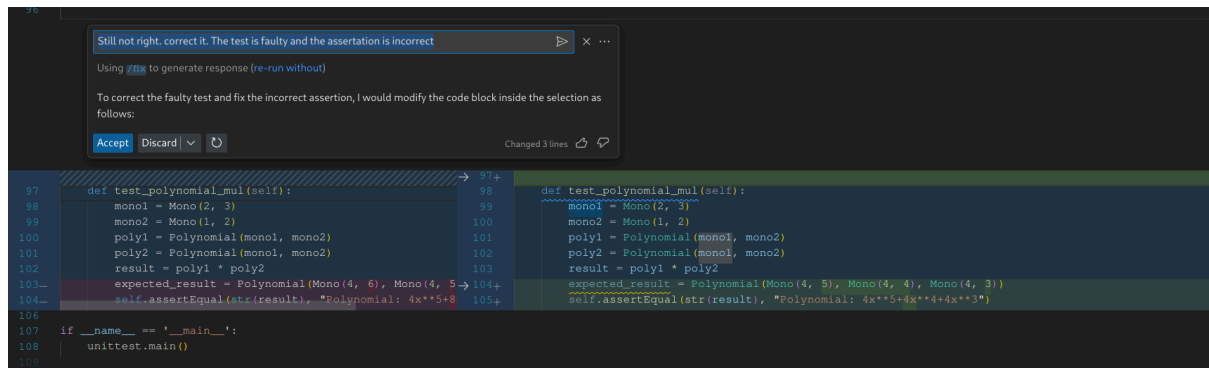


After the generation of the tests, they were correct, but it did not cover a whole lot of the edge cases.

I changed those tests to be correct, but after provide prompts, it couldn't manage to find the mistake



Still not right. correct it. The test is faulty and the assertion is incorrect

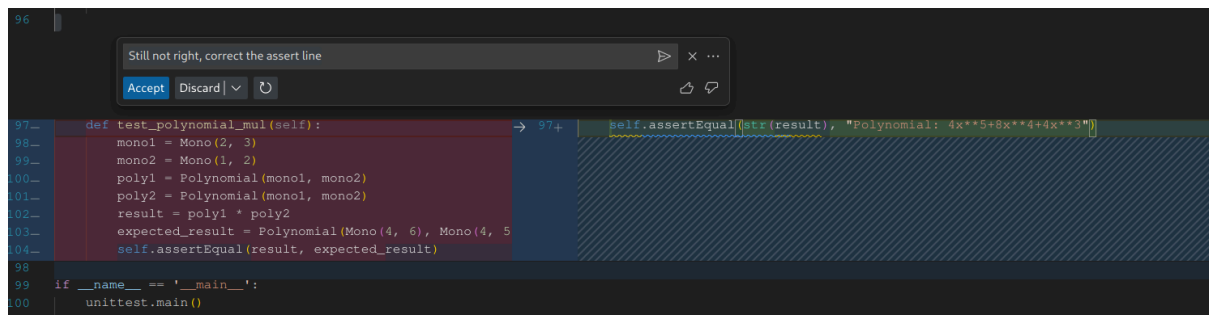
Using `fix` to generate response (re-run without)

To correct the faulty test and fix the incorrect assertion, I would modify the code block inside the selection as follows:

Accept Discard ↕ ↻ Changed 3 lines

```
97 def test_polynomial_mul(self):
98     mono1 = Mono(2, 3)
99     mono2 = Mono(1, 2)
100     poly1 = Polynomial(mono1, mono2)
101     poly2 = Polynomial(mono1, mono2)
102     result = poly1 * poly2
103     expected_result = Polynomial(Mono(4, 6), Mono(4, 5))
104     self.assertEqual(str(result), "Polynomial: 4x**5+8")
105
106 if __name__ == '__main__':
107     unittest.main()
108
```

```
97 def test_polynomial_mul(self):
98     mono1 = Mono(2, 3)
99     mono2 = Mono(1, 2)
100     poly1 = Polynomial(mono1, mono2)
101     poly2 = Polynomial(mono1, mono2)
102     result = poly1 * poly2
103     expected_result = Polynomial(Mono(4, 5), Mono(4, 4), Mono(4, 3))
104     self.assertEqual(str(result), "Polynomial: 4x**5+4x**4+4x**3")
105
```

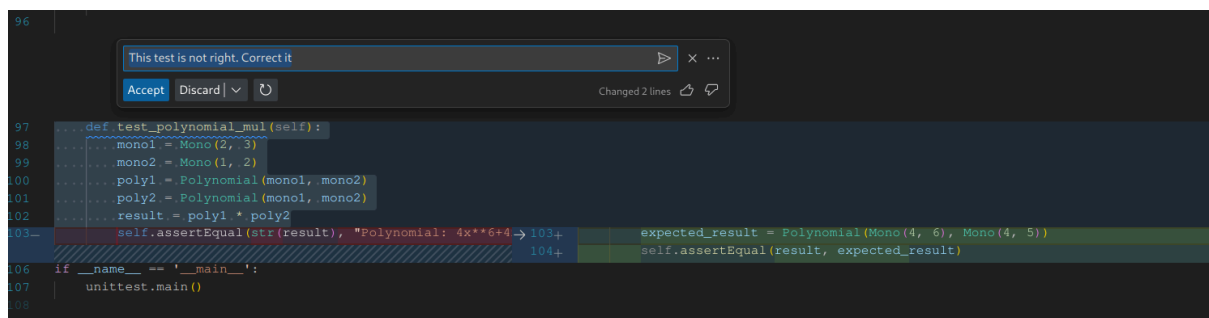


Still not right, correct the assert line

Accept Discard ↕ ↻

```
97 def test_polynomial_mul(self):
98     mono1 = Mono(2, 3)
99     mono2 = Mono(1, 2)
100     poly1 = Polynomial(mono1, mono2)
101     poly2 = Polynomial(mono1, mono2)
102     result = poly1 * poly2
103     expected_result = Polynomial(Mono(4, 6), Mono(4, 5))
104     self.assertEqual(result, expected_result)
105
106 if __name__ == '__main__':
107     unittest.main()
108
```

```
97 self.assertEqual(str(result), "Polynomial: 4x**5+8x**4+4x**3")
104
```



This test is not right. Correct it

Accept Discard ↕ ↻ Changed 2 lines

```
97 def test_polynomial_mul(self):
98     mono1 = Mono(2, 3)
99     mono2 = Mono(1, 2)
100     poly1 = Polynomial(mono1, mono2)
101     poly2 = Polynomial(mono1, mono2)
102     result = poly1 * poly2
103     self.assertEqual(str(result), "Polynomial: 4x**6+4")
104     expected_result = Polynomial(Mono(4, 6), Mono(4, 5))
105     self.assertEqual(result, expected_result)
106
107 if __name__ == '__main__':
108     unittest.main()
109
```

```
97 def test_polynomial_mul(self):
98     mono1 = Mono(2, 3)
99     mono2 = Mono(1, 2)
100     poly1 = Polynomial(mono1, mono2)
101     poly2 = Polynomial(mono1, mono2)
102     result = poly1 * poly2
103     self.assertEqual(str(result), "Polynomial: 4x**6+4")
104     expected_result = Polynomial(Mono(4, 6), Mono(4, 5))
105     self.assertEqual(result, expected_result)
106
107 if __name__ == '__main__':
108     unittest.main()
109
```

The conclusion is that it does some edge cases, but if it comes complex enough, it fails to accomplish the assignment. It may correct wrong unit tests, but it can do it with some well-known algorithms or with small-scale code.

Usually in the cmc the tests cover almost the whole code, making it the best testing of a student's work. But in the case of github copilot, it could not resolve one test that was failing, making the testing not so good.

The code with faulty testing it could not resolve.

```
import unittest
from polynomial import Mono, Polynomial

class TestMono(unittest.TestCase):
    def test_mono_str(self):
```

```

        mono = Mono(2, 3)
        self.assertEqual(str(mono), "Mono: 2x**3")

    def test_mono_repr(self):
        mono = Mono(2, 3)
        self.assertEqual(repr(mono), "Mono(coeff=2, degree=3)")

    def test_mono_eq(self):
        mono1 = Mono(2, 3)
        mono2 = Mono(2, 3)
        self.assertEqual(mono1, mono2)

class TestPolynomial(unittest.TestCase):
    def test_polynomial_str(self):
        mono1 = Mono(2, 3)
        mono2 = Mono(1, 1)
        poly = Polynomial(mono1, mono2)
        self.assertEqual(str(poly), "Polynomial: 2x**3+x")

    def test_polynomial_repr(self):
        mono1 = Mono(2, 3)
        mono2 = Mono(1, 2)
        poly = Polynomial(mono1, mono2)
        self.assertEqual(repr(poly), "Polynomial(Mono(coeff=2, degree=3)
-> Mono(coeff=1, degree=2))")

    def test_polynomial_degree(self):
        mono1 = Mono(2, 3)
        mono2 = Mono(1, 2)
        poly = Polynomial(mono1, mono2)
        self.assertEqual(poly.degree, 3)

    def test_polynomial_copy(self):
        mono1 = Mono(2, 3)
        mono2 = Mono(1, 2)
        poly1 = Polynomial(mono1, mono2)
        poly2 = poly1.copy()
        self.assertEqual(poly1, poly2)

    def test_polynomial_sort(self):
        mono1 = Mono(2, 3)
        mono2 = Mono(1, 2)
        mono3 = Mono(3, 1)

```

```

poly = Polynomial(mono1, mono2, mono3)
poly.sort()
self.assertEqual(str(poly), "Polynomial: 2x**3+x**2+3x")

def test_polynomial_simplify(self):
    mono1 = Mono(2, 3)
    mono2 = Mono(-2, 3)
    mono3 = Mono(1, 2)
    poly = Polynomial(mono1, mono2, mono3)
    poly.simplify()
    self.assertEqual(str(poly), "Polynomial: x**2")

def test_polynomial_eval_at(self):
    mono1 = Mono(2, 3)
    mono2 = Mono(1, 2)
    poly = Polynomial(mono1, mono2)
    result = poly.eval_at(2)
    self.assertEqual(result, 20)

def test_polynomial_eq(self):
    mono1 = Mono(2, 3)
    mono2 = Mono(1, 2)
    poly1 = Polynomial(mono1, mono2)
    poly2 = Polynomial(mono1, mono2)
    self.assertEqual(poly1, poly2)

def test_polynomial_hash(self):
    mono1 = Mono(2, 3)
    mono2 = Mono(1, 2)
    poly = Polynomial(mono1, mono2)
    result = hash(poly)
    self.assertIsInstance(result, int)

def test_polynomial_add(self):
    mono1 = Mono(2, 3)
    mono2 = Mono(1, 2)
    poly1 = Polynomial(mono1, mono2)
    poly2 = Polynomial(mono1, mono2)
    result = poly1 + poly2
    self.assertEqual(str(result), "Polynomial: 4x**3+2x**2")

def test_polynomial_sub(self):
    mono1 = Mono(2, 3)

```

```

mono2 = Mono(1, 2)
poly1 = Polynomial(mono1, mono2)
poly2 = Polynomial(mono1, mono2)
result = poly1 - poly2
self.assertEqual(str(result), "Polynomial: 0")

def test_polynomial_mul(self):
    mono1 = Mono(2, 3)
    mono2 = Mono(1, 2)
    poly1 = Polynomial(mono1, mono2)
    poly2 = Polynomial(mono1, mono2)
    result = poly1 * poly2
    expected_result = Polynomial(Mono(4, 5), Mono(4, 4), Mono(4, 3))
    self.assertEqual(str(result), "Polynomial: 4x**5+4x**4+4x**3")

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

```

And with optimizing the tests. It does indeed do it, but in the process it can ruin everything, like it did with the initial tests.

```

FAIL: test_polynomial_str (__main__.TestPolynomial.test_polynomial_str)
-----
Traceback (most recent call last):
  File "/home/julfy/Documents/2nd_term/my_mini_projects/miniproject2/black_dungeon_mAisters-1/polynomial_test.py", line 25, in test_polynomial_str
    self.assertEqual(str(self.poly), "Polynomial: 2x**3+x")
AssertionError: 'Polynomial: 2x**3+x**2' != 'Polynomial: 2x**3+x'
- Polynomial: 2x**3+x**2
?                  ---
+ Polynomial: 2x**3+x


-----
Ran 15 tests in 0.001s

FAILED (failures=3)
➔ black_dungeon_mAisters-1 git:(BohdanchickHarhuiHlopcjick) ✖

```

And the coverage is still not full, even though I asked for it. It describes the code, but it is obvious.

What did you cover in these tests?

 GitHub Copilot



Used 1 reference >

The tests in this Python script are covering various operations and functionalities of a `Polynomial` class:

1. `test_polynomial_eq`: This test checks the equality operation of the `Polynomial` class. It creates a new polynomial `poly2` as a copy of `self.poly` and then asserts that they are equal.
2. `test_polynomial_hash`: This test checks the hash operation of the `Polynomial` class. It calculates the hash of `self.poly` and asserts that the result is an integer.
3. `test_polynomial_add`: This test checks the addition operation of the `Polynomial` class. It creates a new polynomial `poly2` as a copy of `self.poly`, adds them together, and then asserts that the result is as expected.
4. `test_polynomial_sub`: This test checks the subtraction operation of the `Polynomial` class. It creates a new polynomial `poly2` as a copy of `self.poly`, subtracts `poly2` from `self.poly`, and then asserts that the result is zero.
5. `test_polynomial_mul`: This test checks the multiplication operation of the `Polynomial` class. It creates a new polynomial `poly2` as a copy of `self.poly`, multiplies them together, and then asserts that the result is as expected.

The `if __name__ == '__main__': unittest.main()` block at the end runs all the tests when the script is executed directly.

And still it didn't write about all the test it wrote, so I assume, that it has its own scope of interest.