

**SCHOOL OF COMPUTING**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**UNIT – V – Introduction to Python– SCSA3016**

**Data Structures-Functions-Numpy-Matplotlib-Pandas-Problems based on Computational Complexity-Simple Case Studies based on Python(Binary Search, Common elements in list),Hash tables, Dictionary**

**5.1. INTRODUCTION TO DATA SCIENCE WITH PYTHON**

The main focus of businesses using big data was on building frameworks that can store a large amount of data. Then, frameworks like Hadoop were created, which helped in storing massive amounts of [data](https://www.simplilearn.com/what-is-data-article).With the problem of storage solved, the focus then shifted to processing the data that is stored. This is where data science came in as the future for processing and analysing data. Now, data science has become an integral part of all the businesses that deal with large amounts of data. Companies today hire data scientists and professionals who take the data and turn it into a meaningful resource.

**What is Data Science?** Data science is all about finding and exploring data in the real world and using that knowledge to solve business problems. Some examples of data science are:

* Customer Prediction - System can be trained based on customer behavior patterns to predict the likelihood of a customer buying a product
* Service Planning - Restaurants can predict how many customers will visit on the weekend and plan their food inventory to handle the demand

**Why Python?** When it comes to data science, we need some sort of programming language or tool, like Python. Although there are other [tools for data science](https://www.simplilearn.com/popular-data-science-tools-article), like R and SAS, we will focus on Python and how it is beneficial for data science in this article.

* Python as a programming language has become very popular in recent times. It has been used in data science, [IoT](https://www.simplilearn.com/what-is-iot-how-and-why-it-matters-article), [AI](https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/what-is-artificial-intelligence), and other technologies, which has added to its popularity.
* Python is used as a [programming language for data science](https://www.simplilearn.com/top-data-science-programming-languages-article) because it contains costly tools from a mathematical or statistical perspective. It is one of the significant reasons why data scientists around the world use Python. If you track the trends over the past few years, you will notice that Python has become the programming language of choice, particularly for data science.
* There are several other reasons why Python is one of the most used programming languages for data science, including:
* Speed - Python is relatively faster than other programming languages
* Availability - There are a significant number of packages available that other users have developed, which can be reused
* Design goal - The syntax roles in Python are intuitive and easy to understand, thereby helping in building applications with a readable codebase.

**Python** has been used worldwide for different fields such as making [websites](https://www.edureka.co/blog/django-tutorial/), [artificial intelligence](https://www.edureka.co/blog/pros-and-cons-of-ai/) and much more. But to make all of this possible, **data** plays a very important role which means that this data should be stored efficiently and the access to it must be timely. So how do you achieve this? We use something called Data Structures. With that being said, let us go through the topics we will cover in **Data Structures** in [Python](https://www.edureka.co/blog/python-basics/).

* [What is a Data Structure?](https://www.edureka.co/blog/data-structures-in-python/#datastructure)
* [Types of Data Structures in Python](https://www.edureka.co/blog/data-structures-in-python/#types)
* [Built-in Data Structures](https://www.edureka.co/blog/data-structures-in-python/#builtin)
  + [List](https://www.edureka.co/blog/data-structures-in-python/#list)
  + [Dictionary](https://www.edureka.co/blog/data-structures-in-python/#dictionary)
  + [Tuple](https://www.edureka.co/blog/data-structures-in-python/#tuple)
  + [Sets](https://www.edureka.co/blog/data-structures-in-python/#set)
* [User-Defined Data Structures](https://www.edureka.co/blog/data-structures-in-python/#user)
  + [Arrays vs. List](https://www.edureka.co/blog/data-structures-in-python/#arrayvslist)
  + [Stack](https://www.edureka.co/blog/data-structures-in-python/#stack)
  + [Queue](https://www.edureka.co/blog/data-structures-in-python/#queue)
  + [Trees](https://www.edureka.co/blog/data-structures-in-python/#tree)
  + [Linked Lists](https://www.edureka.co/blog/data-structures-in-python/#linkedlist)
  + [Graphs](https://www.edureka.co/blog/data-structures-in-python/#graph)
  + [HashMaps](https://www.edureka.co/blog/data-structures-in-python/#hashmap)

**5.2. DATA STRUCTURE**

**Organizing**, **managing** and **storing** data is important as it enables easier access and efficient modifications. Data Structures allows you to organize your data in such a way that enables you to store collections of data, relate them and perform operations on them accordingly.

**Types of Data Structures in Python**

Python has **implicit** support for Data Structures which enable you to store and access data. These structures are called [List](https://www.edureka.co/blog/lists-in-python/), [Dictionary](https://www.edureka.co/blog/dictionary-in-python/), [Tuple](https://www.edureka.co/blog/tuple-in-python/) and [Set](https://www.edureka.co/blog/sets-in-python/).

Python allows its users to create their own Data Structures enabling them to have **full control** over their [functionality](https://www.edureka.co/blog/python-functions). The most prominent Data Structures are Stack, Queue, Tree, Linked List and so on which are also available to you in other programming languages. So now that you know what are the types available to you, why don’t we move ahead to the Data Structures and implement them using Python.



**5.2.1. Built-in Data Structures**

As the name suggests, these Data Structures are built-in with Python which makes programming easier and helps programmers use them to obtain solutions faster.

**Lists**

[Lists](https://www.edureka.co/blog/lists-in-python/) are used to store data of different data types in a sequential manner. There are addresses assigned to every element of the list, which is called as Index. The index value starts from 0 and goes on until the last element called the **positive index**. There is also **negative indexing** which starts from -1 enabling you to access elements from the last to first.

**Dictionary**

[Dictionaries](https://www.edureka.co/blog/dictionary-in-python/) are used to store **key-value** pairs. To understand better, think of a phone directory where hundreds and thousands of names and their corresponding numbers have been added. Now the constant values here are Name and the Phone Numbers which are called as the keys. And the various names and phone numbers are the values that have been fed to the keys. If you access the values of the keys, you will obtain all the names and phone numbers. So that is what a key-value pair is. And in Python, this structure is stored using Dictionaries. Let us understand this better with an example program.

**Tuple**

[Tuples](https://www.edureka.co/blog/tuple-in-python/) are the same as lists are with the exception that the data once entered into the tuple cannot be changed no matter what. The only exception is when the data inside the tuple is mutable, only then the tuple data can be changed. The example program will help you understand better.

**Sets**

[Sets](https://www.edureka.co/blog/sets-in-python/) are a collection of unordered elements that are unique. Meaning that even if the data is repeated more than one time, it would be entered into the set only once. It resembles the sets that you have learnt in arithmetic. The operations also are the same as is with the arithmetic sets. An example program would help you understand better.

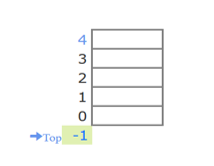
**5.2.3. User-Defined Data Structures**

**Arrays vs. Lists**

Arrays and lists are the same structure with one difference. Lists allow heterogeneous data element storage whereas [Arrays](https://www.edureka.co/blog/arrays-in-python/) allow only homogenous elements to be stored within them.

**Stack**

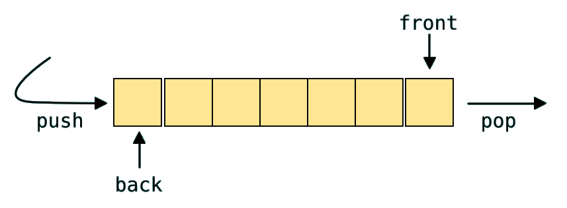
[Stacks](https://www.edureka.co/blog/stack-in-python/) are linear Data Structures which are based on the principle of Last-In-First-Out (LIFO) where data which is entered last will be the first to get accessed. It is built using the array structure and has operations namely, pushing (adding) elements, popping (deleting) elements and accessing elements only from one point in the stack called as the TOP. This TOP is the pointer to the current position of the stack. Stacks are prominently used in applications such as Recursive Programming, reversing words, undo mechanisms in word editors and so forth.



**Figure 5.1: Stack**

**Queue**

A [queue](https://www.edureka.co/blog/queue-data-structure-in-python/) is also a linear data structure which is based on the principle of First-In-First-Out (FIFO) where the data entered first will be accessed first. It is built using the array structure and has operations which can be performed from both ends of the Queue, that is, head-tail or front-back. Operations such as adding and deleting elements are called En-Queue and De-Queue and accessing the elements can be performed. Queues are used as Network Buffers for traffic congestion management, used in Operating Systems for Job Scheduling and many more.



**Figure 5.2: Queue**

**Tree**

Trees are non-linear Data Structures which have root and nodes. The root is the node from where the data originates and the nodes are the other data points that are available to us. The node that precedes is the parent and the node after is called the child. There are levels a tree has to show the depth of information. The last nodes are called the leaves. Trees create a hierarchy which can be used in a lot of real-world applications such as the [HTML](https://www.edureka.co/blog/what-is-html/) pages use trees to distinguish which tag comes under which block. It is also efficient in searching purposes and much more.

**Linked List**

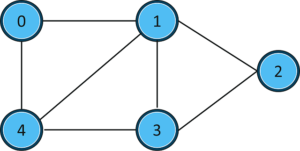
[Linked lists](https://www.edureka.co/blog/linked-list-in-python/) are linear Data Structures which are not stored consequently but are linked with each other using pointers. The node of a linked list is composed of data and a pointer called next. These structures are most widely used in image viewing applications, music player applications and so forth.



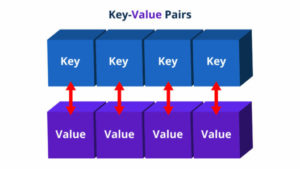
**Figure 5.3: LinkedList**

**Graph**

Graphs are used to store data collection of points called vertices (nodes) and edges (edges). Graphs can be called as the most accurate representation of a real-world map. They are used to find the various cost-to-distance between the various data points called as the nodes and hence find the least path. Many applications such as Google Maps, Uber, and many more use Graphs to find the least distance and increase profits in the best ways.

  
**Figure 5.4:** **HashMaps**

[HashMaps](https://www.edureka.co/blog/dictionary-in-python/) are the same as what dictionaries are in Python. They can be used to implement applications such as phonebooks, populate data according to the lists and much more.



**Figure 5.5: HashMaps**

That wraps up all the prominent Data Structures in Python. I hope you have understood built-in as well as the user-defined [Data Structures](https://www.edureka.co/blog/data-structures-algorithms-in-java/) that we have in Python and why they are important.

**5.3. FUNCTION IN PYTHON**

* A function is a block of code which only runs when it is called.
* You can pass data, known as parameters, into a function.
* A function can return data as a result.
* Python Functions is a block of related statements designed to perform a computational, logical, or evaluative task. The idea is to put some commonly or repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs, we can do the function calls to reuse code contained in it over and over again.
* Functions can be both built-in or user-defined. It helps the program to be concise, non-repetitive, and organized.

**Syntax**

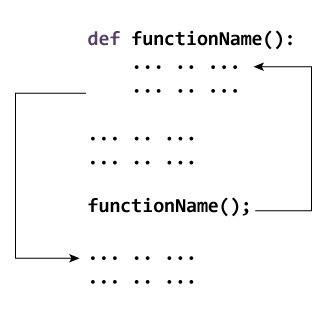
def functionname( parameters ):

"function\_docstring"

function\_suite

return [expression]

**How Function works in Python?**



**Types of Functions**

Basically, we can divide functions into the following two types:

1. [**Built-in functions**](https://www.programiz.com/python-programming/methods/built-in) - Functions that are built into Python.
2. [**User-defined functions**](https://www.programiz.com/python-programming/user-defined-function) - Functions defined by the users themselves.

## **Creating a Function**

In Python a function is defined using the def keyword:

**Example**

def my\_function():

print("Hello from a function")

**Calling a Function**

To call a function, use the function name followed by parenthesis:

**Example**

def my\_function():

print("Hello from a function")

my\_function()

**Function Arguments**

You can call a function by using the following types of formal arguments −

* + Required arguments
  + Keyword arguments
  + Default arguments
  + Variable-length arguments

**Arguments of a Function**

Arguments are the values passed inside the parenthesis of the function. A function can have any number of arguments separated by a comma.

**Example: Python Function with arguments**

In this example, we will create a simple function to check whether the number passed as an argument to the function is even or odd.

# A simple Python function to check

# whether x is even or odd

def evenOdd(x):

if (x % 2 == 0):

print("even")

else:

print("odd")

# Driver code to call the function

evenOdd(2)

evenOdd(3)

**Output**

even

odd

**Types of Arguments**

Python supports various types of arguments that can be passed at the time of the function call.

**Required arguments**

Required arguments are the arguments passed to a function in correct positional order. Here, the number of arguments in the function call should match exactly with the function definition.

# Function definition is here

def printme( str ):

"This prints a passed string into this function"

print str

return;

# Now you can call printme function

printme()

When the above code is executed, it produces the following result −

Traceback (most recent call last):

File "test.py", line 11, in <module>

printme();

TypeError: printme() takes exactly 1 argument (0 given)

**Keyword arguments**

Keyword arguments are related to the function calls. When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name.

# Function definition is here

def printinfo( name, age ):

"This prints a passed info into this function"

print "Name: ", name

print "Age ", age

return;

# Now you can call printinfo function

printinfo( age=50, name="miki" )

When the above code is executed, it produces the following result −

Name: miki

Age 50

**Default arguments**

A default argument is a parameter that assumes a default value if a value is not provided in the function call for that argument. The following example illustrates Default arguments.

|  |
| --- |
| # Python program to demonstrate  # default arguments      def myFun(x, y=50):      print("x: ", x)      print("y: ", y)      # Driver code (We call myFun() with only  # argument)  myFun(10) |

**Output**

('x: ', 10)

('y: ', 50)

**5.4. PYTHON LIBRARIES FOR DATA ANALYSIS**

Python is a simple programming language to learn, and there is some basic stuff that you can do with it, like adding, printing statements, and so on. However, if you want to perform [data analysis](https://www.simplilearn.com/data-analysis-methods-process-types-article), you need to import specific libraries. Some examples include:

* [Pandas](https://www.simplilearn.com/tutorials/python-tutorial/python-pandas) - Used for structured data operations
* [NumPy](https://www.simplilearn.com/tutorials/python-tutorial/numpy-tutorial) - A powerful library that helps you create n-dimensional arrays
* SciPy - Provides scientific capabilities, like linear algebra and Fourier transform
* [Matplotlib](https://www.simplilearn.com/tutorials/python-tutorial/matplotlib) - Primarily used for visualization purposes
* [Scikit-learn](https://www.simplilearn.com/tutorials/python-tutorial/scikit-learn) - Used to perform all machine learning activities

In addition to these, there are other libraries as well, like:

* Networks & I graph
* TensorFlow
* BeautifulSoup
* OS

**5.5. NumPy**

* **NumPy**, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed.
* NumPy is a Python package. It stands for ‘Numerical Python’. It is a library consisting of multidimensional array objects and a collection of routines for processing of array.
* **Numeric**, the ancestor of NumPy, was developed by Jim Hugunin. Another package Numarray was also developed, having some additional functionalities. In 2005, Travis Oliphant created NumPy package by incorporating the features of Numarray into Numeric package. There are many contributors to this open-source project.
* NumPy – A Replacement for MatLab
* NumPy is often used along with packages like **SciPy** (Scientific Python) and **Matplotlib** (plotting library). This combination is widely used as a replacement for MatLab, a popular platform for technical computing. However, Python alternative to MatLab is now seen as a more modern and complete programming language.
* It is open-source, which is an added advantage of NumPy.
* The most important object defined in NumPy is an N-dimensional array type called **ndarray**. It describes the collection of items of the same type. Items in the collection can be accessed using a zero-based index.
* Every item in a ndarray takes the same size as the block in the memory. Each element in ndarray is an object of the data-type object (called **dtype**).
* Any item extracted from ndarray object (by slicing) is represented by a Python object of one of array scalar types.
* NumPy is the fundamental package for scientific computing with Python. It contains:
* Powerful N-dimensional array objects
* Tools for integrating C/C++, and Fortran code
* It has useful linear algebra, Fourier transform, and random number capabilities
* **Operations using NumPy**
* Using NumPy, a developer can perform the following operations −
* Mathematical and logical operations on arrays.
* Fourier transforms and routines for shape manipulation.
* Operations related to linear algebra. NumPy has in-built functions for linear algebra and random number generation.
* An instance of ndarray class can be constructed by different array creation routines described later in the tutorial. The basic ndarray is created using an array function in NumPy as follows

**numpy.array**

* It creates a ndarray from any object exposing an array interface, or from any method that returns an array.

**numpy.array(object, dtype = None, copy = True, order = None, subok = False, ndmin = 0)**

* The **ndarray** object consists of a contiguous one-dimensional segment of computer memory, combined with an indexing scheme that maps each item to a location in the memory block. The memory block holds the elements in row-major order (C style) or a column-major order (FORTRAN or MatLab style).

**The above constructor takes the following parameters** −

|  |  |
| --- | --- |
| Sr.No. | Parameter & Description |
| 1 | **object** Any object exposing the array interface method returns an array or any (nested) sequence. |
| 2 3 | **dtype** The desired data type of array, optional**copy**Optional. By default (true), the object is copied |
| 4 | **order** C (row-major) or F (column-major) or A (any) (default) |
| 5 | **subok** By default, returned array forced to be a base class array. If true, sub-classes passed through |
| 6 | **ndmin** Specifies minimum dimensions of the resultant array |

**Example 1**

import numpy as np

a = np.array([1,2,3])

print(a)

The output is as follows –

[1, 2, 3]

The **ndarray** object consists of a contiguous one-dimensional segment of computer memory, combined with an indexing scheme that maps each item to a location in the memory block.

**5.5.1. NumPy – Data Types**

**bool\_**

Boolean (True or False) stored as a byte

**int\_**

Default integer type (same as C long; normally either int64 or int32)

**intc**

Identical to C int (normally int32 or int64)

**intp**

An integer used for indexing (same as C ssize\_t; normally either int32 or int64)

**int8**

Byte (-128 to 127)

**int16**

Integer (-32768 to 32767)

**float\_**

Shorthand for float64

**float64**

Double precision float: sign bit, 11 bits exponent, 52 bits mantissa

**float64**

Double precision float: sign bit, 11 bits exponent, 52 bits mantissa

**complex\_**

Shorthand for complex128

**complex64**

Complex number, represented by two 32-bit floats (real and imaginary components)

**complex128**

Complex number, represented by two 64-bit floats (real and imaginary components)

NumPy numerical types are instances of dtype (data-type) objects, each having unique characteristics. The dtypes are available as np.bool\_, np.float32, etc.

**Data Type Objects (dtype)**

A data type object describes the interpretation of a fixed block of memory corresponding to an array, depending on the following aspects −

* Type of data (integer, float or Python object)
* Size of data
* Byte order (little-endian or big-endian)
* In case of structured type, the names of fields, data type of each field and part of the memory block taken by each field.
* If the data type is a subarray, its shape and data type

The byte order is decided by prefixing ‘<‘ or ‘>’ to the data type. ‘<‘ means that encoding is little-endian (least significant is stored in smallest address). ‘>’ means that encoding is big-endian (a most significant byte is stored in smallest address).

**A dtype object is constructed using the following syntax** −

numpy.dtype(object, align, copy)

The parameters are −

* **Object** − To be converted to data type object
* **Align** − If true, adds padding to the field to make it similar to C-struct
* **Copy** − Makes a new copy of dtype object. If false, the result is a reference to builtin data type object

**Example 1**

# using array-scalar type

import numpy as np

dt = np.dtype(np.int32)

print(dt)

The output is as follows − int32

**5.5.2. ndarray.shape**

This array attribute returns a tuple consisting of array dimensions. It can also be used to resize the array.

**Example 1**

import numpy as np

a = np.array([[1,2,3],[4,5,6]])

print (a.shape)

The output is as follows −(2, 3)

**Example 2**

# this resizes the ndarray

import numpy as np

a = np.array([[1,2,3],[4,5,6]])

a.shape = (3,2)

print(a)

The output is as follows -[[1, 2][3, 4] [5, 6]]

**5.5.3. ndarray.ndim**

This array attribute returns the number of array dimensions.

**Example 1**

# an array of evenly spaced numbers

import numpy as np

a = np.arange(24)

print(a)

The output is as follows –

[0 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  [16 17 18 19 20 21 22](tel:16%2017%2018%2019%2020%2021%2022) 23]

**Example 2**

# this is one dimensional array

import numpy as np

a = np.arange(24)

a.ndim

# now reshape it

b = a.reshape(2,4,3)

print(b)

# b is having three dimensions

The output is as follows –

[[[ 0,  1,  2]

  [ 3,  4,  5]

  [ 6,  7,  8]

  [ 9, 10, 11]]

  [[12, 13, 14]

   [15, 16, 17]

   [18, 19, 20]

   [21, 22, 23]]]

**5.5.4. numpy.itemsize**

This array attribute returns the length of each element of array in bytes.

**Example 1**

# dtype of array is int8 (1 byte)

import numpy as np

x = np.array([1,2,3,4,5], dtype = np.int8)

print (x.itemsize)

**5.5.5.** **numpy.flags**

The ndarray object has the following attributes. Its current values are returned by this function.

|  |  |
| --- | --- |
| **Sr.No.** | **Attribute & Description** |
| 1 | **C\_CONTIGUOUS (C)**The data is in a single, C-style contiguous segment |
| 2 | **F\_CONTIGUOUS (F)**The data is in a single, Fortran-style contiguous segment |
| 3 | **OWNDATA (O)**The array owns the memory it uses or borrows it from another object |
| 4 | **WRITEABLE (W)**The data area can be written to. Setting this to False locks the data, making it read-only |
| 5 | **ALIGNED (A)**The data and all elements are aligned appropriately for the hardware |
| 6 | **UPDATEIFCOPY (U)**This array is a copy of some other array. When this array is deallocated, the base array will be updated with the contents of this array |

**Example**

The following example shows the current values of flags.

import numpy as np

x = np.array([1,2,3,4,5])

print(x.flags)

The output is as follows −

C\_CONTIGUOUS : True

F\_CONTIGUOUS : True

OWNDATA : True

WRITEABLE : True

ALIGNED : True

UPDATEIFCOPY : False

**5.5.6. NumPy – Array Creation Routines**

A new **ndarray** object can be constructed by any of the following array creation routines or using a low-level ndarray constructor.

numpy.empty

It creates an uninitialized array of specified shape and dtype. It uses the following constructor −

numpy.empty(shape, dtype = float, order = ‘C’)

**The constructor takes the following parameters.**

|  |  |
| --- | --- |
| **Sr.No.** | **Parameter & Description** |
| 1 | **Shape:** Shape of an empty array in int or tuple of int |
| 2 | **Dtype:** Desired output data type. Optional |
| 3 | **Order:** ‘C’ for C-style row-major array, ‘F’ for FORTRAN style column- |

**Example**

The following code shows an example of an empty array.

import numpy as np

x = np.empty([3,2], dtype = int)

print(x)

The output is as follows −[[22649312    1701344351]

[1818321759  1885959276] [16779776    156368896]]

**5.5.7. numpy.zeros**

Returns a new array of specified size, filled with zeros.

numpy.zeros(shape, dtype = float, order = ‘C’)

**Example 1**

# array of five ones. Default dtype is float

import numpy as np

x = np.ones(5)

print(x)

The output is as follows −

[ 1.  1.  1.  1.  1.]

**5.5.8. NumPy – Indexing & Slicing**

Contents of ndarray object can be accessed and modified by indexing or slicing, just like Python’s in-built container objects. items in ndarray object follows zero-based index. Three types of indexing methods are available − **field access, basic slicing** and **advanced indexing**.

* Basic slicing is an extension of Python’s basic concept of slicing to n dimensions. A Python slice object is constructed by giving **start, stop**, and **step** parameters to the built-in **slice** function. This slice object is passed to the array to extract a part of array.

**Example 1**

import numpy as np

a = np.arange(10)

s = slice(2,7,2)

print (a[s])

Its output is as follows −

[2  4  6]

In the above example, an **ndarray** object is prepared by **arange()** function. Then a slice object is defined with start, stop, and step values 2, 7, and 2 respectively. When this slice object is passed to the ndarray, a part of it starting with index 2 up to 7 with a step of 2 is sliced.

**5.5.9. NumPy – Advanced Indexing**

It is possible to make a selection from ndarray that is a non-tuple sequence, ndarray object of integer or Boolean data type, or a tuple with at least one item being a sequence object. Advanced indexing always returns a copy of the data. As against this, the slicing only presents a view.

There are two types of advanced indexing − **Integer** and **Boolean**.

**Integer Indexing**

* This mechanism helps in selecting any arbitrary item in an array based on its N-dimensional index. Each integer array represents the number of indexes into that dimension. When the index consists of as many integer arrays as the dimensions of the target ndarray, it becomes straightforward.
* In the following example, one element of the specified column from each row of ndarray object is selected. Hence, the row index contains all row numbers, and the column index specifies the element to be selected.

**Example 1**

import numpy as np

x = np.array([[1, 2], [3, 4], [5, 6]])

y = x[[0,1,2], [0,1,0]]

print(y)

Its output would be as follows −

[1  4  5]

* The selection includes elements at (0,0), (1,1) and (2,0) from the first array.
* In the following example, elements placed at corners of a 4X3 array are selected. The row indices of selection are [0, 0] and [3,3] whereas the column indices are [0,2] and [0,2].
* Advanced and basic indexing can be combined by using one slice (:) or ellipsis (…) with an index array. The following example uses a slice for the advanced index for column. The result is the same when a slice is used for both. But advanced index results in copy and may have different memory layout.

**Boolean Array Indexing**

This type of advanced indexing is used when the resultant object is meant to be the result of Boolean operations, such as comparison operators.

**Example 1**

In this example, items greater than 5 are returned as a result of Boolean indexing.

import numpy as np

x = np.array([[ 0,  1,  2],[ 3,  4,  5],[ 6,  7,  8],[ 9, 10, 11]])

print (‘Our array is:’)

print(x)

print ‘\n’

# Now we will print the items greater than 5

print (‘The items greater than 5 are:’)

print (x[x > 5])

The output of this program would be −

Our array is:

[[ 0  1  2]

 [ 3  4  5]

 [ 6  7  8]

 [ 9 10 11]]

The items greater than 5 are:

[ 6  7  8  9 10 11]

**5.5.10. NumPy – Broadcasting**

The term **broadcasting** refers to the ability of NumPy to treat arrays of different shapes during arithmetic operations. Arithmetic operations on arrays are usually done on corresponding elements. If two arrays are of exactly the same shape, then these operations are smoothly performed.

**Example 1**

import numpy as np

a = np.array([1,2,3,4])

b = np.array([10,20,30,40])

c = a \* b

print(c)

Its output is as follows −[10   40   90   160]

* If the dimensions of the two arrays are dissimilar, element-to-element operations are not possible. However, operations on arrays of non-similar shapes is still possible in NumPy, because of the broadcasting capability. The smaller array is **broadcast** to the size of the larger array so that they have compatible shapes.

**5.5.11. NumPy – Iterating Over Array**

NumPy package contains an iterator object **numpy.nditer**. It is an efficient multidimensional iterator object using which it is possible to iterate over an array. Each element of an array is visited using Python’s standard Iterator interface.

Let us create a 3X4 array using arrange() function and iterate over it using **nditer**.

**5.5.12. NumPy – Array Manipulation**

Several routines are available in NumPy package for manipulation of elements in ndarray object. They can be classified into the following types −

**Changing Shape**

|  |  |
| --- | --- |
| **Sr.No.** | **Shape & Description** |
| 1 | [reshape](https://www.tutorialspoint.com/numpy/numpy_reshape.htm):Gives a new shape to an array without changing its data |
| 2 | [flat](https://www.tutorialspoint.com/numpy/numpy_ndarray_flat.htm):A 1-D iterator over the array |
| 3 | [flatten](https://www.tutorialspoint.com/numpy/numpy_ndarray_flatten.htm):Returns a copy of the array collapsed into one dimension |
| 4 | [ravel](https://www.tutorialspoint.com/numpy/numpy_ndarray_ravel.htm):Returns a contiguous flattened array |

**Transpose Operations**

|  |  |
| --- | --- |
| **Sr.No.** | **Operation & Description** |
| 1 | [transpose](https://www.tutorialspoint.com/numpy/numpy_transpose.htm):Permutes the dimensions of an array |
| 2 | [ndarray.T](https://www.tutorialspoint.com/numpy/numpy_ndarray_t.htm):Same as self.transpose() |
| 3 | [rollaxis](https://www.tutorialspoint.com/numpy/numpy_rollaxis.htm):Rolls the specified axis backwards |
| 4 | [swapaxes](https://www.tutorialspoint.com/numpy/numpy_swapaxes.htm):Interchanges the two axes of an array |

**Changing Dimensions**

|  |  |
| --- | --- |
| **Sr.No.** | **Dimension & Description** |
| 1 | [broadcast](https://www.tutorialspoint.com/numpy/numpy_broadcast.htm):Produces an object that mimics broadcasting |
| 2 | [broadcast\_to](https://www.tutorialspoint.com/numpy/numpy_broadcast_to.htm):Broadcasts an array to a new shape |
| 3 | [expand\_dims](https://www.tutorialspoint.com/numpy/numpy_expand_dims.htm):Expands the shape of an array |
| 4 | [squeeze](https://www.tutorialspoint.com/numpy/numpy_squeeze.htm):Removes single-dimensional entries from the shape of an array |

**Joining Arrays**

|  |  |
| --- | --- |
| **Sr.No.** | **Array & Description** |
| 1 | [concatenate](https://www.tutorialspoint.com/numpy/numpy_concatenate.htm):Joins a sequence of arrays along an existing axis |
| 2 | [stack](https://www.tutorialspoint.com/numpy/numpy_stack.htm):Joins a sequence of arrays along a new axis |
| 3 | [hstack](https://www.tutorialspoint.com/numpy/numpy_hstack.htm):Stacks arrays in sequence horizontally (column wise) |
| 4 | [vstack](https://www.tutorialspoint.com/numpy/numpy_vstack.htm):Stacks arrays in sequence vertically (row wise) |

**Splitting Arrays**

|  |  |
| --- | --- |
| **Sr.No.** | **Array & Description** |
| 1 | [split](https://www.tutorialspoint.com/numpy/numpy_split.htm):Splits an array into multiple sub-arrays |
| 2 | [hsplit](https://www.tutorialspoint.com/numpy/numpy_hsplit.htm):Splits an array into multiple sub-arrays horizontally (column-wise) |
| 3 | [vsplit](https://www.tutorialspoint.com/numpy/numpy_vsplit.htm):Splits an array into multiple sub-arrays vertically (row-wise) |

**Adding / Removing Elements**

|  |  |
| --- | --- |
| **Sr.No.** | **Element & Description** |
| 1 | [resize](https://www.tutorialspoint.com/numpy/numpy_resize.htm):Returns a new array with the specified shape |
| 2 | [append](https://www.tutorialspoint.com/numpy/numpy_append.htm):Appends the values to the end of an array |
| 3 | [insert](https://www.tutorialspoint.com/numpy/numpy_insert.htm):Inserts the values along the given axis before the given indices |
| 4 | [delete](https://www.tutorialspoint.com/numpy/numpy_delete.htm):Returns a new array with sub-arrays along an axis deleted |
| 5 | [unique](https://www.tutorialspoint.com/numpy/numpy_unique.htm):Finds the unique elements of an array |

**NumPy – Binary Operators**

Following are the functions for bitwise operations available in NumPy package.

|  |  |
| --- | --- |
| **Sr.No.** | **Operation & Description** |
| 1 | [bitwise\_and](https://www.tutorialspoint.com/numpy/numpy_bitwise_and.htm):Computes bitwise AND operation of array elements |
| 2 | [bitwise\_or](https://www.tutorialspoint.com/numpy/numpy_bitwise_or.htm):Computes bitwise OR operation of array elements |
| 3 | [invert](https://www.tutorialspoint.com/numpy/numpy_invert.htm):Computes bitwise NOT |
|  |  |
| 4 | [right\_shift](https://www.tutorialspoint.com/numpy/numpy_right_shift.htm" \t "_blank):Shifts bits of binary representation to the right |

**5.5.13. NumPy – Mathematical Functions**

Quite understandably, NumPy contains a large number of various mathematical operations. NumPy provides standard trigonometric functions, functions for arithmetic operations, handling complex numbers, etc.

**Trigonometric Functions**

NumPy has standard trigonometric functions which return trigonometric ratios for a given angle in radians.

**Example**

import numpy as np

a = np.array([0,30,45,60,90])

print (‘Sine of different angles:’ )

# Convert to radians by multiplying with pi/180

Print(np.sin(a\*np.pi/180))

print (‘\n’)

print(‘Cosine values for angles in array:’\_

print(np.cos(a\*np.pi/180))

print (‘\n’ )

print(‘Tangent values for given angles:’ )

print (np.tan(a\*np.pi/180) )

Here is its output −

Sine of different angles:

[ 0.          0.5         0.70710678  0.8660254   1.        ]

Cosine values for angles in array:

[  1.00000000e+00   8.66025404e-01   7.07106781e-01   5.00000000e-01

   6.12323400e-17]

Tangent values for given angles:

[  0.00000000e+00   5.77350269e-01   1.00000000e+00   1.73205081e+00

   1.63312394e+16]

**arcsin, arcos,** and **arctan** functions return the trigonometric inverse of sin, cos, and tan of the given angle. The result of these functions can be verified by **numpy.degrees() function** by converting radians to degrees.

**Functions for Rounding**

numpy.around()

This is a function that returns the value rounded to the desired precision. The function takes the following parameters.

numpy.around(a,decimals)

Where,

|  |  |
| --- | --- |
| **Sr.No.** | **Parameter & Description** |
| 1 | **a:** Input data |
| 2 | **decimals:** The number of decimals to round to. Default is 0. If negative, the integer is rounded to position to the left of the decimal point |

**5.5.14. NumPy – Statistical Functions**

NumPy has quite a few useful statistical functions for finding minimum, maximum, percentile standard deviation and variance, etc. from the given elements in the array. The functions are explained as follows −

numpy.amin() and numpy.amax()numpy.amin() and numpy.amax()

These functions return the minimum and the maximum from the elements in the given array along the specified axis.

**Example**

import numpy as np

a = np.array([[3,7,5],[8,4,3],[2,4,9]])

print ‘Our array is:’

print(a)

print (‘\n’ )

print ‘Applying amin() function:’

print(np.amin(a,1))

print (‘\n’ )

print ‘Applying amin() function again:’

print (np.amin(a,0))

print (‘\n’ )

print ‘Applying amax() function:’

print (np.amax(a) )

print (‘\n’)

print ‘Applying amax() function again:’

print(np.amax(a, axis = 0))

It will produce the following output −

Our array is:

[[3 7 5]

[8 4 3]

[2 4 9]]

Applying amin() function:

[3 3 2]

Applying amin() function again:

[2 4 3]

Applying amax() function:

9

Applying amax() function again:

[8 7 9]

**5.5.15. numpy.ptp()**

The **numpy.ptp()** function returns the range (maximum-minimum) of values along an axis.

import numpy as np

a = np.array([[3,7,5],[8,4,3],[2,4,9]])

print (‘Our array is:’)

print(a)

print (‘\n’)

print ‘Applying ptp() function:’

print np.ptp(a)

print (‘\n’)

print ‘Applying ptp() function along axis 1:’

print np.ptp(a, axis = 1)

print (‘\n’)

print(‘Applying ptp() function along axis 0:’)

print(np.ptp(a, axis = 0) )

numpy.percentile()

Percentile (or a centile) is a measure used in statistics indicating the value below which a given percentage of observations in a group of observations fall. The function **numpy.percentile()** takes the following arguments.

Where,

|  |  |
| --- | --- |
| **Sr.No.** | **Argument & Description** |
| 1 | **a** Input array |
| 2 | **q** The percentile to compute must be between 0-100 |
| 3 | **axis** The axis along which the percentile is to be calculated |

A variety of sorting related functions are available in NumPy. These sorting functions implement different sorting algorithms, each of them characterized by the speed of execution, worst-case performance, the workspace required and the stability of algorithms. Following table shows the comparison of three sorting algorithms.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **kind** | **speed** | **worst case** | **work space** | **stable** |
| ‘quicksort’ | 1 | O(n^2) | 0 | no |
| ‘mergesort’ | 2 | O(n\*log(n)) | ~n/2 | yes |
| ‘heapsort’ | 3 | O(n\*log(n)) | 0 | no |

**5.5.16. numpy.sort()**

The sort() function returns a sorted copy of the input array. It has the following parameters −

numpy.sort(a, axis, kind, order)

Where,

|  |  |
| --- | --- |
| **Sr.No.** | **Parameter & Description** |
| 1 | **a**Array to be sorted |
| 2 | **axis**The axis along which the array is to be sorted. If none, the array is flattened, sorting on the last axis |
| 3 | **kind**Default is quicksort |
| 4 | **order**If the array contains fields, the order of fields to be sorted |

**5.5.17. NumPy – Byte Swapping**

We have seen that the data stored in the memory of a computer depends on which architecture the CPU uses. It may be little-endian (least significant is stored in the smallest address) or big-endian (most significant byte in the smallest address).

numpy.ndarray.byteswap()

The **numpy.ndarray.byteswap()** function toggles between the two representations: bigendian and little-endian.

**5.5.18. NumPy – Copies & Views**

While executing the functions, some of them return a copy of the input array, while some return the view. When the contents are physically stored in another location, it is called **Copy**. If on the other hand, a different view of the same memory content is provided, we call it as **View**.

**5.5.19. No Copy**

Simple assignments do not make the copy of array object. Instead, it uses the same id() of the original array to access it. The **id()** returns a universal identifier of Python object, similar to the pointer in C.

Furthermore, any changes in either gets reflected in the other. For example, the changing shape of one will change the shape of the other too.

**5.5.20. View or Shallow Copy**

NumPy has **ndarray.view()** method which is a new array object that looks at the same data of the original array. Unlike the earlier case, change in dimensions of the new array doesn’t change dimensions of the original.

**5.5.21. NumPy – Matrix Library**

NumPy package contains a Matrix library **numpy.matlib**. This module has functions that return matrices instead of ndarray objects.

**5.5.22. matlib.empty()**

The **matlib.empty()** function returns a new matrix without initializing the entries. The function takes the following parameters.

numpy.matlib.empty(shape, dtype, order)

Where,

|  |  |
| --- | --- |
| **Sr.No.** | **Parameter & Description** |
| 1 | **shapeint** or tuple of **int** defining the shape of the new matrix |
| 2 | **Dtype** Optional. Data type of the output |
| 3 | **order** C or F |

**Example**

import numpy.matlib

import numpy as np

print(np.matlib.empty((2,2)))

# filled with random data

It will produce the following output −

[[ 2.12199579e-314,   4.24399158e-314]

 [ 4.24399158e-314,   2.12199579e-314]]

numpy.matlib.eye()

This function returns a matrix with 1 along the diagonal elements and the zeros elsewhere. The function takes the following parameters.

numpy.matlib.eye(n, M,k, dtype)

Where,

|  |  |
| --- | --- |
| **Sr.No.** | **Parameter & Description** |
| 1 | **n** The number of rows in the resulting matrix |
| 2 | **M** The number of columns, defaults to n |
| 3 | **k** Index of diagonal |
| 4 | **dtype** Data type of the output |

**Example**

import numpy.matlib

import numpy as np

print np.matlib.eye(n = 3, M = 4, k = 0, dtype = float)

It will produce the following output −

[[ 1.  0.  0.  0.]

 [ 0.  1.  0.  0.]

 [ 0.  0.  1.  0.]]

**5.5.23. Numpy- Linear Algebra**

NumPy package contains **numpy.linalg** module that provides all the functionality required for linear algebra. Some of the important functions in this module are described in the following table.

|  |  |
| --- | --- |
| **Sr.No.** | **Function & Description** |
| 1 | [dot](https://www.tutorialspoint.com/numpy/numpy_dot.htm)  Dot product of the two arrays |
| 2 | [vdot](https://www.tutorialspoint.com/numpy/numpy_vdot.htm)  Dot product of the two vectors |
| 3 | [inner](https://www.tutorialspoint.com/numpy/numpy_inner.htm)  Inner product of the two arrays |
| 4 | [matmul](https://www.tutorialspoint.com/numpy/numpy_matmul.htm)  Matrix product of the two arrays |
| 5 | [determinant](https://www.tutorialspoint.com/numpy/numpy_determinant.htm)  Computes the determinant of the array |
| 6 | [solve](https://www.tutorialspoint.com/numpy/numpy_solve.htm)  Solves the linear matrix equation |
| 7 | [inv](https://www.tutorialspoint.com/numpy/numpy_inv.htm)  Finds the multiplicative inverse of the matrix |

**Addition and Subtraction**

# importing numpy for matrix operations

import numpy

# initializing matrices

x = numpy.array([[1, 2], [4, 5]])

y = numpy.array([[7, 8], [9, 10]])

# using add() to add matrices

print ("The element wise addition of matrix is : ")

print (numpy.add(x,y))

# using subtract() to subtract matrices

print ("The element wise subtraction of matrix is : ")

print (numpy.subtract(x,y))

**Multiplication and Dot Product**

# importing numpy for matrix operations

import numpy

# initializing matrices

x = numpy.array([[1, 2], [4, 5]])

y = numpy.array([[7, 8], [9, 10]])

# using multiply() to multiply matrices element wise

print ("The element wise multiplication of matrix is : ")

print (numpy.multiply(x,y))

# using dot() to multiply matrices

print ("The product of matrices is : ")

print (numpy.dot(x,y))

**Transpose**

“T” :- This argument is used to transpose the specified matrix.

# importing numpy for matrix operations

import numpy

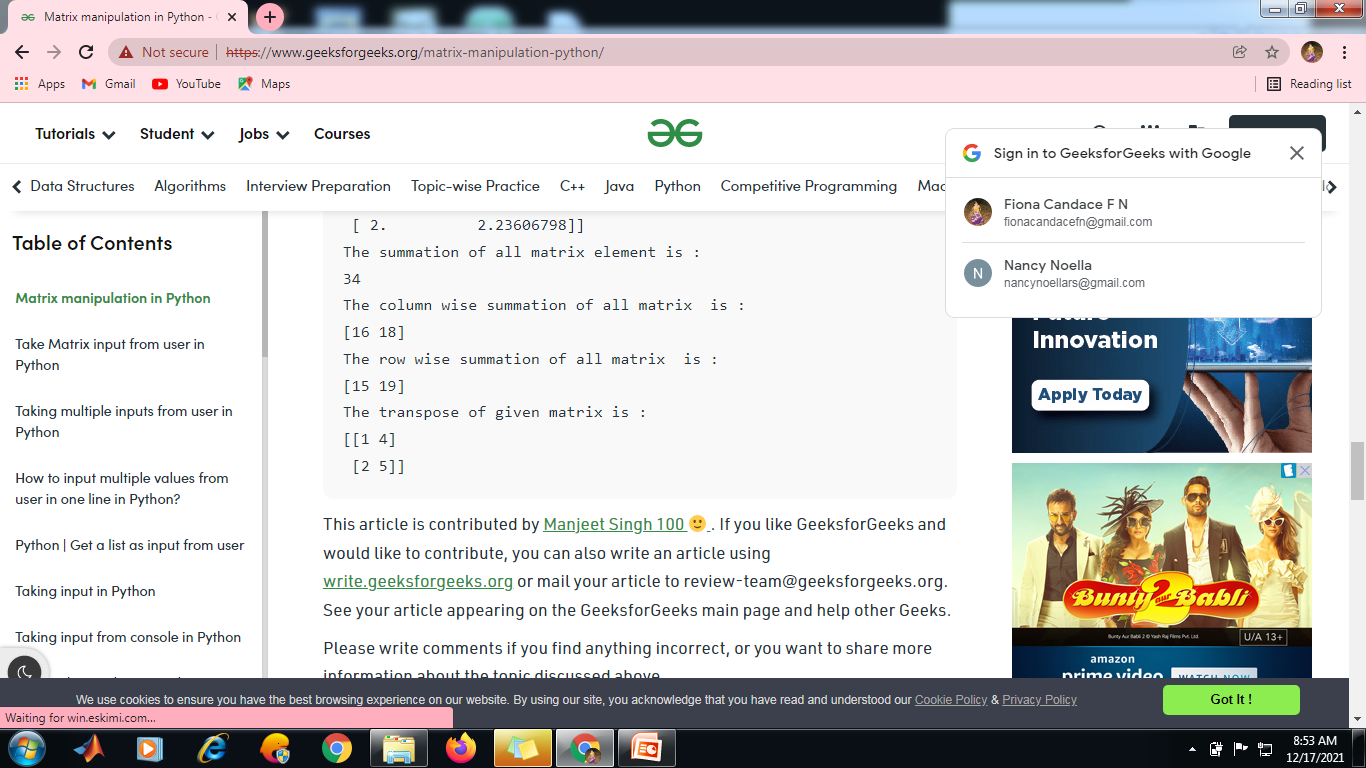
# initializing matrices

x = numpy.array([[1, 2], [4, 5]])

# using "T" to transpose the matrix

print ("The transpose of given matrix is : ")

print (x.T)

****

**Determinant**

****

**Inverse**

# Import required package

import numpy as np

import numpy.linalg as la

# Taking a 3 \* 3 matrix

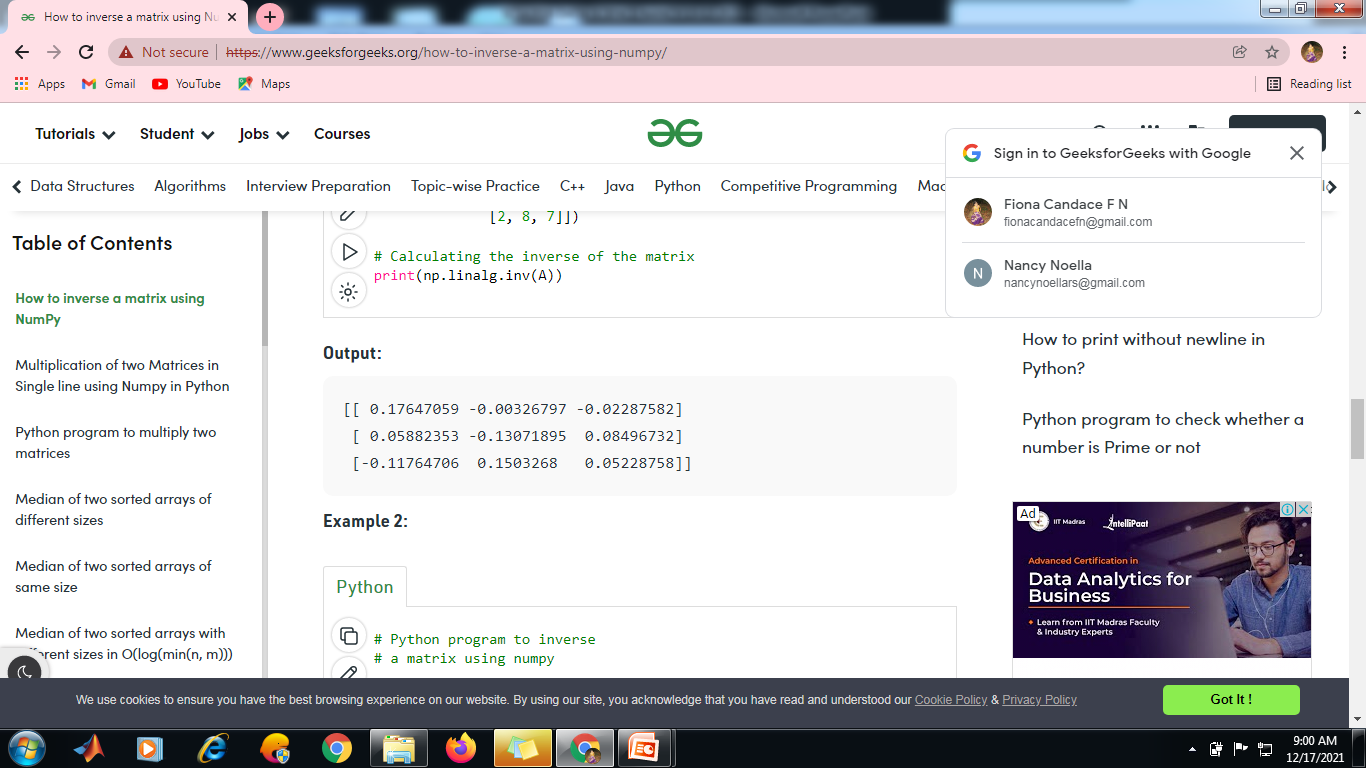
A = np.array([[6, 1, 1],

[4, -2, 5],

[2, 8, 7]])

# Calculating the inverse of the matrix

print(la.inv(A))



**Solve**

**Example-1**

import numpy as np

m\_list = [[4, 3], [-5, 9]]

A = np.array(m\_list)

#To find the inverse of a matrix, the matrix is passed to the linalg.inv() method of the Numpy module

inv\_A = np.linalg.inv(A)

print(inv\_A)

#find the dot product between the inverse of matrix A, and the matrix B.

B = np.array([20, 26])

X = np.linalg.inv(A).dot(B)

print(X)

Output:

[2. 4.]

Here, 2 and 4 are the respective values for the unknowns x and y in *Equation 1*.

**Example 2**

A = np.array([[4, 3, 2], [-2, 2, 3], [3, -5, 2]])

B = np.array([25, -10, -4])

X = np.linalg.inv(A).dot(B)

print(X)

Output:

[ 5. 3. -2.]

**np.arange**

import  numpy as np

#create an array  
arr = np.arange(1,10).reshape(3,3)

#finding the Eigenvalue and Eigenvectors of arr  
np.linalg.eig(arr)

**5.6. Matplotlib**

* Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. It was introduced by John Hunter in the year 2002.
* One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc.

**Installation :**Windows, Linux and macOS distributions have matplotlib and most of its dependencies as wheel packages. Run the following command to install matplotlib package :

python -mpip install -U matplotlib

**Importing matplotlib :**

from matplotlib import pyplot as plt

or

import matplotlib.pyplot as plt

#### **Basic plots in Matplotlib :**

Matplotlib comes with a wide variety of plots. Plots helps to understand trends, patterns, and to make correlations. They’re typically instruments for reasoning about quantitative information. Some of the sample plots are covered here.

1. **Line plot :**

# importing matplotlib module

from matplotlib import pyplot as plt

# x-axis values

x = [5, 2, 9, 4, 7]

# Y-axis values

y = [10, 5, 8, 4, 2]

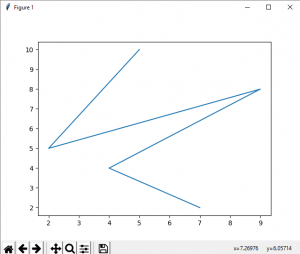
# Function to plot

plt.plot(x,y)

# function to show the plot

plt.show()

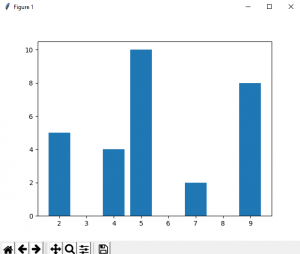
Output:



1. **Bar plot :**

|  |
| --- |
| # importing matplotlib module  from matplotlib import pyplot as plt    # x-axis values  x = [5, 2, 9, 4, 7]    # Y-axis values  y = [10, 5, 8, 4, 2]    # Function to plot the bar  plt.bar(x,y)    # function to show the plot  plt.show() |

Output:



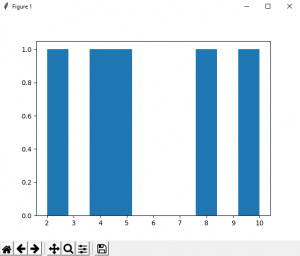
1. **Histogram :**

A histogram is a graph showing frequency distributions.

It is a graph showing the number of observations within each given interval.

|  |
| --- |
| # importing matplotlib module  from matplotlib import pyplot as plt    # Y-axis values  y = [10, 5, 8, 4, 2]    # Function to plot histogram  plt.hist(y)    # Function to show the plot  plt.show() |

Output:



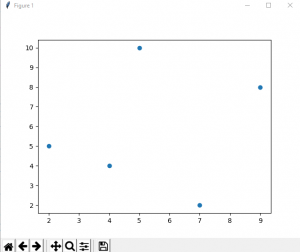
1. **Scatter Plot :**

With Pyplot, you can use the scatter() function to draw a scatter plot.

The scatter() function plots one dot for each observation. It needs two arrays of the same length, one for the values of the x-axis, and one for values on the y-axis:

|  |
| --- |
| # importing matplotlib module  from matplotlib import pyplot as plt    # x-axis values  x = [5, 2, 9, 4, 7]    # Y-axis values  y = [10, 5, 8, 4, 2]    # Function to plot scatter  plt.scatter(x, y)    # function to show the plot  plt.show() |

Output:



**Creating Pie Charts**

With Pyplot, you can use the pie() function to draw pie charts:

import matplotlib.pyplot as plt  
import numpy as np  
  
y = np.array([35, 25, 25, 15])  
  
plt.pie(y)  
plt.show()



**5.7. Pandas**

* **Pandas** is an open-source library that is built on top of NumPy library. It is a Python package that offers various data structures and operations for manipulating numerical data and time series. It is mainly popular for importing and analyzing data much easier. Pandas is fast and it has high-performance & productivity for users.

**What is Python Pandas?**

Pandas is used for data manipulation, analysis and cleaning. Python pandas is well suited for different kinds of data, such as:

* Tabular data with heterogeneously-typed columns
* Ordered and unordered time series data
* Arbitrary matrix data with row & column labels
* Unlabelled data
* Any other form of observational or statistical data sets

**Python Pandas Data Structure**

The primary two components of pandas are the Series and DataFrame.

A Series is essentially a column, and a DataFrame is a multi-dimensional table made up of a collection of Series.



**5.7.1. Series**

**Create a simple Pandas Series from a list:**

import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a)

print(myvar)

**5.7.2. What is a DataFrame?**

A Pandas DataFrame is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns.

It is a widely used data structure of pandas and works with a two-dimensional array with labeled axes (rows and columns). DataFrame is defined as a standard way to store data and has two different indexes, i.e., row index and column index. It consists of the following properties:

* The columns can be heterogeneous types like int, bool, and so on.
* It can be seen as a dictionary of Series structure where both the rows and columns are indexed. It is denoted as "columns" in case of columns and "index" in case of rows.

**Example**

**Create a simple Pandas DataFrame:**

import pandas as pd  
  
data = {  
  "calories": [420, 380, 390],  
  "duration": [50, 40, 45]  
}  
  
#load data into a DataFrame object:  
df = pd.DataFrame(data)  
  
print(df)

**Result**

calories duration

0 420 50

1 380 40

2 390 45

**Create a DataFrame using List:**

We can easily create a DataFrame in Pandas using list.

**import** pandas as pd

# a list of strings

x = ['Python', 'Pandas']

# Calling DataFrame constructor on list

df = pd.DataFrame(x)

**print**(df)

**Output**

0

0 Python

1 Pandas

**Create an empty DataFrame**

The below code shows how to create an empty DataFrame in Pandas:

# importing the pandas library

**import** pandas as pd

df = pd.DataFrame()

**print** (df)

**Output**

Empty DataFrame

Columns: []

Index: []

**Create a DataFrame from Dict of ndarrays/ Lists**

# importing the pandas library

**import** pandas as pd

info = {'ID' :[101, 102, 103],'Department' :['B.Sc','B.Tech','M.Tech',]}

df = pd.DataFrame(info)

**print** (df)

**Output**

ID Department

0 101 B.Sc

1 102 B.Tech

2 103 M.Tech

**Create a DataFrame from Dict of Series:**

# importing the pandas library

**import** pandas as pd

info = {'one' : pd.Series([1, 2, 3, 4, 5, 6], index=['a', 'b', 'c', 'd', 'e', 'f']),

   'two' : pd.Series([1, 2, 3, 4, 5, 6, 7, 8], index=['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'

d1 = pd.DataFrame(info)

**print** (d1)

**Output**

one two

a 1.0 1

b 2.0 2

c 3.0 3

d 4.0 4

e 5.0 5

f 6.0 6

g NaN 7

h NaN 8

**Column Selection**

We can select any column from the DataFrame. Here is the code that demonstrates how to select a column from the DataFrame.

# importing the pandas library

**import** pandas as pd

info = {'one' : pd.Series([1, 2, 3, 4, 5, 6], index=['a', 'b', 'c', 'd', 'e', 'f']),

   'two' : pd.Series([1, 2, 3, 4, 5, 6, 7, 8], index=['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'}

d1 = pd.DataFrame(info)

**print** (d1 ['one'])

Output

a 1.0

b 2.0

c 3.0

d 4.0

e 5.0

f 6.0

g NaN

h NaN

Name: one, dtype: float64

**Column Addition**

We can also add any new column to an existing DataFrame. The below code demonstrates how to add any new column to an existing DataFrame:

# importing the pandas library

**import** pandas as pd

info = {'one' : pd.Series([1, 2, 3, 4, 5], index=['a', 'b', 'c', 'd', 'e']),

   'two' : pd.Series([1, 2, 3, 4, 5, 6], index=['a', 'b', 'c', 'd', 'e', 'f'])}

df = pd.DataFrame(info)

# Add a new column to an existing DataFrame object

**print** ("Add new column by passing series")

df['three']=pd.Series([20,40,60],index=['a','b','c'])

**print** (df)

**print** ("Add new column using existing DataFrame columns")

df['four']=df['one']+df['three']

**print** (df)

**Output**

Add new column by passing series

one two three

a 1.0 1 20.0

b 2.0 2 40.0

c 3.0 3 60.0

d 4.0 4 NaN

e 5.0 5 NaN

f NaN 6 NaN

Add new column using existing DataFrame columns

one two three four

a 1.0 1 20.0 21.0

b 2.0 2 40.0 42.0

c 3.0 3 60.0 63.0

d 4.0 4 NaN NaN

e 5.0 5 NaN NaN

f NaN 6 NaN NaN

**Column Deletion:**

We can also delete any column from the existing DataFrame. This code helps to demonstrate how the column can be deleted from an existing DataFrame:

# importing the pandas library

**import** pandas as pd

info = {'one' : pd.Series([1, 2], index= ['a', 'b']),

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

df = pd.DataFrame(info)

**print** ("The DataFrame:")

**print** (df)

# using del function

**print** ("Delete the first column:")

**del** df['one']

**print** (df)

# using pop function

**print** ("Delete the another column:")

df.pop('two')

**print** (df)

**Output**

The DataFrame:

one two

a 1.0 1

b 2.0 2

c NaN 3

Delete the first column:

two

a 1

b 2

c 3

Delete the another column:

Empty DataFrame

Columns: []

Index: [a, b, c]

**Row Selection, Addition, and Deletion**

**Row Selection:**

We can easily select, add, or delete any row at anytime. First of all, we will understand the row selection. Let's see how we can select a row using different ways that are as follows:

**Selection by Label:**

We can select any row by passing the row label to a **loc** function.

# importing the pandas library

**import** pandas as pd

info = {'one' : pd.Series([1, 2, 3, 4, 5], index=['a', 'b', 'c', 'd', 'e']),

   'two' : pd.Series([1, 2, 3, 4, 5, 6], index=['a', 'b', 'c', 'd', 'e', 'f'])}

df = pd.DataFrame(info)

**print** (df.loc['b'])

**Output**

one 2.0

two 2.0

Name: b, dtype: float64

**Selection by integer location:**

The rows can also be selected by passing the integer location to an **iloc**function.

# importing the pandas library

**import** pandas as pd

info = {'one' : pd.Series([1, 2, 3, 4, 5], index=['a', 'b', 'c', 'd', 'e']),

   'two' : pd.Series([1, 2, 3, 4, 5, 6], index=['a', 'b', 'c', 'd', 'e', 'f'])}

df = pd.DataFrame(info)

**print** (df.iloc[3])

**Output**

one 4.0

two 4.0

Name: d, dtype: float64

**Slice Rows**

It is another method to select multiple rows using **':'** operator.

# importing the pandas library

**import** pandas as pd

info = {'one' : pd.Series([1, 2, 3, 4, 5], index=['a', 'b', 'c', 'd', 'e']),

   'two' : pd.Series([1, 2, 3, 4, 5, 6], index=['a', 'b', 'c', 'd', 'e', 'f'])}

df = pd.DataFrame(info)

**print** (df[2:5])

**Output**

one two

c 3.0 3

d 4.0 4

e 5.0 5

**Addition of rows:**

We can easily add new rows to the DataFrame using **append** function. It add the new rows at the end.

# importing the pandas library

**import** pandas as pd

d = pd.DataFrame([[7, 8], [9, 10]], columns = ['x','y'])

d2 = pd.DataFrame([[11, 12], [13, 14]], columns = ['x','y'])

d = d.append(d2)

**print** (d)

**Output**

x y

0 7 8

1 9 10

0 11 12

1 13 14

**Deletion of rows:**

We can delete or drop any rows from a DataFrame using the **index** label. If in case, the label is duplicate then multiple rows will be deleted.

# importing the pandas library

**import** pandas as pd

a\_info = pd.DataFrame([[4, 5], [6, 7]], columns = ['x','y'])

b\_info = pd.DataFrame([[8, 9], [10, 11]], columns = ['x','y'])

a\_info = a\_info.append(b\_info)

# Drop rows with label 0

a\_info = a\_info.drop(0)

**Output**

x y

1 6 7

1 10 11

# **5.7.3. Pandas Read CSV**

A simple way to store big data sets is to use CSV files (comma separated files).

CSV files contains plain text and is a well know format that can be read by everyone including Pandas.

**Load the CSV into a DataFrame:**

import pandas as pd  
  
df = pd.read\_csv('data.csv')  
  
print(df.to\_string())

**Print the DataFrame without the to\_string() method:**

import pandas as pd  
  
df = pd.read\_csv('data.csv')  
  
print(df)

## **Viewing the Data**

One of the most used method for getting a quick overview of the DataFrame, is the head() method.

The head() method returns the headers and a specified number of rows, starting from the top.

**Example**

import pandas as pd  
  
df = pd.read\_csv('data.csv')  
  
print(df.head(10))

**There is also a tail() method for viewing the *last* rows of the DataFrame.**

The tail() method returns the headers and a specified number of rows, starting from the bottom.

**Example**

Print the last 5 rows of the DataFrame:

print(df.tail())

## **Info About the Data**

The DataFrames object has a method called info(), that gives you more information about the data set.

**Example:** Print information about the data:

print(df.info())

**5.8. PROBLEM BASED ON COMPUTATIONAL COMPLEXITY**

**Computational Complexity**

* Computational complexity is a field from computer science which analyzes algorithms based on the amount resources required for running it. The amount of required resources varies based on the input size, so the complexity is generally expressed as a function of n, where n is the size of the input.
* It is important to note that when analyzing an algorithm we can consider the time complexity and space complexity. The space complexity is basically the amount of memory space required to solve a problem in relation to the input size. Even though the space complexity is important when analyzing an algorithm, in this story we will focus only on the time complexity.

**Time Complexity in Python** Now-a-days, for one problem we can write the solution in n number of ways, but, how can we decide which type is better. We can use different types of algorithms to solve one problem. We need to compare these algorithms and have to choose the best one to solve the problem.

**What is Time Complexity?**

The amount of time it takes to run the program and perform the functions in it is known as **Time Complexity**. By using Time Complexity we can determine whether the program is efficient or we have to use another algorithm which take less time compared to the other one. Reducing Time Complexity of an algorithm is often difficult in Data Science, rather than difficult we can say its a bigger challenge.

We will tell the time complexity of a program by calculating the time taken to run the algorithm in the **worst-case** scenario.

When analyzing the time complexity of an algorithm we may find three cases: **best-case, average-case and worst-case**.

**Example:** Suppose we have the following unsorted list [1, 5, 3, 9, 2, 4, 6, 7, 8] and we need to find the index of a value in this list using linear search.

**best-case:** this is the complexity of solving the problem for the best input. In our example, the best case would be to search for the value 1. Since this is the first value of the list, it would be found in the first iteration.

**average-case:** this is the average complexity of solving the problem. This complexity is defined with respect to the distribution of the values in the input data. Maybe this is not the best example but, based on our sample, we could say that the average-case would be when we’re searching for some value in the “middle” of the list, for example, the value 2.

**worst-case**: this is the complexity of solving the problem for the worst input of size n. In our example, the worst-case would be to search for the value 8, which is the last element from the list.

To quantify the Time Complexity, **Big-O** notation is used.

**5.8.1. Big-O Notation**

**Big-O notation**, sometimes called “asymptotic notation”, **is a mathematical notation that describes the limiting behavior of a function** when the argument tends towards a particular value or infinity.

Big-O notation is used to classify algorithms according to how their run time or space requirements grow as the input size grows. The letter O is used because the growth rate of a function is also referred to as the **order of the function**or **order of the program**. We will always refer order of the function in its worst-case.

A list of some common asymptotic notations is mentioned below,

|  |  |
| --- | --- |
| **Complexity Class** | **Name** |
| O(1) | constant |
| O(logn) | logarithmic |
| O(n) | linear |
| Ο(n log n) | Linear logarithmic |
| Ο(n2) | quadratic |
| Ο(n3) | cubic |
| O(2n) | exponential |

**Example 1: Linear Search- O(n)**

* A **linear search** is the most basic kind of search that is performed. A linear or sequential search, is done when you inspect each item in a list one by one from one end to the other to find a match for what you are searching for.
* Let’s see the example I stated above once again to understand the Linear Search
* We have a list which consists of integers and we have to check whether the number given by the user is present in that list or not.

l = [1,2,3,6,4,9,10,12]

k = 12

The simple code for this is

l = [1,2,3,6,4,9,10,12]

k = 12

for i in range(0, len(l)):

if l[i] == k:

print("Yes")

break

* Here, the worst-case for this algorithm is to check the number which is present in the last element of the given list. So, if we go by the above program, first it’ll start with index 0 and check whether that element in the list is equal to k or not, i.e, one operation and we have to check for every element in the list for worst-case scenario.
* The Time Complexity of the above program is **O(n)**.

**Example 2: O(n2)**

The complexity of an algorithm is said to be quadratic when the steps required to execute an algorithm are a quadratic function of the number of items in the input. Quadratic complexity is denoted as O(n^2). Take a look at the following example to see a function with quadratic complexity:

def quadratic\_algo(items):

for item in items:

for item2 in items:

print(item, ' ' ,item)

quadratic\_algo([4, 5, 6, 8])

In this example, we have an outer loop that iterates through all the items in the input list and then a nested inner loop, which again iterates through all the items in the input list. The total number of steps performed is n \* n, where n is the number of items in the input array.

**Example 3: Merge Sort - O(n\*logn)**.

def mergeSort(alist):

print("Splitting ",alist)

if len(alist)>1:

mid = len(alist)//2

lefthalf = alist[:mid]

righthalf = alist[mid:]

mergeSort(lefthalf)

mergeSort(righthalf)

i=0

j=0

k=0

while i < len(lefthalf) and j < len(righthalf):

if lefthalf[i] <= righthalf[j]:

alist[k]=lefthalf[i]

i=i+1

else:

alist[k]=righthalf[j]

j=j+1

k=k+1

while i < len(lefthalf):

alist[k]=lefthalf[i]

i=i+1

k=k+1

while j < len(righthalf):

alist[k]=righthalf[j]

j=j+1

k=k+1

alist = input('Enter the list of numbers: ').split()

alist = [int(x)for x in alist]

mergeSort(alist)

print('Sorted list: ', end='')

print(alist)

**Output:**

Enter the list of numbers: 56 48 10 2 40

Sorted list: [2, 10, 40, 48, 56]

The Time Complexity for the above program is **O(n\*logn)**.

**Example 4: Insertion sort- O(n2)**

def insertionSort(nlist):

for index in range(1,len(nlist)):

currentvalue = nlist[index]

position = index

while position>0 and nlist[position-1]>currentvalue:

nlist[position]=nlist[position-1]

position = position-1

nlist[position]=currentvalue

nlist = input('Enter the list of numbers: ').split()

nlist = [int(x)for x in nlist]

insertionSort(nlist)

print('Sorted list: ', end='')

print(nlist)

Output:

Enter the list of numbers: 4 5 6 3 1

Sorted list: [1, 3, 4, 5, 6]

The time complexity of insertion sort is O(n2)

**5.9. Simple Case Studies based on Python**

**Binary Search**

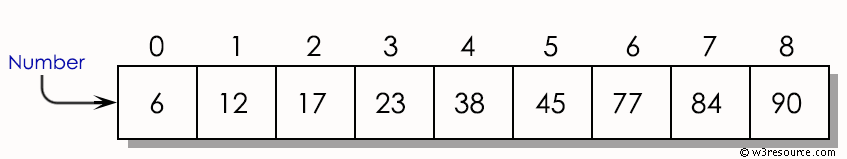
* Binary search is a searching algorithm that works efficiently with a sorted list.
* If a list is already sorted, then the search for an element in the list can be made faster by using ‘divide and conquer’ technique.
* The list is divided into two halves separated by the middle element.
* The binary search follows the following steps:

**Step 1:** The middle element is tested for the required element. If found, then its position is reported else the following test is made.

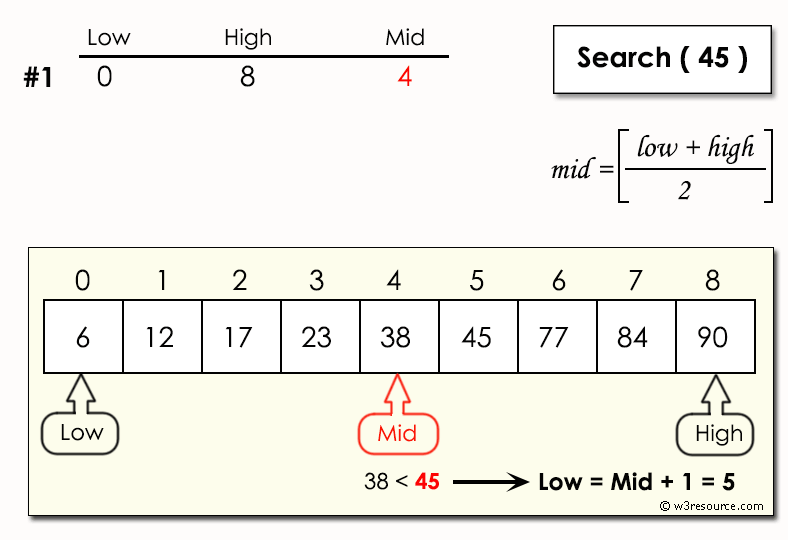
**Step 2:** If search element ‘val’< ‘middle’ element, search the left half of the list, else search the right half of the list.

**Step 3:** Repeat step 1 and 2 on the selected half until the entry is found otherwise report failure.

This search is called binary because in each iteration, the given list is divided into two parts. Then the search becomes limited to half the size of the list to be searched.



**Step 1:**







**# Iterative Binary Search Function method Python Implementation**

**# It returns index of n in given list1 if present,**

**# else returns -1**

def binary\_search(list1, n):

low = 0

high = len(list1) - 1

mid = 0

while low <= high:

# for get integer result

mid = (high + low) // 2

# Check if n is present at mid

if list1[mid] < n:

low = mid + 1

# If n is greater, compare to the right of mid

elif list1[mid] > n:

high = mid - 1

# If n is smaller, compared to the left of mid

else:

return mid

# element was not present in the list, return -1

return -1

# Initial list1

list1 = input('Enter the list of numbers: ').split()

list1 = [int(x)for x in list1]

n=int(input(‘enter the search element’))

# Function call

result = binary\_search(list1, n)

if result != -1:

print("Element is present at index", str(result))

else:

print("Element is not present in list1")

**Output:**

Enter the list of numbers: 10 3 2 13 5 6

enter the search element 2

2

Element is present at index 2

**Binary Search Complexity**

**Time Complexities**

* **Best case complexity**: O(1)
* **Average case complexity**: O(log n)
* **Worst case complexity**: O(log n)

**Space Complexity**

The space complexity of the binary search is O(1).

**Binary Search Applications**

* In libraries of Java, .Net, C++ STL
* While debugging, the binary search is used to pinpoint the place where the error happens.

**5.10. Common elements in list**

Given two lists, print all the common elements of two lists.

Input : list1 = [1, 2, 3, 4, 5]

list2 = [5, 6, 7, 8, 9]

Output : {5}

Explanation: The common elements of

both the lists are 3 and 4

Input : list1 = [1, 2, 3, 4, 5]

list2 = [6, 7, 8, 9]

Output : No common elements

Explanation: They do not have any

elements in common in between them

### **Method 1:Using Set’s & property**

Convert the lists to sets and then print **set1&set2**. set1&set2 returns the common elements set, where set1 is the list1 and set2 is the list2.   
Below is the Python3 implementation of the above approach:

**# Python program to find the common elements**

**# in two lists**

def common\_member(a, b):

    a\_set = set(a)

    b\_set = set(b)

    if (a\_set & b\_set):

        print(a\_set & b\_set)

    else:

        print("No common elements")

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

b = [5, 6, 7, 8, 9]

common\_member(a, b)

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

b = [6, 7, 8, 9]

common\_member(a, b)

**Output:**

{5}

No common elements

### **Method 2:Using Set’s intersection property**

Convert the list to set by conversion. Use the [intersection](https://www.geeksforgeeks.org/intersection-function-python/) function to check if both sets have any elements in common. If they have many elements in common, then print the intersection of both sets.   
Below is the Python3 implementation of the above approach: 

|  |
| --- |
|  |
| # Python program to find common elements in  # both sets using intersection function in  # sets  # function  def common\_member(a, b):      a\_set = set(a)      b\_set = set(b)        # check length      if len(a\_set.intersection(b\_set)) > 0:          return(a\_set.intersection(b\_set))      else:          return("no common elements")  a = [1, 2, 3, 4, 5]  b = [5, 6, 7, 8, 9]  print(common\_member(a, b))    a =[1, 2, 3, 4, 5]  b =[6, 7, 8, 9]  print(common\_member(a, b)) |

**Output:**

{5}

No common elements

**5.11. Hash Table**

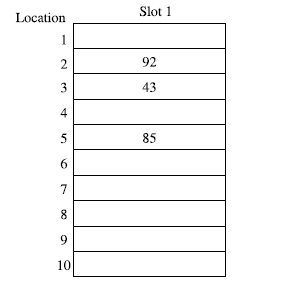
The Hash table data structure stores elements in key-value pairs where

* **Key**- unique integer that is used for indexing the values
* **Value** - data that are associated with keys.



**Figure: Key and Value in Hash table**

* A hash table is a form of list where elements are accessed by a keyword rather than an index number.



**Figure: An ideal hash table**

In Python, the Dictionary data types represent the implementation of hash tables. The Keys in the dictionary satisfy the following requirements.

* The keys of the dictionary are hashable i.e. the are generated by hashing function which generates unique result for each unique value supplied to the hash function.
* The order of data elements in a dictionary is not fixed.

**5.12. DICTIONARY**

* Python dictionary is an unordered collection of items. Each item of a dictionary has a key/value pair.
* Dictionaries are optimized to retrieve values when the key is known.

**Creating Python Dictionary**

* Creating a dictionary is as simple as placing items inside curly braces {} separated by commas.
* An item has a key and a corresponding value that is expressed as a pair (key: value).
* While the values can be of any data type and can repeat, keys must be of immutable type ([string](https://www.programiz.com/python-programming/string), [number](https://www.programiz.com/python-programming/numbers) or [tuple](https://www.programiz.com/python-programming/tuple) with immutable elements) and must be unique.

# empty dictionary

my\_dict = {}

# dictionary with integer keys

my\_dict = {1: 'apple', 2: 'ball'}

# dictionary with mixed keys

my\_dict = {'name': 'John', 1: [2, 4, 3]}

# using dict()

my\_dict = dict({1:'apple', 2:'ball'})

# from sequence having each item as a pair

my\_dict = dict([(1,'apple'), (2,'ball')])

As you can see from above, we can also create a dictionary using the built-in dict()function.

**5.12.1. Accessing Elements from Dictionary**

* While indexing is used with other data types to access values, a dictionary uses keys. Keys can be used either inside square brackets [] or with the get() method.
* If we use the square brackets [], KeyError is raised in case a key is not found in the dictionary. On the other hand, the get() method returns None if the key is not found.

# get vs [] for retrieving elements

my\_dict = {'name': 'Jack', 'age': 26}

# Output: Jack

print(my\_dict['name'])

# Output: 26

print(my\_dict.get('age'))

# Trying to access keys which doesn't exist throws error

# Output None

print(my\_dict.get('address'))

# KeyError

print(my\_dict['address'])

**Output**

Jack

26

None

Traceback (most recent call last):

File "<string>", line 15, in <module>

print(my\_dict['address'])

KeyError: 'address'

**5.12.2. Changing and Adding Dictionary elements**

* Dictionaries are mutable. We can add new items or change the value of existing items using an assignment operator.
* If the key is already present, then the existing value gets updated. In case the key is not present, a new (**key: value**) pair is added to the dictionary.

# Changing and adding Dictionary Elements

my\_dict = {'name': 'Jack', 'age': 26}

# update value

my\_dict['age'] = 27

#Output: {'age': 27, 'name': 'Jack'}

print(my\_dict)

# add item

my\_dict['address'] = 'Downtown'

# Output: {'address': 'Downtown', 'age': 27, 'name': 'Jack'}

print(my\_dict)

**Output**

{'name': 'Jack', 'age': 27}

{'name': 'Jack', 'age': 27, 'address': 'Downtown'}

**5.12.3. Removing elements from Dictionary**

* We can remove a particular item in a dictionary by using the pop() method. This method removes an item with the provided key and returns the value.
* The popitem() method can be used to remove and return an arbitrary (key, value)item pair from the dictionary. All the items can be removed at once, using the clear() method.
* We can also use the del keyword to remove individual items or the entire dictionary itself.

# Removing elements from a dictionary

# create a dictionary

squares = {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

# remove a particular item, returns its value

# Output: 16

print(squares.pop(4))

# Output: {1: 1, 2: 4, 3: 9, 5: 25}

print(squares)

# remove an arbitrary item, return (key,value)

# Output: (5, 25)

print(squares.popitem())

# Output: {1: 1, 2: 4, 3: 9}

print(squares)

# remove all items

squares.clear()

# Output: {}

print(squares)

# delete the dictionary itself

del squares

# Throws Error

print(squares)

**Output**

16

{1: 1, 2: 4, 3: 9, 5: 25}

(5, 25)

{1: 1, 2: 4, 3: 9}

{}

Traceback (most recent call last):

File "<string>", line 30, in <module>

print(squares)

NameError: name 'squares' is not defined

**5.12.4. Python Dictionary Methods**

Methods that are available with a dictionary are tabulated below. Some of them have already been used in the above examples.

|  |  |
| --- | --- |
| **Method** | **Description** |
| [clear()](https://www.programiz.com/python-programming/methods/dictionary/clear) | Removes all items from the dictionary. |
| [copy()](https://www.programiz.com/python-programming/methods/dictionary/copy) | Returns a shallow copy of the dictionary. |
| [fromkeys(seq[, v])](https://www.programiz.com/python-programming/methods/dictionary/fromkeys) | Returns a new dictionary with keys from seq and value equal to v (defaults to None). |
| [get(key[,d])](https://www.programiz.com/python-programming/methods/dictionary/get) | Returns the value of the key. If the key does not exist, returns d (defaults to None). |
| [items()](https://www.programiz.com/python-programming/methods/dictionary/items) | Return a new object of the dictionary's items in (key, value) format. |
| [keys()](https://www.programiz.com/python-programming/methods/dictionary/keys) | Returns a new object of the dictionary's keys. |
| [pop(key[,d])](https://www.programiz.com/python-programming/methods/dictionary/pop) | Removes the item with the key and returns its value or d if key is not found. If d is not provided and the key is not found, it raises KeyError. |
| [popitem()](https://www.programiz.com/python-programming/methods/dictionary/popitem) | Removes and returns an arbitrary item (key, value). Raises KeyError if the dictionary is empty. |
| [setdefault(key[,d])](https://www.programiz.com/python-programming/methods/dictionary/setdefault) | Returns the corresponding value if the key is in the dictionary. If not, inserts the key with a value of d and returns d (defaults to None). |
| [update([other])](https://www.programiz.com/python-programming/methods/dictionary/update) | Updates the dictionary with the key/value pairs from other, overwriting existing keys. |
| [values()](https://www.programiz.com/python-programming/methods/dictionary/values) | Returns a new object of the dictionary's values |

Here are a few example use cases of these methods.

# Dictionary Methods

marks = {}.fromkeys(['Math', 'English', 'Science'], 0)

# Output: {'English': 0, 'Math': 0, 'Science': 0}

print(marks)

for item in marks.items():

print(item)

# Output: ['English', 'Math', 'Science']

print(list(sorted(marks.keys())))

**Output**

{'Math': 0, 'English': 0, 'Science': 0}

('Math', 0)

('English', 0)

('Science', 0)

['English', 'Math', 'Science']

**5.12.5. Python Dictionary Comprehension**

* Dictionary comprehension is an elegant and concise way to create a new dictionary from an iterable in Python.
* Dictionary comprehension consists of an expression pair (**key: value**) followed by a for statement inside curly braces {}.
* Here is an example to make a dictionary with each item being a pair of a number and its square.

# Dictionary Comprehension

squares = {x: x\*x for x in range(6)}

print(squares)

**Output**

{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

This code is equivalent to

squares = {}

for x in range(6):

squares[x] = x\*x

print(squares)

**Output**

{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

A dictionary comprehension can optionally contain more [for](https://www.programiz.com/python-programming/for-loop) or [if](https://www.programiz.com/python-programming/if-elif-else) statements.

An optional if statement can filter out items to form the new dictionary.

Here are some examples to make a dictionary with only odd items.

# Dictionary Comprehension with if conditional

odd\_squares = {x: x\*x for x in range(11) if x % 2 == 1}

print(odd\_squares)

**Output**

{1: 1, 3: 9, 5: 25, 7: 49, 9: 81}

To learn more dictionary comprehensions, visit [Python Dictionary Comprehension](https://www.programiz.com/python-programming/dictionary-comprehension).

**5.12.6. Other Dictionary Operations**

**Dictionary Membership Test**

We can test if a key is in a dictionary or not using the keyword in. Notice that the membership test is only for the keys and not for the values.

# Membership Test for Dictionary Keys

squares = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}

# Output: True

print(1 in squares)

# Output: True

print(2 not in squares)

# membership tests for key only not value

# Output: False

print(49 in squares)

Output

True

True

False

**Iterating Through a Dictionary**

We can iterate through each key in a dictionary using a for loop.

# Iterating through a Dictionary

squares = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}

for i in squares:

print(squares[i])

**Output**

1

9

25

49

81

**Dictionary Built-in Functions**

Built-in functions like all(), any(), len(), cmp(), sorted(), etc. are commonly used with dictionaries to perform different tasks.

|  |  |
| --- | --- |
| Function | Description |
| [all()](https://www.programiz.com/python-programming/methods/built-in/all) | Return True if all keys of the dictionary are True (or if the dictionary is empty). |
| [any()](https://www.programiz.com/python-programming/methods/built-in/any) | Return True if any key of the dictionary is true. If the dictionary is empty, return False. |
| [len()](https://www.programiz.com/python-programming/methods/built-in/len) | Return the length (the number of items) in the dictionary. |
| cmp() | Compares items of two dictionaries. (Not available in Python 3) |
| [sorted()](https://www.programiz.com/python-programming/methods/built-in/sorted) | Return a new sorted list of keys in the dictionary. |

Here are some examples that use built-in functions to work with a dictionary.

# Dictionary Built-in Functions

squares = {0: 0, 1: 1, 3: 9, 5: 25, 7: 49, 9: 81}

# Output: False

print(all(squares))

# Output: True

print(any(squares))

# Output: 6

print(len(squares))

# Output: [0, 1, 3, 5, 7, 9]

print(sorted(squares))

**Output**

False

True

6

[0, 1, 3, 5, 7, 9]

|  |  |  |  |
| --- | --- | --- | --- |
| **Part-A** | | | |
| **Q.No** | **Questions** | **Competence** | **BT Level** |
| 1. | Define Data structure | Remember | BTL 1 |
| 2. | List the types of built-in data structures | Understand | BTL 2 |
| 3. | Define function and write the syntax of function? | Remember | BTL 1 |
| 4. | Define function call | Remember | BTL 1 |
| 5. | Write a python program to find even or odd using function | Apply | BTL 3 |
| 6. | Define pandas | Remember | BTL 1 |
| 7. | Interpret Dataframe | Understand | BTL 2 |
| 8. | Elaborate numpy | Understand | BTL 2 |
| 9. | How to load data in pandas? | Understand | BTL 2 |
| 10. | Write about the libraries used for pre-processing and array operations? | Understand | BTL 2 |
| 11. | List various plots using matplotlib | Understand | BTL 2 |
| 12. | Define matplotlib and How to import matplotlib | Understand | BTL 2 |
| 13. | Interpret dictionary | Understand | BTL 2 |
| 14. | State the use of hash table | Understand | BTL 2 |
| 15. | Define computational complexity | Understand | BTL 2 |
| 16. | Elaborate time complexity | Understand | BTL 2 |
| 17. | Write the time complexity of Linear search with an example program. | Analysis | BTL 4 |
| 18. | State the operations using Numpy | Understand | BTL 2 |
| 19. | Define series | Remember | BTL 1 |
| 20. | How to remove an element from dictionary? | Analysis | BTL 4 |
| **PART B** | | | |
| **Q.No** | **Questions** | **Competence** | **BT Level** |
| 1. | Explain about the python function with an example | Analysis | BTL 4 |
| 2. | Explain the various operations performed using numpy | Analysis | BTL 4 |
| 3. | Explain about Dataframe in pandas | Analysis | BTL 4 |
| 4. | Explain about NumPy – Array Manipulation | Analysis | BTL 4 |
| 5. | Write a python code to create 1D and 2D array using numpy | Analysis | BTL 4 |
| 6. | Explain about the operations performed using Numpy- Linear Algebra with an example program | Analysis | BTL 4 |
| 7. | Explain about the basic plots in Matplotlib | Analysis | BTL 4 |
| 8. | Explain about python pandas data structure with an example | Analysis | BTL 4 |
| 9. | Discuss about the different problems based on computational complexity | Analysis | BTL 4 |
| 10. | Write the python program to find the binary search | Apply | BTL 3 |
| 11. | Write the python program to find common elements in python list using two different methods | Apply | BTL 3 |
| 12. | Write a python code to add and remove elements from dictionary? | Apply | BTL 3 |