**AI ASSISTED CODING LAB 2.2**

**Hall ticket number : 2403A52373**

**Name :KODURU ESWAR REDDY**

**Prompt 01: Write a function that filters out all negative numbers from a given list**

**Code:**

**A computer screen with text on it

AI-generated content may be incorrect.**

**Code explanation:**

def filter\_negative\_numbers(input\_list):

This line defines a function named filter\_negative\_numbers that accepts one argument, input\_list.

  Filters out negative numbers from a list.  
  
  Args:  
    input\_list: A list of numbers.  
  
  Returns:  
    A new list containing only the non-negative numbers from the input list.  
  """

This is a docstring, which explains what the function does, its arguments (Args), and what it returns (Returns).

  return [number for number in input\_list if number >= 0]

This line uses a list comprehension to create and return a new list.

* for number in input\_list: It iterates through each element in the input\_list.
* if number >= 0: It checks if the current number is greater than or equal to 0 (i.e., non-negative).
* [number ... ]: If the condition is true, the number is included in the new list being created.

# Example usage:

This comment indicates that the following lines demonstrate how to use the function.

my\_list = [1, -2, 3, -4, 5, 0, 6]

This line creates a list named my\_list with both positive and negative numbers, and zero.

filtered\_list = filter\_negative\_numbers(my\_list)

This line calls the filter\_negative\_numbers function with my\_list as the argument and assigns the returned filtered list to the variable filtered\_list.

print(filtered\_list)

This line prints the contents of the filtered\_list to the console.

**Output:**

****

**Prompt 02:** generate a Python code that reads a text file and counts the frequency of each word.

**Code:**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**Code explanation:**

From collections import Counter  
import re

Imports the Counter tool for counting and the re module for finding words.

def count\_word\_frequency(filepath):

Defines a function to count word frequency, taking the file path.

    with open(filepath, 'r', encoding='utf-8') as file:

Opens the text file for reading.

        text = file.read().lower()

Reads the file content and converts it to lowercase.

        words = re.findall(r'\b\w+\b', text)

Finds all words in the text.

        return Counter(words)

Counts the frequency of each word and returns the counts.

# Example usage:  
filepath = "sample\_text.txt"  # Replace with your file path  
word\_frequencies = count\_word\_frequency(filepath)

Sets the file path and calls the function to get word frequencies.

for word, count in word\_frequencies.most\_common():  
    print(f"{word}: {count}")

Prints each word and its count, starting with the most frequent.

**Output:**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Prompt 03:** generate a Python class called Book with attributes title, author, and a method summary() that prints the details

**Code:**

**A computer screen shot of text

AI-generated content may be incorrect.**

**Code explanation:**

Class Book:

This line defines a new class named Book.

    def \_\_init\_\_(self, title, author):

This is the constructor method. It's called when you create a new Book object. Self refers to the instance of the class, and title and author are the values you pass when creating the object.

        """  
        Initializes a Book object with title and author.  
  
        Args:  
            Title (str): The title of the book.  
            Author (str): The author of the book.  
        """

This is a docstring explaining the purpose of the \_\_init\_\_ method and its arguments.

        self.title = title  
        self.author = author

These lines assign the values passed for title and author to the title and author attributes of the Book object.

    def summary(self):

This defines a method named summary for the Book class. It takes self as an argument, referring to the specific Book object the method is called on.

        """  
        Prints the details of the book (title and author).  
        """

This is a docstring explaining what the summary method does.

        print(f"Title: {self.title}")  
        print(f"Author: {self.author}")

These lines print the title and author of the Book object using f-strings to format the output.

# Example usage:  
my\_book = Book("The Hitchhiker's Guide to the Galaxy", "Douglas Adams")  
my\_book.summary()

This is an example showing how to create a Book object named my\_book and then call the summary() method on it.

**Output:**

****

**Prompt 04:** Write a program that checks whether a number is an Armstrong number

**Gemini Code:**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Cursor code:**

**A computer screen shot of a program

AI-generated content may be incorrect.**

**Code explanation:**

**Gemini:**

def is\_armstrong\_number(number):

Defines a function to check if a number is an Armstrong number.

  Checks if a number is an Armstrong number.  
  ...  
  """

Docstring explaining what the function does.

  num\_str = str(number)  
  num\_digits = len(num\_str)  
  sum\_of\_powers = 0

Converts the number to a string, gets the number of digits, and initializes a sum.

  For digit in num\_str:  
    sum\_of\_powers += int(digit) \*\* num\_digits

Iterates through each digit, converts it to an integer, raises it to the power of the number of digits, and adds it to the sum.

  return sum\_of\_powers == number

Returns True if the sum of powers equals the original number, otherwise False.

# Example usage:  
num\_to\_check = 153  
# ... (and another example with 123)

Example code showing how to use the function with different numbers.

if is\_armstrong\_number(num\_to\_check):  
  print(f"{num\_to\_check} is an Armstrong number.")  
Else:  
  print(f"{num\_to\_check} is not an Armstrong number.")

Checks if the number is Armstrong and prints the result.

**Output:**

**Gemini:**

**A grey rectangular sign with white text

AI-generated content may be incorrect.**

**CURSOR:**

**A black background with white text

AI-generated content may be incorrect.**

**Prompt 05:** generate code for sorting a list of dictionaries by a specific key (e.g., age).

**Code:**

**GEMINI:**

**A computer screen shot of a program

AI-generated content may be incorrect.**

**CURSOR:**

**A computer screen shot of a program code

AI-generated content may be incorrect.**

**Code explanation:**

**GEMINI:**

def sort\_list\_of\_dicts(list\_of\_dicts, key\_to\_sort\_by):

Defines a function to sort a list of dictionaries by a specified key.

  """  
  Sorts a list of dictionaries by a specific key.  
  ...  
  """

Docstring explaining the function's purpose.

  return sorted(list\_of\_dicts, key=lambda x: x[key\_to\_sort\_by])

Sorts the input list using the sorted() function. The key=lambda x: x[key\_to\_sort\_by] part tells sorted() to use the value associated with key\_to\_sort\_by in each dictionary (x) for sorting.

# Example usage:  
people = [  
    {'name': 'Alice', 'age': 30},  
    {'name': 'Bob', 'age': 25},  
    {'name': 'Charlie', 'age': 35},  
    {'name': 'David', 'age': 25}  
]

Creates an example list of dictionaries representing people.

# Sort by age  
sorted\_by\_age = sort\_list\_of\_dicts(people, 'age')  
print("Sorted by age:")  
For person in sorted\_by\_age:  
    print(person)

Sorts the people list by the 'age' key and prints the result.

# Sort by name  
sorted\_by\_name = sort\_list\_of\_dicts(people, 'name')  
print("\nSorted by name:")  
For person in sorted\_by\_name:  
    print(person)

Sorts the people list by the 'name' key and prints the result.

**Output:**

**GEMINI:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**CURSOR:**

**A screen shot of a computer

AI-generated content may be incorrect.**

**Comparison of gemini and cursor ai:**

**Gemini is faster and easier to access than Cursor AI**

**And more accurate too**