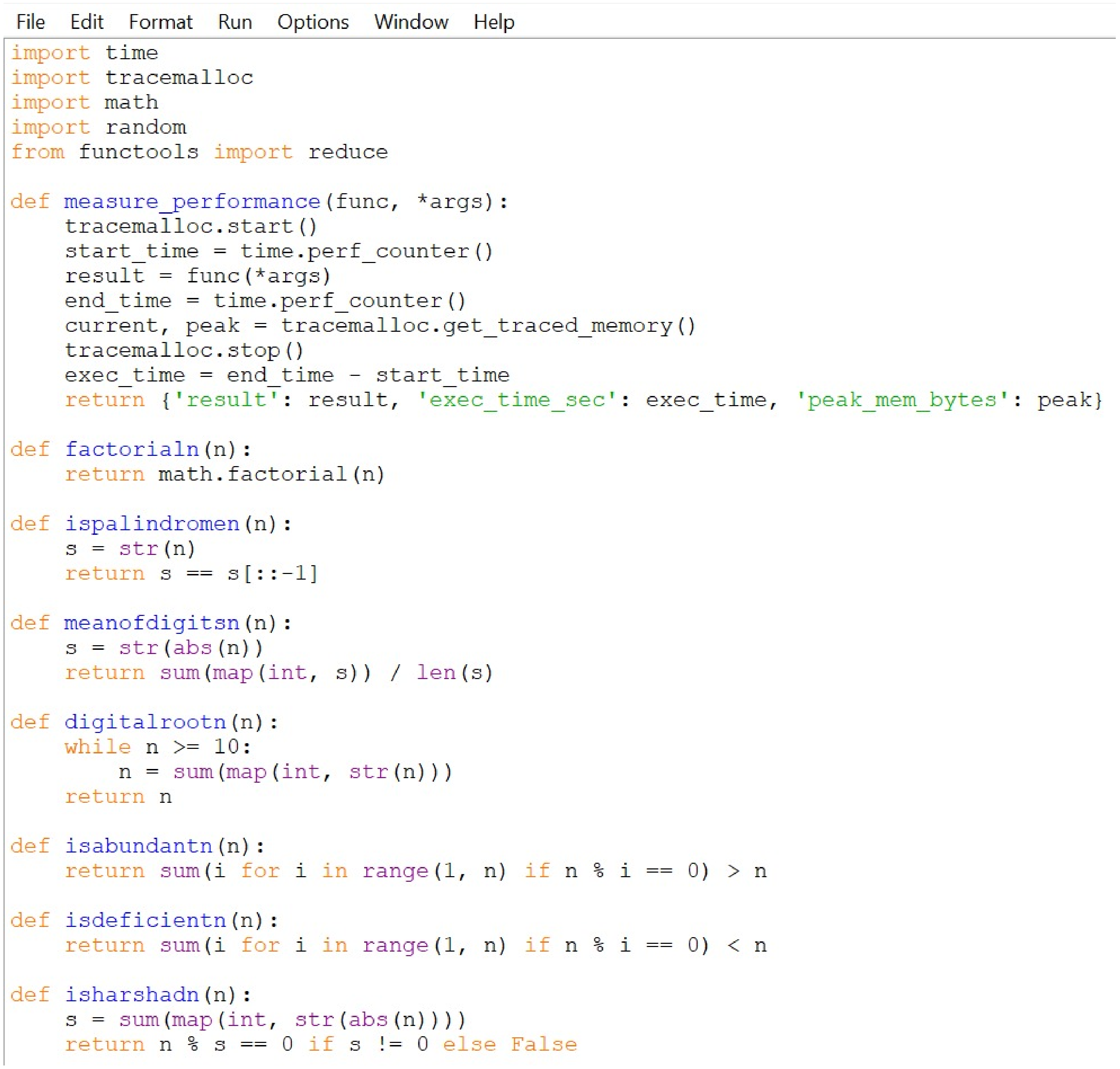
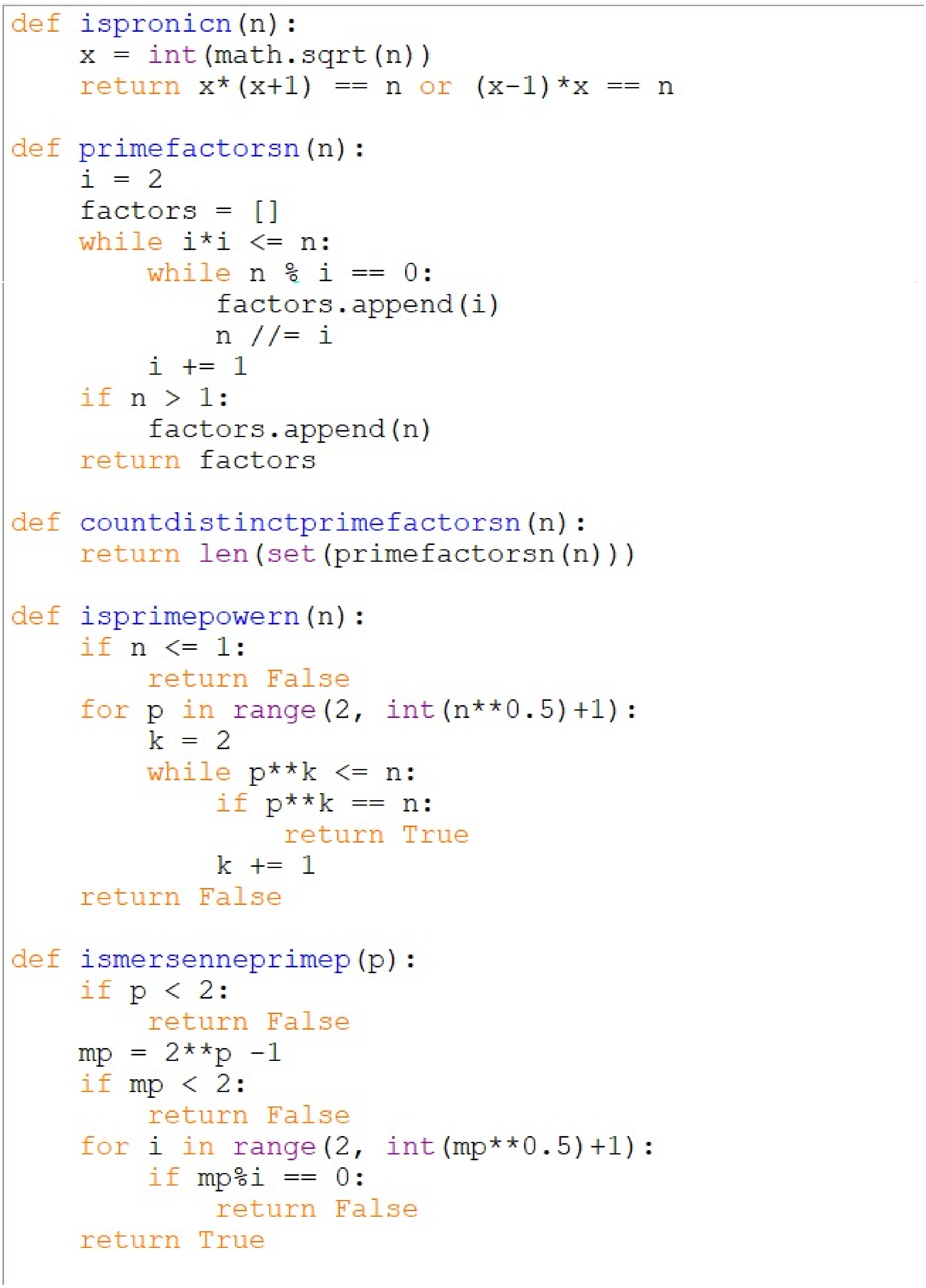


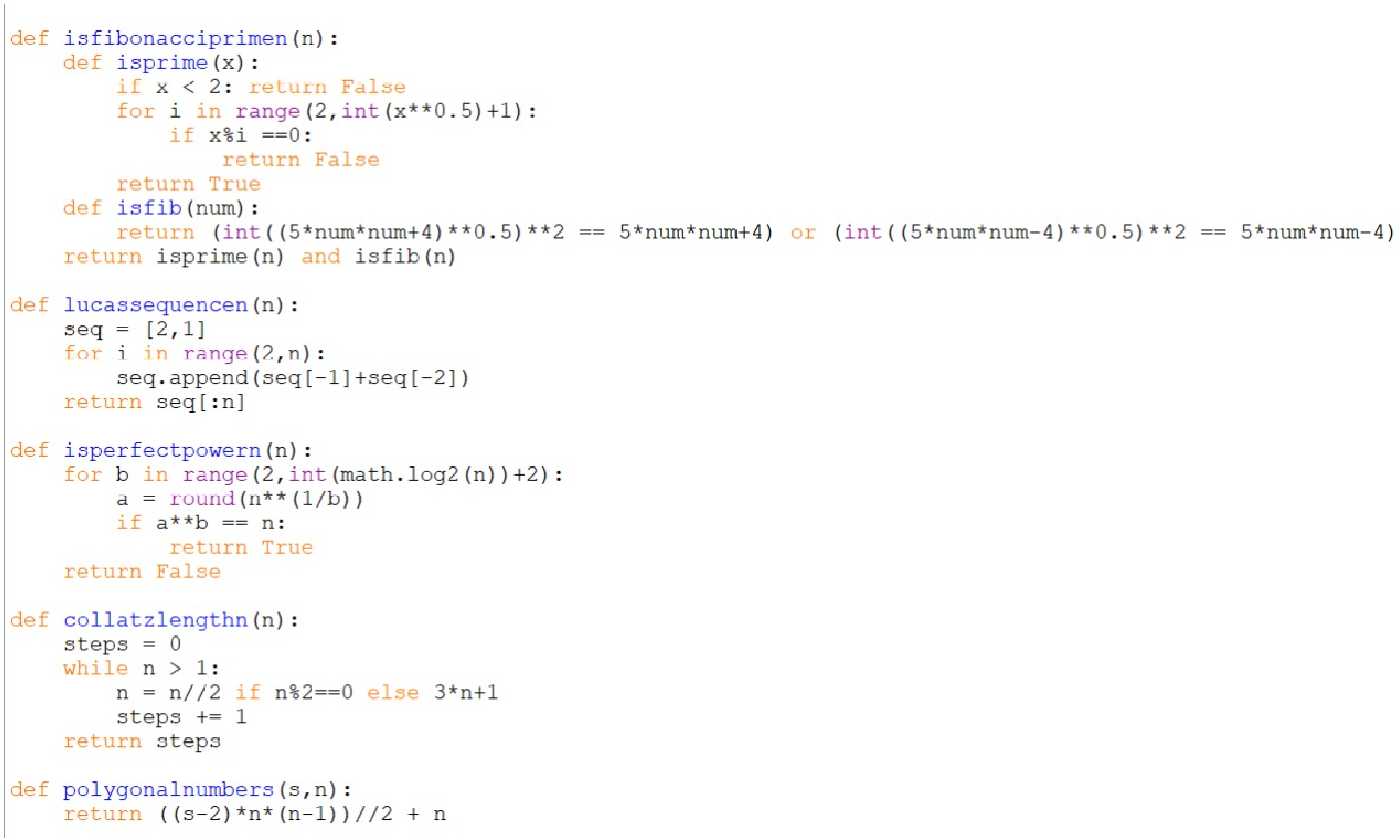
```python
import time
import tracemalloc
import math
import random
from functools import reduce

def measure_performance(func, *args):
    tracemalloc.start()
    start_time = time.perf_counter()
    result = func(*args)
    end_time = time.perf_counter()
    current, peak = tracemalloc.get_traced_memory()
    tracemalloc.stop()
    exec_time = end_time - start_time
    return {'result': result, 'exec_time_sec': exec_time, 'peak_mem_bytes': peak}

def factorialn(n):
    return math.factorial(n)

def ispalindromen(n):
    s = str(n)
    return s == s[::-1]

def meanofdigitsn(n):
    s = str(abs(n))
    return sum(map(int, s)) / len(s)

def digitalrootn(n):
    while n >= 10:
        n = sum(map(int, str(n)))
    return n

def isabundantn(n):
    return sum(i for i in range(1, n) if n % i == 0) > n

def isdeficientn(n):
    return sum(i for i in range(1, n) if n % i == 0) < n

def isharshadn(n):
    s = sum(map(int, str(abs(n))))
    return n % s == 0 if s != 0 else False
```



```python
def ispronicn(n):
    x = int(math.sqrt(n))
    return x*(x+1) == n or (x-1)*x == n

def primefactorsn(n):
    i = 2
    factors = []
    while i*i <= n:
        while n % i == 0:
            factors.append(i)
            n //= i
        i += 1
    if n > 1:
        factors.append(n)
    return factors

def countdistinctprimefactorsn(n):
    return len(set(primefactorsn(n)))

def isprimepowern(n):
    if n <= 1:
        return False
    for p in range(2, int(n**0.5)+1):
        k = 2
        while p**k <= n:
            if p**k == n:
                return True
            k += 1
    return False

def ismersenneprimep(p):
    if p < 2:
        return False
    mp = 2**p -1
    if mp < 2:
        return False
    for i in range(2, int(mp**0.5)+1):
        if mp%i == 0:
            return False
    return True
```

```python
def twinprimeslimit(limit):
    def isprime(x):
        if x < 2:
            return False
        for i in range(2, int(x**0.5)+1):
            if x%i == 0:
                return False
        return True
    result = []
    prev = 2
    for n in range(3, limit+1):
        if isprime(n):
            if n - prev == 2:
                result.append((prev, n))
            prev = n
    return result

def countdivisorsn(n):
    count = 0
    for i in range(1,int(n**0.5)+1):
        if n%i == 0:
            count += 2 if i*i != n else 1
    return count

def aliquotsumn(n):
    return sum(i for i in range(1,n) if n % i == 0)

def areamicable(a,b):
    return aliquotsumn(a) == b and aliquotsumn(b) == a

def multiplicativepersistencen(n):
    steps = 0
    while n >= 10:
        prod = 1
        for d in str(n):
            prod *= int(d)
        n = prod
        steps += 1
    return steps
```

```python
def ishighlycompositen(n):
    n_div = countdivisorsn(n)
    return all(n_div > countdivisorsn(k) for k in range(1,n))

def modexp(base, exponent, modulus):
    return pow(base, exponent, modulus)

def modinverse(a,m):
    for x in range(1,m):
        if (a*x) % m == 1:
            return x
    return None

def crtremaindersmoduli(remainders, moduli):
    def egcd(a,b):
        if a == 0:
            return (b,0,1)
        g,y,x = egcd(b%a, a)
        return (g, x-(b//a)*y, y)
    def modinv(a,m):
        g,x,y = egcd(a,m)
        if g!=1:
            return None
        return x % m
    prod = reduce(lambda a,b: a*b, moduli)
    result = 0
    for r,m in zip(remainders,moduli):
        p=prod//m
        result += r * modinv(p,m)*p
    return result % prod

def isquadraticresidue(a,p):
    return pow(a,(p-1)//2,p)==1

def ordermod(a,n):
    for k in range(1,n):
        if pow(a,k,n)==1:
            return k
    return None
```

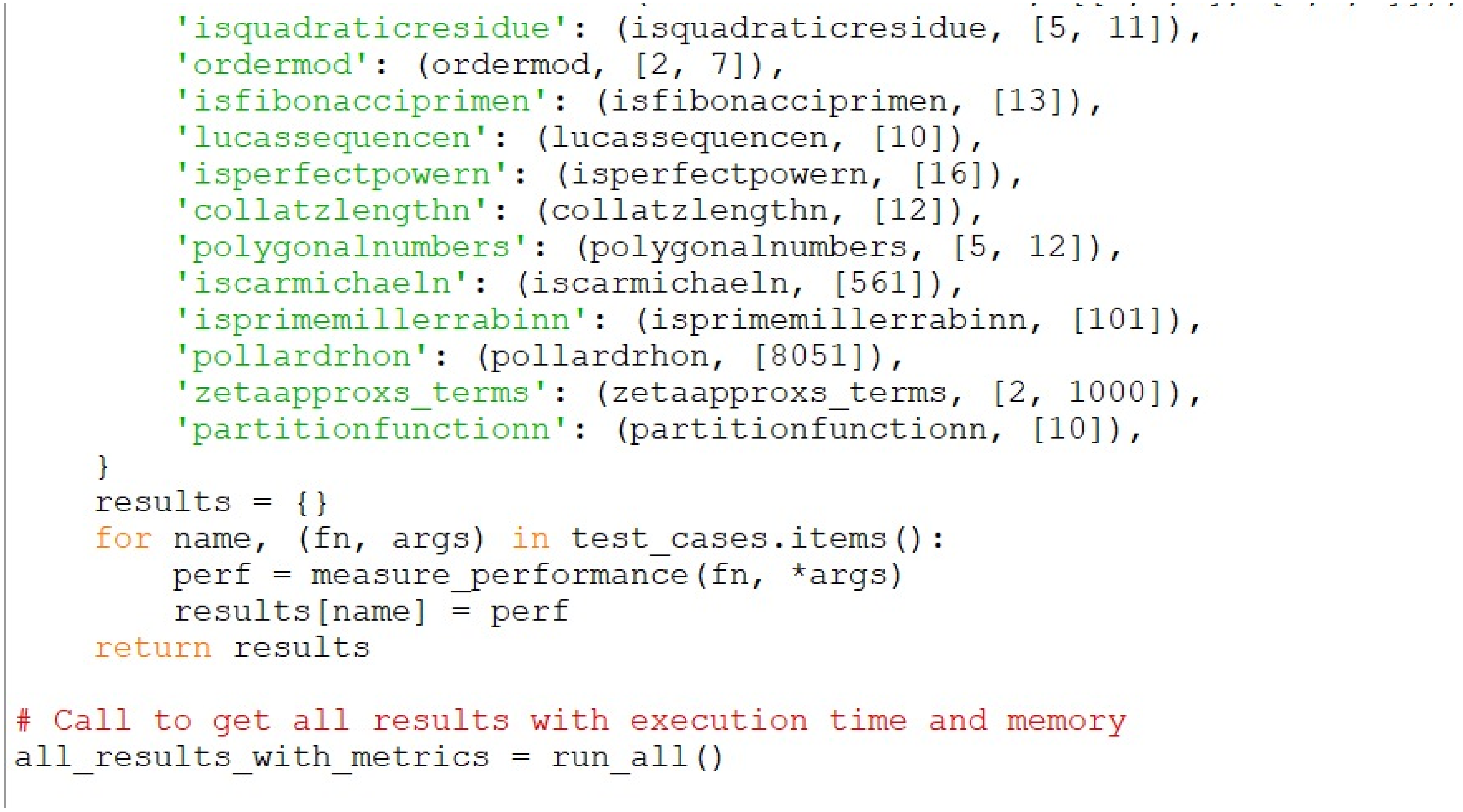


```python
def isfibonacciprimen(n):
    def isprime(x):
        if x < 2: return False
        for i in range(2,int(x**0.5)+1):
            if x%i ==0:
                return False
        return True
    def isfib(num):
        return (int((5*num*num+4)**0.5)**2 == 5*num*num+4) or (int((5*num*num-4)**0.5)**2 == 5*num*num-4)
    return isprime(n) and isfib(n)

def lucassequencen(n):
    seq = [2,1]
    for i in range(2,n):
        seq.append(seq[-1]+seq[-2])
    return seq[:n]

def isperfectpowern(n):
    for b in range(2,int(math.log2(n))+2):
        a = round(n**(1/b))
        if a**b == n:
            return True
    return False

def collatzlengthn(n):
    steps = 0
    while n > 1:
        n = n//2 if n%2==0 else 3*n+1
        steps += 1
    return steps

def polygonalnumbers(s,n):
    return ((s-2)*n*(n-1))//2 + n
```

```python
def iscarmichaeln(n):
    def isprime(x):
        if x < 2: return False
        for i in range(2,int(x**0.5)+1):
            if x%i == 0:
                return False
        return True
    if isprime(n) or n < 2:
        return False
    for a in range(2,n):
        if math.gcd(a,n) == 1 and pow(a,n-1,n) != 1:
            return False
    return True

def isprimemillerrabinn(n, k=5):
    if n <= 1: return False
    if n == 2: return True
    if n%2 == 0: return False
    r,d = 0, n-1
    while d%2 == 0:
        d //= 2
        r += 1
    for _ in range(k):
        a = random.randrange(2, n-1)
        x = pow(a,d,n)
        if x==1 or x==n-1:
            continue
        for __ in range(r-1):
            x = pow(x,2,n)
            if x == n-1:
                break
        else:
            return False
    return True
```

```python
def pollardrhon(n):
    if n%2 == 0:
        return 2
    x,y,c,d = 2,2,1,1
    while d==1:
        x = (x*x + c) % n
        y = (y*y + c) % n
        y = (y*y + c) % n
        d = math.gcd(abs(x-y), n)
    if d == n:
        return None
    return d

def zetaapproxs_terms(s, terms):
    return sum(1/(n**s) for n in range(1, terms+1))

def partitionfunctionn(n):
    result = [1]
    for i in range(1,n+1):
        val = 0
        j = 1
        while True:
            sign = -1 if j%2 == 0 else 1
            pent1 = i - (j*(3*j-1))//2
            pent2 = i - (j*(3*j+1))//2
            if pent1 < 0:
                break
            val += sign * result[pent1]
            if pent2 >= 0:
                val += sign * result[pent2]
            j += 1
        result.append(val)
    return result[n]
```

```python
# Wrapper function which will call with memory and timing
def run_all():
    test_cases = {
        'factorialn': (factorialn, [10]),
        'ispalindromen': (ispalindromen, [121]),
        'meanofdigitsn': (meanofdigitsn, [123456]),
        'digitalrootn': (digitalrootn, [98765]),
        'isabundantn': (isabundantn, [12]),
        'isdeficientn': (isdeficientn, [15]),
        'isharshadn': (isharshadn, [18]),
        'isautomorphicn': (isautomorphicn, [25]),
        'ispronicn': (ispronicn, [20]),
        'primefactorsn': (primefactorsn, [100]),
        'countdistinctprimefactorsn': (countdistinctprimefactorsn, [100]),
        'isprimepowern': (isprimepowern, [27]),
        'ismersenneprimep': (ismersenneprimep, [3]),
        'twinprimeslimit': (twinprimeslimit, [50]),
        'countdivisorsn': (countdivisorsn, [28]),
        'aliquotsumn': (aliquotsumn, [12]),
        'areamicable': (areamicable, [220, 284]),
        'multiplicativepersistencen': (multiplicativepersistencen, [77]),
        'ishighlycompositen': (ishighlycompositen, [12]),
        'modexp': (modexp, [2, 10, 1000]),
        'modinverse': (modinverse, [3, 11]),
        'crtremaindersmoduli': (crtremaindersmoduli, [[2,3,2], [3,5,7]]),
        'isquadraticresidue': (isquadraticresidue, [5, 11]),
        'ordermod': (ordermod, [2, 7]),
        'isfibonacciprimen': (isfibonacciprimen, [13]),
        'lucassequencen': (lucassequencen, [10]),
        'isperfectpowern': (isperfectpowern, [16]),
        'collatzlengthn': (collatzlengthn, [12]),
        'polygonalnumbers': (polygonalnumbers, [5, 12]),
        'iscarmichaeln': (iscarmichaeln, [561]),
        'isprimemillerrabinn': (isprimemillerrabinn, [101]),
        'pollardrhon': (pollardrhon, [8051]),
        'zetaapproxs_terms': (zetaapproxs_terms, [2, 1000]),
        'partitionfunctionn': (partitionfunctionn, [10]),
```



```
        'isquadraticresidue': (isquadraticresidue, [5, 11]),
        'ordermod': (ordermod, [2, 7]),
        'isfibonacciprimen': (isfibonacciprimen, [13]),
        'lucassequencen': (lucassequencen, [10]),
        'isperfectpowern': (isperfectpowern, [16]),
        'collatzlengthn': (collatzlengthn, [12]),
        'polygonalnumbers': (polygonalnumbers, [5, 12]),
        'iscarmichaeln': (iscarmichaeln, [561]),
        'isprimemillerrabinn': (isprimemillerrabinn, [101]),
        'pollardrhon': (pollardrhon, [8051]),
        'zetaapproxs_terms': (zetaapproxs_terms, [2, 1000]),
        'partitionfunctionn': (partitionfunctionn, [10]),
    }
    results = {}
    for name, (fn, args) in test_cases.items():
        perf = measure_performance(fn, *args)
        results[name] = perf
    return results

# Call to get all results with execution time and memory
all_results_with_metrics = run_all()
```