AI Assisted Coding

Assignment-10.3

S.SIRI(2403A51236)-Batch11

Task-1:

Prompt:

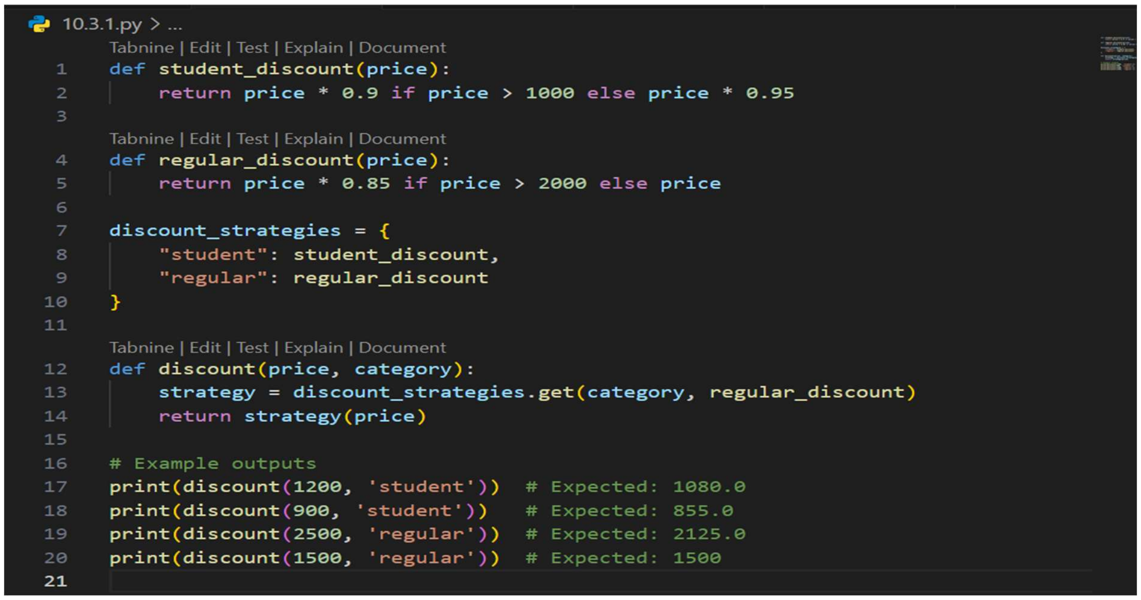
Refactor the following nested conditional Python code for better readability. Aim to simplify the logic using cleaner structures such as dictionaries, helper functions, or other Pythonic approaches:

```
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:

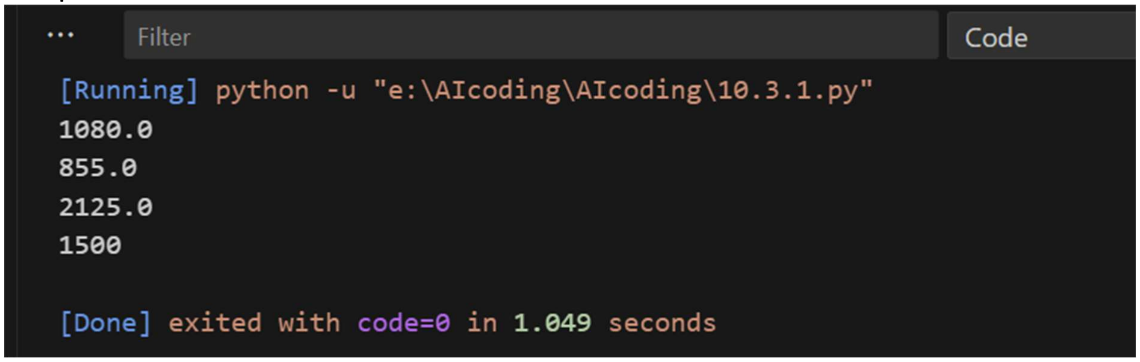
• Refactored code using cleaner logic, possibly a dictionary or separate helper functions.

Code:



```
10.3.1.py > ...
Tabnine | Edit | Test | Explain | Document
def student_discount(price):
    return price * 0.9 if price > 1000 else price * 0.95

Tabnine | Edit | Test | Explain | Document
def regular_discount(price):
    return price * 0.85 if price > 2000 else price

discount_strategies = {
    "student": student_discount,
    "regular": regular_discount
}

Tabnine | Edit | Test | Explain | Document
def discount(price, category):
    strategy = discount_strategies.get(category, regular_discount)
    return strategy(price)

# Example outputs
print(discount(1200, 'student'))  # Expected: 1080.0
print(discount(900, 'student'))   # Expected: 855.0
print(discount(2500, 'regular'))  # Expected: 2125.0
print(discount(1500, 'regular'))  # Expected: 1500
```

Output:



```
[Running] python -u "e:\AIcoding\AIcoding\10.3.1.py"
1080.0
855.0
2125.0
1500

[Done] exited with code=0 in 1.049 seconds
```

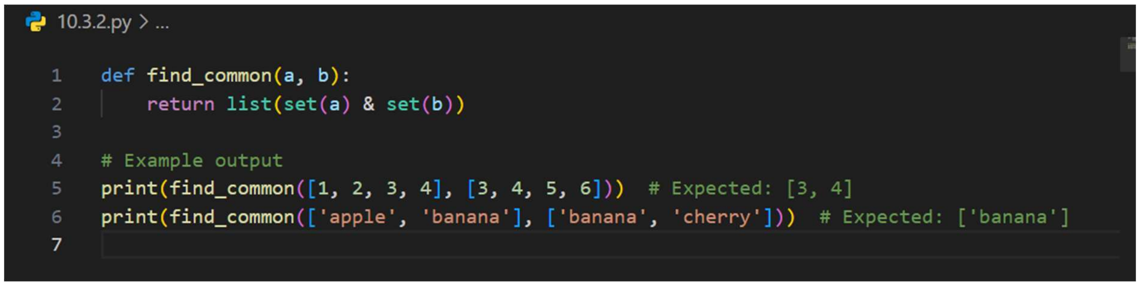
Task-2:

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

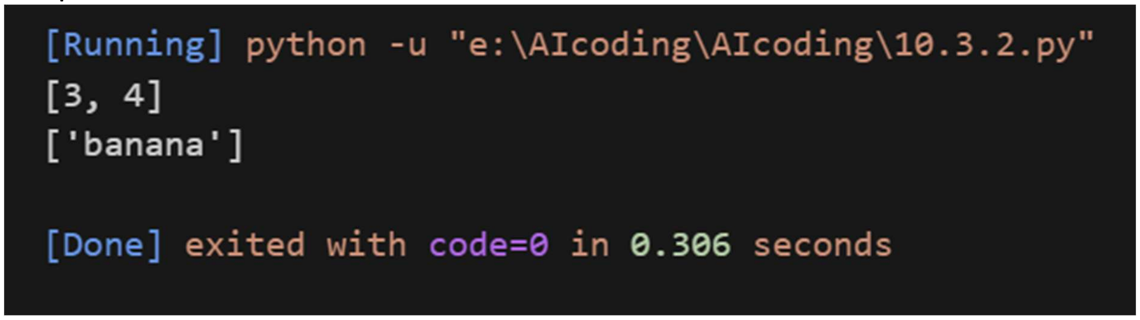
Expected Output:
Cleaner version using Python sets (set(a) & set(b))

Code:



```python
def find_common(a, b):
    return list(set(a) & set(b))

# Example output
print(find_common([1, 2, 3, 4], [3, 4, 5, 6]))  # Expected: [3, 4]
print(find_common(['apple', 'banana'], ['banana', 'cherry']))  # Expected: ['banana']
```

Output:



```
[Running] python -u "e:\AIcoding\AIcoding\10.3.2.py"
[3, 4]
['banana']

[Done] exited with code=0 in 0.306 seconds
```

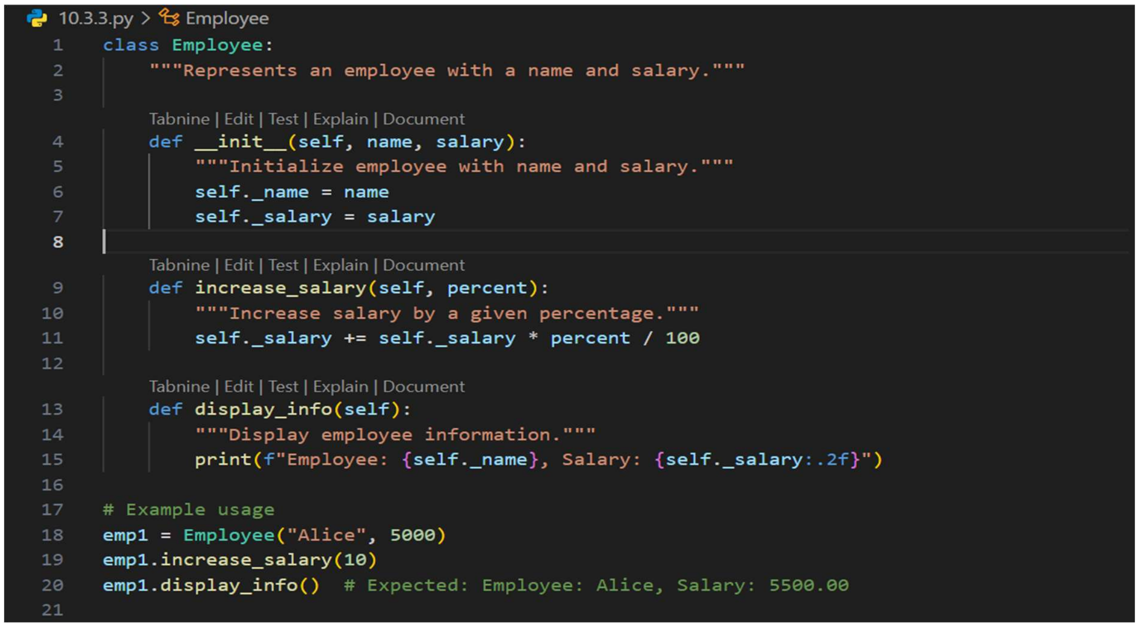
Task-3:

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.

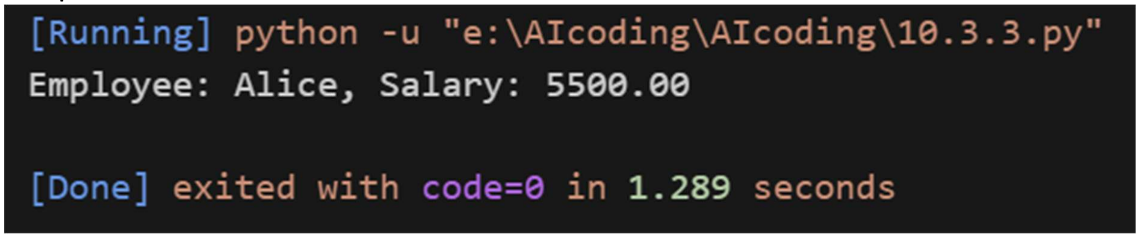
Code:



10.3.3.py > Employee

```python
class Employee:
    """Represents an employee with a name and salary."""

    def __init__(self, name, salary):
        """Initialize employee with name and salary."""
        self._name = name
        self._salary = salary

    def increase_salary(self, percent):
        """Increase salary by a given percentage."""
        self._salary += self._salary * percent / 100

    def display_info(self):
        """Display employee information."""
        print(f"Employee: {self._name}, Salary: {self._salary:.2f}")

# Example usage
emp1 = Employee("Alice", 5000)
emp1.increase_salary(10)
emp1.display_info()  # Expected: Employee: Alice, Salary: 5500.00
```

Output:



```
[Running] python -u "e:\AIcoding\AIcoding\10.3.3.py"
Employee: Alice, Salary: 5500.00

[Done] exited with code=0 in 1.289 seconds
```

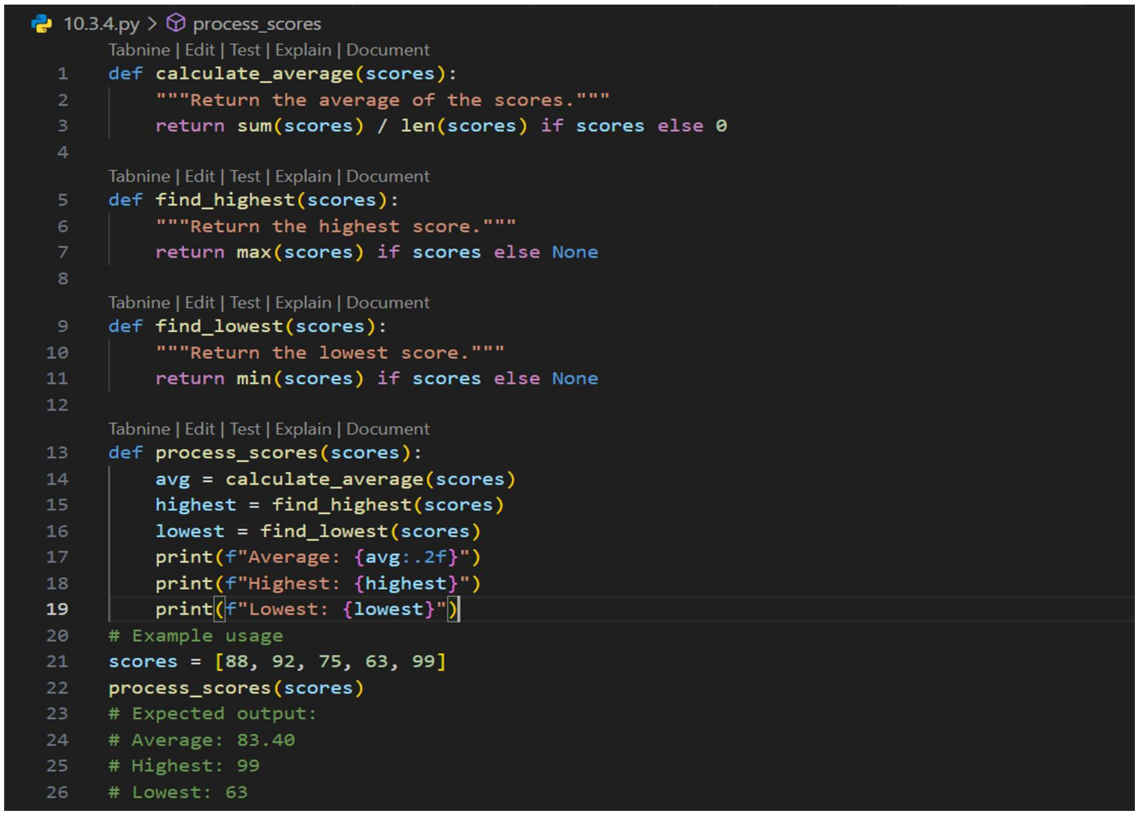
Task-4:

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.

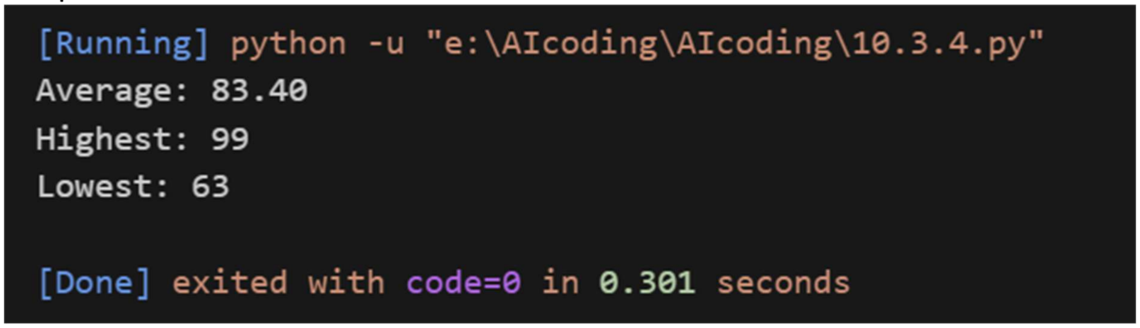
Code:



10.3.4.py > process_scores

```python
def calculate_average(scores):
    """Return the average of the scores."""
    return sum(scores) / len(scores) if scores else 0

def find_highest(scores):
    """Return the highest score."""
    return max(scores) if scores else None

def find_lowest(scores):
    """Return the lowest score."""
    return min(scores) if scores else None

def process_scores(scores):
    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}")
# Example usage
scores = [88, 92, 75, 63, 99]
process_scores(scores)
# Expected output:
# Average: 83.40
# Highest: 99
# Lowest: 63
```

Output:



```
[Running] python -u "e:\AIcoding\AIcoding\10.3.4.py"
Average: 83.40
Highest: 99
Lowest: 63

[Done] exited with code=0 in 0.301 seconds
```

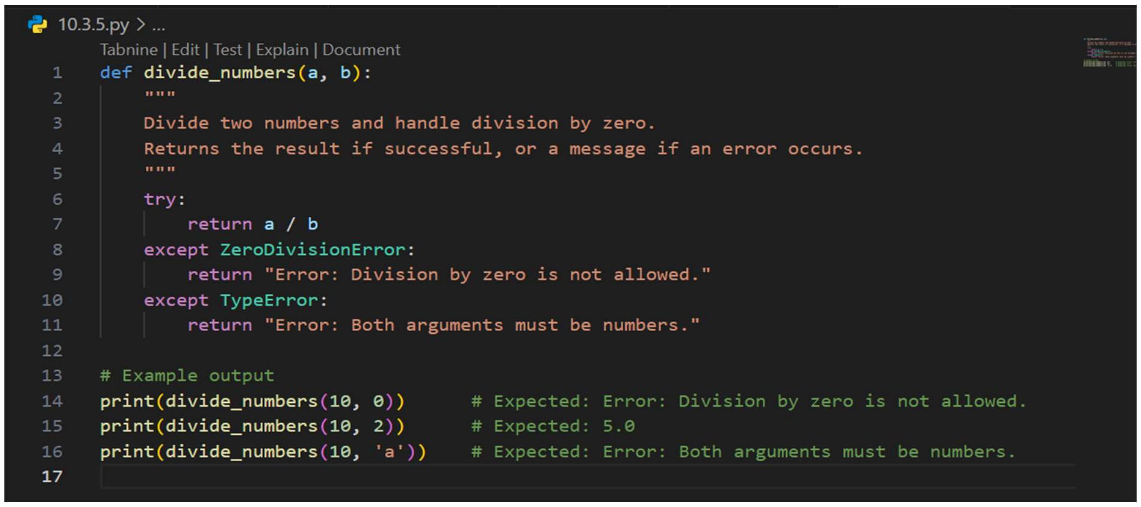
Task-5:

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:

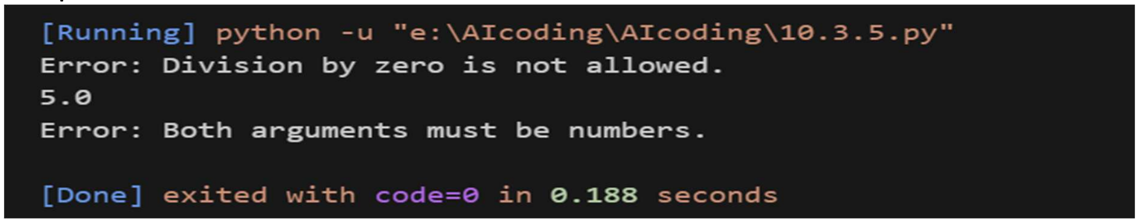


10.3.5.py > ...

Tabnine | Edit | Test | Explain | Document

```python
def divide_numbers(a, b):
    """
    Divide two numbers and handle division by zero.
    Returns the result if successful, or a message if an error occurs.
    """
    try:
        return a / b
    except ZeroDivisionError:
        return "Error: Division by zero is not allowed."
    except TypeError:
        return "Error: Both arguments must be numbers."

# Example output
print(divide_numbers(10, 0))      # Expected: Error: Division by zero is not allowed.
print(divide_numbers(10, 2))      # Expected: 5.0
print(divide_numbers(10, 'a'))    # Expected: Error: Both arguments must be numbers.
```

Output:



```
[Running] python -u "e:\AIcoding\AIcoding\10.3.5.py"
Error: Division by zero is not allowed.
5.0
Error: Both arguments must be numbers.

[Done] exited with code=0 in 0.188 seconds
```

Task-6:

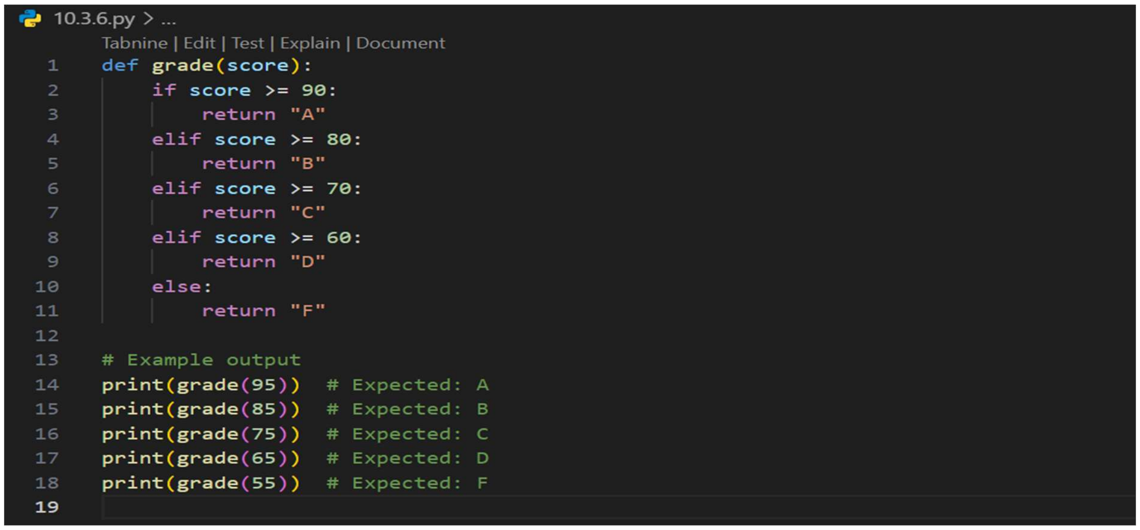
Prompt:

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.

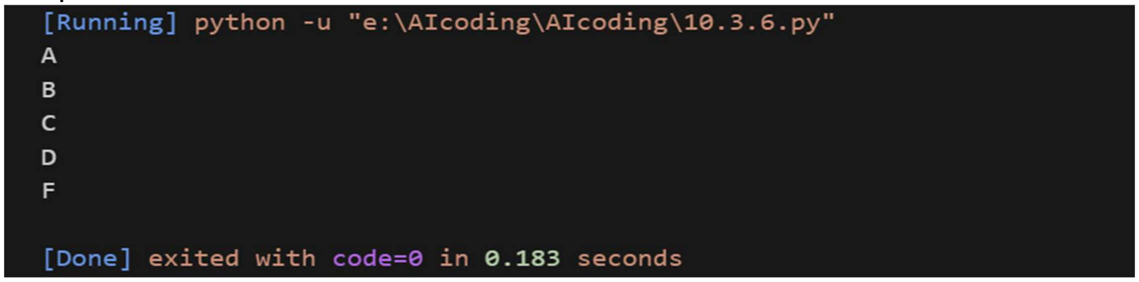
Code:



10.3.6.py > ...

```python
def grade(score):
    if score >= 90:
        return "A"
    elif score >= 80:
        return "B"
    elif score >= 70:
        return "C"
    elif score >= 60:
        return "D"
    else:
        return "F"

# Example output
print(grade(95))  # Expected: A
print(grade(85))  # Expected: B
print(grade(75))  # Expected: C
print(grade(65))  # Expected: D
print(grade(55))  # Expected: F
```

Output:



```
[Running] python -u "e:\AIcoding\AIcoding\10.3.6.py"
A
B
C
D
F

[Done] exited with code=0 in 0.183 seconds
```