

# AI Assisted Coding Assignment-13.5

2303A51543 BT: 29 VELDI.HRUTHIKA 27.02.2026



## Task Description #1 (Refactoring – Removing Global Variables)

- Task: Use AI to eliminate unnecessary global variables from the code.

- Instructions:

- o Identify global variables used across functions.
- o Refactor the code to pass values using function parameters.
- o Improve modularity and testability.

- Sample Legacy Code:

```
rate = 0.1
def calculate_interest(amount):
    return amount * rate
print(calculate_interest(1000))
```

- Expected Output:

- o Refactored version passing rate as a parameter or using a configuration structure

#2303A51543

#Task Description #1 (Refactoring – Removing Global Variables)

#Sample Legacy Code:

```
#rate = 0.1
```

```
#def calculate_interest(amount):
```

```
#return amount * rate
```

```
#print(calculate_interest(1000))
```

#Prompt: Refactor the code to pass values using function parameters.

```
def calculate_interest(amount, rate):
```

```
    """
```

```
    Calculate the interest based on the given amount and rate.
```

```
    Parameters:
```

```
    amount (float): The principal amount for which interest is to be calculated.
```

```
    rate (float): The interest rate to be applied.
```

```
    Returns:
```

```
    float: The calculated interest.
```

```
    Example:
```

```
>>> calculate_interest(1000, 0.1)
```

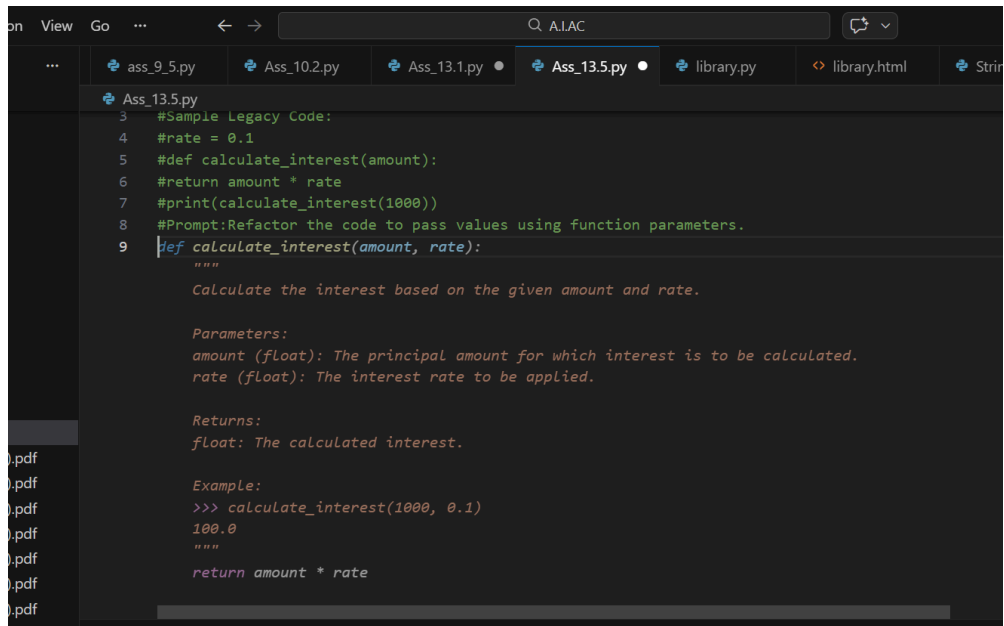
```
100.0
```

```
    """
```

```
    return amount * rate
```

```
# Example usage
```

```
print(calculate_interest(1000, 0.1)) # Expected output: 100.0
```

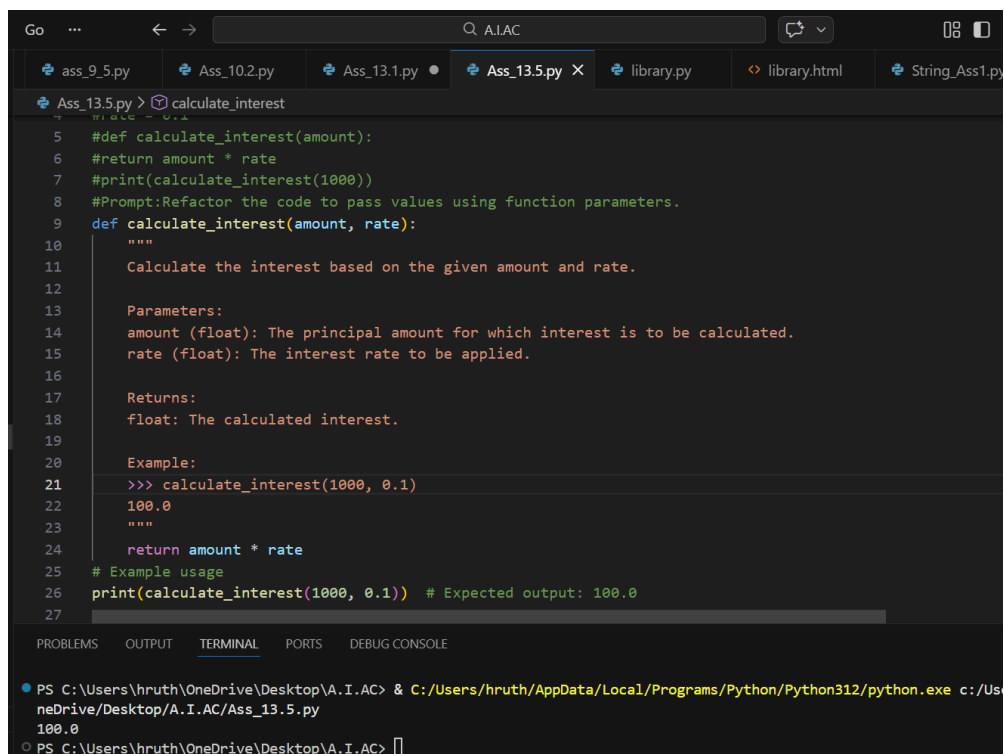


```
on View Go ...
Ass_9_5.py Ass_10.2.py Ass_13.1.py Ass_13.5.py library.py library.html Strin
Ass_13.5.py
3 #Sample Legacy Code:
4 #rate = 0.1
5 #def calculate_interest(amount):
6 #return amount * rate
7 #print(calculate_interest(1000))
8 #Prompt:Refactor the code to pass values using function parameters.
9 def calculate_interest(amount, rate):
    """
    Calculate the interest based on the given amount and rate.

    Parameters:
    amount (float): The principal amount for which interest is to be calculated.
    rate (float): The interest rate to be applied.

    Returns:
    float: The calculated interest.

    Example:
    >>> calculate_interest(1000, 0.1)
    100.0
    """
    return amount * rate
```



```
Go ...
Ass_9_5.py Ass_10.2.py Ass_13.1.py Ass_13.5.py X library.py library.html String_Ass1.py
Ass_13.5.py calculate_interest
5 #def calculate_interest(amount):
6 #return amount * rate
7 #print(calculate_interest(1000))
8 #Prompt:Refactor the code to pass values using function parameters.
9 def calculate_interest(amount, rate):
    """
    Calculate the interest based on the given amount and rate.

    Parameters:
    amount (float): The principal amount for which interest is to be calculated.
    rate (float): The interest rate to be applied.

    Returns:
    float: The calculated interest.

    Example:
    >>> calculate_interest(1000, 0.1)
    100.0
    """
    return amount * rate
24
25 # Example usage
26 print(calculate_interest(1000, 0.1)) # Expected output: 100.0
27

PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE
PS C:\Users\hruth\OneDrive\Desktop\A.I.AC> & C:/Users/hruth/AppData/Local/Programs/Python/Python312/python.exe c:/Us
neDrive/Desktop/A.I.AC/Ass_13.5.py
100.0
PS C:\Users\hruth\OneDrive\Desktop\A.I.AC>
```



### Task Description #2 : (Refactoring Deeply Nested Conditionals)

- Task: Use AI to refactor deeply nested if-elif-else logic into a cleaner structure.

- Focus Areas:

- o Readability

- o Logical simplification

- o Maintainability

Legacy Code:

```
score = 78
```

```
if score >= 90:
```

```
    print("Excellent")
```

```
else:
```

```
    if score >= 75:
```

```
        print("Very Good")
```

```
    else:
```

```
        if score >= 60:
```

```
            print("Good")
```

```
    else:
```

```
        print("Needs Improvement")
```

Expected Outcome:

- o Flattened logic using guard clauses or a mapping-based approach.

```
#Task Description #2 : (Refactoring Deeply Nested Conditionals)
```

```
#score = 78
```

```
#if score >= 90:
```

```
#print("Excellent")
```

```
#else:
```

```
#if score >= 75:
```

```
#print("Very Good")
```

```
#else:
```

```
#if score >= 60:
```

```
#print("Good")
```

```
#else:
```

```
#print("Needs Improvement")
```

```
#Prompt:Refactor deeply nested if-elif-else logic into a cleaner structure.
```

```
def evaluate_score(score):
```

```
    """
```

```
    Evaluate the score and return a corresponding message.
```

```
    Parameters:
```

```
    score (int): The score to be evaluated.
```

```
    Returns:
```

```
    str: A message indicating the evaluation of the score.
```

```
    Example:
```

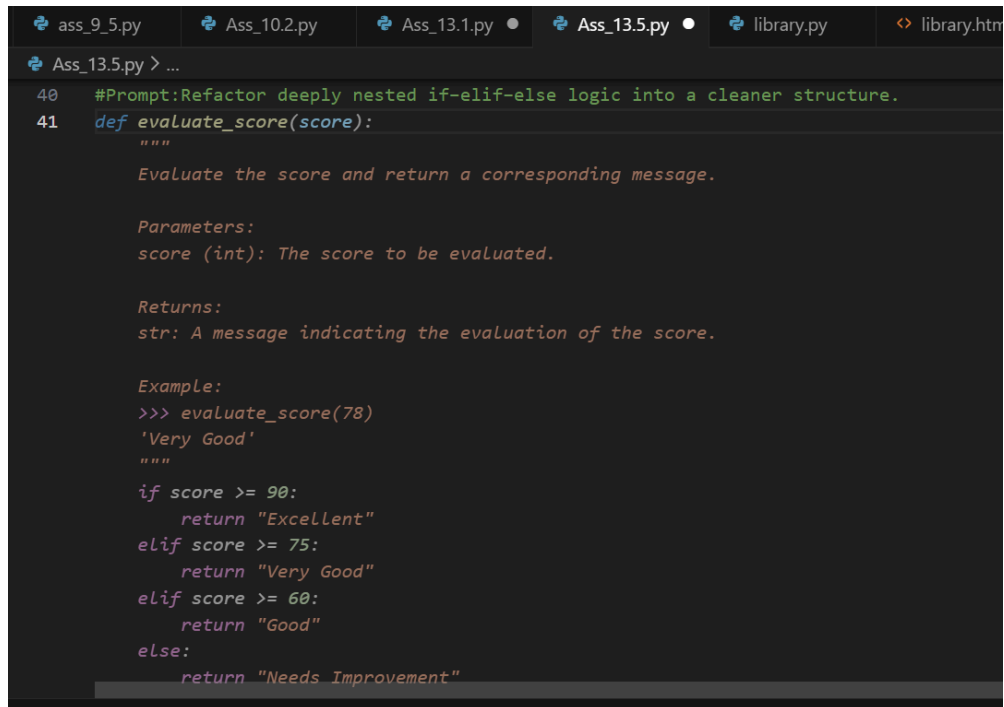
```
    >>> evaluate_score(78)
```

```
    'Very Good'
```

```
    """
```

```
    if score >= 90:
```

```
        return "Excellent"
    elif score >= 75:
        return "Very Good"
    elif score >= 60:
        return "Good"
    else:
        return "Needs Improvement"
# Example usage
print(evaluate_score(78)) # Expected output: "Very Good"
```

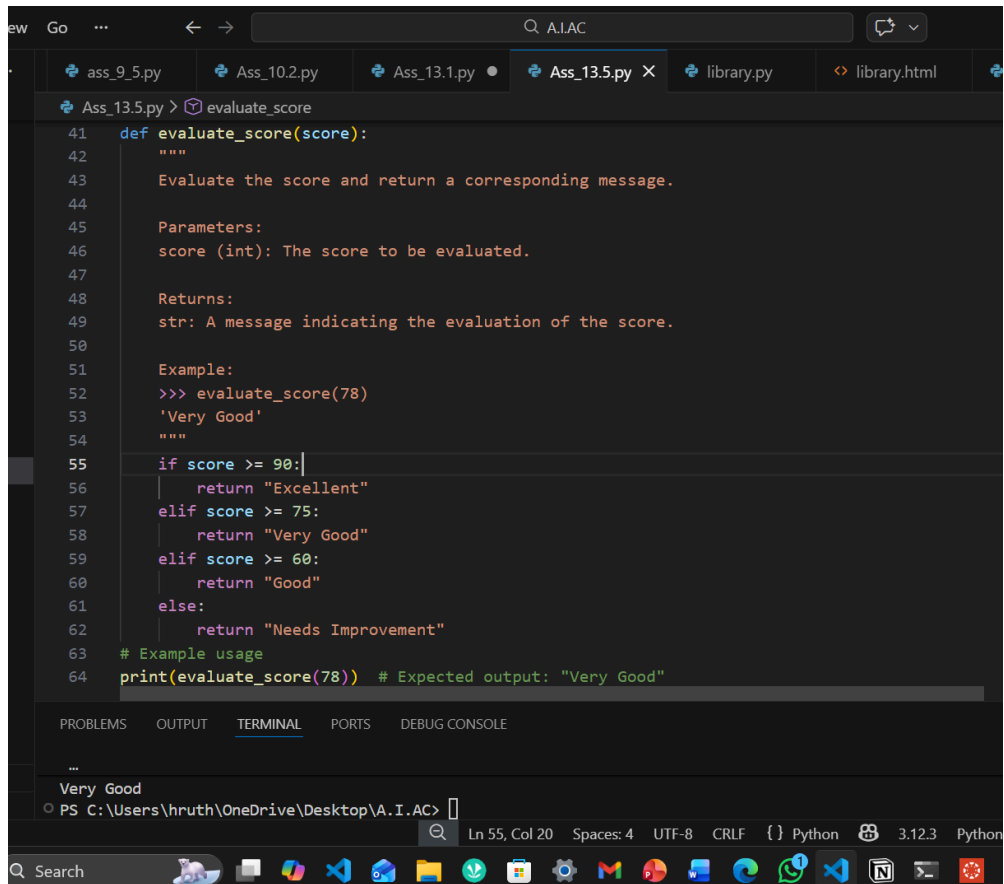


```
Ass_13.5.py > ...
40  #Prompt:Refactor deeply nested if-elif-else logic into a cleaner structure.
41  def evaluate_score(score):
    """
        Evaluate the score and return a corresponding message.

        Parameters:
        score (int): The score to be evaluated.

        Returns:
        str: A message indicating the evaluation of the score.

        Example:
        >>> evaluate_score(78)
        'Very Good'
    """
    if score >= 90:
        return "Excellent"
    elif score >= 75:
        return "Very Good"
    elif score >= 60:
        return "Good"
    else:
        return "Needs Improvement"
```



```
41 def evaluate_score(score):
42     """
43     Evaluate the score and return a corresponding message.
44
45     Parameters:
46     score (int): The score to be evaluated.
47
48     Returns:
49     str: A message indicating the evaluation of the score.
50
51     Example:
52     >>> evaluate_score(78)
53     'Very Good'
54     """
55     if score >= 90:
56         return "Excellent"
57     elif score >= 75:
58         return "Very Good"
59     elif score >= 60:
60         return "Good"
61     else:
62         return "Needs Improvement"
63 # Example usage
64 print(evaluate_score(78)) # Expected output: "Very Good"
```



### Task 3 (Refactoring Repeated File Handling Code)

- Task: Use AI to refactor repeated file open/read/close logic.

- Focus Areas:

- o DRY principle
- o Context managers
- o Function reuse

Legacy Code:

```
f = open("data1.txt")
print(f.read())
f.close()
f = open("data2.txt")
print(f.read())
f.close()
```

Expected Outcome:

- o Reusable function using with open() and parameters.

### #Task 3 (Refactoring Repeated File Handling Code)

```
#f = open("data1.txt")
#print(f.read())
#f.close()
#f = open("data2.txt")
#print(f.read())
```

```

#f.close()
#Prompt:Generate a reusable function using with open() and parameters.
def read_file(file_name):
    """
    Read the contents of a file and return it as a string.
    Parameters:
    file_name (str): The name of the file to be read.
    Returns:
    str: The contents of the file.
    Example:
    >>> read_file("data1.txt")
    'Contents of data1.txt'
    """
    with open(file_name, 'r') as file:
        return file.read()
# Example usage
print(read_file("data1.txt")) # Expected output: Contents of data1.txt
print(read_file("data2.txt")) # Expected output: Contents of data2.txt

```

The screenshot shows a code editor with several tabs at the top: 'Ass\_9.3.py', 'Ass\_10.2.py', 'Ass\_13.1.py', 'Ass\_13.5.py', 'library.py', and 'library.html'. The active tab is 'Ass\_13.5.py'. The code in the editor is as follows:

```

68 #print(f.read())
69 #f.close()
70 #f = open("data2.txt")
71 #print(f.read())
72 #f.close()
73 #Prompt:Generate a reusable function using with open() and parameters.
74 -> def read_file(file_name):
    """
    Read and return the content of a file.

    Parameters:
    file_name (str): The name of the file to be read.

    Returns:
    str: The content of the file.

    Example:
    >>> read_file("data1.txt")
    'Content of data1.txt'
    """
    with open(file_name, "r") as f:
        return f.read()

```

```

File Edit Selection View Go ...
ASS_13.5.py
def read_file(file_name):
    Example:
    >>> read_file("data1.txt")
    'Contents of data1.txt'
    """
    with open(file_name, 'r') as file:
        return file.read()
# Example usage
print(read_file("data1.txt")) # Expected output: Contents of data1.txt
print(read_file("data2.txt")) # Expected output: Contents of data2.txt

PS C:\Users\hruth\OneDrive\Desktop\A.I.AC> & C:/Users/hruth/AppData/Local/Programs/Python/Python311/Python.exe C:\Users\hruth\OneDrive\Desktop\A.I.AC\Ass_13.5.py
Read the contents of a file and return it as a string.

Parameters:
file_name (str): The name of the file to be read.

Returns:
str: The contents of the file.

Example:
>>> read_file("data1.txt")
'Contents of data1.txt'
2303AS1543
ASS-13.5
BATCH-29
A.I. ASSISTED CODING
PS C:\Users\hruth\OneDrive\Desktop\A.I.AC>

```



#### Task 4 (Optimizing Search Logic)

- Task: Refactor inefficient linear searches using appropriate data structures.

##### • Focus Areas:

- o Time complexity
- o Data structure choice

##### Legacy Code:

```

users = ["admin", "guest", "editor", "viewer"]
name = input("Enter username: ")
found = False
for u in users:
    if u == name:
        found = True
print("Access Granted" if found else "Access Denied")

```

##### Expected Outcome:

- o Use of sets or dictionaries with complexity justification.

#### #Task 4 (Optimizing Search Logic)

```
#users = ["admin", "guest", "editor", "viewer"]
```

```
#name = input("Enter username: ")
```

```
#found = False
```

```
#for u in users:
```

```
#if u == name:
```

```
#found = True
```

```
#print("Access Granted" if found else "Access Denied")
```

```
#Prompt:Refactor inefficient linear searches using appropriate data structures.
```

```

users = {"admin": "Administrator", "guest": "Guest User", "editor": "Content Editor", "viewer": "Content Viewer"}

```

```

name = input("Enter username: ")
if name in users:
    print("Access Granted")
else:
    print("Access Denied")

```

```

93
94 #Task 4 (Optimizing Search Logic)
95 #users = ["admin", "guest", "editor", "viewer"]
96 #name = input("Enter username: ")
97 #found = False
98 #for u in users:
99 #if u == name:
100 #found = True
101 #print("Access Granted" if found else "Access Denied")
102 #Prompt:Refactor inefficient linear searches using appropriate data structures.
103 → users = {"admin": "Administrator", "guest": "Guest User", "editor": "Content Editor", "viewer": "Content Viewer"}
    name = input("Enter username: ")
    if name in users:
        print("Access Granted")
    else:
        print("Access Denied")

```

```

93
94 #Task 4 (Optimizing Search Logic)
95 #users = ["admin", "guest", "editor", "viewer"]
96 #name = input("Enter username: ")
97 #found = False
98 #for u in users:
99 #if u == name:
100 #found = True
101 #print("Access Granted" if found else "Access Denied")
102 #Prompt:Refactor inefficient linear searches using appropriate data structures.
103 users = {"admin": "Administrator", "guest": "Guest User", "editor": "Content Editor", "viewer": "Content Viewer"}
104 name = input("Enter username: ")
105 if name in users:
106     print("Access Granted")
107 else:
108     print("Access Denied")
109
Terminal (Ctrl+)

```

PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE

```

PS C:\Users\hruth\OneDrive\Desktop\A.I.AC> & C:/Users/hruth/AppData/Local/Programs/Python/Python312/python.exe c:/Users/hruth/OneDrive/Desktop/A.I.AC/Ass_13.5.py
2303A51543
ASS-13.5
BATCH-29
A.I. ASSISTED CODING
Enter username: guest
Access Granted

```



#### Task 5 (Refactoring Procedural Code into OOP Design)

- Task: Use AI to refactor procedural code into a class-based design.
- Focus Areas:
  - o Object-Oriented principles
  - o Encapsulation
- Legacy Code:
 

```

salary = 50000
tax = salary * 0.2
net = salary - tax
print(net)

```
- Expected Outcome:
  - o A class like EmployeeSalaryCalculator with methods and attributes.



```

#Task 5 (Refactoring Procedural Code into OOP Design)
#salary = 50000
#tax = salary * 0.2
#net = salary - tax
#print(net)
#Prompt:refactor procedural code into a class-based design ike EmployeeSalaryCalculator w
ith methods and attributes.
class EmployeeSalaryCalculator:
    """
    A class to calculate the net salary of an employee after tax deductions.
    Attributes:
    salary (float): The gross salary of the employee.
    tax_rate (float): The tax rate to be applied on the salary.
    Methods:
    calculate_tax(): Calculate the tax based on the salary and tax rate.
    calculate_net_salary(): Calculate the net salary after deducting tax from the gross s
alary.
    """
    def __init__(self, salary, tax_rate=0.2):
        """
        Initialize the EmployeeSalaryCalculator with a salary and an optional tax rate.
        Parameters:
        salary (float): The gross salary of the employee.
        tax_rate (float, optional): The tax rate to be applied. Default is 0.2 (20%).
        """
        self.salary = salary
        self.tax_rate = tax_rate
    def calculate_tax(self):
        """Calculate the tax based on the salary and tax rate."""
        return self.salary * self.tax_rate
    def calculate_net_salary(self):
        """Calculate the net salary after deducting tax from the gross salary."""
        tax = self.calculate_tax()
        return self.salary - tax
# Example usage
employee = EmployeeSalaryCalculator(50000) # Create an instance of EmployeeSalaryCalcula
tor with a salary of 50000
print(employee.calculate_net_salary()) # Expected output: 40000.0

```

```

Ass_13.5.py > ...
113 #net = salary - tax
114 #print(net)
115 #Prompt:refactor procedural code into a class-based design like EmployeeSalaryCalculator with methods and attributes
116 → class EmployeeSalaryCalculator:
    def __init__(self, salary):
        self.salary = salary

    def calculate_tax(self, tax_rate=0.2):
        return self.salary * tax_rate

    def calculate_net_salary(self, tax_rate=0.2):
        tax = self.calculate_tax(tax_rate)
        return self.salary - tax

# Example usage
calculator = EmployeeSalaryCalculator(50000)
print(calculator.calculate_net_salary()) # Expected output: 40000.0

```

```

Ass_13.5.py > ...
16 class EmployeeSalaryCalculator:
24     """
25     calculate_tax(): Calculate the tax based on the salary and tax rate.
26     calculate_net_salary(): Calculate the net salary after deducting tax from the gross salary.
27     """
28     def __init__(self, salary, tax_rate=0.2):
29         """
30         Initialize the EmployeeSalaryCalculator with a salary and an optional tax rate.
31
32         Parameters:
33         salary (float): The gross salary of the employee.
34         tax_rate (float, optional): The tax rate to be applied. Default is 0.2 (20%).
35         """
36         self.salary = salary
37         self.tax_rate = tax_rate
38     def calculate_tax(self):
39         """Calculate the tax based on the salary and tax rate."""
40         return self.salary * self.tax_rate
41     def calculate_net_salary(self):
42         """Calculate the net salary after deducting tax from the gross salary."""
43         tax = self.calculate_tax()
44         return self.salary - tax
45 # Example usage
46 employee = EmployeeSalaryCalculator(50000) # Create an instance of EmployeeSalaryCalculator with a salary of
47 print(employee.calculate_net_salary()) # Expected output: 40000.0

```

PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE

40000.0

C:\Users\hruth\OneDrive\Desktop\A.I.AC>



#### Task 6 (Refactoring for Performance Optimization)

- Task: Use AI to refactor a performance-heavy loop handling large data.

##### • Focus Areas:

- o Algorithmic optimization
- o Use of built-in functions

##### Legacy Code:

```
total = 0
```

```
for i in range(1, 1000000):
```

```
    if i % 2 == 0:
```

```
        total += i
```

```
print(total)
```

##### Expected Outcome:

- o Optimized logic using mathematical formulas or comprehensions, with time comparison.

```
#Task 6 (Refactoring for Performance Optimization)
#total = 0
#for i in range(1, 1000000):
#if i % 2 == 0:
#total += i
#print(total)
#Prompt:Generate Optimized logic code using mathematical formulas or comprehensions, with
time comparison.
total = sum(i for i in range(1, 1000000) if i % 2 == 0)
print(total) # Expected output: 249999500000
```

```
149 #Task 6 (Refactoring for Performance Optimization)
150 #total = 0
151 #for i in range(1, 1000000):
152 #if i % 2 == 0:
153 #total += i
154 #print(total)
155 #Prompt:Generate Optimized logic code using mathematical formulas or comprehensions, with time comparison.
156 total = sum(i for i in range(1, 1000000) if i % 2 == 0)
157 print(total) # Expected output: 249999500000
158
```

249999500000



#### Task 7 (Removing Hidden Side Effects)

- Task: Refactor code that modifies shared mutable state.
- Focus Areas:
  - o Functional-style refactoring
  - o Predictability
- Legacy Code:
 

```
data = []
def add_item(x):
    data.append(x)
add_item(10)
add_item(20)
print(data)
```
- Expected Outcome:
  - o Refactored function returning values instead of mutating globals.

```
#Task 7 (Removing Hidden Side Effects)
#data = []
#def add_item(x):
#data.append(x)
#add_item(10)
#add_item(20)
#print(data)
#Prompt:Refactore function returning values instead of mutating globals.
```

```
def add_item(x, data_list):
    data_list.append(x)
    return data_list
data = []
data = add_item(10, data)
data = add_item(20, data)
print(data)
```

```
#Task 7 (Removing Hidden Side Effects)
#data = []
#def add_item(x):
#    data.append(x)
#add_item(10)
#add_item(20)
#print(data)
#Prompt:Refactore function returning values instead of mutatingglobals.
->def add_item(x, data_list):
    return data_list + [x]
data = []
data = add_item(10, data)
data = add_item(20, data)
print(data)
```

```
158
159
160 #Task 7 (Removing Hidden Side Effects)
161 #data = []
162 #def add_item(x):
163 #    data.append(x)
164 #add_item(10)
165 #add_item(20)
166 #print(data)
167 #Prompt:Refactore function returning values instead of mutating globals.
168 def add_item(x, data_list):
169     data_list.append(x)
170     return data_list
171
172 data = []
173 data = add_item(10, data)
174 data = add_item(20, data)
175 print(data)
```

PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE

...  
[10, 20]

PS C:\Users\hruth\OneDrive\Desktop\A.I.AC>

Ln 172, Col 10 Spaces: 4 UTF-8 CRLF {} Python



#### Task 8 (Refactoring Complex Input Validation Logic)

- Task: Use AI to simplify and modularize complex validation rules.

- Focus Areas:

- o Readability

- o Testability

Legacy Code:

```
password = input("Enter password: ")
```

```
if len(password) >= 8:
```

```
    if any(c.isdigit() for c in password):
```

```
        if any(c.isupper() for c in password):
```

```
            print("Valid Password")
```

```
        else:
```

```
            print("Must contain uppercase")
```

```
    else:
```

```
        print("Must contain digit")
```

```
else:
```

```
    print("Password too short")
```

Expected Outcome:

- o Separate validation functions with clear responsibility

```
#Task 8 (Refactoring Complex Input Validation Logic)
```

```
#password = input("Enter password: ")
```

```
#if len(password) >= 8:
```

```
#if any(c.isdigit() for c in password):
```

```
#if any(c.isupper() for c in password):
```

```
#print("Valid Password")
```

```
#else:
```

```
#print("Must contain uppercase")
```

```
#else:
```

```
#print("Must contain digit")
```

```
#else:
```

```
#print("Password too short")
```

```
#Prompt:Simplify and modularize complex validation. Separate validation functions with clear responsibility
```

```
def is_valid_password(password):
```

```
    if len(password) < 8:
```

```
        return "Password too short"
```

```
    if not any(c.isdigit() for c in password):
```

```
        return "Must contain digit"
```

```
    if not any(c.isupper() for c in password):
```

```
        return "Must contain uppercase"
```

```
    return "Valid Password"
```

```
password = input("Enter password: ")
```

```
print(is_valid_password(password))
```

```
String.py  Ass_8.2.py  ass_9.5.py  Ass_10.2.py  Ass_13.1.py  Ass_13.5.py  data2.txt  data1.
Ass_13.5.py > ...
77 #Task 8 (Refactoring Complex Input Validation Logic)
78 #password = input("Enter password: ")
79 #if len(password) >= 8:
80 #if any(c.isdigit() for c in password):
81 #if any(c.isupper() for c in password):
82 #print("Valid Password")
83 #else:
84 #print("Must contain uppercase")
85 #else:
86 #print("Must contain digit")
87 #else:
88 #print("Password too short")
89 #Prompt:Simplify and modularize complex validation.Separate validation functions with clear responsibility
90 def is_valid_password(password):
    if len(password) < 8:
        return "Password too short"
    if not any(c.isdigit() for c in password):
        return "Must contain digit"
    if not any(c.isupper() for c in password):
        return "Must contain uppercase"
    return "Valid Password"
```

```
176
177 #Task 8 (Refactoring Complex Input Validation Logic)
178 #password = input("Enter password: ")
179 #if len(password) >= 8:
180 #if any(c.isdigit() for c in password):
181 #if any(c.isupper() for c in password):
182 #print("Valid Password")
183 #else:
184 #print("Must contain uppercase")
185 #else:
186 #print("Must contain digit")
187 #else:
188 #print("Password too short")
189 #Prompt:Simplify and modularize complex validation.Separate validation functions with clear responsibility
190 def is_valid_password(password):
191     if len(password) < 8:
192         return "Password too short"
193     if not any(c.isdigit() for c in password):
194         return "Must contain digit"
195     if not any(c.isupper() for c in password):
196         return "Must contain uppercase"
197     return "Valid Password"
198 password = input("Enter password: ")
199 print(is_valid_password(password))

PROBLEMS  OUTPUT  TERMINAL  PORTS  DEBUG CONSOLE

Enter password: Hruthika@srus#12
Valid Password
PS C:\Users\hruth\OneDrive\Desktop\A.I.AC> |
```