**21CSS101J - PROGRAMMING FOR PROBLEM SOLVING**

**UNIT IV**

**INTRODUCTION TO PYTHON**

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

**Python is Interpreted** − Python is processed at runtime by the interpreter. You do not need

to compile your program before executing it. This is similar to PERL and PHP.

**Python is Interactive** − You can actually sit at a Python prompt and interact with the

interpreter directly to write your programs.

**Python is Object-Oriented** − Python supports Object-Oriented style or technique of

programming that encapsulates code within objects.

**Python is a Beginner's Language** − Python is a great language for the beginner-level

programmers and supports the development of a wide range of applications from simple

text processing to WWW browsers to games.

**Uses of Python:**

* Python can be used on a server to create web applications.
* Python can be used alongside software to create workflows.
* Python can connect to database systems. It can also read and modify files.
* Python can be used to handle big data and perform complex mathematics.
* Python can be used for rapid prototyping, or for production-ready software development.

Python's features include −

**Easy-to-learn** − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.

**Easy-to-read** − Python code is more clearly defined and visible to the eyes.

**Easy-to-maintain** − Python's source code is fairly easy-to-maintain.

**A broad standard library** − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.

**Interactive Mode** − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.

**Portable** − Python can run on a wide variety of hardware platforms and has the same interface on all platforms.

**Extendable** − You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.

**Databases** − Python provides interfaces to all major commercial databases.

**GUI Programming** − Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

**Scalable** − Python provides a better structure and support for large programs than shell scripting.

**INTRODUCTION TO GOOGLE COLAB**

Colab, or "Colaboratory", allows you to write and execute Python in your browser, with

* Zero configuration required
* Access to GPUs free of charge
* Easy sharing

Google Colab is a document that allows you to write, run, and share Python code within your browser. It is a version of the popular Jupyter Notebook within the Google suite of tools. Jupyter Notebooks (and therefore Google Colab) allow you to create a document containing executable code along with text, images, HTML, LaTeX, etc. which is then stored in your google drive and shareable to peers and colleagues for editing, commenting, and viewing.

**Starting a document**

In order to create a new document, access Google Colab at colab.research.google.com. Once here, you can begin a new document, or “notebook,” in one of two ways. Upon visiting the site, a box with your recently visited Colab documents will appear. Select “New Notebook” to begin a new document. Alternatively, to create a new document from any screen, select “File” in the top left corner, then select “New Notebook” from the dropdown box.

**Basic Functions**

After you have created a new notebook, you will see an empty code cell. Python code can be entered into these code cells and executed at any time by either clicking the Play button to the left of the code cell or by pressing

Command/Ctrl+Enter. On your keyboard..

At the top of your notebook, you will find two buttons:

+Code and +Text

Add a new code cell by clicking the “+ Code” button in the top left corner of the document. Add a text cell by clicking the “+ Text” button in the top left corner of the document.

When a cell is selected, a toolbar will appear in the top right corner of the cell. This toolbar contains functions specific to that cell. Options include moving the cell up and down, adding comments, and deleting the cell.

Sharing a Colab notebook As with other Google Apps, Colab Notebooks can be shared. Look for the “share” button in the top right-hand corner of the window. Google Colab documents can also be shared in Google Drive, just you do with other types of documents.

**BASIC DATATYPES**

Python Data Types are used to define the type of a variable. It defines what type of data we are going to store in a variable. The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters.

Python has various built-in data types which we will discuss with in this tutorial:

* Numeric - int, float, complex
* String - str
* Sequence - list, tuple, range
* Binary - bytes, bytearray, memoryview
* Mapping - dict
* Boolean - bool
* Set - set, frozenset
* None - NoneType

**Integers (int)**: Used to store whole numbers without decimal points.

X=10

Y=-5

**Floating-Point Numbers (float)**: Used to store decimal numbers.

**Pi= 3**.14159

**Gravity**  = 9.81

**Strings (str)**: Used to store sequences of characters, enclosed in single ('') or double ("") quotes.

**Name**  = “Alice”

**Message = ‘ Hello, World !’**

**Booleans (bool)**: Used to represent truth values, either **True** or **False**.

**Is\_active = true**

**Is\_registered = False**

**Lists (list)**: Used to store ordered collections of items, which can be of different data types.

**Fruits = [“apple”,”Banana”, “Orange”]**

**Numbers= [1,2,3,4,5]**

**Tuples (tuple)**: Similar to lists, but immutable (cannot be modified after creation).

**Coordinates = (3,5)**

**Colors = (“red”,”green”, “blue”)**

**Dictionaries (dict)**: Used to store key-value pairs, where each key maps to a value.

**Person = {“name”** : “Bob” , “age” : 30 , “city” = “Newyork”}

**Scores = { “math”** : 90 , “science” : 85 , “history” : 80}

**Sets (set)**: Used to store unique items without any specific order.

**Unique\_numbers = { 1,2,3,4,5}**

**NoneType (None)**: Used to represent the absence of a value or a null value.

**Result = None**

**WORKING WITH STRING FUNCTIONS**

Strings are amongst the most popular types in Python. We can create them simply by enclosing characters in quotes. Python treats single quotes the same as double quotes. Creating strings is as simple as assigning a value to a variable. For example −

var1 = 'Hello World!'

var2 = "Python Programming"

Accessing Values in Strings

Python does not support a character type; these are treated as strings of length one, thus also considered a substring.

To access substrings, use the square brackets for slicing along with the index or indices to obtain your substring. For example −

var1 = 'Hello World!'

var2 = "Python Programming"

print "var1[0]: ", var1[0]

print "var2[1:5]: ", var2[1:5]

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

var1[0]: H

var2[1:5]: ytho

Updating Strings

You can "update" an existing string by (re)assigning a variable to another string. The new value can be related to its previous value or to a completely different string altogether. For example −

var1 = 'Hello World!'

print "Updated String :- ", var1[:6] + 'Python'

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

Updated String :- Hello Python

**WORKING WITH INPUT/OUTPUT FUNCTIONS**

The simplest way to produce output is using the *print* statement where you can pass zero or more expressions separated by commas. This function converts the expressions you pass into a string and writes the result to standard output as follows –

print "Python is really a great language,", "isn't it?"

This produces the following result on your standard screen –

Python is really a great language, isn't it?

Reading Keyboard Input

Python provides two built-in functions to read a line of text from standard input, which by default comes from the keyboard. These functions are −

* raw\_input
* input

## The *raw\_input* Function

The *raw\_input([prompt])* function reads one line from standard input and returns it as a string (removing the trailing newline).

#!/usr/bin/python

str = raw\_input("Enter your input: ")

print "Received input is : ", str

This prompts you to enter any string and it would display same string on the screen. When I typed "Hello Python!", its output is like this −

Enter your input: Hello Python

Received input is : Hello Python

## The *input* Function

The *input([prompt])* function is equivalent to raw\_input, except that it assumes the input is a valid Python expression and returns the evaluated result to you.

#!/usr/bin/python

str = input("Enter your input: ")

print "Received input is : ", str

This would produce the following result against the entered input −

Enter your input: [x\*5 for x in range(2,10,2)]

Recieved input is : [10, 20, 30, 40]

## Opening and Closing Files

Until now, you have been reading and writing to the standard input and output. Now, we will see how to use actual data files.

Python provides basic functions and methods necessary to manipulate files by default. You can do most of the file manipulation using a **file** object.

## The *open* Function

Before you can read or write a file, you have to open it using Python's built-in *open()* function. This function creates a **file** object, which would be utilized to call other support methods associated with it.

### **Syntax**

file object = open(file\_name [, access\_mode][, buffering])

Here are parameter details −

* **file\_name** − The file\_name argument is a string value that contains the name of the file that you want to access.
* **access\_mode** − The access\_mode determines the mode in which the file has to be opened, i.e., read, write, append, etc. A complete list of possible values is given below in the table. This is optional parameter and the default file access mode is read (r).
* **buffering** − If the buffering value is set to 0, no buffering takes place. If the buffering value is 1, line buffering is performed while accessing a file. If you specify the buffering value as an integer greater than 1, then buffering action is performed with the indicated buffer size. If negative, the buffer size is the system default(default behavior).

## The *close()* Method

The close() method of a *file* object flushes any unwritten information and closes the file object, after which no more writing can be done.

Python automatically closes a file when the reference object of a file is reassigned to another file. It is a good practice to use the close() method to close a file.

fileObject.close()

# Open a file

fo = open("foo.txt", "wb")

print "Name of the file: ", fo.name

# Close opend file

fo.close()

This produces the following result −

Name of the file: foo.txt

## The *write()* Method

The *write()* method writes any string to an open file. It is important to note that Python strings can have binary data and not just text.

The write() method does not add a newline character ('\n') to the end of the string −

### **Syntax**

fileObject.write(string)

# Open a file

fo = open("foo.txt", "wb")

fo.write( "Python is a great language.\nYeah its great!!\n")

# Close opend file

fo.close()

The above method would create *foo.txt* file and would write given content in that file and finally it would close that file. If you would open this file, it would have following content.

Python is a great language.

Yeah its great!!

## The *read()* Method

The *read()* method reads a string from an open file. It is important to note that Python strings can have binary data. apart from text data.

fileObject.read([count])

# Open a file

fo = open("foo.txt", "r+")

str = fo.read(10);

print "Read String is : ", str

# Close opend file

fo.close()

This produces the following result −

Read String is : Python is

## The rename() Method

The *rename()* method takes two arguments, the current filename and the new filename.

os.rename(current\_file\_name, new\_file\_name)

import os

# Rename a file from test1.txt to test2.txt

os.rename( "test1.txt", "test2.txt" )

## The *remove()* Method

You can use the *remove()* method to delete files by supplying the name of the file to be deleted as the argument.

os.remove(file\_name)

#!/usr/bin/python

import os

# Delete file test2.txt

os.remove("text2.txt")

**PYTHON SINGLE/MULTILINE COMMENTS**

Python comments are programmer-readable explanation or annotations in the Python source code. They are added with the purpose of making the source code easier for humans to understand, and are ignored by Python interpreter. Comments enhance the readability of the code and help the programmers to understand the code very carefully.

Just like most modern languages, Python supports single-line (or end-of-line) and multi-line (block) comments. Python comments are very much similar to the comments available in PHP, BASH and Perl Programming languages.

There are three types of comments available in Python

* Single line Comments
* Multiline Comments
* Docstring Comments

## Single Line Comments

A hash sign (#) that is not inside a string literal begins a comment. All characters after the # and up to the end of the physical line are part of the comment and the Python interpreter ignores them.

print ("Hello, World!")

You can type a comment on the same line after a statement or expression –

name = "Madisetti" # This is again comment

## Multi-Line Comments

'''

This is a multiline

comment.

'''

## Docstring Comments

Python docstrings provide a convenient way to provide a help documentation with Python modules, functions, classes, and methods. The **docstring** is then made available via the \_\_doc\_\_ attribute.

def add(a, b):

"""Function to add the value of a and b"""

return a+b

print(add.\_\_doc\_\_)

**ERROR HANDLING IN PYTHON**

An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions. In general, when a Python script encounters a situation that it cannot cope with, it raises an exception. An exception is a Python object that represents an error.

When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits.

## Handling an exception

If you have some *suspicious* code that may raise an exception, you can defend your program by placing the suspicious code in a **try:** block. After the try: block, include an **except:** statement, followed by a block of code which handles the problem as elegantly as possible.

try:

You do your operations here;

......................

except *ExceptionI*:

If there is ExceptionI, then execute this block.

except *ExceptionII*:

If there is ExceptionII, then execute this block.

......................

else:

If there is no exception then execute this block

Here are few important points about the above-mentioned syntax −

* A single try statement can have multiple except statements. This is useful when the try block contains statements that may throw different types of exceptions.
* You can also provide a generic except clause, which handles any exception.
* After the except clause(s), you can include an else-clause. The code in the else-block executes if the code in the try: block does not raise an exception.
* The else-block is a good place for code that does not need the try: block's protection.

try:

fh = open("testfile", "w")

fh.write("This is my test file for exception handling!!")

except IOError:

print "Error: can\'t find file or read data"

else:

print "Written content in the file successfully"

fh.close()

This produces the following result −

Written content in the file successfully

## The *except* Clause with No Exceptions

try:

You do your operations here;

......................

except:

If there is any exception, then execute this block.

......................

else:

If there is no exception then execute this block.

## This kind of a try-except statement catches all the exceptions that occur. Using this kind of try-except statement is not considered a good programming practice though, because it catches all exceptions but does not make the programmer identify the root cause of the problem that may occur.

## The try-finally Clause

You can use a **finally:** block along with a **try:** block. The finally block is a place to put any code that must execute, whether the try-block raised an exception or not. The syntax of the try-finally statement is this −

try:

You do your operations here;

......................

Due to any exception, this may be skipped.

finally:

This would always be executed.

......................

## CONDITIONAL AND LOOPING STATEMENTS

Decision making is anticipation of conditions occurring while execution of the program and specifying actions taken according to the conditions.

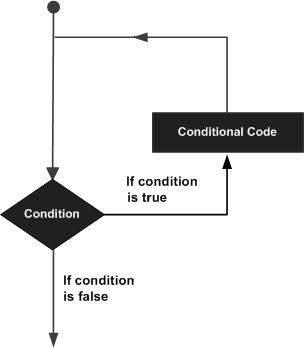
Decision structures evaluate multiple expressions which produce TRUE or FALSE as outcome. You need to determine which action to take and which statements to execute if outcome is TRUE or FALSE otherwise.

Following is the general form of a typical decision making structure found in most of the programming languages −

## Decision making statements in Python

## Python programming language assumes any non-zero and non-null values as TRUE, and if it is either zero or null, then it is assumed as FALSE value.

A loop statement allows us to execute a statement or group of statements multiple times. The following diagram illustrates a loop statement –



| **Sr.No.** | **Loop Type & Description** |
| --- | --- |
| 1 | [while loop](https://www.tutorialspoint.com/python/python_while_loop.htm)  Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body. |
| 2 | [for loop](https://www.tutorialspoint.com/python/python_for_loop.htm)  Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable. |
| 3 | [nested loops](https://www.tutorialspoint.com/python/python_nested_loops.htm)  You can use one or more loop inside any another while, for or do..while loop. |

## Loop Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

Python supports the following control statements. Click the following links to check their detail.

Let us go through the loop control statements briefly

| **Sr.No.** | **Control Statement & Description** |
| --- | --- |
| 1 | [break statement](https://www.tutorialspoint.com/python/python_break_statement.htm)  Terminates the loop statement and transfers execution to the statement immediately following the loop. |
| 2 | [continue statement](https://www.tutorialspoint.com/python/python_continue_statement.htm)  Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating. |
| 3 | [pass statement](https://www.tutorialspoint.com/python/python_pass_statement.htm)  The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute. |

**WORKING WITH LIST DATASTRUCTURES**

The list is a most versatile datatype available in Python which can be written as a list of comma-separated values (items) between square brackets. Important thing about a list is that items in a list need not be of the same type.

Creating a list is as simple as putting different comma-separated values between square brackets. For example −

list1 = ['physics', 'chemistry', 1997, 2000];

list2 = [1, 2, 3, 4, 5 ];

list3 = ["a", "b", "c", "d"]

Similar to string indices, list indices start at 0, and lists can be sliced, concatenated and so on.

**Accessing Values in Lists**

To access values in lists, use the square brackets for slicing along with the index or indices to obtain value available at that index. For example −

list1 = ['physics', 'chemistry', 1997, 2000];

list2 = [1, 2, 3, 4, 5, 6, 7 ];

print "list1[0]: ", list1[0]

print "list2[1:5]: ", list2[1:5]

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

list1[0]: physics

list2[1:5]: [2, 3, 4, 5]

## Updating Lists

You can update single or multiple elements of lists by giving the slice on the left-hand side of the assignment operator, and you can add to elements in a list with the append() method. For example

list = ['physics', 'chemistry', 1997, 2000];

print "Value available at index 2 : "

print list[2]

list[2] = 2001;

print "New value available at index 2 : "

print list[2]

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

Value available at index 2 :

1997

New value available at index 2 :

2001

**Delete List Elements**

To remove a list element, you can use either the del statement if you know exactly which element(s) you are deleting or the remove() method if you do not know. For example −

list1 = ['physics', 'chemistry', 1997, 2000];

print list1

del list1[2];

print "After deleting value at index 2 : "

print list1

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

['physics', 'chemistry', 1997, 2000]

After deleting value at index 2 :

['physics', 'chemistry', 2000]

**Basic List Operations**

Lists respond to the + and \* operators much like strings; they mean concatenation and repetition here too, except that the result is a new list, not a string.

In fact, lists respond to all of the general sequence operations we used on strings in the prior chapter.

| **Python Expression** | **Results** | **Description** |
| --- | --- | --- |
| len([1, 2, 3]) | 3 | Length |
| [1, 2, 3] + [4, 5, 6] | [1, 2, 3, 4, 5, 6] | Concatenation |
| ['Hi!'] \* 4 | ['Hi!', 'Hi!', 'Hi!', 'Hi!'] | Repetition |
| 3 in [1, 2, 3] | True | Membership |
| for x in [1, 2, 3]: print x, | 1 2 3 | Iteration |

**Indexing, Slicing, and Matrixes**

Because lists are sequences, indexing and slicing work the same way for lists as they do for strings.

Assuming following input −

L = ['spam', 'Spam', 'SPAM!']

| **Python Expression** | **Results** | **Description** |
| --- | --- | --- |
| L[2] | SPAM! | Offsets start at zero |
| L[-2] | Spam | Negative: count from the right |
| L[1:] | ['Spam', 'SPAM!'] | Slicing fetches sections |

**Built-in List Functions & Methods**

Python includes the following list functions −

| **Sr.No.** | **Function with Description** |
| --- | --- |
| 1 | [cmp(list1, list2)](https://www.tutorialspoint.com/python/list_cmp.htm)  Compares elements of both lists. |
| 2 | [len(list)](https://www.tutorialspoint.com/python/list_len.htm)  Gives the total length of the list. |
| 3 | [max(list)](https://www.tutorialspoint.com/python/list_max.htm)  Returns item from the list with max value. |
| 4 | [min(list)](https://www.tutorialspoint.com/python/list_min.htm)  Returns item from the list with min value. |
| 5 | [list(seq)](https://www.tutorialspoint.com/python/list_list.htm)  Converts a tuple into list. |

**WORKING WITH TUPLES DATASTRUCTURES**

A tuple is a collection of objects which ordered and immutable. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

Creating a tuple is as simple as putting different comma-separated values. Optionally you can put these comma-separated values between parentheses also. For example −

tup1 = ('physics', 'chemistry', 1997, 2000);

tup2 = (1, 2, 3, 4, 5 );

tup3 = "a", "b", "c", "d";

The empty tuple is written as two parentheses containing nothing −

tup1 = ();

To write a tuple containing a single value you have to include a comma, even though there is only one value −

tup1 = (50,);

**Accessing Values in Tuples**

To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index. For example −

tup1 = ('physics', 'chemistry', 1997, 2000);

tup2 = (1, 2, 3, 4, 5, 6, 7 );

print "tup1[0]: ", tup1[0];

print "tup2[1:5]: ", tup2[1:5];

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

tup1[0]: physics

tup2[1:5]: [2, 3, 4, 5]

**Updating Tuples**

Tuples are immutable which means you cannot update or change the values of tuple elements. You are able to take portions of existing tuples to create new tuples as the following example demonstrates −

tup1 = (12, 34.56);

tup2 = ('abc', 'xyz');

# Following action is not valid for tuples

# tup1[0] = 100;

# So let's create a new tuple as follows

tup3 = tup1 + tup2;

print tup3;

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

(12, 34.56, 'abc', 'xyz')

**Delete Tuple Elements**

Removing individual tuple elements is not possible. There is, of course, nothing wrong with putting together another tuple with the undesired elements discarded.

To explicitly remove an entire tuple, just use the **del** statement. For example −

tup = ('physics', 'chemistry', 1997, 2000);

print tup;

del tup;

print "After deleting tup : ";

print tup;

This produces the following result. Note an exception raised, this is because after **del tup** tuple does not exist any more −

('physics', 'chemistry', 1997, 2000)

After deleting tup :

Traceback (most recent call last):

File "test.py", line 9, in <module>

print tup;

NameError: name 'tup' is not defined

**Basic Tuples Operations**

Tuples respond to the + and \* operators much like strings; they mean concatenation and repetition here too, except that the result is a new tuple, not a string.

In fact, tuples respond to all of the general sequence operations we used on strings in the prior chapter −

| **Python Expression** | **Results** | **Description** |
| --- | --- | --- |
| len((1, 2, 3)) | 3 | Length |
| (1, 2, 3) + (4, 5, 6) | (1, 2, 3, 4, 5, 6) | Concatenation |
| ('Hi!',) \* 4 | ('Hi!', 'Hi!', 'Hi!', 'Hi!') | Repetition |
| 3 in (1, 2, 3) | True | Membership |
| for x in (1, 2, 3): print x, | 1 2 3 | Iteration |

**Indexing, Slicing, and Matrixes**

Because tuples are sequences, indexing and slicing work the same way for tuples as they do for strings. Assuming following input −

L = ('spam', 'Spam', 'SPAM!')

| **Python Expression** | **Results** | **Description** |
| --- | --- | --- |
| L[2] | 'SPAM!' | Offsets start at zero |
| L[-2] | 'Spam' | Negative: count from the right |
| L[1:] | ['Spam', 'SPAM!'] | Slicing fetches sections |

**Built-in Tuple Functions**

Python includes the following tuple functions −

| **Sr.No.** | **Function with Description** |
| --- | --- |
| 1 | [cmp(tuple1, tuple2)](https://www.tutorialspoint.com/python/tuple_cmp.htm)  Compares elements of both tuples. |
| 2 | [len(tuple)](https://www.tutorialspoint.com/python/tuple_len.htm)  Gives the total length of the tuple. |
| 3 | [max(tuple)](https://www.tutorialspoint.com/python/tuple_max.htm)  Returns item from the tuple with max value. |
| 4 | [min(tuple)](https://www.tutorialspoint.com/python/tuple_min.htm)  Returns item from the tuple with min value. |
| 5 | [tuple(seq)](https://www.tutorialspoint.com/python/tuple_tuple.htm)  Converts a list into tuple. |

**WORKING WITH DICTIONARIES**

Each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces. An empty dictionary without any items is written with just two curly braces, like this: {}.

Keys are unique within a dictionary while values may not be. The values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or tuples.

Accessing Values in Dictionary

To access dictionary elements, you can use the familiar square brackets along with the key to obtain its value. Following is a simple example −

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

print "dict['Name']: ", dict['Name']

print "dict['Age']: ", dict['Age']

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

dict['Name']: Zara

dict['Age']: 7

**Updating Dictionary**

You can update a dictionary by adding a new entry or a key-value pair, modifying an existing entry, or deleting an existing entry as shown below in the simple example −

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

dict['Age'] = 8; # update existing entry

dict['School'] = "DPS School"; # Add new entry

print "dict['Age']: ", dict['Age']

print "dict['School']: ", dict['School']

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

dict['Age']: 8

dict['School']: DPS School

**Delete Dictionary Elements**

You can either remove individual dictionary elements or clear the entire contents of a dictionary. You can also delete entire dictionary in a single operation.

To explicitly remove an entire dictionary, just use the **del** statement. Following is a simple example −

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

del dict['Name']; # remove entry with key 'Name'

dict.clear(); # remove all entries in dict

del dict ; # delete entire dictionary

print "dict['Age']: ", dict['Age']

print "dict['School']: ", dict['School']

**Built-in Dictionary Functions & Methods**

Python includes the following dictionary functions −

| **Sr.No.** | **Function with Description** |
| --- | --- |
| 1 | [cmp(dict1, dict2)](https://www.tutorialspoint.com/python/dictionary_cmp.htm)  Compares elements of both dict. |
| 2 | [len(dict)](https://www.tutorialspoint.com/python/dictionary_len.htm)  Gives the total length of the dictionary. This would be equal to the number of items in the dictionary. |
| 3 | [str(dict)](https://www.tutorialspoint.com/python/dictionary_str.htm)  Produces a printable string representation of a dictionary |
| 4 | [type(variable)](https://www.tutorialspoint.com/python/dictionary_type.htm)  Returns the type of the passed variable. If passed variable is dictionary, then it would return a dictionary type. |

**INTRODUCTION TO PYTHON LIBRARIES**

Each of Python's open-source libraries has its own source code. A collection of code scripts that can be used iteratively to save time is known as a library. It is like a physical library in that it has resources that can be used again, as the name suggests.

A collection of modules that are linked together is also known as a Python library. It has code bundles that can be used again and again in different programs. For programmers, it makes Python programming easier and simpler. Since then, we will not need to compose the same code for various projects. Python libraries are heavily used in a variety of fields, including data visualization, machine learning, and computer science.

As previously stated, a Python library is nothing more than a collection of code scripts or modules of code that can be used in a program for specific operations. We use libraries to avoid having to rewrite existing program code. However, the process is as follows: In the MS Windows environment, the library files have a DLL (Dynamic Load Libraries) extension. The linker automatically looks for a library when we run our program and import it. It interprets the program in accordance with the functions extracted from the library. This is how we use library strategies in our program. Later, we'll go over the process of incorporating libraries into Python programs in greater depth.

## Standard Libraries of Python

The Python Standard Library contains all of Python's syntax, semantics, and tokens. It has built-in modules that allow the user to access I/O and a few other essential modules as well as fundamental functions. The Python libraries have been written in the C language generally. The Python standard library has more than 200 core modules. Because of all of these factors, Python is a powerful programming language.

### **Matplotlib**

This library is responsible for the plotting of numerical data. It is utilized in data analysis for this reason. An open-source library plots superior quality figures, for example, pie outlines, scatterplots, boxplots, and diagrams, in addition to other things.

### **NumPy**

One of the most popular open-source Python packages, NumPy focuses on scientific and mathematical computation. It makes it easy to work with large matrices and multidimensional data thanks to built-in mathematical functions that make it easy to compute. It can be used as an N-dimensional container for all kinds of data, including linear algebra. An N-dimensional array with rows and columns is defined by the NumPy Array Python object. It can also be used as a random number generator because of this.

NumPy is preferred over lists in Python because it uses less memory, is faster, and is easier to use.

Pictures, sound waves, and other parallel crude streams can be addressed as a multi-faceted exhibit of genuine qualities involving the NumPy interface for perception. NumPy is required for full-stack developers to use this machine learning library.

### **Pandas**

Pandas is an open-source library authorized under the Berkeley Programming Conveyance (BSD). This well-known library is frequently utilized in the field of data science. They're generally utilized for examination, control, and cleaning of information, in addition to other things. Without having to switch to another programming language like R, Pandas enables us to carry out straightforward data modelling and analysis.

### **SciPy**

Scipy is a Python library. Scientific computing, information processing, and high-level computing are the primary uses for this open-source library. The library contains a large number of easy-to-use methods and functions for quick and easy computation. Scipy can be utilized for numerical calculations close by NumPy.

Some of SciPy's subpackages include cluster, fftpack, constants, integrate, io, linalg, interpolate, ndimage, odr, optimize, signal, spatial, special, sparse, and stats.

### **Scikit- learn**

Additionally, Scikit-learn is a Python-based open-source machine learning library. This library supports both supervised and unsupervised learning methods. This library already comes pre-installed with a number of well-known algorithms as well as the SciPy, NumPy, and Matplotlib packages. Spotify music recommendations are the Scikit-most-learn application that is most widely used.

### **Seaborn**

This package makes statistical model visualization possible. The library, which is largely based on Matplotlib, makes statistical graphics possible by:

1. Variable examination by means of a Programming interface in view of datasets.
2. Make complex representations effortlessly, including multi-plot frameworks.
3. To compare data subsets, univariate and bivariate visualizations are utilized.
4. A wide range of color schemes are available for pattern displays.
5. Direct relapse assessment and plotting are done consequently.

### **TensorFlow**

TensorFlow is a fast, open-source library for numerical calculations. It is utilized by ML and deep learning algorithms as well. It was developed by researchers in the Google Brain group of the Google AI organization and is now widely used by researchers in physics, mathematics, and machine learning for complex mathematical computations.

### **Keras**

Keras is a Python-based open-source neural network library that enables in-depth research into deep neural networks. Keras emerges as a viable option as deep learning becomes more common because, according to its developers, it is an API (Application Programming Interface) designed for humans rather than machines. Keras has a higher rate of adoption in the research community and industry than TensorFlow or Theano. The TensorFlow backend engine should be downloaded first before Keras can be installed.

### **Scrapy**

Scrapy is a web scratching device that scratches numerous pages in less than a moment. Scrapy is additionally an open-source Python library structure for extricating information from sites. It is a high-speed, high-level scraping and crawling web library that goes by the name "Scrapinghub ltd."

### **PyGame**

The Standard Directmedia Library (SDL)'s graphics, audio, and input libraries are accessible through a straightforward interface that can be used on any platform by this library. With the Python programming language, computer graphics, and acoustic libraries, it is used to create video games.

### **PyBrain**

When compared to the other Python learning libraries, PyBrain is a quick and straightforward machine learning library. From the various Python libraries that are available, PyBrain is also an open-source library for ML algorithms that is suitable for any beginning researcher.

### **Statsmodels**

Statsmodels is a Python library for statistical model estimation and analysis. The results of statistical tests and other tasks carried out in the library are of high quality.

The interface's ease of use The Python programming language is utilized extensively in numerous actual-world applications. Because it is a dynamically written high-level language, its use in error debugging is rapidly growing. Python is becoming more and more prevalent in well-known programs like YouTube and DropBox. Clients can likewise play out numerous assignments without expecting to type their code, because of the openness of Python libraries.

**HIGH DIMENSIONAL ARRAYS**

An array with multiple dimensions can represent relational tables and matrices and is made up of many one-dimensional arrays, multi-dimensional arrays are frequently used to store data for mathematical computations, image processing, and maintaining records.

To understand and implement multi-dimensional arrays in Python, the [NumPy](https://numpy.org/doc/stable/user/index.html#user) package is used. It is a Python library that gives users access to a multidimensional array object, a variety of derived objects (such as masked arrays and matrices), and a selection of functions for quick operations on arrays and multi-dimensional matrices.

The standard way of Python language creates lists which are very similar to arrays but remember, there is a very slight difference between a List and [an array in Python](https://www.askpython.com/python-modules/numpy/python-numpy-arrays) programming language. You can learn more about the differences between [lists vs arrays in python.](https://www.askpython.com/python/difference-between-python-list-vs-array) In this article let’s look purely at arrays. The following example will show the difference between the datatype of creating a 2D array created by the standard way and by using the Numpy package.

#creating 2D array without any package

arr1 **=** [[0]**\***3]**\***2

print(arr1)

print(type(arr1))

#creating 2D array with numpy package

print("\n")

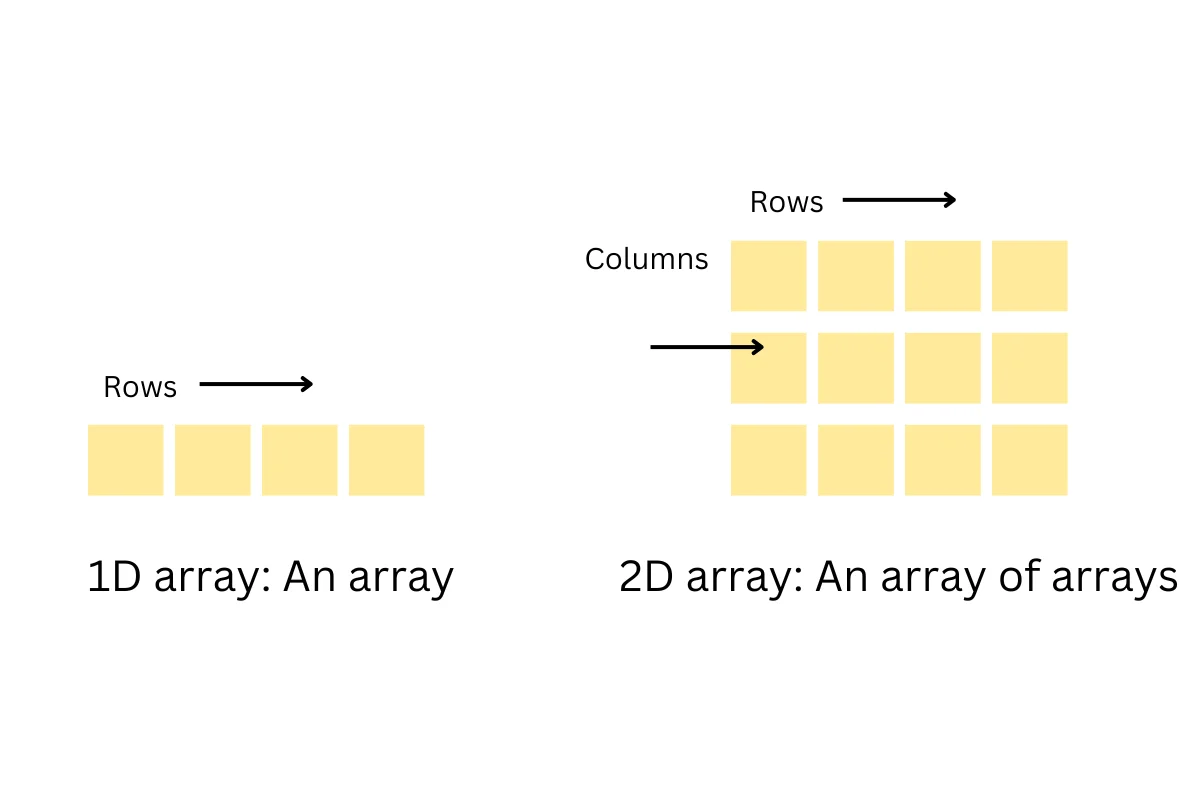
arr2 **=** np.array([[1,2,3,4], [5,6,7,8]])

print(arr2)

print(type(arr2))

## Two-dimensional (2D) array

An array of arrays is a simple definition that applies to the two-dimensional array. The rows and columns of the matrices that make up the 2D array’s organizational structure can be viewed as its primary component. The 2D array can be visualized as a table (a square or rectangle) with rows and columns of elements. The image below depicts the structure of the two-dimensional array.

****

### **Implementing 2D array in Python**

Let’s start with implementing a 2 dimensional array using the numpy array method.

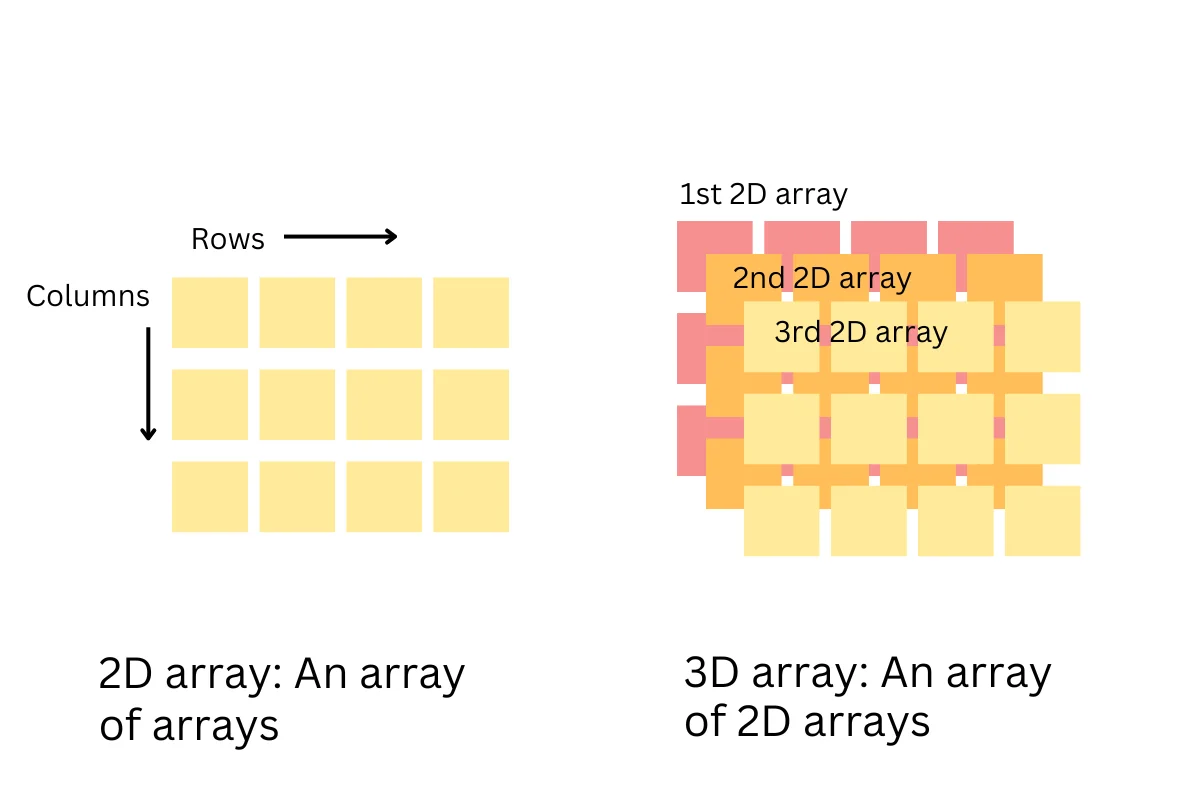
| arr **=** np.array([array1 values..][array2 values...]) |
| --- |

* arr: array’s name
* np.array: function of the Numpy package

| array\_1 **=** np.array([[1,2,3,4],                      [5,6,7,8]])    print("Output")  print(array\_1) |
| --- |

## Three-dimensional (3D) array in Python

A 3-D (three-dimensional) array is mainly composed of an array of 2-D arrays. The rows, columns, and page elements can be viewed as primary components of a 3D array. We can visualize it as multiple tables comprising of rows and columns attached (like a cube). The image below depicts the structure of the three-dimensional array.

****

| arr **=** numpy.array([2D\_array1 values],[2D\_array2 values]...) |
| --- |

* arr: array’s name
* np.array: function of the Numpy package

| array\_2 **=** np.array([[[1,2,3],[3,4,5]],                     [[6,7,8],[9,8,7]],                     [[6,5,4],[3,2,1]]])    print("Output")  print(array\_2) |
| --- |