

# Assignment – 6.5

2303A51599

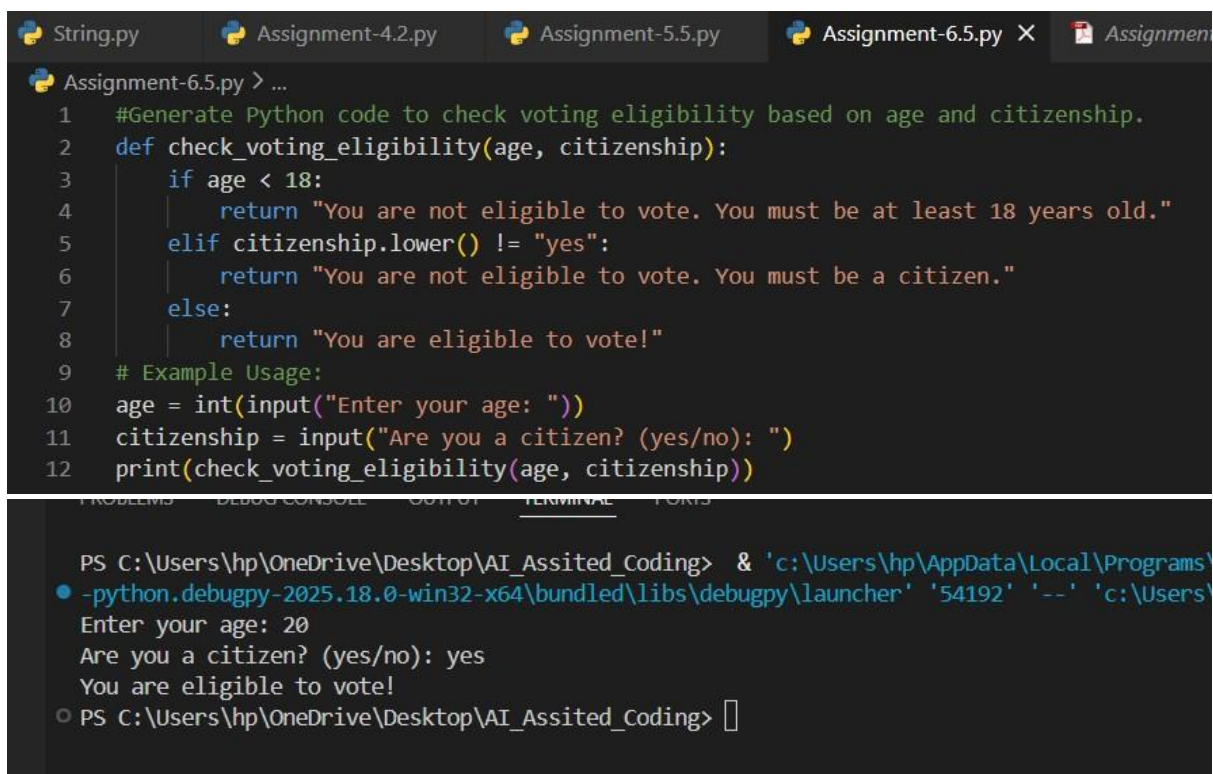
Batch - 29

Task 1: Use an AI tool to generate eligibility logic

**PROMPT :**

**Generate Python code to check voting eligibility based on age and citizenship.**

**CODE and OUTPUT :**



```
String.py Assignment-4.2.py Assignment-5.5.py Assignment-6.5.py X Assignment-6.5.py
Assignment-6.5.py > ...
1 #Generate Python code to check voting eligibility based on age and citizenship.
2 def check_voting_eligibility(age, citizenship):
3     if age < 18:
4         return "You are not eligible to vote. You must be at least 18 years old."
5     elif citizenship.lower() != "yes":
6         return "You are not eligible to vote. You must be a citizen."
7     else:
8         return "You are eligible to vote!"
9 # Example Usage:
10 age = int(input("Enter your age: "))
11 citizenship = input("Are you a citizen? (yes/no): ")
12 print(check_voting_eligibility(age, citizenship))

PROBLEMS DEBUG CONSOLE OUTPUT TERMINAL PORTS
PS C:\Users\hp\OneDrive\Desktop\AI_Assited_Coding> & 'c:\Users\hp\AppData\Local\Programs\Python\Python311\python.exe' -python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher '54192' '--' 'c:\Users\hp\OneDrive\Desktop\AI_Assited_Coding\Assignment-6.5.py'
Enter your age: 20
Are you a citizen? (yes/no): yes
You are eligible to vote!
PS C:\Users\hp\OneDrive\Desktop\AI_Assited_Coding> 
```

**Justification :**

In this task, I used an AI tool to generate Python code for checking voting eligibility based on age and citizenship. The AI correctly used conditional statements with logical operators to verify both conditions. I tested the program with different age and citizenship values to confirm accurate output. This task helped me understand how AI can quickly generate correct conditional logic, but I also realized the importance of checking boundary cases and validating inputs.

Task 2: Use an AI tool to process strings using loops.

### PROMPT :

Generate Python code to count vowels and consonants in a string using a loop.

### CODE and OUTPUT :

```
13
14 #Generate Python code to count vowels and consonants in a string using a loop(use loops only).
15 def count_vowels_consonants(s):
16     vowels = 'aeiouAEIOU'
17     vowel_count = 0
18     consonant_count = 0
19     for char in s:
20         if char.isalpha(): # Check if the character is an alphabet
21             if char in vowels:
22                 vowel_count += 1
23             else:
24                 consonant_count += 1
25     return vowel_count, consonant_count
26 # Example Usage:
27 input_string = input("Enter a string: ")
28 vowels, consonants = count_vowels_consonants(input_string)
29 print(f"Number of vowels: {vowels}")
30 print(f"Number of consonants: {consonants}")
31
--
PS C:\Users\hp\OneDrive\Desktop\AI_Assited Coding> c::; cd 'c:\Users\hp\OneDrive\Desktop\AI_Assited Coding\Assignment-6.5.py'
14\python.exe' 'c:\Users\hp\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\python.exe'
Enter a string: Apple
Number of vowels: 2
Number of consonants: 3
PS C:\Users\hp\OneDrive\Desktop\AI_Assited Coding> 
```

### Justification :

For this task, the AI generated a loop-based program to count vowels and consonants in a string. The for loop was used to iterate through each character, and nested conditions were applied to classify vowels and consonants. I verified the output using multiple test strings. I observed that AI-generated code is efficient, but understanding how the loop and counters work is important to avoid logical errors.

Task 3: Use an AI tool to generate a complete program using classes, loops, and conditionals.

### PROMPT :

Generate a Python program for a library management system using classes, loops, and conditional statements.

## CODE and OUTPUT :

```
Assignment-6.5.py > ...
32 #Generate a Python program for a library management system using classes, loops, and conditional statements
33 class Book:
34     def __init__(self, title, author):
35         self.title = title
36         self.author = author
37         self.is_available = True
38 class Library:
39     def __init__(self):
40         self.books = []
41     def add_book(self, book):
42         self.books.append(book)
43     def borrow_book(self, title):
44         for book in self.books:
45             if book.title == title:
46                 if book.is_available:
47                     book.is_available = False
48                     return f"You have borrowed '{book.title}' by {book.author}."
49                 else:
50                     return f"Sorry, '{book.title}' is currently not available."
51         return "Book not found in the library."
52     def return_book(self, title):
53         for book in self.books:
54             if book.title == title:
55                 if not book.is_available:
56                     book.is_available = True
57                     return f"You have returned '{book.title}'. Thank you!"
58                 else:
59                     return f"'{book.title}' was not borrowed."
60         return "Book not found in the library."
61 # Example Usage:
62 library = Library()
63 book1 = Book("The Great Gatsby", "F. Scott Fitzgerald")
64 book2 = Book("To Kill a Mockingbird", "Harper Lee")
65 library.add_book(book1)
66 library.add_book(book2)
67 print(library.borrow_book("The Great Gatsby"))
68 print(library.borrow_book("The Great Gatsby"))
69 print(library.return_book("The Great Gatsby"))
70 print(library.return_book("The Great Gatsby"))
71
26 def extract_numbers(s):
PROBLEMS  DEBUG CONSOLE  OUTPUT  TERMINAL  PORTS
● PS C:\Users\hp\OneDrive\Desktop\AI_Assited_Coding> c::; cd 'c:\Users\hp\OneDrive\
scode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launch
You have borrowed 'The Great Gatsby' by F. Scott Fitzgerald.
Sorry, 'The Great Gatsby' is currently not available.
You have returned 'The Great Gatsby'. Thank you!
'The Great Gatsby' was not borrowed.
○ PS C:\Users\hp\OneDrive\Desktop\AI_Assited_Coding> []
```

Justification :

In this task, AI generated a complete program using classes, loops, and conditional statements. The class structure was well organized with separate methods for adding, displaying, and removing books. The menu-driven loop allowed continuous user interaction. After reviewing the code, I noticed that data is not stored permanently, which is a limitation. This task showed that AI is useful for building program structure, but optimization and enhancement depend on the programmer.

Task 4: Use an AI tool to generate an attendance management class.

#### PROMPT :

Generate a Python class to mark and display student attendance using loops.

#### CODE and OUTPUT :

```
72
73 #Generate a Python class to mark and display student attendance using loops (Use only loops ).
74 class Student:
75     def __init__(self, name):
76         self.name = name
77         self.attendance = []
78     def mark_attendance(self, day):
79         self.attendance.append(day)
80     def display_attendance(self):
81         if not self.attendance:
82             return f"{self.name} has no attendance records."
83         attendance_record = f"{self.name}'s Attendance:\n"
84         for day in self.attendance:
85             attendance_record += f"- {day}\n"
86         return attendance_record
87 #Example Usage:
88 student = Student("Alice")
89 student.mark_attendance("Monday")
90 student.mark_attendance("Wednesday")
91 print(student.display_attendance())
92
```

```
PS C:\Users\hp\OneDrive\Desktop\AI_Assited_Coding> c::; cd 'c:\Users\hp\OneDr
\launcher' '53986' '--' 'c:\Users\hp\OneDrive\Desktop\AI_Assited_Coding\Assig
Alice's Attendance:
- Monday
- Wednesday
```

#### Justification:

The AI-generated attendance system used a class and dictionary to manage student records. The logic for marking and displaying attendance was correct and easy to understand. I tested the program with different student names to verify the output. However, I identified that the program lacks input validation for attendance status. This highlights the need for careful review of AI-generated solutions.

Task 5: Use an AI tool to complete a navigation menu.

**PROMPT :**

Generate a Python program using loops and conditionals to simulate an ATM menu.

**CODE and OUTPUT :**



```

88
89
90 #Generate a Python program using loops and conditionals to simulate an ATM menu.
91 class ATM:
92     def __init__(self, balance=0):
93         self.balance = balance
94     def display_menu(self):
95         print("Welcome to the ATM!")
96         print("1. Check Balance")
97         print("2. Deposit")
98         print("3. Withdraw")
99         print("4. Exit")
100     def check_balance(self):
101         return f"Your current balance is: ${self.balance:.2f}"
102     def deposit(self, amount):
103         if amount > 0:
104             self.balance += amount
105             return f"You have deposited: ${amount:.2f}. New balance: ${self.balance:.2f}"
106         else:
107             return "Deposit amount must be positive."
108     def withdraw(self, amount):
109         if amount > self.balance:
110             return "Insufficient funds."
111         elif amount <= 0:
112             return "Withdrawal amount must be positive."
113         else:
114             self.balance -= amount
115             return f"You have withdrawn: ${amount:.2f}. New balance: ${self.balance:.2f}"
116 # Example Usage:
117 atm = ATM(1000) # Initialize ATM with a balance of $1000
118 while True:
119     atm.display_menu()
120     choice = input("Please select an option (1-4): ")
121     if choice == '1':
122         print(atm.check_balance())
123     elif choice == '2':
124         amount = float(input("Enter the amount to deposit: "))
125         print(atm.deposit(amount))
126     elif choice == '3':
127         amount = float(input("Enter the amount to withdraw: "))
128         print(atm.withdraw(amount))
129
130     elif choice == '4':
131         print("Thank you for using the ATM. Goodbye!")
132     else:
133         print("Invalid option. Please try again.")
134

```

```

PS C:\Users\hp\OneDrive\Desktop\AI_Assited_Coding> c;; cd 'c:\Users\hp\OneD
scode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\
Welcome to the ATM!
1. Check Balance
2. Deposit
3. Withdraw
4. Exit
Please select an option (1-4): 

```

### Justification:

In this task, AI generated a menu-driven ATM program using loops and conditional statements. The logic for checking balance, deposit, and withdrawal was correct. The program properly

handled insufficient balance conditions. However, I observed that negative inputs and invalid values are not restricted. This shows that while AI provides a strong base implementation, developers must review and improve the code for better reliability and security.