Ви

i have this code, and i want you to make function of this, make uniitests for it: a=int(input()) b=int(input()) c=int(input()) d=int(input()) e=int(input()) if (a or b or c or d or e)>100 or (a or b or c or d or e)<0: print(None) exit()

grade=round(((a+b+c+d+e)/5), 1)

if 100>=grade>=90: print(f"Average grade = {grade} -> A") elif 90>grade>=80: print(f"Average grade = {grade} -> B") elif 80>grade>=70: print(f"Average grade = {grade} -> C") elif 70>grade>=60: print(f"Average grade = {grade} -> D") else: print(f"Average grade = {grade} -> F")

Copilot

Sure, I can help you with that. Here's your code refactored into a function and a corresponding unit test using the `unittest` module in Python:

```python
def calculate_grade(a, b, c, d, e):
    if (a or b or c or d or e)>100 or (a or b or c or d or e)<0:
        return None

    grade=round(((a+b+c+d+e)/5), 1)

    if 100>=grade>=90:
        return f"Average grade = {grade} -> A"
    elif 90>grade>=80:
        return f"Average grade = {grade} -> B"
    elif 80>grade>=70:
        return f"Average grade = {grade} -> C"
    elif 70>grade>=60:
        return f"Average grade = {grade} -> D"
    else:
        return f"Average grade = {grade} -> F"

# Unit tests
import unittest

class TestGrades(unittest.TestCase):
    def test_calculate_grade(self):
        self.assertEqual(calculate_grade(90, 85, 95, 88, 92), "Average grade = 90.0 -> A")
        self.assertEqual(calculate_grade(80, 85, 75, 88, 82), "Average grade = 82.0 -> B")
        self.assertEqual(calculate_grade(70, 65, 75, 68, 72), "Average grade = 70.0 -> C")
        self.assertEqual(calculate_grade(60, 65, 55, 68, 62), "Average grade = 62.0 -> D")
        self.assertEqual(calculate_grade(50, 45, 55, 48, 52), "Average grade = 50.0 -> F")
        self.assertEqual(calculate_grade(110, 85, 95, 88, 92), None)
        self.assertEqual(calculate_grade(-10, 85, 95, 88, 92), None)

if __name__ == '__main__':
    unittest.main()
```

This code defines a function `calculate_grade` that takes five grades as input and returns the average grade and corresponding letter grade. The `TestGrades` class contains unit tests for this function, checking various grade combinations and edge cases. You can run these tests by running the script. If all tests pass, it means your function is working as expected. If any test fails, it will raise an error with a message indicating what went wrong. This can help you identify and fix problems in your function. Remember to always write tests for your functions to ensure they work correctly!

Ви

try to write some edge cases for it

Copilot

Sure, here are some additional edge cases that you can consider for your unit tests:

```
class TestGrades(unittest.TestCase):
    def test_calculate_grade(self):
        # Existing tests...

        # Edge cases
        self.assertEqual(calculate_grade(0, 0, 0, 0, 0), "Average grade = 0.0 -> F")
        self.assertEqual(calculate_grade(100, 100, 100, 100, 100), "Average grade = 100.0 -> A")
        self.assertEqual(calculate_grade(100, 0, 100, 0, 100), "Average grade = 60.0 -> D")
        self.assertEqual(calculate_grade(0, 100, 0, 100, 0), "Average grade = 40.0 -> F")
        self.assertEqual(calculate_grade(50, 50, 50, 50, 50), "Average grade = 50.0 -> F")
        self.assertEqual(calculate_grade(101, 85, 95, 88, 92), None)
        self.assertEqual(calculate_grade(-1, 85, 95, 88, 92), None)

if __name__ == '__main__':
    unittest.main()
```

These tests cover scenarios where all grades are the same, grades are at the maximum and minimum possible values, and grades are just outside the valid range. These edge cases can help ensure your function handles all possible inputs correctly. Remember, good test coverage is key to reliable code!

Ви

does my function right, cause if any attribute but not first, is less than 0 it doesn't return None

Copilot

You're correct. The current condition `(a or b or c or d or e)>100 or (a or b or c or d or e)<0` in your function isn't checking each variable individually. The `or` operator in Python returns the first "truthy" value it encounters, or the last value if none are "truthy". So if `a` is not 0, it won't check `b`, `c`, `d`, or `e`.

To fix this, you should check each variable individually like this:

```
def calculate_grade(a, b, c, d, e):
    if any(i > 100 or i < 0 for i in [a, b, c, d, e]):
        return None

    grade=round(((a+b+c+d+e)/5), 1)

    if 100>=grade>=90:
        return f"Average grade = {grade} -> A"
    elif 90>grade>=80:
        return f"Average grade = {grade} -> B"
    elif 80>grade>=70:
        return f"Average grade = {grade} -> C"
    elif 70>grade>=60:
        return f"Average grade = {grade} -> D"
    else:
        return f"Average grade = {grade} -> F"
```

In this updated function, the `any()` function is used with a generator expression to check each grade individually. If any grade is less than 0 or greater than 100, the function immediately returns `None`. This ensures that all grades are validated correctly. Good catch! 👍

Ви

i can't use list, also write unittest for all edge cases

Copilot

I see, if you can't use lists, you can modify the condition to check each variable individually. Here's the updated function and the corresponding unit tests:

```python
def calculate_grade(a, b, c, d, e):
    if a > 100 or a < 0 or b > 100 or b < 0 or c > 100 or c < 0 or d > 100 or d < 0 or e > 100 or e < 0:
        return None

    grade=round(((a+b+c+d+e)/5), 1)

    if 100>=grade>=90:
        return f"Average grade = {grade} -> A"
    elif 90>grade>=80:
        return f"Average grade = {grade} -> B"
    elif 80>grade>=70:
        return f"Average grade = {grade} -> C"
    elif 70>grade>=60:
        return f"Average grade = {grade} -> D"
    else:
        return f"Average grade = {grade} -> F"

# Unit tests
import unittest

class TestGrades(unittest.TestCase):
    def test_calculate_grade(self):
        self.assertEqual(calculate_grade(90, 85, 95, 88, 92), "Average grade = 90.0 -> A")
        self.assertEqual(calculate_grade(80, 85, 75, 88, 82), "Average grade = 82.0 -> B")
        self.assertEqual(calculate_grade(70, 65, 75, 68, 72), "Average grade = 70.0 -> C")
        self.assertEqual(calculate_grade(60, 65, 55, 68, 62), "Average grade = 62.0 -> D")
        self.assertEqual(calculate_grade(50, 45, 55, 48, 52), "Average grade = 50.0 -> F")
        self.assertEqual(calculate_grade(110, 85, 95, 88, 92), None)
        self.assertEqual(calculate_grade(-10, 85, 95, 88, 92), None)
        self.assertEqual(calculate_grade(0, 0, 0, 0, 0), "Average grade = 0.0 -> F")
        self.assertEqual(calculate_grade(100, 100, 100, 100, 100), "Average grade = 100.0 -> A")
        self.assertEqual(calculate_grade(100, 0, 100, 0, 100), "Average grade = 60.0 -> D")
        self.assertEqual(calculate_grade(0, 100, 0, 100, 0), "Average grade = 40.0 -> F")
        self.assertEqual(calculate_grade(50, 50, 50, 50, 50), "Average grade = 50.0 -> F")
        self.assertEqual(calculate_grade(101, 85, 95, 88, 92), None)
        self.assertEqual(calculate_grade(-1, 85, 95, 88, 92), None)

if __name__ == '__main__':
    unittest.main()
```

This code now checks each grade individually in the `calculate_grade` function. The unit tests cover a variety of scenarios, including edge cases where grades are at the maximum and minimum possible values, and grades are just outside the valid range. These tests can help ensure your function handles all possible inputs correctly. Remember, good test coverage is key to reliable code! 👍