

AI-ASSISTED CODING

ASSIGNMENT-10.3

2403A51267

BATCH-12

TASK-1:

Provide AI with the following nested conditional code and ask it to simplify and refactor for readability

PROMPT

write the following nested conditional code and ask it to simplify and refactor for readability.

Nested code:

```
ai.py > ...
1  # write the following nested conditional code and ask it to simplify and refactor for readability.
2  def discount(price, category):
3      if category == "student":
4          if price > 1000:
5              return price * 0.9
6          else:
7              return price * 0.95
8      else:
9          if price > 2000:
10             return price * 0.85
11          else:
12             return price * 0.9
13  discount(1200, "student")
14
```

Output code:

```
ai.py > ...
1 # write the following nested conditional code and ask it to simplify and refactor for readability.
2 def student_discount(price):
3     return price * 0.9 if price > 1000 else price * 0.95
4
5 def regular_discount(price):
6     return price * 0.85 if price > 2000 else price
7
8 discount_rules = {
9     "student": student_discount,
10    "regular": regular_discount
11 }
12
13 def discount(price, category):
14     apply_discount = discount_rules.get(category, regular_discount)
15     return apply_discount(price)
16 print(discount(1200, "student"))
17 print(discount(800, "student"))
18 print(discount(2500, "regular"))
19 print(discount(1500, "regular"))
20 print(discount(1800, "unknown"))
21
```

TASK-2 :

Give AI this messy loop and ask it to refactor and optimize.

PROMPT

#write this messy loop and ask it to refactor and optimize.

Messy code:

```
ai.py > ...
1 #write this messy loop and ask it to refactor and optimize.
2 def find_common(a, b):
3     res = []
4     for i in a:
5         for j in b:
6             if i == j:
7                 res.append(i)
8     return res
9 find_common([1,2,3,4],[3,4,5,6])
```

Output code :

```
#write this messy loop and ask it to refactor and optimize.  
def find_common(a, b):  
    return list(set(a) & set(b))  
print(find_common([1,2,3,4],[3,4,5,6]))
```

TASK-3:

Provide this class with poor readability and ask AI to improve:

- Naming conventions
- Encapsulation
- Readability & maintainability

PROMPT

#Write this class with poor readability and ask AI to improve

Naming conventions

Encapsulation

Readability & maintainability

Poor readability code:

```
ai.py > ...
1  #Write this class with poor readability and ask AI to improve
2  # Naming conventions
3  # Encapsulation
4  # Readability & maintainability Python Script
5  class emp:
6      def __init__(self,n,s):
7          self.n=n
8          self.s=s
9      def inc(self,p):
10         self.s=self.s+(self.s*p/100)
11     def pr(self):
12         print("emp:",self.n,"salary:",self.s)
13 e1=emp("ajay",10000)
14 e1.inc(10)
15 e1.pr()
```

Output code :

```
ai.py > ...
1  #Write this class with poor readability and ask AI to improve
2  # Naming conventions
3  # Encapsulation
4  # Readability & maintainability Python Script
5  class Employee:
6      def __init__(self, name: str, salary: float):
7          self._name = name
8          self._salary = salary
9
10     def increase_salary(self, percent: float):
11         self._salary += self._salary * (percent / 100)
12
13     def display_info(self):
14         print(f"Employee: {self._name}, Salary: ${self._salary:,.2f}")
15
16     def get_name(self) -> str:
17         return self._name
18
19     def get_salary(self) -> float:
20         return self._salary
21 emp1 = Employee("Alice Johnson", 50000)
22 emp1.display_info()
23 emp1.increase_salary(10)
24 emp1.display_info()
25
26
```

TASK-4 :

Give AI this long unstructured function and let it modularize into smaller helper functions

Prompt

Write the long unstructured function and let it modularize into smaller helper functions

Unstructured function :

```
#Write the long unstructured function and let it modularize into smaller helper functions
def process_scores(scores):
    total = 0
    for s in scores:
        total += s
    avg = total / len(scores)

    highest = scores[0]
    for s in scores:
        if s > highest:
            highest = s

    lowest = scores[0]
    for s in scores:
        if s < lowest:
            lowest = s

    print("Average:", avg)
    print("Highest:", highest)
    print("Lowest:", lowest)
    process_scores([10, 20, 30, 40, 50])
```

Output code :

```
.py / ...
#Write the long unstructured function and let it modularize into smaller helper functions
def calculate_average(scores):
    return sum(scores) / len(scores) if scores else 0

def find_highest(scores):
    return max(scores) if scores else None

def find_lowest(scores):
    return min(scores) if scores else None

def process_scores(scores):
    if not scores:
        print("No scores provided.")
        return

    avg = calculate_average(scores)
    highest = find_highest(scores)
    lowest = find_lowest(scores)

    print(f"Average: {avg:.2f}")
    print(f"Highest: {highest}")

    print(f"Lowest: {lowest}")
    scores = [85, 90, 78, 92, 88]
    process_scores(scores)
```

TASK-5 :

Provide AI with this faulty code and ask it to improve error handling, naming, and readability

PROMPT

#Write this faulty code and ask it to improve error handling, naming, and readability

Faulty code :

```
ai.py > ...
1 #Write this faulty code and ask it to improve error handling, naming, and readability
2 def div(a,b):
3     return a/b
4
5 print(div(10,0)) # ZeroDivisionError
6
```

Output code :

```
ai.py > divide_numbers
1 #Write this faulty code and ask it to improve error handling, naming, and readability
2 def divide_numbers(numerator, denominator):
3     try:
4         return numerator / denominator
5     except ZeroDivisionError:
6         return "Error: Cannot divide by zero."
7
8 result = divide_numbers(10, 0)
9 print(result)
10
```

TASK-6:

Use AI to simplify overly complex logic.

Sample Input Code

PROMPT

write a python program to simplify overly complex logic.

Sample Input Code

#Example : grade = 85

Complex logic :

```
#write a python program to simplify overly complex logic.
#Example : grade = 85
def grade(score):
    if score >= 90:
        return "A"
    else:
        if score >= 80:
            return "B"
        else:
            if score >= 70:
                return "C"
            else:
                if score >= 60:
                    return "D"
                else:
                    return "F"
grade(85)
```

Output code :

```
ai.py > ...
1  #write a python program to simplify overly complex logic.
2  #Example : grade = 85
3  def grade(score):
4      if not (0 <= score <= 100):
5          return "Invalid score"
6      return (
7          "A" if score >= 90 else
8          "B" if score >= 80 else
9          "C" if score >= 70 else
10         "D" if score >= 60 else
11         "F"
12     )
13  print(grade(85))
14
15
16
```


