

NAME:PENDEM HARSHITHA

ID:2403A510C9

BATCH:05

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B.Tech		Assignment Type: Lab	Academic Year: 2025-2026
Course Coordinator Name STUDENT DETAILS:	Venkataramana Veeramsetty. PENDAM HARSHITHA Batch-05 2403a510C9		
Instructor(s) Name	Dr. V. Venkataramana (Co-Ordinator) Dr. T. Sampath Kumar Dr. Pramoda Patro Dr. Brij Kishor Tiwari Dr. J. Ravichander Dr. Mohammand Ali Shaik Dr. Anirodh Kumar Mr. S. Naresh Kumar Dr. RAJESH VELPULA Mr. Kundhan Kumar Ms. Ch. Rajitha Mr. M Prakash Mr. B. Raju Intern 1 (Dharmateja) Intern 2 (Sai Prasad) Intern 3 (Sowmya) NS_2 (Mounika)		
Course Code	24CS002PC215	Course Title	AI Assisted Coding
Year/Sem	II/I	Regulation	R24
Date and Day of Assignment	Week 1 - Monday	Time(s)	
Duration	2 Hours	Applicable to Batches	24CSBTB01 To 24CSBTB39
Assignment Number: 2.1 (Present assignment number) / 24 (Total number of assignments)			
Q.No.	Question	Expected Time to complete	
1	Lab 2: Exploring Additional AI Coding Tools – Gemini (Colab) and	Week 1 - Monday	

	<p>CursorAI</p> <p>LabObjectives:</p> <ul style="list-style-type: none">• To explore and evaluate the functionality of Google Gemini for AI-assisted coding within Google Colab.• To understand and use Cursor AI for code generation, explanation, and refactoring.• To compare outputs and usability between Gemini, GitHub Copilot, and Cursor AI.• To perform code optimization and documentation using AI tools. <p>LabOutcomes (LOs):</p> <p>After completing this lab, students will be able to:</p> <ul style="list-style-type: none">• Generate Python code using Google Gemini in Google Colab.• Analyze the effectiveness of code explanations and suggestions by Gemini.• Set up and use Cursor AI for AI-powered coding assistance.• Evaluate and refactor code using Cursor AI features.• Compare AI tool behavior and code quality across different platforms.	
	<p>TaskDescription#1</p> <ul style="list-style-type: none">• Use Google Gemini in Colab to write a Python function that reads a list of numbers and calculates the mean, minimum, and maximum values. <p>ExpectedOutput#1</p> <ul style="list-style-type: none">• Functional code with correct output and screenshot. <p>Task Description #1 • Use Google Gemini in Colab to write a Python function that reads a list of numbers and calculates the mean, minimum, and maximum values. Expected Output #1 • Functional code with correct output and screenshot.</p> <pre>▶ Start coding or generate with AI. ▶ def analyze_numbers(numbers): """ Calculates the mean, minimum, and maximum values from a list of numbers. Args: numbers: A list of numbers (integers or floats). Returns: A dictionary containing the mean, minimum, and maximum values. Returns None if the input list is empty. """ if not numbers: return None mean_value = sum(numbers) / len(numbers) min_value = min(numbers) max_value = max(numbers) return { "mean": mean_value, "minimum": min_value, "maximum": max_value }</pre>	

	<div><div><div><div><div></div><div></div></div><div><pre>mean_value = sum(numbers) / len(numbers) min_value = min(numbers) max_value = max(numbers) return { "mean": mean_value, "minimum": min_value, "maximum": max_value } # Example usage: my_list = [10, 20, 30, 40, 50] result = analyze_numbers(my_list) if result: print(f"Analysis of the list {my_list}:") print(f"Mean: {result['mean']}") print(f"Minimum: {result['minimum']}") print(f"Maximum: {result['maximum']}") else: print("The list is empty.")</pre></div><div><div>Analysis of the list [10, 20, 30, 40, 50]:</div><div>Mean: 30.0</div><div>Minimum: 10</div><div>Maximum: 50</div></div></div><div>Double-click (or enter) to edit</div></div></div>	
	<div><div>TaskDescription#2</div><div><ul style="list-style-type: none">Compare Gemini and Copilot outputs for a Python function that checks whether a number is an Armstrong number. Document the steps, prompts, and outputs.</div><div>ExpectedOutput#2</div><div><ul style="list-style-type: none">Side-by-side comparison table with observations and screenshots.</div><div>Python function that checks whether a number is an Armstrong number. Document the steps, prompts, and outputs.</div><div><div><div><div></div><div></div></div><div>Start coding or generate with AI.</div></div><div>Double-click (or enter) to edit</div><div><div><div><div></div><div></div></div><div><pre>[2] def is_armstrong_number(number): """ Checks if a number is an Armstrong number. An Armstrong number (or narcissistic number) is a number that is the sum of its own digits each raised to the power of the number of digits. Args: number: An integer. Returns: True if the number is an Armstrong number, False otherwise. """ # Convert the number to a string to easily access its digits num_str = str(number) num_digits = len(num_str) armstrong_sum = 0 # Calculate the sum of each digit raised to the power of the number of digits for digit in num_str:</pre></div></div></div></div></div>	

```
# Convert the number to a string to easily access its digits
num_str = str(number)
num_digits = len(num_str)
armstrong_sum = 0

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

# Check if the calculated sum is equal to the original number
return armstrong_sum == 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.

```
def is_armstrong(number):
```

 Python function that checks whether a number is an Armstrong number. Document the steps, prompts, and outputs.

Accept Close ↺ ↵

in\Desktop\B.Tech\First Year(1)\Python Programs\partterns.py

output:

```
num_str = str(number)
num_digits = len(num_str)
sum_of_powers = sum(int(digit) ** num_digits for digit in num_str)
return sum_of_powers == number
```

Feature	GeminiCode	CopilotCode
FunctionName	<code>is_armstrong_number (number)</code>	<code>is_armstrong (number)</code>
LogicStyle	Stringconversion+forloop	Arithmeticwith%andwhileloop
Docstring	Detailedwith Args&Returns	Minimalornone
Readability	High-commentedandwell-structured	Moderate-morecompact,sometimesnocomments
TestCases	Shown(153, 123)	Usuallyincludes153;othersneedprompting
OutputExample	153isanArmstrong number.123 isnotanArmstrong number.	Sameoutput

TaskDescription#3

	<ul style="list-style-type: none">• AskGeminitoexplainaPythonfunction(e.g.,is_prime(n)or is_palindrome(s)) line by line.• Chooseeitheraprime-checkingorpalindrome-checkingfunction and document the explanation provided by Gemini. <p>ExpectedOutput#3</p> <ul style="list-style-type: none">• Detailedexplanationwiththecodesnippetand Gemini’s response. <div><div>• Ask Gemini to explain a Python function (e.g., is_prime(n) or is_palindrome(s)) line by line. • Choose either a prime-checking or palindrome-checking function and document the explanation provided by Gemini.</div><div><div>Start coding or generate with AI.</div><div>[] Start coding or generate with AI.</div><div>Steps to check for an Armstrong number:<ol style="list-style-type: none">1. Define the function <code>is_armstrong_number(number)</code>: This function takes an integer <code>number</code> as input.2. Convert the number to a string: This allows easy iteration through the digits of the number.3. Get the number of digits: Determine the length of the string representation of the number.4. Initialize <code>armstrong_sum</code> to 0: This variable will store the sum of the digits raised to the power of the number of digits.5. Iterate through each digit:<ul style="list-style-type: none">◦ Convert the digit back to an integer.◦ Raise the integer digit to the power of the total number of digits.◦ Add the result to <code>armstrong_sum</code>.6. Compare the sum with the original number: If <code>armstrong_sum</code> is equal to the original <code>number</code>, it is an Armstrong number.7. Return True or False: Based on the comparison result.</div><div><div>0s</div><div><div>def is_palindrome(s): """ Checks if a string is a palindrome. A palindrome is a string that reads the same forwards and backwards. Args: s: The input string. Returns: True if the string is a palindrome, False otherwise. """ # Remove spaces and convert to lowercase for case-insensitive check s = s.replace(" ", "").lower() # Compare the string with its reverse return s == s[::-1] # Example usage: text1 = "racecar" text2 = "hello world" text3 = "Madam" print(f"{text1} is a palindrome: {is_palindrome(text1)}") print(f"{text2} is a palindrome: {is_palindrome(text2)}") print(f"{text3} is a palindrome: {is_palindrome(text3)}")</div><div><div>↗</div><div>'racecar' is a palindrome: True 'hello world' is a palindrome: False 'Madam' is a palindrome: True</div></div></div></div></div></div>	
	<p>TaskDescription#4</p> <ul style="list-style-type: none">• Install and configure Cursor AI. Use it to generate a Python function(e.g.,sumofthefirstNnaturalnumbers)andtestits	

	<p>output.</p> <ul style="list-style-type: none">Optionally, compare Cursor AI's generated code with Gemini's output. <p>Expected Output #4</p> <ul style="list-style-type: none">Screenshots of Cursor AI setup, prompts used, and generated code with output. <p>• Install and configure Cursor AI. Use it to generate a Python function (e.g., sum of the first N natural numbers) and test its output. • Optionally, compare Cursor AI's generated code with Gemini's output. Expected Output #4 • Screenshots of Cursor AI setup, prompts used and generated code with output.</p> <div><div>[] Start coding or generate with AI.</div><div><div>def sum_of_first_n_natural_numbers(n): """ Calculates the sum of the first N natural numbers. Args: n: An integer representing the number of natural numbers. Returns: The sum of the first N natural numbers. """ if n < 1: return 0 else: return n * (n + 1) // 2 # Example usage: n_value = 10</div></div><div><div>def sum_of_first_n_natural_numbers(n): """ Calculates the sum of the first N natural numbers. Args: n: An integer representing the number of natural numbers. Returns: The sum of the first N natural numbers. """ if n < 1: return 0 else: return n * (n + 1) // 2 # Example usage: n_value = 10 sum_n = sum_of_first_n_natural_numbers(n_value) print(f"The sum of the first {n_value} natural numbers is: {sum_n}") n_value = 5 sum_n = sum_of_first_n_natural_numbers(n_value) print(f"The sum of the first {n_value} natural numbers is: {sum_n}")</div><div>The sum of the first 10 natural numbers is: 55 The sum of the first 5 natural numbers is: 15</div></div></div>	
	<p>Task Description #5</p> <ul style="list-style-type: none">Students need to write a Python program to calculate the sum of odd numbers and even numbers in a given tuple.Refactor the code to improve logic and readability. <p>Expected Output #5</p>	

- Student-writtenrefactoredcodewithexplanationsandoutput screenshots

Python program to calculate the sum of odd numbers and even numbers in a given tuple. • Refactor the code to improve logic and readability

```
def sum_odd_even(numbers_tuple):  
    """  
    Calculates the sum of odd and even numbers in a tuple.  
  
    Args:  
        numbers_tuple: A tuple of numbers (integers or floats).  
  
    Returns:  
        A dictionary containing the sum of odd numbers and the sum of even numbers.  
    """  
    sum_even = 0  
    sum_odd = 0  
  
    for number in numbers_tuple:  
        # Check if the number is an integer before performing modulo operation  
        if isinstance(number, int):  
            if number % 2 == 0:  
                sum_even += number  
            else:  
                sum_odd += number  
        # You might want to handle non-integer types differently,  
        # for now, we'll skip them for the odd/even check.  
  
    return {"sum_even": sum_even, "sum_odd": sum_odd}
```

```
    sum_odd += number  
    # You might want to handle non-integer types differently,  
    # for now, we'll skip them for the odd/even check.  
  
    return {"sum_even": sum_even, "sum_odd": sum_odd}  
  
# Example usage:  
my_tuple = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)  
result = sum_odd_even(my_tuple)  
  
print(f"Original tuple: {my_tuple}")  
print(f"Sum of even numbers: {result['sum_even']}")  
print(f"Sum of odd numbers: {result['sum_odd']}")  
  
my_tuple_2 = (11, 22, 33, 44, 55, 66)  
result_2 = sum_odd_even(my_tuple_2)  
  
print(f"\nOriginal tuple: {my_tuple_2}")  
print(f"Sum of even numbers: {result_2['sum_even']}")  
print(f"Sum of odd numbers: {result_2['sum_odd']}")
```

```
Original tuple: (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)  
Sum of even numbers: 30  
Sum of odd numbers: 25
```

```
Original tuple: (11, 22, 33, 44, 55, 66)  
Sum of even numbers: 132  
Sum of odd numbers: 99
```


Note:

- Students must submit a single Word document including:
 - Prompts used for AI tools

- Copilot/Gemini/Cursor outputs
- Code explanations
- Screenshots of outputs and environments

Evaluation Criteria:

Criteria	Max Marks
Successful Use of Gemini in Colab (Task #1 & #2)	1.0
Code Explanation Accuracy (Gemini) (Task #3)	0.5
Cursor AI Setup and Usage (Task #4)	0.5
Refactoring and Improvement Analysis (Task #5)	0.5
Total	2.5 Marks



```


def analyze_numbers(numbers):
    mean_value = sum(numbers) / len(numbers)
    min_value = min(numbers)
    max_value = max(numbers)

    return {
        "mean": mean_value,
        "minimum": min_value,
        "maximum": max_value
    }

# Example usage:
my_list = [10, 20, 30, 40, 50]
result = analyze_numbers(my_list)

if result:
    print(f"Analysis of the list {my_list}:")
    print(f"Mean: {result['mean']}")
    print(f"Minimum: {result['minimum']}")
    print(f"Maximum: {result['maximum']}")
else:
    print("The list is empty.")

```



```

Analysis of the list [10, 20, 30, 40, 50]:
Mean: 30.0
Minimum: 10
Maximum: 50

```

Double-click (or enter) to edit


TaskDescription#2

- Compare Gemini and Copilot outputs for a Python function that checks whether a number is an Armstrong number. Document the steps, prompts, and outputs.


ExpectedOutput#2

- Side-by-side comparison table with observations and screenshots.

Python function that checks whether a number is an Armstrong number. Document the steps, prompts, and outputs.

 Start coding or generate with AI.

Double-click (or enter) to edit


0s

```
[2] def is_armstrong_number(number):  
    """  
    Checks if a number is an Armstrong number.  
  
    An Armstrong number (or narcissistic number) is a number that is the sum  
    of its own digits each raised to the power of the number of digits.  
  
    Args:  
        number: An integer.  
  
    Returns:  
        True if the number is an Armstrong number, False otherwise.  
    """  
    # Convert the number to a string to easily access its digits  
    num_str = str(number)  
    num_digits = len(num_str)  
    armstrong_sum = 0  
  
    # Calculate the sum of each digit raised to the power of the number of digits  
    for digit in num_str:
```

```
# Convert the number to a string to easily access its digits
num_str = str(number)
num_digits = len(num_str)
armstrong_sum = 0

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

# Check if the calculated sum is equal to the original number
return armstrong_sum == 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.

```
def is_armstrong(number):
```

 Python function that checks whether a number is an Armstrong number. Document the steps, prompts, and outputs.

GPT-4.1

Accept Close ↺ ▼

```
ir\Desktop\B.Tech\First Year(1)\Python Programs\partterns.py
```

οὐκ ἔστιν:

```
num_str = str(number)
num_digits = len(num_str)
sum_of_powers = sum(int(digit) ** num_digits for digit in num_str)
return sum_of_powers == number
```

Feature	GeminiCode	CopilotCode
FunctionName	<code>is_armstrong_number(number)</code>	<code>is_armstrong(number)</code>
LogicStyle	Stringconversion+forloop	Arithmeticwith%andwhileloop
Docstring	Detailedwith Args&Returns	Minimalornone
Readability	High-commentedandwell-structured	Moderate-morecompact,sometimesnocomments
TestCases	<code>Shown(153, 123)</code>	<code>Usuallyincludes153;othersneedprompting</code>
OutputExample	<code>153isanArmstrong number.123 isnotanArmstrong number.</code>	Sameoutput

TaskDescription#3

	<ul style="list-style-type: none">• AskGeminitoexplainaPythonfunction(e.g.,is_prime(n)or is_palindrome(s)) line by line.• Chooseeitheraprime-checkingorpalindrome-checkingfunction and document the explanation provided by Gemini. <p>ExpectedOutput#3</p> <ul style="list-style-type: none">• Detailedexplanationwiththecodesnippetand Gemini’s response. <div><div>• Ask Gemini to explain a Python function (e.g., is_prime(n) or is_palindrome(s)) line by line. • Choose either a prime-checking or palindrome-checking function and document the explanation provided by Gemini.</div><div><div>Start coding or generate with AI.</div><div>[] Start coding or generate with AI.</div></div><div>Steps to check for an Armstrong number:<ol style="list-style-type: none">1. Define the function is_armstrong_number(number): This function takes an integer number as input.2. Convert the number to a string: This allows easy iteration through the digits of the number.3. Get the number of digits: Determine the length of the string representation of the number.4. Initialize armstrong_sum to 0: This variable will store the sum of the digits raised to the power of the number of digits.5. Iterate through each digit:<ul style="list-style-type: none">◦ Convert the digit back to an integer.◦ Raise the integer digit to the power of the total number of digits.◦ Add the result to armstrong_sum.6. Compare the sum with the original number: If armstrong_sum is equal to the original number, it is an Armstrong number.7. Return True or False: Based on the comparison result.</div><div><div>0s</div><div><div>def is_palindrome(s): """ Checks if a string is a palindrome. A palindrome is a string that reads the same forwards and backwards. Args: s: The input string. Returns: True if the string is a palindrome, False otherwise. """ # Remove spaces and convert to lowercase for case-insensitive check s = s.replace(" ", "").lower() # Compare the string with its reverse return s == s[::-1] # Example usage: text1 = "racecar" text2 = "hello world" text3 = "Madam" print(f"{text1} is a palindrome: {is_palindrome(text1)}") print(f"{text2} is a palindrome: {is_palindrome(text2)}") print(f"{text3} is a palindrome: {is_palindrome(text3)}")</div><div><div>↗</div><div>'racecar' is a palindrome: True 'hello world' is a palindrome: False 'Madam' is a palindrome: True</div></div></div></div></div>	
	<p>TaskDescription#4</p> <ul style="list-style-type: none">• Install and configure Cursor AI. Use it to generate a Python function(e.g.,sumofthefirstNnaturalnumbers)andtestits	

	<p>output.</p> <ul style="list-style-type: none">Optionally, compare Cursor AI's generated code with Gemini's output. <p>Expected Output #4</p> <ul style="list-style-type: none">Screenshots of Cursor AI setup, prompts used, and generated code with output. <p>• Install and configure Cursor AI. Use it to generate a Python function (e.g., sum of the first N natural numbers) and test its output. • Optionally, compare Cursor AI's generated code with Gemini's output. Expected Output #4 • Screenshots of Cursor AI setup, prompts used and generated code with output.</p> <div><div>[] Start coding or generate with AI.</div><div><div>def sum_of_first_n_natural_numbers(n): """ Calculates the sum of the first N natural numbers. Args: n: An integer representing the number of natural numbers. Returns: The sum of the first N natural numbers. """ if n < 1: return 0 else: return n * (n + 1) // 2 # Example usage: n_value = 10</div></div><div><div>def sum_of_first_n_natural_numbers(n): """ Calculates the sum of the first N natural numbers. Args: n: An integer representing the number of natural numbers. Returns: The sum of the first N natural numbers. """ if n < 1: return 0 else: return n * (n + 1) // 2 # Example usage: n_value = 10 sum_n = sum_of_first_n_natural_numbers(n_value) print(f"The sum of the first {n_value} natural numbers is: {sum_n}") n_value = 5 sum_n = sum_of_first_n_natural_numbers(n_value) print(f"The sum of the first {n_value} natural numbers is: {sum_n}")</div><div>The sum of the first 10 natural numbers is: 55 The sum of the first 5 natural numbers is: 15</div></div></div>	
	<p>Task Description #5</p> <ul style="list-style-type: none">Students need to write a Python program to calculate the sum of odd numbers and even numbers in a given tuple.Refactor the code to improve logic and readability. <p>Expected Output #5</p>	

- Student-written refactored code with explanations and output screenshots

Python program to calculate the sum of odd numbers and even numbers in a given tuple. • Refactor the code to improve logic and readability

```
def sum_odd_even(numbers_tuple):
    """
    Calculates the sum of odd and even numbers in a tuple.

    Args:
        numbers_tuple: A tuple of numbers (integers or floats).

    Returns:
        A dictionary containing the sum of odd numbers and the sum of even numbers.
    """
    sum_even = 0
    sum_odd = 0

    for number in numbers_tuple:
        # Check if the number is an integer before performing modulo operation
        if isinstance(number, int):
            if number % 2 == 0:
                sum_even += number
            else:
                sum_odd += number
        # You might want to handle non-integer types differently,
        # for now, we'll skip them for the odd/even check.

    return {"sum_even": sum_even, "sum_odd": sum_odd}
```

```
sum_odd += number
# You might want to handle non-integer types differently,
# for now, we'll skip them for the odd/even check.

return {"sum_even": sum_even, "sum_odd": sum_odd}

# Example usage:
my_tuple = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
result = sum_odd_even(my_tuple)

print(f"Original tuple: {my_tuple}")
print(f"Sum of even numbers: {result['sum_even']}")
print(f"Sum of odd numbers: {result['sum_odd']}")

my_tuple_2 = (11, 22, 33, 44, 55, 66)
result_2 = sum_odd_even(my_tuple_2)

print(f"\nOriginal tuple: {my_tuple_2}")
print(f"Sum of even numbers: {result_2['sum_even']}")
print(f"Sum of odd numbers: {result_2['sum_odd']}")
```

```
Original tuple: (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
Sum of even numbers: 30
Sum of odd numbers: 25
```

```
Original tuple: (11, 22, 33, 44, 55, 66)
Sum of even numbers: 132
Sum of odd numbers: 99
```

Note:

- Students must submit a single Word document including:
 - Prompts used for AI tools

- Copilot/Gemini/Cursor outputs
- Code explanations
- Screenshots of outputs and environments

Evaluation Criteria:

Criteria	Max Marks
Successful Use of Gemini in Colab (Task#1 & #2)	1.0
Code Explanation Accuracy (Gemini) (Task#3)	0.5
Cursor AI Setup and Usage (Task#4)	0.5
Refactoring and Improvement Analysis (Task#5)	0.5
Total	2.5 Marks