

Hello and Welcome to AI Camp



Python Programming

Tobni Mohamed Islam



Python in AI: A Powerful Partnership

- Python's syntax is clear and concise
- The vast array of open-source libraries and frameworks available in Python
- Python has emerged as the preferred programming language for Artificial Intelligence (AI) and Machine Learning (ML)
- Its active community support and continuous development make Python an ideal choice for staying at the forefront of AI advancements.

Table of Contents

1. Python Fundamentals

- 1.1 Introduction to google collab (basic usage)
- 1.3 Variables and data types (integers, floats, strings, lists ...)
- 1.4 Control structures (if statements, loops)
- 1.5 Functions and modules

2. Introduction to Numpy

- 2.1 Basics of numpy arrays (creation, indexing and slicing)
- 2.2 Performing basic mathematical operations with Numpy arrays

3. Introduction to Pandas

- 3.1 Pandas Series and DataFrames
- 3.2 Basic operations (selecting, filtering, modifying data)

4. Introduction to data visualization

- 4.1 Creating simple plots with matplotlib
- 4.2 Adding labels, titles and legends to plots

Python Fundamentals

Google colab - Basic usage

- Google Colab is a cloud-based platform that allows you to write and execute Python code in a collaborative environment, directly from your browser.
- It provides free access to GPUs, making it a powerful tool for machine learning and data analysis tasks.
- To get started, go to colab.research.google.com, sign in with your Google account, and create a new Colab notebook.

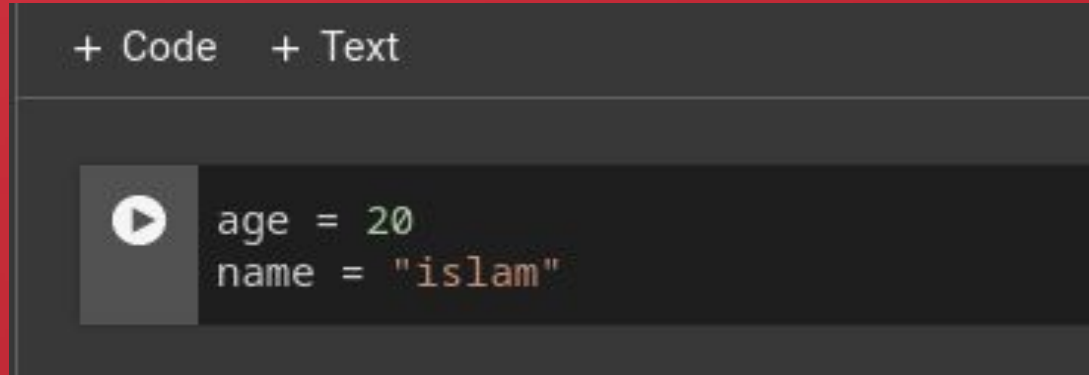
Google colab - Basic usage

- The Colab interface consists of cells, each of which can contain either code or text.
- The toolbar at the top allows you to execute cells, add new cells, and manage the runtime.
- To run a cell, you can click the play button in the toolbar, use the keyboard shortcut (Shift + Enter), or select "Runtime" > "Run cell" from the menu.
- The order of execution matters; cells are executed sequentially.

Variables and Data Types

Python is a dynamically-typed language, The interpreter determines the type dynamically.

- **Naming conventions: Start with a letter or underscore, followed by letters, numbers, or underscores.**



```
+ Code + Text  
  
▶ age = 20  
  name = "islam"
```

Data Types Overview

- Python has several built-in data types:
 - Integer (`int`)
 - Float (`float`)
 - String (`str`)
 - Boolean (`bool`)
- Each data type is used to represent different kinds of information.

Numeric Data Types

- **Integer (int):**
 - Represents whole numbers.

```
+ Code + Text
```

```
age = 20
```

- **Float (float):**
 - Represents floating-point numbers (decimals)

```
+ Code + Text
```

```
height = 1.75
```

Textual Data Types

- **String (str):**
 - Represents text.
 - Created using single or double quotes.



A screenshot of a code editor interface. At the top, there are two tabs labeled '+ Code' and '+ Text'. Below the tabs, a line of code is displayed: `name = "islam"`. To the left of the code, there is a green checkmark icon and the text '1s', indicating a successful execution or a short duration.

Collections in python

Python offers various collection types to store multiple values:

- Lists (`list`)
- Tuples (`tuple`)
- Dictionaries (`dict`)
- Sets (`set`)

Lists and Tuples

- **List (list):**
 - Ordered mutable collection of values

```
+ Code + Text
```

```
✓ 0s [play] fruits = ["apple", "orange", "banana"]
```

- **Tuple (tuple):**
 - Ordered immutable collection of values

```
+ Code + Text
```

```
✓ 0s [play] coordinates = (4,5)
```

Dictionaries and Sets


- **Dictionary (dict):**
 - Unordered collection of key-value pairs

```
+ Code + Text
```

```
✓ 0s  person = {"name": "islam", "age": 20}
```

- **Set (set):**
 - Unordered unique collection of values

```
+ Code + Text
```

```
✓ 0s  unique_numbers = {1,2,3,4,5}
```


Type Conversion

- Sometimes, you may need to convert between data types
- Use functions like `int()`, `float()`, `str()`, and `bool()` for type conversion.

```
[30] age = "25"
     age_int = int(age)

print(type(age))
print(type(age_int))

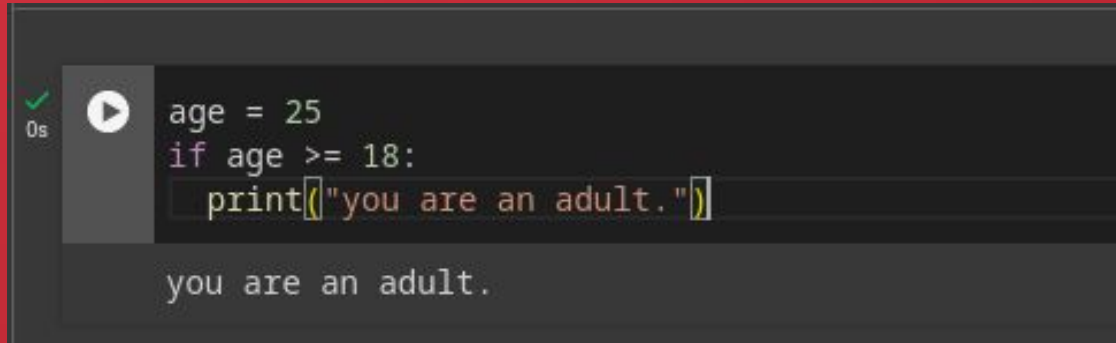
<class 'str'>
<class 'int'>
```

Control Structures

- Control structures allow you to manage the flow of your program based on certain conditions and loops.
- They include:
 - If-else statements
 - for loops
 - while loops

If-Else Statements

- Allows you to execute a block of code if a condition is true.



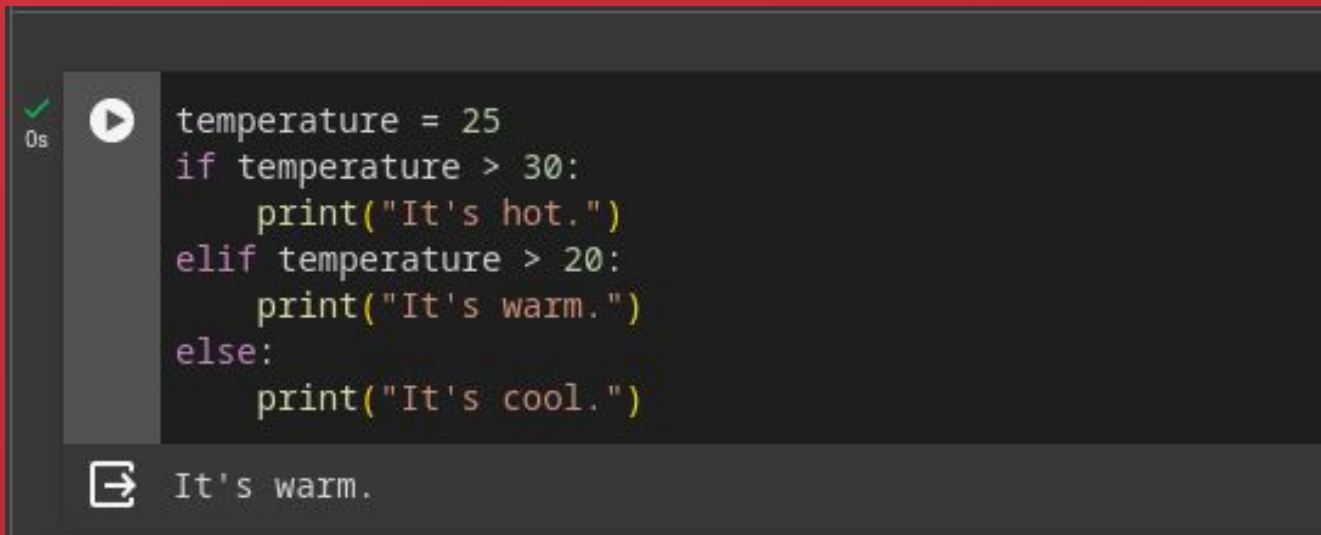
The image shows a code execution environment with a dark background. On the left, there is a green checkmark icon and a play button icon. Below the checkmark is the text '0s'. To the right of the play button is a code editor containing the following Python code:

```
age = 25
if age >= 18:
    print("you are an adult.")
```

Below the code editor, the output of the code is displayed: "you are an adult."

Elif Statements

- Stands for "else if" and allows you to check multiple conditions.




A screenshot of a code editor or terminal window with a dark background. On the left, there is a vertical sidebar with a green checkmark and '0s' at the top, and a play button icon below it. The main area contains Python code using an elif statement. Below the code, there is a light gray bar showing the output of the code execution.

```
temperature = 25
if temperature > 30:
    print("It's hot.")
elif temperature > 20:
    print("It's warm.")
else:
    print("It's cool.")
```

It's warm.

For loop

- Used for iterating over a sequence (e.g., list, tuple, string).

```
✓ 0s  fruits = ["apple", "orange", "banana"]  
for fruit in fruits:  
    print(fruit)
```

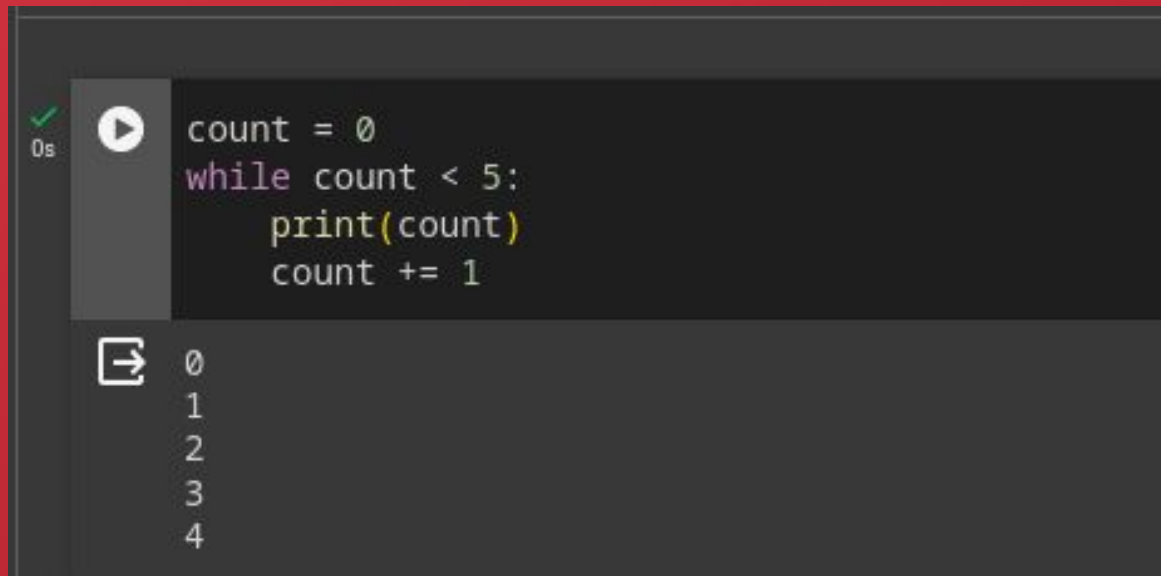
apple
orange
banana

```
✓ 0s  fruits = ["apple", "orange", "banana"]  
for i in range(len(fruits)):  
    print(fruits[i])
```

apple
orange
banana

While loop

- Repeats a block of code as long as a condition is true.



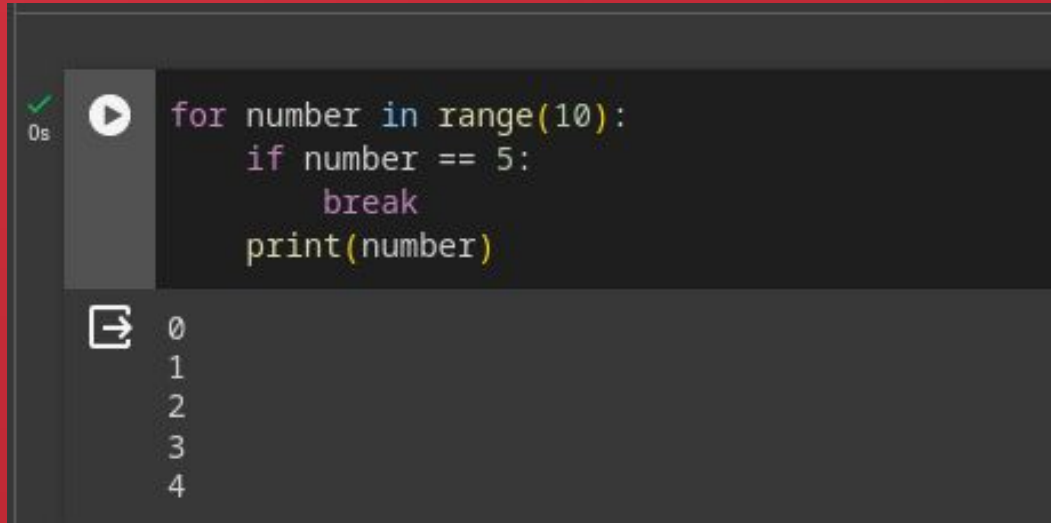
The screenshot shows a code editor with a dark background. On the left, there is a green checkmark and '0s' indicating successful execution. Below that is a play button icon. The code is written in Python and consists of four lines: 'count = 0', 'while count < 5:', ' print(count)', and ' count += 1'. Below the code, there is a list of numbers: 0, 1, 2, 3, 4, which are the output of the loop.

```
count = 0
while count < 5:
    print(count)
    count += 1
```

0
1
2
3
4

Break Statement

- Exits the loop prematurely.



The image shows a code editor with a dark background. On the left, there is a vertical toolbar with a green checkmark and '0s' at the top, a play button in the middle, and a console icon at the bottom. The code is written in a light-colored font. The code is a for loop that iterates over the range(10). Inside the loop, there is an if statement that checks if the current number is equal to 5. If it is, the loop is broken. Otherwise, the number is printed. The output of the code is shown in the console, displaying the numbers 0, 1, 2, 3, and 4, which are the values of the loop before it reaches the break statement at 5.

```
for number in range(10):  
    if number == 5:  
        break  
    print(number)
```

0
1
2
3
4

Functions in Python

- Functions in Python are blocks of reusable code that perform a specific task.
- They help in modularizing code, making it easier to understand, maintain, and reuse.

Writing a Function in Python

- **Function Definition:**
- Start with the `def` keyword, followed by the function name and parameters.

A code editor snippet with a dark background. On the left, there is a green checkmark icon and a play button icon. To the right of these icons, the text "0s" is visible. The main part of the snippet contains the following Python code:

```
def greet(name):  
    print("Hello, " + name + "!")
```

Writing a Function in Python

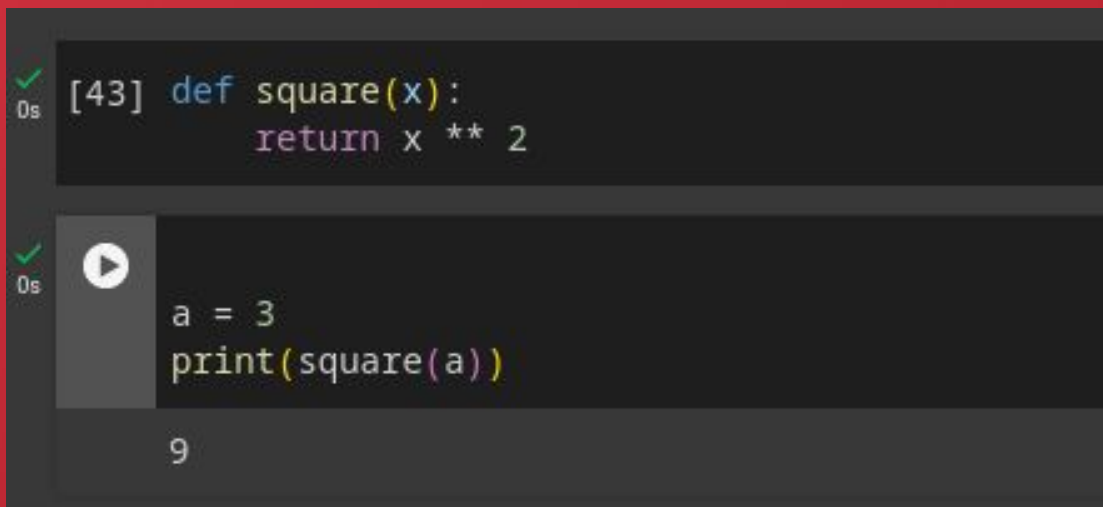
- **Function Call:**
- Use the function name followed by parentheses to execute the function.



A screenshot of a code execution environment. On the left, there is a green checkmark and a play button icon. Below the play button, the text "0s" indicates the execution time. The main area shows the code `greet("islam")` in a dark background. Below the code, the output `Hello, islam!` is displayed in a lighter background.

Writing a Function in Python

- **Return Statement:**
- Use the return statement to send a value back from the function.



The screenshot shows a Jupyter Notebook interface with two cells. The first cell contains a function definition: `def square(x):` followed by an indented `return x ** 2`. The second cell contains a play button icon, followed by `a = 3` and `print(square(a))`. Below the code in the second cell, the output `9` is displayed. Both cells have a green checkmark and '0s' in the left margin, indicating successful execution.

```
[43] def square(x):  
      return x ** 2
```

```
a = 3  
print(square(a))
```

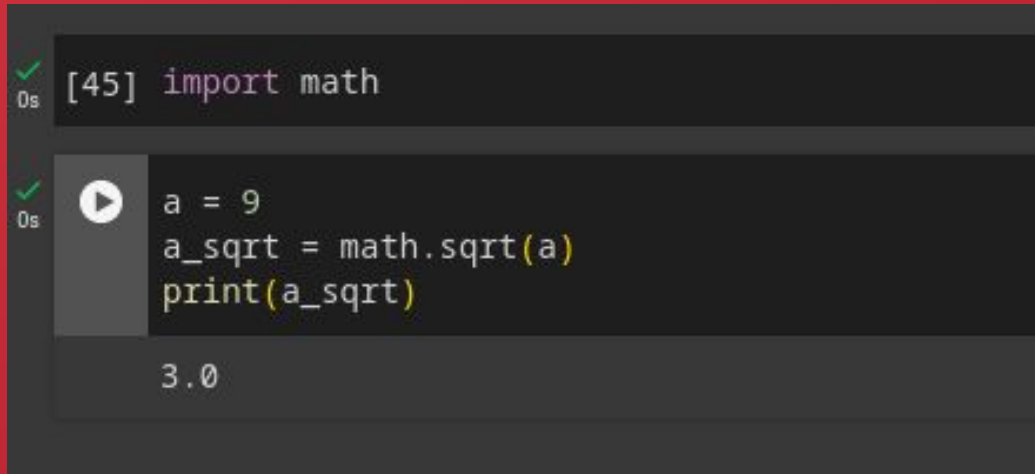
9

Modules in Python

- Modules are files containing Python code that can be reused in other Python files.
- They allow you to organize code into separate files for better structure

Modules in Python

- To use a module, you need to import it using the `import` keyword.



A screenshot of a Jupyter Notebook interface with a dark theme. The first cell contains the code `[45] import math` and has a green checkmark icon and '0s' execution time. The second cell contains the code `a = 9`, `a_sqrt = math.sqrt(a)`, and `print(a_sqrt)`, preceded by a play button icon, and also has a green checkmark icon and '0s' execution time. Below the code, the output `3.0` is displayed.

```
[45] import math
```

```
a = 9
a_sqrt = math.sqrt(a)
print(a_sqrt)
```

3.0

Q&A Session

Introduction to Numpy

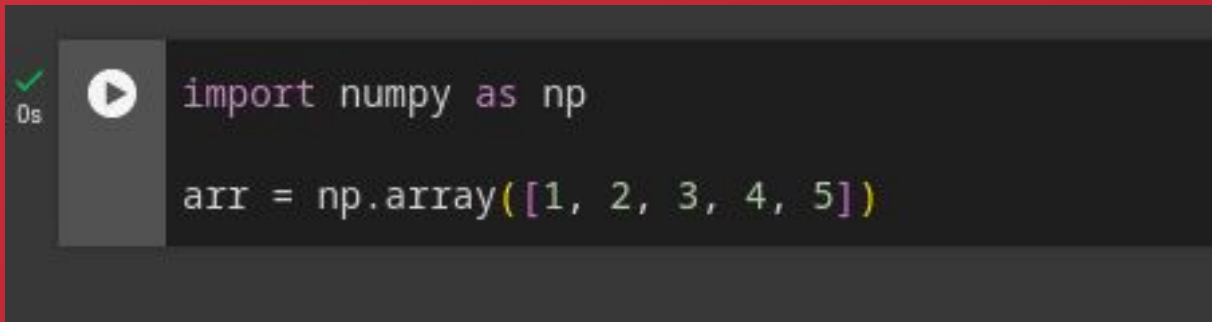
- NumPy is a powerful library for numerical computing in Python.
- It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.
- NumPy is a fundamental tool for data scientists, researchers, and engineers working with numerical data.

Basics of Numpy Arrays

Basics of Numpy arrays

Array Creation:

- Create arrays using the `numpy.array()` function.

A code execution interface with a dark background. On the left, there is a green checkmark icon and the text '0s'. Next to it is a play button icon. To the right of the play button, the following Python code is displayed:

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

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

Basics of Numpy arrays

Array Indexing:

- Access elements of an array using indexing (0-based).

A screenshot of a Jupyter Notebook cell. On the left, there is a green checkmark and a play button icon. The code in the cell is `print(arr[0])` followed by a comment `# Output: 1`. Below the code, the output `1` is displayed.

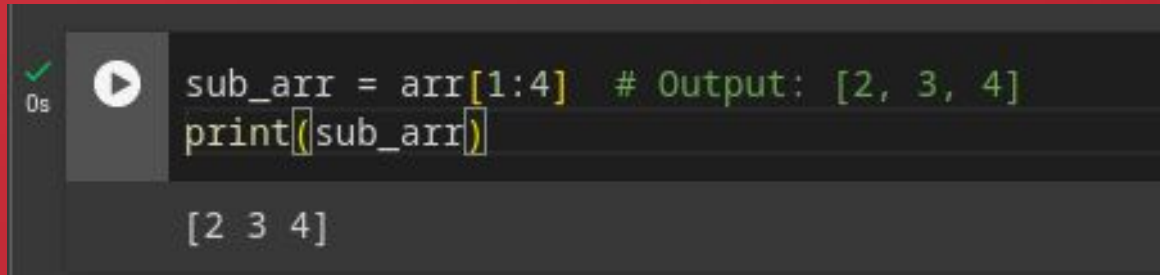
```
✓ 0s [play] print(arr[0]) # Output: 1
```


1

Basics of Numpy arrays

Array Slicing:

- Extract subarrays using slicing.

A screenshot of a Jupyter Notebook code cell. On the left, there is a green checkmark icon and a play button icon. The code cell contains two lines of Python code: `sub_arr = arr[1:4]` followed by a comment `# Output: [2, 3, 4]`, and `print(sub_arr)`. Below the code, the output is displayed as `[2 3 4]`.


```
✓ 0s  sub_arr = arr[1:4] # Output: [2, 3, 4]  
print(sub_arr)  
  
[2 3 4]
```

Multi-Dimensional Arrays

Multi-Dimensional Arrays

- NumPy supports multi-dimensional arrays.

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

✓ 0s  `print(matrix[1, 2])` # Output: 6

6

Array Operations

Array Operations

Element-wise Operations:

- NumPy allows for element-wise operations on arrays.

```
✓ [57] result = matrix * 2  
0s print(result)  
  
[[ 2  4  6]  
 [ 8 10 12]]
```

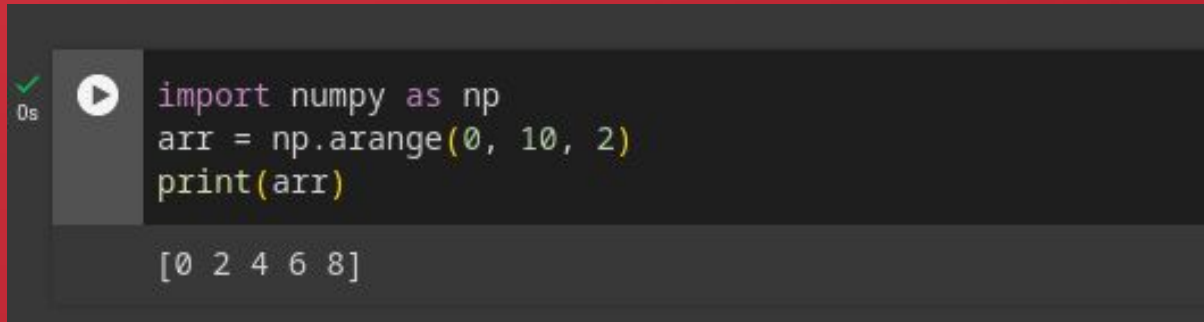
```
✓ [58] result = matrix + 5  
0s print(result)  
  
[[ 6  7  8]  
 [ 9 10 11]]
```


Numpy Functions

Numpy Functions

`np.arange()` - Create an Array with a Range:

- Generates an array with regularly spaced values.
- Syntax: `np.arange(start, stop, step)`

A screenshot of a Jupyter Notebook code cell. On the left, there is a green checkmark and a play button icon. The code cell contains the following Python code:

```
import numpy as np
arr = np.arange(0, 10, 2)
print(arr)
```

 Below the code, the output is displayed as `[0 2 4 6 8]`.


```
import numpy as np
arr = np.arange(0, 10, 2)
print(arr)
```

[0 2 4 6 8]

Numpy Functions

`np.zeros()` and `np.ones()` - Create Arrays of Zeros and Ones:

- Generates arrays filled with zeros or ones.
- Syntax: `np.zeros(shape)` and `np.ones(shape)`

```
✓ 0s  zeros_arr = np.zeros((3, 3))
ones_arr = np.ones((2, 4))


print(zeros_arr)
print(ones_arr)
```

```
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]]
```

Numpy Functions

`np.random.rand()` - Random Values in a Given Shape:

- Generates an array of random values in the $[0.0, 1.0)$ interval with a given shape.
- Syntax: `np.random.rand(d0, d1, ..., dn)`


```
✓ 1s  random_values = np.random.rand(2, 3)
print(random_values)


[[0.98256224 0.08184861 0.29765565]
 [0.29965464 0.39622773 0.85310264]]
```

Numpy Functions

`np.reshape()` - Reshape an Array:

- Changes the shape of an array without changing its data.
- Syntax: `np.reshape(array, new_shape)`


```
0s  original_array = np.arange(6)
    reshaped_array = np.reshape(original_array, (2, 3))
    print(reshaped_array)
```


```
 [[0 1 2]
 [3 4 5]]
```

Numpy Functions

`np.sum()`:

- Computes the sum of array elements
- Syntax: `np.sum(array, axis)`

```
0s  arr = np.array([[1,2,3],[4,5,6]])  
total_sum = np.sum(arr)  
print(total_sum)
```


 21

Introduction to Pandas

- Pandas is a powerful library for data manipulation and analysis in Python.
- It provides two primary data structures: Series and DataFrame.
- Pandas is widely used in data science, machine learning, and data analysis tasks.

Pandas Series

- **Series Definition:**
 - A one-dimensional labeled array capable of holding any data type.

```
0s  import pandas as pd

data = [1, 3, 5, 7, 9]
series = pd.Series(data)
print(series)
```

0	1
1	3
2	5
3	7
4	9

dtype: int64

Pandas DataFrame

- **DataFrame Definition:**

- A two-dimensional labeled data structure with columns that can be of different types

```
✓ 0s ▶ import pandas as pd

data = {'Name': ['Alice', 'Bob', 'Charlie'],
        'Age': [25, 30, 22],
        'City': ['New York', 'San Francisco', 'Los Angeles']}

df = pd.DataFrame(data)

print(df)
```

	Name	Age	City
0	Alice	25	New York
1	Bob	30	San Francisco
2	Charlie	22	Los Angeles

Selecting Data in Pandas

Selecting Columns:

- Use square brackets or dot notation.

```
✓ 0s ▶ ages = df['Age']  
      print(ages)
```


0	25
1	30
2	22

Name: Age, dtype: int64

Filtering Data in Pandas

Filtering Rows Based on Conditions:

- Use boolean indexing.

```
0s  young_people = df[df['Age'] < 30]  
print(young_people)
```

	Name	Age	City
0	Alice	25	New York
2	Charlie	22	Los Angeles

Modifying Data in Pandas

Adding a New Column:

- Simply assign values to a new column.

```
[73] df['Salary'] = [50000, 60000, 45000]
```

✓
0s



```
df['Salary'] = 50000
```

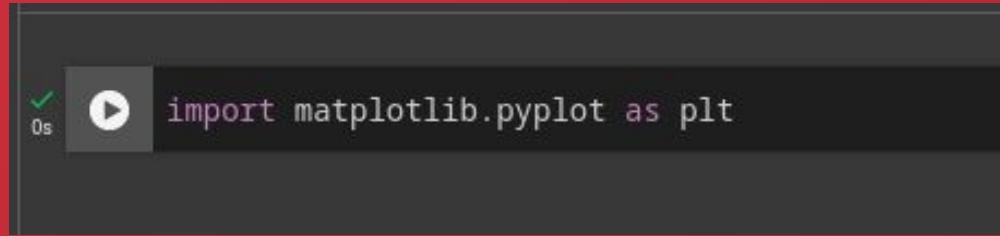
Introduction to Data Visualization with Matplotlib

- Matplotlib is a widely-used library for creating static, animated, and interactive visualizations in Python.
- It provides a variety of plotting options to help you explore and communicate your data effectively.

Basics of matplotlib

Importing Matplotlib:

- Start by importing the library.

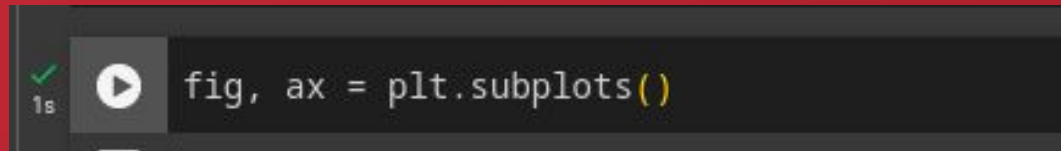


```
import matplotlib.pyplot as plt
```

A Jupyter Notebook cell with a green checkmark and a play button icon. The code `import matplotlib.pyplot as plt` is entered in the text area.

Creating a Figure and Axes:

- The basic structure for a Matplotlib plot involves creating a figure and one or more axes.




```
fig, ax = plt.subplots()
```

A Jupyter Notebook cell with a green checkmark and a play button icon. The code `fig, ax = plt.subplots()` is entered in the text area.

Line Plots with matplotlib


Plotting a Line:

- Use the `plot()` function to create a line plot.

```
✓ 0s  x = [1, 2, 3, 4, 5]  
y = [2, 4, 6, 8, 10]  
  
ax.plot(x, y, label='Line Plot')
```

Customizing Line Style and Color:

- Customize the appearance of the line.

```
✓ 0s  ax.plot(x, y, linestyle='--', color='green', marker='o', label='Customized Line')
```

Scatter Plots with matplotlib

Plotting a scatter points:

- Use the `scatter()` function to create a scatter plot.

```
▶ x = [1, 2, 3, 4, 5]  
  y = [2, 4, 6, 8, 10]  
  
  ax.plot(x, y, label='scatter Plot')
```

Customizing marker Style and Color:

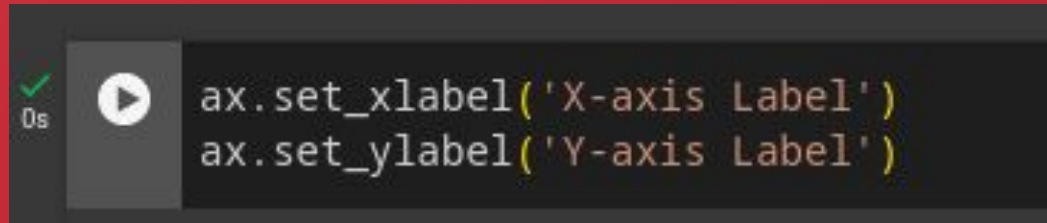
- Customize the appearance of the markers.

```
▶ ax.scatter(x, y, marker='s', color='red', label='Customized Scatter')
```

Adding Labels and Titles

Adding Labels:

- Use `set_xlabel()` and `set_ylabel()` to add labels to the x and y-axes.

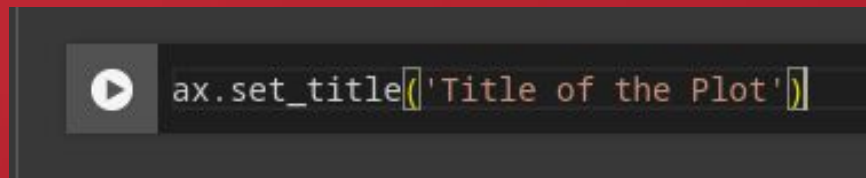
A code editor snippet with a dark background. On the left, there is a green checkmark icon and a play button icon. The code text is as follows:

```
ax.set_xlabel('X-axis Label')  
ax.set_ylabel('Y-axis Label')
```

0s

Adding a Title:

- Use `set_title()` to add a title to the plot.

A code editor snippet with a dark background. On the left, there is a play button icon. The code text is as follows:

```
ax.set_title('Title of the Plot')
```

Continue Your Learning Journey

- Explore the documentations to deepen your understanding of the field.
- Data Science and Machine Learning:
 - **Kaggle**: Participate in competitions and access datasets for hands-on experience.
- Interactive Coding Practice:
 - **HackerRank**
 - **LeetCode**

Challenge

- a simple programming problem
- mathematical operation to do with numpy
- a tabular dataset to play with

Thank you for attending

any questions ?

