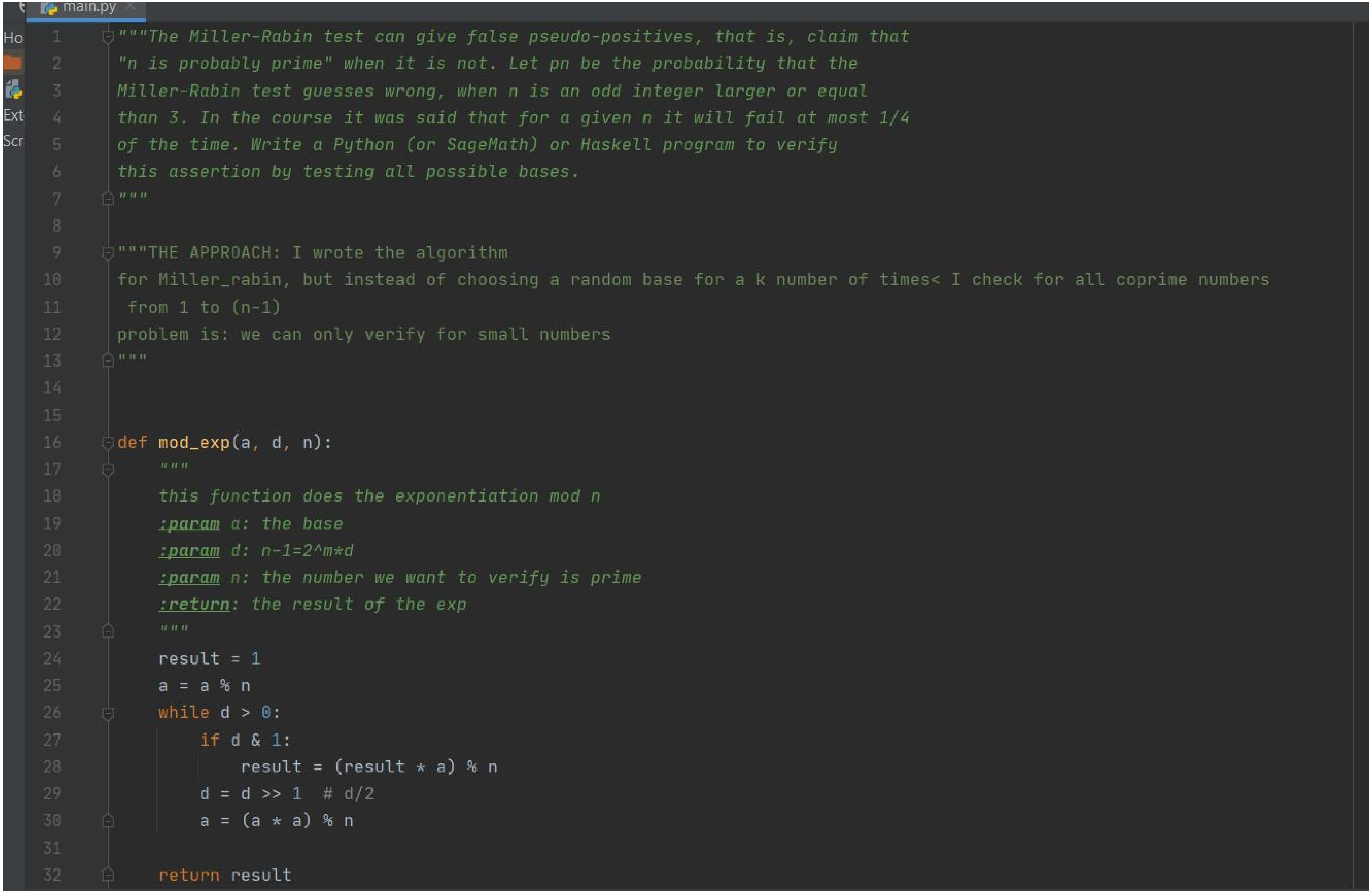
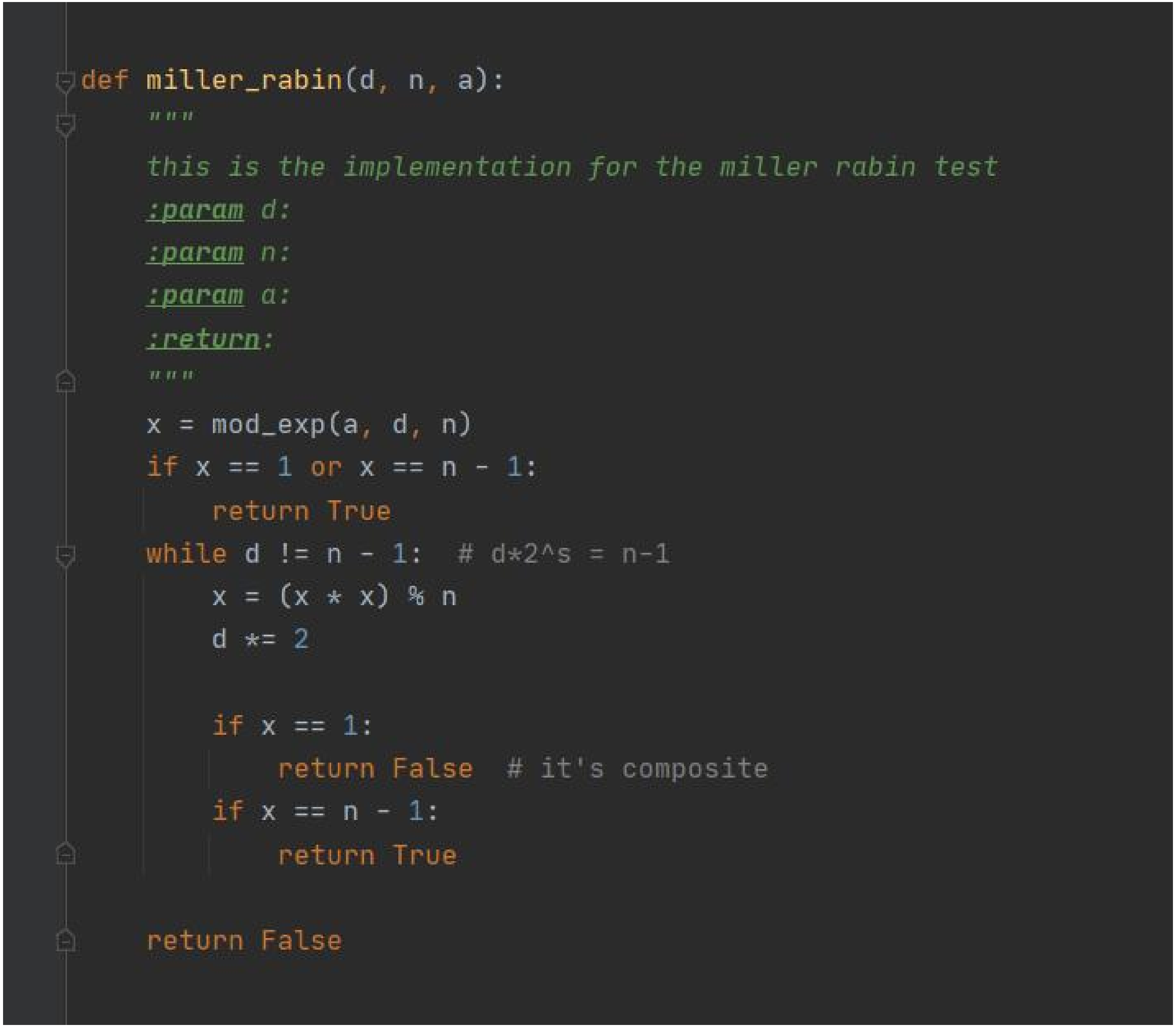
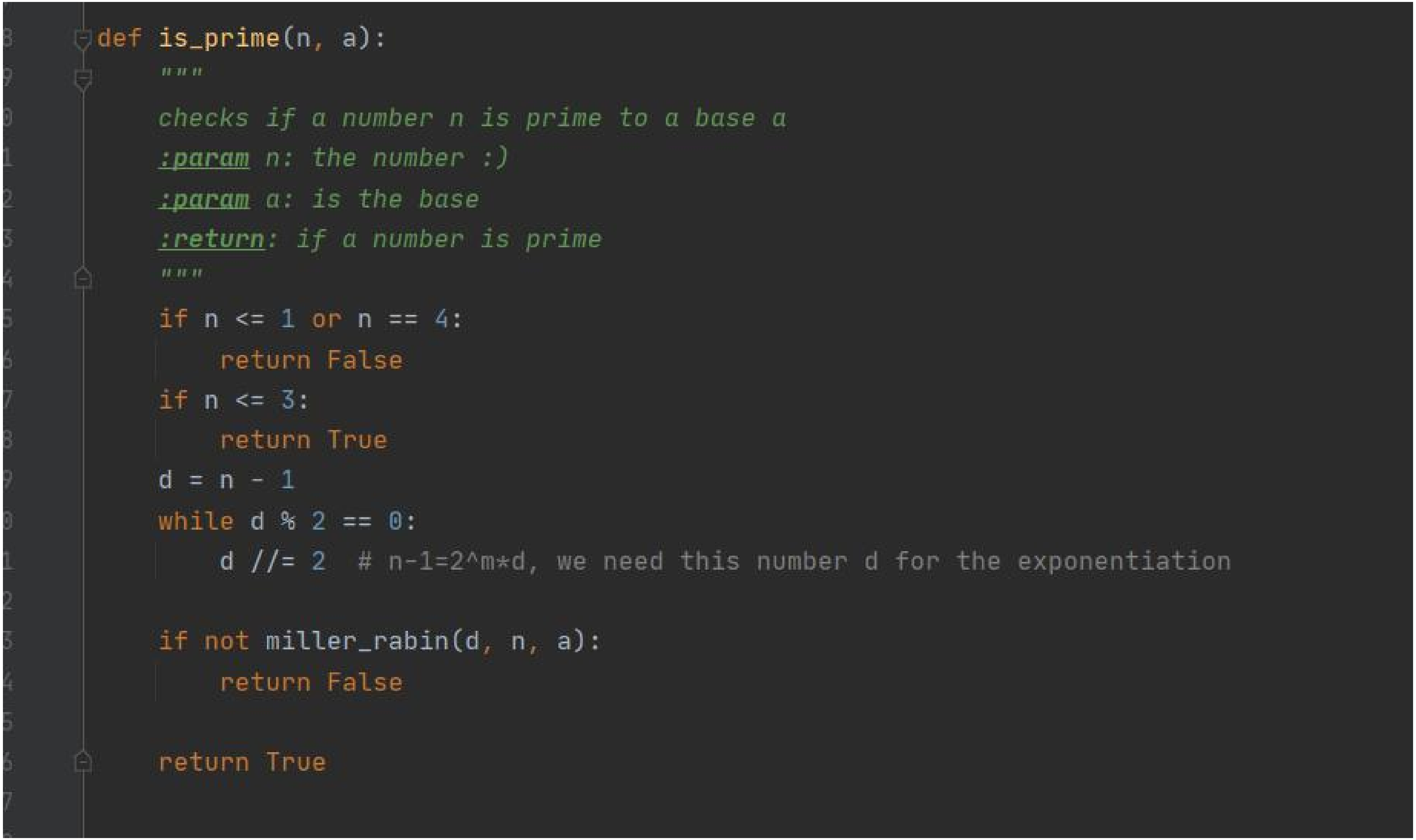


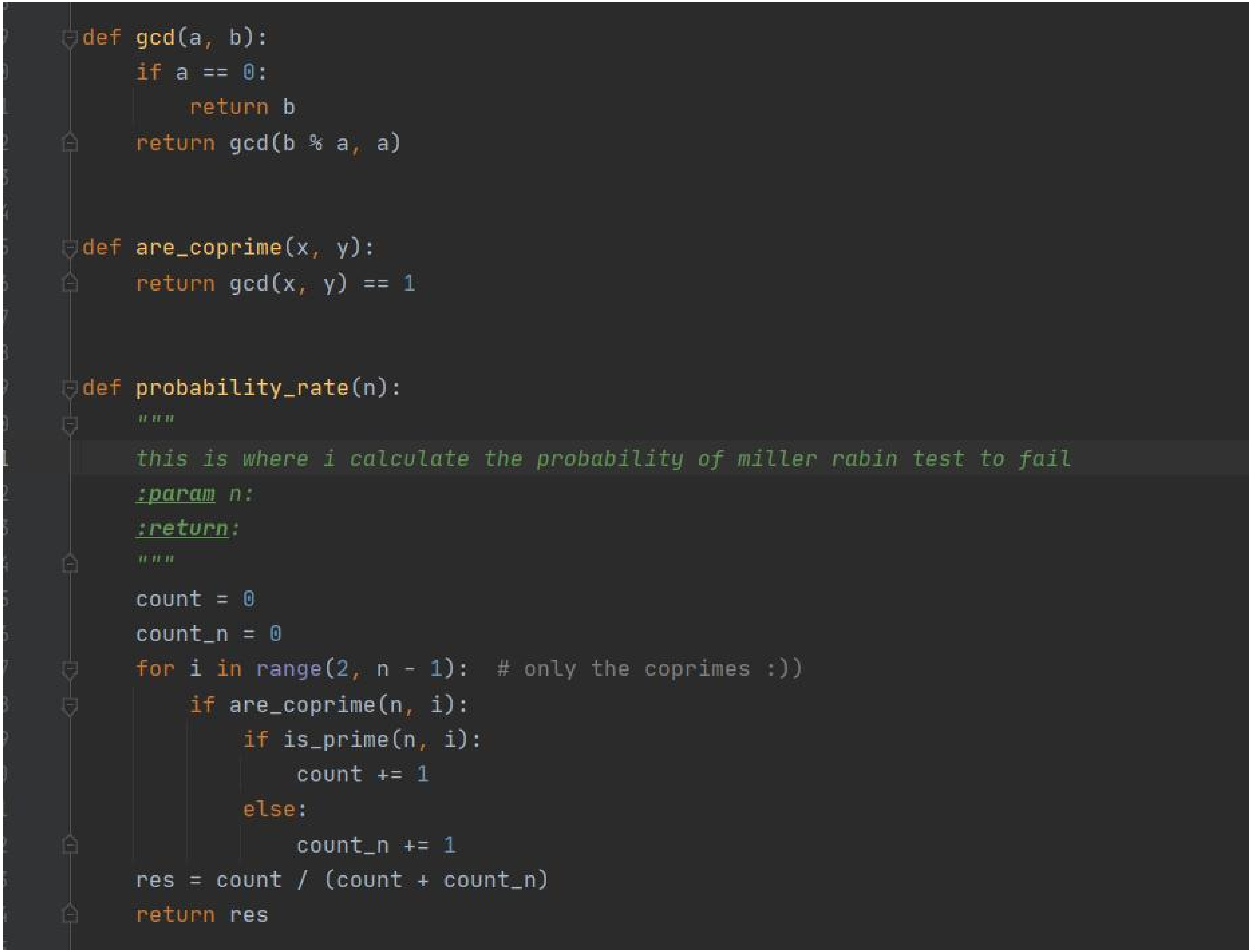
```python
"""The Miller-Rabin test can give false pseudo-positives, that is, claim that
"n is probably prime" when it is not. Let pn be the probability that the
Miller-Rabin test guesses wrong, when n is an odd integer larger or equal
than 3. In the course it was said that for a given n it will fail at most 1/4
of the time. Write a Python (or SageMath) or Haskell program to verify
this assertion by testing all possible bases.
"""

"""THE APPROACH: I wrote the algorithm
for Miller_rabin, but instead of choosing a random base for a k number of times< I check for all coprime numbers
 from 1 to (n-1)
problem is: we can only verify for small numbers
"""


def mod_exp(a, d, n):
    """
    this function does the exponentiation mod n
    :param a: the base
    :param d: n-1=2^m*d
    :param n: the number we want to verify is prime
    :return: the result of the exp
    """
    result = 1
    a = a % n
    while d > 0:
        if d & 1:
            result = (result * a) % n
        d = d >> 1  # d/2
        a = (a * a) % n

    return result
```



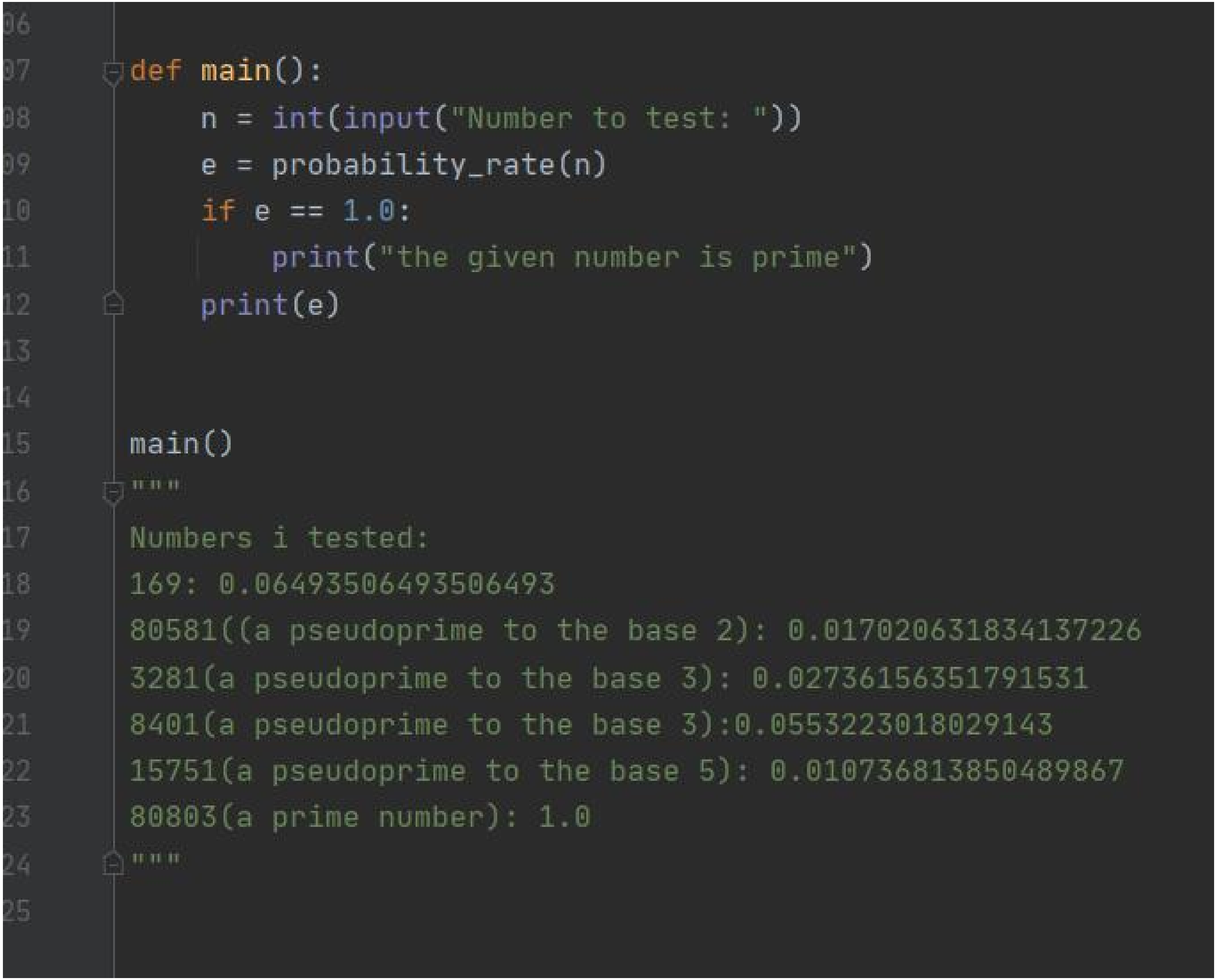
```python
def miller_rabin(d, n, a):
    """
    this is the implementation for the miller rabin test
    :param d:
    :param n:
    :param a:
    :return:
    """
    x = mod_exp(a, d, n)
    if x == 1 or x == n - 1:
        return True
    while d != n - 1:  # d*2^s = n-1
        x = (x * x) % n
        d *= 2

        if x == 1:
            return False  # it's composite
        if x == n - 1:
            return True

    return False
```



```python
def is_prime(n, a):
    """
    checks if a number n is prime to a base a
    :param n: the number :)
    :param a: is the base
    :return: if a number is prime
    """
    if n <= 1 or n == 4:
        return False
    if n <= 3:
        return True
    d = n - 1
    while d % 2 == 0:
        d //= 2  # n-1=2^m*d, we need this number d for the exponentiation

    if not miller_rabin(d, n, a):
        return False

    return True
```



```python
def gcd(a, b):
    if a == 0:
        return b
    return gcd(b % a, a)


def are_coprime(x, y):
    return gcd(x, y) == 1


def probability_rate(n):
    """
    this is where i calculate the probability of miller rabin test to fail
    :param n:
    :return:
    """
    count = 0
    count_n = 0
    for i in range(2, n - 1):  # only the coprimes :))
        if are_coprime(n, i):
            if is_prime(n, i):
                count += 1
            else:
                count_n += 1
    res = count / (count + count_n)
    return res
```



```python

def main():
    n = int(input("Number to test: "))
    e = probability_rate(n)
    if e == 1.0:
        print("the given number is prime")
    print(e)


main()
"""
Numbers i tested:
169: 0.06493506493506493
80581((a pseudoprime to the base 2): 0.017020631834137226
3281(a pseudoprime to the base 3): 0.0273615635179153l
8401(a pseudoprime to the base 3):0.05532230180291 43
15751(a pseudoprime to the base 5): 0.010736813850489867
80803(a prime number): 1.0
"""

```