AI Assisted Coding

# Assignment-10.3

## K.Sindhu meenan(2403A51250)-Batch11

Task-1:

Prompt:

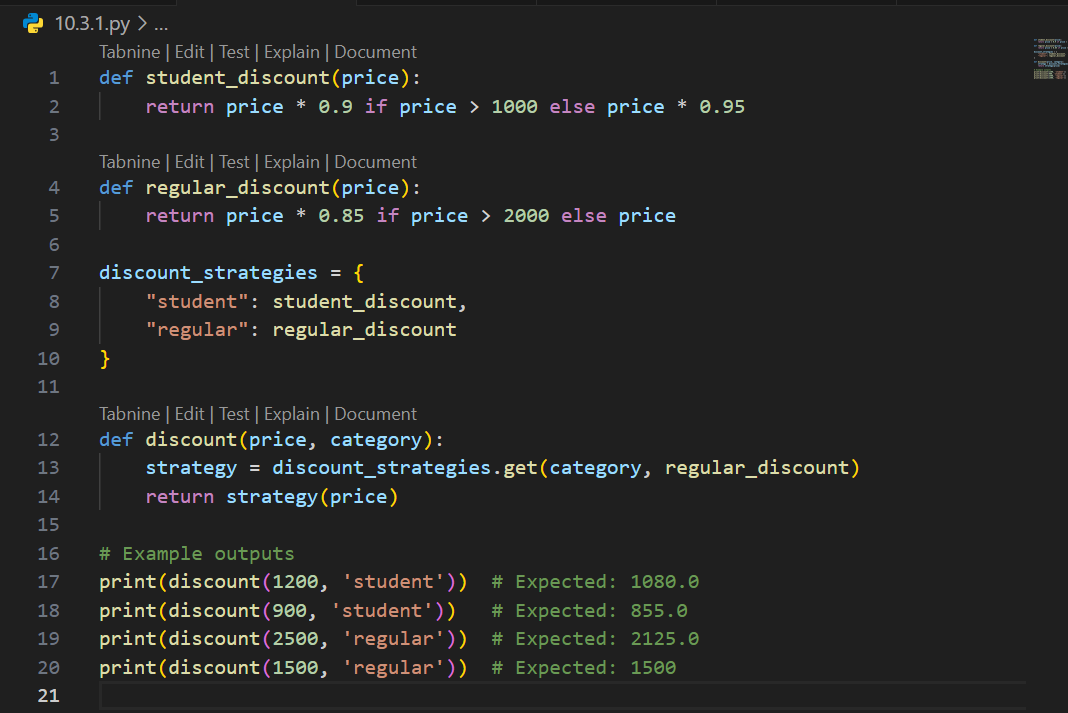
Refactor the following nested conditional Python code for better readability. Aim to simplify the logic using cleaner structures such as dictionaries, helper functions, or other Pythonic approaches:

def discount(price, category):

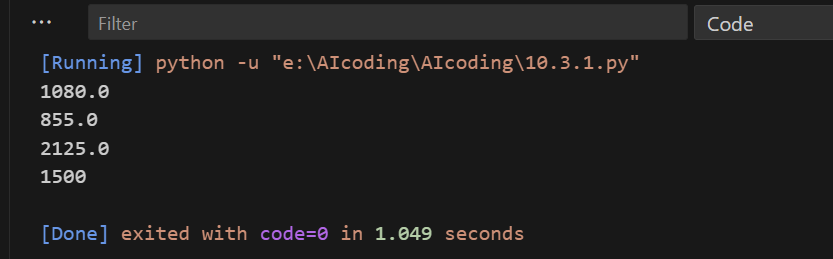
if category == "student":  
if price > 1000:  
return price \* 0.9  
else:  
return price \* 0.95  
else:  
if price > 2000:  
return price \* 0.85  
else:  
return price

Expected Output:  
• Refactored code using cleaner logic, possibly a dictionary or separate  
helper functions.

Code:



Output:



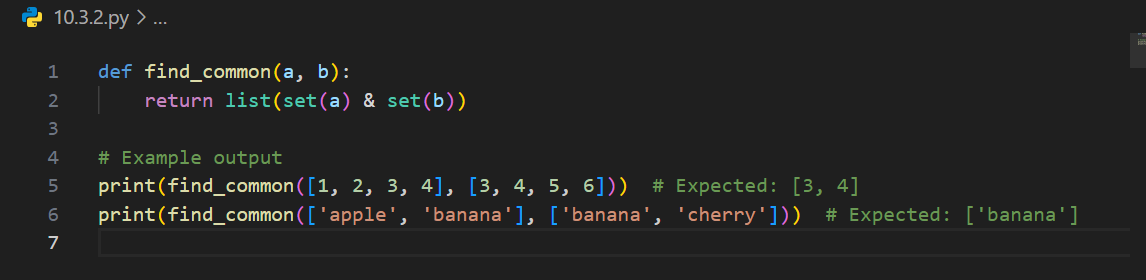
Task-2:

Prompt:  
Refactor the following Python code to eliminate redundant nested loops. Optimize it using more efficient Python features such as sets:

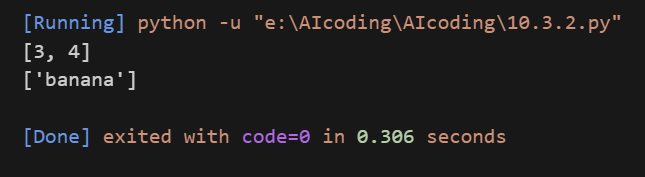
def find\_common(a, b):  
res = []  
for i in a:  
for j in b:  
if i == j:  
res.append(i)  
return res

Expected Output:  
Cleaner version using Python sets (set(a) & set(b))

Code:



Output:



Task-3:

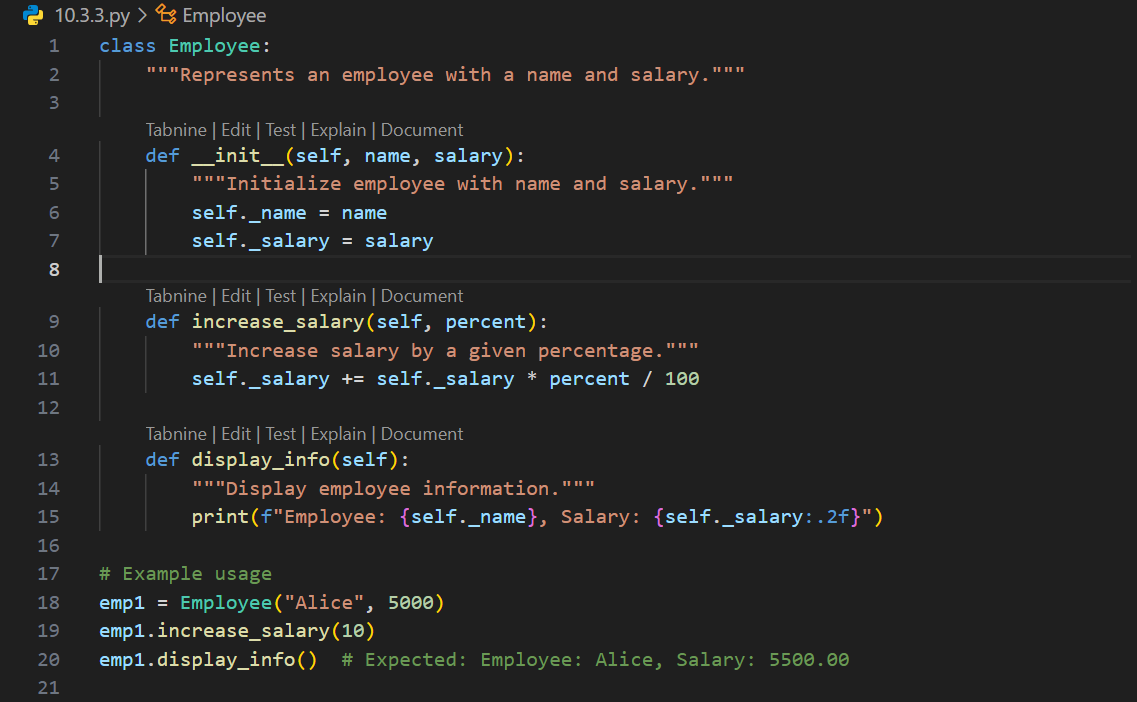
Prompt:

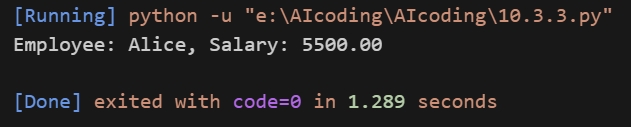
Refactor the following Python class to improve readability and maintainability. Apply proper naming conventions, encapsulation, and clear method responsibilities. Add docstrings for better understanding.

class emp:  
def \_\_init\_\_(self,n,s):  
self.n=n  
self.s=s  
def inc(self,p):  
self.s=self.s+(self.s\*p/100)  
def pr(self):  
print("emp:",self.n,"salary:",self.s)

Expected Output:  
• Employee class with meaningful methods (increase\_salary,  
display\_info), formatted output, and added docstrings.

Code:

  
Output:



Task-4:

Prompt:

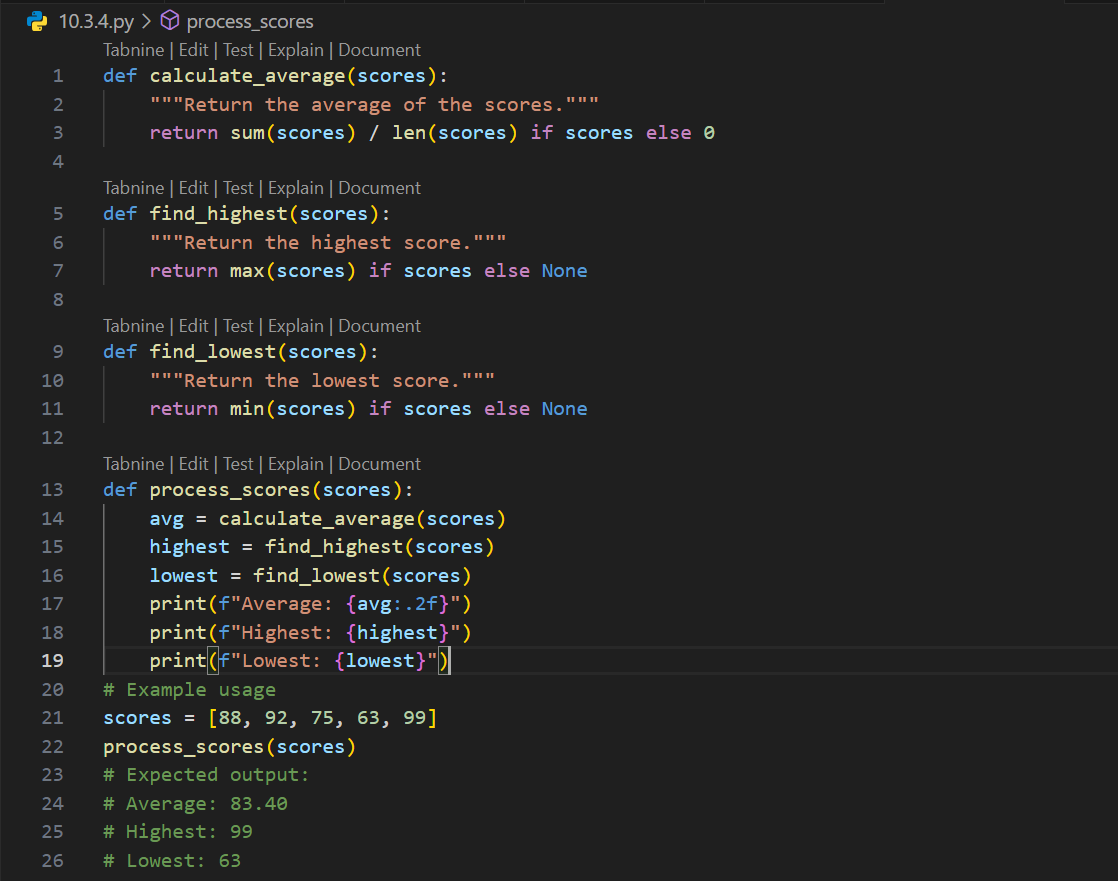
Refactor the following long, unstructured Python function by breaking it into smaller, reusable helper functions. Improve readability and maintainability by modularizing the logic.

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:

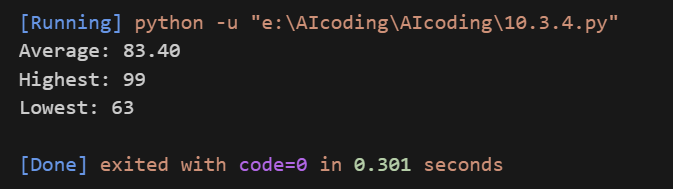
highest = s  
lowest = scores[0]  
for s in scores:  
if s < lowest:  
lowest = s  
print("Average:", avg)  
print("Highest:", highest)  
print("Lowest:", lowest)

Expected Output:  
• Split into functions: calculate\_average, find\_highest, find\_lowest.  
• Clean main process\_scores() using helper functions.

Code:



Output:



Task-5:

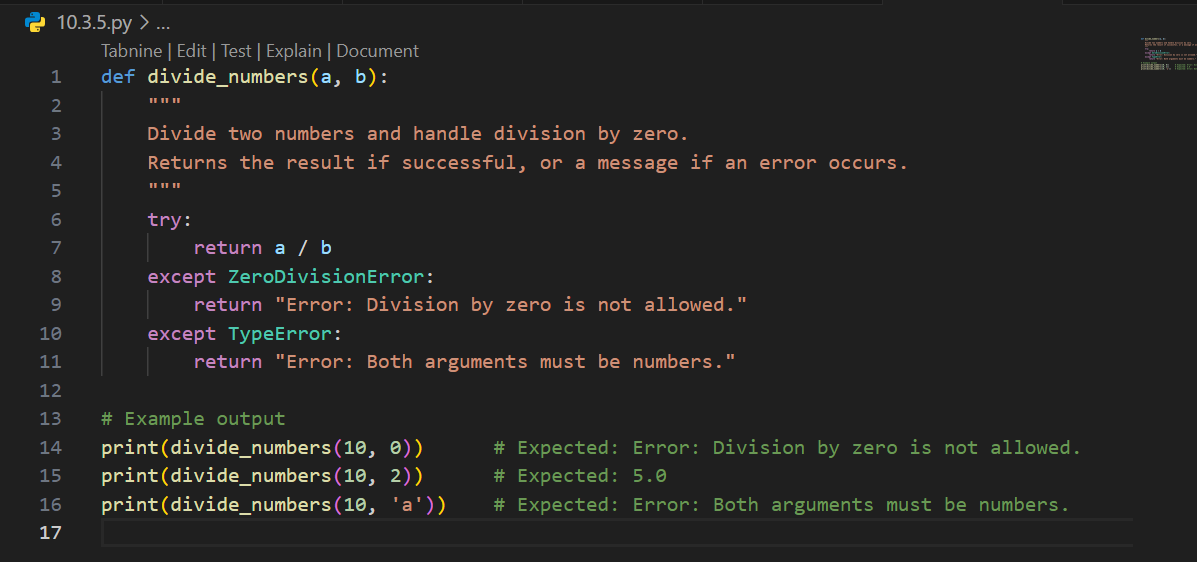
Prompt:

Review and refactor the following Python code to improve error handling, naming conventions, and readability. Add a docstring that explains the function and its error handling.

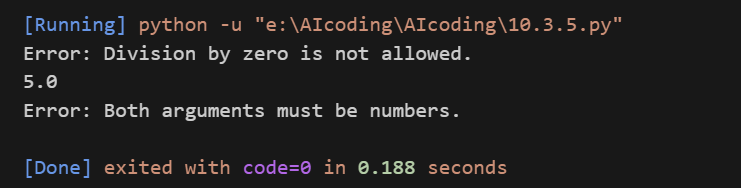
def div(a,b):  
return a/b  
print(div(10,0))

Expected Output:  
• Function with proper error handling using try-except.  
• Better naming (divide\_numbers).  
• AI-generated docstring explaining error handling.

Code:



Output:



Task-6:

Prompt:

Simplify the following overly complex Python function that uses deeply nested conditionals. Refactor it into a cleaner version using elif statements or a dictionary mapping for better readability and maintainability.

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"

Expected Output:  
• Cleaner logic using elif or dictionary mapping.

Code:



Output:

