

ASSIGNMENT-2

NAME:-P LAHARI

BATCH-16

HTNO:-2303A51071

Task 1: Book Class Generation (Using Cursor AI)

Python Code: Book Class

The screenshot shows the Thonny Python IDE interface. The top window displays the Python code for a `Book` class. The bottom window, titled "Shell", shows the output of running the code.

```
Thonny - <untitled> @ 5:29
File Edit View Run Tools Help
<untitled> * x
1 # Book class representing a simple library book
2 class Book:
3     def __init__(self, title, author):
4         self.title = title
5         self.author = author
6     def summary(self):
7         return f"Title: {self.title}, Author: {self.author}"
8
9 # Example usage
10 book1 = Book("Artificial Intelligence", "Stuart Russell")
11 print(book1.summary())
12

Shell x
>>> %Run -c $EDITOR_CONTENT
Title: Artificial Intelligence, Author: Stuart Russell
>>>
```

Task 2: Sorting Dictionaries with AI

◆ Using Google Gemini

The screenshot shows the Thonny IDE interface. The top menu bar includes File, Edit, View, Run, Tools, and Help. Below the menu is a toolbar with icons for file operations like Open, Save, and Run. The main window has two tabs: <untitled> and Shell. The <untitled> tab contains the following Python code:

```
1 # Sorting a list of dictionaries by age using Gemini-generated code
2 users = [
3     {"name": "Ravi", "age": 25},
4     {"name": "Anita", "age": 22},
5     {"name": "Suresh", "age": 30}
6 ]
7 sorted_users = sorted(users, key=lambda x: x["age"])
8 print(sorted_users)
9
```

The Shell tab shows the output of running the code:

```
>>> %Run -c $EDITOR_CONTENT
[{'name': 'Anita', 'age': 22}, {'name': 'Ravi', 'age': 25}, {'name': 'Suresh', 'age': 30}]
>>>
```

Using Cursor AI

The screenshot shows the Thonny Python IDE interface. The top menu bar includes File, Edit, View, Run, Tools, and Help. Below the menu is a toolbar with icons for file operations like Open, Save, and Run. A central code editor window titled '<untitled>' contains the following Python code:

```
1 # Sorting dictionaries by age with improved readability
2 users = [
3     {"name": "Ravi", "age": 25},
4     {"name": "Anita", "age": 22},
5     {"name": "Suresh", "age": 30}
6 ]
7 def sort_by_age(user_list):
8     return sorted(user_list, key=lambda user: user["age"])
9
10 print(sort_by_age(users))
11
```

Below the code editor is a 'Shell' tab showing the output of running the script:

```
>>> %Run -c $EDITOR_CONTENT
[{'name': 'Anita', 'age': 22}, {'name': 'Ravi', 'age': 25}, {'name': 'Suresh', 'age': 30}]
>>>
```

Comparison (Short Note):

The solution generated by Gemini is short and direct, making it quick to understand for simple tasks. Cursor AI, on the other hand, focused more on clean structure by using a function, which improves readability and makes the code reusable. While both approaches have similar performance, the Cursor AI version is more suitable for larger projects where maintainability and clarity are important.

Task 3: Calculator Using Functions (Gemini)

Calculator Code

The screenshot shows the Thonny Python IDE interface. The top menu bar includes File, Edit, View, Run, Tools, and Help. Below the menu is a toolbar with icons for file operations like Open, Save, and Run. The main code editor window is titled '<untitled>' and contains the following Python code:

```
1 # Basic calculator using functions
2 def add(a, b):
3     return a + b
4 def subtract(a, b):
5     return a - b
6 def multiply(a, b):
7     return a * b
8 def divide(a, b):
9     if b == 0:
10        return "Division by zero is not allowed"
11    return a / b
12 # Example usage
13 x = int(input("Enter first number: "))
14 y = int(input("Enter second number: "))
15 print("Addition:", add(x, y))
16 print("Subtraction:", subtract(x, y))
17 print("Multiplication:", multiply(x, y))
18 print("Division:", divide(x, y))
19
```

Below the code editor is a shell window titled 'Shell' which displays the program's output:

```
>>> %Run -c $EDITOR_CONTENT
Enter first number: 5
Enter second number: 18
Addition: 23
Subtraction: -13
Multiplication: 90
Division: 0.2777777777777778
```

Task 4: Armstrong Number Optimization

◆ Version 1: Gemini-Generated Armstrong Program

The screenshot shows the Thonny Python IDE interface. The top window is the code editor with the title "Thonny - <untitled> @ 16:1". It contains the following Python code:

```
1 # Armstrong number check (basic version)
2
3 num = int(input("Enter a number: "))
4 temp = num
5 sum = 0
6
7 while temp > 0:
8     digit = temp % 10
9     sum += digit ** 3
10    temp //= 10
11
12 if sum == num:
13     print(num, "is an Armstrong number")
14 else:
15     print(num, "is not an Armstrong number")
16
```

The bottom window is the shell, titled "Shell". It shows the output of running the script:

```
>>> %Run -c $EDITOR_CONTENT
Enter a number: 153
153 is an Armstrong number
>>> |
```

Version 2: Optimized Using Cursor AI

The screenshot shows the Thonny Python IDE interface. The top window is titled 'Thonny - <untitled> @ 13:1' and contains a code editor with the following Python script:

```
1 # Optimized Armstrong number program with better readability
2
3 num = int(input("Enter a number: "))
4 digits = str(num)
5 power = len(digits)
6
7 armstrong_sum = sum(int(digit) ** power for digit in digits)
8
9 if armstrong_sum == num:
10     print(f"{num} is an Armstrong number")
11 else:
12     print(f"{num} is not an Armstrong number")
13
```

The bottom window is titled 'Shell' and shows the following interaction:

```
>>> %Run -c $EDITOR_CONTENT
Enter a number: 142
142 is not an Armstrong number
>>> |
```

Summary of Improvements:

- **Reduced lines of code**
- **Removed unnecessary variables**
- **Improved readability using Python built-in functions**
- **Works for Armstrong numbers of any length**