cherry"]
print("\n",fruits)
fruits.clear()
print(fruits)
fruits = ["apple", "banana", "cherry"]
del fruits
```
```
 ['apple', 'banana', 'cherry']
[]
```

Exercise

What will be the result of the following syntax:

`mylist = ['apple', 'apple', 'banana', 'cherry']`

`print(mylist[1])`

In [ ]:

Exercise

What should we do if we want it to print the last 4 values?

`thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]`

In [71]:
```
# Check if Item Exists
# To determine if a specified item is present in a list use the in keyword:
fruits = ["apple", "banana", "cherry"]
print("\n",fruits)
if "apple" in fruits:
    print("Yes, 'apple' is in the fruits list")
```
```
 ['apple', 'banana', 'cherry']
Yes, 'apple' is in the fruits list
```

In [72]:
```
fruits = ["apple", "banana", "cherry"]
print("\n",fruits)
for x in fruits:
    print(x)
```
```
 ['apple', 'banana', 'cherry']
apple
banana
cherry
```

In [73]:
```
fruits = ["apple", "banana", "cherry"]
print("\n",fruits)
for i in range(len(fruits)):
    print(fruits[i])
```
```
 ['apple', 'banana', 'cherry']
apple
banana
cherry
```

test

Based on a list of fruits, you want a new list, containing only the fruits with the letter "a" in the name.

Without list comprehension you will have to write a for statement with a conditional test inside:

fruits = ["apple", "banana", "cherry", "kiwi", "mango"] newlist = []

for x in fruits: if "a" in x: newlist.append(x)

print(newlist)

7. Arrays

In Python, arrays are commonly implemented using lists or the NumPy library.

Let's use a basic list to act as an array.

In [76]:
```
# Example of a one-dimensional array (list in Python)
numbers = [1, 2, 3, 4, 5]
print("\nArray (list) of numbers:", numbers)

# Example of iterating over an array using a for loop
print("Iterating over the array:")
for num in numbers:
    print(num)
```
```
Array (list) of numbers: [1, 2, 3, 4, 5]
Iterating over the array:
1
2
3
4
5
```

8. Matrices

A matrix can be represented as a list of lists in Python.

Each inner list represents a row.

In [77]:
```
# Define a 3x3 matrix using a list of lists in Python
matrix = [
    [1, 2, 3],  # Row 1
    [4, 5, 6],  # Row 2
    [7, 8, 9]   # Row 3
]
print(matrix)
```
```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

In [78]:
```
# Print the matrix (each row is printed as a list)
print("\nMatrix (list of lists):")
for row in matrix:
    print(row)  # Print each row
```
```
Matrix (list of lists):
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]
```

In [79]:
```
# Accessing a specific element in the matrix (row 2, column 3)
print("\nElement at row 2, column 3:", matrix[1][2])  # Matrix indices start from 0
```
```
Element at row 2, column 3: 6
```

In [80]:
```
# Modify an element in the matrix (changing the second element of the first row)
matrix[0][1] = 99  # Changing the second element of the first row
print("\nModified matrix:")
for row in matrix:
    print(row)
```
```
Modified matrix:
[1, 99, 3]
[4, 5, 6]
[7, 8, 9]
```

9. Using `NumPy` for Advanced Array and Matrix Operations

NumPy is a powerful library for numerical computing. Let's use it for arrays and matrices.

Why is NumPy Faster Than Lists?

NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.

NumPy is a Python library and is written partially in Python, but most of the parts that require fast computation are written in C or C++.

`Installation of NumPy`

Install it using this command: `pip install numpy`

In [84]:
```
import numpy

arr = numpy.array([1, 2, 3, 4, 5])

print(arr)
```
```
[1 2 3 4 5]
```

In [85]:
```
# NumPy as np
# Importing NumPy for numerical operations
import numpy as np
# Create a NumPy one-dimensional array
array = np.array([1, 2, 3, 4, 5])
print("\nNumPy Array:", array)
```
```
NumPy Array: [1 2 3 4 5]
```

Dimensions in Arrays

In [88]:
```
# 0-D Arrays
import numpy as np

arr = np.array(42)# Create a 0-D array with value 42
print(arr)
```
```
42
```

In [89]:
```
#1-D Arrays
import numpy as np

arr = np.array([1, 2, 3, 4, 5])
print(arr)
```
```
[1 2 3 4 5]
```

In [97]:
```
# 2-D Arrays
import numpy as np

arr = np.array([
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9],
    ])
print(arr)
```
```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

In [98]:
```
# 3-D arrays
import numpy as np

arr = np.array([
    [
        [1, 2, 3],
        [4, 5, 6]],
    [
        [1, 2, 3],
        [4, 5, 6]],
    [
        [1, 2, 3],
        [4, 5, 6]]
    ])
print(arr)
```
```
[[[1 2 3]
  [4 5 6]]

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

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

In [99]:
```
# Check Number of Dimensions
# NumPy Arrays provides the ndim attribute that returns an integer that tells us how many dimensions the array have.
import numpy as np

a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```
```
0
1
2
3
```

In [100…
```
# Create a NumPy 2D matrix (list of lists converted to a NumPy array)
numpy_matrix = np.array([
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
])
print("\nNumPy Matrix:")
print(numpy_matrix)

# Perform matrix operations

# Transpose the matrix (flip rows and columns)
print("\nMatrix Transpose:")
print(numpy_matrix.T)  # Transpose the matrix

# Perform element-wise matrix addition (add the transposed matrix to the original matrix)
print("\nMatrix Addition:")
print(numpy_matrix.T + numpy_matrix)  # Element-wise addition
```
```
NumPy Matrix:
[[1 2 3]
 [4 5 6]
 [7 8 9]]

Matrix Transpose:
[[1 4 7]
 [2 5 8]
 [3 6 9]]

Matrix Addition:
[[ 2  6 10]
 [ 6 10 14]
 [10 14 18]]
```

Searching Arrays

In [139…
```
# You can search an array for a certain value, and return the indexes that get a match.
# To search an array, use the where() method.

import numpy as np

arr = np.array([1, 2, 3, 4, 5, 4, 4])

x = np.where(arr == 4)

print(x)

print(x[0][1])
```
```
(array([3, 5, 6], dtype=int64),)
5
```

In [ ]:

In [ ]: