In [1]:
```
1 l = list([1,2,3,4])
```

In [2]:
```
1 l.upper()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[2], line 1
----> 1 l.upper()

AttributeError: 'list' object has no attribute 'upper'
```

In [ ]:
```
1 print(type(l))
```

In [ ]:
```
1 string = 'str'
2 print(type(string))
```

In [ ]:
```
1 string.append()
```

Object Oriented Programming :

- Object-Oriented Programming is a way of organizing and designing code in Python (and many other programming languages) that mimics how we think about and interact with objects in the real world.
- In OOP, we create "objects" that represent real-world entities, and these objects have "attributes" (characteristics) and "methods" (actions) that define their behavior.
- It brings structure, organization, and reusability to our code, making it more efficient and easier to manage.
- OOP is a fundamental concept in modern programming and widely used in creating complex applications and software systems.

1. Class:

- In OOP, a class is like a blueprint that defines the structure of an object.
- A class describes what a specific type of object can do and what attributes it has.
- For example, the DOG class will describe what a Dog is and what attributes and methods it should have.

In [ ]:
```python
class Dog:
    def __init__(self, name, age, breed):
        self.name = name
        self.age = age
        self.breed = breed

    def bark(self):
        return "Woof! Woof!"

    def fetch(self, item):
        return f"{self.name} fetches the {item}."

    def greet(self,name):
        return f"{self.name} greets {name}."

```

2. Object:

- An object is a specific instance created from a class.
- In our Dog example, each individual dog we create will be an object.
- Each dog object will have its own unique name, age, and breed.

In [ ]:
```python
dog1 = Dog("Buddy", 3, "Labrador")
dog2 = Dog("Max", 5, "Golden Retriever")
```

In [ ]:
```python
dog2.bark()
```

In [ ]:
```python
dog1.bark()
```

In [ ]:
```python
dog1.greet('abc')
```

In [ ]:
```python
dog2.greet('xyz')
```

3. Attributes:

- Attributes are characteristics of an object.
- In our Dog example, the attributes are the name, age, and breed of each dog.
- They help describe the dog's identity.

In [ ]:
```python
print(dog1.name)  # Output: "Buddy"
print(dog2.age)   # Output: 5

```

4. Methods:

- Methods are actions that an object can perform.
- In our Dog example, methods could be actions like barking or fetching an item.
- Methods define what the dog can do.

In [ ]:
```
print(dog1.bark())  # Output: "Woof! Woof!"
print(dog2.fetch("ball"))  # Output: "Max fetches the ball."
```

Advantages of OOP:

OOP provides several benefits, including:

1. `Code Reusability:` You can create multiple objects from a single class, promoting code reuse and reducing duplication.
2. `Modularity:` OOP allows you to divide your code into small, manageable pieces (objects), making it easier to maintain and understand.
3. `Encapsulation:` Encapsulation hides the internal details of an object, making it easier to use and less prone to errors.
4. `Inheritance:` Inheritance allows you to create new classes that inherit attributes and methods from existing classes, promoting code reuse and extending functionality.
5. `Polymorphism:` Polymorphism allows you to use objects of different classes interchangeably, providing flexibility in coding.

In [ ]:
```
str.append()
```

In [ ]:
```
list.upper()
```

In [ ]:
```
class Account:
    def __init__(self,a_number, c_name, balance):
        self.accoun_number = a_number
        self.customer_name = c_name
        self.balance = balance

    def deposit(self, amount : float) -> float:
        self.balance += amount
        print("Successfully Deposited")

    def withdraw(self, amount):
        if amount < self.balance:
            self.balance -= amount
            print(f"Successfully withdrawn {amount}")
        else:
            print("Gareeb")

    def get_balance(self):
        return self.balance

    def display_info(self):
        print(f"Your account number is : {self.accoun_number}")
        print(f"Your balance is : {self.balance}")
        print(f"YOur name is : {self.customer_name}")
```

In [ ]:
```
c1 = Account('122334455', 'Justin', 2000)
```

In [ ]:
```
c1.deposit(500)
```

In [ ]:
```
c1.balance
```

In [ ]:
```
c1.withdraw(600)
```

In [ ]:
```
c1.balance
```

In [ ]:
```
c1.display_info()
```

In [ ]:
```
c2 = Account('234234','Dustin',5000)
```

In [ ]:
```
c2.display_info()
```

In [ ]:
```
c2.deposit(5000)
```

In [ ]:
```
c2.withdraw(3000)
```

In [ ]:
```
c1.deposit(50000)
```

In [ ]:
```
c1.display_info()
```

In [ ]:
```
c2.display_info()
```

In [ ]:
```
class Account:
    def __init__(self, account_number, customer_name, balance):
        self.account_number = account_number
        self.customer_name = customer_name
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        if self.balance >= amount:
            self.balance -= amount
        else:
            print("Insufficient balance!")

    def get_balance(self):
        return self.balance

    def display_info(self):
        print(f"Account Number: {self.account_number}")
        print(f"Customer Name: {self.customer_name}")
        print(f"Balance: {self.balance}")
```

In [ ]:

```python
# Create a new account
account1 = Account("A001", "Alice", 1000)

# Deposit and withdraw from the account
account1.deposit(500)
account1.withdraw(200)

# Display the account information
account1.display_info()
```

In [ ]:

```python

```

In [ ]:

```python
id(c1)
```

In [2]:

```
class Atm:
    def __init__(self):
        self.pin = ''
        self.balance = 0
        self.menu()

    def menu(self):
        user_input = input("""
        Hi how can I help you?
        1. Press 1 to create pin
        2. Press 2 to change pin
        3. Press 3 to check balance
        4. Press 4 to withdraw
        5. Anything else to exit
        """)

        if user_input == '1':
            self.create_pin()
        elif user_input == '2':
            self.change_pin()
        elif user_input == '3':
            self.check_balance()
        elif user_input == '4':
            self.withdraw()
        else:
            exit()


    def create_pin(self):
        self.pin =  input('Enter your pin: ')

        user_balance = int(input('Enter balance: '))
        self.balance = user_balance

        print('Pin created successfully!')
        self.menu()


    def change_pin(self):
        old_pin = input('Enter old pin: ')

        if old_pin == self.pin:
            new_pin = input('Enter new pin: ')
            self.pin = new_pin
            print('Pin change successful!')
            self.menu()
        else:
            print('Cannot change pin! Incorrect old pin.')
            self.menu()


    def check_balance(self):
        old_pin = input('Enter old pin: ')

        if old_pin == self.pin:
            print(f"YOur balance is {self.balance}")
            self.menu()
        else :
            print("CHor chor")
            self.menu()


    def withdraw(self):
        old_pin = input('Enter old pin: ')

```

```
        if old_pin == self.pin:
            amount_to_withdraw = int(input("Enter the amount : "))

            if amount_to_withdraw < self.balance :
                self.balance -= amount_to_withdraw
                print(f"Amount withdrawn successfully, balance is {self.balance}")
                self.menu()
            else :
                print("Paise nahi hai tere pass itne")
                self.menu()
        else :
            print("CHor chor")
            self.menu()
```

In [3]:
```
badlapur_atm  = Atm()
```

```
        Hi how can I help you?
        1. Press 1 to create pin
        2. Press 2 to change pin
        3. Press 3 to check balance
        4. Press 4 to withdraw
        5. Anything else to exit
        1
Enter your pin: 8899
Enter balance: 60000
Pin created successfully!

        Hi how can I help you?
        1. Press 1 to create pin
        2. Press 2 to change pin
        3. Press 3 to check balance
        4. Press 4 to withdraw
        5. Anything else to exit
        3
Enter old pin: 8899
YOur balance is 60000

        Hi how can I help you?
        1. Press 1 to create pin
        2. Press 2 to change pin
        3. Press 3 to check balance
        4. Press 4 to withdraw
        5. Anything else to exit
        4
Enter old pin: 8899
Enter the amount : 70000
Paise nahi hai tere pass itne

        Hi how can I help you?
        1. Press 1 to create pin
        2. Press 2 to change pin
        3. Press 3 to check balance
        4. Press 4 to withdraw
        5. Anything else to exit
        4
Enter old pin: 8899
Enter the amount : 5000
Amount withdrawn successfully, balance is 55000

        Hi how can I help you?
        1. Press 1 to create pin
        2. Press 2 to change pin
        3. Press 3 to check balance
        4. Press 4 to withdraw
        5. Anything else to exit
        q
```

In [3]:

```python
class Atm:
    def __init__(self):
        print(id(self))
        self.pin = ''
        self.balance = 0
        self.menu()

    def menu(self):
        user_input = input("""
        Hi how can I help you?
        1. Press 1 to create pin
        2. Press 2 to change pin
        3. Press 3 to check balance
        4. Press 4 to withdraw
        5. Anything else to exit
        """)

        if user_input == '1':
            self.create_pin()
        elif user_input == '2':
            self.change_pin()
        elif user_input == '3':
            self.check_balance()
        elif user_input == '4':
            self.withdraw()
        else:
            exit()

    def create_pin(self):
        user_pin = input('Enter your pin: ')
        self.pin = user_pin

        user_balance = int(input('Enter balance: '))
        self.balance = user_balance

        print('Pin created successfully!')
        self.menu()

    def change_pin(self):
        old_pin = input('Enter old pin: ')

        if old_pin == self.pin:
            new_pin = input('Enter new pin: ')
            self.pin = new_pin
            print('Pin change successful!')
            self.menu()
        else:
            print('Cannot change pin! Incorrect old pin.')
            self.menu()

    def check_balance(self):
        user_pin = input('Enter your pin: ')
        if user_pin == self.pin:
            print('Your balance is', self.balance)
        else:
            print('Incorrect pin!')
        self.menu()

    def withdraw(self):
        user_pin = input('Enter your pin: ')
        if user_pin == self.pin:
            amount = int(input('Enter the amount: '))
            if amount <= self.balance:
                self.balance -= amount
                print('Withdrawal successful. Balance is', self.balance)
```

```
            else:
                print('Insufficient balance!')
        else:
            print('Incorrect pin!')
        self.menu()
```

In [4]:
```
# Create an ATM object and start the banking operations
atm = Atm()
```

```
2256885205456

        Hi how can I help you?
        1. Press 1 to create pin
        2. Press 2 to change pin
        3. Press 3 to check balance
        4. Press 4 to withdraw
        5. Anything else to exit
        h
```

In [ ]:
```

```

In [ ]:
```
atm2 = Atm()
```

In [ ]:
```
atm2.withdraw()
```

In [ ]:
```

```

In [ ]:
```
class Fraction:
    def __init__(self, x,y):
        self.num = x
        self.den = y

    def __str__(self):
        return f"{self.num}/{self.den}"

    def __add__(self, other):
        new_num = self.num * other.den + other.num * self.num
        new_den = self.den * other.den
        return f"{new_num}/{new_den}"

    def __sub__(self, other):
        new_num = self.num * other.den - other.num * self.num
        new_den = self.den * other.den
        return f"{new_num}/{new_den}"

    def __mul__(self, other):
        new_num = self.num * other.num
        new_den = self.den * other.den
        return f"{new_num}/{new_den}"

    def __truediv__(self, other):
        new_num = self.num * other.den
        new_den = self.den * other.num
        return f"{new_num}/{new_den}"

    def convert_to_decimal(self):
        return self.num / self.den
```

In [ ]:
```
obj1 = Fraction(1,4)
print(obj1)
```

In [ ]:
```
obj2 = Fraction(4,6)
print(obj2)
```

In [ ]:
```
print(obj1+obj2)
print(obj1-obj2)
print(obj1*obj2)
print(obj1/obj2)
```

In [ ]:
```
obj1.convert_to_decimal()
```

In [ ]:
```

```

Q. Write OOP classes to handle the following scenarios:

- A user can create and view 2D coordinates
- A user can find out the distance between 2 coordinates
- A user can find find the distance of a coordinate from origin
- A user can check if a point lies on a given line
- A user can find the distance between a given 2D point and a given line

In [ ]:

```python
class Point:
    def __init__(self,x,y):
        self.x_cod = x
        self.y_cod = y

    def __str__(self):
        return f"<{self.x_cod}, {self.y_cod}>"

    def euclidean_distance(self,other):
        return ((self.x_cod-other.x_cod)**2 + (self.y_cod-other.y_cod)**2)**0.5

    def distance_from_origin(self):
#         return (self.x_cod**2 + self.y_cod**2)**0.5
        return self.euclidean_distance(Point(0,0))


class Line:
    def __init__(self, A,B,C):
        self.A = A
        self.B = B
        self.C = C

    def __str__(self):
        return f"{self.A}x + {self.B}y + {self.C} = 0"

    def point_on_line(line,point):
        if line.A*point.x_cod + line.B*point.y_cod + line.C == 0:
            return "Lies on Line."
        else:
            return "Does not lie on Line."

    def shortest_distance_l_TO_p(line,point):
        return abs(line.A*point.x_cod + line.B*point.y_cod + line.C)/(line.A**2
```

In [ ]:

```python
p1 = Point(0,0)
print(p1)
```

In [ ]:

```python
p1.euclidean_distance(Point(2,1))
```

In [ ]:

```python
p2 = Point(3,3)
p2.distance_from_origin()
```

In [ ]:

```python
l1 = Line(2,7,1)
print(l1)
```

In [ ]:

```python
l1.point_on_line(Point(-4,7))
```

In [ ]:

```python
l1.shortest_distance_l_TO_p(p2)
```

In [ ]:
```
l1 = Line(1,1,-2)
p1 = Point(1,1)

print(l1)
print(p1)

l1.shortest_distance_l_TO_p(p1)
```

Encapsulation:

- Encapsulation is the concept of bundling data (attributes) and methods (functions) that operate on that data within a class.
- It allows the class to control access to its data, ensuring data integrity and security.

In [3]:
```
class BankAccount:
    def __init__(self, account_number, balance):
        self.account_number = account_number
        self.__balance = balance

    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount

    def withdraw(self, amount):
        if 0 < amount <= self._balance:
            self.__balance -= amount

    def get_balance(self):
        return self.__balance

```

In [4]:
```
cust1 = BankAccount(45454,5000)

```

In [5]:
```
cust1.account_number
```

Out[5]: 45454

In [6]:
```
cust1.get_balance()
```

Out[6]: 5000

In [11]:
```
cust1._balance = 8000
cust1._balance
```

Out[11]: 8000

In [ ]:
```
cust1.
```

Inheritance:

- Inheritance allows a class (child class) to inherit properties and behaviors from another class (parent class).
- It promotes code reusability and establishes an "is-a" relationship between classes. Child classes can override or extend methods and attributes of the parent class.

In [59]:
```
class Animal:
    def __init__(self, species):
        self.species = species

    def make_sound(self):
        print("Generic animal sound")

    def eat(self):
        print(f"{self.species} is eating.")


class Dog(Animal):
    def __init__(self, breed):
        super().__init__("Dog")
        self.breed = breed

    def make_sound(self):
        print("Woof!")

```

In [60]:
```
lion = Animal("lion")
dog1 = Dog('chuvava')
```

In [67]:
```
dog1.make_sound()
```

```
Woof!
```

In [66]:
```
lion.make_sound()
```

```
Generic animal sound
```

In [65]:
```
lion.species
```

Out[65]:
```
'lion'
```

In [62]:
```
dog1.species
dog1.breed
```

Out[62]:
```
'chuvava'
```

In [25]:
```
d1 = Animal('cat')
d1.species
d1.make_sound()

d2 = Dog('breed1')
d2.make_sound()
d2.species

d2.eat()
d2.another_method()
```

```
Generic animal sound
Woof!
Dog is eating.
```

In [2]:
```
# Using inheritance
dog = Dog("Labrador")
print(dog.species)  # Output: "Dog"
dog.make_sound()    # Output: "Woof!"
```

```
Dog
Woof!
```

Polymorphism:

- The literal meaning of polymorphism is the condition of occurrence in different forms.
- It refers to the use of a single type entity (method, operator or object) to represent different types in different scenarios.
- Polymorphism is often achieved through method overriding.

In [13]:
```
class Car:
#     def __init__(self):
#         self.name = 'parent clas'

    def start_engine(self):
        print("Normal Engine Started......")

    def add_brake():
        pass

class Electric_car(Car):
    def __init__(self, make, model):
        self.make = make
        self.model = model

    def start_engine(self):
        print("Electric Engine Started.....")



class Hybrid_car(Car):
    def __init__(self, make, model):
        self.make = make
        self.model = model

    def start_engine(self):
        print("Hybrid Engine Started.....")

```

In [14]:
```
normal_car = Car()
my_EV = Electric_car('BMV','tesla')
my_HC = Hybrid_car('Hyundai','verna')
```

In [15]:
```
normal_car.start_engine()
```

```
Normal Engine Started......
```

In [16]:
```
my_EV.start_engine()
```

```
Electric Engine Started.....
```

In [17]:
```
my_HC.start_engine()
```

```
Hybrid Engine Started.....
```

In [2]:
```
class Car:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model

    def move(self):
        print("Drive!")


class Boat:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model

    def move(self):
        print("Sail!")

class Plane:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model

    def move(self):
        print("Fly!")
```

In [ ]:
```

```

In [3]:
```
car1 = Car("Ford", "Mustang")       #Create a Car class
boat1 = Boat("Ibiza", "Touring 20") #Create a Boat class
plane1 = Plane("Boeing", "747")
```

In [6]:
```
plane1.move()
```

```
Fly!
```

In [5]:
```
boat1.move()
```

```
Sail!
```

In [4]:
```
car1.move()
```

```
Drive!
```

In [19]:
```
class Shape:
    def area(self,radius):
        return 3.14 * (radius**2)

    def area(self, a,b):
        return a*b
```

In [21]:
```
obj = Shape()
obj.area(5)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[21], line 2
      1 obj = Shape()
----> 2 obj.area(5)

TypeError: area() missing 1 required positional argument: 'b'
```

In [25]:
```
def fun1(a, b= 1):
    return a +b
```

In [26]:
```
print(fun1(3,5))
```

```
8
```

In [27]:
```
print(fun1(3))
```

```
4
```

In [49]:
```
class Shape:
    def area(self, a, b = 0, c = 0):
        if b == 0 and c == 0:
            return 3.14 * a**2
        elif c == 0 and b!=0:
            return a*b
        else:
            return a*b*c


```

In [50]:
```
circle = Shape()
rectange = Shape()
```

In [51]:
```
circle.area(5)
```

Out[51]:
```
78.5
```

In [52]:
```
rectange.area(3,4)
```

Out[52]:
```
12
```

In [53]:
```
obj1 = Shape()
```

In [56]:
```
obj1.area(a = 4)
```

Out[56]:
```
50.24
```

In [57]:
```
obj1.area(2,3,4)
```

Out[57]:
```
24
```

In [58]:
```
obj1.area(2,3)
```

Out[58]:
```
6
```

In [42]:
```
class Shape:
    def draw(self):
        pass

class Circle(Shape):
    def draw(self):
        print("Drawing a circle")

class Square(Shape):
    def draw(self):
        print("Drawing a square")


```

In [44]:
```
# Using polymorphism
shapes = [Circle(), Square()]
for shape in shapes:
    shape.draw()
```

```
Drawing a circle
Drawing a square
```

ABSTRACTION :

- Abstraction is the process of hiding the implementation details of a class from the user and exposing only the essential features.
- Abstract classes cannot be instantiated and serve as templates for other classes to inherit from.

In [49]:

```python
from abc import ABC, abstractmethod

class BankAccount(ABC):
    def __init__(self, account_number, balance):
        self.account_number = account_number
        self.balance = balance

    @abstractmethod
    def deposit(self, amount):
        pass

    @abstractmethod
    def withdraw(self, amount):
        pass

    def get_balance(self):
        return self.balance

    @abstractmethod
    def security(self):
        pass



class SavingsAccount(BankAccount):
    def __init__(self, account_number, balance):
        super().__init__(account_number, balance)

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount

    def withdraw(self, amount):
        if 0 < amount <= self.balance:
            self.balance -= amount

    def security(self):
        print("Is secure")



class CurrentAccount(BankAccount):
    def __init__(self, account_number, balance):
        super().__init__(account_number, balance)

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount

    def withdraw(self, amount):
        if 0 < amount <= self.balance:
            self.balance -= amount

    def security(self):
        print("Is secure")
```

In [50]:

```python
# Using abstraction
savings_account = SavingsAccount("12345", 5000)
current_account = CurrentAccount("67890", 10000)
```

In [51]:
```
savings_account.deposit(2000)
current_account.withdraw(500)

print("Savings Account Balance:", savings_account.get_balance())  # Output: 7000
print("Current Account Balance:", current_account.get_balance())  # Output: 9500
```

```
Savings Account Balance: 7000
Current Account Balance: 9500
```

- The user of the Bank Account Management System only interacts with the simple interface provided by the abstract base class (BankAccount). The internal details of how deposits and withdrawals are handled are hidden (abstracted) from the user, providing a clean and easy-to-use interface for managing bank accounts.

In [56]:
```
l =[1,2,4,6]
for i in l:
    print(i)

for index, value in enumerate(l):
    print(index, value)
```

```
1
2
4
6
0 1
1 2
2 4
3 6
```

In [ ]:
```
[1,2,3,4,5]
task_managet.mark_complte(5)
```

In [102]:

```
'''
Project Overview:
The Task Manager will have the following features:

1. Add a new task with a title and description.
2. View all tasks with their details.
3. Mark a task as completed.
4. Remove a task from the list.
'''



class Task:
    def __init__(self, title, description):
        self.title = title
        self.description = description
        self.completed = False


    def mark_completed(self):
        self.completed = True




class TaskManager:
    def __init__(self):
        self.tasks = []

    def add_task(self, title, description):
        task = Task(title,description)
        self.tasks.append(task)

    def view_task(self):
        for index, task in enumerate(self.tasks,start=1):
            status = "completd" if task.completed else "Not completed"
            print(f"{index}. Title: {task.title}, Description: {task.description

    def mark_completed(self, task_index):
        if 1 <= task_index <= len(self.tasks):
            task = self.tasks[task_index-1]
            task.mark_completed()
        else:
            print('Invalid task number...')

    def remove_task(self, task_index):
        if 1 <= task_index <= len(self.tasks):
            self.tasks.pop(task_index-1)
        else:
            print("Invalid Task index.")


```

In [103]:

```
tm = TaskManager()
```

In [104]:

```
tm.add_task('t1','d1')
tm.add_task('t2','d2')
tm.add_task('t3','d3')
```

In [105]:
```
1 tm.view_task()
```

```
1. Title: t1, Description: d1, Status: Not completed
2. Title: t2, Description: d2, Status: Not completed
3. Title: t3, Description: d3, Status: Not completed
```

In [115]:
```
1 tm.mark_completed(2)
```

In [116]:
```
1 tm.view_task()
```

```
1. Title: t2, Description: d2, Status: Not completed
2. Title: t3, Description: d3, Status: completd
```

In [117]:
```
1 tm.mark_completed(1)
```

In [118]:
```
1 tm.view_task()
```

```
1. Title: t2, Description: d2, Status: completd
2. Title: t3, Description: d3, Status: completd
```

In [119]:
```
1 tm.remove_task(1)
```

In [120]:
```
1 tm.view_task()
```

```
1. Title: t3, Description: d3, Status: completd
```

In [121]:

```
def main():
     task_manager = TaskManager()

     while True:
        print("\nTask Manager Menu:")
        print("1. Add Task")
        print("2. View Tasks")
        print("3. Mark Task as Completed")
        print("4. Remove Task")
        print("5. Exit")

        choice = input("Enter your choice: ")

        if choice == '1':
            title = input("Enter task title: ")
            description = input("Enter task description: ")
            task_manager.add_task(title, description)
            print("Task added succssully!..")

        elif choice == '2':
            task_manager.view_task()

        elif choice == '3':
            task_index = int(input("Enter the task number to mark as completed:
            task_manager.mark_completed(task_index)

        elif choice == '4':
            task_index = int(input("Enter the task number to remove: "))
            task_manager.remove_task(task_index)

        elif choice == '5':
            print("Exiting Task Manager.")
            break
        else:
            print("Invalid choice. Please try again.")

```

In [122]:

```
1 main()
```

```
Task Manager Menu:
1. Add Task
2. View Tasks
3. Mark Task as Completed
4. Remove Task
5. Exit
Enter your choice: 1
Enter task title: t1
Enter task description: ghjfhdgsf
Task added succssully!..

Task Manager Menu:
1. Add Task
2. View Tasks
3. Mark Task as Completed
4. Remove Task
5. Exit
Enter your choice: 2
1. Title: t1, Description: ghjfhdgsf, Status: Not completed

Task Manager Menu:
1. Add Task
2. View Tasks
3. Mark Task as Completed
4. Remove Task
5. Exit
Enter your choice: 3
Enter the task number to mark as completed: 1

Task Manager Menu:
1. Add Task
2. View Tasks
3. Mark Task as Completed
4. Remove Task
5. Exit
Enter your choice: 4
Enter the task number to remove: 1

Task Manager Menu:
1. Add Task
2. View Tasks
3. Mark Task as Completed
4. Remove Task
5. Exit
Enter your choice: 2

Task Manager Menu:
1. Add Task
2. View Tasks
3. Mark Task as Completed
4. Remove Task
5. Exit
Enter your choice: 1
Enter task title: 5
Enter task description: fhgjg
Task added succssully!..

Task Manager Menu:
1. Add Task
2. View Tasks
3. Mark Task as Completed
4. Remove Task
5. Exit
```

```
Enter your choice: 5
Exiting Task Manager.
```

In [ ]:

```
1
```