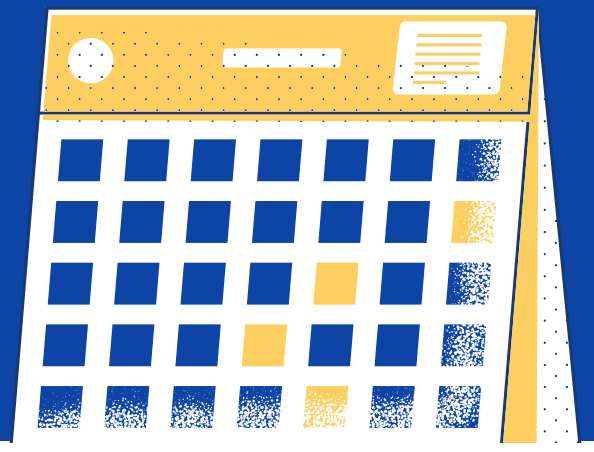
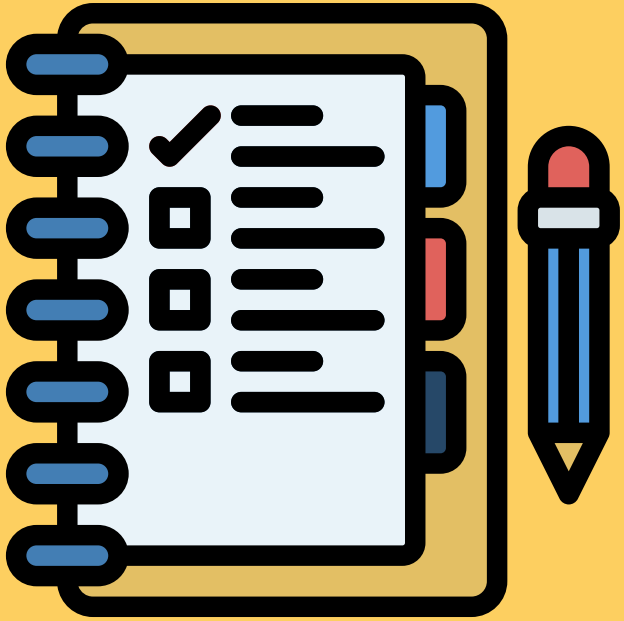
Data Fundamentals

Scripting, NumPy, and Pandas

GOALS



- Install Python and setup a Python development environment.
- Run scripts and use Python's interpreter.
- Manage user input and exceptions.
- Perform file operations.
- Import scripts and libraries.
- Edit and execute Python scripts.
- Understand and utilize Numpy and Pandas libraries for data analysis.

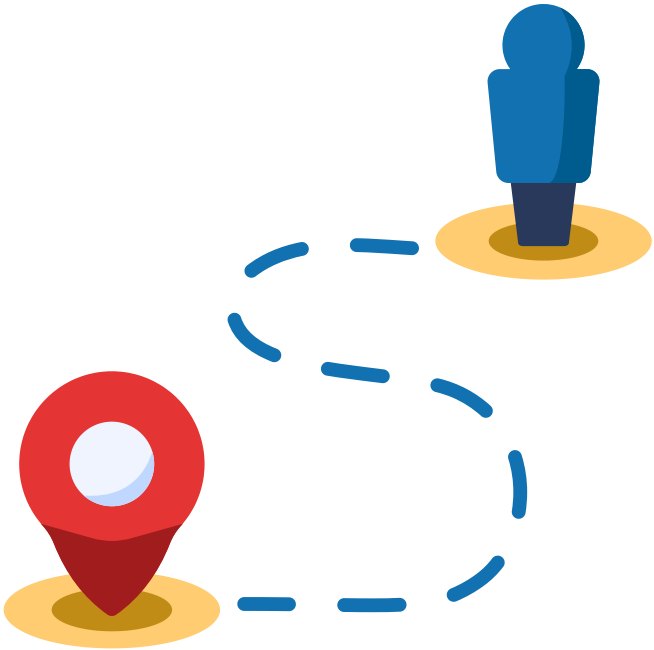


AGENDA

- Welcome
- Running Python Scripts & Interpreter Use
- User Input Interaction
- Exception Handling
- File I/O Operations
- Exploring Some Built-in Functions
- Python Imports: Local, Standard & Third-party Libraries
- Overview of Numpy and Pandas Libraries
- Q&A

Behind every data point, there's a story waiting to be told.

ROADMAP



- **Scripting**
  - Python Installation and Environment Setup
  - Interacting with User Input
  - Handling Exceptions
    - Errors and Exceptions
    - Handling Errors
    - Accessing Error Messages
  - Reading and Writing Files
  - Importing
    - Local Scripts
    - Standard Library
    - Third-Party Libraries
    - Techniques for Importing Modules
  - Editing a Python Script
- **Numpy**
  - Creating and Saving NumPy ndarrays
  - Built-in Functions to Create ndarrays
  - Accessing, Deleting, and Inserting Elements Into ndarrays
  - Slicing ndarrays
  - Boolean Indexing, Set Operations, and Sorting
  - Arithmetic operations and Broadcasting
- **Pandas**
  - Why Use pandas?
  - Creating pandas Series
  - Accessing and Deleting Elements in pandas Series
  - Arithmetic Operations on pandas Series
  - Creating pandas DataFrames
  - Dealing with NaN
  - Loading Data into a pandas DataFrame

SCRIPTING

INTERACTING WITH USER INPUT

Python provides the 'input()' function for interactive user input. It prompts the user for input, and returns the entered string.

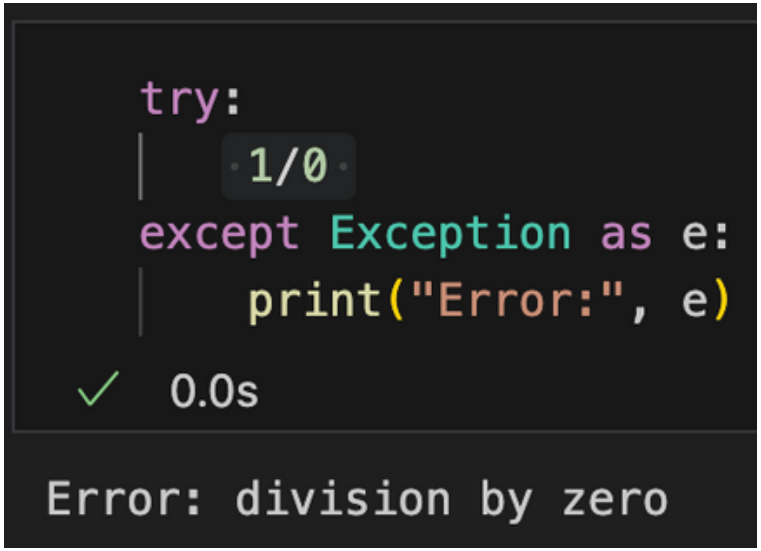
```
# Get the user's name as input
user_name = input("Please enter your name: ")

# Display a personalized greeting
print(f"Hello, {user_name}! Nice to meet you.")
```

✓ 3.7s

```
Hello, Mahmoud! Nice to meet you.
```

HANDLING EXCEPTIONS

The **except** block catches any exception that occurs during the execution of the **try** block. It uses the **as e** syntax to capture the exception object for further inspection.



HANDLING ERRORS

In this code, a **try** block is used to execute the potentially problematic code, which is the division of **1/0**. Since dividing by zero raises a **ZeroDivisionError** in Python, the except block is triggered, specifically catching the **ZeroDivisionError**. When this exception occurs, the code inside the **except** block is executed, which sets the value of **x** to 0.

```
try:
    x = 1/0
except ZeroDivisionError:
    x = 0
```

READING AND WRITING FILES

Python's built-in functions like '**open()**', '**read()**', '**write()**', '**close()**' etc. allow for reading from and writing to files.

```python
try:
    # Writing to a file
    with open("example.txt", "w") as file:
        file.write("Hello, this is an example text.\n")
        file.write("This is the second line.\n")

    # Reading from a file
    with open("example.txt", "r") as file:
        # Read the entire content
        content = file.read()
        print("Read the entire content:\n", content)

        # Read the file line by line
        print("\nRead the file line by line:")
        file.seek(0)  # Reset the file pointer to the beginning
        for line in file:
            print(line.strip())  # Strip newlines for a cleaner output

except FileNotFoundError as e:
    print("File not found:", e)
except PermissionError as e:
    print("Permission error:", e)
except Exception as e:
    print("An error occurred:", e)
```

```python
try:
    # Writing to a file
    file = open("example.txt", "w")
    file.write("Hello, this is an example text.\n")
    file.write("This is the second line.\n")
    file.close()

    # Reading from a file
    file = open("example.txt", "r")
    # Read the entire content
    content = file.read()
    print("Read the entire content:\n", content)
    file.close()

    # Reading the file line by line
    file = open("example.txt", "r")
    print("\nRead the file line by line:")
    for line in file:
        print(line.strip())  # Strip newlines for a cleaner output
    file.close()

except FileNotFoundError as e:
    print("File not found:", e)
except PermissionError as e:
    print("Permission error:", e)
except Exception as e:
    print("An error occurred:", e)
```

IMPORTING

Importing in Python is a way to access code from different modules or scripts. You can import standard library modules, third-party modules, or your own scripts.

```python
# Importing standard library modules
import math
import random

# Importing third-party modules (Assuming 'requests' is a third-party module you've installed)
import requests

# User-defined script/module (Assuming 'my_module.py' contains a function called 'greet')
import my_module

try:
    # Using standard library modules
    print("Using standard library modules:")
    print("Square root of 25:", math.sqrt(25))
    print("Random integer between 1 and 10:", random.randint(1, 10))
    print()

    # Using third-party modules
    print("Using third-party modules:")
    response = requests.get("https://www.example.com")
    print("Status code:", response.status_code)
    print()

    # Using user-defined module
    print("Using user-defined module:")
    my_module.greet("Mahmoud")
    print()

except Exception as e:
    print("An error occurred:", e)
```

TECHNIQUES FOR IMPORTING MODULES

Python provides several ways to import modules: you can import the whole module, import specific functions, or rename the module during import for convenience.

```
# Standard Import
import math

# Import with Alias
import random as rnd

# Import Specific Items
from datetime import datetime, timedelta

# Import All Items (Not Recommended)
from statistics import *

# Conditional Import
condition = True
if condition:
    import os
else:
    import sys

# Dynamic Import
module_name = 'json'
importlib = __import__(module_name)
```

INTRODUCTION TO NUMPY

NumPy, or Numerical Python, is a library for the Python programming language that provides support for large multi-dimensional arrays and matrices. It also contains a variety of mathematical functions to operate on these arrays.

CREATING AND SAVING NUMPY NDARRAYS

NumPy's primary data structure is the ndarray, which stands for 'n-dimensional array'. These arrays can be created using multiple methods and can also be saved and loaded from files.

```
import numpy as np

# Creating ndarray
arr = np.array([1, 2, 3, 4])
print(arr.ndim)  # Prints 1 (dimension)
print(arr.shape)  # Prints (4,) (shape)
print(arr.dtype)  # Prints int64 (data type)
print(arr.size)  # Prints 4 (size)

# Saving and Loading ndarray
np.save('my_array', arr)
loaded_arr = np.load('my_array.npy')
print(loaded_arr)  # Prints [1 2 3 4]
```

BUILT-IN FUNCTIONS TO CREATE NDARRAYS

NumPy provides a variety of built-in functions to create ndarrays of different shapes and fill them with specific or random values.

```
zeros = np.zeros((3, 4))
#output
"""
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
"""
ones = np.ones((3, 2))
#output
"""
[[1. 1.]
 [1. 1.]
 [1. 1.]]
"""
full = np.full((2, 3), 5)
#output
"""
[[5 5 5]
 [5 5 5]]
"""
```

```
eye = np.eye(5)
#output
# identity matrix
"""
[[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]]
"""
diag = np.diag([10, 20, 30, 50])
#output
"""
[[10  0  0  0]
 [ 0 20  0  0]
 [ 0  0 30  0]
 [ 0  0  0 50]]
 """
arr_range = np.arange(1, 14, 3)#[ 1  4  7 10 13]
linspace = np.linspace(0, 25, 10)#[ 0.          2.77777778  5.55555556 ...
random_floats = np.random.random((3, 3)) #random floats between 0 and 1
random_ints = np.random.randint(4, 15, size=(3, 2)) #random ints between 4 and 15
```

ACCESSING, DELETING, AND INSERTING ELEMENTS INTO NDARRAYS

Elements of ndarrays can be accessed, modified, deleted, or inserted using indices. Additionally, ndarrays can be appended or stacked together.

```
# Accessing and modifying 1-D arrays
arr = np.array([1, 2, 3, 4])
print(arr[0])  # Prints 1
arr[0] = 5
print(arr)  # Prints [5 2 3 4]

# Deleting elements
arr = np.delete(arr, [0, 2])
print(arr)  # Prints [2 4]

# Appending elements
arr = np.append(arr, [6, 7])
print(arr)  # Prints [2 4 6 7]

# Inserting elements
arr = np.insert(arr, 1, [8, 9])
print(arr) # Prints [2 8 9 4 6 7]
```

SLICING NDARRAYS

Slicing allows access to sub-arrays from ndarrays. Additionally, ndarrays can be used as indices to select specific rows or columns.

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

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

BOOLEAN INDEXING, SET OPERATIONS, AND SORTING

NumPy provides powerful features such as Boolean indexing, set operations, and built-in sorting mechanisms.

```
arr = np.array([1, 2, 3, 4, 5, 6])
print(arr[arr > 3])  # Boolean Indexing, prints [4 5 6]

x = np.array([1, 2, 3, 4])
y = np.array([3, 4, 5, 6])
print(np.intersect1d(x, y))  # Prints [3 4]
print(np.setdiff1d(x, y))  # Prints [1 2]
print(np.union1d(x, y))  # Prints [1 2 3 4 5 6]

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

ARITHMETIC OPERATIONS AND BROADCASTING

NumPy supports element-wise arithmetic operations and broadcasting for arrays of different shapes.

```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
print(arr1 + arr2)  # Prints [5 7 9]
print(arr1 * arr2)  # Prints [4 10 18]

arr3 = np.array([1])
print(arr1 + arr3)  # Broadcasting, prints [2 3 4]
```

INTRODUCTION TO PANDAS

Pandas is a popular Python library for data manipulation and analysis.

CREATING AND MODIFYING PANDAS SERIES

Pandas Series can be created with the pd.Series() function, and elements can be accessed, modified, and deleted using various methods.

```
groceries = pd.Series(data=[30, 6, 'Yes', 'No'], index=['eggs', 'apples', 'milk', 'bread'])
groceries['eggs'] = 12
groceries.drop('bread', inplace=True)
```

ARITHMETIC OPERATIONS ON PANDAS SERIES

Arithmetic operations can be performed element-wise on Pandas Series, and functions from the NumPy library can also be applied to them.

```
import numpy as np
fruits = pd.Series(data=[10, 6, 3], index=['apples', 'oranges', 'bananas'])
fruits * 2
np.sqrt(fruits)
```

CREATING PANDAS DATAFRAMES

Pandas DataFrames can be created from dictionaries of Pandas Series or lists, with custom row labels.

```
items = {'Bob': pd.Series(data=[245, 25, 55], index=['bike', 'pants', 'watch']),
         'Alice': pd.Series(data=[40, 110, 500, 45], index=['book', 'glasses', 'bike', 'pants'])}
pd.DataFrame(items)
```

DEALING WITH NAN

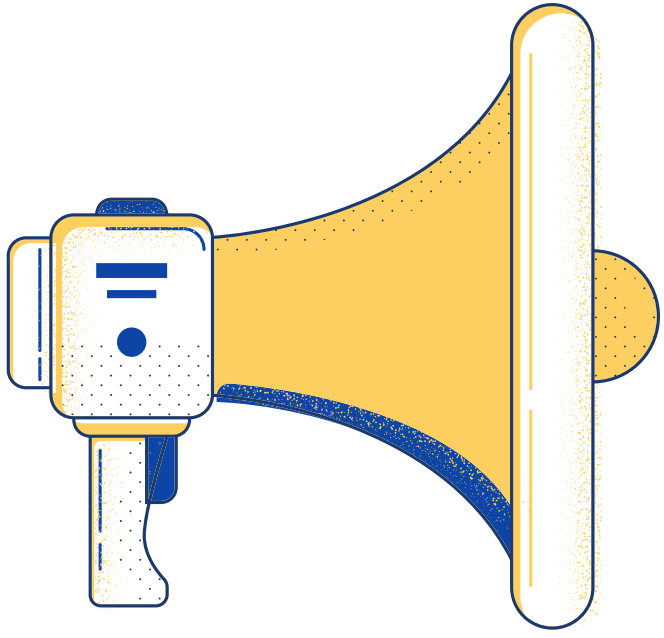
Pandas provides functions to detect, eliminate, and replace NaN values in a DataFrame.

```
df.isnull().sum()
df.dropna(axis=0) # axis=1 for columns, axis=0 for rows
df.fillna(value=0)
df.interpolate(method='linear', axis=0)
```

LOADING AND ANALYZING DATA WITH PANDAS

Pandas can load data from various formats (such as CSV), and provides methods to inspect and perform statistical operations on the data.

```
df = pd.read_csv('data.csv')
df.head()
df.describe()
df.groupby('column_name').mean()
```



Q&A Session: Let's explore and understand together

RESOURCES

- Scripting
- Numpy
- Pandas
- Oman Tourist Data Explorer V2
- Online Python
- Session-9 jupyter notebook
- Python Installation & ENV 1
- Python Installation & ENV 2
- Python Installation & ENV 3
- Pandas vs NumPy
- Numpy_Cheat_Sheet
- Pandas_Cheat_Sheet

Your presence today has added value to our shared learning journey. Thank you for joining us!