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 current code works correctly and follows Python's syntax rules, but it is not the most efficient solution. Here are areas on which it could be improved:

1. The `is_prime(n)` function has a time complexity of O(N) because of the loop `for i in range(2, n)`. This can be improved to O(sqrt(N)) by changing the range to `for i in range(2, int(sqrt(n)) + 1)` because a larger factor of the number must be a multiple of a smaller factor that has already been checked.

2. The function `is_prime(n)` is called for all numbers from 1 to n in the function `prime_numbers_up_to_n(n)`. This could be optimized with the use of a Sieve of Eratosthenes, a simple, ancient algorithm for finding all prime numbers up to any given limit.

Here is an optimized version of your code with the above suggestions implemented:

```python
import math

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

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

print(prime_numbers_up_to_n(50))
```

In this code, `math.isqrt(n)` returns the integer square root of n and it's used to optimize the is_prime function. This function is available since Python 3.8. If you are using older versions, you can use `math.sqrt()` function with `int()` function: `int(math.sqrt(n))`.

Note that further efficiency could be achieved by replacing the `is_prime()` function and the `for i in range(2, n+1)` loop with an implementation of the Sieve of Eratosthenes.

Generated Tests:

```
import unittest

def is_prime(n):
    if n <= 1:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False
    for i in range(3, int(n**0.5) + 1, 2):
        if n % i == 0:
```

```
            return False
    return True

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


class TestPrimeFunctions(unittest.TestCase):

    def test_is_prime(self):
        self.assertFalse(is_prime(1), "is_prime(1) should be False")
        self.assertTrue(is_prime(2), "is_prime(2) should be True")
        self.assertTrue(is_prime(13), "is_prime(13) should be True")
        self.assertFalse(is_prime(24), "is_prime(24) should be False")
        with self.assertRaises(TypeError):
            is_prime('string')
        with self.assertRaises(TypeError):
            is_prime(None)

    def test_prime_numbers_up_to_n(self):
        self.assertEqual(prime_numbers_up_to_n(20),
                         [2, 3, 5, 7, 11, 13, 17, 19],
                         "Unexpected return for prime_numbers_up_to_n(20)")
        self.assertEqual(prime_numbers_up_to_n(1),
                         [],
                         "Unexpected return for prime_numbers_up_to_n(1)")
        with self.assertRaises(TypeError):
            prime_numbers_up_to_n('string')
        with self.assertRaises(TypeError):
            prime_numbers_up_to_n(None)


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\tmpqg7j_2od
plugins: anyio-4.8.0
collected 2 items

test_generated.py ..                                                     [100%]

============================== 2 passed in 0.03s ===============================
```