Module 1:

A. Tools for Coding:

Python Development Tools Development tools help us to build fast and reliable Python solutions. It includes Integrated Development Environment (IDE), Python package manager, and productive extensions. These tools have made it easy to test the software, debug, and deploy solutions in production.

Development tools

1. Jupyter Notebook

Jupyter Notebook is a web-based IDE for experimenting with code and displaying the results. It is fairly popular among data scientists and machine learning practitioners. It allows them to run and test small sets of code and view results instead of running the whole file.

The Jupyter Notebook lets us add a description and heading using markdown and export the result in the form of PDF and .ipynb files.

When you mix scientific computation with Python development, you get a Jupyter Notebook. Nowadays, teachers are using it for teaching data science courses, data analysts are using it to create reports, and machine learning engineers are using experimentation and building high-performing model architecture.

A free cloud Jupyter notebook tool is DataLab, built by DataCamp. DataLab runs fully in the browser, so you don't have to install anything else on your computer to start using it. Discover our Jupyter tutorial to learn how to use it.

It is not going anywhere in the future, people are building production-ready solutions on it, and tech giants like AWS are also incorporating it into the cloud computing ecosystems.

2. Pip

Pip is a tool that uses Python Package Index to install and manage Python software. There are 393,343 projects for you to download and install with lightning speed. The Python ecosystem works on it.

pip install <package_name>

Powered By Pip is not just an installer. You can create and manage Python environments, install dependencies, and install packages from third-party repositories using URLs. Learn more about pip by following the PIP Python Tutorial tutorial.

python -m pip install -r requirements.txt

Powered By 3. VSCode Visual Studio Code is free, lightweight, and a powerful code editor. You can build, test, deploy, and maintain all types of applications without leaving

the software window. It comes with syntax highlighting, code auto-completing, language, Git, and in-line debug support. You can use extensions to pre-build systems and deploy applications to the cloud.

VSCode is the most popular IDE in the world, and its popularity is mainly due to free extensions that improve user experience. The extensions allow data scientists to run experiments on the Jupyter notebook, edit markdown files, integrate SQL server, collaborate on projects, autocomplete code, and in-line code help. Instead of using multiple software, you can use extensions and run everything from VSCode software like bash terminal and browser.

Python Data Analysis Tools Data analysis tools allow users to ingest, clean, and manipulate data for statistical analysis. Every data professional must understand the core functionality of these tools to perform data analysis, machine learning, data engineering, and business intelligence tasks.

Data Analysis Tools

4. pandas

pandas is a gateway into the world of data science. The first thing you learn as a beginner is to load a CSV file using read_csv(). pandas is an essential tool for all data professionals.

You can load a dataset, clean it, manipulate it, calculate statistics, create visualizations, and save the data into various file formats. The pandas API is simple and intuitive. You can load and save CSV and text files, Microsoft Excel, SQL databases, and the fast HDF5 file format.

Learn more about pandas by taking our Data Manipulation with pandas course.

5. Numpy

NumPy is a fundamental Python package for scientific computations, and most modern tools are built upon it. As a data scientist, you will use the Numpy array for mathematical calculations and data wrangling. It provides multidimensional array objects to perform fast operations such as logical, shape manipulation, sorting, selection, basic statics operation, and random simulation.

Numpy will help you understand the fundamentals of mathematics in data science and how to convert complex equations into Python code. You can use it to create machine learning models, customized statical formulas, scientific simulations, and perform advanced data analytics tasks.

Learn more about NumPy by taking our Introduction to NumPy course.

6. SQLAlchemy

SQLAlchemy is a Python SQL toolkit for you to access and manage relational databases. It uses Object Relational Mapper to provide powerful features and flexibility

of SQL.

This tool is necessary for data scientists and analytics who are used to perform data processing and analytics in Python. You can either use SQL scripts to perform data analysis or use an object-based approach where you can use an intuitive Python API to perform similar tasks in effective ways.

Learn more about SQLAlchemy by taking an Introduction to Databases course on DataCamp.

7. Dask

Dask is an essential tool for processing big data or files. It uses parallel computing to perform similar tasks by libraries like NumPy, pandas, and scikit-learn.

Running a simple logical function on a large dataset of 4GB will take at least 10 minutes. Even with better machines, you cannot improve processing times to a few seconds. Dask uses dynamic task scheduling and parallel data collection to achieve fast results with the same machine.

The API is similar to pandas and scikit-learn. It is flexible, native to Python, it can scale up (1000 core) and down (single core), and provides rapid feedback and diagnostics to aid humans.

Python Data Visualization Tools Data visualization gives life to data analysis. If you want to explain things to non-technical executives, you need to tell a data story by displaying a bar chart, line plot, scatter plot, heat maps, and histograms. The visualization tools help data analytics create interactive, colorful, and clean visualization with few lines of code.

Data Visualization Tools

8. Matplotlib

Matplotlib is a gateway to the world of data visualization. You will learn about it in many data visualization introductions.

With Matplotlib, you can create fully customizable static, animated, and interactive visualizations. It's intuitive, and you can use it to plot 3D, multilevel, and detailed visualization. There are hundreds of examples of different visualizations available in the gallery.

You can learn more about Matplotlib in our Data Visualization with Matplotlib course..

9. Seaborn

Seaborn is a high-level interface based on Matplotlib for creating attractive statistical graphics. Similar to Matplotlib, you can produce interactive visualization by typing a single line of code.

It is highly adaptable and works wonders when you are new to data visualization. For customizing, you can always use matplotlib to create multiple graphs, edit axis, title, or

even colors. In some cases, seaborn will calculate everything for you and display distplot, violin plot, residplot, implot, joint plot, and boxplot.

Learn more about Seaborn by taking a Data Visualization with Seaborn course on DataCamp.

10. Plotly

When you want the features of Tableau or PowerBI, you use the Plotly Python library to display interactive and publication-quality graphs. You can zoom into a graph, isolate a single bar, filter things, and even animate it to your needs.

It comes with custom controls and allows you to animate your visualizations and work on data transformation. Plotly also contains Jupyter widgets, 3D charts, AI charts, financial charts, and scientific charts.

Plotly is the best tool to create data analytics Jupyter-based reports. Instead of creating multiple static plots, you can make one and add custom controls to explore and explain data insights.

You can discover how to utilize Plotly with our Data Visualization with Plotly course.

11. Pandas-profiling

Pandas-profiling is an AutoEDA tool for creating exploratory data analytics reports with a single line of code. The report includes column types, missing values, unique values, quantile statistics, descriptive statistics, histogram, correlation, text analysis, and file and image analysis.

It is quite a helpful tool when you have less time to explore. For example, during technical tests, preparation for team meetings, and participating in the competition.

B. Python Review:

Python Programming: A Comprehensive Review Python, a versatile, high-level programming language, has become one of the most popular languages for both beginners and seasoned developers. Known for its readability, simplicity, and expansive standard library, Python has established itself in various domains like web development, data analysis, machine learning, automation, and scientific computing.

Pros of Python Programming Readability and Simplicity: Python's syntax is clear and intuitive. Its use of indentation, as opposed to brackets or keywords, makes it more readable. This simplicity is particularly beneficial for beginners who are learning how to code. It allows them to focus more on solving problems than wrestling with complex syntax.

Extensive Libraries and Frameworks: Python has an impressive collection of libraries and frameworks. For example, libraries like NumPy, Pandas, and Matplotlib are essential for data analysis, while TensorFlow and PyTorch are widely used in machine learning.

Flask and Django simplify web development, making Python a truly multi-purpose language.

Cross-Platform Compatibility: Python is cross-platform, meaning that programs written in Python can run on multiple operating systems without modification. This flexibility is useful for developers working across different environments.

Community Support: Python has an enormous and active community. Whether you're a beginner looking for tutorials or an experienced developer facing a challenge, there's almost always an answer or a resource available. The strong community also means that Python is continuously evolving with contributions from developers worldwide.

Productivity and Flexibility: Python's dynamic typing and interpreted nature allow for rapid development and testing. It is ideal for prototype building, making Python a popular choice for startups and projects with tight deadlines.

Cons of Python Programming Speed Limitations: While Python is efficient for many tasks, it is an interpreted language, which makes it slower than compiled languages like C or C++. For performance-critical applications such as real-time systems or games, this may be a disadvantage.

Weak in Mobile and Game Development: Although Python is excellent for web development, data science, and automation, it is not the first choice for mobile or game development. There are frameworks like Kivy for mobile apps, but they aren't as mature or widely adopted as alternatives like Swift for iOS or Java/Kotlin for Android.

High Memory Usage: Python's flexible data types come at the cost of higher memory consumption. In environments where memory is a constraint, such as embedded systems, Python may not be the best choice.

Basics of Python

In [1]:
```
import sys
import keyword
import operator
from datetime import datetime
import os
```

In [3]:
```
# Keywords Keywords are the reserved words in Python and can't be used as
print(keyword.kwlist)  # List all Python Keywords
```

```
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'brea
k', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finall
y', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonloc
al', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yiel
d']
```

In [5]:
```
len(keyword.kwlist)  # Python contains 35 keywords
```

Out[5]: 35

In [7]:
```
#Identifiers
An identifier is a name given to entities like class, functions, variable
1var=10 # Identifier can't start with a digit
.
```

```
  Cell In[7], line 3
    1var=10 # Identifier can't start with a digit
    ^
SyntaxError: invalid decimal literal
```

In [9]:
```
val2@=35 # Identifier can't use special symbols
```

```
---------------------------------------------------------------------------
-
NameError                                 Traceback (most recent call las
t)
Cell In[9], line 1
----> 1 val2@=35

NameError: name 'val2' is not defined
```

In [11]:
```
import=125 # Keywords can't be used as identifiers
```

```
  Cell In[11], line 1
    import=125 # Keywords can't be used as identifiers
           ^
SyntaxError: invalid syntax
```

In [13]:
```
""" Correct way of defining an identifier
(Identifiers can be a combination of letters in lowercase (a to z) or upp
"""
val12=10
```

In [15]:
```
val_=99
```

In [1]:
```
#1st basic program
'''Python is the best programming
language for machine learning and data science'''
# for single line comment
'''for multiple line comment'''
print ("hello world")
```

```
hello world
```

Variables

In [15]:
```
a= '''I love bengaluru weather
and Brigade road is a must visit place'''
b = 2021
c = 12.15
print(a)
print (b)
print (c)
# for the boolean valuse true or false
d = True
print (d)
# to denote nothing is called none
e = None
print (e)
```

```
I love bengaluru weather
and Brigade road is a must visit place
2021
12.15
True
None
```

In [17]:
```
#printing the data type of variables used by a,b,c,d,e
# using built in function
print (type(a))
print (type(b))
print (type(c))
print (type(d))
print (type(e))
```

```
<class 'str'>
<class 'int'>
<class 'float'>
<class 'bool'>
<class 'NoneType'>
```

Rules for naming variables ''' 1. Variables cannot start with digits for ex: 222geluvaraj 2. Varibles name can contain digits, alphabets, underscores 3. A variable name can only start with alphabets or underscore Ex: geluvaraj or _geuvaraj 4. No white space is allowed to be used inside a variable name for Ex: _geluvaraj 222 5. Variables are case sensitive a=100 and A=200 are very much different'''

Operators

In [19]:
```
# Arithematic Operators

a = 600
b = 562

print("The value of 600+562 is",600+562)
print("The value of 600-562 is",600-562)
print("The value of 600/562 is",600/562)
print("The value of 600*562 is",600*562)
```

```
The value of 600+562 is 1162
The value of 600-562 is 38
The value of 600/562 is 1.0676156583629892
The value of 600*562 is 337200
```

In [21]:
```
# Assignment operators

a = 365
a += 100
b = 500
b -= 56
c = 998
c *= 98
d = 300
d /= 3
print (a)
print (b)
print (c)
print (d)
```

```
465
444
97804
100.0
```

In [23]:
```
# comparison operators
a = (100 > 300)
print (a)
b = (100 < 300)
print (b)
c = (100 >= 300)
print (c)
d = (100 <= 300)
print (d)
e = (100 != 300)
print (e)
f = (100 == 300)
print (f)
```

```
False
True
False
True
True
False
```

In [25]:
```
#Logical operators

# for AND

bool1 = True
bool2 = False
print (" the value of bool1 and bool2 is " , bool1 and bool2)
# as per boolean algebra if any of the one both must be true or false els
bool11 = False
bool12 = False
print (" the value of bool11 and bool12 is " , bool11 and bool12)
# 1 1 = 1
# 1 0 = 0
# 0 1 = 0
# 0 0 = 1
```

```
the value of bool1 and bool2 is  False
the value of bool11 and bool12 is  False
```

In [27]:
```
# for OR

bool1 = True
bool2 = False
print (" the value of bool1 or bool2 is " , bool1 or bool2)
bool21 = True
bool23 = True
print (" the value of bool21 or bool23 is " , bool21 or bool23)
bool11 = False
bool12 = True
print (" the value of bool11 or bool12 is " , bool11 or bool12)
bool41 = False
bool42 = False
print (" the value of bool41 or bool42 is " , bool41 or bool42)

#As per boolean algebra for or
# 1 0 1
# 0 1 1
# 1 1 1
# 0 0 0
```

```
the value of bool1 or bool2 is  True
the value of bool21 or bool23 is  True
the value of bool11 or bool12 is  True
the value of bool41 or bool42 is  False
```

In [29]:
```
# for NOT
# it will be use against one value which directly reverses it
bool15= False
print(" the value of bool15 is", not bool15)
bool16 = True
print(" the value of bool16 is", not bool16)
```

```
the value of bool15 is True
the value of bool16 is False
```

Typecast is a way of changing an object from one data type to the next.

It is used in computer programming to ensure a function handles the variables correctly.

In [31]:
```
a =("98562")
print(type(a))
```

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

In [33]:
```
# now lets convert this into integer
a= int(a)
```

In [35]:
```
print(type(a))
```

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

In [37]:
```
g=(96.56)
print(type(g))
```

```
<class 'float'>
```

In [39]:
```
g=int(g)
```

In [41]:
```
print(type(g))
```

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

In [43]:
```
# string is a data type in python
# string is sequence of characters enclosed in quotes
# string can be writen in three ways
# single quote
# double quote
# triple quote

# why need all these quotes

a ='king'
b= "king"
```

```
c='''king'''
print(a)
print(b)
print(c)

#because when there is a requirement of apstrophe the library gets confus

#d ='king's'
#print(d)
#invalid syntax

# so it can be represented as

e = '''king's'''
print(e)

f="king'''s"
print(f)

g= '''king's'''
print(g)

h='king"s'
print(h)
```

```
king
king
king
king's
king'''s
king's
king"s
```

In [45]:
```
greeting = 'Welome to Evening VIT batch'
batch = ' 2024 batch'
print (type(greeting))
print (greeting+batch)
c= greeting+batch
print(c)
# This is concatenation program adding of 2 strings
```

```
<class 'str'>
Welome to Evening VIT batch 2024 batch
Welome to Evening VIT batch 2024 batch
```

In [47]:
```
#indexing of the strings

name = "geluvaraj"
print(name[0])
```

```
g
```

In [49]:
```
# Examples of Arithmetic Operator
a = 9
b = 4

# Addition of numbers
add = a + b

# Subtraction of numbers
sub = a - b
```

```
# Multiplication of number
mul = a * b

# Division(float) of number
div1 = a / b

# Division(floor) of number
div2 = a // b

# Modulo of both number
mod = a % b

# Power
p = a ** b

# print results
print(add)
print(sub)
print(mul)
print(div1)
print(div2)
print(mod)
print(p)
```

```
13
5
36
2.25
2
1
6561
```

Input operations and print

In [52]:
```
# write a python program to find the remainder when a number is divided

a = 152
b =15
print(" The remainder is:", a%b)
```

```
The remainder is: 2
```

In [54]:
```
#addition of 2 numbers
a=89554
b=55462
print("the sum of a and b is ", a+b)
```

```
the sum of a and b is  145016
```

In [56]:
```
#find the average of 2 numbers

a= input("Enter a number a:")
b= input("Enter a number b:")
a= int(a)
b= int(b)
avg= (a+b)/2
print (" The avg of a and b:", avg)
```

```
The avg of a and b: 5.5
```

In [58]:
```
# Python Program to find the area of triangle

a = 5
b = 6
c = 7

# Uncomment below to take inputs from the user
# a = float(input('Enter first side: '))
# b = float(input('Enter second side: '))
# c = float(input('Enter third side: '))

# calculate the semi-perimeter
s = (a + b + c) / 2

# calculate the area
area = (s*(s-a)*(s-b)*(s-c)) ** 0.5
print('The area of the triangle is',area)
```

```
The area of the triangle is 14.696938456699069
```

Conditional statements:

In [60]:
```
x=10
if (x<40):
    print("FAIL")
```

```
FAIL
```

In [62]:
```
x=-1
if (x<0):
    print("Pass")
```

```
Pass
```

In [70]:
```
# python program to illustrate If statement

i = 5
if (i > 15):
    print("20 is less than 15")
print("I am Not in if")
```

```
I am Not in if
```

In [72]:
```
age=int(input("Enter the Person age:"))
if (age>18):
    print("You are eligible to VOTE")
else:
    print("You are not eligible to vote")
```

```
You are eligible to VOTE
```

In [74]:
```
X=int(input("Enter X:"))
if (X%2 == 0):
    print ("X is even:",X)
else:
    print ("X is odd:",X)
```

```
X is odd: 5
```

Nested if else

In [76]:
```
marks=float(input("Enter marks:"))
if marks>=60:
    if marks<70:
        print("First Class")
    else:
        print("Distinction")
```

```
Distinction
```

In [78]:
```
# User enters the year
year = int(input("Enter Year: "))

# Leap Year Check
if year % 4 == 0:  #1st codition is checked
    if year % 100 == 0: #2nd checks for century year
        if year % 400 ==0: #3rd checks completely divides by 400 or not
                print(year, "is a Leap Year")  # all conditions are true
        else:
            print(year, "is not a Leap Year")  # is not divided by 400 pr
    else:
        print(year, "is a Leap Year") # for the 1st condition
else:
    print(year, "is not a Leap Year") # for the 1st condition
```

```
2005 is not a Leap Year
```

if and elif

In [82]:
```
'''write a program to take input from users and check whether it is numbe
whether it is a lower case or uppercase'''

ch=input("Enter the character:")
if(ch>='A' and ch<='Z'):
    print("Uppercase character was entered")
elif(ch>='a' and ch<='z'):
    print("Lowercase character was entered")
elif(ch>'0=' and ch<='9'):
    print("A number was entered")
print("U DID A GREAT JOB")
```

```
Lowercase character was entered
U DID A GREAT JOB
```

In [84]:
```
'''Write a pgm to enter the marks of a student in four sbjects. then calc
display the grade obtained by the student. if the student scores an aggre
distinction. if aggregate is 60>= and <75 then the grade is first divisio
the grade is second division. if aggrgate is 40>= and <50 hen the grade i
grade is fail'''

marks1=int(input("Enter the marks1:"))
marks2=int(input("enter the marks2:"))
marks3=int(input("Enter the marks3:"))
marks4=int(input("Enter the marks4:"))
total=marks1+marks2+marks3+marks4
avg=(total)/4
if(avg>=75):
    print("Distinction")
elif(avg>=60 and avg<75):
    print("First Division")
elif(avg>=50 and avg<60):
```

```
    print("Second Division")
elif(avg>=40 and avg<50):
    print("Thrird Division")
else:
    print("Fail")
```

```
Second Division
```

While loop

In [86]:
```
#Program to print first 10 numbers using while loop

i=0
while(i<=10):
    print(i)
    i=i+1
```

```
0
1
2
3
4
5
6
7
8
9
10
```

In [88]:
```
#program to print "VIT" 5 times
i=0
while(i<=5):
    print("VIT")
    i=i+1
```

```
VIT
VIT
VIT
VIT
VIT
VIT
```

In [90]:
```
#program to separate values printed on the same line using the tab
i=0
while(i<10):
    print(i,end= '\t')
    i=i+1
```

```
0	1	2	3	4	5	6	7	8	9
```

In [92]:
```
#write a program to calculate the sum and average of first 10 numbers usi
i=0
sum=0
while(i<=10):
    sum=sum+i
    i=i+1
avg=float(sum)/10
print("The sum of the numbers is:",sum)
print("The average of first 10 numbers is:",avg)
```

```
The sum of the numbers is: 55
The average of first 10 numbers is: 5.5
```

In [96]:
```
#write a program to calculate the sum of numbers from M to N
m=int(input("Enter a number for m:"))
n=int(input("Enter a number for n:"))
sum=0
while(m<=n):
    sum=sum+m
    m=m+1
print("SUM of m and n is:",sum)
```

```
SUM of m and n is: 27
```

In [98]:
```
n= int(input("Enter Number:"))
reverse=0
while(n>0):
    digit=n%10
    reverse=reverse*10+digit
    n=n//10
print("reverse of the number:",reverse)
```

```
reverse of the number: 988
```

In [100…
```
#write a program to find the entered number is Armstrong number or not
'''Hint: An Armstrong number of 3 digits is an integer such that the sum
its digits is equal to the number itself For Ex:, 371 is armstrong number
becuause: 3*3*3+7*7*7+1*1*1=371(n power is 3 here )
one more ex: 1634 = 1*1*1*1+6*6*6*6+3*3*3*3+4*4*4*4(n power is 4 here)'''

number=int(input("Enter a postive number:"))
sum=0
original=number
while(number>0):
    remainder=number%10
    sum=sum+remainder**3
    number=number//10
if(sum==original):
    print("The entered number is Armstrong number:")
else:
    print("The entered number is not a Armstrong number")

#The pgm works 3 powers if u want the program to work for more digits kee
```

```
The entered number is Armstrong number:
```

In [108…
```
'''write a program to read the numbers until -1 is encountered. find the
enetered by the user'''

neg_count=0
neg_s=0
pos_count=0
pos_s=0
print("Enter -1 to exit")
while(1):
    num=int(input("Enter any number:"))
    if(num==-1):
        break
    if(num<0):
        neg_count=neg_count+1
        neg_s=neg_s+num
```

```
    else:
        pos_count=pos_count+1
        pos_s=pos_s+num
neg_avg = float(neg_s)/neg_count
pos_avg = float(pos_s)/pos_count
print("The average of positive numbers", pos_avg)
print("The average of negative numbers", neg_avg)
```

```
Enter -1 to exit
The average of positive numbers 7.25
The average of negative numbers -5.333333333333333
```

For loop:

In [112…
```
a=5
for i in range(a):
    print(a)
```

```
5
5
5
5
5
```

In [114…
```
a=5
for i in range(a):
    print(i)
```

```
0
1
2
3
4
```

In [124…
```
q=range(1,5)
for i in q:
    print(i)
```

```
1
2
3
4
```

In [128…
```
for i in range(1,5):
    print(i)
```

```
1
2
3
4
```

In [132…
```
a=range(10,0,-1)                    #(beg,end,step)
for i in a:
    print(i)
```

```
10
9
8
7
6
5
4
3
2
1
```

In [134…
```
#wap a pgm using for loop tocalculate the average of first n natural numb

n=int(input("Enter the value of n:"))
avg=0
sum=0
for i in range(1,n+1):
    sum=(n*(n+1)/2)
avg=sum/n
print("The sum of first",n,"natural numbers is",sum)
print("The average of first",n,"natural numbers is",avg)
```

```
The sum of first 5 natural numbers is 15.0
The average of first 5 natural numbers is 3.0
```

In [136…
```
#wap for factorial of a number
n=int(input("Enter a number:"))
result=1
for i in range(n,0,-1):
    result=result*i
print("The factorial of a number",n,"is",result)
```

```
The factorial of a number 5 is 120
```

In [138…
```
#WAP to print number entered is prime or not prime
n=int(input("Enter the number:"))
for i in range(2,n):
    if(n%i==0):
        print(n,"is a not prime number")
        break
else:
    print(n, "is prime number")
```

```
5 is prime number
```

In [140…
```
n=int(input("enter how many number you want in the fionacci series"))
first=0
second=1
for i in range(n):
    print(first)
    temp=first
    first=second
    second=temp+second
print()
```

```
0
1
1
2
3
5
8
13
21
34
55
89
144
233
377
```

In [142…
```python
# Asking the user input for printing stars horizontal

n=int(input("Enter the number of stars:"))
for i in range(1,n+1):
    print('*',end=" ")
```

```
* * * * *
```

In [144…
```python
# Asking the user input for printing stars vertical
n=int(input("Enter the number of stars:"))
for i in range(1,n+1):
    print('*')
```

```
*
*
*
*
*
```

In [146…
```python
# WAp to print the stars 5 separates rows and 5 columns
# pgm logic is correct, to much of repetative tasks, let us use the power
for i in range(5):
    print('#',end=' ')
print()
for j in range(5):
    print('#', end=' ')
print()
for k in range(5):
    print('#',end=' ')
print()
for l in range(5):
    print('#', end=' ')
print()
for m in range(5):
    print('#', end=' ')
```

```
# # # # #
# # # # #
# # # # #
# # # # #
# # # # #
```

In [148…
```python
for i in range(2):
    print('outerloop',i)
```

```
    for j in range(3):
        print('Innerloop',j)
print("Rest in Peace looping")
```

```
outerloop 0
Innerloop 0
Innerloop 1
Innerloop 2
outerloop 1
Innerloop 0
Innerloop 1
Innerloop 2
Rest in Peace looping
```

In [150…
```
for i in range(2):
    for j in range(3):
        print('Innerloop',j)
    print('outerloop',i)
print("Rest in Peace looping")
```

```
Innerloop 0
Innerloop 1
Innerloop 2
outerloop 0
Innerloop 0
Innerloop 1
Innerloop 2
outerloop 1
Rest in Peace looping
```

In [152…
```
for i in range(5):
    for j in range(i+1):
        print('*',end='')
    print()
```

```
*
**
***
****
*****
```

In [154…
```
for i in range(5):
    for k in range(i+1):
        print(' ',end='')
    for j in range(5-i):
        print('#',end='')
    print()
```

```
#####
 ####
  ###
   ##
    #
```

In [156…
```
n=5
for i in range(n):
    for j in range(i,n):
        print(' ',end='')
    for k in range(i+1):
        print('2',end='')
    print()
```

```
    2
   22
  222
 2222
22222
```

In [158…
```
n=5
for i in range(n):
    for j in range(i):
        print(' ',end='')
    for k in range(n-i):
        print('*',end='')
    for l in range(n-i-1):
        print('*',end='')
    print()
```

```
*********
 *******
  *****
   ***
    *
```

Break, continue and pass

In [160…
```
#wap for vending machine for candies

Candies=int(input("Enter the number of candies:"))
i=1
while(i<=Candies):
    print("Candies")
    i=i+1
```

```
Candies
Candies
Candies
Candies
Candies
```

In [162…
```
#what if the candies available in the machine are only 3

available=3
Candies=int(input("Enter the number of candies:"))
i=1
while(i<=Candies):
    if(i>available):
        print("out of stock")
        break
    print("Candies")
    i=i+1
print("babye")
```

```
Candies
Candies
Candies
out of stock
babye
```

In [164…
```
'''WAP to print the numbers from 1 to 100 and remove which are divisble b
for i in range(1,100):
    if(i%3==0 or i%5==0):
        continue
```

```
    print(i)
print("Happy Corona days")
```

```
1
2
4
7
8
11
13
14
16
17
19
22
23
26
28
29
31
32
34
37
38
41
43
44
46
47
49
52
53
56
58
59
61
62
64
67
68
71
73
74
76
77
79
82
83
86
88
89
91
92
94
97
98
Happy Corona days
```

In [166…

```
'''WAp tto print the numbers from 1 to 100 and print odd numbers'''
```

```python
for i in range(1,101):
    if(i%2==0):
        pass
    else:
        print(i)
```

```
1
3
5
7
9
11
13
15
17
19
21
23
25
27
29
31
33
35
37
39
41
43
45
47
49
51
53
55
57
59
61
63
65
67
69
71
73
75
77
79
81
83
85
87
89
91
93
95
97
99
```

Functions:

A function is a block of organized and reusable program code that performs a single, specific and well defined task. Python enables its programmers to break up a program into functions, each of which can be written more or less independently of the others. Fig explains how a function func1() is called to Perform a well defined task then program Control is passed to the first statement in the Function. All the statements in the function are Executed and then the program control is passed to the statement following the one that called a function.

Defining a function means specifying its name, parameters that are expected, and the set of instructions. Once the basic structure of a function is finalized. It can be executed by calling it. The function call statement invokes the function. When function is invoked, the program control jumps in the called function to execute the statements that are a part of that function. Once the called function is executed. The program control passes back to the calling function. The syntax of calling A function that does not accept parameters is simply the name of the function is followed by parenthesis. function_name:() Function call statements has the following syntax when it accepts parameters Function call statement has the following syntax, when it accepts parameters. Function_name(variable1,variabl:e2..) When the function is called, the interpreter checks that the correct number and type of arguments are used in the function call. It also checks the type of the returned value.

In [1]:
```
# def is a keyword, sub is a function name x and y are function arguments
def sub(x,y):
    return(x-y)
x=20
y=10
print(sub(x,y))      # function is called
```

```
10
```

In [3]:
```
def func():
    for i in range(4):
        print("Vitians")
func()
```

```
Vitians
Vitians
Vitians
Vitians
```

In [5]:
```
def mul(a,b):               # def is a keyword, sub is a function name x a
    return(a*b)
a=20
b=10
print(mul(a,b))
```

```
200
```

In [7]:
```
def func(i):
    print("Hello world",i)
i=10
func(i)
```

```
Hello world 10
```

In [9]:
```
def func(i):                    #function definition header accepts a vari
    print("Hello world",i)
j=10
func(j)                           # function is called using variable j
```

```
Hello world 10
```

In [11]:
```
def func(i):
    print("Hello world",i)
func(5+2+3+5*6)
```

```
Hello world 40
```

In [13]:
```
# Program to add 2 integers using functions
def total(a,b):
    result=a+b
    print("Sum of",a,"and",b,"=",result)
a=int(input("Enter the first number:"))
b=int(input("Enter the second number:"))
total(a,b)
```

```
Sum of 5 and 6 = 11
```

In [15]:
```
#Program can be written in another way by chaning the variable names in t
def add(a,b):
    c=a+b
    return c
x=int(input("Enter a number a:"))
y=int(input("Enter a number b:"))
Z=add(x,y)
print("Addition of two numbers is",Z)
```

```
Addition of two numbers is 13
```

In [17]:
```
def display():
    print("In function")
    print("About to execute return statement")
    return
    print("This line will never be displayed")
display()
print("back to the caller")
```

```
In function
About to execute return statement
back to the caller
```

In [19]:
```
def returnfunction(num):
    num=num*2
    return num
x=returnfunction(4)
x
```

Out[19]:
```
8
```

In [25]:
```
# A program for default argument assumes a default value if a value is no
def display(name,course='BTECH'):
    print("Name:"+name)
    print("Course:",course)
display(course='MS',name="Raj")    #keyword arguments
display(name='King GJ')              #default argument for course
```

```
Name:Raj
Course: MS
Name:King GJ
Course: BTECH
```

In [27]:
```python
# program to demonstrate keyword arguments
def display(str,int_x,float_y):
    print("The string is:",str)
    print("The integer value is:",int_x)
    print("The floating point value is:",float_y)
display(str="i love u",int_x=3000,float_y=0.0)
```

```
The string is: i love u
The integer value is: 3000
The floating point value is: 0.0
```

In [29]:
```python
# program to demonstrate keyword arguments using assignment operators and
def display(name,age,salary):
    print("Name:",name)
    print("Age:",age)
    print("Salary:",salary)
N='GJ'
A=29
S=123456
display(salary=S,age=A,name=N)
```

```
Name: GJ
Age: 29
Salary: 123456
```

In [31]:
```python
def fun(name,*fav_movies_series):
    print('\n', name,"likes to watch")
    for movies in fav_movies_series:
        print(movies,end=" ")
fun("GJ","Multiverse of Madness","Moon Knight","Rings of power","Thor-Lov
fun("KING","Black adam","The Batman","Aquman-The lost kingdoms")
fun("Raj")
```

```
 GJ likes to watch
Multiverse of Madness Moon Knight Rings of power Thor-Love and thunder
 KING likes to watch
Black adam The Batman Aquman-The lost kingdoms
 Raj likes to watch
```

In [ ]:
```
Strings:
```

In [33]:
```python
# defining strings in Python
# all of the following are equivalent
first_string = 'Hello'
print(first_string)

second_string = "Hello"
print(second_string)

third_string = '''Hello'''
print(third_string)

# triple quotes string can extend multiple lines
sentence_string = '''Hello, welcome to New horizon college of engineering
print(sentence_string)
```

```
Hello
Hello
Hello
Hello, welcome to New horizon college of engineering
```

In [35]:
```python
#Accessing string characters in Python
str = 'GELUVARAJ'
print('str = ', str)

print('str[0:9]=',str)

#first character
print('str[0] = ',str[0])

#last character
print('str[-1] =',str[-1])

#slicing 2nd to 5th character
print('str[1:5] = ', str[1:5])

#slicing 6th to 2nd last character
print('str[5:-2] = ', str[5:-2])

#slicing the string -9 to 6
print('str[-9:5]=',str[-9:5])

#
print('str[3] = ',str[3:])
```

```
str =  GELUVARAJ
str[0:9]= GELUVARAJ
str[0] =  G
str[-1] = J
str[1:5] =  ELUV
str[5:-2] =  AR
str[-9:5]= GELUV
str[3] =  UVARAJ
```

In [37]:
```python
# Concatenation of a string
a='King of'
b=' Hearts'
print(a+b)
```

```
King of Hearts
```

In [39]:
```python
# Iterating through a string and finding the letters count
count = 0
for letter in 'New horizon college of engineering':
    if(letter == 'l'):
        count = count+1
print(count,'letters found')
```

```
2 letters found
```

In [41]:
```python
text = 'Everything is fair in love and war'
# Splits at space
print(text.split())

word = 'Everything,is, fair, in, love, and, war'
# Splits at ','
print(word.split(','))
```

```
word = 'Everything:is:fair:in:love:and,:war'
# Splitting at ':'
print(word.split(':'))

word ='catbatsatfatratchat'
# Splitting at t
print(word.split('t'))
```

```
['Everything', 'is', 'fair', 'in', 'love', 'and', 'war']
['Everything', 'is', ' fair', ' in', ' love', ' and', ' war']
['Everything', 'is', 'fair', 'in', 'love', 'and,', 'war']
['ca', 'ba', 'sa', 'fa', 'ra', 'cha', '']
```

In [43]:
```
#Removing a Character from String using the Naive method
input_str = "Geluvaraj"

# Printing original string
print ("Original string: ",input_str)

result_str = ""

for i in range(0, len(input_str)):
    if i != 3:
        result_str = result_str + input_str[i] # done slicing and concate

# Printing string after removal
print ("String after removal of i'th character : ",result_str)
```

```
Original string:  Geluvaraj
String after removal of i'th character :  Gelvaraj
```

In [45]:
```
#Removal of Character from a String using Slicing and Concatenation
str = "Geluvaraj"
print ("Original string: " + str)

# Removing char at pos 3
# using slice + concatenation
res_str = str[:2]
print ("String after removal of character: " + res_str)
```

```
Original string: Geluvaraj
String after removal of character: Ge
```

In [47]:
```
#Comparison Operator
string1 = "Geluvaraj"
string2 = "Geluvaraj1"
string3 = "Geluvaraj2"
string4 = "Geluvaraj3"
print(string1==string4)
print(string2==string3)
print(string1!=string4)
print(string2!=string3)
```

```
False
False
True
True
```

In [51]:
```
name = "GELUVARAJ"
YOB= 1990
Height = 5.6
```

```
string1 = 'Hey %s' % (name)
print(string1)

string2 = 'my YOB is %d' % (YOB)
print(string2)

string3= 'My name is %s, my year if birth is %d' % (name, YOB)
print(string3)

string3= 'My name is %s, my height is %f' % (name, Height)
print(string3)
```

```
Hey GELUVARAJ
my YOB is 1990
My name is GELUVARAJ, my year if birth is 1990
My name is GELUVARAJ, my height is 5.600000
```

In [53]:
```
#You can use multiple format conversion types in a single print statement
variable = 12

string = "Variable as integer = %d \n\
Variable as float = %f" %(variable, variable)

print (string)
```

```
Variable as integer = 12
Variable as float = 12.000000
```

In [55]:
```
a = 5
b = 10
print(f"My age is {2 * (a + b)}.")
```

```
My age is 30.
```

Built-in functions:

In [57]:
```
str1 = "geluvaraj"
str2 = str1.capitalize()
print(str2)
```

```
Geluvaraj
```

In [59]:
```
str1 = 'HELLO FROM RAJ'
str2 = str1.casefold()
print(str2)
```

```
hello from raj
```

In [61]:
```
str1 = "geluvaraj"
str2 = str1.index('l')
print(str2)
```

```
2
```

In [71]:
```
str1 = "VIT"
str2 = str1.lower()
print(str2)
```

```
vit
```

In [73]:
```
str1 = "VIT"
str2 = str1.swapcase()
print(str2)
```

```
vit
```

In [67]:
```
str1 = "geluvaraj"
str2 = str1.upper()
print(str2)
```

```
GELUVARAJ
```

In [ ]:
```
LISTS : A list is an ordered sequence of values. It is a data structure i
For example,
ls1=[10,-4, 25, 13]
ls2=[“Tiger”, “Lion”, “Cheetah”]
Here, ls1 is a list containing four integers, and ls2 is a list containin
ls3=[3.5, ‘Tiger’, 10, [3,4]]
Here, ls3 contains a float, a string, an integer and a list. This illustr
An empty list can be created any of the following ways -
>>> ls =[]                                    ls=list()
>>> type(ls)                 or               type(ls)
<class 'list’>                               <class ‘list’>
```

In [76]:
```
ls=[34, 'hi', [2,3],-5]
print(ls[3*1])
```

```
-5
```

In [78]:
```
ls=[34, 'hi','hello',-5]
for item in ls:
     print(item)
```

```
34
hi
hello
-5
```

In [80]:
```
ls=[34, 'hi','hello',-5]
for item in ls:
     print(ls)
```

```
[34, 'hi', 'hello', -5]
[34, 'hi', 'hello', -5]
[34, 'hi', 'hello', -5]
[34, 'hi', 'hello', -5]
```

In [82]:
```
ls=[1,2,3,4]
for i in range(len(ls)):
    ls[i]=ls[i]**2
    print(ls)
```

```
[1, 2, 3, 4]
[1, 4, 3, 4]
[1, 4, 9, 4]
[1, 4, 9, 16]
```

In [84]:
```
list1 =['red','green','yellow','black']
for i in range(len(list1)):
    print(list1[i])
```

```
red
green
yellow
black
```

In [86]:
```
ls1=[1,2,3]
ls2=[5,6,7]
print(ls1+ls2)
```

```
[1, 2, 3, 5, 6, 7]
```

In [88]:
```
ls1=[1,2,3]
print(ls1*3)
```

```
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

In [90]:
```
t=['a','b','c','d','e']                    #reverse a list
print(t[::-1])
```

```
['e', 'd', 'c', 'b', 'a']
```

In [92]:
```
print(t[:])
```

```
['a', 'b', 'c', 'd', 'e']
```

In [94]:
```
print(t[:3])
```

```
['a', 'b', 'c']
```

In [98]:
```
print(t[:-2])
```

```
['a', 'b', 'c']
```

In [100…
```
list1 = ['red','green','blue']
'green' in list1
```

Out[100… True

In [102…
```
list1 = ['red','green','blue']
'violet' in list1
```

Out[102… False

In [104…
```
#to remove last element from the list
ls=[3,6,-2,8,10]
x=ls.pop()           #10 is removed from list and stored in x
print(ls)
print(x)
```

```
[3, 6, -2, 8]
10
```

In [106…
```
t = ['a', 'b', 'c']
x = t.pop(1)               #item at index 1 is popped
print(t)
print(x)
```

```
['a', 'c']
b
```

In [1]:
```
ls=[3,6,-2,8,1]
del ls[2] #item at index 2 is deleted
```

```
print(ls)
```

```
[3, 6, 8, 1]
```

In [3]:
```
ls=[3,6,-2,8,1]
del ls[1:4]          #deleting all elements from index 1 to 3
print(ls)
```

```
[3, 1]
```

In [5]:
```
ls=['a','b','c','d','e']
del ls[1::2]
print(ls)
```

```
['a', 'c', 'e']
```

In [114…
```
list1 = [10,20,30,40,50]                    #length of the list
len(list1)
```

Out[114… 5

In [116…
```
str1='aeiou'                              #string converted to list
list1 = list(str1)
print(list1)
```

```
['a', 'e', 'i', 'o', 'u']
```

In [118…
```
list1 = [10,20,30,40]                          #adding an element to the lis
list1.append(50)
print(list1)

list1 = [10,20,30,40]
list1.append([50,60])
print(list1)
```

```
[10, 20, 30, 40, 50]
[10, 20, 30, 40, [50, 60]]
```

In [120…
```
list1 = [10,20,30,40,50]  #inserts element 25 at index value 2
list1.insert(2,25)
print(list1)


list1 = [10,20,30,40,50]  #inserts element 0 at index value 0
list1.insert(0,0)
print(list1)
```

```
[10, 20, 25, 30, 40, 50]
[0, 10, 20, 30, 40, 50]
```

In [122…
```
list1 = ['Tiger','Zebra','Lion', 'Cat', 'Elephant' ,'Dog']
list1.sort()
print(list1)


list1 = [34,66,12,89,28,99]
list1.sort(reverse = True)       #to sort in descending order
print(list1)
```

```
['Cat', 'Dog', 'Elephant', 'Lion', 'Tiger', 'Zebra']
[99, 89, 66, 34, 28, 12]
```

In [124…
```
list1 = [23,45,11,67,85,56]          #creates a new list
list2 = sorted(list1)
print(list1)
print(list2)
```

```
[23, 45, 11, 67, 85, 56]
[11, 23, 45, 56, 67, 85]
```

Program 4-1 Write a program to allow user to perform any those list operation given in a menu. The menu is:

1.

Append an elemen2.t Insert an eleme3.nt Append a list to the given l4.ist Modify an existing ele5.ment Delete an existing element from its pos6.ition Delete an existing element with a given7. value Sort the list in the ascendin8.g order Sort the list in descendi9.ng order Display the list.

In [ ]:
```
#Menu driven program to do various list operations
myList = [22,4,16,38,13] #myList having 5 elements
choice = 0
for attempt in range (3):
    print ("Attempt number:", attempt)
    print("The list 'myList' has the following elements", myList)
    print("\nLIST OPERATIONS")
    print(" 1. Append an element")
    print(" 2. Insert an element at the desired position")
    print(" 3. Append a list to the given list")
    print(" 4. Modify an existing element")
    print(" 5. Delete an existing element by its position")
    print(" 6. Delete an existing element by its value")
    print(" 7. Sort the list in ascending order")
    print(" 8. Sort the list in descending order")
    print(" 9. Display the list")
    choice = int(input("ENTER YOUR CHOICE (1-9): "))
#append element
    if choice == 1:
        element = eval(input("Enter the element to be appended: "))
        myList.append(element)
        print("The element has been appended\n",myList)
#insert an element at desired position
    elif choice == 2:
        element = eval(input("Enter the element to be inserted: "))
        pos = int(input("Enter the position:"))
        myList.insert(pos,element)
        print("The element has been inserted\n",myList)
#append a list to the given list
    elif choice == 3:
        newList = eval(input("Enter the list to be appended: "))
        myList.extend(newList)
        print("The list has been appended\n",myList)
#modify an existing element
    elif choice == 4:
        i = int(input("Enter the position of the element to be modified:
        if i < len(myList):
            newElement = eval(input("Enter the new element: "))
            oldElement = myList[i]
            myList[i] = newElement
```

```
            print("The element",oldElement,"has been modified\n",myList)
        else:
            print("Position of the element is more then the length of lis
#delete an existing element by position
    elif choice == 5:
        i = int(input("Enter the position of the element to be deleted: "
        if i < len(myList):
            element = myList.pop(i)
            print("The element",element,"has been deleted\n",myList)
        else:
            print("\nPosition of the element is more then the length of l
#delete an existing element by value
    elif choice == 6:
        element = int(input("\nEnter the element to be deleted: "))
        if element in myList:
            myList.remove(element)
            print("\nThe element",element,"has been deleted\n",myList)
        else:
            print("\nElement",element,"is not present in the list")
#list in sorted order
    elif choice == 7:
        myList.sort()
        print("\nThe list has been sorted",myList)
#list in reverse sorted order
    elif choice == 8:
        myList.sort(reverse = True)
        print("\nThe list has been sorted in reverse order",myList)
#display the list
    elif choice == 9:
        print("\nThe list is:",myList)
    else:
        print("Choice is not valid")
```

```
Attempt number: 0
The list 'myList' has the following elements [22, 4, 16, 38, 13]

LIST OPERATIONS
 1. Append an element
 2. Insert an element at the desired position
 3. Append a list to the given list
 4. Modify an existing element
 5. Delete an existing element by its position
 6. Delete an existing element by its value
 7. Sort the list in ascending order
 8. Sort the list in descending order
 9. Display the list
The list is: [22, 4, 16, 38, 13]
Attempt number: 1
The list 'myList' has the following elements [22, 4, 16, 38, 13]

LIST OPERATIONS
 1. Append an element
 2. Insert an element at the desired position
 3. Append a list to the given list
 4. Modify an existing element
 5. Delete an existing element by its position
 6. Delete an existing element by its value
 7. Sort the list in ascending order
 8. Sort the list in descending order
 9. Display the list
```

```
The element has been appended
 [22, 4, 16, 38, 13, 25]
Attempt number: 2
The list 'myList' has the following elements [22, 4, 16, 38, 13, 25]

LIST OPERATIONS
 1. Append an element
 2. Insert an element at the desired position
 3. Append a list to the given list
 4. Modify an existing element
 5. Delete an existing element by its position
 6. Delete an existing element by its value
 7. Sort the list in ascending order
 8. Sort the list in descending order
 9. Display the list
```

Try all the built in functions of Lists:

1.len() 2.list() 3.extend() 4.insert() 5.count() 6.find() 7.remove() 8.pop() 9.reverse() 10.min() 11.max() 12.sum()

TUPLES : A tuple is a sequence of items, similar to lists. The values stored in the tuple can be of any type and they are indexed using integers. Unlike lists, tuples are immutable. That is, values within tuples cannot be modified/reassigned. Tuples are comparable and hashable objects. Hence, they can be made as keys in dictionaries.

In [7]:
```
t='Mango','apple','banana'         #without parentheses
print(t)
```

```
('Mango', 'apple', 'banana')
```

In [9]:
```
t1=('Mango', 'apple', 'banana')
print(t1)
type(t1)
```

```
('Mango', 'apple', 'banana')
```

Out[9]:
```
tuple
```

In [11]:
```
t=tuple([3,[12,5],'Hi'])
print(t)
type(t)
```

```
(3, [12, 5], 'Hi')
```

Out[11]:
```
tuple
```

In [13]:
```
t=('Mango', 'Banana', 'Apple')          #accessing element using index
print(t[1])
```

```
Banana
```

In [15]:
```
t=('Mango', 'Banana', 'Apple')
print(t[1:])                          #slicing using 1st element
print(t[-1])
```

```
('Banana', 'Apple')
Apple
```

In [17]:
```
tup = (5,11,22)
for i in range(0,len(tup)):
    print(tup[i])
```

```
5
11
22
```

In [19]:
```
T = (10,20,30,40,50)
for var in T:
    print (T.index(var),var)
```

```
0 10
1 20
2 30
3 40
4 50
```

In [21]:
```
T = (10,20,30,40,50)
for var in range(len(T)):
    print(var,T[var])
```

```
0 10
1 20
2 30
3 40
4 50
```

In [23]:
```
#updating a tuples
tup1 = (1, 2)
tup2 = ('a','b')
tup3 = tup1 + tup2
print (tup3)
```

```
(1, 2, 'a', 'b')
```

In [25]:
```
tup1 = ('a', 'b', 'c', 'd', 'e', 'f')
tup1[1:3]
```

Out[25]:
```
('b', 'c')
```

In [27]:
```
tup1[0:6:2]
```

Out[27]:
```
('a', 'c', 'e')
```

In [29]:
```
tup1[3:]
```

Out[29]:
```
('d', 'e', 'f')
```

In [31]:
```
tup1=tuple(range(4))
print(tup1)
```

```
(0, 1, 2, 3)
```

In [33]:
```
tup3=tuple(range(2, 10, 2))
print(tup3)
```

```
(2, 4, 6, 8)
```

In [35]:
```
tup1 = (1, 2,3)
tup3 = tup1[0:1] + tup1[2:]
```

```
print (tup3)
```

```
(1, 3)
```

In [37]:
```
#packing and unpacking of tuple

tup1 = (1,2,3,4)
(a, b, c, d) = tup1
print(tup1)
```

```
(1, 2, 3, 4)
```

Write a Python program to unpack a tuple in several variables.

In [41]:
```
#create a tuple
tuplex = 4, 8, 3
print(tuplex)
n1, n2, n3 = tuplex
#unpack a tuple in variables
print(n1 + n2 + n3)
```

```
(4, 8, 3)
15
```

Write a Python program to find the repeated items of a tuple.

In [43]:
```
tuplex = 2, 4, 5, 6, 2, 3, 4, 4, 7
print(tuplex)
count = tuplex.count(4)
print(count)
```

```
(2, 4, 5, 6, 2, 3, 4, 4, 7)
3
```

Write a Python program to find the index of an item of a tuple

In [45]:
```
#create a tuple
tuplex = tuple("index tuple")
print(tuplex)
#get index of the first item whose value is passed as parameter
index = tuplex.index("p")
print(index)
#define the index from which you want to search
index = tuplex.index("p")
print(index)
#define the segment of the tuple to be searched
index = tuplex.index("e")
print(index)
```

```
('i', 'n', 'd', 'e', 'x', ' ', 't', 'u', 'p', 'l', 'e')
8
8
3
```

Write a Python program to replace last value of tuples in a list.

In [47]:
```
l = [(10, 20, 40), (40, 50, 60), (70, 80, 90)]
print([t[:-1] + (100,) for t in l])
```

```
[(10, 20, 100), (40, 50, 100), (70, 80, 100)]
```

In [ ]:
```
Write a Python program to remove an empty tuple(s) from a list of tuples.
```

In [53]:
```
list=[(1,2,3),(4,5),()]
for tuple in list:
    if(tuple==0):
        list.remove(tuple)
print(list)
```

```
[(1, 2, 3), (4, 5), ()]
```

Try all the built in functions of Lists:

1.len() 2.list() 3.extend() 4.insert() 5.count() 6.find() 7.remove() 8.pop() 9.reverse()
10.min() 11.max() 12.sum()

Sets: A set is an unordered collection of data that is iterable, mutable, and has no duplicate elements. Python's set class represents the mathematical notion of a set. The major advantage of using a set, as opposed to a list, is that it has a highly optimized method for checking whether a specific element is contained in the set. The set data type is based on a data structure known as a hash table. A set is an unordered collection of items. Every element must be unique and immutable. Duplicates are eliminated Elements are Immutable like tuples, integers, strings etc. However, the set itself is mutable. It is possible to add or remove items to set. We can change its state—its length and content. Sets can be used to perform mathematical set operations like union, intersection, symmetric difference etc.

In [55]:
```
print("Create a new set:")
x = set()
print(x)
print(type(x))
print("\nCreate a non empty set:")
n = set([0, 1, 2, 3, 4])
print(n)
print(type(n))
print("\nUsing a literal:")
a = {1,2,3,'foo','bar'}
print(type(a))
print(a)
```

```
Create a new set:
set()
<class 'set'>

Create a non empty set:
{0, 1, 2, 3, 4}
<class 'set'>

Using a literal:
<class 'set'>
{1, 2, 3, 'bar', 'foo'}
```

In [59]:
```
#A new empty set
color_set = set()
print(color_set)
print("\nAdd single element:")
color_set.add("Red")
print(color_set)
print("\nAdd multiple items:")
color_set.update(["Blue", "Green"])
print(color_set)
```

```
set()

Add single element:
{'Red'}

Add multiple items:
{'Red', 'Green', 'Blue'}
```

In [61]:
```
A = {1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 4, 4,
print(A)
```

```
{1, 2, 3, 4, 5}
```

In [67]:
```
num_set = set([0, 1, 3, 4, 5])
print("Original set:")
print(num_set)
num_set.pop()
print("\nAfter removing the first element from the said set:")
print(num_set)
```

```
Original set:
{0, 1, 3, 4, 5}

After removing the first element from the said set:
{1, 3, 4, 5}
```

In [69]:
```
num_set = set([0, 1, 2, 3, 4, 5])
print("Original set elements:")
print(num_set)
num_set.discard(3)
print(num_set)
num_set.discard(0)
print(num_set)
num_set.discard(5)
print(num_set)
```

```
Original set elements:
{0, 1, 2, 3, 4, 5}
{0, 1, 2, 4, 5}
{1, 2, 4, 5}
{1, 2, 4}
```

In [71]:
```
myset = {'Apples', 'Bananas', 'Oranges'}
myset.discard('Apples')
print(myset)
```

```
{'Bananas', 'Oranges'}
```

In [73]:
```
setx = set(["green", "blue"])
sety = set(["blue", "yellow"])
print("Original set elements:")
print(setx)
print(sety)
print("\nIntersection of two said sets:")
setz = setx & sety
print(setz)
```

```
Original set elements:
{'green', 'blue'}
{'yellow', 'blue'}

Intersection of two said sets:
{'blue'}
```

In [75]:
```
setc1 = set(["green", "blue"])
setc2 = set(["blue", "yellow"])
print("Original sets:")
print(setc1)
print(setc2)
setc = setc1.union(setc2)
print("\nUnion of above sets:")
print(setc)
setn1 = set([1, 1, 2, 3, 4, 5])
setn2 = set([1, 5, 6, 7, 8, 9])
print("\nOriginal sets:")
print(setn1)
print(setn2)
print("\nUnion of above sets:")
setn = setn1.union(setn2)
print(setn)
```

```
Original sets:
{'green', 'blue'}
{'yellow', 'blue'}

Union of above sets:
{'green', 'yellow', 'blue'}

Original sets:
{1, 2, 3, 4, 5}
{1, 5, 6, 7, 8, 9}

Union of above sets:
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

In [77]:
```
setc1 = set(["green", "blue"])
setc2 = set(["blue", "yellow"])
print("Original sets:")
print(setc1)
print(setc2)
r1 = setc1.difference(setc2)
print("\nDifference of setc1 - setc2:")
print(r1)
r2 = setc2.difference(setc1)
print("\nDifference of setc2 - setc1:")
print(r2)
setn1 = set([1, 1, 2, 3, 4, 5])
setn2 = set([1, 5, 6, 7, 8, 9])
print("\nOriginal sets:")
print(setn1)
print(setn2)
r1 = setn1.difference(setn2)
print("\nDifference of setn1 - setn2:")
print(r1)
r2 = setn2.difference(setn1)
print("\nDifference of setn2 - setn1:")
print(r2)
```

```
Original sets:
{'green', 'blue'}
{'yellow', 'blue'}

Difference of setc1 - setc2:
{'green'}

Difference of setc2 - setc1:
{'yellow'}

Original sets:
{1, 2, 3, 4, 5}
{1, 5, 6, 7, 8, 9}

Difference of setn1 - setn2:
{2, 3, 4}

Difference of setn2 - setn1:
{8, 9, 6, 7}
```

In [79]:

```python
setc1 = set(["green", "blue"])
setc2 = set(["blue", "yellow"])
print("Original sets:")
print(setc1)
print(setc2)
r1 = setc1.difference(setc2)
print("\nDifference of setc1 - setc2:")
print(r1)
r2 = setc2.difference(setc1)
print("\nDifference of setc2 - setc1:")
print(r2)
setn1 = set([1, 1, 2, 3, 4, 5])
setn2 = set([1, 5, 6, 7, 8, 9])
print("\nOriginal sets:")
print(setn1)
print(setn2)
r1 = setn1.difference(setn2)
print("\nDifference of setn1 - setn2:")
print(r1)
r2 = setn2.difference(setn1)
print("\nDifference of setn2 - setn1:")
print(r2)
```

```
Original sets:
{'green', 'blue'}
{'yellow', 'blue'}

Difference of setc1 - setc2:
{'green'}

Difference of setc2 - setc1:
{'yellow'}

Original sets:
{1, 2, 3, 4, 5}
{1, 5, 6, 7, 8, 9}

Difference of setn1 - setn2:
{2, 3, 4}

Difference of setn2 - setn1:
{8, 9, 6, 7}
```

In [81]:
```
setp = set(["Red", "Green"])
setq = set(["Green", "Red"])
#A shallow copy
setr = setp.copy()
print(setr)
```

```
{'Red', 'Green'}
```

In [83]:
```
setc = {"Red", "Green", "Black", "White"}
print("Original set elements:")
print(setc)
print("\nAfter removing all elements of the said set.")
setc.clear()
print(setc)
```

```
Original set elements:
{'Red', 'Green', 'Black', 'White'}

After removing all elements of the said set.
set()
```

In [85]:
```
x = {1,2,3,4}
y = {4,5,6,7}
z = {8}
print("Original set elements:")
print(x)
print(y)
print(z)
print("\nConfirm two given sets have no element(s) in common:")
print("\nCompare x and y:")
print(x.isdisjoint(y))
print("\nCompare x and z:")
print(z.isdisjoint(x))
print("\nCompare y and z:")
print(y.isdisjoint(z))
```

```
Original set elements:
{1, 2, 3, 4}
{4, 5, 6, 7}
{8}

Confirm two given sets have no element(s) in common:

Compare x and y:
False

Compare x and z:
True

Compare y and z:
True
```

the symmetric difference, also known as the disjunctive union, of two sets is the set of elements which are in either of the sets and not in their intersection.

In [87]:
```
setc1 = set(["green", "blue"])
setc2 = set(["blue", "yellow"])
print("Original sets:")
print(setc1)
print(setc2)
r1 = setc1.symmetric_difference(setc2)
print("\nSymmetric difference of setc1 - setc2:")
print(r1)
r2 = setc2.symmetric_difference(setc1)
print("\nSymmetric difference of setc2 - setc1:")
print(r2)
setn1 = set([1, 1, 2, 3, 4, 5])
setn2 = set([1, 5, 6, 7, 8, 9])
print("\nOriginal sets:")
print(setn1)
print(setn2)
r1 = setn1.symmetric_difference(setn2)
print("\nSymmetric difference of setn1 - setn2:")
print(r1)
r2 = setn2.symmetric_difference(setn1)
print("\nSymmetric difference of setn2 - setn1:")
print(r2)
```

```
Original sets:
{'green', 'blue'}
{'yellow', 'blue'}

Symmetric difference of setc1 - setc2:
{'green', 'yellow'}

Symmetric difference of setc2 - setc1:
{'green', 'yellow'}

Original sets:
{1, 2, 3, 4, 5}
{1, 5, 6, 7, 8, 9}

Symmetric difference of setn1 - setn2:
{2, 3, 4, 6, 7, 8, 9}

Symmetric difference of setn2 - setn1:
{2, 3, 4, 6, 7, 8, 9}
```

In [89]:

```python
# set of letters
s = {'g', 'e', 'k', 's'}

# adding 's'
s.add('f')
print('Set after updating:', s)

# Discarding element from the set
s.discard('g')
print('\nSet after updating:', s)

# Removing element from the set
s.remove('e')
print('\nSet after updating:', s)

# Popping elements from the set
print('\nPopped element', s.pop())
print('Set after updating:', s)

s.clear()
print('\nSet after updating:', s)
```

```
Set after updating: {'k', 'e', 's', 'g', 'f'}

Set after updating: {'k', 'e', 's', 'f'}

Set after updating: {'k', 's', 'f'}

Popped element k
Set after updating: {'s', 'f'}

Set after updating: set()
```

Dictionary: The data type dictionary falls under mapping. It is a mapping between a set of keys and a set of values. The key-value pair is called an item. A key is separated from its value by a colon(:) and consecutive items are separated by commas. Items in dictionaries are unordered, so we may not get back the data in the same order in which we had entered the data initially in the dictionary

In [91]:
```
# empty dictionary
my_dict = {}
print(my_dict)

# dictionary with integer keys
my_dict = {1: 'Geluva', 2: 'Raj'}
print(my_dict)

# dictionary with mixed keys
my_dict = {'name': 'Raj', 1: [2, 4, 3]}
print(my_dict)

# using dict()
my_dict = dict({1:'King', 2:'Maker'})
print(my_dict)

# from sequence having each item as a pair
my_dict = dict([(1,'Geluva'), (2,'Raj')])
print(my_dict)
```

```
{}
{1: 'Geluva', 2: 'Raj'}
{'name': 'Raj', 1: [2, 4, 3]}
{1: 'King', 2: 'Maker'}
{1: 'Geluva', 2: 'Raj'}
```

In [95]:
```
# Changing and adding Dictionary Elements
my_dict = {'name': 'Geluvaraj', 'age': 29}

# update value
my_dict['age'] = 30

#Output: {'age': 30, 'name': 'Geluvaraj'}
print(my_dict)

# add item
my_dict['address'] = 'Bangalore'

# Output: {'address': 'Bangalore', 'age': 30, 'name': 'Geluvaraj'}
print(my_dict)
```

```
{'name': 'Geluvaraj', 'age': 30}
{'name': 'Geluvaraj', 'age': 30, 'address': 'Bangalore'}
```

In [101…
```
# Removing elements from a dictionary

# create a dictionary
square_roots = {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
print(square_roots)

# remove a particular item, returns its value
# Output: 16
print(square_roots.pop(4))

# Output: {1: 1, 2: 4, 3: 9, 5: 25}
print(square_roots)

# remove an arbitrary item, return (key,value)
#The popitem() method removes the item that was last inserted into the di
# Output: (5, 25)
```

```
print(square_roots.popitem())

# Output: {1: 1, 2: 4, 3: 9}
print(square_roots)

# remove all items
square_roots.clear()

# Output: {}
print(square_roots)

# delete the dictionary itself
del square_roots
```

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
16
{1: 1, 2: 4, 3: 9, 5: 25}
(5, 25)
{1: 1, 2: 4, 3: 9}
{}
```

In [107…

```
# Create a dictionary with specified keys and default value 0
marks = {}.fromkeys(['Maths', 'Python Programming', 'Chemistry', 'Electro

# Print the dictionary
print(marks)

# Loop through dictionary items and print them
for item in marks.items():
    print(item)

# Print the sorted list of dictionary keys
print(list(sorted(marks.keys())))
```

```
{'Maths': 0, 'Python Programming': 0, 'Chemistry': 0, 'Electronics': 0}
('Maths', 0)
('Python Programming', 0)
('Chemistry', 0)
('Electronics', 0)
```

```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
Cell In[107], line 12
      9     print(item)
     11 # Print the sorted list of dictionary keys
---> 12 print(list(sorted(marks.keys())))

TypeError: 'list' object is not callable
```

In [109…

```
# Membership Test for Dictionary Keys
square_roots = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}

# Output: True
print(1 in square_roots)

# Output: True
print(2 not in square_roots)

# membership tests for key only not value
```

```
# Output: False
print(49 in square_roots)
```

```
True
True
False
```

In [111…
```
# Iterating through a Dictionary
squares = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}
for i in squares:
    print(squares[i])
```

```
1
9
25
49
81
```

In [113…
```
# Dictionary Built-in Functions
square_roots  = {0: 0, 1: 1, 3: 9, 5: 25, 7: 49, 9: 81}

# Output: False
print(all(square_roots))

# Output: True
#Return True if any key of the dictionary is true. If the dictionary is e
print(any(square_roots))

# Output: 6
print(len(square_roots))

# Output: [0, 1, 3, 5, 7, 9]
print(sorted(square_roots))


d = {0:10, 1:20}
print(d)
d.update({2:30})
print(d)
```

```
False
True
6
[0, 1, 3, 5, 7, 9]
{0: 10, 1: 20}
{0: 10, 1: 20, 2: 30}
```

Python program to check whether a given key already exists in a dictionary.

d = {1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60} def is_key_present(x): if x in d: print('Key is present in the dictionary') else: print('Key is not present in the dictionary') is_key_present(5) is_key_present(9)

In [117…
```
#Python script to merge two Python dictionaries.
d1 = {'a': 100, 'b': 200}
d2 = {'x': 300, 'y': 200}
```

```
d = d1.copy()
d.update(d2)
print(d)
```

```
{'a': 100, 'b': 200, 'x': 300, 'y': 200}
```

In [119…
```
#Python program to sum all the items in a dictionary.
my_dict = {'data1':100,'data2':-54,'data3':247}
print(sum(my_dict.values()))
```

```
293
```

write a program to enter names of employees and their salaries as input and store them in a dictionary. Here n is to input by the user.

Program to create a dictionary which stores names of employees and their salary

In [121…
```
n = int(input("Enter the number of employees whose data to be stored: "))
count = 1
employee = dict() #create an empty dictionary
for count in range (n):
    name = input("Enter the name of the Employee: ")
    salary = int(input("Enter the salary: "))
    employee[name] = salary
print("\n\nEMPLOYEE_NAME\tSALARY")
for k in employee:
    print(k,'\t\t',employee[k])
```

```
EMPLOYEE_NAME	SALARY
Raj		 25000
```