AI Powered Code-Review and Testing

Language: python

Your Code:

```
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True

def prime_numbers_up_to_n(n):
    primes = []
    for i in range(1, n+1):
        if is_prime(i):
            primes.append(i)
    return primes

# Run function
print(prime_numbers_up_to_n(50))
```

Code Review:

The code is correct but it is not fully optimized and does not follow all the best practices. Here are some potential improvements:

1. **Efficiency**: In the `is_prime(n)` function, you are checking all the numbers up to `n` to see whether they are factors of `n`. This is unnecessary. It is enough to check up to `sqrt(n)` because a larger factor of `n` would be a multiple of smaller factor that has already been checked.

2. **Naming convention**: Function names are good but it could be better if they are more representative of what they do. So, `prime_numbers_up_to_n(n)` could be renamed to `find_primes(n)` to clearly represent its purpose.

3. **Usage of list comprehensions** over traditional loops where possible is a more pythonic way of doing things and also faster.

Here's an improved version of your code:

```python
import math

def is_prime(n):
    if n < 2:
        return False
    for i in range(2, math.isqrt(n) + 1):  # only check up to sqrt(n)
        if n % i == 0:
            return False
    return True

def find_primes(n):
    return [i for i in range(1, n + 1) if is_prime(i)]  # list comprehension

# Run function
print(find_primes(50))
```

This version of the code reduces the unnecessary computations, making it more efficient. Also, it uses list comprehension, which makes the code more concise and pythonic. And `find_primes` now talks clearly about its purpose.

Generated Tests:

```
import unittest
import math

def is_prime(n):
    if n <= 1:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False

    sqrt_n = int(math.floor(math.sqrt(n)))
    for i in range(3, sqrt_n + 1, 2):
        if n % i == 0:
            return False
    return True

def prime_numbers_up_to_n(n):
    primes = []
    for possible_prime in range(2, n + 1):
        if is_prime(possible_prime):
            primes.append(possible_prime)
    return primes

class TestPrimeFunctions(unittest.TestCase):

    def test_is_prime(self):
        self.assertEqual(is_prime(2), True)
        self.assertEqual(is_prime(3), True)
        self.assertEqual(is_prime(4), False)
        self.assertEqual(is_prime(29), True)
        self.assertRaises(ValueError, is_prime, -1)

    def test_prime_numbers_up_to_n(self):
        self.assertEqual(prime_numbers_up_to_n(10), [2, 3, 5, 7])
        self.assertEqual(prime_numbers_up_to_n(30), [2, 3, 5, 7, 11, 13, 17, 19, 23, 29])
        self.assertRaises(ValueError, prime_numbers_up_to_n, -1)

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

Test Report:

```
============================= test session starts =============================
platform win32 -- Python 3.11.9, pytest-8.3.4, pluggy-1.5.0
rootdir: C:\Users\Kamal\AppData\Local\Temp\tmpa8b80_3j
plugins: anyio-4.8.0
collected 2 items

test_generated.py FF                                                     [100%]

================================== FAILURES ===================================
_______________________ TestPrimeFunctions.test_is_prime _______________________
```

```
self = <test_generated.TestPrimeFunctions testMethod=test_is_prime>

    def test_is_prime(self):
        self.assertEqual(is_prime(2), True)
        self.assertEqual(is_prime(3), True)
        self.assertEqual(is_prime(4), False)
        self.assertEqual(is_prime(29), True)
>       self.assertRaises(ValueError, is_prime, -1)
E       AssertionError: ValueError not raised by is_prime

test_generated.py:32: AssertionError
________________ TestPrimeFunctions.test_prime_numbers_up_to_n _________________

self = <test_generated.TestPrimeFunctions testMethod=test_prime_numbers_up_to_n>

    def test_prime_numbers_up_to_n(self):
        self.assertEqual(prime_numbers_up_to_n(10), [2, 3, 5, 7])
        self.assertEqual(prime_numbers_up_to_n(30), [2, 3, 5, 7, 11, 13, 17, 19, 23, 29])
>       self.assertRaises(ValueError, prime_numbers_up_to_n, -1)
E       AssertionError: ValueError not raised by prime_numbers_up_to_n

test_generated.py:37: AssertionError
=========================== short test summary info ============================
FAILED test_generated.py::TestPrimeFunctions::test_is_prime - AssertionError:...
FAILED test_generated.py::TestPrimeFunctions::test_prime_numbers_up_to_n - As...
============================== 2 failed in 0.09s ===============================
```