# Database Connection

There are the following steps to connect a python application to our database.

* Import mysql.connector module
* Create the connection object.
* Create the cursor object
* Execute the query

1. **import** mysql.connector
2. myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "root", database = "mydb") #Create the connection object
3. **print**(myconn) #printing the connection object
4. cur = myconn.cursor() #creating the cursor object
5. **print**(cur)

## Adding a record to the table

In python, we can mention the format specifier (%s) in place of values.

We provide the actual values in the form of tuple in the execute() method of the cursor.

1. **import** mysql.connector
2. myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "google",database = "PythonDB")
3. cur = myconn.cursor()
4. sql = "insert into Employee(name, id, salary, dept\_id, branch\_name) values (%s, %s, %s, %s, %s)"
5. #The row values are provided in the form of tuple
6. val = ("John", 110, 25000.00, 201, "Newyork")
7. **try**:
8. cur.execute(sql,val) #inserting the values into the table
9. myconn.commit() #commit the transaction
10. **except**:
11. myconn.rollback()
12. **print**(cur.rowcount,"record inserted!")
13. myconn.close()

## Insert multiple rows

We can also insert multiple rows at once using the python script. The multiple rows are mentioned as the list of various tuples.

Each element of the list is treated as one particular row, whereas each element of the tuple is treated as one particular column value (attribute).

## Row ID

In SQL, a particular row is represented by an insertion id which is known as row id. We can get the last inserted row id by using the attribute lastrowid of the cursor object.

1. **import** mysql.connector
2. myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "google",database = "PythonDB")
3. cur = myconn.cursor() #creating the cursor object
4. sql = "insert into Employee(name, id, salary, dept\_id, branch\_name) values (%s, %s, %s, %s, %s)"
5. val = [("John", 102, 25000.00, 201, "Newyork"),("David",103,25000.00,202,"Port of spain"),("Nick",104,90000.00,201,"Newyork")]
6. **try**:
7. cur.executemany(sql,val) #inserting the values into the table
8. myconn.commit() #commit the transaction
9. **print**(cur.rowcount,"records inserted!")
10. **print**(cur.rowcount,"record inserted! id:",cur.lastrowid) //it will return last row id
11. **except**:
12. myconn.rollback()
13. myconn.close()

# Read Operation

Python provides the fetchall() method returns the data stored inside the table in the form of rows. We can iterate the result to get the individual rows.

## The fetchone() method

The fetchone() method is used to fetch only one row from the table. The fetchone() method returns the next row of the result-set.

1. **import** mysql.connector
2. myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "google",database = "PythonDB")
3. cur = myconn.cursor()
4. **try**:
5. cur.execute("select \* from Employee")
6. result = cur.fetchall()
7. result = cur.fetchone() #fetching the first row from the cursor object
8. **for** x **in** result:
9. **print**(x);
10. **except**:
11. myconn.rollback()
12. myconn.close()

## Formatting the result

We can format the result by iterating over the result produced by the fetchall() or fetchone() method of cursor object since the result exists as the tuple object which is not readable.

1. **import** mysql.connector
2. myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "google",database = "PythonDB")
3. cur = myconn.cursor()
4. **try**:
5. cur.execute("select name, id, salary from Employee")
6. result = cur.fetchall() #fetching the rows from the cursor object
7. **print**("Name id Salary");
8. **for** row **in** result:
9. **print**("%s %d %d"%(row[0],row[1],row[2]))
10. **except**:
11. myconn.rollback()
12. myconn.close()

# Performing Transactions

Transactions ensure the data consistency of the database. We have to make sure that more than one applications must not modify the records while performing the database operations. The transactions have the following properties.

1. **Atomicity**Either the transaction completes, or nothing happens. If a transaction contains 4 queries then all these queries must be executed, or none of them must be executed.
2. **Consistency**The database must be consistent before the transaction starts and the database must also be consistent after the transaction is completed.
3. **Isolation**Intermediate results of a transaction are not visible outside the current transaction.
4. **Durability**Once a transaction was committed, the effects are persistent, even after a system failure.

## Python commit() method

Python provides the commit() method which ensures the changes made to

the database consistently take place.

All the operations that modify the records of the database do not take place until the commit() is called.

## Python rollback() method

The rollback() method is used to revert the changes that are done to the database. This method is useful in the sense that, if some error occurs during the database operations, we can rollback that transaction to maintain the database consistency.

**Conn.rollback()**

# How to take input in Python?

These methods are given below.

* input(prompt)
* raw\_input(prompt) //used in older version

# How to Convert Python List to String

Iteration, comprehension, join(), and map() are all methods for converting a list into a string.

1. a\_list = ["Python", "Convert", "List", "String", "Method"]
2. string = " ".join( a\_list ) # it is read as join elements of a\_list with a separator (" ")

**Note**: all list item should be string type

### Using join() Method and map() Method

Using the combination of the map() and join() methods in Python gives a method for converting lists into strings. Unlike the join() method, we can use this method if the list has elements of int data type.

We did use the map() method to convert the integer elements into a string before converting the whole list to a string because the join() method can only accept string elements.

**map(function, iterable)**

1. iterable = ["Python", "Convert", 11, "List", 12, "String", "Method"]
2. string = " ".join (map (str, iterable))
3. # Converting to string using list comprehension
4. string = " ".join ([str( elements ) **for** elements **in** iterable])

# How to append element in the list

* **append(elmt) -** It appends the value at the end of the list.
* **insert(index, elmt) -** It inserts the value at the specified index position.if index is out of range, will add at end.
* **extends(iterable) -** It extends the list by adding the iterable object.

# How to compare two lists in Python

* **The cmp() function:**The [Python](https://www.javatpoint.com/python-tutorial) cmp() function compares the two Python objects & returns the integer values -1, 0, 1 according to the comparison.(not used in 3.x)
* **set() function and == operator:(**convert list to set the compare with == operator)
* **The sort() function and == operator:**Python **sort()** function is used to sort the lists. same list's elements are the same index position it means; lists are equal.
* **The collections.Counter() function:**The collection module provides the counter(), which compare the list efficiently. It stores the data in dictionary format <value>:<frequency> and counts the frequency of the list's items.**(collections.Counter(list1) == collections.Counter(list2))**
* The reduce() and map() function

# How to convert int to string in Python

**Using the str() function:** str(n)

**Using the "%s" integer:** con\_n = "% s" % n

**Using the .format() function:** con\_n = "{}".format(n)

**Using f-string:** conv\_n = f'{n}'

# How to Create a Dictionary in Python

We can create dic using {} and dict() method.

For adding element **we can use update() method or dic[key]=value**

If the key-value combination is present in the dictionary, the value for that key is modified; if this is not the case, a new key is created and added to the Dictionary with the given value.

1. # Adding a nested Key to our dictionary
2. Dictionary[5] = {'Nested\_key' :{1 : 'Nested', 2 : 'Key'}}

# How to create a virtual environment in Python

Suppose we are developing two applications that require an older version of the library, and other applications require a new version of libraries in the same [Python installation](https://www.javatpoint.com/how-to-install-python). Sometimes, it leads to a problem.

To resolve this conflicting requirement, Python offers to create the **virtual environment**. The module **venv** is used to create and manage a virtual environment.

# How to declare a variable in Python

Python is a dynamic-typed language, which means we don't need to mention the variable type or declare before using it.

* The first character of the variable can be an alphabet or (\_) underscore.
* Special characters (@, #, %, ^, &, \*) should not be used in variable name.
* Variable names are case sensitive. Ex - age and AGE are two different variables.
* Reserve words cannot be declared as variables.

[Python](https://www.javatpoint.com/python-tutorial) supports three types of numbers - integer, floating point numbers, and complex.

# matplotlib in Python

It represents the data through the graphical form. The graphical form can be a Scatter Plot, Bar Graph, Histogram, Area Plot, Pie Plot, etc. The [matplotlib](https://www.javatpoint.com/matplotlib) library is generally used to data visualization. Data visualization allows us to make a effective decision for organization.

# How to print in same line in Python

For printing in sam line use “**end**”.

**print**(i, end = " ")

# How to read JSON file in Python

we need to import json module, and it provides the **load()** function to read the JSON file.

1. with open(r'C:\Users\DEVANSH SHARMA\student.json') as f:
2. data = json.load(f)
3. **print**(data)

In the above code, we have used the **open()** function to read the JSON file. The **load()** function is parsed the JSON file and returned the dictionary named data.

### What is a scripting language?

The scripting language is referred to perform the task based on automating a repeated task. It includes same types of steps while implementing the procedure or program. It reduces time and cuts the costs further.

### Features of Scripting Language

* It runs faster and complete task efficiently.
* It is easy to learn and easy to use.
* No IDEs are required to write code.
* It is suitable for automation tasks.
* Scripting language doesn't require the memory to run the program.
* Less line code requires completing the task as compared to other languages.

# How to concatenate two strings in Python

* Using + operators
* Using join() method //**print**("".join([str1, str2]))
* Using % method //**print**("% s % s" % (str1, str2))
* Using format() function //**print**("{} {}".format(str1, str2))

# How to convert list to dictionary in Python

### Method - 1 Using Dictionary Comprehension

1. student = ["James", "Abhinay", "Peter", "Bicky"]
2. student\_dictionary = { stu : "Passed" **for** stu **in** student } //{n: n\*n **for** n **in** list1}

### Method - 2 Using zip() function

The **zip()** function is used to zip the two values together. First, we need to create an iterator and initialize to any variable and then typecast to the **dict()** function.

res\_dct = dict(zip(list, list))

## Symmetric Difference of two sets

Symmetric difference of two sets is calculated by ^ operator or **symmetric\_difference()** method. Symmetric differ of sets, it removes that element which is present in both sets.

Generator Expression

The representation of generator expression is similar to the Python list comprehension. The only difference is that **square bracket is replaced by round parentheses**.

* **BeautifulSoup :**BeautifulSoup is a Python library that is used to pull data of HTML and XML files.It is mainly designed for web scrapping. It works with parser to provide a natural way of navigating, searching, and modifying the parse tree.

# Python SimpleImputer module

A scikit-learn class that we can use to handle the missing values in the data from the dataset of a predictive model is called SimpleImputer class. With the help of this class, we can replace NaN (missing values) values in the dataset with a specified placeholder. We can implement and use this module class by using the SimpleImputer() method in the program.

# How to clear Python shell

1. Import os
2. #Type
3. os.system('CLS')

# How to create a DataFrames in Python

A Data Frame is a two-dimension collection of data. It is a data structure where data is stored in tabular form. Datasets are arranged in rows and columns; we can store multiple datasets in the data frame. We can perform various arithmetic operations, such as adding column/row selection and columns/rows in the data frame.

We can import the DataFrames from the external storage; these storages can be referred to as the [SQL](https://www.javatpoint.com/sql-tutorial) Database, CSV file, and an Excel file. We can also use the lists, dictionary, and from a list of dictionary, etc.

### An empty dataframe

We can create a basic empty Dataframe. The dataframe constructor needs to be called

to create the DataFrame.

1. **import** pandas as pd
2. df = pd.DataFrame() //# Calling DataFrame constructor
3. print(df)

### Method - 2: Create a dataframe using List

We can create dataframe using a single list or list of lists.

1. **import** pandas as pd
2. lst = ['Java', 'Python', 'C', 'C++', 'JavaScript', 'Swift', 'Go']
3. dframe = pd.DataFrame(lst) //# Calling DataFrame constructor on list
4. print(dframe)

### Method - 3: Create Dataframe from dict of ndarray/lists

The dict of ndarray/lists can be used to create a dataframe, all the **ndarray** must be of same length. The index will be a range(n) by default; where n denotes the array length.

1. **import** pandas as pd
2. data = {'Name': ['Tom', 'Joseph', 'Krish', 'John'], 'Age': [20, 21, 19, 18]}
3. df = pd.DataFrame(data) //# Create DataFrame
4. print(df)

### Method - 4: Create a indexes Dataframe using arrays

Let's understand the following example to create the indexes dataframe using arrays.

1. **import** pandas as pd
2. data = {'Name':['Renaul', 'Duster', 'Maruti', 'Honda City'], 'Ratings':[9.0, 8.0, 5.0, 3.0]}
3. df = pd.DataFrame(data, index =['pos1', 'pos2', 'pos3', 'pos4'])
4. print(df)

In the above code, we have defined the column name with the various car names and their ratings. We used the array to create indexes.

### Method - 5: Create Dataframe from list of dicts

We can pass the lists of dictionaries as input data to create the Pandas dataframe. The column names are taken as keys by default.

1. **import** pandas as pd
2. data = [{'A': 10, 'B': 20, 'C':30}, {'x':100, 'y': 200, 'z': 300}] //# assign values to lists.
3. df = pd.DataFrame(data) //# Creates DataFrame.
4. print(df)
5. **import** pandas as pd
6. data = [{'x': 1, 'y': 2}, {'A': 15, 'B': 17, 'C': 19}] # assigns values to lists.
7. # With two column indices, values same # as dictionary keys
8. dframe1 = pd.DataFrame(data, index =['first', 'second'], columns =['x', 'y'])
9. # With two column indices with # one index with other name
10. dframe2 = pd.DataFrame(data, index =['first', 'second'], columns =['x', 'y1'])
11. print (dframe1, "\n")
12. print (dframe2)

### Method - 6: Create Dataframe using the zip() function

The zip() function is used to merge the two lists.

1. **import** pandas as pd
2. Name = ['tom', 'krish', 'arun', 'juli']
3. Marks = [95, 63, 54, 47]
4. list\_tuples = list(zip(Name, Marks)) # and merge them by using zip().
5. print(list\_tuples)
6. # Converting lists of tuples into # pandas Dataframe.
7. dframe = pd.DataFrame(list\_tuples, columns=['Name', 'Marks'])
8. print(dframe)

### Method - 7: Create Dataframe from Dicts of series

The dictionary can be passed to create a dataframe. We can use the Dicts of series where the subsequent index is the union of all the series of passed index value.

1. **import** pandas as pd
2. d = {'Electronics' : pd.Series([97, 56, 87, 45], index =['John', 'Abhinay', 'Peter', 'And']),
3. 'Civil' : pd.Series([97, 88, 44, 96], index =['John', 'Abhinay', 'Peter', 'Andrew'])}
4. dframe = pd.DataFrame(d) # creates Dataframe.
5. print(dframe)

# How to develop a game in Python

Python provides a built-in library called [**pygame,**](https://www.javatpoint.com/pygame) which used to develop the game.

Pygame is a cross-platform library that is used to design video games.

It includes computer graphics and sound libraries to give the standard game experience to the user.

Here is the following example of creating a simple pygame window.

1. **import** pygame
2. pygame.init()
3. screen = pygame.display.set\_mode((400,500))
4. done = False
5. **while** not done:
6. **for** event in pygame.event.get():
7. **if** event.type == pygame.QUIT:
8. done = True
9. pygame.display.flip()

All graphics will draw in the pygame window.

Let's understand the basic syntax of the above program.

**import pygame -** It is the module which allows us to work with all function of pygame.

**pygame.init() -** It is used to initialize all the required modules of the pygame.

**pygame.display.set\_mode((width, height)) -** It is used to resized the window size. It will return the surface object. The surface object is used to perform graphical operations.

**pygame.event.get() -** It makes the event queue empty. If we do not call it, the window messages will start to pile up and, the game will become unresponsive in the opinion of the operating system.

**pygame.QUIT -** It is used to dismiss the event when we click on the cross button at the corner of the window.

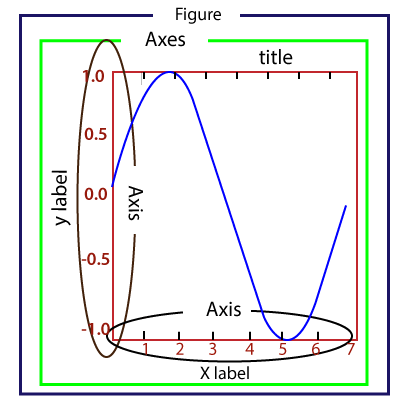
**pygame.display.flip() -** It is used to reflect any update to the game. If we make any change then we need to call the display.flip() function.

We can draw any shape to the pygame surface includes adding images, attractive font. Pygame provides many built-in functions to draw the geometrical shape to the screen. These shapes are initial stage of developing a game.

# How to plot a graph in Python

Python provides one of a most popular plotting library called **Matplotlib**. It is open-source, cross-platform for making 2D plots for from data in array. It is generally used for data visualization and represent through the various graphs.

A graph contains the following parts. Let's understand these parts.



**Figure:** It is a whole figure which may hold one or more axes (plots). We can think of a Figure as a canvas that holds plots.

**Axes:** A Figure can contain several Axes. It consists of two or three (in the case of 3D) Axis objects. Each Axes is comprised of a title, an x-label, and a y-label.

**Axis:** Axises are the number of line like objects and responsible for generating the graph limits.

**Artist:** An artist is the all which we see on the graph like Text objects, Line2D objects, and collection objects. Most Artists are tied to Axes.

1. from matplotlib **import** pyplot as plt
2. plt.plot([1,2,3],[4,5,1]) #ploting our canvas
3. plt.show() #display the graph

## Ploting Different Type of Graphs

We can plot the various graph using the pyplot module.

1. Line Graph

The line chart is used to display the information as a series of the line. It is easy to plot.

1. from matplotlib **import** pyplot as plt
2. x = [1,2,3]
3. y = [10,11,12]
4. plt.plot(x,y)
5. plt.title("Line graph")
6. plt.ylabel('Y axis')
7. plt.xlabel('X axis')
8. plt.show()

Line can be modified using the various functions. It makes the graph more attractive.

1. from matplotlib **import** pyplot as plt
2. from matplotlib **import** style
3. style.use('ggplot')
4. x = [10, 12, 13]
5. y = [8, 16, 6]
6. x2 = [8, 15, 11]
7. y2 = [6, 15, 7]
8. plt.plot(x, y, 'b', label='line one', linewidth=5)
9. plt.plot(x2, y2, 'r', label='line two', linewidth=5)
10. plt.title('Epic Info')
11. fig = plt.figure()
12. plt.ylabel('Y axis')
13. plt.xlabel('X axis')
14. plt.show()

### 2. Bar Graph

Bar graph is one of the most common graphs and it is used to represent the data associated with the categorical variables. The **bar()** function accepts three arguments - categorical variables, values, and color.

1. from matplotlib **import** pyplot as plt
2. Names = ['Arun','James','Ricky','Patrick']
3. Marks = [51,87,45,67]
4. plt.bar(Names,Marks,color = 'blue')
5. plt.title('Result')
6. plt.xlabel('Names')
7. plt.ylabel('Marks')
8. plt.show()

### 3. Pie Chart

A chart is a circular graph which is divided into the sub-part or segment. It is used to represent the percentage or proportional data where each slice of pie represents a particular category.

1. from matplotlib **import** pyplot as plt
2. # Pie chart, where the slices will be ordered and plotted counter-clockwise:
3. Aus\_Players = 'Smith', 'Finch', 'Warner', 'Lumberchane'
4. Runs = [42, 32, 18, 24]
5. explode = (0.1, 0, 0, 0) # it "explode" the 1st slice
6. fig1, ax1 = plt.subplots()
7. ax1.pie(Runs, explode=explode, labels=Aus\_Players, autopct='%1.1f%%',
8. shadow=True, startangle=90)
9. ax1.axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle.
10. plt.show()

### 4. Histogram

The histogram and bar graph is quite similar but there is a minor difference them. A histogram is used to represent the distribution, and bar chart is used to compare the different entities. A histogram is generally used to plot the frequency of a number of values compared to a set of values ranges.

1. from matplotlib **import** pyplot as plt
2. from matplotlib **import** pyplot as plt
3. percentage = [97,54,45,10, 20, 10, 30,97,50,71,40,49,40,74,95,80,65,82,70,65,55,70,75,60,52,44,43,42,45]
4. number\_of\_student = [0,10,20,30,40,50,60,70,80,90,100]
5. plt.hist(percentage, number\_of\_student, histtype='bar', rwidth=0.8)
6. plt.xlabel('percentage')
7. plt.ylabel('Number of people')
8. plt.title('Histogram')
9. plt.show()

Ex:2;

1. from matplotlib **import** pyplot as plt
2. **import** numpy as np
3. plt.style.use('fivethirtyeight')
4. mu = 50
5. sigma = 7
6. x = np.random.normal(mu, sigma, size=200)
7. fig, ax = plt.subplots()
8. ax.hist(x, 20)
9. ax.set\_title('Historgram')
10. ax.set\_xlabel('bin range')
11. ax.set\_ylabel('frequency')
12. fig.tight\_layout()
13. plt.show()

### 5. Scatter Plot

The scatter plot is used to compare the variable with respect to the other variables. It is defined as how one variable affected the other variable. The data is represented as a collection of points.

1. from matplotlib **import** pyplot as plt
2. from matplotlib **import** style
3. style.use('ggplot')
4. x = [4,8,12]
5. y = [19,11,7]
6. x2 = [7,10,12]
7. y2 = [8,18,24]
8. plt.scatter(x, y)
9. plt.scatter(x2, y2, color='g')
10. plt.title('Epic Info')
11. plt.ylabel('Y axis')
12. plt.xlabel('X axis')
13. plt.show()

# How to sort a dictionary in Python

* Sorting by keys //print(sorted(names.keys()))
* Sorting by values
* Sorting Algorithm:

1. daynames = { 'one' : 'Monday' , 'six' : 'Saturday' ,'three' : 'Wednesday' , 'two' : 'Tuesday' , 'five': 'Friday' , 'seven': 'Sunday' }
2. print(daynames)
3. number = { 'one' : 1 , 'two' : 2 , 'three' : 3 , 'four' : 4 , 'five' : 5 , 'six' : 6 , 'seven' : 7}
4. print(sorted(daynames , key=number.\_\_getitem\_\_))
5. print([daynames[i] **for** i in sorted(daynames , key=number.\_\_getitem\_\_)])

* Reversing the sorted order:
  + a = {'a':2 ,'b':1 ,'c':3 ,'d':4 ,'e':5 ,'f':6 }
  + print(sorted(a.values() , reverse= True))

# Strong Number in Python

A Strong number is a special number whose sum of the all digit factorial should be equal to the number itself.

we get 1! + 4! + 5! =1+24+120 = 145, which is exactly the same as the given number. So we can say that 145 is a strong number.

# How to Convert Text to Speech in Python

The gTTS [API](https://www.javatpoint.com/api-full-form) provides the facility to convert text files into different languages such as English, Hindi, German, Tamil, French, and many more. We can also play the audio speech in fast or slow mode.

However, as its latest update we cannot change the speech file; it will generate by the system and not changeable.

To convert text files into, we will use another offline library called **pyttsx3**.

1. # Import the gTTS module for text to speech conversion
2. **from** gtts **import** gTTS
3. # This module is imported so that we can play the converted audio
4. **from** playsound **import** playsound
5. # It is a text value that we want to convert to audio
6. text\_val = 'All the best for your exam.'
7. language = 'en' # Here are converting in English Language
8. # Pasing txt & languge to engine, here we have asign slow=False. Which denotes
9. # the module that the transformed audio should # have a high speed
10. obj = gTTS(text=text\_val, lang=language, slow=False)
11. #Here we are saving the transformed audio in a mp3 file named # exam.mp3
12. obj.save("exam.mp3")
13. playsound("exam.mp3") # Play the exam.mp3 file

In the above code, we have imported the API and use the gTTS function. The **gTTS()** function which takes three arguments -

* The first argument is a text value that we want to convert into a speech.
* The second argument is a specified language. It supports many languages. We can convert the text into the audio file.
* The third argument represents the speed of the speech. We have passed **slow** value as false; it means the speech will be at normal speed.

We saved this file as **exam.py**, which can be accessible anytime, and then we have used the **playsound()** function to listen the audio file at runtime.

### Offline API

We have used the Google API, but what if we want to convert text to speech using offline. Python provides the pyttsx3 library, which looks for TTS engines pre-installed in our platform.

1. **import** pyttsx3
2. engine = pyttsx3.init() # initialize Text-to-speech engine
3. text = "Python is a great programming language" # convert this text to speech
4. engine.say(text)
5. engine.runAndWait() # play the speech

In the above code, we have used the **say()** method and passed the text as an argument. It is used to add a word to speak to the queue, while the **runAndWait()** method runs the real event loop until all commands queued up.

It also provides some additional properties that we can use according to our needs. Let's get the details of speaking rate:

1. rate = engine.getProperty("rate") # get details of speaking rate
2. **print**(rate)

# Bubble Sort in Python

It can be easy to implement into the code, which is much beneficial for beginner software developers. But it is the worst algorithm for sorting the elements in every except because it checks every time the array is sorted or not.

## Concept of Bubble Sort

The bubble sort uses a straightforward logic that works by repeating swapping the adjacent elements if they are not in the right order. It compares one pair at a time and swaps if the first element is greater than the second element; otherwise, move further to the next pair of elements for comparison.

1. def bubble\_sort(list1): # Creating a bubble sort function
2. for i in range(0,len(list1)-1): # Outer loop for traverse the entire list
3. for j in range(len(list1)-1):
4. if(list1[j]**>**list1[j+1]):
5. temp = list1[j]
6. list1[j] = list1[j+1]
7. list1[j+1] = temp
8. return list1
9. list1 = [5, 3, 8, 6, 7, 2]
10. print("The unsorted list is: ", list1)
11. print("The sorted list is: ", bubble\_sort(list1)) # Calling the bubble sort function

# Logging in Python

1. **DEBUG -** It is used to provide detailed information and only use it when there is diagnosing problems.
2. **INFO -** It provides the information regarding that things are working as we want.
3. **WARNING -** It is used to warn that something happened unexpectedly, or we will face the problem in the upcoming time.
4. **ERROR -** It is used to inform when we are in some serious trouble, the software hasn't executed some programs.
5. **CRITICAL -** It specifies the serious error, the program itself may be incapable of remaining executing.

These corresponding numerical values of the levels are given below.

| **Level** | **Numeric Values** |
| --- | --- |
| NOTSET | 0 |
| DEBUG | 10 |
| INFO | 20 |
| WARNING | 30 |
| ERROR | 40 |
| CRITICAL | 50 |

Let's have a look at the several logger objects offered by the module itself.

* **Logger.info(msg) :** It is used to log a message with level INFO on this logger.
* **Logger.warning(msg) :** It is used to log a message with level WARNING on this logger.
* **Logger.error(msg) :** It is used to log a message with level ERROR on this logger.
* **Logger.critical(msg) :** It is used to log a message with level CRITICAL on this logger.
* **Logger.log(lvl,msg) :** It is used to logs a message with integer level lvl on this logger.
* **Logger.exception(msg) :** It is used to log a message with level ERROR on this logger.
* **Logger.setLevel(lvl) :** It is used to sets the beginning of this logger to lvl. It will ignore all the messages which are written below.
* **Logger.addFilter(filt) :** It is used to add a specific filter filt to the to this logger.
* **Logger.removeFilter(filt) :** It is used to eliminates a specific filter filt to the to this logger.
* **Logger.filter(record) :** It put on the filter of logger to the record. If the record available and to be handled then returns True. Otherwise, it will return False.
* **Logger.addHandler(hdlr) :** It is used to add a particular handler hdlr to the to this logger.
* **Logger.removeHandler(hdlr) :** It is used to eliminate a particular handler hdlr to this logger.
* **Logger.hasHandlers() :** It is used to verify if the logger contains any handler configured or not.

## Basic Configurations

The main task of logging is to store the records events in a file. The logging module provides the **basicConfig(\*\*kwarg)**, used to configure the logging.

It accepts some of the commonly used argument as follows.

* **level -** The specified severity level is set by the root level.
* **filename -** It specifies a file.
* **filemode -** It opens a file in a specific mode. The default mode of the opening file is a, which means we can append the content.
* **format -** The format defines the format of the log message.

1. import logging
2. logging.basicConfig(filename='msg.log', filemode='w', format='%(name)s - %(levelname)s - %(message)s')
3. logging.warning('This will get logged to a file')

We can also define own logger by creating an object of the **Logger** class.

1. import logging
2. logger = logging.getLogger('first\_logger')
3. logger.warning('This is a warning message')

# Insertion Sort in Python

## The Concept of Insertion Sort

The array spilled virtually in the two parts in the insertion sort - An **unsorted part** and **sorted** part.

The sorted part contains the first element of the array and other unsorted subpart contains the rest of the array. The first element in the unsorted array is compared to the sorted array so that we can place it into a proper sub-array.

It focuses on inserting the elements by moving all elements if the right-side value is smaller than the left side.

It will repeatedly happen until the all element is inserted at correct place.

To sort the array using insertion sort below is the algorithm of insertion sort.

* Spilt a list in two parts - sorted and unsorted.
* Iterate from arr[1] to arr[n] over the given array.
* Compare the current element to the next element.
* If the current element is smaller than the next element, compare to the element before, Move to the greater elements one position up to make space for the swapped element.

1. def insertion\_sort(list1): # creating a function for insertion
2. for i in range(1, len(list1)): #outer loop to traverse through 1 to len(list1)
3. value = list1[i]
4. # Move elements of list1[0..i-1], that are
5. # greater than value, to one position ahead of their current position
6. j = i - 1
7. while j **>**= 0 and value **<** **list1**[j]:
8. list1[j + 1] = list1[j]
9. j -= 1
10. list1[j + 1] = value
11. return list1
12. list1 = [10, 5, 13, 8, 2] # Driver code to test above
13. print("The unsorted list is:", list1)
14. print("The sorted list1 is:", insertion\_sort(list1))

## Time Complexity in Insertion Sort

Insertion sort is a slow algorithm; sometimes, it seems too slow for extensive dataset. However, it is efficient for small lists or array.

The time complexity of the insertion sort is - **O(n2).** It uses the two loops for iteration.

Another important advantage of the insertion sort is that; it is used by the popular sorting algorithm called **Shell sort.**

The auxiliary space in insertion sort: **O(1)**

# Binary Search in Python

There are many searching algorithms but the binary search is most popular among them.

The elements in the list must be sorted to apply the binary search algorithm. If elements are not sorted then sort them first.

## Concept of Binary Search

In the binary search algorithm, we can find the element position using the following methods.

* Recursive Method
* Iterative Method

The divide and conquer approach technique is followed by the recursive method. In this method, a function is called itself again and again until it found an element in the list.

A set of statements is repeated multiple times to find an element's index position in the iterative method. The **while** loop is used for accomplish this task.

Binary search is more effective than the linear search because we don't need to search each list index. The list must be sorted to achieve the binary search algorithm.

# Linear Search in Python

Python is one of the most popular and powerful languages. It takes a few lines to execute the code, which makes it much user-friendly language.