1. This program is a simplified task tracking system implemented using Python classes.
    - The `Task` class represents a single task or to-do item.
    - It has an `__init__` method that initializes a task with two attributes:
        - `task_name` : This attribute stores the name or description of the task.
        - `due_date` : This attribute stores the due date of the task.
    - The `TaskTracker` class serves as a container for managing multiple tasks.
    - It has an `__init__` method that initializes an empty list called `tasks` to store the tasks.
    - `add_task(self, task)` : This method allows you to add a task to the task tracker's list of tasks. It takes a `task` object as a parameter and appends it to the `tasks` list.
    - `display_tasks(self)` : This method displays the list of tasks stored in the `tasks` list. It iterates through the list and prints the name and due date of each task.
    - In the main part of the program, an instance of the `TaskTracker` class called `task_tracker` is created.
    - Two `Task` objects, `task1` and `task2`, are created and initialized with task names and due dates.
    - The `add_task` method is used to add these tasks to the `task_tracker` instance.
    - Finally, the `display_tasks` method is called to display the list of tasks along with their names and due dates.

```
class Task:

  def __init__(self,task_name,due_date):

    self.task_name = task_name
    self.due_date = due_date

class TaskTracker:
  tasks = []

  #def __init__(self):
#    self.tasks = []

  def add_task(self,task):
    self.tasks.append(task)

  def display_task(self):
    for task in self.tasks:
      print(f"The task {task.task_name} has {task.due_date} due date")

#main program

task_tracker = TaskTracker()
task1 = Task('To complete PowerBI Course','21-11-2024')
task2 = Task('To complete Data Analysis Course','22-12-2024')
task3 = Task('To complete Machine Learning Course','02-02-2025')
task_tracker.add_task(task1)
task_tracker.add_task(task2)
task_tracker.add_task(task3)
task_tracker.display_task()
```

```
The task To complete PowerBI Course has 21-11-2024 due date
The task To complete Data Analysis Course has 22-12-2024 due date
The task To complete Machine Learning Course has 02-02-2025 due date
```

2.This program simulates a basic "Inventory Management" system.

- The `Product` class represents a product in the inventory.
- It has the following attributes:
    - `product_id` : A unique identifier for the product.
    - `name` : The name or description of the product.
    - `price` : The price of the product.
    - `quantity` : The quantity of the product available in the inventory.
- The `__init__` method initializes these attributes when a `Product` object is created.
- The `Inventory` class manages a list of products in the inventory.
- It has the following methods:

- `add_product(self, product_id, name, price, quantity)` : This method allows you to add a product to the inventory. It creates a `Product` object with the given details and appends it to the list of products.
- `update_quantity(self, product_id, new_quantity)` : This method updates the quantity of a product in the inventory. It searches for the product by `product_id` and updates its `quantity` attribute with the new quantity.
- `display_inventory(self)` : This method displays the current inventory by iterating through the list of products and printing their details.

- In the main part of the program, an instance of the `Inventory` class called `inventory` is created.
- Two products are added to the inventory using the `add_product` method with their respective details.
- The quantity of one of the products is updated using the `update_quantity` method.
- Finally, the `display_inventory` method is called to display the current inventory with product details.

```
class Product:

  def __init__(self,product_id,name,price,quantity):
    self.product_id = product_id
    self.name = name
    self.price = price
    self.quantity = quantity

class Inventory:

  list_of_products = [] #class attribute

  def add_product(self,product):
    self.list_of_products.append(product)

  def update_quantity(self,product_id,new_quantity):
    for id in self.list_of_products:
      if id.product_id == product_id:
        id.quantity = new_quantity
        Inventory.display_inventory(self)



  def display_inventory(self):
    for product in self.list_of_products:
      print(f"Product {product.name} with {product.product_id} has quantity of {product.quantity} for the price of {product.price}'

#main program

pro1 = Product(101,'Mobile',10000,2)
print(pro1.name)
pro2 = Product(102,'Laptop',30000,3)
inventory = Inventory()
inventory.add_product(pro1)
inventory.add_product(pro2)
inventory.display_inventory()
print("After updating quantity")

#updating quantity
inventory.update_quantity(101,10)
```

```
Mobile
Product Mobile with 101 has quantity of 2 for the price of 10000
Product Laptop with 102 has quantity of 3 for the price of 30000
After updating quantity
Product Mobile with 101 has quantity of 10 for the price of 10000
Product Laptop with 102 has quantity of 3 for the price of 30000
```

. This program calculates and displays the Fibonacci sequence up to a specified number of terms. Fibonacci Function

- Generates the Fibonacci sequence up to `n` terms using a `while` loop.
- Returns the sequence as a list. Get Input Function (`get_input`):
- Prompts the user for the number of Fibonacci terms.
- Ensures valid input (a positive integer) through error handling. Display Function (`display_fibonacci_sequence`)

- Prints the generated Fibonacci sequence. In the main program:
- It obtains the user's input for the number of terms.
- Calls the `fibonacci` function to generate the sequence.
- Uses the `display_fibonacci_sequence` function to print the sequence. The program is designed for generating and displaying Fibonacci sequences with input validation.

```
def display_fibonacci_sequence(n):
  a = 0
  b = 1
  sequence = [a,b]
  while(n>2):
    c = b + a
    sequence.append(c)
    a = b
    b = c


    n = n - 1

  print(sequence)


def fibonacci(n):


  if n<=0:
    return "Invalid Input"
  else:
    display_fibonacci_sequence(n)

#main program
n = int(input("Enter number of terms: "))
fibonacci(n)
```

```
Enter number of terms: 6
[0, 1, 1, 2, 3, 5]
```

```
class Book:
    def __init__(self, title, author, available_copies,total_copies):
        self.title = title
        self.author = author
        self.available_copies = available_copies
        self.total_copies = available_copies
    def borrow(self,num_copies):
        if num_copies <0:
            print("Error: Please borrow at least one copy.")

        if num_copies < self.available_copies:
            self.available_copies -= num_copies
            print(f"{num_copies} copies of '{self.title}' borrowed successfully.")
        else:
            print(f"Error: Only {self.available_copies} copies of '{self.title}' available.")
    def return_book(self, num_copies):
        if num_copies < 0:
            print("Error: Please return at least one copy.")
            return
        if num_copies <= self.total_copies - self.available_copies:
            self.available_copies += num_copies
            print(f"{num_copies} copies of '{self.title}' returned successfully.")
        else:
            print("Error: Invalid return quantity.")
    def display_info(self):
        print("Book Information:")
        print(f"Title: {self.title}")
        print(f"Author: {self.author}")
        print(f"Available Copies: {self.available_copies}")
        print(f"Total copies: {self.total_copies}")
book1 = Book("Python Programming", "John Smith", 5,10)
book1.display_info()
book1.borrow(3)
book1.display_info()
book1.return_book(2)
book1.display_info()
book1.return_book(6)
book1.borrow(0)
book1.borrow(10)
```

```
Book Information:
Title: Python Programming
Author: John Smith
Available Copies: 5
Total copies: 5
3 copies of 'Python Programming' borrowed successfully.
Book Information:
Title: Python Programming
Author: John Smith
Available Copies: 2
Total copies: 5
2 copies of 'Python Programming' returned successfully.
Book Information:
Title: Python Programming
Author: John Smith
Available Copies: 4
Total copies: 5
Error: Invalid return quantity.
0 copies of 'Python Programming' borrowed successfully.
Error: Only 4 copies of 'Python Programming' available.
```

```
class Product:
    def __init__(self, name, price, stock_quantity):
        self.name = name
        self.price = price
        self.stock_quantity = stock_quantity
    def purchase(self, quantity):
        if quantity <= 0:
            print("Error: Please purchase at least one item.")
            return
        if quantity <= self.stock_quantity:
            self.stock_quantity -= quantity
            print(f"Purchased {quantity} {self.name}(s) for Rs. {self.price * quantity:.2f}.")
        else:
```

```
        -----
            print(f"Error: Only {self.stock_quantity} {self.name}(s) available.")
    def display_info(self):
        print("Product Information:")
        print(f"Name: {self.name}")
        print(f"Price: Rs.{self.price:.2f}")
        print(f"Stock Quantity: {self.stock_quantity}")
class ShoppingCart:
    def __init__(self):
        self.items = []
    def add_item(self, product, quantity):
        if quantity <= 0:
            print("Error: Please add at least one item.")
            return
        self.items.append((product, quantity))
        print(f"Added {quantity} {product.name}(s) to the cart.")
    def checkout(self):
        total_price = 0
        print("Shopping Cart:")
        for product, quantity in self.items:
            if quantity <= product.stock_quantity:
                total_price += product.price * quantity
                print(f"{product.name}: {quantity} @ Rs.{product.price:.2f} each")
                product.stock_quantity -= quantity
            else:
                print(f"Error: Not enough stock for {product.name}.")
        print(f"Total Price: Rs.{total_price:.2f}")
        #self.items = []

product1 = Product("Laptop", 1000, 5)
product2 = Product("Headphones", 100, 10)
cart = ShoppingCart()

product1.display_info()
product2.display_info()
cart.add_item(product1, 2)
cart.add_item(product2, 12)
cart.add_item(product1, -1)
cart.checkout()
product1.display_info()
product2.display_info()

product1.purchase(10)
cart.add_item(product1, 3)
cart.checkout()
```

```
Product Information:
Name: Laptop
Price: Rs.1000.00
Stock Quantity: 5
Product Information:
Name: Headphones
Price: Rs.100.00
Stock Quantity: 10
Added 2 Laptop(s) to the cart.
Added 12 Headphones(s) to the cart.
Error: Please add at least one item.
Shopping Cart:
Laptop: 2 @ Rs.1000.00 each
Error: Not enough stock for Headphones.
```