**New Age University**

# Python Programming Course

E-Book

### Course Description

This comprehensive 8-week Python Programming course aims to provide students with a solid foundation in Python, which is commonly used in the development of many types of software, web designs and even machine learning (AI). The course is designed to cover both fundamental and advanced topics, ranging from *Basic Syntax and Data Types, Functions and Modules, Object-Oriented Programming (OOP), Python Libraries & Modules* to *Machine Learning and Asynchronous Programming.* Through a blend of theoretical explanations and practical exercises students will gain hands-on experience and essential skills required to develop efficient, effective, and scalable Python applications. The course will culminate in a Capstone Project that integrates all learned concepts, preparing students for real-world Python development challenges.

### Learning Objectives

• Understand the core syntax, data types, and control flow statements in Python.

• Develop proficiency in writing reusable functions and organizing code into modules.

• Gain experience in reading from and writing to files, as well as handling various data formats.

• Understand and implement fundamental data structures such as lists, tuples, dictionaries, and sets.

• Use libraries like pandas for data manipulation and analysis, and apply techniques to work with real-world datasets.

• Create Visualizations of data using matplotlib and seaborn, and create meaningful plots and charts.

• Understand machine learning concepts and apply them using scikit-learn for supervised and unsupervised learning tasks.

• Explore Advanced Python Topics like Object-Oriented Programming (OOP), Functional Programming, Asynchronous Programming, Exception Handling, and working with JSON and APIs.

• Complete a Capstone Project by applying all learned concepts to demonstrate practical Python programming skills and problem-solving abilities.

### Course Structure

The course is divided into 8 weeks, each focusing on different aspects of Python programming:

* **Week 1**: Introduction to Python Basics
* **Week 2**: Control Flow and Functions
* **Week 3**: Working with Files and Data
* **Week 4**: Data Structures and Algorithms
* **Week 5**: Data Analysis with pandas
* **Week 6**: Data Visualization with matplotlib and seaborn
* **Week 7**: Machine Learning with scikit-learn
* **Week 8**: Advanced Topics in Python

Each week includes theoretical explanations, practical examples, and exercises to reinforce learning and ensure participants can apply what they've learned in real-world scenarios.

# Week 1: Introduction to Python Basics

**Overview:** This week introduces the foundational concepts of Python programming. Students will learn about Python’s syntax, basic data types, and control flow statements. This week aims to provide a solid foundation for programming in Python, ensuring that students are comfortable with the language’s basics before advancing to more complex topics.

**Topics Covered:**

1. Overview of Python Programming Language
2. Installing Python and Setting Up
3. Basic Syntax and Data Types
4. Control Flow Statements
5. Functions and Modules
6. Summary
7. Exercises and Real-World Applications

#### 1. Overview of Python Programming Language

**Introduction:** Python is a high-level, interpreted programming language known for its readability and simplicity. It was created by Guido van Rossum and first released in 1991. Python’s design philosophy emphasizes code readability with its notable use of significant indentation.

**Key Features:**

* **Readable and Maintainable Code:** Python’s syntax is clear and concise, making it easier to read and write.
* **Versatile and Multi-Purpose:** Python supports various programming paradigms, including procedural, object-oriented, and functional programming.
* **Extensive Libraries and Frameworks:** Python has a rich ecosystem of libraries and frameworks, such as NumPy for numerical computations and Django for web development.
* **Cross-Platform Compatibility:** Python runs on various operating systems, including Windows, macOS, and Linux.

**Applications:**

* Web Development: Using frameworks like Django and Flask.
* Data Analysis: With libraries such as Pandas and NumPy.
* Machine Learning: Using libraries like Scikit-learn and TensorFlow.
* Automation and Scripting: For automating repetitive tasks and writing scripts.

#### 2. Installing Python and Setting Up the Development Environment

**Installation Steps:**

1. **Download Python:**
   * Go to the official Python website [python.org](https://www.python.org/).
   * Download the latest stable release for your operating system.
2. **Install Python:**
   * Run the installer and follow the instructions.
   * Ensure that you check the box to add Python to your system PATH.
3. **Setting Up an IDE:**
   * **PyCharm:** A powerful IDE specifically designed for Python.
   * **Visual Studio Code (VSCode):** A versatile editor with Python support through extensions.
   * **Jupyter Notebook:** An interactive environment for coding and data analysis.

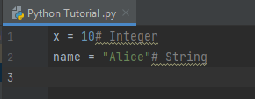
**Configuration:**

* Verify installation by opening a terminal or command prompt and typing python --version or python3 --version.
* Configure your IDE to use the installed Python interpreter.

#### 3. Basic Syntax and Data Types

**Variables and Data Types:**

* **Variables:** Containers for storing data values. Variables do not require explicit declaration.



* **Data Types:**
  + **Integers:** Whole numbers, e.g., 5, 100
  + **Floats:** Decimal numbers, e.g., 3.14, 2.0
  + **Strings:** Sequence of characters enclosed in quotes, e.g., "Hello, World!"
  + **Booleans:** Represents True or False

**Data Structures:**

* **Lists:** Ordered, mutable collection of items.



* **Tuples:** Ordered, immutable collection of items.

**Python Code**

coordinates = (10.0, 20.0)

* **Dictionaries:** Unordered collection of key-value pairs.

python

Copy code

person = {"name": "Alice", "age": 25}

**Type Conversion:**

* Convert between types using built-in functions.

python

Copy code

num\_str = "123"

num\_int = int(num\_str) # Convert string to integer

#### 4. Control Flow Statements

**Conditional Statements:**

* **if, elif, and else** statements allow branching based on conditions.

python

Copy code

age = 18if age < 18:

print("Minor")

elif age == 18:

print("Just turned adult")

else:

print("Adult")

**Looping Structures:**

* **For Loop:** Iterates over a sequence (list, tuple, string).

python

Copy code

for i inrange(5):

print(i)

* **While Loop:** Repeats as long as a condition is true.

python

Copy code

count = 0while count < 5:

print(count)

count += 1

**Control Flow Keywords:**

* **break:** Exits a loop.
* **continue:** Skips the rest of the code inside a loop for the current iteration.

#### 5. Functions and Modules

**Defining Functions:**

* Functions are reusable blocks of code.

python

Copy code

defgreet(name):

returnf"Hello, {name}!"print(greet("Alice"))

**Function Arguments:**

* **Positional Arguments:** Must be provided in the correct order.
* **Keyword Arguments:** Specified by name.

python

Copy code

defgreet(name, age=25):

returnf"Hello, {name}. You are {age} years old."print(greet("Alice", age=30))

**Importing Modules:**

* Use import to include standard or third-party modules.

python

Copy code

import math

print(math.sqrt(16))

**Using Modules:**

* Python modules help organize code into manageable sections.

#### Activity

* **Discussion:** Discuss how Python’s versatility can be applied in various fields like web development, data analysis, and automation. Share examples of Python applications you have encountered or used.

#### Reading Materials

* **Article:** "The Evolution of Python Programming Language" – Explore the development and growth of Python.
* **Book Chapter:** "Introduction to Python" from "Python Crash Course" by Eric Matthes – A practical guide to learning Python.

# Week 2: Control Flow and Functions

**Overview:** In Week 2, students will deepen their understanding of Python’s control flow mechanisms and functions. This week focuses on advanced control flow, including nested conditions and exception handling, as well as developing modular code through functions and modules. Students will also learn about error handling and debugging techniques to create robust and maintainable Python code.

**Topics Covered:**

1. Advanced Control Flow Statements
2. Function Definitions and Scope
3. Modular Programming
4. Error Handling and Debugging
5. Summary
6. Hands-On Exercises and Real-World Applications

#### 1. Advanced Control Flow Statements

**Nested Conditions and Loops:**

* **Nested Conditions:** You can place one if statement inside another. This is useful for more complex decision-making scenarios.

python

Copy code

x = 10if x > 5:

if x < 15:

print("x is between 5 and 15")

else:

print("x is 15 or greater")

else:

print("x is 5 or less")

* **Nested Loops:** Loops inside other loops allow for more complex iterations, such as processing multi-dimensional data.

python

Copy code

for i inrange(3):

for j inrange(2):

print(f"i = {i}, j = {j}")

**Exception Handling:**

* **try, except, finally:** Manage errors gracefully and ensure code execution regardless of whether an error occurred.

python

Copy code

try:

x = 10 / 0except ZeroDivisionError:

print("Cannot divide by zero")

finally:

print("This will execute no matter what")

#### 2. Function Definitions and Scope

**Local and Global Variables:**

* **Local Variables:** Defined within a function and accessible only inside that function.

python

Copy code

defmy\_function():

local\_var = 10print(local\_var)

my\_function()

* **Global Variables:** Defined outside any function and accessible throughout the script.

python

Copy code

global\_var = 20defmy\_function():

print(global\_var)

my\_function()

**Default and Keyword Arguments:**

* **Default Arguments:** Provide default values for function parameters.

python

Copy code

defgreet(name, greeting="Hello"):

returnf"{greeting}, {name}!"print(greet("Alice"))

print(greet("Bob", "Hi"))

* **Keyword Arguments:** Allow specifying arguments by name when calling the function.

python

Copy code

defgreet(name, age):

returnf"Hello, {name}. You are {age} years old."print(greet(age=30, name="Alice"))

**Lambda Functions and Anonymous Functions:**

* **Lambda Functions:** Small, anonymous functions created using the lambda keyword.

python

Copy code

add = lambda x, y: x + y

print(add(5, 3))

#### 3. Modular Programming

**Creating and Importing Modules:**

* **Creating Modules:** Save reusable code in .py files. Example: mymodule.py

python

Copy code

defsay\_hello(name):

returnf"Hello, {name}!"

* **Importing Modules:** Use the import statement to access functions from other files.

python

Copy code

import mymodule

print(mymodule.say\_hello("Alice"))

**Organizing Code into Packages:**

* **Packages:** Directories containing multiple modules and an \_\_init\_\_.py file.

plaintext

Copy code

mypackage/

\_\_init\_\_.py

module1.py

module2.py

* **Using Packages:**

python

Copy code

from mypackage import module1

#### 4. Error Handling and Debugging

**Common Errors and Debugging Techniques:**

* **Syntax Errors:** Mistakes in code structure that prevent execution.
* **Runtime Errors:** Errors that occur during program execution, such as division by zero.
* **Logical Errors:** Errors where the code runs but produces incorrect results.

**Using Python’s Built-in Debugging Tools:**

* **print() Statements:** Basic technique for tracing values.
* **pdb Module:** Python’s built-in debugger for interactive debugging.

python

Copy code

import pdb

pdb.set\_trace()

#### 5. Hands-On Exercises

**Writing Functions for Common Tasks:**

* Implement functions for tasks like calculating factorials, generating Fibonacci sequences, or processing strings.

**Implementing Control Flow in Practical Examples:**

* Create scripts that use nested loops and conditional statements to solve practical problems, such as a simple text-based game or a basic data analysis task.

#### Activity

* **Discussion:** Share experiences with debugging and error handling. Discuss common challenges and effective strategies for managing and resolving code issues.

#### Reading Materials

* **Article:** "Advanced Python Programming: Control Flow and Functions" – Explore in-depth explanations of Python’s control structures and modular programming.
* **Book Chapter:** "Functions and Modules" from "Python Crash Course" by Eric Matthes – Learn about advanced function features and organizing code.

# Week 3: Working with Files and Data

**Overview:** In Week 3, students will delve into file handling and data manipulation using Python. This week covers essential techniques for reading from and writing to various file formats, parsing different data formats, and performing basic data cleaning and preprocessing. Students will also be introduced to key data analysis libraries, pandas and numpy, which are fundamental tools for data analysis.

**Topics Covered:**

1. **Reading and Writing Files**
2. **Data Formats and Parsing**
3. **Data Cleaning and Preprocessing**
4. **Introduction to Data Analysis Libraries**
5. **Hands-On Exercises**

#### 1. Reading and Writing Files

**File Operations:**

* **Opening Files:** Use the open() function to access files in different modes ('r', 'w', 'a', 'b').

python

Copy code

file = open('example.txt', 'r') # Open file for reading

* **Reading Files:** Read file content using methods like read(), readline(), and readlines().

python

Copy code

content = file.read() # Read entire file content

* **Writing Files:** Write data to files using write() and writelines().

python

Copy code

file = open('example.txt', 'w') # Open file for writing

file.write("Hello, World!")

* **Closing Files:** Always close files using close() to free system resources.

python

Copy code

file.close()

**Working with Text and Binary Files:**

* **Text Files:** Handle files containing plain text.

python

Copy code

withopen('textfile.txt', 'r') as file:

text = file.read()

* **Binary Files:** Read and write files in binary mode for non-text data.

python

Copy code

withopen('image.jpg', 'rb') as file:

binary\_data = file.read()

#### 2. Data Formats and Parsing

**Handling CSV Files:**

* **Reading CSV Files:** Use the csv module to read and parse CSV data.

python

Copy code

import csv

withopen('data.csv', 'r') as file:

reader = csv.reader(file)

for row in reader:

print(row)

* **Writing CSV Files:** Write data to CSV files using csv.writer.

python

Copy code

withopen('data.csv', 'w', newline='') as file:

writer = csv.writer(file)

writer.writerow(['Name', 'Age'])

writer.writerow(['Alice', 30])

**Handling JSON Files:**

* **Reading JSON Files:** Use the json module to parse JSON data.

python

Copy code

import json

withopen('data.json', 'r') as file:

data = json.load(file)

print(data)

* **Writing JSON Files:** Convert Python objects to JSON format and write to files.

python

Copy code

withopen('data.json', 'w') as file:

json.dump(data, file, indent=4)

**Handling XML Files:**

* **Parsing XML Files:** Use libraries like xml.etree.ElementTree to work with XML data.

python

Copy code

import xml.etree.ElementTree as ET

tree = ET.parse('data.xml')

root = tree.getroot()

for child in root:

print(child.tag, child.attrib)

#### 3. Data Cleaning and Preprocessing

**Basic Data Cleaning Techniques:**

* **Removing Duplicates:** Identify and remove duplicate records.

python

Copy code

df.drop\_duplicates(inplace=True)

* **Filtering Outliers:** Detect and handle outliers in data.

python

Copy code

df = df[df['value'] < threshold]

**Handling Missing and Inconsistent Data:**

* **Filling Missing Values:** Use methods like fillna() to handle missing data.

python

Copy code

df.fillna(method='ffill', inplace=True)

* **Removing Missing Values:** Drop rows or columns with missing data.

python

Copy code

df.dropna(inplace=True)

* **Correcting Inconsistencies:** Standardize data formats and correct inconsistencies.

python

Copy code

df['date'] = pd.to\_datetime(df['date'])

#### 4. Introduction to Data Analysis Libraries

**Overview of pandas and numpy:**

* **pandas:** A powerful library for data manipulation and analysis, providing data structures like DataFrame and Series.

python

Copy code

import pandas as pd

* **numpy:** A library for numerical operations, providing support for arrays and mathematical functions.

python

Copy code

import numpy as np

**Basic Operations with pandas DataFrames:**

* **Creating DataFrames:** Construct DataFrames from various data sources.

python

Copy code

df = pd.DataFrame({

'Name': ['Alice', 'Bob'],

'Age': [30, 25]

})

* **DataFrame Operations:** Perform common operations like filtering, grouping, and aggregating.

python

Copy code

df\_filtered = df[df['Age'] > 25]

* **Reading and Writing Data:** Use pandas to read from and write to files in CSV, Excel, and other formats.

python

Copy code

df.to\_csv('output.csv', index=False)

#### 5. Hands-On Exercises

**Reading from and Writing to Files:**

* Practice reading from and writing to text and binary files.
* Implement file operations with various file formats (CSV, JSON, XML).

**Parsing and Manipulating Data from Different Sources:**

* Use pandas to read data from CSV and Excel files, and perform basic data manipulations.
* Clean and preprocess data, handling missing values and correcting inconsistencies.

#### Activity

* **Project:** Develop a small project involving reading data from a file, cleaning and preprocessing it, and then performing basic analysis.

#### Reading Materials

* **Article:** "File Handling and Data Manipulation in Python" – Explore techniques for managing and processing files and data.
* **Book Chapter:** "Working with Data in Python" from "Python for Data Analysis" by Wes McKinney – Detailed instructions on using pandas for data manipulation.

# Week 4: Data Structures and Algorithms

**Overview:** In Week 4, students will explore fundamental data structures and algorithms in Python. This week aims to provide a comprehensive understanding of how different data structures are implemented and utilized in programming. Students will learn how to efficiently solve problems using various algorithms and understand the concept of algorithmic complexity.

**Topics Covered:**

1. **Basic Data Structures**
2. **Advanced Data Structures**
3. **Algorithms and Problem Solving**
4. **Implementing Algorithms in Python**
5. **Hands-On Exercises**

#### 1. Basic Data Structures

**Lists:**

* **Definition:** Ordered, mutable collections of items.
* **Operations:** Accessing elements, appending, inserting, removing.
* **Example:**

python

Copy code

my\_list = [1, 2, 3]

my\_list.append(4)

**Tuples:**

* **Definition:** Ordered, immutable collections of items.
* **Operations:** Accessing elements, concatenation, slicing.
* **Example:**

python

Copy code

my\_tuple = (1, 2, 3)

**Sets:**

* **Definition:** Unordered collections of unique items.
* **Operations:** Adding, removing, set operations (union, intersection).
* **Example:**

python

Copy code

my\_set = {1, 2, 3}

my\_set.add(4)

**Dictionaries:**

* **Definition:** Unordered collections of key-value pairs.
* **Operations:** Accessing values, adding key-value pairs, deleting items.
* **Example:**

python

Copy code

my\_dict = {'a': 1, 'b': 2}

my\_dict['c'] = 3

#### 2. Advanced Data Structures

**Stacks:**

* **Definition:** Last-In-First-Out (LIFO) data structure.
* **Operations:** Push (add), pop (remove), peek (view top element).
* **Example:**

python

Copy code

stack = []

stack.append(1) # Push

stack.pop() # Pop

**Queues:**

* **Definition:** First-In-First-Out (FIFO) data structure.
* **Operations:** Enqueue (add), dequeue (remove), peek (view front element).
* **Example:**

python

Copy code

from collections import deque

queue = deque()

queue.append(1) # Enqueue

queue.popleft() # Dequeue

**Linked Lists:**

* **Definition:** A sequence of nodes, where each node contains data and a reference to the next node.
* **Operations:** Traversing, inserting, deleting nodes.
* **Example:** Custom implementation of a linked list node and operations.

**Trees:**

* **Definition:** Hierarchical data structure with nodes connected by edges.
* **Operations:** Traversal (pre-order, in-order, post-order), insertion, deletion.

**Graphs:**

* **Definition:** A collection of nodes (vertices) and edges connecting them.
* **Operations:** Graph traversal (BFS, DFS), finding shortest paths.

#### 3. Algorithms and Problem Solving

**Sorting Algorithms:**

* **Bubble Sort:** Simple comparison-based sorting.
* **Quick Sort:** Efficient, divide-and-conquer sorting algorithm.
* **Merge Sort:** Divide-and-conquer sorting with merge operations.
* **Example:**

python

Copy code

defquick\_sort(arr):

iflen(arr) <= 1:

return arr

pivot = arr[len(arr) // 2]

left = [x for x in arr if x < pivot]

middle = [x for x in arr if x == pivot]

right = [x for x in arr if x > pivot]

return quick\_sort(left) + middle + quick\_sort(right)

**Searching Algorithms:**

* **Linear Search:** Sequentially searches through a list.
* **Binary Search:** Efficiently searches a sorted list by dividing the search interval.
* **Example:**

python

Copy code

defbinary\_search(arr, target):

low, high = 0, len(arr) - 1while low <= high:

mid = (low + high) // 2if arr[mid] == target:

return mid

elif arr[mid] < target:

low = mid + 1else:

high = mid - 1return -1

**Algorithmic Complexity and Big O Notation:**

* **Definition:** Measures the efficiency of an algorithm in terms of time and space.
* **Common Complexities:** O(1), O(log n), O(n), O(n log n), O(n^2).

#### 4. Implementing Algorithms in Python

**Coding Common Algorithms:**

* **Sorting Algorithms:** Implement various sorting algorithms.
* **Searching Algorithms:** Implement linear and binary search algorithms.

**Analyzing Performance and Efficiency:**

* **Benchmarking:** Measure execution time and efficiency of algorithms.
* **Example:**

python

Copy code

import time

start\_time = time.time()

quick\_sort([3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5])

end\_time = time.time()

print("Execution Time:", end\_time - start\_time)

#### 5. Hands-On Exercises

**Implementing and Using Various Data Structures:**

* Write Python code to implement and use stacks, queues, linked lists, trees, and graphs.

**Solving Problems Using Different Algorithms:**

* Apply sorting and searching algorithms to solve practical problems and analyze their performance.

#### Activity

* **Project:** Develop a small application involving different data structures and algorithms. This could include tasks like implementing a to-do list with stack operations or a ticket booking system with queue operations.

#### Reading Materials

* **Article:** "Understanding Data Structures and Algorithms in Python" – An overview of various data structures and their applications.
* **Book Chapter:** "Data Structures and Algorithms" from "Python Data Structures and Algorithms" by Benjamin Baka – In-depth explanations and examples of data structures and algorithms in Python.

# Week 5: Data Manipulation and Analysis with pandas

**Overview:** This week focuses on using the pandas library for data manipulation and analysis. Students will learn to handle and process large datasets, perform data cleaning and transformation, and apply various analytical techniques using pandas DataFrames and Series.

**Topics Covered:**

1. **Introduction to** pandas
2. **Data Cleaning and Transformation**
3. **Data Filtering and Selection**
4. **Merging and Joining Data**
5. **Hands-On Exercises**

#### 1. Introduction to pandas

**Overview of DataFrames and Series:**

* **DataFrame:** A 2-dimensional, size-mutable, and potentially heterogeneous tabular data structure with labeled axes (rows and columns).
  + **Creation:** Using dictionaries, lists, or from existing data sources.
  + **Example:**

python

Copy code

import pandas as pd

data = {'Name': ['Alice', 'Bob'], 'Age': [25, 30]}

df = pd.DataFrame(data)

* **Series:** A 1-dimensional labeled array capable of holding any data type.
  + **Creation:** From lists or arrays.
  + **Example:**

python

Copy code

s = pd.Series([1, 2, 3], index=['a', 'b', 'c'])

**Basic Operations with pandas:**

* **Reading Data:** From CSV, Excel, SQL databases.
  + **Example:**

python

Copy code

df = pd.read\_csv('data.csv')

* **Writing Data:** To various formats including CSV and Excel.
  + **Example:**

python

Copy code

df.to\_csv('output.csv', index=False)

#### 2. Data Cleaning and Transformation

**Handling Missing Values:**

* **Identifying Missing Values:** Using methods like isnull() and notnull().
  + **Example:**

python

Copy code

df.isnull().sum()

* **Filling Missing Values:** Using fillna().
  + **Example:**

python

Copy code

df.fillna(0, inplace=True)

* **Dropping Missing Values:** Using dropna().
  + **Example:**

python

Copy code

df.dropna(inplace=True)

**Data Aggregation and Grouping:**

* **Grouping Data:** Using groupby() to perform aggregate operations.
  + **Example:**

python

Copy code

grouped = df.groupby('column\_name').mean()

* **Aggregation Functions:** Sum, mean, count, etc.
  + **Example:**

python

Copy code

df.groupby('category').agg({'value': ['sum', 'mean']})

#### 3. Data Filtering and Selection

**Indexing and Slicing:**

* **Selecting Rows and Columns:** Using .loc[], .iloc[], and direct column access.
  + **Example:**

python

Copy code

df.loc[0] # Selects the first row

df['column\_name'] # Selects a column

* **Slicing DataFrames:** Using slicing operations for rows and columns.
  + **Example:**

python

Copy code

df[1:5] # Slices rows from index 1 to 5

**Filtering and Querying Data:**

* **Boolean Indexing:** Filtering rows based on conditions.
  + **Example:**

python

Copy code

df[df['age'] > 30]

* **Query Method:** Using query() for complex filtering.
  + **Example:**

python

Copy code

df.query('age > 30')

#### 4. Merging and Joining Data

**Combining DataFrames:**

* **Concatenation:** Using concat() to stack DataFrames.
  + **Example:**

python

Copy code

df\_combined = pd.concat([df1, df2])

* **Merging:** Using merge() to combine DataFrames on common columns.
  + **Example:**

python

Copy code

df\_merged = pd.merge(df1, df2, on='common\_column')

**Handling Relationships Between Datasets:**

* **Joins:** Inner, outer, left, and right joins.
  + **Example:**

python

Copy code

df\_inner = pd.merge(df1, df2, how='inner', on='key')

#### 5. Hands-On Exercises

**Performing Data Cleaning and Manipulation Tasks:**

* **Exercise 1:** Load a dataset, handle missing values, and perform data cleaning.
* **Exercise 2:** Aggregate data by specific columns and calculate summary statistics.

**Analyzing Real-World Datasets Using pandas:**

* **Exercise 3:** Read a real-world dataset, filter and query data, and create various visualizations using pandas.

#### Activity

* **Project:** Analyze a provided dataset, perform data cleaning, transformation, and aggregation. Prepare a summary report based on your findings and present the results using visualizations.

#### Reading Materials

* **Article:** "Data Manipulation with Pandas" – An introduction to key data manipulation techniques in pandas.
* **Book Chapter:** "Data Cleaning and Analysis with Pandas" from "Python for Data Analysis" by Wes McKinney – In-depth exploration of data cleaning and manipulation techniques.

# Week 6: Data Visualization with matplotlib and seaborn

**Overview:** This week focuses on data visualization using the matplotlib and seaborn libraries. Students will learn how to create various types of visualizations, customize them for clarity and impact, and use these visualizations to communicate insights effectively.

**Topics Covered:**

1. **Introduction to** matplotlib
2. **Advanced Plotting with** matplotlib
3. **Introduction to** seaborn
4. **Visualizing Complex Data**
5. **Hands-On Exercises**

#### 1. Introduction to matplotlib

**Creating Basic Plots:**

* **Line Plots:** Basic line plots to show trends over time or categories.
  + **Example:**

python

Copy code

import matplotlib.pyplot as plt

plt.plot(x, y)

plt.title('Line Plot')

plt.xlabel('X-axis')

plt.ylabel('Y-axis')

plt.show()

* **Bar Plots:** Useful for comparing quantities across categories.
  + **Example:**

python

Copy code

plt.bar(categories, values)

plt.title('Bar Plot')

plt.xlabel('Categories')

plt.ylabel('Values')

plt.show()

* **Scatter Plots:** To visualize relationships between two variables.
  + **Example:**

python

Copy code

plt.scatter(x, y)

plt.title('Scatter Plot')

plt.xlabel('X-axis')

plt.ylabel('Y-axis')

plt.show()

**Customizing Plots:**

* **Titles, Labels, and Legends:** Adding titles, axis labels, and legends for better readability.
  + **Example:**

python

Copy code

plt.plot(x, y)

plt.title('Customized Plot')

plt.xlabel('X-axis Label')

plt.ylabel('Y-axis Label')

plt.legend(['Data Series'])

plt.show()

#### 2. Advanced Plotting with matplotlib

**Subplots and Multiple Plots:**

* **Creating Subplots:** Display multiple plots in a single figure using subplot().
  + **Example:**

python

Copy code

fig, (ax1, ax2) = plt.subplots(1, 2)

ax1.plot(x, y1)

ax2.bar(categories, values)

* **Multiple Plot Customizations:** Customizing each subplot individually.
  + **Example:**

python

Copy code

ax1.set\_title('Line Plot')

ax2.set\_title('Bar Plot')

**Customizing Plot Aesthetics:**

* **Styling:** Using styles and color maps to improve visual appeal.
  + **Example:**

python

Copy code

plt.style.use('ggplot')

plt.plot(x, y, color='green', linestyle='--')

#### 3. Introduction to seaborn

**Creating Statistical Plots:**

* **Histograms:** Visualize distributions of a dataset.
  + **Example:**

python

Copy code

import seaborn as sns

sns.histplot(data, bins=30)

* **Box Plots:** Show distributions and detect outliers.
  + **Example:**

python

Copy code

sns.boxplot(x='category', y='value', data=df)

* **Pair Plots:** Visualize relationships between multiple variables.
  + **Example:**

python

Copy code

sns.pairplot(df)

**Customizing Visualizations with seaborn:**

* **Themes and Color Palettes:** Enhancing plots with predefined themes and color palettes.
  + **Example:**

python

Copy code

sns.set\_theme(style="whitegrid")

sns.barplot(x='category', y='value', data=df, palette='pastel')

#### 4. Visualizing Complex Data

**Plotting Time Series Data:**

* **Creating Time Series Plots:** Visualizing data over time to observe trends and patterns.
  + **Example:**

python

Copy code

plt.plot(time\_series\_data.index, time\_series\_data.values)

**Geographical Data Visualization:**

* **Basic Maps:** Plotting data on geographical maps (optional, if applicable).
  + **Example:** Using libraries like folium or geopandas.

**Interactive Visualization Libraries (Optional):**

* **Interactive Plots:** Exploring libraries like plotly for interactive and dynamic visualizations.

#### 5. Hands-On Exercises

**Creating and Customizing Plots:**

* **Exercise 1:** Create various types of plots using matplotlib and customize them with titles, labels, and legends.
* **Exercise 2:** Use seaborn to generate statistical plots and apply different themes and color palettes.

**Visualizing Datasets:**

* **Exercise 3:** Visualize a provided dataset using both matplotlib and seaborn. Include a variety of plot types and customization options to effectively communicate insights from the data.

#### Activity

* **Project:** Choose a dataset and create a series of visualizations using matplotlib and seaborn. Present your visualizations with an explanation of the insights gained and how the visualizations help in understanding the data.

#### Reading Materials

* **Article:** "A Comprehensive Guide to Data Visualization with matplotlib and seaborn" – Overview of visualization techniques and best practices.
* **Book Chapter:** "Data Visualization with matplotlib and seaborn" from "Python Data Science Handbook" by Jake VanderPlas – Detailed guide on using these libraries for effective data visualization.

# Week 7: Machine Learning with scikit-learn

**Overview:** This week provides an introduction to machine learning concepts and their implementation using the scikit-learn library. Students will learn how to build, evaluate, and refine machine learning models, and apply these techniques to various data analysis tasks.

**Topics Covered:**

1. **Introduction to Machine Learning**
2. **Supervised Learning with** scikit-learn
3. **Unsupervised Learning with** scikit-learn
4. **Model Tuning and Evaluation**
5. **Hands-On Exercises**

#### Overview1. Introduction to Machine Learning

**Overview of Supervised and Unsupervised Learning:**

* **Supervised Learning:** Involves training a model on labeled data to predict outcomes or classify data into predefined categories.
  + **Example Tasks:** Regression (predicting continuous values) and Classification (categorizing data into classes).
* **Unsupervised Learning:** Involves finding patterns or structures in unlabeled data without predefined categories.
  + **Example Tasks:** Clustering (grouping similar data points) and Dimensionality Reduction (reducing the number of features).

**Key Concepts and Terminology:**

* **Features and Labels:** Features are input variables used by the model, while labels are the outcomes or target variables.
* **Training and Testing Data:** Data is split into training sets to build the model and testing sets to evaluate its performance.
* **Overfitting and Underfitting:** Overfitting occurs when a model learns too much from the training data, while underfitting occurs when it fails to capture underlying patterns.

#### 2. Supervised Learning with scikit-learn

**Implementing Regression Models:**

* **Linear Regression:** Predicts a continuous outcome based on one or more input features.
  + **Example Code:**

python

Copy code

from sklearn.linear\_model import LinearRegression

model = LinearRegression()

model.fit(X\_train, y\_train)

predictions = model.predict(X\_test)

**Implementing Classification Models:**

* **Logistic Regression:** Used for binary classification tasks.
  + **Example Code:**

python

Copy code

from sklearn.linear\_model import LogisticRegression

model = LogisticRegression()

model.fit(X\_train, y\_train)

predictions = model.predict(X\_test)

**Evaluating Model Performance:**

* **Metrics:** Accuracy, precision, recall, F1 score for classification; Mean Squared Error (MSE), R-squared for regression.
  + **Example Code:**

python

Copy code

from sklearn.metrics import accuracy\_score, mean\_squared\_error

accuracy = accuracy\_score(y\_test, predictions)

mse = mean\_squared\_error(y\_test, predictions)

#### 3. Unsupervised Learning with scikit-learn

**Implementing Clustering Algorithms:**

* **K-Means Clustering:** Groups data points into K clusters based on feature similarity.
  + **Example Code:**

python

Copy code

from sklearn.cluster import KMeans

model = KMeans(n\_clusters=3)

model.fit(X)

labels = model.predict(X)

**Dimensionality Reduction:**

* **Principal Component Analysis (PCA):** Reduces the number of features while retaining most of the variance in the data.
  + **Example Code:**

python

Copy code

from sklearn.decomposition import PCA

pca = PCA(n\_components=2)

reduced\_data = pca.fit\_transform(X)

**Analyzing Clustering Results:**

* **Visualizing Clusters:** Use visualization tools to interpret clustering results.
  + **Example Code:**

python

Copy code

import matplotlib.pyplot as plt

plt.scatter(reduced\_data[:, 0], reduced\_data[:, 1], c=labels)

#### 4. Model Tuning and Evaluation

**Hyperparameter Tuning:**

* **Grid Search:** Finds the best combination of hyperparameters for a model.
  + **Example Code:**

python

Copy code

from sklearn.model\_selection import GridSearchCV

parameters = {'n\_neighbors': [3, 5, 7]}

grid\_search = GridSearchCV(KNeighborsClassifier(), parameters)

grid\_search.fit(X\_train, y\_train)

best\_params = grid\_search.best\_params\_

**Cross-Validation:**

* **K-Fold Cross-Validation:** Evaluates the model’s performance on different subsets of the data.
  + **Example Code:**

python

Copy code

from sklearn.model\_selection import cross\_val\_score

scores = cross\_val\_score(model, X, y, cv=5)

**Model Selection and Evaluation Metrics:**

* **Choosing the Best Model:** Compare performance metrics to select the best model.
  + **Example Code:**

python

Copy code

from sklearn.metrics import classification\_report

print(classification\_report(y\_test, predictions))

#### 5. Hands-On Exercises

**Building and Evaluating Machine Learning Models:**

* **Exercise 1:** Implement and evaluate regression and classification models on provided datasets.
* **Exercise 2:** Apply clustering and dimensionality reduction techniques to real-world datasets.

**Applying Machine Learning Techniques:**

* **Exercise 3:** Use scikit-learn to solve a practical problem, such as predicting outcomes or clustering data, and evaluate the effectiveness of your models.

#### Activity

* **Project:** Select a dataset and apply various machine learning techniques using scikit-learn. Build models for both supervised and unsupervised learning tasks, tune the models, and present your findings along with the evaluation results.

#### Reading Materials

* **Article:** "A Comprehensive Guide to Machine Learning with scikit-learn" – Overview of key machine learning concepts and practical implementation.
* **Book Chapter:** "Machine Learning with scikit-learn" from "Python Machine Learning" by Sebastian Raschka and Vahid Mirjalili – Detailed guide on machine learning techniques and best practices.

# Week 8: Advanced Topics in Python

**Overview:** In the final week, students will explore advanced Python programming topics, including Object-Oriented Programming (OOP), Functional Programming, and Asynchronous Programming. Additionally, students will learn how to work with JSON data and APIs. The week will culminate in a Capstone Project, allowing students to integrate and apply their knowledge from the entire course.

**Topics Covered:**

1. **Object-Oriented Programming (OOP)**
2. **Functional Programming**
3. **Asynchronous Programming with** asyncio
4. **Handling JSON and APIs**
5. **Capstone Project**
6. **Hands-On Exercises**

#### 1. Object-Oriented Programming (OOP)

**Classes and Objects:**

* **Classes:** Blueprints for creating objects. Define attributes (data) and methods (functions).
  + **Example Code:**

python

Copy code

classCar:

def\_\_init\_\_(self, make, model):

self.make = make

self.model = model

defdisplay\_info(self):

returnf"{self.make} {self.model}"

* **Objects:** Instances of classes.
  + **Example Code:**

python

Copy code

my\_car = Car("Toyota", "Corolla")

print(my\_car.display\_info())

**Inheritance and Polymorphism:**

* **Inheritance:** Mechanism to create a new class using details from an existing class.
  + **Example Code:**

python

Copy code

classElectricCar(Car):

def\_\_init\_\_(self, make, model, battery\_size=75):

super().\_\_init\_\_(make, model)

self.battery\_size = battery\_size

* **Polymorphism:** The ability to use a unified interface for different data types.
  + **Example Code:**

python

Copy code

defprint\_car\_info(car):

print(car.display\_info())

**Encapsulation and Abstraction:**

* **Encapsulation:** Restrict access to some of an object's components.
  + **Example Code:**

python

Copy code

classCar:

def\_\_init\_\_(self, make, model):

self.\_\_make = make # Private attribute

* **Abstraction:** Hiding complex implementation details and showing only the necessary features.
  + **Example Code:**

python

Copy code

from abc import ABC, abstractmethod

classShape(ABC):

@abstractmethoddefarea(self):

pass

#### 2. Functional Programming

**Pure Functions and Immutability:**

* **Pure Functions:** Functions where the output depends only on the input values and have no side effects.
  + **Example Code:**

python

Copy code

defadd(x, y):

return x + y

* **Immutability:** Data cannot be modified after it is created.
  + **Example Code:**

python

Copy code

# Tuples are immutable

my\_tuple = (1, 2, 3)

**Higher-Order Functions:**

* \*\*Functions that take other functions as arguments or return functions as results.
  + **Example Code:**

python

Copy code

defapply\_function(func, value):

return func(value)

result = apply\_function(lambda x: x\*\*2, 4)

**Using Lambda Functions and Map/Reduce Operations:**

* **Lambda Functions:** Small anonymous functions defined with the lambda keyword.
  + **Example Code:**

python

Copy code

square = lambda x: x\*\*2

* **Map/Reduce Operations:**
  + **Map:** Apply a function to all items in an input list.
    - **Example Code:**

python

Copy code

numbers = [1, 2, 3]

squared = list(map(lambda x: x\*\*2, numbers))

* + **Reduce:** Apply a function of two arguments cumulatively to the items of a sequence.
    - **Example Code:**

python

Copy code

from functools import reduce

sum\_result = reduce(lambda x, y: x + y, numbers)

#### 3. Asynchronous Programming with asyncio

**Coroutines and Event Loops:**

* **Coroutines:** Functions that can pause execution and yield control to the event loop.
  + **Example Code:**

python

Copy code

import asyncio

asyncdefhello():

print("Hello")

await asyncio.sleep(1)

print("World")

* **Event Loops:** Manage the execution of asynchronous tasks.
  + **Example Code:**

python

Copy code

asyncio.run(hello())

**Implementing Asynchronous Tasks:**

* **Handling Multiple Tasks Concurrently:**
  + **Example Code:**

python

Copy code

asyncdefmain():

await asyncio.gather(hello(), hello())

asyncio.run(main())

#### 4. Handling JSON and APIs

**Parsing JSON Data:**

* **Reading JSON Files and Parsing JSON Strings:**
  + **Example Code:**

python

Copy code

import json

json\_data = '{"name": "John", "age": 30}'

data = json.loads(json\_data)

**Making API Requests and Handling Responses:**

* **Using the requests Library:**
  + **Example Code:**

python

Copy code

import requests

response = requests.get('https://api.example.com/data')

data = response.json()

#### 5. Capstone Project

**Integrating Concepts:**

* **Developing a Comprehensive Python Application:**
  + Students will design and implement a Python application that integrates OOP, functional programming, asynchronous programming, and data handling.

**Project Requirements:**

* **Incorporate Advanced Programming Techniques:** Use classes, functions, and async tasks.
* **Handle JSON Data and APIs:** Fetch, process, and display data from web APIs.
* **Demonstrate Problem-Solving Skills:** Address a real-world problem or create a practical tool.

#### 6. Hands-On Exercises

**Implementing Advanced Programming Techniques:**

* **Exercise 1:** Implement classes, functions, and async operations in a small project.
* **Exercise 2:** Work with JSON data and make API requests to integrate with a Python application.

**Completing and Presenting the Capstone Project:**

* **Exercise 3:** Finalize and present the Capstone Project, demonstrating the use of all learned concepts and techniques.

#### Activity

* **Capstone Project Presentation:** Present your Capstone Project to the class, explaining your design choices, the challenges you faced, and how you solved them. Discuss how you applied advanced programming techniques and integrated various Python features.

#### Reading Materials

* **Article:** "Advanced Python Programming Techniques" – Covers advanced programming concepts and their applications.
* **Book Chapter:** "Mastering Object-Oriented Programming in Python" from "Fluent Python" by Luciano Ramalho – Detailed insights into advanced OOP concepts in Python.