

AI ASSISTED CODING

LAB ASSIGNMENT – 2.2

NAME : M.Shiva
HALL.NO : 2403A52377
BATCH.NO : AI 14
ASSIGNMENT NO : 2.2

PROMPT 01 :

i need a pyhton code , unction that filters out all negative numbers from a given list

CODE :

```
def filter_negative_numbers(input_list):  
    """  
    Filters out negative numbers from a given list.  
  
    Args:  
        input_list: A list of numbers.  
  
    Returns:  
        A new list containing only the non-negative numbers from the input list.  
    """  
    return [number for number in input_list if number >= 0]  
  
# Example usage:  
my_list = [1, -2, 3, -4, 5, 0, -6]  
filtered_list = filter_negative_numbers(my_list)  
print(filtered_list)
```

OUTPUT :

```
[1, 3, 5, 0]
```

CODE EXPLANATION :

- **Function Definition:** The function `filter_negative_numbers` is defined to take a list of numbers as input.

- **Docstring:** The docstring explains the purpose, input, and output of the function clearly.
- **List Comprehension:** The function uses a list comprehension to iterate through each number in the input list.
- **Condition Check:** Inside the list comprehension, it includes only numbers that are greater than or equal to 0 (i.e., non-negative).
- **Return Value:** It returns a new list that contains only the non-negative numbers.
- **Example Usage:** A sample list `my_list` is passed to the function, and the filtered result is stored in `filtered_list`.
- **Output:** The `print(filtered_list)` statement displays the result: `[1, 3, 5, 0]`.

PROMPT 02 :

I NEED A python code that reads a text file and counts the frequency of each word.

CODE :

```

import re

def count_word_frequency(filepath):
    """
    Reads a text file and counts the frequency of each word.

    Args:
        filepath: The path to the text file.

    Returns:
        A dictionary where keys are words and values are their frequencies.
    """
    word_frequency = {}
    try:
        with open(filepath, 'r') as file:
            text = file.read().lower() # Read the file and convert to lowercase
            words = re.findall(r'\b\w+\b', text) # Find all words using regex

            for word in words:
                word_frequency[word] = word_frequency.get(word, 0) + 1
    except FileNotFoundError:
        print(f"Error: File not found at {filepath}")
        return None
    return word_frequency

# Example usage:
# Create a dummy text file for demonstration
with open("sample.txt", "w") as f:
    f.write("This is a sample text file.\n")
    f.write("This file is for demonstrating word frequency counting.\n")
    f.write("Sample text, sample file.")

filepath = "sample.txt"
word_counts = count_word_frequency(filepath)

if word_counts:
    # Print word frequencies
    for word, count in word_counts.items():
        print(f"{word}: {count}")

```

OUTPUT :

```

↔ this: 2
   is: 2
   a: 1
   sample: 3
   text: 2
   file: 3
   for: 1
   demonstrating: 1
   word: 1
   frequency: 1
   counting: 1

```

CODE EXPLANATION :

- **Function Definition:** The `count_word_frequency` function takes a file path as input and returns a dictionary of word frequencies.
- **Docstring:** Describes the function's purpose, input (`filepath`), and output (a dictionary of word counts).
- **Dictionary Initialization:** `word_frequency` is initialized as an empty dictionary to store word counts.
- **File Reading:** The file is opened in read mode, and its content is read and converted to lowercase to ensure case-insensitive matching.
- **Regex Word Extraction:** `re.findall(r'\b\w+\b', text)` extracts all words using regular expressions, where `\w+` matches word characters.
- **Word Counting:** A loop iterates through each word, using `dict.get()` to increment the count for each occurrence.
- **Error Handling:** A `try-except` block catches `FileNotFoundError` and prints a message if the file doesn't exist.
- **Return Value:** The function returns the dictionary of word frequencies if successful.
- **Example Usage:**
 - A sample file (`sample.txt`) is created with some text content.
 - The function is called with this file's path.
 - If the result is valid, the script prints the frequency of each word.

PROMPT 03 :

I NEED A PYHTON CODE FOR a Python class called Book with attributes title, author, and a method summary() that prints the details

CODE :

```

class Book:
    """
    Represents a book with a title and author.
    """
    def __init__(self, title, author):
        """
        Initializes a new Book instance.

        Args:
            title: The title of the book.
            author: The author of the book.
        """
        self.title = title
        self.author = author

    def summary(self):
        """
        Prints the details of the book.
        """
        print(f"Title: {self.title}")
        print(f"Author: {self.author}")

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

```

OUTPUT :

```

Title: The Hitchhiker's Guide to the Galaxy
Author: Douglas Adams

```

CODE EXPLANATION :

- **Class Definition:** The `Book` class is defined to represent a book with basic details like title and author.
- **Docstring:** Provides a brief description of what the class represents.
- **Constructor (`__init__`):**
 - Takes `title` and `author` as parameters.
 - Initializes instance variables `self.title` and `self.author` with the given values.
- **`summary()` Method:**
 - A method that prints the book's title and author in a formatted way.
- **Example Usage:**

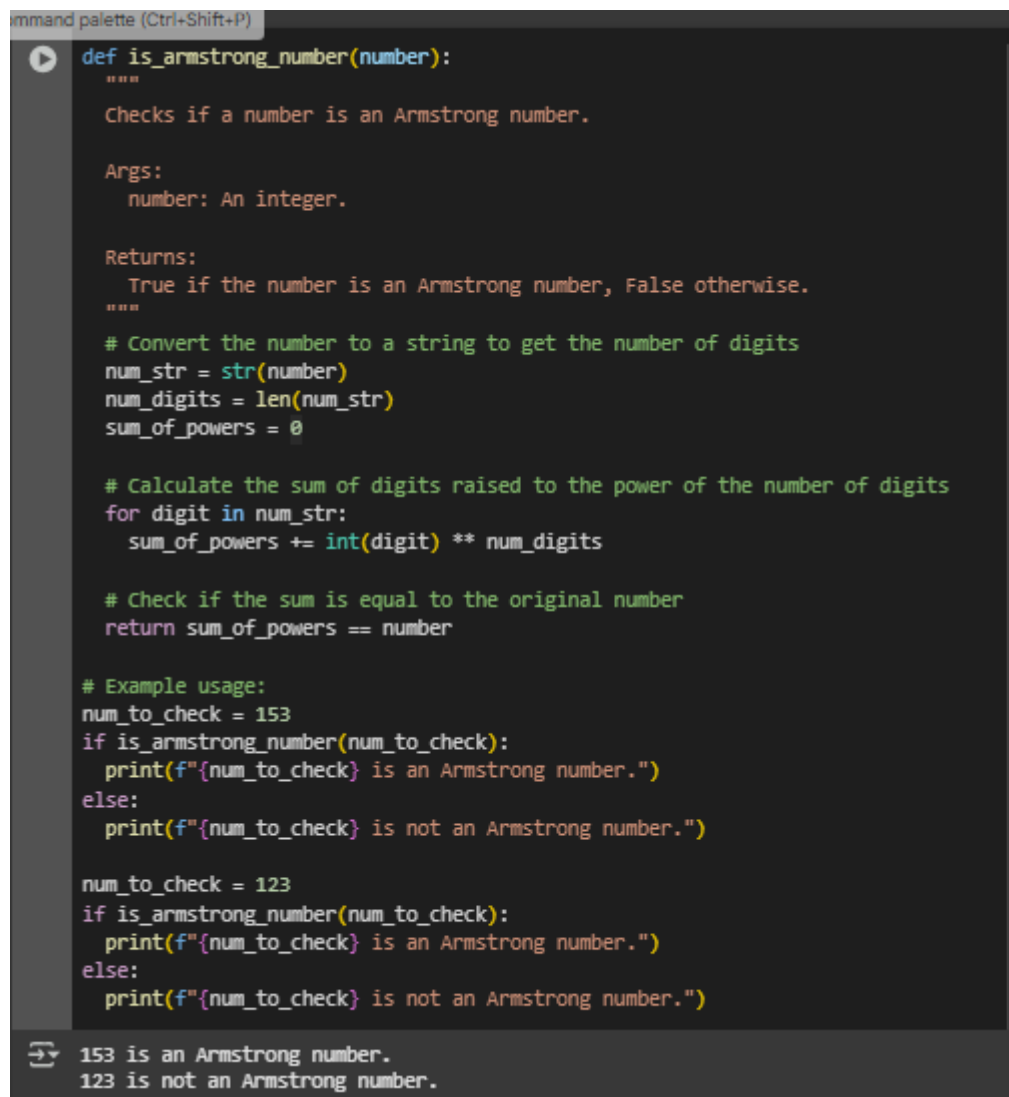
- A `Book` object is created with a title and author.
- The `summary()` method is called to display the book's details.

PROMPT 04 :

GEMINI : i need a code that program that checks whether a number is an Armstrong number

CURSOR : modify the given code in simple terms

CODE & OUTPUT (GEMINI) :



```
command palette (Ctrl-Shift-P)

def is_armstrong_number(number):
    """
    Checks if a number is an Armstrong number.

    Args:
        number: An integer.

    Returns:
        True if the number is an Armstrong number, False otherwise.
    """
    # Convert the number to a string to get the number of digits
    num_str = str(number)
    num_digits = len(num_str)
    sum_of_powers = 0

    # Calculate the sum of digits raised to the power of the number of digits
    for digit in num_str:
        sum_of_powers += int(digit) ** num_digits

    # Check if the sum is equal to the original number
    return sum_of_powers == number

# Example usage:
num_to_check = 153
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.")

num_to_check = 123
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.")
```

153 is an Armstrong number.
123 is not an Armstrong number.

CODE & OUTPUT (CURSOR) :

```

cursor002.py > is_armstrong_number
1  def is_armstrong_number(n):
2      if n < 0:
3          return False
4      s = str(n)
5      power = len(s)
6      return n == sum(int(d) ** power for d in s)
7
8  # Example usage:
9  for num in (153, 123):
10     if is_armstrong_number(num):
11         print(f"{num} is an Armstrong number.")
12     else:
13         print(f"{num} is not an Armstrong number.")

```

```

PS C:\Users\SATISH\Documents\ai coding> c:; cd 'C:\AppData\Local\Programs\Python\Python313\python.exe'
y-2025.6.0-win32-x64\bundled\libs\debugpy\launcher
sor002.py'
153 is an Armstrong number.
123 is not an Armstrong number.
PS C:\Users\SATISH\Documents\ai coding> 

```

CODE EXPLANATION :

- **Function Definition:** The `is_armstrong_number` function checks whether a given integer is an Armstrong number.
- **Docstring:** Explains the purpose, input (`number`), and output (`True` or `False`) of the function.
- **Digit Count:** Converts the number to a string to easily count how many digits it has.
- **Power Sum Calculation:** Uses a loop to raise each digit to the power of the total number of digits and adds the result to `sum_of_powers`.
- **Comparison:** Compares `sum_of_powers` with the original number to determine if it is an Armstrong number.
- **Return Value:** Returns `True` if the condition is met; otherwise, returns `False`.
- **Example Usage:**
 - Checks if 153 is an Armstrong number (which it is).
 - Then checks 123, which is not an Armstrong number.
 - Prints appropriate messages for both cases.

PROMPT 05 :

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

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

CODE & OUTPUT (GEMINI) :

```
data = [
    {'name': 'Alice', 'age': 30},
    {'name': 'Bob', 'age': 25},
    {'name': 'Charlie', 'age': 35},
    {'name': 'David', 'age': 28}
]
# Sort the list of dictionaries by 'age'
sorted_data = sorted(data, key=lambda x: x['age'])

# Print the sorted list
print(sorted_data)
```

[[{'name': 'Bob', 'age': 25}, {'name': 'David', 'age': 28}, {'name': 'Alice', 'age': 30}, {'name': 'Charlie', 'age': 35}]]

CODE & OUTPUT (CURSOR) :


```
1 # Sample data
2 people = [
3     {"name": "Alice", "age": 30},
4     {"name": "Bob", "age": 25},
5     {"name": "Charlie", "age": 35},
6 ]
7 # 1) New sorted list (ascending by age)
8 sorted_people = sorted(people, key=lambda x: x["age"])
9 print(sorted_people)
10 # 2) New sorted list (descending by age)
11 sorted_people_desc = sorted(people, key=lambda x: x["age"], reverse=True)
12 print(sorted_people_desc)
13 # 3) In-place sort (ascending by age)
14 people.sort(key=lambda x: x["age"])
15 print(people)
16 # 4) Safe if some dicts may miss the key: put missing at the end
17 people_mixed = [
18     {"name": "Alice", "age": 30},
19     {"name": "Dana"}, # no age
20     {"name": "Bob", "age": 25},
21 ]
22 sorted_safe = sorted(people_mixed, key=lambda x: x.get("age", float("inf")))
23 print(sorted_safe)
```

Problems Output Debug Console Terminal Ports Python Debug Console

```
PS C:\Users\SATISH\Documents\ai coding> & 'c:\Users\SATISH\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\SATISH\cursor\extensions\ms-python.debugpy-2025.6.0-win32-x64\bundled\libs\debugpy\launcher' '60320' '--' 'C:\Users\SATISH\Documents\ai coding\cursor002.py'
[{'name': 'Bob', 'age': 25}, {'name': 'Alice', 'age': 30}, {'name': 'Charlie', 'age': 35}]
[{'name': 'Charlie', 'age': 35}, {'name': 'Alice', 'age': 30}, {'name': 'Bob', 'age': 25}]
[{'name': 'Bob', 'age': 25}, {'name': 'Alice', 'age': 30}, {'name': 'Charlie', 'age': 35}]
[{'name': 'Bob', 'age': 25}, {'name': 'Alice', 'age': 30}, {'name': 'Dana'}]
PS C:\Users\SATISH\Documents\ai coding>
```

CODE EXPLANATION :

- **Data Structure:** A list named `data` is created, containing dictionaries. Each dictionary holds a person's name and age.
- **Sorting:**
 - The built-in `sorted()` function is used to sort the list.
 - A `lambda` function is passed as the `key`, which extracts the `'age'` value from each dictionary for comparison.
- **Sorted Result:** The list is sorted in ascending order based on the `'age'` field.
- **Output:** The sorted list is printed, showing people ordered from youngest to oldest.