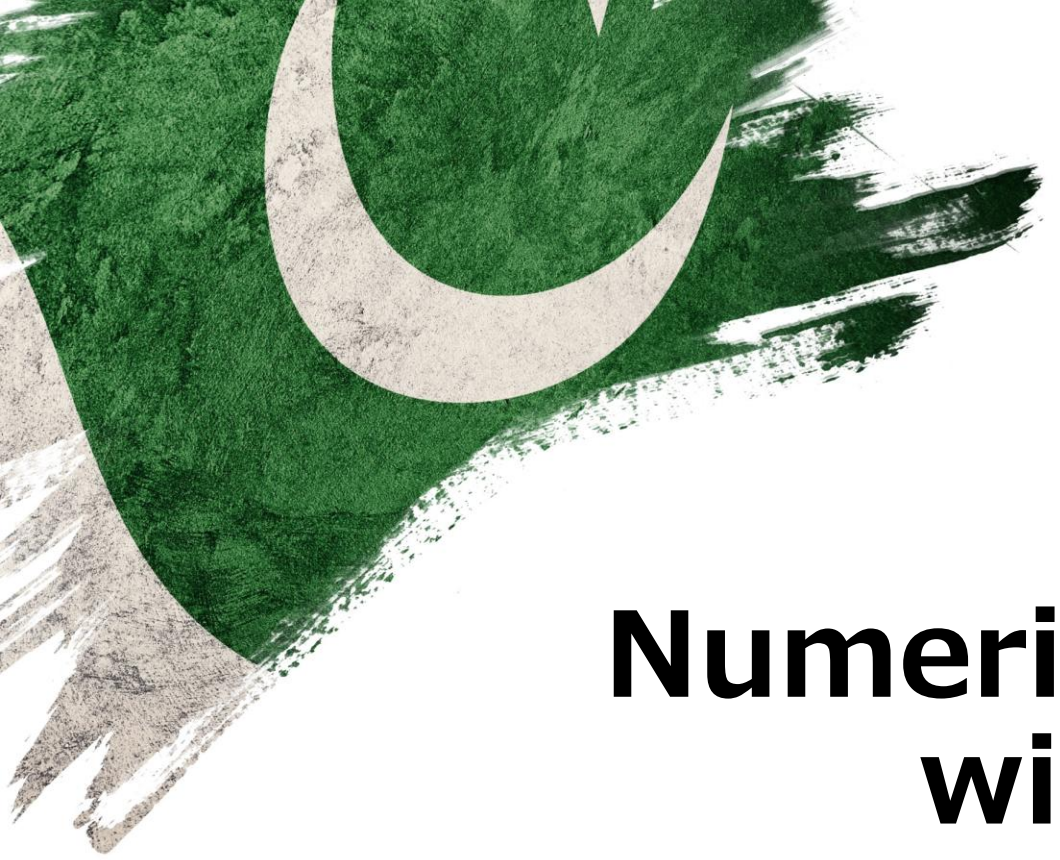

Numerical Computing with NumPy

Class 4

11/2/2025

Acknowledgement

The series of the IT & Japanese language course is Supported by AOTS and OEC.

Ministry of Economy, Trade and Industry

Overseas Employment Corporation

What you have Learnt Last Week

We were focused on following points.

- Introduction of Python, Variables & Operators
- Data Types & String Manipulation
- Introduction of Google colab and Jupyternotebook
- Basic Python Syntax and Data Types
- Lower, upper, length (len), random and split function
- Random module
- Usage of control and loop flow statement
- How to define and use functions
- Usage of lambda function

What you will Learn Today

We will focus on following points.

- Recursive Function and Doc string
- Local and Global Variables
- Understanding of 1D, and 2D NumPy arrays
- Array indexing and slicing
- Upload code on Github
- Quiz
- Q&A Session

Functions as First-Class Objects

Assigning Functions to Variables and Passing Functions as Arguments

[Assigning Functions to Variables]

```
def greet():
    print("Hello!")
say_hello = greet
say_hello()
# Output: Hello!
```

[Passing function as arguments]

```
def apply_function(func, value):
    return func(value)
print(apply_function(lambda x: x * 2, 10))
# Output: 20
```

Functions as First-Class Objects

Assigning Functions to Variables and Passing Functions as Arguments

[Example]

```
def add(x, y):
    return x + y

def operate_on_numbers(func, a, b):
    return func(a, b)

result = operate_on_numbers(add, 5, 10)
print(result)  # Outputs: 15
```

Returning Functions from Other Functions:

A function can return another function instead of a regular value like an integer or string.

[Example]

```
def multiplier(factor):
    def multiply(num):
        return num * factor
    return multiply
double = multiplier(2)
print(double(5))
# Output: 10
```

Recursive Function

Recursive case calls the function with modified arguments.

[Recursive Function]

```
def factorial(n):
    if n == 1:
        return 1
    return n * factorial(n - 1)
print(factorial(5))
# Output: 120
```

A **stack overflow** occurs when a program consumes too much memory in the call stack, usually due to excessive recursion without a proper termination condition

Docstrings in Function

Adding Documentation and Accessing Docstrings Using help()

[Docstrings]

```
def greet(name):
    """This function greets the user by their name."""
    print(f"Hello, {name}!")
greet("Alice")
# Output: Hello, Alice!
-----------------------------------------------------
#Accessing Docstrings Using help()
help(greet)
# Output: This function greets the user by their name.
```

Scope Definition

Scope determines where a variable can be accessed within the code.

[Difference between Local and Global Scope]

- **Local Scope:** Variables defined inside a function (only accessible within that function)
- **Global Scope:** Variables defined outside any function (accessible anywhere in the code)

Local Variables

Local variables cannot be accessed outside their function

[Local variables]

```
def my_function():
    x = 10  # Local variable
    print("Inside function:", x)

my_function()

# print(x)
```

[Key Points]

- Defined inside a function.
- Accessible only within the same function.
- Local variables cannot be accessed outside their function.
- Each function has its own independent local variables

Global Variables

Global variables persist throughout the program

[Global variables]

```
y = 20  # Global variable

def my_function():
    print("Inside function:", y)

my_function()
print("Outside function:", y)  # Works fine
```

[Key Points]

- Defined outside any function.
- Accessible throughout the program, including inside functions.
- Global variables persist throughout the program.
- Can be modified inside a function using the global keyword.

Modifying Global Variables

"global" Keyword Used to modify a global variable inside a function

[Using global Keyword]

```
z = 5  # Global variable

def modify_global():
    global z  # Declaring z as global inside function
    z = 10  # Modifying global variable

modify_global()
print(z)  # Output: 10
```

Modifying Global Variables

"nonlocal" Keyword Used inside nested functions to modify an enclosing scope variable.

[Using nonlocal Keyword]

```
def outer():
    x = 5  # Enclosing variable

    def inner():
        nonlocal x  # Modify the variable in the enclosing function
        x = 10
    inner()
    print("Outer function x:", x)  # Output: 10
outer()
```

Variable Shadowing

When a local variable has the same name as a global variable.

```
y = 50  # Global variable

def test():
    y = 10  # Local variable (shadows global y)
    print(y)  # Output: 10

test()
print(y)  # Output: 50
```

Best Practices about Global Variables

Using Function Arguments and Return Values

```
def add_numbers(a, b):
# Instead of modifying a global variable
   return a + b

result = add_numbers(3, 5)
print(result)  # Output: 8
```

[Best Practices]

- Avoid excessive use of global variables
  (they make debugging harder).
- Use function parameters instead of relying on global scope.
- Return values instead of modifying global variables

NumPy

NumPy (Numerical Python) is a library in Python

[Features]

- Provides high-performance multidimensional arrays and functions to work with them efficiently.
- Used for numerical computations in Python

[Installation and Import]

- pip install numpy
- import numpy as np

NumPy vs Python List

NumPy arrays are faster and more memory-efficient than Python lists

[Example]

```
import numpy as np

# Python List
py_list = [1, 2, 3, 4]
# NumPy Array
np_array = np.array([1, 2, 3, 4])

print(type(py_list))    # <class 'list'>
print(type(np_array))   # <class 'numpy.ndarray'>
```

Creating NumPy Arrays

Creating 1D and 2D arrays using Numpy

[1D Array]

```
arr_1d = np.array([1, 2, 3, 4, 5])
print(arr_1d)
#Output
[1 2 3 4 5]
```

[2D Array]

```
arr_2d = np.array([[1, 2, 3], [4, 5, 6]])
print(arr_2d)
# Check the number of dimensions
print(array.ndim)  #2
#Output
[[1 2 3]
 [4 5 6]]
```

Creating NumPy Arrays

Creating arrays using function

[array filled with zeros]

```
zeros_array = np.zeros((2,3))  # 2 rows, 3 columns
print(zeros_array)
#Output
[[0. 0. 0.]
 [0. 0. 0.]]
```

[array filled with ones]

```
ones_array = np.ones((2, 3))
print(ones_array)
#Output
[[1. 1. 1.]
 [1. 1. 1.]]
```

Creating NumPy Arrays

Creating arrays using function

[array filled specified value]

```
#2x3 matrix filled with 7
full_array = np.full((2, 3), 7)
print(full_array)
#Output
[[7 7 7]
 [7 7 7]]
```

[array filled with evenly spaced values]

```
#5 evenly spaced values from
 1 to 10
linspace_array = np.linspace(1, 10, 5)
print(linspace_array)
#Output
[ 1.    3.25  5.5   7.75 10.  ]
```

[array with a range of numbers]

```
#Start at 1, end at 10
(exclusive), step of 2
range_array = np.arange(1, 10, 2)
print(range_array)
#Output
[1 3 5 7 9]
```

Creating random Arrays

Creating random matrices using random module

[array between 0 & 1]

```
#3x3 random numbers
random_array = np.random.rand(3, 3)
print(random_array)
#Output
[[0.5488135  0.71518937 0.60276338]
 [0.54488318 0.4236548  0.64589411]
 [0.43758721 0.891773   0.96366276]]
```

[array between 1 & 99]

```
## Random integers from 1 to 99
random_int_array = np.random.randint(1, 100, (3, 3))
print(random_int_array)
#Output
[[34 87 56]
 [12 90 23]
 [78 45 67]]
```

Understanding Shape, Size, and Dimension

Array properties and reshaping arrays

[Checking array properties]

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

print(arr.shape)  # (2, 3) → 2 rows, 3 columns
print(arr.ndim)   # 2 → 2D array
print(arr.size)   # 6 → Total number of elements
```

[Reshaping Arrays]

```
arr = np.array([1, 2, 3, 4, 5, 6])
# Convert 1D to 2D (2 rows, 3 columns)
reshaped_arr = arr.reshape(2, 3)
print(reshaped_arr)
#Output
[[1 2 3]
 [4 5 6]]
```

Basic Array Operations

Element wise operation and applying mathematical functions

[Mathematical function]

```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])

print(arr1 + arr2)  # [5 7 9]
print(arr1 - arr2)  # [-3 -3 -3]
print(arr1 * arr2)  # [4 10 18]
print(arr1 / arr2)  # [0.25 0.4  0.5 ]
```

[Trigonometric functions]

```
arr = np.array([0, np.pi/2, np.pi])
print(np.sin(arr))  # [0.  1.  0.]

print(np.exp(arr))  # [ 1.  4.81  23.14]

print(np.sqrt(arr + 1))  # [1. 1.41 1.73]
```

Basic Array Operations

Finding Minimum, Maximum, Sum

[Example]

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

print(np.min(arr))  # 1
print(np.max(arr))  # 6
print(np.sum(arr))  # 21
```

Array Indexing

Accessing elements in a 1D and 2D array

[1D Array]

```
import numpy as np

arr = np.array([10, 20, 30, 40, 50])

print(arr[0])   # First element → 10
print(arr[2])   # Third element → 30
print(arr[-1])  # Last element → 50
print(arr[-3])  # Third last element → 30
```

[2D Array]

```
arr_2d = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])

print(arr_2d[0, 1])  # First row, second column → 2
print(arr_2d[2, 2])  # Third row, third column → 9
print(arr_2d[-1, 0]) # Last row, first column → 7
```

Slicing NumPy Arrays

Slicing 1D Array (arr[start:end:step])

[1D Array]

```
arr = np.array([10, 20, 30, 40, 50, 60, 70])

print(arr[1:5])  # Elements from index 1 to 4 → [20 30 40 50]
print(arr[:4])   # First 4 elements → [10 20 30 40]
print(arr[3:])   # From index 3 to end → [40 50 60 70]
print(arr[::2])  # Every 2nd element → [10 30 50 70]
print(arr[::-1])  # Reverse array → [70 60 50 40 30 20 10]
```

Slicing NumPy Arrays

Slicing 2D Array (arr[row_start:row_end, col_start:col_end])

[2D Array]

```
arr_2d = np.array([[1, 2, 3, 4],
                   [5, 6, 7, 8],
                   [9, 10, 11, 12]])

print(arr_2d[0:2, 1:3])  # Rows 0-1, Columns 1-2 → [[2 3] [6 7]]
print(arr_2d[:, 2])      # All rows, Column 2 → [3 7 11]
print(arr_2d[1, :])      # Row 1, all columns → [5 6 7 8]
print(arr_2d[:, ::2])    # All rows, every 2nd column → [[1 3] [5 7] [9 11]]
```

Conditional Filtering (np.where())

[Example]

```
arr = np.array([10, 20, 30, 40, 50])

result = np.where(arr > 25)  # Get indices where values > 25
print(result)  # (array([2, 3, 4]),) → Indices 2, 3, and 4

filtered_values = arr[np.where(arr > 25)]
print(filtered_values)  # [30 40 50]
```

Using np.take() and np.put()

Fetch elements using index array - np.take()

Modify values at specific indices - np.put()

[np.take()]

```
arr = np.array([10, 20, 30, 40, 50])

print(np.take(arr, [0, 2, 4]))  # [10 30 50]
```

[np.put()]

```
arr = np.array([10, 20, 30, 40, 50])
# Change index 1 to 99 and index 3 to 77
np.put(arr, [1, 3], [99, 77])

print(arr)  # [10 99 30 77 50]
```

Quiz Section

Quiz

Everyone student should click on submit button before time ends otherwise MCQs will not be submitted

[Guidelines of MCQs]

1. There are 20 MCQs
2. Time duration will be 10 minutes
3. This link will be share on 6:10pm (Pakistan time)
4. MCQs will start from 6:15pm (Pakistan time)
5. This is exact time and this will not change
6. Everyone student should click on submit button otherwise MCQs will not be submitted after time will finish
7. Every student should submit Github profile and LinkedIn post link for every class. It include in your performance

Assignment

Assignment should be submit before the next class

[Assignments Requirements]

1. Create a post of today's lecture and post on LinkedIn.
2. Make sure to tag @Plus W @Pak-Japan Centre and instructors LinkedIn profile
3. Upload your code of assignment and lecture on GitHub and share your GitHub profile in respective your region group WhatsApp group
4. If you have any query regarding assignment, please share on your region WhatsApp group.
5. Students who already done assignment, please support other students

Q&A Session

ありがとうございます。
Thank you.
شکریا

For the World with Diverse Individualities