Python Introduction

Python is a widely used general-purpose, high level programming language. It was created by Guido van Rossum in 1991 and further developed by the Python Software Foundation. It was designed with an emphasis on code readability, and its syntax allows programmers to express their concepts in fewer lines of code.

Python is a programming language that lets you work quickly and integrate systems more efficiently.

Creating a Comment

In [1]:
```
#This is a comment
'''
This is a multiline comment in Python.
You can write multiple lines of text here.
These lines will be ignored by the interpreter.
'''
print("Python revision Notebook")
```

```
Python revision Notebook
```

Creating Variables

In [45]:
```
i = 10 #int
f = 10.0 # float
st = "String" # str
l = [1,2,3,4,4] # list
t = (5,6,7,8,8) # tuple
s = {9,10,11,12,12} # set
d = {'Name':"Rachit","Age":25} # dict
b = True # bool
c = 10 + 5j # complex
r = range(10) # range
print(f"Variable : {i} is type : {type(i)}")
print(f"Variable : {f} is type : {type(f)}")
print(f"Variable : {st} is type : {type(st)}")
print(f"Variable : {l} is type : {type(l)}")
print(f"Variable : {t} is type : {type(t)}")
print(f"Variable : {s} is type : {type(s)}")
print(f"Variable : {d} is type : {type(d)}")
print(f"Variable : {b} is type : {type(b)}")
print(f"Variable : {c} is type : {type(c)}")
print(f"Variable : {r} is type : {type(r)}")
```

```
Variable : 10 is type : <class 'int'>
Variable : 10.0 is type : <class 'float'>
Variable : String is type : <class 'str'>
Variable : [1, 2, 3, 4, 4] is type : <class 'list'>
Variable : (5, 6, 7, 8, 8) is type : <class 'tuple'>
Variable : {9, 10, 11, 12} is type : <class 'set'>
Variable : {'Name': 'Rachit', 'Age': 25} is type : <class 'dict'>
Variable : True is type : <class 'bool'>
Variable : (10+5j) is type : <class 'complex'>
Variable : range(0, 10) is type : <class 'range'>
```

Variable Type Casting

In [47]:
```python
print("Integer to Float")
num = 10
num_float = float(num)
print(f"{num} before type casting {type(num)}")
print(f"{num_float} after type casting  {type(num_float)}")
print("")

print("Float to Integer")
num = 10.5
num_int = int(num)
print(f"{num} before type casting  {type(num)}")
print(f"{num_int} after type casting  {type(num_int)}")
print("")

print("String to Integer or Float")
string = "123"
num_int = int(string)
num_float = float(string)
print(f"{num} before type casting  {type(num)}")
print(f"{num_int} after type casting  {type(num_int)}")
print(f"{num_float} after type casting  {type(num_float)}")
print("")

print("Integer or Float to String")
num_int = 123
num_float = 123.45
num_str_int = str(num_int)
num_str_float = str(num_float)
print(f"{num_int} before type casting  {type(num_int)}")
print(f"{num_float} before type casting  {type(num_float)}")
print(f"{num_str_int} after type casting  {type(num_str_int)}")
print(f"{num_str_int} after type casting  {type(num_str_int)}")
print("")

print("String to List")
fruits_str = "apple,banana,orange"
fruits_list = fruits_str.split(",")
print(f"{fruits_str} before type casting  {type(fruits_str)}")
print(f"{fruits_list} after type casting  {type(fruits_list)}")
print("")

print("List to String")
fruits_list = ['apple', 'banana', 'orange']
fruits_str = ",".join(fruits_list)
print(f"{fruits_list} before type casting  {type(fruits_list)}")
```

```
print(f"{fruits_str} after type casting  {type(fruits_str)}")
print("")

print("String to Boolean")
bool_str = "True"
bool_val = bool(bool_str)
print(f"{bool_str} before type casting  {type(bool_str)}")
print(f"{bool_val} after type casting  {type(bool_val)}")
print("")

print("Boolean to String")
bool_val = True
bool_str = str(bool_val)
bool_int = int(bool_val)
bool_float = float(bool_val)
print(f"{bool_str} before type casting  {type(bool_str)}")
print(f"{bool_val} after type casting  {type(bool_val)}")
print(f"{bool_int} after type casting  {type(bool_int)}")
print(f"{bool_float} after type casting  {type(bool_float)}")
```

```
Integer to Float
10 before type casting <class 'int'>
10.0 after type casting  <class 'float'>

Float to Integer
10.5 before type casting  <class 'float'>
10 after type casting  <class 'int'>

String to Integer or Float
10.5 before type casting  <class 'float'>
123 after type casting  <class 'int'>
123.0 after type casting  <class 'float'>

Integer or Float to String
123 before type casting  <class 'int'>
123.45 before type casting  <class 'float'>
123 after type casting  <class 'str'>
123 after type casting  <class 'str'>

String to List
apple,banana,orange before type casting  <class 'str'>
['apple', 'banana', 'orange'] after type casting  <class 'list'>

List to String
['apple', 'banana', 'orange'] before type casting  <class 'list'>
apple,banana,orange after type casting  <class 'str'>

String to Boolean
True before type casting  <class 'str'>
True after type casting  <class 'bool'>

Boolean to String
True before type casting  <class 'str'>
True after type casting  <class 'bool'>
1 after type casting  <class 'int'>
1.0 after type casting  <class 'float'>
```

Assign Multiple Values

In [4]:
```
# Assigning and unpacking multiple values
x, y, z = 10, 20, 30
print(x)
print(y)
print(z)
print("")
# Tuple packing
values = x, y, z
print("Tuple packed values :",values)
print("")

# Unpacking values again into separate variables
a, b, c = values

print(a)
print(b)
print(c)
print("")
print("List packing works the same ways tuple unpacking works")
x, y, z = [10, 20, 30]
print(x)
print(y)
print(z)
```

```
10
20
30

Tuple packed values : (10, 20, 30)

10
20
30

List packing works the same ways tuple unpacking works
10
20
30
```

Global Variables

In [5]:
```
# Declare a variable
var = 10
# function definition
def modify_global_variable():
    # Access the global variable
    print("Global variable value:", var)

    # creating local variable for checking
    local_var = 30

    # Modify the global variable
    global global_var
    global_var = 20

    # Access the modified global variable
    print("Modified global variable value:", global_var)
```

```
# function definition to check global variable working
def use_global_variable():
    # Access the global variable
    print("Global variable value:", global_var)

    # checking if we have access of local variable in above function
    try:
        print("Global variable value:",local_var)
    except NameError as e:
        print(e)

# Call the functions
modify_global_variable()
use_global_variable()
```

```
Global variable value: 10
Modified global variable value: 20
Global variable value: 20
name 'local_var' is not defined
```

Data Type And Their Methods

In [6]:

```
# String and Methods
s = " Hello World"
print("Variable :",end = "")
print(s)
print("")

print(s.lower(),": \nConverted the string to lowercase.")
print("")
print(s.upper(),": \nConverted the string to uppercase.")
print("")
print(s.strip(),": \nRemoved leading and trailing whitespace from the string.")
print("")
print(s.split(),": \nSplited the string into a list of substrings based on a default de
print("")
print(" ".join(s),": \nConcatenated elements of an iterable using the string as a separ
print("")
print(s.replace(" ",""),": \nReplaced occurrences of a substring ' ' with another substr
print("")
print(s.find("World"),": \nReturned the index of the first occurrence of a substring 'W
print("")
print(s.startswith("World"),": \nChecking if the string starts with a specific substring
print("")
print(s.endswith("World"),": \nChecking if the string ends with a specific substring 'W

print("")
print("hence Strings are not mutable in Python which means that its value cannot be upd
print(f"Variable :{s}")
```

```
Variable : Hello World

 hello world :
Converted the string to lowercase.

 HELLO WORLD :
Converted the string to uppercase.

Hello World :
Removed leading and trailing whitespace from the string.

['Hello', 'World'] :
Splited the string into a list of substrings based on a default delimiter '' .

  H e l l o   W o r l d :
Concatenated elements of an iterable using the string as a separator.

HelloWorld :
Replaced occurrences of a substring ' ' with another substring ''.

7 :
Returned the index of the first occurrence of a substring 'World'

False :
Checking if the string starts with a specific substring 'World' .

True :
Checking if the string ends with a specific substring 'World' .

hence Strings are not mutable in Python which means that its value cannot be updated.
Variable : Hello World
```

In [7]:
```python
# Integer (int) and Methods
i = 10
print("Variable :",end = "")
print(i)
print("")

print(i.bit_length(), "\nReturns the number of bits required to represent the integer")
print(i.to_bytes(2,byteorder='big'),"\nConverts the integer to a byte representation by
print(i.from_bytes(i.to_bytes(2,byteorder='big'), byteorder='big'), "\nConverts a byte
print(hex(i), "\nReturns the hexadecimal representation of the integer.")
print(abs(-i), "\nReturns the absolute value of the integer.")
```

```
Variable :10

4
Returns the number of bits required to represent the integer
b'\x00\n'
Converts the integer to a byte representation byteorder('big-endian').
10
Converts a byte representation back to an integer.
0xa
Returns the hexadecimal representation of the integer.
10
Returns the absolute value of the integer.
```

In [8]:
```python
# Float (float) and Methods
f = 10.543
print("Variable :",end = "")
```

```
print(f)
print("")

print(f.is_integer(), "\nChecks if the float value is an integer.")
print("")
print(f.as_integer_ratio(), " Returns the float value as a numerator and denominator.")
print("")
print(round(f,2), "Rounds the float value to a specified number of decimal places.")
```

```
Variable :10.543

False
Checks if the float value is an integer.

(1483795339730223, 140737488355328)  Returns the float value as a numerator and denomin
ator.

10.54 Rounds the float value to a specified number of decimal places.
```

In [9]:

```
# List (list) and Methods:
l = [0,1,2,3,4,5]
print("Variable :",end = "")
print(l)
print("")


l.append(6)
print(l, "\nAdds an element to the end of the list.")
print("")
l.extend([7,8,9,10])
print(l, "\nAppends all elements from another list to the current list.")
print("")
l.insert(10,10)
print(l, "\nInserts an element at a specified position in the list.")
print("")
l.remove(0)
print(l, "\nRemoves the first occurrence of a specified element from the list.")
print("")
l.pop()
print(l, "\nRemoves and returns an element at a specified index in the list.")
print("")
l.sort()
print(l, "\nSorts the elements of the list in ascending order.")
print("")
l.reverse()
print(l, "\nReverses the order of elements in the list.")
```

```
Variable :[0, 1, 2, 3, 4, 5]

[0, 1, 2, 3, 4, 5, 6]
Adds an element to the end of the list.

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Appends all elements from another list to the current list.

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10]
Inserts an element at a specified position in the list.

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10]
Removes the first occurrence of a specified element from the list.

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Removes and returns an element at a specified index in the list.

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Sorts the elements of the list in ascending order.

[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
Reverses the order of elements in the list.
```

In [10]:

```python
# Tuple (tuple) and Methods
t = (1,2,3,4,5)
print("Variable :",end = "")
print(t)
print("")

print(t.count(4), "\nReturns the number of occurrences of a specified element in the tu
print("")
print(t.index(4), "\nReturns the index of the first occurrence of a specified element i
```

```
Variable :(1, 2, 3, 4, 5)

1
Returns the number of occurrences of a specified element in the tuple.

3
Returns the index of the first occurrence of a specified element in the tuple.
```

In [11]:

```python
# Dictionary (dict) and Methods
d = {'name': 'John', 'age': 30, 'city': 'New York'}
prof = {'profession': 'Engineer'}
print("Variable :",end = "")
print(d)
print("")

print(d.keys())
print("Returns a view object of all keys in the dictionary.")
print("")
print(d.values())
print("Returns a view object of all values in the dictionary.")
print("")
print(d.items())
print("Returns a view object of all key-value pairs in the dictionary.")
print("")
print(d.get('name'))
print("Returns the value associated with a specified key, or a default value if the key
print("")
```

```
print(d.pop('city'))
print(f"Removes and returns the value associated with a specified key.\nUpdated {d}")
print("")
d.update(prof)
print(f"Updates the dictionary with key-value pairs from another dictionary. \nUpdated 
```

```
Variable :{'name': 'John', 'age': 30, 'city': 'New York'}

dict_keys(['name', 'age', 'city'])
Returns a view object of all keys in the dictionary.

dict_values(['John', 30, 'New York'])
Returns a view object of all values in the dictionary.

dict_items([('name', 'John'), ('age', 30), ('city', 'New York')])
Returns a view object of all key-value pairs in the dictionary.

John
Returns the value associated with a specified key, or a default value if the key does n
ot exist.

New York
Removes and returns the value associated with a specified key.
Updated {'name': 'John', 'age': 30}

Updates the dictionary with key-value pairs from another dictionary.
Updated {'name': 'John', 'age': 30, 'profession': 'Engineer'}
```

Python Conditions Statements Examples

In [12]:

```
# Automated ticket vending machine
age = int(input("Enter the age :"))
has_student_card = int(input("Do you have student card type 1 or else 0 :"))
destination = input('type one "City Center" or "Suburb"')

ticket_price = 0

# Determine ticket price based on age and destination
if age < 18:
    ticket_price = 5
elif age >= 65:
    ticket_price = 7.5
elif has_student_card:
    ticket_price = 6
else:
    ticket_price = 10

# Apply discount for specific destinations
if destination == "City Center":
    ticket_price *= 0.8
elif destination == "Suburb":
    ticket_price *= 0.9

# Display the final ticket price
print(f"The ticket price for {destination} is Rs : {ticket_price:.2f}")
```

```
Enter the age :25
Do you have student card type 1 or else 0 :0
type one "City Center" or "Suburb"City Center
The ticket price for City Center is Rs : 8.00
```

While loop

In [13]:
```python
# Number guessing game
import random

secret_number = random.randint(1, 100)
guess = None
initial_attempts = 0
total_attempts = 5
print("Welcome to the Number Guessing Game!")

while guess != secret_number and initial_attempts != total_attempts:
    guess = int(input("Enter your guess (1-100): "))
    initial_attempts += 1
    if guess == secret_number:
        print(f"Congratulations! You guessed the correct number {secret_number} in {atte

    elif guess < secret_number:
        print("Too low! Try again.")

    elif guess > secret_number:
        print("Too high! Try again.")

print("You have exhausted your attempts.")
```

```
Welcome to the Number Guessing Game!
Enter your guess (1-100): 50
Too low! Try again.
Enter your guess (1-100): 60
Too high! Try again.
Enter your guess (1-100): 55
Too low! Try again.
Enter your guess (1-100): 56
Too low! Try again.
Enter your guess (1-100): 59
Too high! Try again.
You have exhausted your attempts.
```

For loop

In [14]:
```python
# Finding prime numbers in a given range
lower = 10
upper = 50

print(f"Prime numbers between {lower} and {upper} are: ")

for num in range(lower, upper + 1):
    if num > 1:
        for i in range(2, int(num/2) + 1):
            if (num % i) == 0:
                break
```

```
        else:
            print(num)
```

```
Prime numbers between 10 and 50 are:
11
13
17
19
23
29
31
37
41
43
47
```

Function in Python

In [15]:

```
# Data Analysis: Calculating Mean, Median, and Mode

#function definition
def calculate_mean(numbers):
    return sum(numbers) / len(numbers)

#function definition
def calculate_median(numbers):
    sorted_numbers = sorted(numbers)
    n = len(sorted_numbers)
    if n % 2 == 0:
        mid1 = sorted_numbers[n // 2]
        mid2 = sorted_numbers[n // 2 - 1]
        median = (mid1 + mid2) / 2
    else:
        median = sorted_numbers[n // 2]
    return median

#function definition
def calculate_mode(numbers):
    counts = {}
    for num in numbers:
        counts[num] = counts.get(num, 0) + 1
    max_count = max(counts.values())
    mode = [num for num, count in counts.items() if count == max_count]
    return mode

# driver code
data = [2, 4, 6, 6, 8, 10, 10, 10, 12]
mean = calculate_mean(data)
median = calculate_median(data)
mode = calculate_mode(data)

print("Data:", data)
print("Mean:", mean)
print("Median:", median)
print("Mode:", mode)
```

```
Data: [2, 4, 6, 6, 8, 10, 10, 10, 12]
Mean: 7.555555555555555
Median: 8
Mode: [10]
```

*args and **kwargs

In [16]:
```python
def send_email(self,subject,sender_email,password,body,*email,**kwargs):
        port = 587
        smtp_server = "smtp.gmail.com"
        msg['Subject'] = subject
        sender_email = sender_email
        password = password

        msg = MIMEMultipart()
        msg['From'] = sender_email
        msg['To'] = email
        msg['Cc'] = cc
        msg["Bcc"] = bcc

        body = body

        msg.attach(MIMEText(body, 'plain'))

        a.save(a.filename)
        attachment = open(file, "rb")
        filename = a.name

        p = MIMEBase('application', 'octet-stream')
        p.set_payload((attachment).read())
        p.add_header('Content-Disposition', "attachment; filename= %s" % filename)
        encoders.encode_base64(p)
        msg.attach(p)


        context = ssl.create_default_context()
        with smtplib.SMTP(smtp_server, port) as server:
            server.ehlo()
            server.starttls(context=context)
            server.ehlo()
            server.login(sender_email, password)
            server.sendmail(sender_email,email, msg.as_string())
            server.quit()
            time.sleep(3)

        logging.info("Mail Sent")

# driver code
# send_email("Important Announcement", "john@example.com", "qwertyuiop","Thank you","ab
# tried and tested
```

Lambda functions

Lambda functions, also known as anonymous functions, are a powerful tool in Python for creating small, one-time functions without explicitly defining them using the def keyword. They

are commonly used in scenarios where a function is required as an argument or when a small function needs to be defined inline.

In [17]:
```
employees = [
    {"name": "Alice", "age": 28, "salary": 50000},
    {"name": "Bob", "age": 32, "salary": 60000},
    {"name": "Charlie", "age": 24, "salary": 45000}
]

sorted_employees = sorted(employees, key=lambda emp: emp["salary"])
print(sorted_employees)
```

```
[{'name': 'Charlie', 'age': 24, 'salary': 45000}, {'name': 'Alice', 'age': 28, 'salar
y': 50000}, {'name': 'Bob', 'age': 32, 'salary': 60000}]
```

Decorator

A decorator in Python is a way to modify or enhance the functionality of a function or class without directly modifying its source code. It allows you to wrap a function or class with another function, commonly known as the decorator function, which can add additional behavior or modify the input/output of the original function.

In [18]:
```
# example 1: Authorization Decorator
# function definition
def check_authorization(username, userpassword):
    name = "RachitMore"
    password = "rachitmore"
    if username == name and userpassword == password:
        return True
    else:
        return False

def authorization_decorator(func):
    def wrapper(username, userpassword):
        try:
            if check_authorization(username, userpassword):
                return func(username, userpassword)
            else:
                raise PermissionError("Unauthorized access")
        except Exception as e:
            return e
    return wrapper

@authorization_decorator
def protected_function(username, userpassword):
    print("Access granted")

name = "RachitMore"
password = "rachitmore"
protected_function(name, password)
```

```
Access granted
```

In [19]:
```
# example 2: Logging Decorator
# function definition
def log_decorator(func):
```

```
    def wrapper(*args, **kwargs):
        print(f"Calling function: {func.__name__}")
        result = func(*args, **kwargs)
        print(f"Function {func.__name__} executed")
        return result
    return wrapper

@log_decorator
def add(a, b):
    return a + b

@log_decorator
def multiply(a, b):
    return a * b

print(add(2, 3))  # Output: 5
print(multiply(4, 5))  # Output: 20
```

```
Calling function: add
Function add executed
5
Calling function: multiply
Function multiply executed
20
```

In [20]:
```
# example 3: Memoization Decorator
# function definition
def memoize(func):
    cache = {}

    def wrapper(*args):
        if args in cache:
            return cache[args]
        result = func(*args)
        cache[args] = result
        return result

    return wrapper

@memoize
def fibonacci(n):
    if n <= 1:
        return n
    return fibonacci(n-1) + fibonacci(n-2)

print(fibonacci(10))  # Output: 55
```

```
55
```

In [21]:
```
# example 4: Timer Decorator
# function definition
import time

def timer_decorator(func):
    def wrapper(*args, **kwargs):
        start_time = time.time()
        result = func(*args, **kwargs)
        end_time = time.time()
        execution_time = end_time - start_time
        print(f"Function {func.__name__} executed in {execution_time} seconds")
```

```
        return result
    return wrapper

@timer_decorator
def long_running_function():
    time.sleep(3)
    print("Long running function executed")

long_running_function()  # Output: Function long_running_function executed in 3.0032861
```

```
Long running function executed
Function long_running_function executed in 3.0134758949279785 seconds
```

In [22]:
```
# example 5: Retry Decorator
# function definition
import random

def retry_decorator(max_attempts=3):
    def decorator(func):
        def wrapper(*args, **kwargs):
            attempts = 0
            while attempts < max_attempts:
                try:
                    result = func(*args, **kwargs)
                    return result
                except Exception as e:
                    attempts += 1
                    print(f"Exception occurred. Retrying ({attempts}/{max_attempts})...'
                    time.sleep(random.uniform(0.5, 1.0))
            raise Exception("Max retry attempts exceeded")
        return wrapper
    return decorator

@retry_decorator(max_attempts=5)
def unstable_function():
    if random.random() < 0.8:
        raise Exception("Unstable function failed")
    else:
        print("Unstable function succeeded")

unstable_function()  # Output: Retries up to 5 times if the function fails, and eventua
```

```
Unstable function succeeded
```

File Handling (Reading and Writing Files)

In [23]:
```
# Reading from a file using with statement
import csv
import json

# reading Input from the User
name = input("Enter your name: ")
print(f"Hello, {name}!")

# Writing and Reading txt Files
file = open("example.txt", "w")
file.write("Hello, World!")
file.close()
```

```
file = open("example.txt", "r")
content = file.read()
file.close()
print(content)


# Writing and Reading csv Files
data = [
    ['Name', 'Age', 'City'],
    ['John', '25', 'New York'],
    ['Alice', '32', 'London'],
    ['Bob', '28', 'Paris']
]

# writing csv file
with open('data.csv', 'w', newline='') as file:
    csv_writer = csv.writer(file)
    csv_writer.writerows(data)
    file.close()

# reading csv file
with open('data.csv', 'r') as file:
    csv_reader = csv.reader(file)
    for row in csv_reader:
        print(row)
    file.close()


# Writing and Reading JSON file
data = {
    'name': 'John',
    'age': 30,
    'city': 'New York'
}

# writing json file
with open('data.json', 'w') as file:
    json.dump(data, file)
    file.close()

# reading json file
with open('data.json', 'r') as file:
    data = json.load(file)
    file.close()
    print(data)
```

```
Enter your name: Rachit More
Hello, Rachit More!
Hello, World!
['Name', 'Age', 'City']
['John', '25', 'New York']
['Alice', '32', 'London']
['Bob', '28', 'Paris']
{'name': 'John', 'age': 30, 'city': 'New York'}
```

In [24]:

```
import pandas as pd
# Writing and Reading Exel file

df = pd.DataFrame({
```

```
    'Name': ['John', 'Alice', 'Bob'],
    'Age': [25, 32, 28]
})

# writing excel file
df.to_excel('data.xlsx', index=False)

# reading excel file
df = pd.read_excel('data.xlsx')
print(df)
```

```
    Name  Age
0   John   25
1  Alice   32
2    Bob   28
```

In [25]:

```
# 'pickle' module in Python provides functionality for serializing and deserializing
# python objects, allowing you to save and load complex data structures.
import pickle

data = {'name': 'John', 'age': 30}

# writing pickle file
with open('data.pickle', 'wb') as file:
    pickle.dump(data, file)

# reading pickle file
with open('data.pickle', 'rb') as file:
    data = pickle.load(file)
    print(data)
```

```
{'name': 'John', 'age': 30}
```

In [26]:

```
# Connecting to web page and web scraping
import requests
from bs4 import BeautifulSoup

# make a GET request to the web page
response = requests.get('https://www.google.com/')

# create a Beautiful Soup object with the response content
soup = BeautifulSoup(response.content, 'html.parser')

# find and print the title of the web page
title = soup.title
print("Page Title:", title.text)

# extracting all the text from a page
print(soup.get_text(),"printing soup ")

# find and print all the links on the web page
links = soup.find_all('a')
print("Links on the Page:")
for link in links:
    print(link.get('href'))
```

```
Page Title: Google
GoogleSearch Images Maps Play YouTube News Gmail Drive More »Web History | Settings | S
ign in Advanced searchGoogle offered in:  हिन्दी বাংলা తెలుగు मराठी தமிழ் ગુજરાતી ಕನ್ನಡ മല
യാളം ਪੰਜਾਬੀ AdvertisingBusiness SolutionsAbout GoogleGoogle.co.in© 2023 - Privacy - Ter
ms   printing soup
Links on the Page:
https://www.google.com/imghp?hl=en&tab=wi
https://maps.google.co.in/maps?hl=en&tab=wl
https://play.google.com/?hl=en&tab=w8
https://www.youtube.com/?tab=w1
https://news.google.com/?tab=wn
https://mail.google.com/mail/?tab=wm
https://drive.google.com/?tab=wo
https://www.google.co.in/intl/en/about/products?tab=wh
http://www.google.co.in/history/optout?hl=en
/preferences?hl=en
https://accounts.google.com/ServiceLogin?hl=en&passive=true&continue=https://www.googl
e.com/&ec=GAZAAQ
/advanced_search?hl=en-IN&authuser=0
https://www.google.com/setprefs?sig=0_cOaKaoCOg9KdHjE8IM6x-egRmhQ%3D&hl=hi&source=homep
age&sa=X&ved=0ahUKEwjFheSly___AhVrZvUHHfwEB-sQ2ZgBCAU
https://www.google.com/setprefs?sig=0_cOaKaoCOg9KdHjE8IM6x-egRmhQ%3D&hl=bn&source=homep
age&sa=X&ved=0ahUKEwjFheSly___AhVrZvUHHfwEB-sQ2ZgBCAY
https://www.google.com/setprefs?sig=0_cOaKaoCOg9KdHjE8IM6x-egRmhQ%3D&hl=te&source=homep
age&sa=X&ved=0ahUKEwjFheSly___AhVrZvUHHfwEB-sQ2ZgBCAc
https://www.google.com/setprefs?sig=0_cOaKaoCOg9KdHjE8IM6x-egRmhQ%3D&hl=mr&source=homep
age&sa=X&ved=0ahUKEwjFheSly___AhVrZvUHHfwEB-sQ2ZgBCAg
https://www.google.com/setprefs?sig=0_cOaKaoCOg9KdHjE8IM6x-egRmhQ%3D&hl=ta&source=homep
age&sa=X&ved=0ahUKEwjFheSly___AhVrZvUHHfwEB-sQ2ZgBCAk
https://www.google.com/setprefs?sig=0_cOaKaoCOg9KdHjE8IM6x-egRmhQ%3D&hl=gu&source=homep
age&sa=X&ved=0ahUKEwjFheSly___AhVrZvUHHfwEB-sQ2ZgBCAo
https://www.google.com/setprefs?sig=0_cOaKaoCOg9KdHjE8IM6x-egRmhQ%3D&hl=kn&source=homep
age&sa=X&ved=0ahUKEwjFheSly___AhVrZvUHHfwEB-sQ2ZgBCAs
https://www.google.com/setprefs?sig=0_cOaKaoCOg9KdHjE8IM6x-egRmhQ%3D&hl=ml&source=homep
age&sa=X&ved=0ahUKEwjFheSly___AhVrZvUHHfwEB-sQ2ZgBCAw
https://www.google.com/setprefs?sig=0_cOaKaoCOg9KdHjE8IM6x-egRmhQ%3D&hl=pa&source=homep
age&sa=X&ved=0ahUKEwjFheSly___AhVrZvUHHfwEB-sQ2ZgBCA0
/intl/en/ads/
http://www.google.co.in/services/
/intl/en/about.html
https://www.google.com/setprefdomain?prefdom=IN&prev=https://www.google.co.in/&sig=K_1p
tZr-oeO8qagbJJ0B3nRT6yC0k%3D
/intl/en/policies/privacy/
/intl/en/policies/terms/
```

In [27]:

```python
# Reading image files into binary format
with open('download.jfif', 'rb') as file:
    data = file.read()# Binary data

# Print the first 5 bytes of the image data
for i in range(5):
    byte = data[i]
    print(f"Byte {i}: {byte} (hex: {byte:02X}, binary: {bin(byte)[2:].zfill(8)})")

# creating a list of image array
image_array = []
for i in range(len(data)):
    image_array.append((data[i]))
```

```
# printing image data array
print(image_array)
```

```
Byte 0: 255 (hex: FF, binary: 11111111)
Byte 1: 216 (hex: D8, binary: 11011000)
Byte 2: 255 (hex: FF, binary: 11111111)
Byte 3: 224 (hex: E0, binary: 11100000)
Byte 4: 0 (hex: 00, binary: 00000000)
[255, 216, 255, 224, 0, 16, 74, 70, 73, 70, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 255, 219, 0,
132, 0, 10, 7, 8, 20, 20, 20, 24, 20, 20, 20, 24, 24, 24, 24, 26, 26, 26, 26, 24, 24, 2
4, 26, 27, 26, 24, 24, 24, 27, 27, 25, 24, 26, 27, 24, 27, 33, 45, 36, 27, 29, 42, 33,
24, 26, 37, 55, 37, 42, 46, 49, 52, 53, 52, 26, 35, 58, 63, 58, 51, 62, 45, 51, 52, 49,
1, 11, 11, 11, 16, 15, 16, 29, 18, 18, 31, 51, 42, 35, 42, 51, 51, 51, 51, 51, 51, 51,
51, 51, 51, 51, 51, 51, 51, 51, 49, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51,
51, 49, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 51, 25
5, 192, 0, 17, 8, 0, 154, 1, 72, 3, 1, 34, 0, 2, 17, 1, 3, 17, 1, 255, 196, 0, 27, 0,
0, 1, 5, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 1, 3, 4, 5, 6, 7, 255, 196, 0, 7
0, 16, 0, 2, 1, 3, 2, 4, 3, 4, 5, 8, 8, 4, 7, 0, 0, 0, 1, 2, 17, 0, 3, 33, 4, 18, 5, 4
9, 65, 81, 19, 34, 97, 50, 113, 129, 145, 6, 66, 82, 161, 209, 20, 35, 98, 114, 177, 17
8, 193, 225, 21, 51, 67, 83, 130, 146, 179, 240, 115, 147, 162, 210, 84, 99, 131, 148,
163, 211, 212, 255, 196, 0, 25, 1, 0, 3, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 2, 3, 4, 5, 255, 196, 0, 41, 17, 0, 2, 2, 3, 1, 0, 1, 2, 5, 5, 1, 0, 0, 0, 0, 0, 0,
1, 2, 17, 3, 18, 33, 49, 65, 4, 81, 19, 34, 50, 97, 113, 129, 145, 161, 209, 225, 20, 2
55, 218, 0, 12, 3, 1, 0, 2, 17, 3, 17, 0, 63, 0, 241, 154, 84, 244, 170, 132, 53, 61, 4
2, 84, 0, 169, 82, 165, 84, 2, 165, 79, 20, 241, 77, 32, 26, 149, 60, 82, 167, 64, 42,
113, 77, 78, 41, 160, 30, 136, 80, 138, 33, 86, 144, 4, 40, 133, 48, 20, 64, 86, 177, 1
37, 13, 142, 41, 197, 32, 40, 128, 173, 227, 18, 27, 16, 163, 81, 76, 5, 26, 138, 186,
4, 72, 130, 167, 65, 80, 165, 78, 180, 153, 162, 101, 155, 66, 173, 219, 21, 78, 217, 1
71, 118, 205, 97, 52, 116, 66, 69, 149, 20, 68, 80, 161, 162, 38, 179, 163, 77, 128, 10
6, 106, 144, 167, 150, 125, 98, 163, 53, 104, 134, 199, 90, 53, 106, 142, 156, 26, 100,
236, 90, 70, 171, 8, 106, 154, 26, 183, 104, 86, 114, 41, 72, 157, 90, 164, 86, 52, 41,
109, 137, 128, 9, 35, 152, 0, 152, 248, 81, 149, 131, 4, 65, 237, 202, 176, 116, 104, 1
64, 16, 115, 72, 189, 58, 219, 36, 18, 1, 129, 204, 253, 255, 0, 176, 77, 70, 212, 169,
6, 194, 107, 149, 25, 185, 64, 230, 162, 103, 171, 81, 33, 204, 180, 46, 84, 136, 245,
73, 94, 165, 87, 169, 113, 13, 139, 234, 244, 213, 93, 30, 149, 103, 161, 91, 158, 81,
74, 149, 61, 58, 57, 70, 165, 79, 79, 78, 128, 106, 122, 84, 169, 208, 8, 85, 222, 31,
163, 241, 95, 96, 49, 130, 121, 22, 38, 58, 42, 174, 88, 250, 14, 128, 158, 149, 81, 6
9, 88, 211, 92, 40, 202, 226, 37, 88, 48, 156, 137, 83, 34, 71, 81, 138, 218, 17, 177,
89, 38, 179, 65, 114, 217, 59, 212, 196, 144, 28, 101, 24, 143, 178, 252, 155, 225, 85,
74, 215, 72, 156, 96, 166, 210, 251, 212, 186, 238, 193, 55, 1, 18, 64, 82, 29, 133, 19
7, 7, 108, 226, 228, 25, 6, 32, 197, 75, 165, 225, 167, 86, 235, 111, 79, 106, 221, 24
7, 101, 98, 222, 19, 139, 14, 8, 102, 96, 21, 46, 66, 237, 11, 183, 59, 14, 100, 110, 2
29, 84, 215, 220, 44, 229, 10, 211, 84, 174, 40, 34, 165, 198, 152, 199, 2, 136, 10, 10
1, 163, 81, 85, 20, 75, 97, 1, 70, 171, 73, 5, 78, 137, 93, 112, 141, 25, 250, 70, 169,
70, 18, 167, 91, 117, 42, 218, 170, 43, 82, 160, 183, 68, 169, 93, 2, 112, 34, 160, 27,
183, 21, 15, 216, 130, 215, 7, 189, 6, 23, 220, 196, 31, 74, 181, 103, 232, 245, 183, 3
2, 45, 238, 124, 229, 8, 63, 8, 102, 17, 220, 146, 35, 61, 171, 55, 150, 43, 228, 106,
12, 230, 85, 42, 84, 21, 210, 107, 184, 110, 158, 216, 199, 155, 152, 95, 49, 14, 74, 1
56, 238, 57, 11, 131, 137, 65, 252, 107, 62, 248, 69, 219, 182, 208, 130, 178, 11, 57,
102, 35, 43, 157, 140, 0, 243, 41, 196, 3, 243, 164, 178, 39, 226, 30, 180, 82, 81, 86,
237, 138, 54, 42, 71, 245, 101, 15, 112, 91, 57, 206, 27, 211, 246, 85, 157, 61, 148, 5
6, 59, 135, 168, 143, 217, 252, 234, 101, 34, 209, 10, 81, 156, 85, 213, 225, 243, 236,
48, 39, 160, 62, 83, 247, 227, 228, 102, 163, 58, 118, 86, 150, 6, 103, 32, 224, 252, 1
07, 61, 147, 52, 233, 80, 208, 19, 86, 47, 137, 36, 242, 170, 229, 106, 145, 12, 64, 20
9, 3, 66, 22, 137, 86, 152, 137, 237, 86, 167, 14, 73, 108, 16, 8, 136, 39, 144, 37, 14
9, 102, 15, 190, 179, 45, 10, 213, 225, 158, 215, 249, 63, 212, 74, 199, 33, 162, 101,
141, 70, 174, 219, 32, 85, 66, 14, 55, 49, 51, 185, 132, 230, 35, 24, 32, 64, 229, 21,
5, 221, 83, 59, 110, 118, 44, 96, 9, 39, 48, 4, 1, 62, 128, 10, 173, 24, 161, 21, 146,
130, 67, 178, 250, 106, 4, 65, 4, 129, 217, 160, 17, 51, 4, 70, 115, 82, 162, 6, 182, 2
28, 184, 193, 128, 144, 103, 32, 182, 224, 121, 1, 229, 136, 245, 172, 244, 53, 123, 7
8, 60, 143, 240, 253, 215, 165, 37, 67, 178, 133, 202, 174, 198, 173, 92, 90, 174, 194,
180, 137, 44, 16, 213, 42, 181, 68, 5, 72, 180, 48, 36, 86, 165, 76, 41, 84, 208, 236,
243, 40, 167, 165, 79, 21, 20, 98, 53, 60, 82, 138, 32, 42, 146, 36, 104, 165, 20, 81,
```

```
8, 178, 165, 176, 123, 130, 93, 176, 225, 79, 125, 177, 202, 216, 63, 174, 236, 127, 11
5, 109, 94, 213, 105, 216, 177, 45, 102, 75, 29, 199, 195, 102, 230, 121, 224, 239, 12
9, 62, 157, 160, 199, 58, 223, 210, 49, 236, 171, 123, 183, 194, 252, 173, 170, 126, 21
8, 99, 196, 67, 123, 73, 49, 203, 42, 64, 255, 0, 152, 174, 126, 250, 27, 119, 255, 0,
69, 104, 150, 219, 47, 178, 187, 164, 46, 160, 144, 192, 8, 155, 36, 1, 130, 103, 217,
61, 189, 213, 145, 114, 183, 127, 164, 173, 236, 184, 25, 75, 59, 33, 84, 153, 5, 11, 1
6, 24, 231, 120, 43, 179, 112, 133, 41, 238, 237, 130, 230, 162, 79, 129, 242, 1, 167,
87, 32, 16, 14, 13, 9, 166, 174, 118, 49, 233, 230, 154, 149, 72, 15, 52, 166, 154, 14
9, 59, 1, 230, 149, 52, 210, 165, 96, 89, 6, 140, 26, 1, 68, 43, 189, 51, 52, 72, 13, 2
0, 212, 98, 158, 168, 100, 147, 76, 77, 13, 35, 64, 199, 38, 128, 154, 115, 66, 104, 1
6, 36, 211, 19, 73, 168, 77, 75, 98, 21, 53, 35, 77, 89, 49, 164, 53, 35, 74, 152, 210,
178, 198, 38, 154, 145, 165, 80, 202, 67, 26, 84, 169, 84, 148, 34, 105, 166, 154, 149,
33, 4, 40, 129, 161, 20, 244, 128, 48, 104, 247, 84, 66, 158, 132, 201, 100, 155, 169,
139, 208, 83, 26, 189, 216, 169, 5, 186, 155, 117, 13, 42, 55, 97, 65, 238, 165, 186, 1
30, 149, 61, 216, 245, 65, 110, 165, 186, 134, 149, 45, 152, 80, 123, 168, 73, 166, 16
5, 73, 201, 136, 99, 74, 145, 165, 80, 2, 165, 74, 149, 0, 42, 106, 122, 99, 72, 5, 74,
149, 42, 64, 127, 255, 217]
```

These pixel values represent the intensity of each pixel. In grayscale images, a pixel value of 0 represents black, and 255 represents white.where this array consist of number is stored as an 8-bit integer giving a range of possible values from 0 to 255. Values in between make up the different shades of gray.

Exception Handling

In [28]:
```python
# example 1
# function definition
def divide_numbers(a, b):
    try:
        result = a / b
        print(f"The result of {a} divided by {b} is: {result}")
    except ZeroDivisionError:
        print("Error: Cannot divide by zero.")
    except TypeError:
        print("Error: Invalid operand types.")
    except Exception as e:
        print("An error occurred:", str(e))

    finally:
        print("Function Executed")

# driver code
divide_numbers(10, 2)  # Valid division
divide_numbers(10, 0)  # Division by zero error
divide_numbers(10, "2")  # Type error
divide_numbers(10, [])  # Other exceptions
```

```
The result of 10 divided by 2 is: 5.0
Function Executed
Error: Cannot divide by zero.
Function Executed
Error: Invalid operand types.
Function Executed
Error: Invalid operand types.
Function Executed
```

In [29]:
```python
# example 2
# function definition
def file_load(filename:object)-> str:
    """Returns multilines string."""

    try:
        file = open(filename, "r")
        content = file.read()
        file.close()

        return content

    except FileNotFoundError:
        print("File not found.")
    except PermissionError:
        print("Permission denied.")
    except Exception as e:
        print("An error occurred:", str(e))

filename = "nonexistent.txt"
file_load(filename)
```

```
File not found.
```

Object-Oriented Programming

Object-Oriented Programming (OOP) is a programming paradigm that organizes code into objects, which are instances of classes. It focuses on encapsulating data and behavior into reusable and modular components.

- Class: A class is a blueprint or template that defines the structure and behavior of objects. It encapsulates data (attributes) and functions (methods) that operate on that data. It serves as a blueprint for creating instances of objects.
- Object: An object is an instance of a class. It represents a specific entity or concept. Objects have their own state (values of attributes) and behavior (methods).

In [30]:
```python
# class definition
class Car:
    def __init__(self, make, model):
        self.make = make
        self.model = model

    def start_engine(self):
        print(f"The {self.make} {self.model} engine is starting.")

# Creating objects
car1 = Car("Tata", "Nexon")
car2 = Car("Honda", "Amaze")

# Accessing attributes
print(car1.make)  # Output: Toyota
print(car2.model)  # Output: Accord

# Calling methods
```

```
car1.start_engine()  # Output: The Toyota Camry engine is starting.
car2.start_engine()  # Output: The Honda Accord engine is starting.
```

```
Tata
Amaze
The Tata Nexon engine is starting.
The Honda Amaze engine is starting.
```

Encapsulation

- Encapsulation: Encapsulation is the concept of bundling data and methods together within a class. It provides the mechanism to hide internal implementation details and expose only relevant information to the outside world. It helps in achieving data integrity and code organization.

In [31]:
```
# class definition
class BankAccount:
    def __init__(self, account_number, balance):
        self.__account_number = account_number  # Private attribute
        self.__balance = balance  # Private attribute

    def get_account_number(self):
        return self.__account_number

    def get_balance(self):
        return self.__balance

    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount
        else:
            print("Invalid deposit amount.")

    def withdraw(self, amount):
        if amount > 0 and amount <= self.__balance:
            self.__balance -= amount
        else:
            print("Invalid withdrawal amount or insufficient balance.")


# driver code
account = BankAccount("123456789", 10000)

print("Account Number:", account.get_account_number())
print("Balance:", account.get_balance())

account.deposit(5000)
print("Balance after deposit:", account.get_balance())

account.withdraw(2000)
print("Balance after withdrawal:", account.get_balance())
```

```
Account Number: 123456789
Balance: 10000
Balance after deposit: 15000
Balance after withdrawal: 13000
```

Inheritance

- Inheritance: Inheritance is the process by which one class acquires the properties (attributes and methods) of another class. It allows the creation of a hierarchy of classes, with each derived class inheriting and extending the characteristics of its parent class. Inheritance promotes code reuse and supports the "is-a" relationship between classes.

In [32]:
```
# # class definition
class University:
    def __init__(self, universityName, universityLocation):
        self.universityName = universityName
        self.universityLocation = universityLocation

    def University_info(self):
        print(f"Name of university is {self.universityName} and it is located at {self.

# class definition
class College(University):
    def __init__(self, collegeName, collegeLocation, universityName, universityLocation
        super().__init__(universityName, universityLocation)
        self.collegeName = collegeName
        self.collegeLocation = collegeLocation

    def College_info(self):
        print(f"""Name of College is {self.collegeName} and located at {self.collegeLoca
              and affiliated by {self.universityName} ({self.universityLocation})""")
# class definition
class Student(College):
    def __init__(self, name, age, branch, collegeName, collegeLocation, universityName,
        super().__init__(collegeName, collegeLocation, universityName, universityLocati
        self.name = name
        self.age = age
        self.branch = branch

    def info(self):
        print(f"""Name of Student is {self.name} and age is {self.age} and branch is {s
Study in {self.collegeName} college located at {self.collegeLocation}
and affiliated by {self.universityName} ({self.universityLocation})""")


# driver code
obj = Student(name="Rachit More", age=25, branch="Mechanical Engineeering", collegeName=

#finding the information of student
obj.info()
print("")

#finding the information of college
obj.College_info()
print("")

#finding the information of university
obj.University_info()
```

```
Name of Student is Rachit More and age is 25 and branch is Mechanical Engineeering
Study in XYZ college located at Bhopal
and affiliated by RGPV (Bhopal)

Name of College is XYZ and located at Bhopal
             and affiliated by RGPV (Bhopal)

Name of university is RGPV and it is located at Bhopal
```

Polymorphism

- Polymorphism: Polymorphism allows objects of different classes to be treated as interchangeable entities, as long as they share a common interface or base class. It allows methods to be overridden in subclasses, enabling different implementations for the same method signature. Polymorphism promotes code flexibility and extensibility.

In [33]:
```python
# # class definition
class SortingAlgorithm:
    def sort(self, data):
        raise NotImplementedError("Subclass must implement sort() method")

# Creating Bubble sorting algorithm class
class BubbleSort(SortingAlgorithm):
    def sort(self, arr):
        n = len(arr)

        # Traverse through all array elements
        for i in range(n):
            # Last i elements are already in place
            for j in range(0, n-i-1):
                # Swap adjacent elements if they are in the wrong order
                if arr[j] > arr[j+1]:
                    arr[j], arr[j+1] = arr[j+1], arr[j]

        return  arr

    def info(self):
        return "BubbleSort:"


# Creating Quick sorting algorithm class
class QuickSort(SortingAlgorithm):

    def sort(self, arr):
        if len(arr) <= 1:
            return arr

        else:
            pivot = arr[0]
            smaller = [x for x in arr[1:] if x <= pivot]
            greater = [x for x in arr[1:] if x > pivot]

            return self.sort(smaller) + [pivot] + self.sort(greater)

    def info(self):
        return "QuickSort:"
```

```
# Creating merge sorting algorithm class
class MergeSort(SortingAlgorithm):
    def sort(self, arr):
        if len(arr) <= 1:
            return arr

        mid = len(arr) // 2
        left_half = arr[:mid]
        right_half = arr[mid:]

        left_half = self.sort(left_half)
        right_half = self.sort(right_half)

        return self.merge(left_half, right_half)

    def merge(self, left, right):
        result = []
        left_idx, right_idx = 0, 0

        while left_idx < len(left) and right_idx < len(right):
            if left[left_idx] < right[right_idx]:
                result.append(left[left_idx])
                left_idx += 1
            else:
                result.append(right[right_idx])
                right_idx += 1

        result.extend(left[left_idx:])
        result.extend(right[right_idx:])

        return result

    def info(self):
        return "MergeSort:"


# driver code
data = [7, 2, 5, 1, 9, 3]

sorting_algorithms = [BubbleSort(), QuickSort(), MergeSort()]

for algorithm in sorting_algorithms:
    print(algorithm.info())
    print(algorithm.sort(data))
```

```
BubbleSort:
[1, 2, 3, 5, 7, 9]
QuickSort:
[1, 2, 3, 5, 7, 9]
MergeSort:
[1, 2, 3, 5, 7, 9]
```

Abstraction

- Abstraction: Abstraction is the process of simplifying complex systems by focusing on the essential features while hiding unnecessary details. In OOP, abstraction is achieved through abstract classes and interfaces, which provide a way to define common behaviors without

specifying their exact implementation. Abstract classes cannot be instantiated, and they serve as a base for concrete classes.

In [34]:
```python
from abc import ABC, abstractmethod
# Payment Gateway
# function definition
class PaymentGateway(ABC):
    @abstractmethod
    def process_payment(self, amount):
        pass

    @abstractmethod
    def refund_payment(self, transaction_id):
        pass

class StripeGateway(PaymentGateway):
    def process_payment(self, amount):
        print(f"Processing payment via Stripe: Rs: {amount}")

    def refund_payment(self, transaction_id):
        print(f"Refunding payment via Stripe: {transaction_id}")

class PayPalGateway(PaymentGateway):
    def process_payment(self, amount):
        print(f"Processing payment via PayPal: Rs: {amount}")

    def refund_payment(self, transaction_id):
        print(f"Refunding payment via PayPal: {transaction_id}")

# driver code:
stripe_gateway = StripeGateway()
stripe_gateway.process_payment(100)
stripe_gateway.refund_payment("T12345")

paypal_gateway = PayPalGateway()
paypal_gateway.process_payment(200)
paypal_gateway.refund_payment("T56787")
```

```
Processing payment via Stripe: Rs: 100
Refunding payment via Stripe: T12345
Processing payment via PayPal: Rs: 200
Refunding payment via PayPal: T56787
```

Association

- Association: Association represents a relationship between two or more classes. It describes how objects from different classes interact with each other. Associations can be one-to-one, one-to-many, or many-to-many, and they are often implemented through attributes or method parameters.

In [35]:
```python
# class definition
class School:
    def __init__(self, name):
        self.name = name
        self.students = []
```

```
    def enroll_student(self, student):
        self.students.append(student)

# class definition
class Student:
    def __init__(self, name, grade):
        self.name = name
        self.grade = grade

# driver code
# Create school object
school = School("ABC School")

# Create student objects
student1 = Student("John", 5)
student2 = Student("Alice", 6)
student3 = Student("Bob", 5)

# Enroll students in the school
school.enroll_student(student1)
school.enroll_student(student2)
school.enroll_student(student3)

# Print the school name and enrolled students
print(f"School: {school.name}")
print("Enrolled Students:")
for student in school.students:
    print(f"Name: {student.name}, Grade: {student.grade}")
```

```
School: ABC School
Enrolled Students:
Name: John, Grade: 5
Name: Alice, Grade: 6
Name: Bob, Grade: 5
```

Composition

- Composition: Composition is a strong form of association where one class contains objects of another class as part of its own structure. The composed objects cannot exist independently, and their lifecycle is tied to the lifecycle of the containing class. Composition allows the creation of complex objects by combining simpler objects.

In [36]:

```
# class definition
class Student:
    def __init__(self, name):
        self.name = name

    def study(self):
        print(f"{self.name} is studying.")

# class definition
class School:
    def __init__(self, name):
        self.name = name
        self.students = []

    def enroll_student(self, student):
```

```
        self.students.append(student)

    def conduct_class(self):
        print(f"Class is in session at {self.name}.")
        for student in self.students:
            student.study()

# driver code
# Create student objects
student1 = Student("Alice")
student2 = Student("Bob")
student3 = Student("Charlie")

# Create school object
school = School("ABC School")

# Enroll students in the school
school.enroll_student(student1)
school.enroll_student(student2)
school.enroll_student(student3)

# Conduct a class in the school
school.conduct_class()
```

```
Class is in session at ABC School.
Alice is studying.
Bob is studying.
Charlie is studying.
```

Aggregation

- Aggregation: Aggregation is a form of association where one class represents a part of the other class, but the part can exist independently. The aggregated objects have their own lifecycle and can exist outside the scope of the containing class. Aggregation represents a "has-a" relationship between classes.

In [37]:
```
# class definition
class Department:
    def __init__(self, name):
        self.name = name

    def get_department_name(self):
        return self.name

# class definition
class University:
    def __init__(self, name):
        self.name = name
        self.departments = []

    def add_department(self, department):
        self.departments.append(department)

    def get_departments(self):
        return [department.get_department_name() for department in self.departments]

# driver code
```

```
# Create department objects
dept1 = Department("Computer Science")
dept2 = Department("Electrical Engineering")

# Create university object
university = University("ABC University")

# Add departments to the university
university.add_department(dept1)
university.add_department(dept2)

# Get the departments of the university
departments = university.get_departments()
print(f"Departments of {university.name}: {', '.join(departments)}")
```

```
Departments of ABC University: Computer Science, Electrical Engineering
```

Decorator in class

In [38]:

```
# decorator definition
def debug_methods(cls):
    for name, value in vars(cls).items():
        if callable(value):
            setattr(cls, name, debug_method(value))
    return cls

def debug_method(func):
    def wrapper(*args, **kwargs):
        print(f"Calling method: {func.__name__}")
        result = func(*args, **kwargs)
        print(f"Method {func.__name__} executed")
        return result
    return wrapper

@debug_methods
class Calculator:
    def add(self, a, b):
        return a + b

    def subtract(self, a, b):
        return a - b

# driver code
calc = Calculator()
print(calc.add(2, 3))  # Output: Calling method: add, Method add executed, 5
```

```
Calling method: add
Method add executed
5
```

Static methods

Static methods are defined using the @staticmethod decorator, which indicates that the following method should be treated as a static method. The decorator is placed above the method definition.

Static methods do not have access to the instance-specific data or the instance itself. They are self-contained and can be used without any instance-related context. They are useful for grouping related utility functions within a class or when a method doesn't require access to instance-specific data.

In [39]:
```
import logging
# class definition
class Logger:
    FORMAT = "%(asctime)s - %(levelname)s - %(message)s"

    @staticmethod
    def log_info(message):
        logging.basicConfig(format=Logger.FORMAT, level=logging.INFO)
        logging.info(message)

    @staticmethod
    def log_error(message):
        logging.basicConfig(format=Logger.FORMAT, level=logging.ERROR)
        logging.error(message)

# driver code
Logger.log_info("This is an informational message")
Logger.log_error("This is an error message")
```

```
2023-07-08 22:31:53,225 - INFO - This is an informational message
2023-07-08 22:31:53,227 - ERROR - This is an error message
```

Dunder methods

Dunder methods are special methods in Python that allow classes to define their behavior for built-in operations or syntax. They are called "magic methods" because they add special behavior to the objects of a class, making them behave like built-in types or supporting specific operations.

In [40]:
```
# class definition
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):
        return f"({self.x}, {self.y})"

    def __add__(self, other):
        if isinstance(other, Point):
            return Point(self.x + other.x, self.y + other.y)
        else:
            raise TypeError("Unsupported operand type: Point and {}".format(type(other)

    def __eq__(self, other):
        if isinstance(other, Point):
            return self.x == other.x and self.y == other.y
        else:
            return False
```

```
# driver code
# Create two Point objects
p1 = Point(1, 2)
p2 = Point(3, 4)

# Test the __str__ method
print(p1)  # Output: (1, 2)

# Test the __add__ method
p3 = p1 + p2
print(p3)  # Output: (4, 6)

# Test the __eq__ method
print(p1 == p2)  # Output: False
print(p1 == Point(1, 2))  # Output: True
```

```
(1, 2)
(4, 6)
False
True
```

Generators and Iterators

In [41]:

```
# Iterators
# function definition
class NumberIterator:
    def __init__(self, limit):
        self.limit = limit
        self.current = 0

    def __iter__(self):
        return self

    def __next__(self):
        if self.current < self.limit:
            number = self.current
            self.current += 1
            return number
        else:
            raise StopIteration

# driver code
# Using the custom iterator
iterator = NumberIterator(5)
for num in iterator:
    print(num)
print("")

# Generators
# function definition
def fibonacci_generator():
    a, b = 0, 1
    while True:
        yield a
        a, b = b, a + b

# driver code
# Using the generator
fib_gen = fibonacci_generator()
```

```
for _ in range(5):
    print(next(fib_gen))
```

```
0
1
2
3
4

0
1
1
2
3
```

Regular expression

A regular expression (regex) is a sequence of characters that defines a search pattern. It is a powerful tool used for pattern matching, searching, and manipulating text. Regular expressions are supported in various programming languages and text editors.

Implementation of OOPS and Regular Expression

In [42]:
```
import re
# class definition
class Validate:
    def __init__(self, username, email, phone, date):
        self.username = username
        self.email = email
        self.phone = phone
        self.date = date

    def info(self):
        return self.username,self.email,self.phone,self.date

    # Validating email addresses
    def validate_email(self):
        pattern = r"^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$"
        result = re.match(pattern, self.email)
        if result:
            return True
        else:
            print(self.email,"Invalid email")
            print(self.info())
            return False

    # Extracting phone numbers
    def validate_phone_numbers(self):
        pattern = r"\d{3}-\d{3}-\d{4}"
        result = re.findall(pattern, self.phone)
        if result:
            return True
        else:
            print(self.phone,"Invalid phone")
            print(self.info())
            return False
```

```
    # Data validation
    def validate_username(self):
        pattern = r"^[a-zA-Z0-9_ ]+$"
        result = re.match(pattern, self.username)
        if result:
            return True
        else:
            print(self.username,"Invalid username")
            print(self.info())
            return False


    # Data extraction
    def validate_dates(self):
        pattern = r"\d{1,2}[/-]\d{1,2}[/-]\d{2,4}"
        result = re.findall(pattern, self.date)
        if result:
            return True
        else:
            print(self.date,"Invalid dates")
            print(self.info())
            return False

    # Data extraction
    def data_validate(self):
        if self.validate_username() and self.validate_email() and self.validate_phone_nu
            return True
        else:
            print("Something Went Worng")
            return False
```

In [43]:

```
# class definition
class Student:
    def __init__(self, name, roll_number, email, enrolled_date, phone_number):
        self.name = name
        self.roll_number = roll_number
        self.email = email
        self.enrolled_date = enrolled_date
        self.phone_number = phone_number


    def display(self):
        print(f"Name: {self.name}")
        print(f"Roll Number: {self.roll_number}")
        print(f"Email: {self.email}")
        print(f"Enrolled Date: {self.enrolled_date}")
        print(f"Phone Number: {self.phone_number}")
        print()

# class definition
class StudentManagementSystem(Validate):
    def __init__(self):
        self.students = []
      # Initialize the parent class with empty values

    def add_student(self, name, roll_number, email, enrolled_date, phone_number):
        super().__init__(name, email, phone_number, enrolled_date)# initializing the su
        if super().data_validate():# Checking Validation of data
```

```
            student = Student(name, roll_number, email, enrolled_date, phone_number)
            self.students.append(student)

        else:
            print("Please enter valid details.\n")

    def display_all_students(self):
        for student in self.students:
            student.display()

    def search_student_by_roll_number(self, roll_number):
        for student in self.students:
            if student.roll_number == roll_number:
                student.display()
                return
        print("Student not found.")

    def search_students_by_name(self, name):
        matching_students = []
        for student in self.students:
            if re.search(name, student.name, re.IGNORECASE):
                matching_students.append(student)

        if matching_students:
            for student in matching_students:
                student.display()
        else:
            print("No students found with the given name.")

    def remove_student(self, roll_number):
        for student in self.students:
            if student.roll_number == roll_number:
                self.students.remove(student)
                print(f"Student with roll number {roll_number} removed.")
                return
        print("Student not found.")

# driver code:
# Create the student management system
sms = StudentManagementSystem()

# Add students
sms.add_student("rachit", "1001", "rachitmore@gmail.com", "08/07/2023", "123-456-7890")
sms.add_student("Sunny", "1002", "sunny@example.com", "10/02/2023", "123-456-0789")
sms.add_student("krish", "1003", "krish@example.com", "15/06/2023", "098-765-432")

# Display all students
print("All Students:")
sms.display_all_students()

# Search for a student by roll number
print("Search by Roll Number:")
sms.search_student_by_roll_number("1002")

# Search for students by name
print("Search by Name:")
sms.search_students_by_name("b")

# Remove a student
print("Remove Student:")
```

```
sms.remove_student("1003")

# Display all students after removal
print("All Students after removal:")
sms.display_all_students()
```

```
098-765-432 Invalid phone
('krish', 'krish@example.com', '098-765-432', '15/06/2023')
Something Went Worng
Please enter valid details.

All Students:
Name: rachit
Roll Number: 1001
Email: rachitmore@gmail.com
Enrolled Date: 08/07/2023
Phone Number: 123-456-7890

Name: Sunny
Roll Number: 1002
Email: sunny@example.com
Enrolled Date: 10/02/2023
Phone Number: 123-456-0789

Search by Roll Number:
Name: Sunny
Roll Number: 1002
Email: sunny@example.com
Enrolled Date: 10/02/2023
Phone Number: 123-456-0789

Search by Name:
No students found with the given name.
Remove Student:
Student not found.
All Students after removal:
Name: rachit
Roll Number: 1001
Email: rachitmore@gmail.com
Enrolled Date: 08/07/2023
Phone Number: 123-456-7890

Name: Sunny
Roll Number: 1002
Email: sunny@example.com
Enrolled Date: 10/02/2023
Phone Number: 123-456-0789
```

In [ ]: