**TASK=1**

**Code generated by Gemini colab:**

def analyze\_numbers(numbers):

  """

  Calculates the mean, minimum, and maximum values from a list of numbers.

  Args:

    numbers: A list of numbers.

  Returns:

    A tuple containing the mean, minimum, and maximum values.

    Returns None if the list is empty.

  """

  if not numbers:

    return None

  minimum = min(numbers)

  maximum = max(numbers)

  mean = sum(numbers) / len(numbers)

  return mean, minimum, maximum

# Example usage:

my\_list = [10, 20, 5, 30, 15]

result = analyze\_numbers(my\_list)

if result:

  mean\_value, min\_value, max\_value = result

  print(f"List: {my\_list}")

  print(f"Mean: {mean\_value}")

  print(f"Minimum: {min\_value}")

  print(f"Maximum: {max\_value}")

else:

  print("The list is empty.")

**OUTPUT:**

List: [10, 20, 5, 30, 15]

Mean: 16.0

Minimum: 5

Maximum: 30

**Code generated by Copilot:**

def calculate\_stats(numbers):

    """

    Reads a list of numbers and calculates the mean, minimum, and maximum value.

    Parameters:

    - numbers (list of float/int): The list of numbers.

    Returns:

    - mean (float): The average value.

    - minimum (float/int): The smallest value.

    - maximum (float/int): The largest value.

    """

    if not numbers:

        print("The list is empty.")

        return None, None, None

    mean = sum(numbers) / len(numbers)

    minimum = min(numbers)

    maximum = max(numbers)

    return mean, minimum, maximum

# Example usage:

num\_list = [float(x) for x in input("Enter numbers separated by space: ").split()]

mean, minimum, maximum = calculate\_stats(num\_list)

print(f"Mean: {mean}")

print(f"Minimum: {minimum}")

print(f"Maximum: {maximum}")

**OUTPUT:**

Enter numbers separated by space:

10 20 5 30 15

Mean: 16.0

Minimum: 5.0

Maximum: 30.0

**Difference:**

The code generated by the colab is static-way whereas the code generated by copilot is in dynamic-way.

**TASK=3**

**Code snippet by Gemini Colab:**

def find\_prime\_numbers\_in\_range(start, end):

  """

  Finds all prime numbers within a given range (inclusive).

  Args:

    start: The starting integer of the range.

    end: The ending integer of the range.

  Returns:

    A list of prime numbers within the specified range.

  """

  prime\_numbers = []

  for num in range(start, end + 1):

    # Prime numbers are greater than 1

    if num > 1:

      for i in range(2, int(num\*\*0.5) + 1):

        if (num % i) == 0:

          break  # Not a prime number

      else:

        prime\_numbers.append(num)  # It is a prime number

  return prime\_numbers

# Example usage:

start\_range = 10

end\_range = 50

prime\_list = find\_prime\_numbers\_in\_range(start\_range, end\_range)

print(f"Prime numbers between {start\_range} and {end\_range}: {prime\_list}")  
  
**OUTPUT:**

Prime numbers between 10 and 50:[11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]  
  
  
**CODE EXPLAINATION GIVEN BY GEMINI:**

Here's a breakdown of the code:

* **def find\_prime\_numbers\_in\_range(start, end):**: This line defines the function named find\_prime\_numbers\_in\_range that accepts two arguments: start and end, representing the beginning and end of the range to check.
* **prime\_numbers = []**: This initializes an empty list called prime\_numbers which will store the prime numbers found within the range.
* **for num in range(start, end + 1):**: This loop iterates through each number from start to end (inclusive).
* **if num > 1:**: This condition checks if the current number num is greater than 1, because prime numbers must be greater than 1.
* **`for i in range(2, int(num**0.5) + 1):\*\*: This nested loop iterates from 2 up to the square root ofnum`. We only need to check for divisors up to the square root of a number to determine if it's prime.
* **if (num % i) == 0:**: This condition checks if num is divisible by i. If it is, it means num has a divisor other than 1 and itself, so it's not a prime number.
* **break**: If a divisor is found, the inner loop is exited using break because we already know the number is not prime.
* **else:**: This else block is associated with the inner for loop. It is executed only if the inner loop completes without finding any divisors (i.e., the break statement was not executed). This indicates that num is a prime number.
* **prime\_numbers.append(num)**: If num is determined to be prime, it is added to the prime\_numbers list.
* **return prime\_numbers**: After checking all numbers in the range, the function returns the prime\_numbers list containing all the prime numbers found.
* **Example Usage**: The code then demonstrates how to use the function by setting a start\_range and end\_range, calling the function, and printing the resulting list of prime numbers.

**TASK#4**

**Home work (7/8/2025)**

**CODE:**

from datetime import datetime

# Store all records here

records = []

def add\_record():

date = datetime.today().strftime('%Y-%m-%d')

engine = input("Enter Engine Number (1-4): ")

fuel\_type = input("Enter Fuel Type (Petrol/Diesel): ").capitalize()

liters = float(input("Enter No. of Liters: "))

rate = float(input("Enter Rate per Liter: "))

total = liters \* rate

payment\_mode = input("Enter Payment Mode (Cash/UPI/Card/Credit/Debit): ").capitalize()

record = {

"date": date,

"engine": engine,

"fuel\_type": fuel\_type,

"liters": liters,

"rate": rate,

"total": total,

"payment\_mode": payment\_mode

}

records.append(record)

print("\n✅ Record Added Successfully!\n")

def show\_summary():

total\_money = 0

payment\_summary = {}

print("\n----- DAILY SUMMARY -----")

for record in records:

total\_money += record["total"]

mode = record["payment\_mode"]

payment\_summary[mode] = payment\_summary.get(mode, 0) + record["total"]

for mode, amount in payment\_summary.items():

print(f"{mode}: ₹{amount:.2f}")

print(f"\nTotal Collection Today: ₹{total\_money:.2f}")

print("--------------------------\n")

def main():

while True:

print("====== HP Petrol Bunk Management ======")

print("1. Add New Record")

print("2. Show Daily Summary")

print("3. Exit")

choice = input("Enter your choice (1/2/3): ")

if choice == '1':

add\_record()

elif choice == '2':

show\_summary()

elif choice == '3':

print("Exiting... Have a great day!")

break

else:

print("Invalid choice. Please try again.\n")

if \_\_name\_\_ == "\_\_main\_\_":

main()  
  
  
  
  
**OUTPUT:**

====== HP Petrol Bunk Management ======

1. Add New Record

2. Show Daily Summary

3. Exit

Enter your choice (1/2/3): 1

Enter Engine Number (1-4): 4

Enter Fuel Type (Petrol/Diesel): petrol

Enter No. of Liters: 500

Enter Rate per Liter: 99

Enter Payment Mode (Cash/UPI/Card/Credit/Debit): Debit

✅ Record Added Successfully!

====== HP Petrol Bunk Management ======

1. Add New Record

2. Show Daily Summary

3. Exit

Enter your choice (1/2/3): 2

----- DAILY SUMMARY -----

Debit: ₹49500.00

Total Collection Today: ₹49500.00

--------------------------

**On 8/8/2025**

**Task#1**

**charGPT**

1)

def factorial\_recursive(n):

if n == 0 or n == 1:

return 1

else:

return n \* factorial\_recursive(n - 1)

# Example usage

num = int(input("Enter a number: "))

print(f"Factorial of {num} is {factorial\_recursive(num)}")

2)

def factorial\_recursive(n):

if n == 0 or n == 1:

return 1

else:

return n \* factorial\_recursive(n - 1)

# Example usage

num = int(input("Enter a number: "))

print(f"Factorial of {num} is {factorial\_recursive(num)}")

**Copilot:**

1)recursive:

def factorial\_recursive(n):

if n < 0:

raise ValueError("Factorial is not defined for negative numbers.")

if n == 0 or n == 1:

return 1

return n \* factorial\_recursive(n - 1)

# Example usage

print(factorial\_recursive(5))

# Output: 120

2)iterative:

def factorial\_iterative(n):

if n < 0:

raise ValueError("Factorial is not defined for negative numbers.")

result = 1

for i in range(2, n + 1):

result \*= i

return result

# Example usage

print(factorial\_iterative(5)) # Output: 120

**3)Module**

import math

print(math.factorial(5))

# Output: 120

Gemini colab

1)

def factorial(n):

  """

  Calculates the factorial of a non-negative integer.

  Args:

    n: A non-negative integer.

  Returns:

    The factorial of n.

  Raises:

    ValueError: If n is a negative integer.

  """

  if not isinstance(n, int) or n < 0:

    raise ValueError("Input must be a non-negative integer")

  if n == 0:

    return 1

  else:

    return n \* factorial(n-1)

# Example usage:

print(factorial(5))

print(factorial(0))

# print(factorial(-1)) # Uncommenting this will raise a ValueError

**Output:**

120

1

**task#2**

provide a clear example input-output prompt to generate a sorting function

**CODE:**

def sort\_list(items, reverse=False):

"""

Sorts a list in ascending or descending order.

:param items: list of numbers or strings

:param reverse: False = Ascending, True = Descending

:return: sorted list

"""

return sorted(items, reverse=reverse)

# Example Usage

numbers = [12, 4, 56, 17, 8, 99]

words = ["banana", "apple", "cherry", "date"]

print("Original Numbers:", numbers)

print("Ascending:", sort\_list(numbers))

print("Descending:", sort\_list(numbers, reverse=True))

print("\nOriginal Words:", words)

print("Ascending:", sort\_list(words))

print("Descending:", sort\_list(words, reverse=True))

**OUTPUT:**

Original Numbers: [12, 4, 56, 17, 8, 99]

Ascending: [4, 8, 12, 17, 56, 99]

Descending: [99, 56, 17, 12, 8, 4]

Original Words: ['banana', 'apple', 'cherry', 'date']

Ascending: ['apple', 'banana', 'cherry', 'date']

Descending: ['date', 'cherry', 'banana', 'apple']

**task#3**

vague prompt ”generate python code to calculate power bill”

improve it step by step

**code:**

from datetime import datetime

# Store all bills for the day

bills = []

def calculate\_bill(units):

"""Calculate bill amount based on slabs"""

if units <= 50:

amount = units \* 2.5

elif units <= 150:

amount = (50 \* 2.5) + ((units - 50) \* 3.0)

elif units <= 250:

amount = (50 \* 2.5) + (100 \* 3.0) + ((units - 150) \* 3.5)

else:

amount = (50 \* 2.5) + (100 \* 3.0) + (100 \* 3.5) + ((units - 250) \* 5.0)

fixed\_charge = 50

return amount + fixed\_charge

def add\_customer\_bill():

"""Add new customer bill"""

customer\_id = len(bills) + 1

name = input("Enter Customer Name: ")

units = float(input("Enter Units Consumed: "))

date = datetime.today().strftime('%Y-%m-%d')

bill\_amount = calculate\_bill(units)

bill = {

"id": customer\_id,

"name": name,

"units": units,

"date": date,

"amount": bill\_amount

}

bills.append(bill)

print("\n✅ Bill Generated Successfully!\n")

print("----- ELECTRICITY BILL -----")

print(f"Bill No : {customer\_id}")

print(f"Date : {date}")

print(f"Customer Name : {name}")

print(f"Units Consumed: {units}")

print(f"Total Amount : ₹{bill\_amount:.2f}")

print("----------------------------\n")

def show\_daily\_summary():

"""Show total collection for the day"""

if not bills:

print("\nNo bills generated yet.\n")

return

total\_collection = sum(bill["amount"] for bill in bills)

print("\n===== DAILY COLLECTION SUMMARY =====")

for bill in bills:

print(f"{bill['id']}. {bill['name']} - Units: {bill['units']} - ₹{bill['amount']:.2f}")

print("------------------------------------")

print(f"Total Collection Today: ₹{total\_collection:.2f}")

print("====================================\n")

def main():

while True:

print("====== Electricity Bill System ======")

print("1. Generate New Bill")

print("2. Show Daily Summary")

print("3. Exit")

choice = input("Enter choice (1/2/3): ")

if choice == '1':

add\_customer\_bill()

elif choice == '2':

show\_daily\_summary()

elif choice == '3':

print("Exiting... Have a nice day!")

break

else:

print("❌ Invalid choice. Try again.\n")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**OUTPUT:**

====== Electricity Bill System ======

1. Generate New Bill

2. Show Daily Summary

3. Exit

Enter choice (1/2/3): 1

Enter Customer Name: sravs

Enter Units Consumed: 200

✅ Bill Generated Successfully!

----- ELECTRICITY BILL -----

Bill No : 1

Date : 2025-08-08

Customer Name : sravs

Units Consumed: 200.0

Total Amount : ₹650.00

**Code:**

from datetime import datetime, timedelta

def generate\_tsspdcl\_bill():

print("=== TSSPDCL ELECTRICITY BILL GENERATOR ===")

# Consumer details

consumer\_name = input("Enter Consumer Name: ")

service\_no = input("Enter Service Number: ")

address = input("Enter Address: ")

# Billing details

bill\_month = input("Enter Bill Month (e.g., AUG 2025): ")

current\_date = datetime.now().strftime("%d-%m-%Y")

due\_date = (datetime.now() + timedelta(days=15)).strftime("%d-%m-%Y")

# Meter readings

prev\_reading = float(input("Enter Previous Reading (kWh): "))

curr\_reading = float(input("Enter Current Reading (kWh): "))

# Calculations

units\_consumed = curr\_reading - prev\_reading

rate\_per\_unit = float(input("Enter Rate per Unit (₹): "))

energy\_charges = units\_consumed \* rate\_per\_unit

fixed\_charges = float(input("Enter Fixed Charges (₹): "))

gst\_rate = 18 # %

gst\_amount = (energy\_charges + fixed\_charges) \* gst\_rate / 100

total\_amount = energy\_charges + fixed\_charges + gst\_amount

# Print bill

print("\n" + "="\*50)

print("TELANGANA STATE SOUTHERN POWER DISTRIBUTION COMPANY LTD (TSSPDCL)")

print("="\*50)

print(f"Consumer Name : {consumer\_name}")

print(f"Service No. : {service\_no}")

print(f"Address : {address}")

print(f"Bill Month : {bill\_month}")

print(f"Bill Date : {current\_date}")

print(f"Due Date : {due\_date}")

print("-"\*50)

print(f"Previous Reading : {prev\_reading} kWh")

print(f"Current Reading : {curr\_reading} kWh")

print(f"Units Consumed : {units\_consumed} kWh")

print(f"Rate per Unit : ₹{rate\_per\_unit}")

print("-"\*50)

print(f"Energy Charges : ₹{energy\_charges:.2f}")

print(f"Fixed Charges : ₹{fixed\_charges:.2f}")

print(f"GST ({gst\_rate}%) : ₹{gst\_amount:.2f}")

print("="\*50)

print(f"Total Amount Payable: ₹{total\_amount:.2f}")

print("="\*50)

print("Please pay your bill before the due date to avoid penalties.")

print("="\*50)

# Run the bill generator

generate\_tsspdcl\_bill()

**SAMPLE INPUT:**

=== TSSPDCL ELECTRICITY BILL GENERATOR ===

Enter Consumer Name: sravs

Enter Service Number: 234

Enter Address: nrpt

Enter Bill Month (e.g., AUG 2025): Jun 2025

Enter Previous Reading (kWh): 12

Enter Current Reading (kWh): 20

Enter Rate per Unit (₹): 99

Enter Fixed Charges (₹): 10

**OUTPUT:**

==================================================

TELANGANA STATE SOUTHERN POWER DISTRIBUTION COMPANY LTD (TSSPDCL)

==================================================

Consumer Name : sravs

Service No. : 234

Address : nrpt

Bill Month : Jun 2025

Bill Date : 08-08-2025

Due Date : 23-08-2025

--------------------------------------------------

Previous Reading : 12.0 kWh

Current Reading : 20.0 kWh

Units Consumed : 8.0 kWh

Rate per Unit : ₹99.0

--------------------------------------------------

Energy Charges : ₹792.00

Fixed Charges : ₹10.00

GST (18%) : ₹144.36

==================================================

Total Amount Payable: ₹946.36

==================================================

Please pay your bill before the due date to avoid penalties.

==================================================