

**DATA SCIENCE**  
 **CERTIFICATE**  
  
This is to certify that Ms. Alishba D/O Karim Bux has successfully completed Artificial Intelligence lab work during spring semester 2025.  
  
  
  
Lab Course Supervisor: Furqan Ahmed Abbasi

Signature:

[Lab 1. Python Basics1.1 Data Types in Python 2](#_Toc601143585)

[1.2 Operators in Python 2](#_Toc1170426440)

[1.3 Input and Output 4](#_Toc1155854151)

[Labs Task 4](#_Toc2116009068)

[Lab 2. Lists and Strings 5](#_Toc1712456880)

[2.1 Lists in Python 5](#_Toc1119759994)

[2.2 Strings in Python 6](#_Toc642489203)

[Lab Task 7](#_Toc1354706848)

[Lab 3. Loops and Control Flow in Python 8](#_Toc64798036)

[3.1 Loop Control Statements 8](#_Toc423007248)

[3.2 Loop Examples 10](#_Toc1654889399)

[3.3 Useful List Functions with Loops 12](#_Toc1549892028)

[Lab Task 13](#_Toc1542807539)

[4.1 Nested Lists (Matrices) 15](#_Toc729326413)

[4.2 List Comprehension 16](#_Toc2025472828)

[4.3 Removing Duplicates from a List 17](#_Toc1366478418)

[Lab Task 18](#_Toc321634791)

[Lab 05. Tuple 18](#_Toc1530654899)

[Lab Task 23](#_Toc1250999290)

[Lab 6: Conditions in Python 25](#_Toc1937351026)

[What is a Condition? 25](#_Toc1100697953)

[Lab Task 27](#_Toc684994174)

[Lab 7: Loops in Python 29](#_Toc224013071)

[What is a Loop? 30](#_Toc1151943603)

[Types of Loops: 30](#_Toc2054489685)

[For Loop: 30](#_Toc2132401607)

[While Loop: 30](#_Toc2019905513)

[Lab Task 31](#_Toc1396335739)

[What is OOP? 33](#_Toc1930148228)

[1. What is a Class? 34](#_Toc1366652524)

[2. What is an Object? 34](#_Toc793184465)

[3. What is the \_\_init\_\_() Method? 34](#_Toc1689036225)

[4. What is a Method? 35](#_Toc1970260865)

[5. Attributes and Methods Summary 36](#_Toc582125349)

[6. Inheritance 36](#_Toc1862037855)

[7. Encapsulation 37](#_Toc265763459)

[8. Polymorphism 38](#_Toc446659725)

[1. NumPy – Numerical Python 44](#_Toc1138900198)

[2. Pandas – Data Handling Library 45](#_Toc246854006)

[3. Matplotlib – Drawing Graphs 46](#_Toc1125533009)

[Lab Task 47](#_Toc873298557)

[Introduction to Scikit-learn (Machine Learning Basics) 50](#_Toc875484816)

[List of Libraries which is used, 50](#_Toc962638782)

[Lab Task: 50](#_Toc412673457)

[Lab Task: 50](#_Toc1457029326)

[Lab Task: 50](#_Toc251397194)

[Lab Objectives 50](#_Toc2093517011)

[Lab Task 50](#_Toc582482)

[Task 1: 50](#_Toc379146551)

[Task 2: 50](#_Toc293915160)

[Task 3: 50](#_Toc1519949829)

# **Lab 1. Python Basics** 1.1 Data Types in Python

In Python, data types define the kind of value a variable can hold. Python is a dynamically typed language, meaning you don’t need to declare the data type explicitly. It automatically detects the type of data when a value is assigned to a variable.

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Description** | **Example** |
| int | Integer: whole number values | 10, -3, 0 |
| float | Floating-point (decimal) numbers | 3.14, -0.5 |
| str | String: sequence of characters | "Python", 'Hello' |
| bool | Boolean values | True, False |

**Type Conversion**

To convert one data type into another:

num\_str = "10"

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

pi = str(3.14) # Converts float to string

Use int(), float(), str(), or bool() as needed.

### 1.2 Operators in Python

**Arithmetic Operators**

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Meaning** | **Example** | **Result** |
| + | Addition | 2 + 3 | 5 |
| - | Subtraction | 7 - 4 | 3 |
| \* | Multiplication | 5 \* 3 | 15 |
| / | Division | 6 / 2 | 3.0 |
| // | Floor Division | 7 // 2 | 3 |
| % | Modulus (Remainder) | 8 % 3 | 2 |
| \*\* | Exponentiation | 2 \*\* 3 | 8 |

**Comparison Operators**

These return True or False:

a = 5

b = 3

print(a > b) # True

print(a == b) # False

**Logical Operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| and | Both conditions must be true | True and False → False |
| or | At least one condition must be true | True or False → True |
| not | Reverses the boolean value | not True → False |

#### 1.3 Input and Output

**input() Function**

name = input("Enter your name: ")

print("Hello, " + name)

Note: Input is always taken as a string. Convert when necessary:

age = int(input("Enter your age: "))

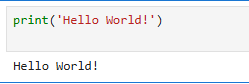
**Formatted Output (f-Strings)**

name = "Ali"

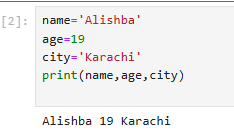
print(f"Welcome, {name}!") # Output: Welcome, Ali!

##### **Labs Task**

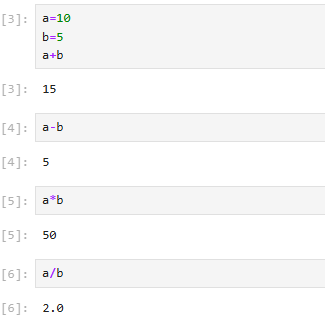
1. Write a Python program to print "Hello, World!".



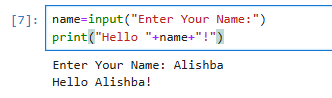
1. Declare variables for name, age, and city, then print them.



3.Perform basic arithmetic operations (+, -, \*, /) and print results.



4.Take user input for their name using input() and greet them.



# **Lab 2. Lists and Strings**

## 2.1 Lists in Python

A list is an ordered collection of elements enclosed in square brackets ([ ]). Lists are used to store multiple items in a single variable.

**Characteristics of Lists:**

* Mutable: You can change, add, or remove elements.
* Indexed: Elements have positions starting from 0.
* Heterogeneous: A list can contain different types of data (e.g., integers, strings, booleans).

|  |  |  |
| --- | --- | --- |
| **Method** | **Description** | **Example** |
| index() | Get index of item | fruits.index("banana") → 1 |
| insert() | Insert at position | fruits.insert(1, "grape") |
| pop() | Remove by index | fruits.pop(0) |
| sort() | Sort list | fruits.sort() |
| reverse() | Reverse list | fruits.reverse() |

**List Slicing**

numbers = [1, 2, 3, 4, 5]

print(numbers[1:4]) # [2, 3, 4]

Syntax: list[start:stop:step]

### 2.2 Strings in Python

A string is a sequence of characters enclosed in single quotes (' ') or double quotes (" "). Strings are one of the most commonly used data types for handling textual data.

**Characteristics of Strings:**

* Immutable: Once a string is created, it cannot be modified.
* Indexed: Each character has a position (starting from 0).
* Iterable: You can loop through each character in a string.

|  |  |  |
| --- | --- | --- |
| **Method** | **Purpose** | **Example** |
| len() | Length of string | len("Python") → 6 |
| strip() | Removes spaces | " Hello ".strip() → "Hello" |
| split() | Converts to list | "a b c".split() → ['a', 'b', 'c'] |
| join() | Joins list to string | " ".join(['a','b']) → "a b" |

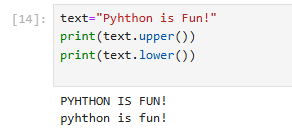
text = "Python is great"

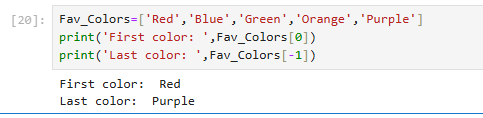
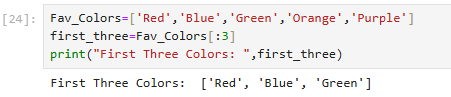
words = text.split()

sentence = " ".join(words)

#### Lab Task

1. Store "Python is fun!" in a variable, print it in uppercase and lowercase.



1. Create a list of 5 favorite colors, then print the first and last color.  
   
2. Use slicing to print the first three colors from the list.  
   
3. Add a new color to the list using append() and remove one using remove().  
   

# **Lab 3. Loops and Control Flow in Python**

Loops are essential in programming for performing repetitive tasks efficiently. Python provides two primary loop types—for and while—along with special control statements (break, continue, and else) to manage loop execution.

## 3.1 Loop Control Statements

**break**

* Immediately terminates the loop, regardless of the loop condition.
* Commonly used when a certain condition is met and there's no need to continue looping.

**Example**:

python

CopyEdit

for i in range(10):

if i == 5:

break

print(i)

# Output: 0 1 2 3 4

**continue**

* Skips the current iteration and moves to the next cycle of the loop.
* Useful for skipping specific values or conditions.

**Example**:

python

CopyEdit

for i in range(5):

if i == 2:

continue

print(i)

# Output: 0 1 3 4

**else with Loops**

* The else block executes after the loop finishes normally (without break).
* Often used for search operations.

**Example**:

python

CopyEdit

for i in range(5):

print(i)

else:

print("Loop completed without break")

**With break**:

python

CopyEdit

for i in range(5):

if i == 3:

break

print(i)

else:

print("Will not run") # This will not execute

### 3.2 Loop Examples

**for Loop**

* Used to iterate over a sequence such as a list, tuple, string, or range of numbers.
* Syntax:

python

CopyEdit

for variable in sequence:

# code block

**Example**:

python

CopyEdit

for i in range(5):

print("Iteration:", i)

**Output**:

makefile

CopyEdit

Iteration: 0

Iteration: 1

Iteration: 2

Iteration: 3

Iteration: 4

**while Loop**

* Repeats a block of code as long as a condition is True.
* Syntax:

python

CopyEdit

while condition:

# code block

**Example**:

python

CopyEdit

count = 0

while count < 5:

print("Count is", count)

count += 1

**Output**:

csharp

CopyEdit

Count is 0

Count is 1

Count is 2

Count is 3

Count is 4

#### 3.3 Useful List Functions with Loops

**sum()**

* Returns the total sum of numeric elements in a list.

python

CopyEdit

numbers = [3, 1, 4]

total = sum(numbers)

print("Sum:", total) # Output: Sum: 8

**sorted()**

* Returns a new sorted list (original list remains unchanged).
* Can be used in ascending or descending order.

python

CopyEdit

numbers = [3, 1, 4]

ascending = sorted(numbers) # [1, 3, 4]

descending = sorted(numbers, reverse=True) # [4, 3, 1]

print("Ascending:", ascending)

print("Descending:", descending)

**Using Functions Inside Loops**

**Example**: Find the sum of only even numbers in a list.

python

CopyEdit

numbers = [1, 2, 3, 4, 5, 6]

even\_sum = 0

for num in numbers:

if num % 2 == 0:

even\_sum += num

print("Sum of even numbers:", even\_sum)

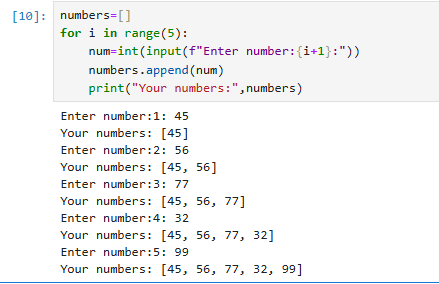
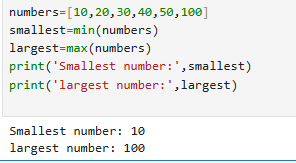
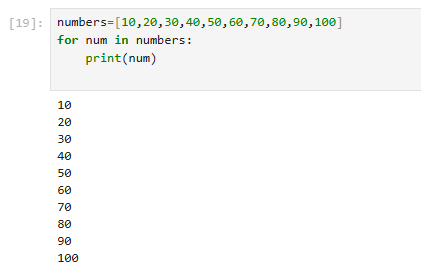
**Output**:

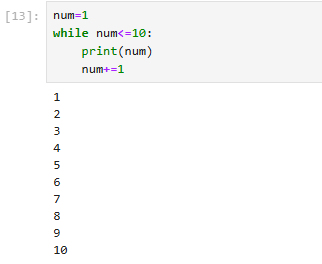
yaml

CopyEdit

Sum of even numbers: 12

##### Lab Task

1. Ask the user for 5 numbers and store them in a list.  
   
2. Find and print the largest and smallest number in the list using max() and min().  
   
3. Use a for loop to print each item in the list.  
   
4. Use a while loop to print numbers from 1 to 10.

  
  
**Lab 4. Advanced List**

Python provides advanced tools to work with collections of data, such as nested lists, list comprehension, and methods to remove duplicates efficiently. These tools help simplify data processing tasks and improve code readability.

## 4.1 Nested Lists (Matrices)

A nested list is a list that contains other lists as its elements. This structure is commonly used to represent matrices or 2D data grids.

**Syntax:**

python

CopyEdit

matrix = [

[1, 2],

[3, 4]

]

* matrix[0][1] refers to row 0, column 1, which outputs 2.
* Access elements using double indexing: [row][column].

**Use Cases:**

* Representing tables, grids, spreadsheets, and graph data.
* Useful in data science, AI (images as 2D arrays), and numerical computations.

**Example:**

python

CopyEdit

matrix = [

[10, 20, 30],

[40, 50, 60],

[70, 80, 90]

]

print(matrix[1][2]) # Output: 60

### 4.2 List Comprehension

List comprehension provides a concise way to create new lists from existing sequences, using a single line of readable syntax.

Basic Format:

python

CopyEdit

[expression for item in iterable]

Without Condition (Simple Case):

python

CopyEdit

squares = [x\*\*2 for x in range(5)]

# Output: [0, 1, 4, 9, 16]

With Condition (Filtered Case):

python

CopyEdit

evens = [x for x in range(10) if x % 2 == 0]

# Output: [0, 2, 4, 6, 8]

Advanced Example:

Create a list of words with more than 3 letters:

python

CopyEdit

words = ["the", "sky", "is", "beautiful"]

long\_words = [word for word in words if len(word) > 3]

# Output: ['beautiful']

#### 4.3 Removing Duplicates from a List

Duplicate values can be removed in multiple ways in Python.

Using set() (Unordered):

A set is a collection that stores only unique elements.

python

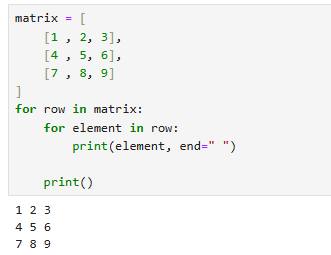
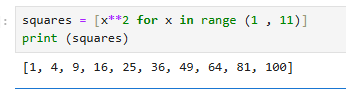
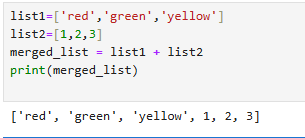
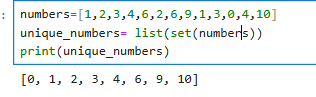
CopyEdit

data = [1, 2, 2, 3, 3, 4]

unique = list(set(data))

# Output: [1, 2, 3, 4] (order may vary)

##### Lab Task

1. Create a 3x3 matrix as a nested list and print its elements.  
   
2. Use list comprehension to create a list of squares from 1 to 10.  
   
3. Merge two lists using the + operator and print the result.  
   
4. Write a program to remove duplicates from a list.  
   

**Lab 05. Tuple**  
Tuples are used to store multiple items in a single variable.  
Tuple is one of four built-in data types in Python used to store collections of data. The other three are List, Set, and Dictionary, all with different qualities and usage.  
A tuple is a collection which is ordered and unchangeable.

**Example – Create a Tuple:**  
thistuple = ("apple", "banana", "cherry")  
print(thistuple)  
Output:  
('apple', 'banana', 'cherry')

**Tuple Items**

* Ordered
* Unchangeable
* Allow duplicate values
* Tuple items are indexed: index [0], index [1], etc.

**Ordered**  
Items have a defined order that will not change.

**Unchangeable**  
Items cannot be added, changed, or removed after creation.

**Allow Duplicates**  
Tuples allow duplicate values.

**Example – Duplicates in Tuple:**  
thistuple = ("apple", "banana", "cherry", "apple", "cherry")  
print(thistuple)  
Output:  
('apple', 'banana', 'cherry', 'apple', 'cherry')

**Tuple Length**  
Use len() to determine the number of items.

**Example:**  
thistuple = ("apple", "banana", "cherry")  
print(len(thistuple))  
Output:  
3

**Create Tuple with One Item**  
Use a comma after the single item.

**Example – One Item Tuple:**  
thistuple = ("apple",)  
print(type(thistuple))  
Output:  
<class 'tuple'>

**Not a Tuple:**  
thistuple = ("apple")  
print(type(thistuple))  
Output:  
<class 'str'>

**Tuple Items – Data Types**  
Tuples can hold any data type.

**Example – Different Data Types:**  
tuple1 = ("apple", "banana", "cherry")  
tuple2 = (1, 5, 7, 9, 3)  
tuple3 = (True, False, False)  
print(tuple1)  
print(tuple2)  
print(tuple3)  
Output:  
('apple', 'banana', 'cherry')  
(1, 5, 7, 9, 3)  
(True, False, False)

**Example – Mixed Data Types:**  
tuple1 = ("abc", 34, True, 40, "male")  
print(tuple1)  
Output:  
('abc', 34, True, 40, 'male')

**type() Function**  
Check the data type of a tuple using type().

**Example:**  
mytuple = ("apple", "banana", "cherry")  
print(type(mytuple))  
Output:  
<class 'tuple'>

**The tuple() Constructor**  
You can also create tuples using the constructor.

**Example – Using tuple():**  
thistuple = tuple(("apple", "banana", "cherry"))  
print(thistuple)  
Output:  
('apple', 'banana', 'cherry')

**Python Collections (Arrays)**

* **List:** Ordered, changeable, allows duplicates.
* **Tuple:** Ordered, unchangeable, allows duplicates.
* **Set:** Unordered, unchangeable\*, unindexed, no duplicates.
* **Dictionary:** Ordered\*\*, changeable, no duplicates.

**Range of Indexes**  
Use slicing to access a range of items.

**Example:**  
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")  
print(thistuple[2:5])  
Output:  
('cherry', 'orange', 'kiwi')

**Example:**  
print(thistuple[:4])  
Output:  
('apple', 'banana', 'cherry', 'orange')

**Example:**  
print(thistuple[2:])  
Output:  
('cherry', 'orange', 'kiwi', 'melon', 'mango')

**Example – Negative Indexing:**  
print(thistuple[-4:-1])  
Output:  
('orange', 'kiwi', 'melon')

**Item Existence**  
Check if an item exists using in keyword.

**Example:**  
thistuple = ("apple", "banana", "cherry")  
if "apple" in thistuple:  
print("Yes, 'apple' is in the fruits tuple")  
Output:  
Yes, 'apple' is in the fruits tuple

**Unpacking a Tuple**  
Assign tuple items to variables.

**Example – Unpacking:**  
fruits = ("apple", "banana", "cherry")  
(green, yellow, red) = fruits  
print(green)  
print(yellow)  
print(red)  
Output:  
apple  
banana  
cherry

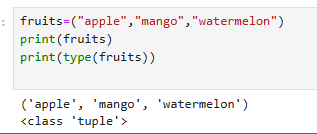
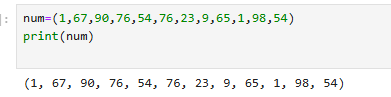
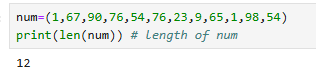
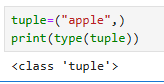
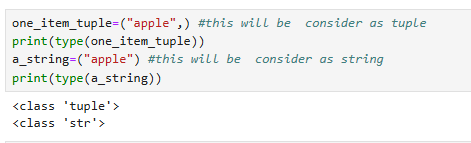
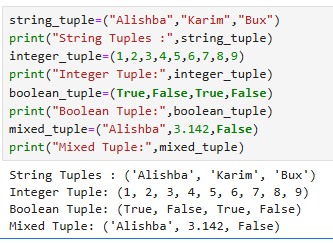
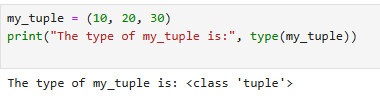
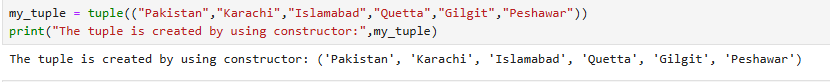
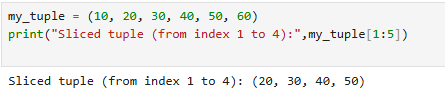
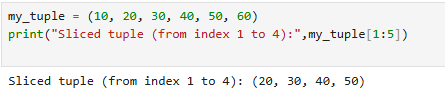
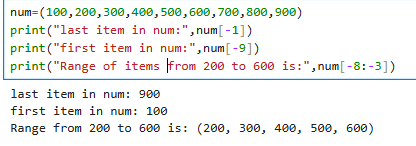
**Loop Through a Tuple**  
Use a for loop.

**Example – For Loop:**  
thistuple = ("apple", "banana", "cherry")  
for x in thistuple:  
print(x)  
Output:  
apple  
banana  
cherry

**Loop Through Index Numbers**  
Use range() and len().

**Example:**  
thistuple = ("apple", "banana", "cherry")  
for i in range(len(thistuple)):  
print(thistuple[i])  
Output:  
apple  
banana  
cherry

## Lab Task

1. Create a tuple containing three fruit names and display it.  
   
2. Create a tuple that includes duplicate values and print it.  
     
   
3. Find and print the length of a given tuple.  
   
4. Create a tuple with only one item and verify that it is recognized as a tuple.  
   
5. Demonstrate the difference between a one-item tuple and a string.  
   
6. Create tuples with:
   * String data
   * Integer data
   * Boolean data
   * Mixed data types  
     Print each tuple.  
     
7. Write a program to check the data type of a given tuple.  
   
8. Create a tuple using the tuple() constructor and print it.  
   
9. Access and print a range of items from a tuple using slicing.  
   
10. Use negative indexing to access and print specific items from a tuple.  
      
    

**Lab 6: Conditions in Python**

## What is a Condition?

A condition in programming allows the computer to make decisions based on logical expressions. In Python, this is known as decision making. Conditions control the flow of a program by allowing different code blocks to execute depending on whether a certain condition evaluates to True or False.

**Keywords Used:**

if: Executes a block of code if the condition is True.

elif: Checks another condition if the previous condition was False.

else: Executes if all conditions are False.  
**Syntax:**

if condition:

# code to execute if condition is True

elif another\_condition:

# code to execute if another\_condition is True

else:

# code to execute if all above conditions are False

Characteristics:

Conditions are used for branching or decision making.

Conditional blocks must be indented properly in Python.

Examples:

Simple if Condition

marks = 75

if marks >= 50:

print("You Passed")

Output:

You Passed

if-else Condition

marks = 40

if marks >= 50:

print("You Passed")

else:

print("You Failed")

Output:

You Failed

if-elif-else Condition

marks = 85

if marks >= 90:

print("Grade A+")

elif marks >= 80:

print("Grade A")

elif marks >= 70:

print("Grade B")

else:

print("Grade C")

Output:

Grade A

Even or Odd Number

number = 4

if number % 2 == 0:

print("Even Number")

else:

print("Odd Number")

Output:

Even Number

Find Greater Number

a = 10

b = 20

if a > b:

print("A is Greater")

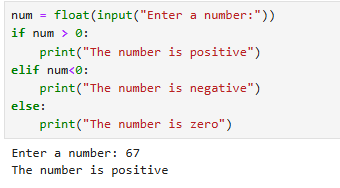
else:

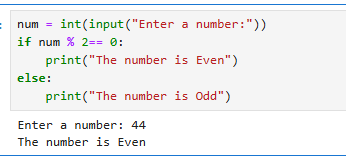
print("B is Greater")

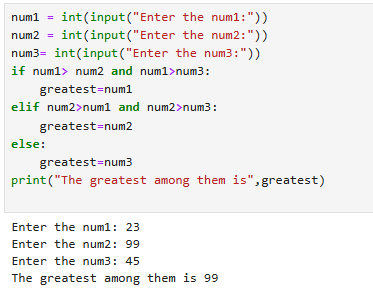
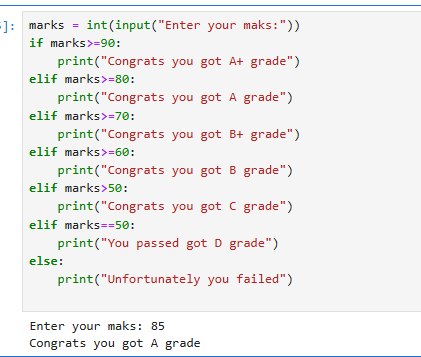
Output:

B is Greater

### Lab Task

1.Write a Python program to check if a number is positive, negative, or zero.  


2.Write a program to determine if a number is even or odd.  


3.Write a program to find the greatest among three numbers.  
  
4.Write a program that assigns grades based on marks entered by the user.  


# **Lab 7: Loops in Python**

## What is a Loop?

A loop in Python is used to execute a block of code repeatedly as long as a certain condition is met. This helps in reducing code redundancy and automating repetitive tasks.

### Types of Loops:

1. for loop – used to iterate over a sequence (like a list, tuple, or string)
2. while loop – used when the number of iterations is not known in advance

### For Loop:

* Best used when the number of iterations is known.
* Commonly used with the range() function.

**Syntax:**

for variable in sequence:

# block of code

### While Loop:

* Used when you want to run a loop until a condition becomes **False**.

**Syntax:**

while condition:

# block of code

**Examples:**

1. **Print numbers 1 to 5 using for loop**

for i in range(1, 6):

print(i)

1. **Print items from a list**

fruits = ["Apple", "Mango", "Banana"]

for fruit in fruits:

print(fruit)

1. **Table of 5**

for i in range(1, 11):

print(5 \* i)

1. **Print numbers 1 to 5 using while loop**

i = 1

while i <= 5:

print(i)

i += 1

1. **Print even numbers 2 to 10**

i = 2

while i <= 10:

print(i)

i += 2

1. **Sum of numbers 1 to 5 using for loop**

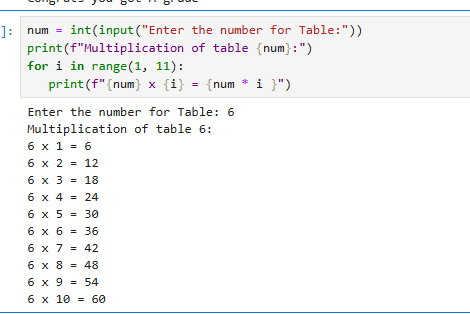
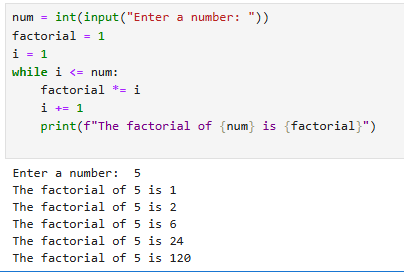
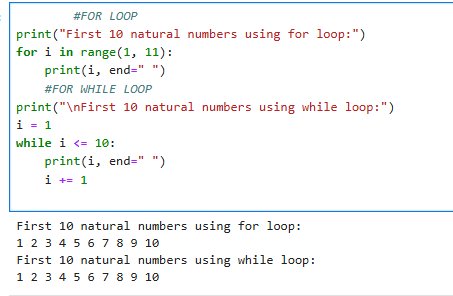
sum = 0

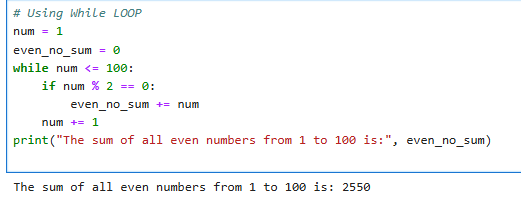
for i in range(1, 6):

sum += i

print("Sum =", sum)

#### **Lab Task**

1. Write a program to print the multiplication table of any number entered by the user.  
   
2. Write a program to calculate factorial of a number using while loop.  
   
3. Write a program to print first 10 natural numbers using both for and while loop.  
   

Write a program to calculate and display the sum of all even numbers from 1 to 100.  
  
  
**Lab 8: Python OOP (Object-Oriented Programming)**

**What is OOP?**  
OOP = Object-Oriented Programming

Object-Oriented Programming (OOP) is a programming method that structures code using objects, which combine data (called attributes) and functions (called methods). Python uses classes to define objects.

Key Concepts:

A class is a blueprint or template.

An object is an instance created from a class.

OOP helps organize code, reuse logic, and make programs easier to manage and scale.

## 1. What is a Class?

Definition:  
A class is a structure used to define how objects will behave and what data they hold.

Syntax:

class Car:

pass

This defines an empty class named Car.

## 2. What is an Object?

Definition:  
An object is a specific instance of a class, created using the class definition.

Example:

my\_car = Car()

Here, my\_car is an object of the class Car.

## 3. What is the \_\_init\_\_() Method?

Definition:  
The \_\_init\_\_() method is a special method used to initialize object properties when an object is created.

Example 1:

class Person:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

p1 = Person("John", 36)

print(p1.name)

print(p1.age)

Output:

John

36

Example 2:

class Car:

def \_\_init\_\_(self, brand, color):

self.brand = brand

self.color = color

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

print(my\_car.brand)

print(my\_car.color)

## 4. What is a Method?

Definition:  
A method is a function defined inside a class that performs actions using class data.

Example:

class Car:

def \_\_init\_\_(self, brand, color):

self.brand = brand

self.color = color

def start(self):

print(f"{self.brand} car is starting.")

my\_car = Car("Honda", "Blue")

my\_car.start()

Output:

Honda car is starting.

## 

## 5. Attributes and Methods Summary

|  |  |  |
| --- | --- | --- |
| Term | Meaning | Example |
| Attribute | Variable in class | self.name |
| Method | Function in class | def show\_info(self): |

Example:

class Student:

def \_\_init\_\_(self, name, grade):

self.name = name

self.grade = grade

def show\_info(self):

print(f"{self.name} is in grade {self.grade}.")

s1 = Student("Ali", "10th")

s1.show\_info()

Output:

Ali is in grade 10th.

### 6. Inheritance

Definition:  
Inheritance allows one class (child) to get properties and methods from another class (parent).

Example:

class Animal:

def sound(self):

print("Animal makes sound")

class Dog(Animal):

def sound(self):

print("Dog barks")

d = Dog()

d.sound()

Output:

Dog barks

#### 7. Encapsulation

Definition:  
Encapsulation means hiding internal object data using private variables and accessing them with methods.

Example:

class Bank:

def \_\_init\_\_(self):

self.\_\_balance = 0 # Private variable

def deposit(self, amount):

self.\_\_balance += amount

def show\_balance(self):

print(f"Balance: {self.\_\_balance}")

acc = Bank()

acc.deposit(500)

acc.show\_balance()

Output:

Balance: 500

##### 8. Polymorphism

Definition:  
Polymorphism allows the same method name to have different behaviors based on the object.

Example:

class Bird:

def sound(self):

print("Tweet")

class Duck:

def sound(self):

print("Quack")

def make\_sound(animal):

animal.sound()

make\_sound(Bird())

make\_sound(Duck())

Output:

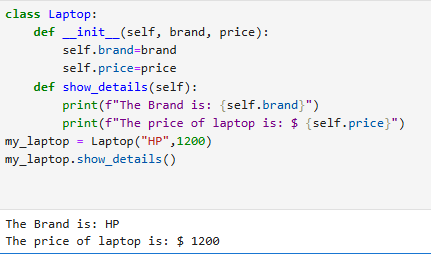
Tweet

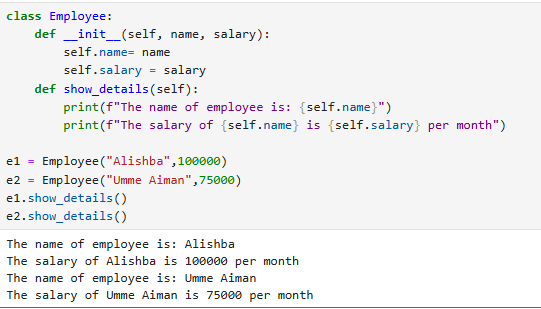
Quack

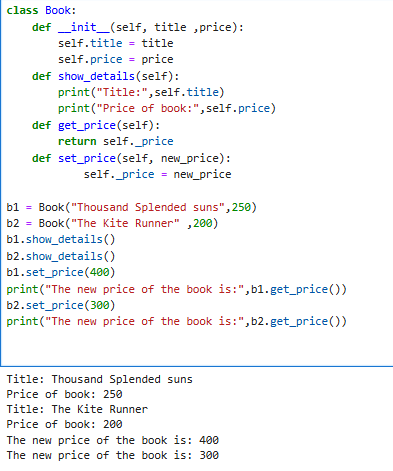
OOP Recap Table

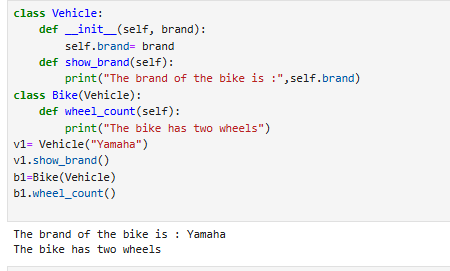
|  |  |  |
| --- | --- | --- |
| Term | Meaning | Example |
| Class | Blueprint/template | class Car: |
| Object | Real object created from class | my\_car = Car() |
| Attribute | Variable inside a class | self.name |
| Method | Function inside a class | def start(self): |
| Constructor | Special method to initialize objects | init() |
| Inheritance | One class uses another's features | class Dog(Animal) |
| Encapsulation | Hiding data inside a class | self.\_\_balance |
| Polymorphism | One method behaves differently | sound() in many classes |

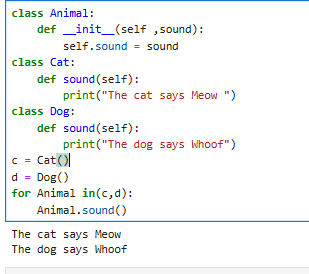
Lab Task

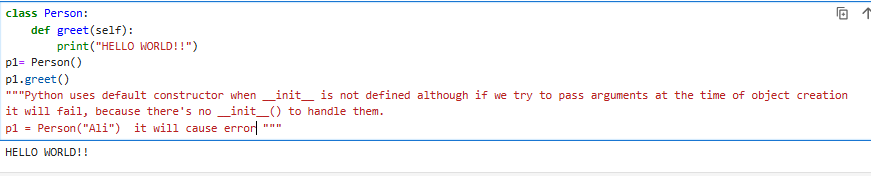
1.Create a class Laptop with attributes brand and price, and a method show\_details() to print them.  


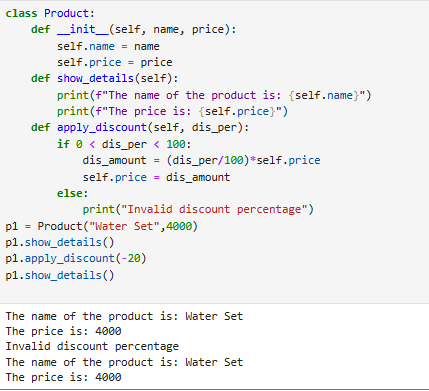
2.Write a class Employee with \_\_init\_\_() method to set name and salary. Create two objects and display their information.  


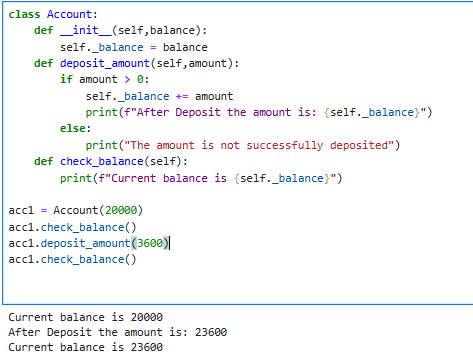
3.Create a class Book and use the concept of encapsulation to protect the price attribute.  


4.Use inheritance to create a class Vehicle and a child class Bike with its own method wheel\_count().  


5.Demonstrate polymorphism using classes Cat and Dog, both having a method sound().  


6.What will happen if you create a class but do not define \_\_init\_\_()? Try it and explain.  


7.Add a method apply\_discount() to a class Product which changes the price based on given discount percentage.  


8.Write a class Account with private balance and public methods deposit() and check\_balance(). Prevent direct access to balance.  
  
  
**Lab 09. NumPy, Pandas & Matplotlib**

**Objective:** Understand and apply Python libraries for numerical computation, data handling, and visualization as a foundation for AI model development.

## 1. NumPy – Numerical Python

* NumPy helps you store and process numbers in arrays (like lists but faster).
* It’s used in AI for fast mathematical operations on data.
* Especially useful when working with matrix operations, like in neural networks.

**Code and Detailed Explanation**

# Step 1: Import the NumPy library

import numpy as np

We import NumPy and give it a nickname np to use it easily.

# Step 2: Create a basic list

my\_list = [1, 2, 3, 4]

This is a normal Python list storing 4 numbers.

# Step 3: Convert list into a NumPy array

my\_array = np.array(my\_list)

NumPy array is like a faster, more powerful list made for math operations.

# Step 4: Perform math on the array

print("Original array:", my\_array)

print("Add 10 to each number:", my\_array + 10)

print("Multiply each number by 2:", my\_array \* 2)

These operations are vectorized (no loop needed).

# Step 5: Create a 2D array (like a matrix)

my\_2d\_array = np.array([[1, 2], [3, 4]])

print("2D Array:\n", my\_2d\_array)

print("Sum of all values:", np.sum(my\_2d\_array))

print("Transpose of 2D array:\n", my\_2d\_array.T)

2D arrays are like tables and are used for AI inputs. .T transposes rows into columns.

### 2. Pandas – Data Handling Library

* Pandas is used for data analysis.
* Helps you work with tables, like rows and columns in Excel.
* In AI, we often clean, filter, and organize data before sending it to the model.

**Code and Detailed Explanation**

# Step 1: Import the Pandas library

import pandas as pd

We use pd as a short name to access Pandas.

# Step 2: Create sample data

data = {

'Name': ['Ali', 'Sara', 'John'],

'Marks': [85, 90, 78]

}

A Python dictionary holding student names and their marks.

# Step 3: Convert data into a DataFrame

df = pd.DataFrame(data)

print("Student Data:\n", df)

DataFrame is a Pandas table. It looks like a spreadsheet.

# Step 4: View only the names column

print("Names:\n", df['Name'])

This selects just one column from the table.

# Step 5: Filter students with marks > 80

print("Marks > 80:\n", df[df['Marks'] > 80])

Returns only the rows where the condition is true.

#### 3. Matplotlib – Drawing Graphs

Matplotlib is used to visualize data through charts and graphs.

* Helps us “see” trends and patterns, which is important before making an AI model.

**Code and Detailed Explanation**

# Step 1: Import the plotting library

import matplotlib.pyplot as plt

We use plt as a short name to access plotting functions.

# Step 2: Sample data

names = ['Ali', 'Sara', 'John']

marks = [85, 90, 78]

These are same values as we used in Pandas.

# Step 3: Bar chart

plt.bar(names, marks)

plt.title("Student Marks")

plt.xlabel("Students")

plt.ylabel("Marks")

plt.show()

Shows student marks in vertical bars.

# Step 4: Line chart

plt.plot(names, marks, marker='o')

plt.title("Student Marks Line Chart")

plt.xlabel("Students")

plt.ylabel("Marks")

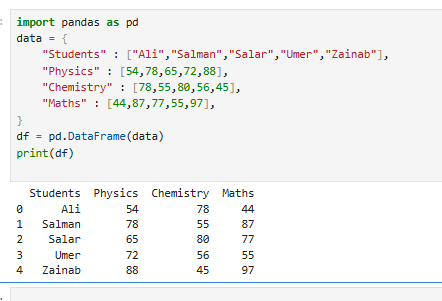
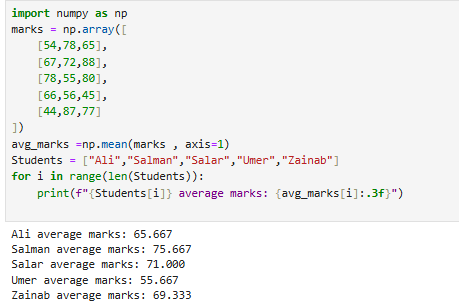
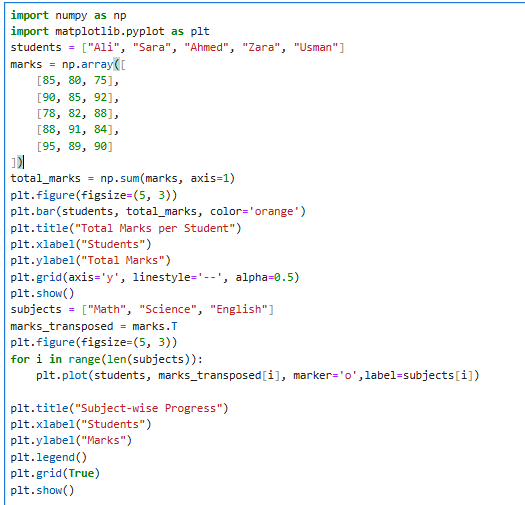
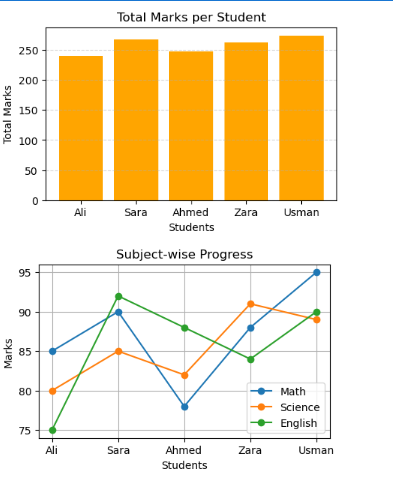
plt.show()

Draws a line joining the points (like progress tracking).

##### Lab Task

**Title: Student Marks Analyzer using NumPy, Pandas, and Matplotlib**

**Instructions:**

1. Use **Pandas** to create a table with 5 students and their marks in 3 subjects.  
   
2. Use **NumPy** to:
   * Convert marks into arrays.
   * Calculate the average mark of each student.  
     
3. Use **Matplotlib** to:
   * Draw a **bar chart** of total marks per student.
   * Draw a **line chart** showing the progress of each subject.  
       
     

**Lab10 – Integrated Lab on Scikit-learn, Preprocessing, and AI Pipelines**

**Objective:**

* + To introduce basic machine learning using **Scikit-learn**, along with **data preprocessing** and **building AI pipelines** in Python.

## **Introduction to Scikit-learn (Machine Learning Basics)**

* + **Machine Learning (ML)** is the core part of Artificial Intelligence (AI). It allows computers to learn from data and make decisions.
  + **Supervised Learning** is a type of ML where we teach the model using labeled data (input + output).
  + **Scikit-learn** is a Python library used to build ML models easily.

**Common Terms:**

**Dataset**: Collection of data (features + target)

**Model**: Mathematical formula to make predictions

**fit()**: Train the model

**predict()**: Make prediction

**accuracy\_score**: Measure how correct the model is

### List of Libraries which is used,

**Scikit-learn (sklearn) – Core AI & ML Library**

|  |  |  |
| --- | --- | --- |
| **Module** | **Function** | **Purpose** |
| sklearn.datasets | load\_iris(), load\_digits(), load\_breast\_cancer() | Loads free, ready-to-use datasets for practice |
| sklearn.model\_selection | train\_test\_split() | Splits your data into training and testing sets |
| sklearn.tree | DecisionTreeClassifier() | A model that uses decision trees for classification |
| sklearn.neighbors | KNeighborsClassifier() | A model that classifies using the nearest neighbors |
| sklearn.linear\_model | LogisticRegression() | A simple model for binary/multi-class classification |
| sklearn.preprocessing | LabelEncoder, StandardScaler | Prepares the data: encodes text, scales numbers |
| sklearn.pipeline | Pipeline() | Combines multiple steps (scaling + modeling) into one clean unit |
| sklearn.metrics | accuracy\_score, classification\_report | Checks how well your model is performing |

Code Example:

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

**Load dataset**

data = load\_iris()

**Split data**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data.data, data.target, test\_size=0.2)

**Create and train model**

model = DecisionTreeClassifier()

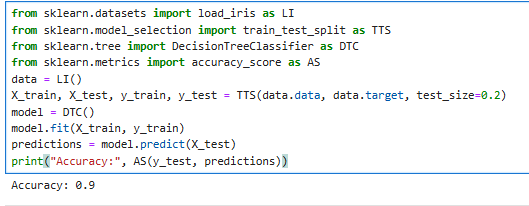
model.fit(X\_train, y\_train)

**Predict and evaluate**

predictions = model.predict(X\_test)

print("Accuracy:", accuracy\_score(y\_test, predictions))

#### **Lab Task:**

Replace DecisionTreeClassifier with KNeighborsClassifier and compare results.  


Data Preprocessing and Feature Engineering

Before training a model, we must clean and prepare data. This step is called Data Preprocessing.

**Common Techniques:**

Missing Values: Fill empty data using mean, median, or mode.

Label Encoding: Convert text data into numbers.

Feature Scaling: Normalize values for better model performance.

Code Example:

import pandas as pd

from sklearn.preprocessing import LabelEncoder, StandardScaler

**Sample dataset**

df = pd.DataFrame({

'Age': [25, 30, None, 35],

'Gender': ['Male', 'Female', 'Female', 'Male']

})

**Handle missing value**

df['Age'].fillna(df['Age'].mean(), inplace=True)

**Encode Gender**

df['Gender'] = LabelEncoder().fit\_transform(df['Gender'])

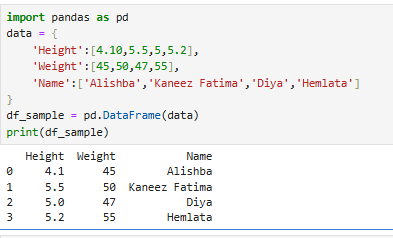
**Scale Age**

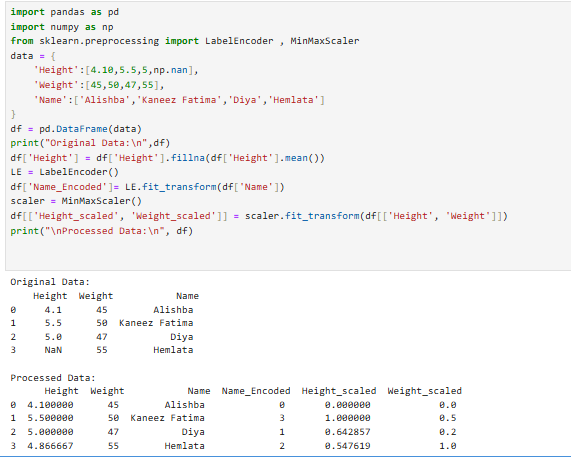
scaler = StandardScaler()

df['Age'] = scaler.fit\_transform(df[['Age']])

print(df)

##### **Lab Task:**

Create a similar dataset with 2 numeric columns and 1 text column.  


Apply missing value handling, label encoding, and scaling.  


Building an AI Pipeline

A Pipeline is a step-by-step process where we combine data preprocessing and model training into one unit.

Why Pipelines?

Cleaner code

Avoids mistakes

Good for real-world applications

Key Terms:

Pipeline: Combines multiple steps

StandardScaler: Normalizes data

LogisticRegression: A classification algorithm

classification\_report: Shows model performance

Code Example:

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.datasets import load\_breast\_cancer

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report

# Load data

data = load\_breast\_cancer()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data.data, data.target, test\_size=0.2)

# Create pipeline

pipeline = Pipeline([

('scaler', StandardScaler()),

('classifier', LogisticRegression())

])

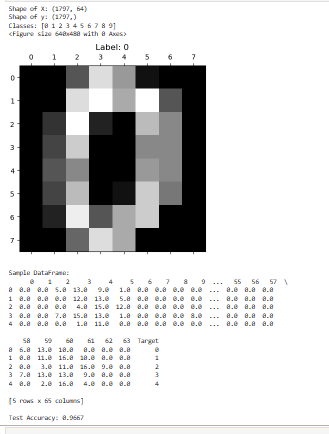
# Train and evaluate

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

y\_pred = pipeline.predict(X\_test)

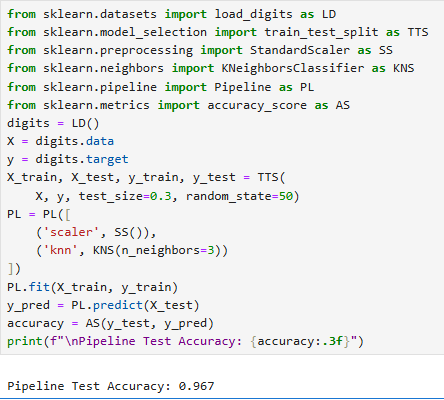
print(classification\_report(y\_test, y\_pred))

###### **Lab Task:**

Use load\_digits() dataset.  
  


Create a pipeline with StandardScaler and KNeighborsClassifier.

|  |  |
| --- | --- |
| Topic | What You Learned |
| Scikit-learn | Build and test ML models |
| Preprocessing | Clean and transform raw data |
| Pipeline | Combine steps for full AI workflow |

  
**Lab 11. Model Evaluation and Cross-Validation in Machine Learning**

## Lab Objectives

* Understand why model evaluation is important
* Explain overfitting and underfitting
* Use **cross-validation** to evaluate ML models more accurately
* Compare different models to choose the best one

**Beginner-Friendly Theory**

**1. What is Model Evaluation?**

When we build an AI model, we need to check:

“How good is this model?”

Model evaluation helps us test our model’s **accuracy**, **performance**, and **generalization**.

We don’t want a model that:

* Only works well on training data (memorizing)
* Fails to work on new, unseen data

**2. What is Train-Test Split?**

We divide the dataset into:

* **Training Set**: For training the model (usually 70–80%)
* **Testing Set**: For testing the model (remaining 20–30%)

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

But this has a **problem**:

One test split may not show the real performance of a model!

**3. Overfitting and Underfitting**

|  |  |  |
| --- | --- | --- |
| **Concept** | **Description** | **Example** |
| **Overfitting** | Model memorizes training data, fails on test data | 100% accuracy on train, 60% on test |
| **Underfitting** | Model is too simple, performs poorly on both | 60% train and 50% test accuracy |

**Visualization:**

We’ll use graphs in class to show how a model can overfit or underfit a curve.

**4. What is Cross-Validation?**

Instead of splitting data once, we split it **multiple times**. This is called **K-Fold Cross-Validation**.

**K-Fold Cross Validation:**

* Split the dataset into **K parts** (e.g. K=5)
* Train & test the model **K times**
* Each time, use a different fold for testing
* **Final result = average of all K scores**

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

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

print("CV Score:", scores)

print("Average Score:", scores.mean())

**5. Comparing Models**

We can apply cross-validation to multiple models:

from sklearn.datasets import load\_iris

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

from sklearn.tree import DecisionTreeClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

# Load dataset

data = load\_iris()

X, y = data.data, data.target

# Create models

models = {

'Decision Tree': DecisionTreeClassifier(),

'Logistic Regression': LogisticRegression(max\_iter=200),

'KNN': KNeighborsClassifier()

}

# Evaluate each model

for name, model in models.items():

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

print(f"{name} → CV Accuracy: {scores.mean():.2f}")

**Optional: Plot Results Using Matplotlib**

import matplotlib.pyplot as plt

names = list(models.keys())

accuracies = [cross\_val\_score(m, X, y, cv=5).mean() for m in models.values()]

plt.bar(names, accuracies, color='skyblue')

plt.title("Model Accuracy Comparison")

plt.ylabel("Cross-Validation Accuracy")

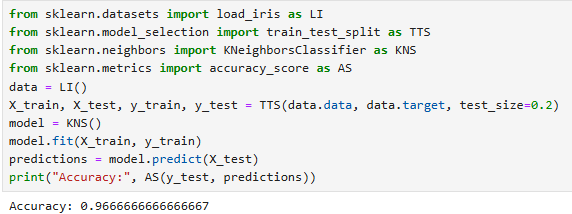
plt.ylim(0.8, 1.0)

plt.show()

### **Lab Task**

### **Task 1:**

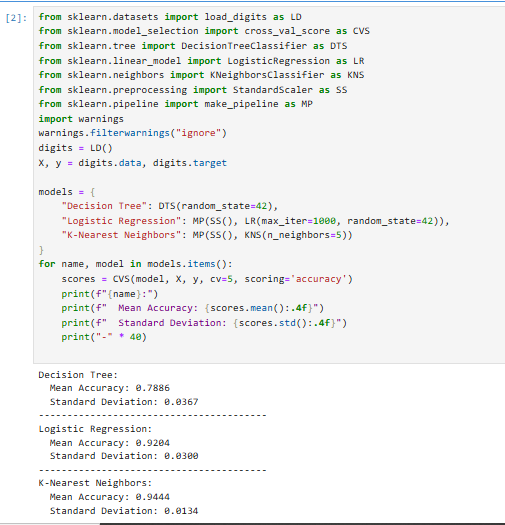
Use cross-validation to evaluate the performance of KNeighborsClassifier on the Iris dataset.



#### **Task 2:**

Use load\_digits() dataset and compare the performance of:

* Decision Tree
* Logistic Regression
* K-Nearest Neighbors (KNN)



##### **Task 3:**

Create a bar chart of accuracy results using matplotlib.  
