**National University of Computer and Emerging Sciences**



Lab Manual 01

Introduction to Data Science Lab

BDS 3C

Course Instructor: Mr. Saif Ul Islam

Lab Instructor: Mr. Raja Ijaz , Ms Amina Zulfiqar

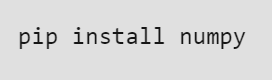
# Numpy

# Introduction to Numpy

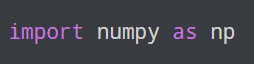
NumPy stands for Numerical Python, is an open-source Python library that provides support for large, multi-dimensional arrays and matrices. It also has a collection of high-level mathematical functions to operate on arrays.

* **Installing and importing the library**

We can install NumPy in Python using the *pip install* statement.

****

We can import NumPy in Python using the *import* statement.



The code above imports the *numpy* library in our program as an alias *np*.

After this import statement, we can use NumPy functions and objects by calling them with *np*.

* **Arrays in NumPy**

NumPy’s main object is the homogeneous multidimensional array.

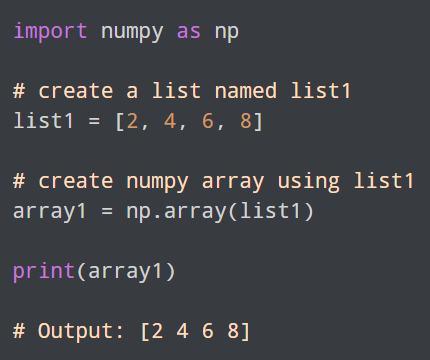
* It is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers.
* In NumPy, dimensions are called *axes*. The number of axes is *rank*.
* NumPy’s array class is called ndarray. It is also known by the alias array.
* **NumPy Array Creation**

An array allows us to store a collection of multiple values in a single data structure.

The NumPy array is similar to a list, but with added benefits such as being faster and more memory efficient.

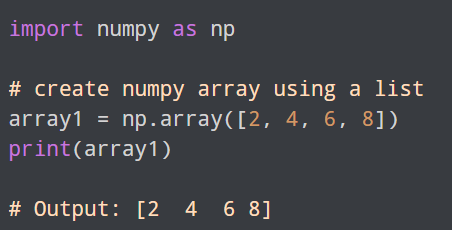
Numpy library provides various methods to work with data. To leverage all those features, we first need to create numpy arrays. There are multiple techniques to generate arrays in NumPy, and we will explore each of them below.

**We can create a NumPy array using a Python List**

****

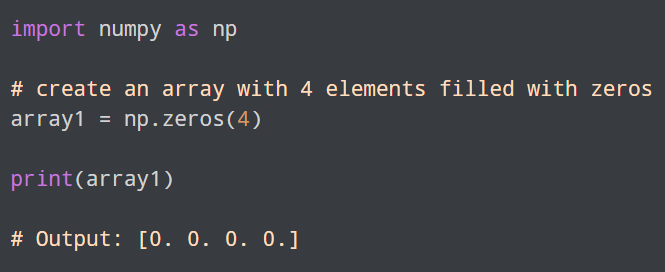
In the above example, we first imported the numpy library as np and created a list named list1. we have created an array by passing list1 as an argument to the np.array() function.

Instead of creating a list and using the list variable with the np.array() function, we can directly pass list elements as an argument.



* **Create an Array Using np.zeros()**

The np.zeros() function allows us to create an array filled with all zeros.

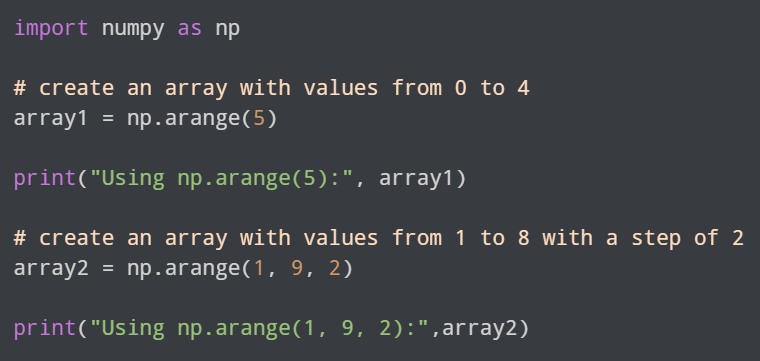


we have created an array named array1 with **4** elements all initialized to **0** using the np.zeros(4) function.

Similarly we can use np.ones() to create an array filled with values **1.**

* **Create an Array With np.arange()**

The np.arange() function returns an array with values within a specified interval.



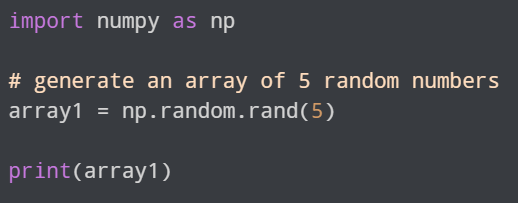
A number on a black background

Description automatically generated

we have created arrays using the np.arange() function.

* np.arange(5) - create an array with 5 elements, where the values range from **0** to **4**
* np.arange(1, 9, 2) - create an array with 5 elements, where the values range from **1** to **8** with a step of **2**.
* **Create an Array With np.random.rand()**

The np.random.rand() function is used to create an array of random numbers. Let's see an example to create an array of 5 random numbers.

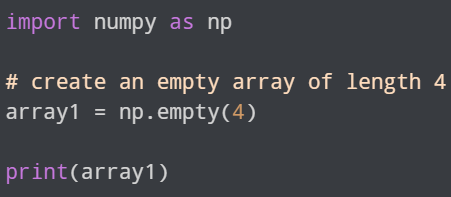




In the above example, we have used the np.random.rand() function to create an array array1 with **5** random numbers. This code generates a different output each time we run it.

* **Create an Empty NumPy Array**

To create an empty NumPy array, we use the np.empty() function.





we have created an empty array of length **4** using the np.empty() function. If we look into the output of the code, we can see the empty array is actually not empty, it has some values in it. It is because although we are creating an empty array, NumPy will try to add some value to it. The values stored in the array are arbitrary and have no significance.

* **NumPy N-D Array Creation**

NumPy is not restricted to 1-D arrays, it can have arrays of multiple dimensions, also known as N-dimensional arrays or ndarrays.

An N-dimensional array refers to the number of dimensions in which the array is organized.

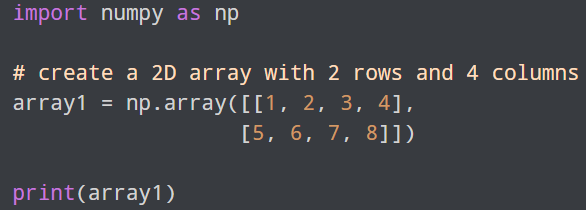
An array can have any number of dimensions and each dimension can have any number of elements.

For example, a 2D array represents a table with rows and columns, while a 3D array represents a cube with width, height, and depth.

* **N-D Array Creation From List of Lists**

To create an N-dimensional NumPy array from a Python List, we can use the np.array() function and pass the list as an argument.

Let's create a 2D NumPy array with **2** rows and **4** columns using lists.



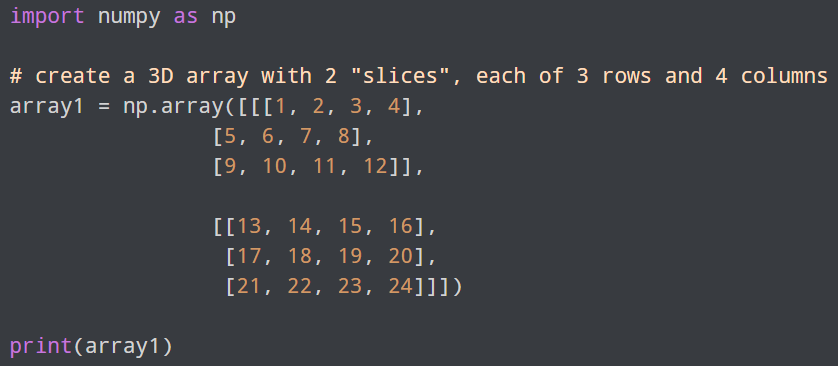
A number on a black background

Description automatically generated

we first created a 2D list (list of lists) [[1, 2, 3, 4], [5, 6, 7, 8]] with **2** rows and **4** columns. We then passed the list to the np.array() function to create a 2D array.

* **Create a 3-D NumPy Array**

Let's say we want to create a 3-D NumPy array consisting of two **"slices"** where each slice has **3** rows and **4** columns.



A black background with white numbers

Description automatically generated

we created a 3D list [list of lists of lists] and passed it to the np.array() function. This creates the 3-D array named array1.

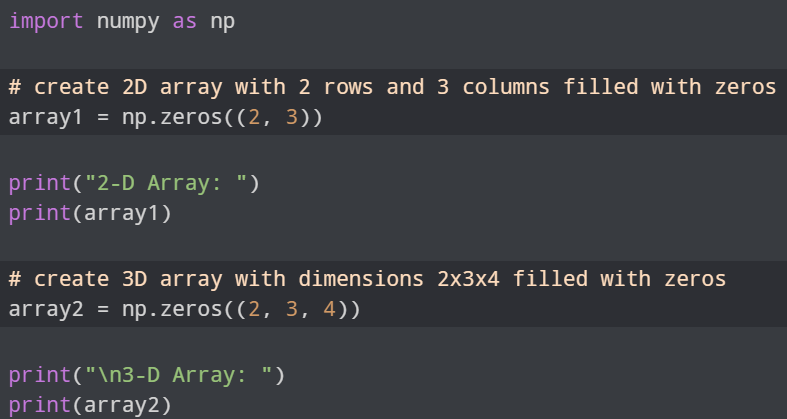
In the 3D list,

* The outermost list contains two elements, which are lists representing the two "slices" of the array. Each slice is a 2-D array with **3** rows and **4** columns.
* The innermost lists represent the individual rows of the 2-D arrays.

We saw how to create N-d NumPy arrays from Python lists. Now we'll see how we can create them from scratch.

To create multidimensional arrays from scratch we use functions such as

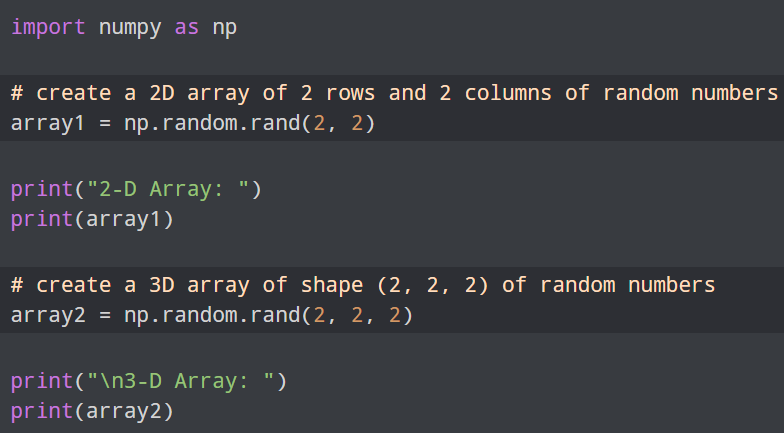
* **np.zeros()**
* **np.arange()**
* **np.random.rand()**

****

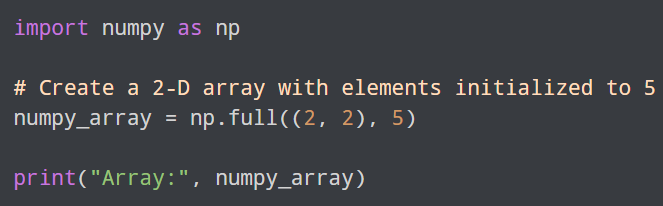
A screenshot of a computer code

Description automatically generated

**The np.random.rand() function is used to create an array of random numbers.**



**we can use the np.full() function to create a multidimensional array with a specified value.**



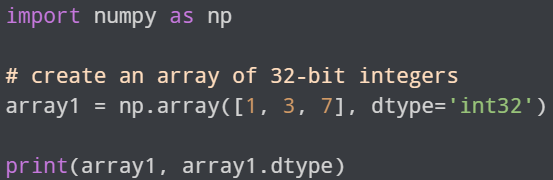
**NumPy Data Types**

NumPy offers a wider range of numerical data types than what is available in Python. Here's the list of most commonly used numeric data types in NumPy:

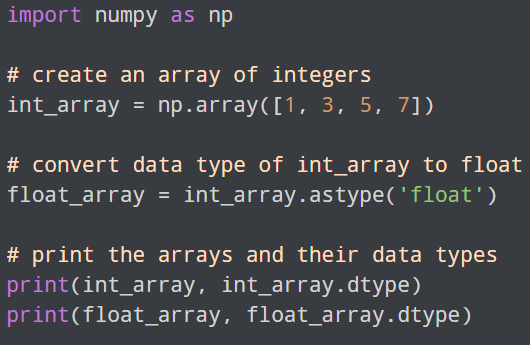
* int8, int16, int32, int64 - signed integer types with different bit sizes
* uint8, uint16, uint32, uint64 - unsigned integer types with different bit sizes
* float32, float64 - floating-point types with different precision levels
* complex64, complex128 - complex number types with different precision levels

**we can use the *dtype* attribute**

we can create an array with a defined data type by passing the dtype parameter while calling the np.array() function



**we can convert the data type of an array using the astype() method**



* **NumPy Array Attributes**

Attributes are properties of NumPy arrays that provide information about the array's shape, size, data type, dimension.

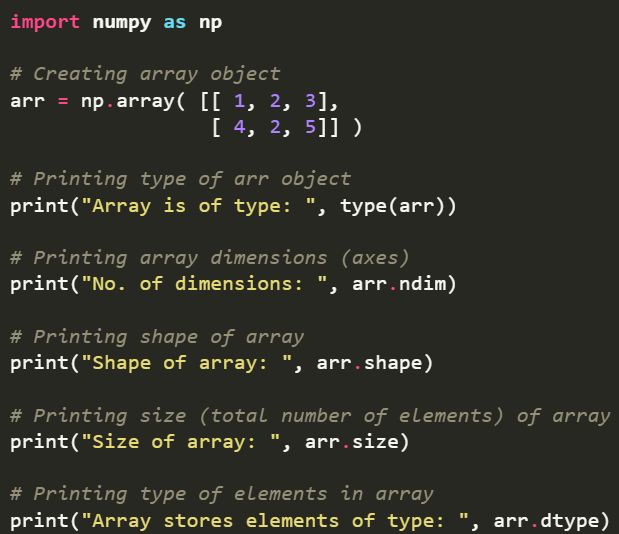
Here are some of the commonly used NumPy attributes:

|  |  |
| --- | --- |
| Attributes | Description |
| ndim | returns number of dimension of the array |
| size | returns number of elements in the array |
| dtype | returns data type of elements in the array |
| shape | returns the size of the array in each dimension. |
| itemsize | returns the size (in bytes) of each elements in the array |
| data | returns the buffer containing actual elements of the array in memory |

**For Example:**

we are creating a two-dimensional array that has the rank of 2 as it has 2 axes.

The first axis(dimension) is of length 2, i.e., the number of rows, and the second axis(dimension) is of length 3, i.e., the number of columns. The overall shape of the array can be represented as (2, 3)

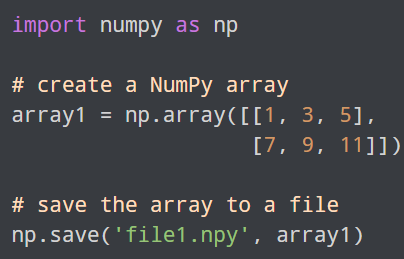


* **NumPy Input Output**

NumPy offers input/output (I/O) functions for loading and saving data to and from files. Input/output functions support a variety of file formats, including binary and text formats.

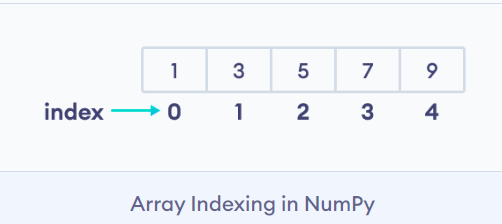
Here are some of the commonly used NumPy Input/Output functions:

|  |  |
| --- | --- |
| Function | Description |
| save() | saves an array to a binary file in the NumPy .npy format. |
| load() | loads data from a binary file in the NumPy .npy format |
| savetxt() | saves an array to a text file in a specific format |
| loadtxt() | loads data from a text file. |



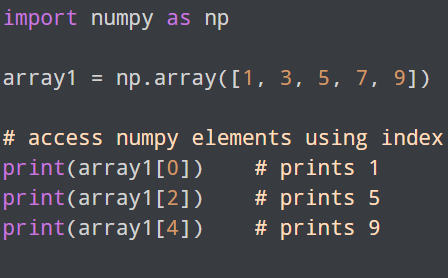
* **Numpy Array Indexing**

Each element in an array is associated with a number. The number is known as an **array index**.

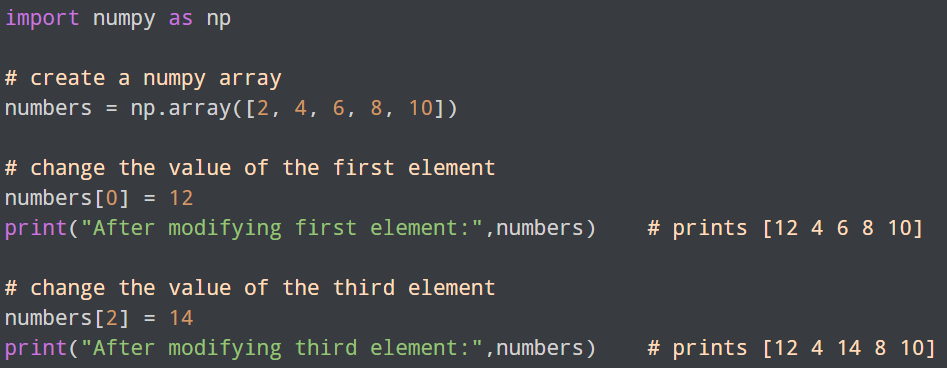


In the above array, 5 is the 3rd element. However, its index is 2.

This is because the array indexing starts from 0, that is, the first element of the array has index 0, the second element has index 1, and so on. We can use indices to access individual elements of a NumPy array.



* **Modify Array Elements Using Index**

****

* **NumPy Negative Array Indexing**

NumPy allows negative indexing for its array. The index of -1 refers to the last item, -2 to the second last item and so on.

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

* **NumPy Array Slicing**

Array Slicing is the process of extracting a portion of an array. With slicing, we can easily access elements in the array. It can be done on one or more dimensions of a NumPy array.

**Syntax of NumPy Array Slicing:**

****

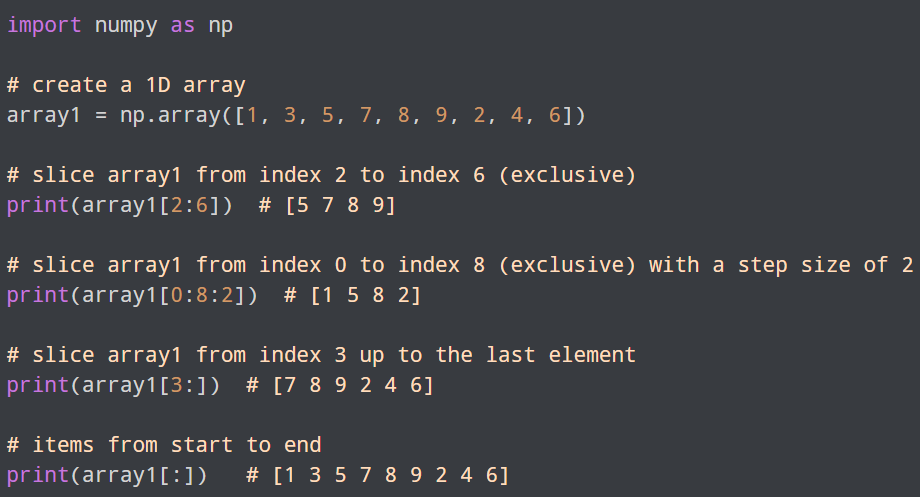
Here,

* start - index of the first element to be included in the slice
* stop - index of the last element (exclusive)
* step - step size between each element in the slice

When we slice arrays, the start index is inclusive but the stop index is exclusive.

* If we omit start, slicing starts from the first element
* If we omit stop, slicing continues up to the last element
* If we omit step, default step size is 1

it's possible to access the portion of an array using the slicing operator **:** .

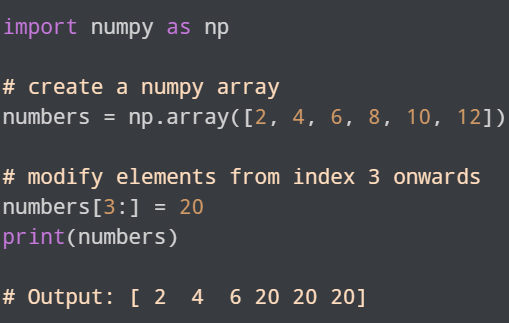


we have created the array named array1 with **9** elements.

Then, we used the slicing operator **:** to slice array elements.

* array1[2:6] - slices array1 from index **2** to index **6**, not including index **6**
* array1[0:8:2] - slices array1 from index **0** to index **8**, not including index **8**
* array1[3:] - slices array1 from index **3** up to the last element
* array1[:] - returns all items from beginning to end

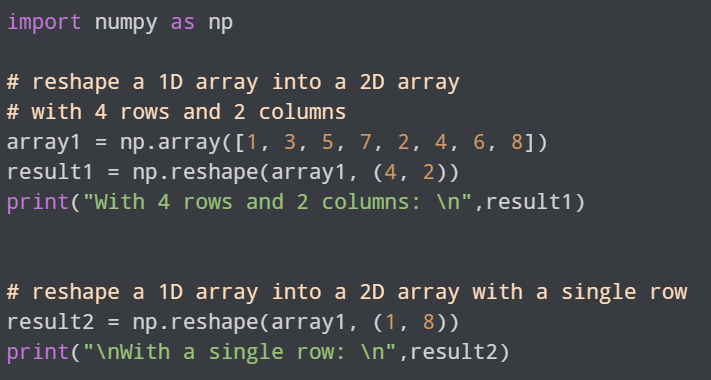
**Modify Array Elements Using Slicing**

****

numbers[3:] = 20 replaces all the elements from index **3** onwards with new value **20**.

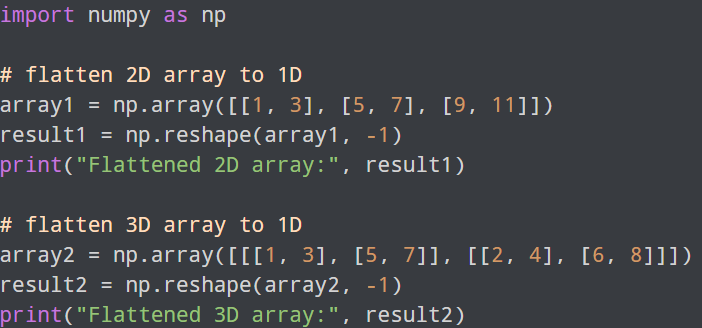
**NumPy Array Reshaping**

NumPy array reshaping simply means changing the shape of an array without changing its data.



**Flatten N-d Array to 1-D Array Using reshape()**

Flattening an array simply means converting a multidimensional array into a 1D array. To flatten an N-d array to a 1-D array we can use reshape() and pass "-1" as an argument.



**Common NumPy Array Functions**

|  |  |
| --- | --- |
| Array Operations | Functions |
| Array Creation Functions | np.array(), np.zeros(), np.ones(), np.empty(), etc. |
| Array Manipulation Functions | np.reshape(), np.transpose(), etc. |
| Array Mathematical Functions | np.add(), np.subtract(), np.sqrt(), np.power(), etc. |
| Array Statistical Functions | np.median(), np.mean(), np.std(), and np.var(). |
| Array Input and Output Functions | np.save(), np.load(), np.loadtxt(), etc. |

Here's a comprehensive list of commonly used NumPy functions, categorized by their use cases:

**1. Array Creation**

* np.array(): Create an array from a list, tuple, or other data.
* np.zeros(): Create an array of all zeros.
* np.ones(): Create an array of all ones.
* np.empty(): Create an uninitialized array.
* np.full(): Create an array of a specified shape and fill it with a specified value.
* np.arange(): Create an array with evenly spaced values.
* np.linspace(): Create an array with evenly spaced values over a specified interval.
* np.random.rand(): Create an array of random values between 0 and 1.
* np.random.randn(): Create an array of random values from the standard normal distribution.
* np.random.randint(): Create an array of random integers within a given range.

**2. Array Properties**

* array.shape: Get the shape of an array.
* array.size: Get the number of elements in an array.
* array.ndim: Get the number of dimensions of an array.
* array.dtype: Get the data type of the array.

**3. Array Manipulation**

* np.reshape(): Reshape an array.
* np.ravel(): Flatten an array.
* np.flatten(): Return a flattened version of the array.
* np.transpose(): Transpose the array.
* np.tile(): Construct an array by repeating the original array.
* np.repeat(): Repeat elements of an array.
* np.split(): Split an array into multiple sub-arrays.
* np.concatenate(): Concatenate two or more arrays.
* np.vstack(), np.hstack(): Stack arrays vertically or horizontally.
* np.hsplit(), np.vsplit(): Split an array into multiple sub-arrays horizontally or vertically.

**4. Indexing and Slicing**

* array[]: Basic indexing and slicing of arrays.
* np.take(): Take elements from an array along an axis.
* np.put(): Replace specified elements of an array with values.

**5. Mathematical Operations**

* np.add(), np.subtract(), np.multiply(), np.divide(): Basic arithmetic operations.
* np.dot(): Dot product of two arrays.
* np.matmul(): Matrix product of two arrays.
* np.sqrt(): Square root.
* np.sum(): Sum of elements along a given axis.
* np.mean(): Mean value of an array along a specified axis.
* np.median(): Median value of an array.
* np.var(): Variance of array elements.
* np.std(): Standard deviation of array elements.
* np.min(), np.max(): Minimum and maximum values of an array.
* np.argmin(), np.argmax(): Indices of the minimum and maximum values.
* np.cumsum(): Cumulative sum of array elements.
* np.prod(): Product of array elements.
* np.power(): Raise elements of an array to a specified power.

**6. Sorting and Searching**

* np.sort(): Sort elements of an array.
* np.argsort(): Returns the indices that would sort an array.
* np.searchsorted(): Find indices where elements should be inserted to maintain order.
* np.unique(): Find the unique elements in an array.

**7. Saving and Loading Arrays**

* np.save(): Save an array to a binary file in NumPy format.
* np.load(): Load an array from a binary file in NumPy format.
* np.savetxt(): Save an array to a text file.
* np.loadtxt(): Load data from a text file.

# Pandas

Pandas is a Python library used for data manipulation and analysis. Pandas provide a convenient way to analyze and clean data.

**What is Pandas Used for?**

Pandas is a powerful library generally used for:

* Data Cleaning
* Data Transformation
* Data Analysis
* Machine Learning
* Data Visualization

**Install Pandas**

****

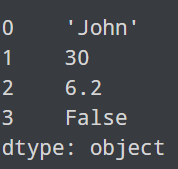
**Import Pandas in Python**

**A close up of a logo

Description automatically generated**

**Pandas Series**

A Pandas Series is a one-dimensional labeled array-like object that can hold data of any type. A Pandas Series can be thought of as a column in a spreadsheet or a single column of a DataFrame. It consists of two main components: the labels and the data.



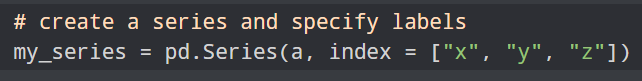
Here, the series has two columns, labels (**0**, **1**, **2** and **3**) and data ('John', 30, 6.2, False). The labels are the index values assigned to each data point, while the data represents the actual values stored in the Series.

**There are multiple ways to create a Pandas Series, but the most common way is by using a Python list.**

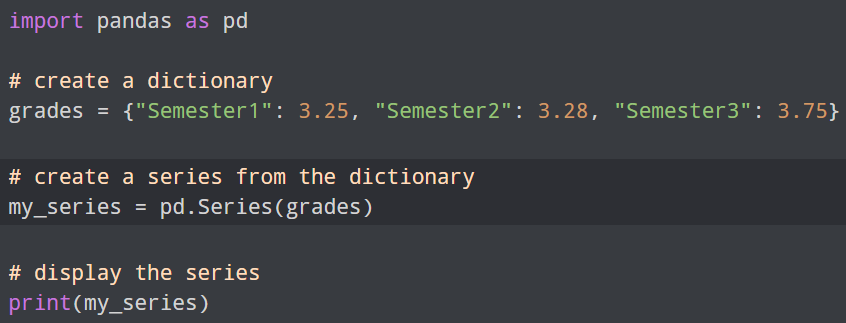
**A screenshot of a computer program

Description automatically generated**

We can also specify labels while creating the series using the index argument in the Series() method.



**You can also create a Pandas Series from a Python dictionary**.



**Pandas DataFrame**

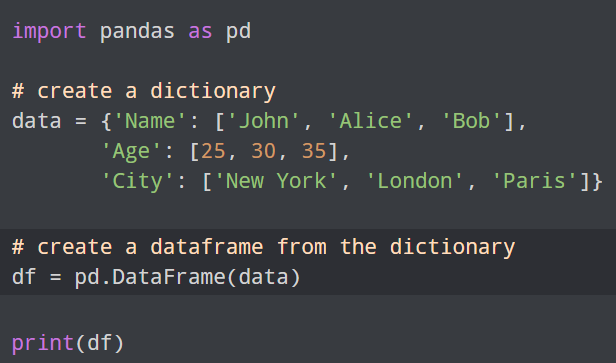
A DataFrame is like a table where the data is organized in rows and columns. It is a two-dimensional data structure like a two-dimensional array

**Create a Pandas DataFrame**

We can create a Pandas DataFrame in the following ways:

* Using Python Dictionary
* Using Python List
* From a File
* Creating an Empty DataFrame

**We can create a dataframe using a dictionary by passing it to the DataFrame() function.**

****

**We can also create a DataFrame using a two-dimensional list.**

**A screen shot of a computer program

Description automatically generated**

**Another common way to create a DataFrame is by loading data from a CSV (Comma-Separated Values) file.**

**A screen shot of a computer code

Description automatically generated**

we used the read\_csv() function which reads the CSV file data.csv, and automatically creates a DataFrame object df, containing data from the CSV file. We can also create a DataFrame using other file types like JSON, Excel spreadsheet, SQL database, etc. The methods to read different file types are listed below:

* JSON - read\_json()
* Excel spreadsheet - read\_excel()
* SQL - read\_sql()

**Pandas Index**

In Pandas, an index refers to the labeled array that identifies rows or columns in a DataFrame or a Series.

Pandas offers several ways to create indexes. Some common methods are as follows:

* Default Index
* Setting Index
* Creating a Range Index

**View Data in a Pandas DataFrame**

A Pandas Dataframe can be displayed as any other Python variable using the **print()** function.

However, when dealing with very large DataFrames with large numbers of rows and columns, the print() function is unable to display the whole DataFrame. Instead, it prints only a part of the DataFrame.

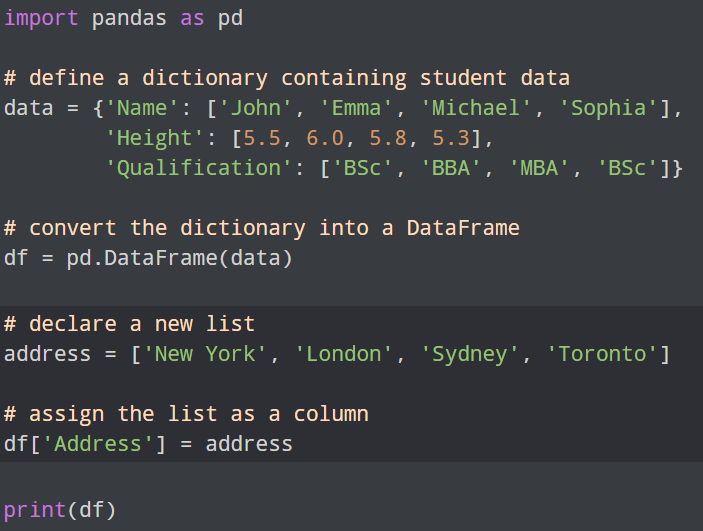
In the case of large DataFrames, we can use **head(), tail() and info()** methods to get the overview of the DataFrame.

**Pandas DataFrame Manipulation**

DataFrame manipulation in Pandas involves editing and modifying existing DataFrames. Some common DataFrame manipulation operations are:

* Adding rows/columns
* Removing rows/columns
* Renaming rows/columns

**We can add a new column to an existing Pandas DataFrame by simply declaring a new list as a column.**

****

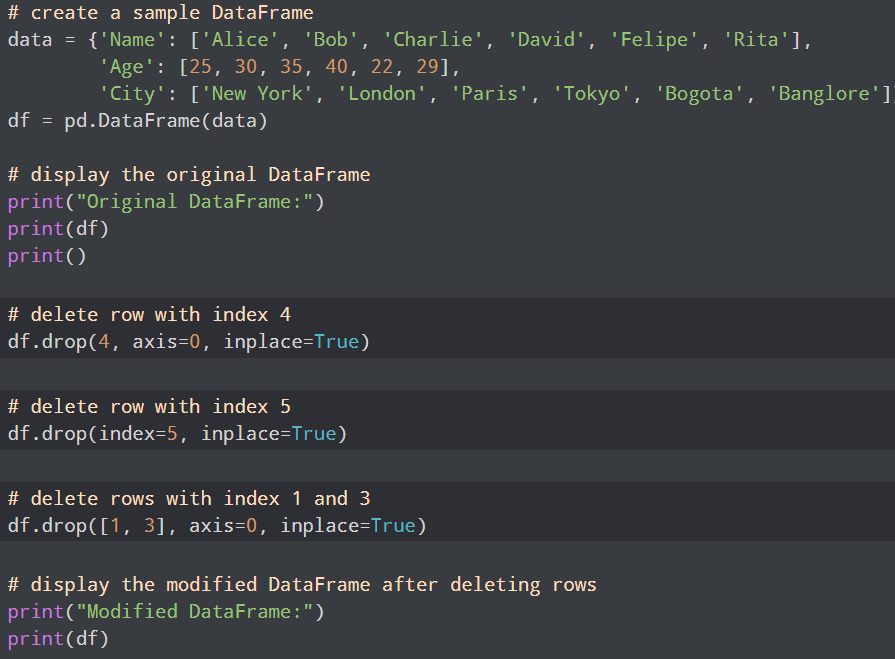
Adding rows to a DataFrame is not quite as straightforward as adding columns in Pandas. We use the .loc property to add a new row to a Pandas DataFrame.

A screenshot of a computer program

Description automatically generated

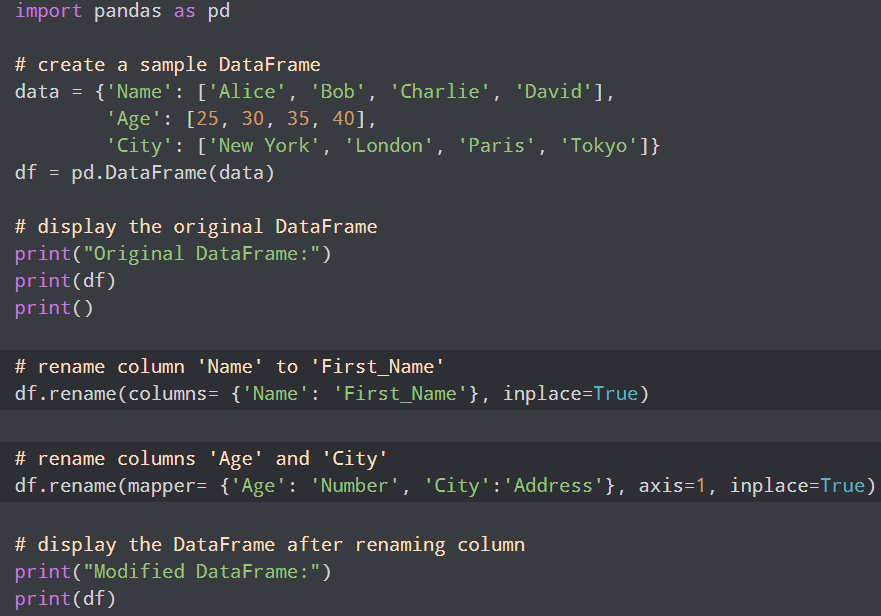
**Remove Rows/Columns from a Pandas DataFrame**

We can use **drop()** to delete rows and columns from a DataFrame.

****

**Rename Labels in a DataFrame**

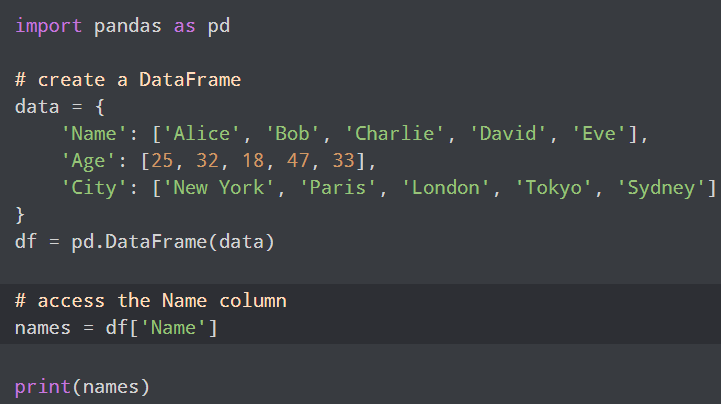
We can rename columns in a Pandas DataFrame using the rename() function.



**Pandas Indexing and Slicing**

Indexing refers to accessing rows and columns of data from a DataFrame, whereas slicing refers to accessing a range of rows and columns. We can access data or range of data from a DataFrame using different methods.

**We can access columns of a DataFrame using the bracket ([]) operator.**



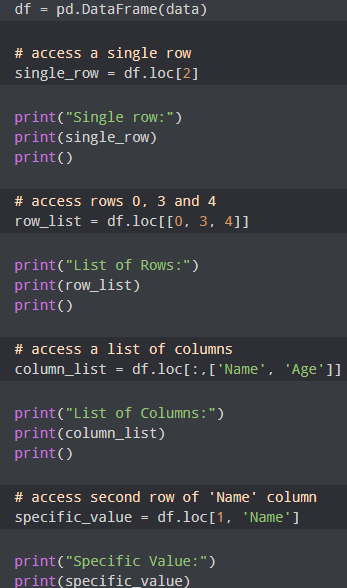
**Pandas .loc**

we use the .loc property to access and modify data within a DataFrame using label-based indexing. It allows us to select specific rows and columns based on their labels.

A black and white text

Description automatically generated

* row\_indexer - selects rows by their labels, can be a single label, a list of labels, or a boolean array
* column\_indexer - selects columns, can also be a single label, a list of labels, or a boolean array



we used .loc to access a row, a list of rows, a list of columns and a specific value using the respective labels.

In the line,

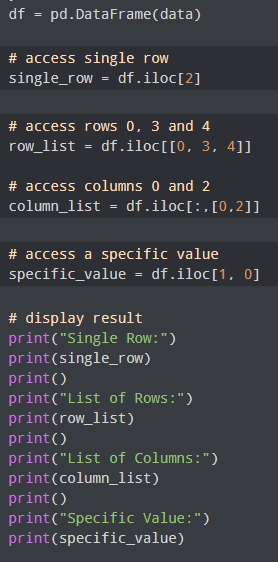
column\_list = df.loc[:,['Name', 'Age']]

The : operator indicates that all the rows are to be selected.

**Pandas .iloc**

In Pandas, the .iloc property is used to access and modify data within a DataFrame using **integer-based** indexing. It allows us to select specific rows and columns based on their integer locations.



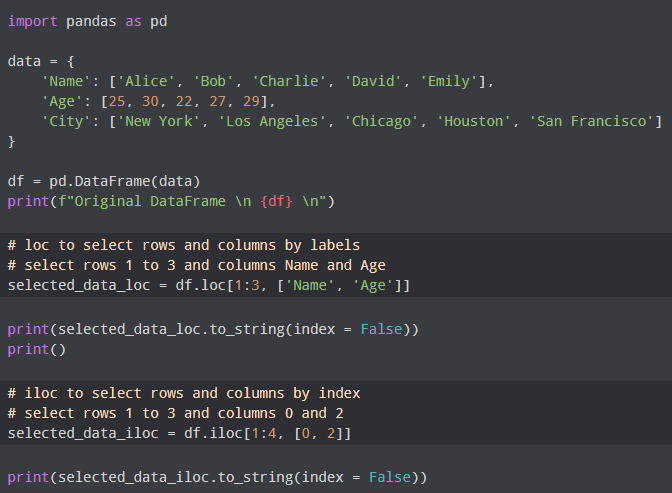


we used .iloc to access a row, a list of rows, a list of columns and a specific value using the respective integer values.

**.loc vs .iloc**

The main differences between .loc and .iloc are as follows:

|  |  |  |
| --- | --- | --- |
| Basis | .loc | .iloc |
| Indexing | Label-based indexing | Integer-based indexing |
| Endpoint | Endpoint is included | Endpoint is not included |
| Boolean indexing | Boolean indexing is supported | Boolean indexing is not supported |



Here's a comprehensive list of commonly used pandas functions, categorized by their use cases:

**1. DataFrame Creation**

* pd.DataFrame(): Create a DataFrame from dictionaries, lists, NumPy arrays, etc.
* pd.read\_csv(): Read a CSV file into a DataFrame.
* pd.read\_excel(): Read an Excel file into a DataFrame.
* pd.read\_json(): Read a JSON file into a DataFrame.
* pd.read\_sql(): Read from a SQL database into a DataFrame.

**2. Basic DataFrame Operations**

* df.head(): View the first few rows of a DataFrame.
* df.tail(): View the last few rows of a DataFrame.
* df.shape: Get the dimensions of the DataFrame (rows, columns).
* df.columns: Get column labels of the DataFrame.
* df.index: Get row labels of the DataFrame.
* df.dtypes: Get data types of each column.
* df.info(): Get a summary of the DataFrame.
* df.describe(): Generate descriptive statistics.

**3. Indexing and Selection**

* df['column']: Select a single column from the DataFrame.
* df[['col1', 'col2']]: Select multiple columns.
* df.loc[]: Select rows and columns by labels.
* df.iloc[]: Select rows and columns by integer-location based indexing.
* df.at[]: Access a single value for a row/column label pair.
* df.iat[]: Access a single value for a row/column position pair.
* df.sample(): Return a random sample of items from the DataFrame.

**4. Modifying DataFrames**

* df['new\_column'] = value: Add a new column.
* df.drop(): Drop rows or columns.
* df.rename(): Rename columns or rows.
* df.set\_index(): Set the DataFrame index using existing columns.
* df.reset\_index(): Reset the index of the DataFrame.

**5. Data Cleaning**

* df.isnull(): Detect missing values.
* df.notnull(): Detect existing values.
* df.dropna(): Drop rows or columns with missing data.
* df.fillna(): Fill missing values with a specified value.
* df.interpolate(): Interpolate missing data.

**6. Sorting and Ranking**

* df.sort\_values(): Sort by values.
* df.sort\_index(): Sort by the index.
* df.rank(): Compute numerical data ranks.

**7. Data Aggregation**

* df.groupby(): Group data by a column or multiple columns.
* df.agg(): Aggregate using one or more operations over specified axis.
* df.cumsum(): Cumulative sum.
* df.cumprod(): Cumulative product.
* df.mean(), df.median(), df.sum(), df.min(), df.max(): Summary statistics.

**8. Merging and Joining DataFrames**

* pd.concat(): Concatenate DataFrames along a particular axis.
* pd.merge(): Merge DataFrames using a key (similar to SQL joins).
* df.join(): Join DataFrames on a key column(s).

**9. Pivoting and Reshaping Data**

* df.pivot(): Reshape data based on column values.
* df.pivot\_table(): Create a pivot table as a DataFrame.
* df.melt(): "Unpivot" a DataFrame from wide to long format.

**10. Exporting DataFrames**

* df.to\_csv(): Write a DataFrame to a CSV file.
* df.to\_excel(): Write a DataFrame to an Excel file.
* df.to\_json(): Write a DataFrame to a JSON file.
* df.to\_sql(): Write a DataFrame to a SQL database.

**NumPy Array Questions**

1. Create a 1D array of 10 integers ranging from 1 to 10. Slice the array to extract the first five elements and display them.
2. Generate a 3x3 matrix with random integers between 1 and 50. Find and display the sum of all elements in the matrix.
3. Create an array of 20 evenly spaced values between 0 and 5 using np.linspace(). Display the array.
4. Create a 4x4 identity matrix and display its transpose.
5. Create a 2D array with dimensions 4x4 using random integers. Find and display the minimum and maximum values in the array.
6. Create a 1D array of 12 integers, then reshape it into a 3x4 matrix. Display both the original array and the reshaped matrix.
7. Given a 5x5 array of random integers, replace all elements greater than 20 with the value 20 and display the updated array.

**Pandas DataFrame Questions**

1. Create a DataFrame with two columns: Name and Age, with 5 rows of data. Display the DataFrame.
2. Import a CSV file containing student data (Name, Age, Grade) into a pandas DataFrame. Display the first 5 rows of the DataFrame.
3. Given a DataFrame with columns: Product, Price, and Quantity, calculate the total value (Price \* Quantity) for each row and store it in a new column Total. Display the updated DataFrame.
4. Create a DataFrame with missing values in one of the columns. Use the fillna() function to fill the missing values with a specified value and display the DataFrame.
5. Create a DataFrame with 5 rows and 3 columns. Sort the DataFrame by the values in one of the columns and display the sorted DataFrame.
6. Given a DataFrame with Name and Score columns, filter and display the rows where the score is greater than 50.
7. Create a DataFrame with 3 columns (A, B, C) and 5 rows of random integers. Add a new column D that contains the cumulative sum of column A. Display the updated DataFrame.

# Submission Instructions

Always read the submission instructions carefully.

* Rename your Jupyter notebook to **AI\_Lab3\_roll number** and download the notebook as **.ipynb**

extension.

* To download the required file, go to **File->Download .ipynb**
* Only submit the **.ipynb** file. DO NOT **zip** or **rar** your submission file
* Submit this file on Google Classroom under the relevant assignment.
* Late submissions will not be accepted