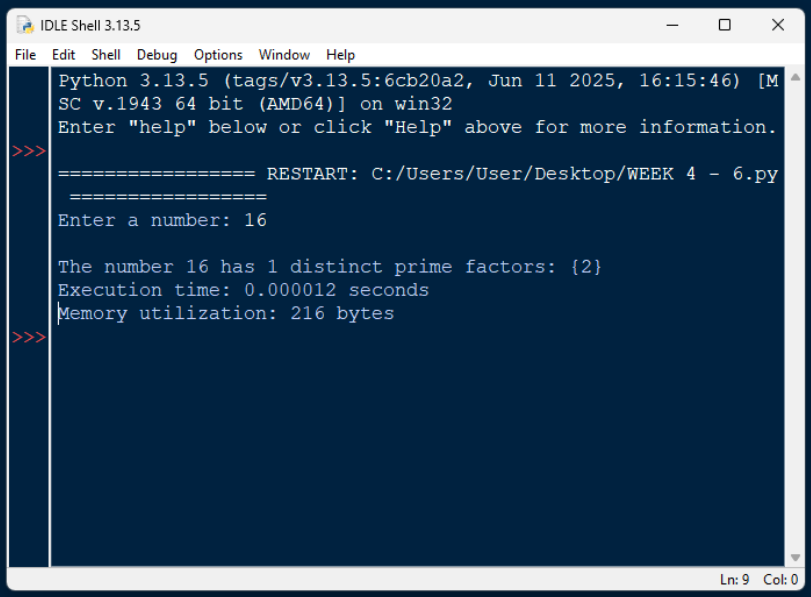
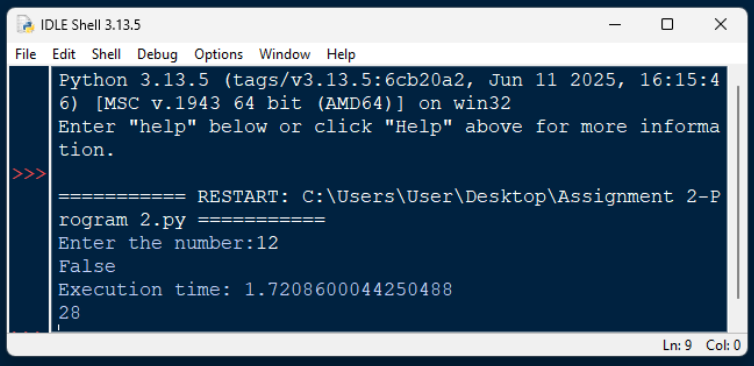
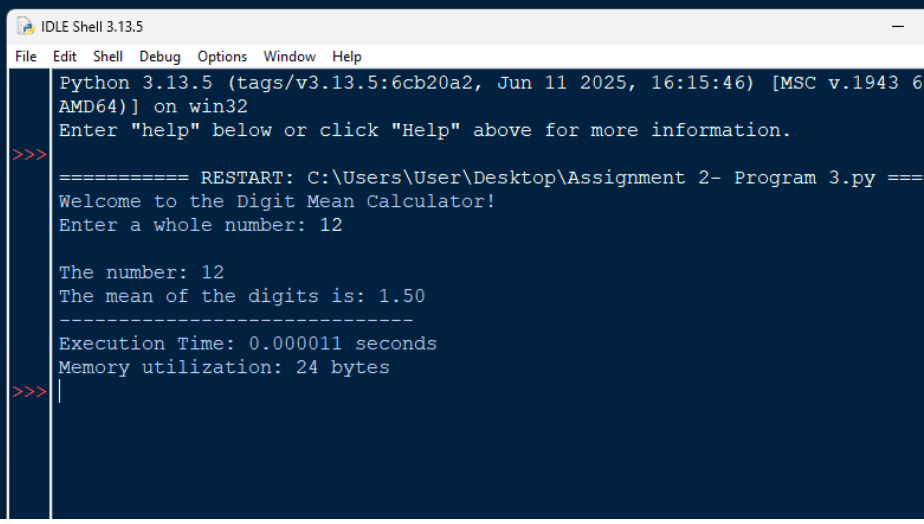
```python
#Write a function factorial(n) that calculates the factorial of
#a non-negative integer n (n!).

import time
import sys
var=1
def factorial(n):

    if n < 0:
        return "Factorial does not exist for negative numbers."
        start_time = time.time()
    else:
        fact = 1
        for i in range(1, n + 1):
            fact = fact*i
        return fact
num = int(input("Enter a number:"))
print(factorial(num))
end_time = time.time()
print("-" * 30)
print(f"Execution Time: {end_time - start_time:.6f} seconds")
print(f"Memory utilization:{sys.getsizeof(11)} bytes")
```



```python
#Write a function is_palindrome(n) that checks if a number
#reads the same forwards and backwards.

import time
import sys

var=1

start_time=time.time()

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

n=int(input("Enter the number:"))
print(is_palindrome(n))

end_time=time.time()

execution_time=end_time-start_time

print("Execution time:",execution_time)
print(sys.getsizeof(var))
```



```python
import time
import sys

def mean_of_digits(n: int) -> float:
    num = abs(n)

    if num == 0:
        return 0.0

    total_sum = 0
    count = 0
    while num > 0:
        digit = num % 10
        total_sum += digit
        count += 1
        num //= 10

    return total_sum / count

print("Welcome to the Digit Mean Calculator!")

try:
    user_input = input("Enter a whole number: ")
    number = int(user_input)

    start_time = time.time()
    result = mean_of_digits(number)
    end_time = time.time()

    print(f"\nThe number: {number}")
    print(f"The mean of the digits is: {result:.2f}")
```



Assignment 2- Program 4.py - C:\Users\User\Desktop\Assignment 2- Program 4.py (3.13.5)

File Edit Format Run Options Window Help

```
import time
import sys

def digital_root_formula(n):

  if n < 0:
    raise ValueError("Digital root is typically defined for non-negative integers.")
  if n == 0:
    return 0
  else:

    return 1 + (n - 1) % 9

# --- Input Handling and Execution ---

try:

  n = int(input("Enter the non-negative integer: "))


  if n < 0:
    print("Error: Please enter a non-negative integer.")
  else:
    start_time_formula = time.time()
    result_formula = digital_root_formula(n)
    end_time_formula = time.time()
    execution_time_formula = end_time_formula - start_time_formula

    print("\n-------------------------")
    print(f"Input Number: {n}")
    print(f"Result (Formula): {result_formula}")
    print(f"Formula Execution time: {execution_time_formula:.9f} seconds")
    print("-------------------------")
```

IDLE Shell 3.13.5

File Edit Shell Debug Options Window Help

```
Python 3.13.5 (tags/v3.13.5:6cb20a2, Jun 11 2025, 16:15:46) [M
SC v.1943 64 bit (AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
========== RESTART: C:\Users\User\Desktop\Assignment 2- Progr
am 4.py ==========
Enter the non-negative integer: 5

-------------------------
Input Number: 5
Result (Formula): 5
Formula Execution time: 0.000005960 seconds
-------------------------
>>>
```

Ln: 12

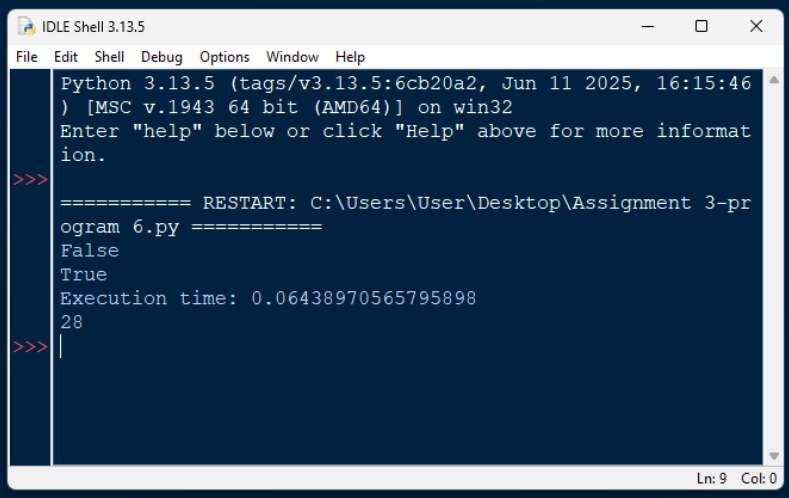
```python
import time
import sys
import math

def is_abundant(n):

  if n <= 0:

    return False
  if n == 1:

    return False

  sum_of_proper_divisors = 1


  limit = int(math.sqrt(n))

  for i in range(2, limit + 1):
    if n % i == 0:
      # i is a divisor.
      sum_of_proper_divisors += i


      if i * i != n:
        sum_of_proper_divisors += n // i


  return sum_of_proper_divisors > n
```

IDLE Shell 3.13.5

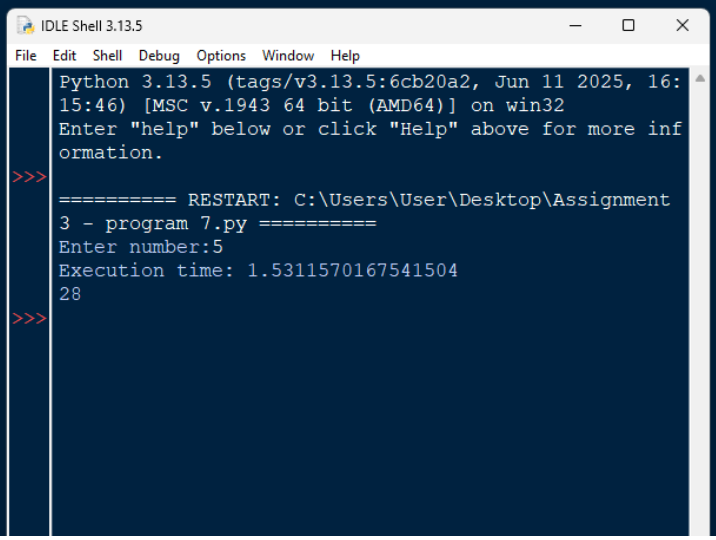
File Edit Shell Debug Options Window Help

```
Python 3.13.5 (tags/v3.13.5:6cb20a2, Jun 11 2025, 16:15:46) [MSC v.1943 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
========== RESTART: C:\Users\User\Desktop\Assignment 2- Program 5.py ==========
Enter a positive integer to check for abundance: 5

------------------------------------
Number checked (n): 5
Is 5 Abundant? **False**
Execution Time (Optimized): 0.000016 seconds
------------------------------------
>>>
```

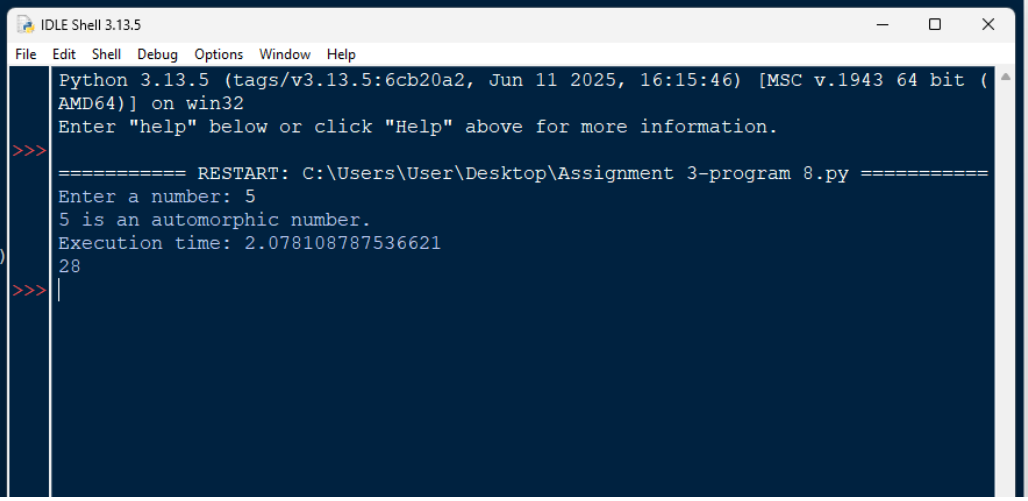
```python
import time
import sys
var=1
start_time=time.time()

def is_deficient(x):
    if x<= 0:
        print(False)
        return

    sum_of_divisors = 0
    for i in range(1,x):
        if x%i == 0:
            sum_of_divisors+= i

    if sum_of_divisors<x:
        print(True)
    else:
        print(False)
is_deficient(6)
is_deficient(4)  # 1,2   1+2=3, where 4>3(true is printed)

end_time=time.time()
execution_time=end_time-start_time
print("Execution time:", execution_time)
print(sys.getsizeof(var))
```



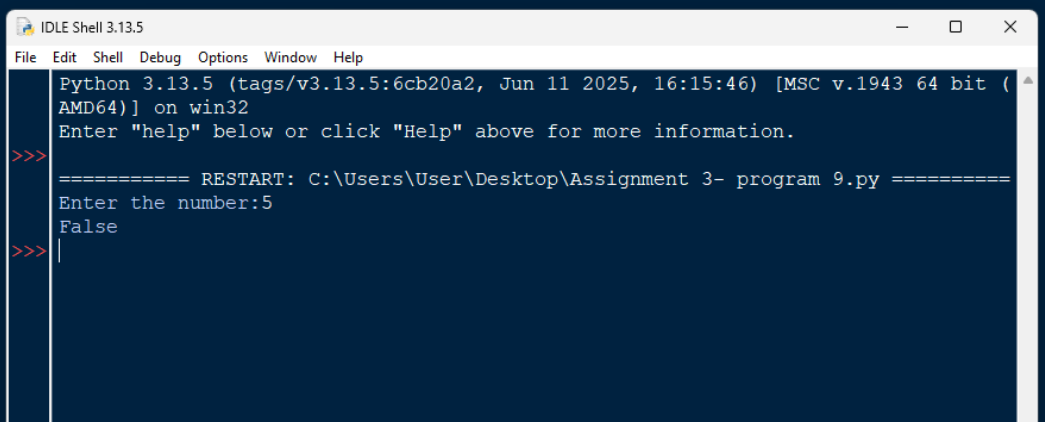
```python
import time
import sys
var=1
start_time=time.time()

def is_harshad(n):
    # Calculate sum of digits
    digit_sum = 0
    temp = n
    while temp > 0:
        digit_sum += temp % 10
        temp //= 10

    # Check divisibility by digit sum
    return n % digit_sum == 0

n=int(input("Enter number:"))

end_time=time.time()
execution_time=end_time-start_time
print("Execution time:", execution_time)
print(sys.getsizeof(var))
```



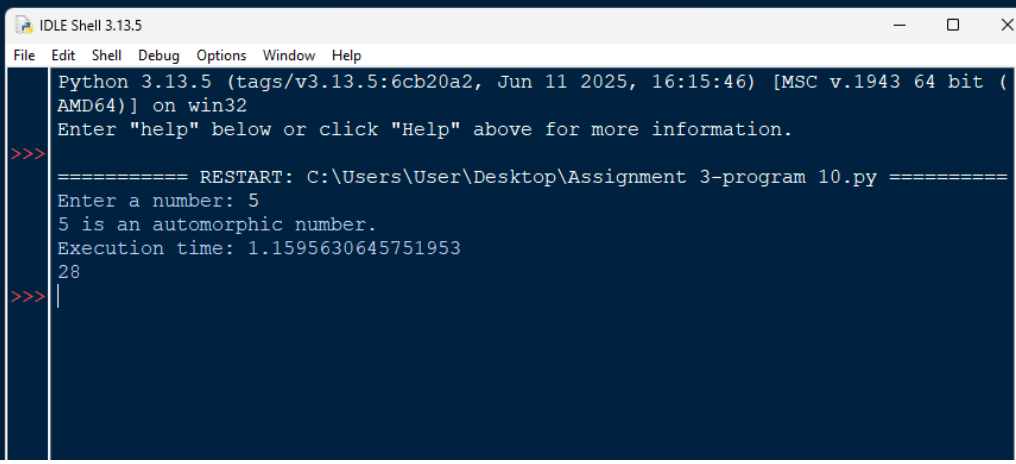
```python
import time
import sys
var=1
start_time=time.time()

def is_automorphic(n):
    square = n * n
    return str(square).endswith(str(n))

num = int(input("Enter a number: "))

if is_automorphic(num):
    print(num, "is an automorphic number.")
else:
    print(num, "is not an automorphic number.")

end_time=time.time()
execution_time=end_time-start_time
print("Execution time:", execution_time)
print(sys.getsizeof(var))
```



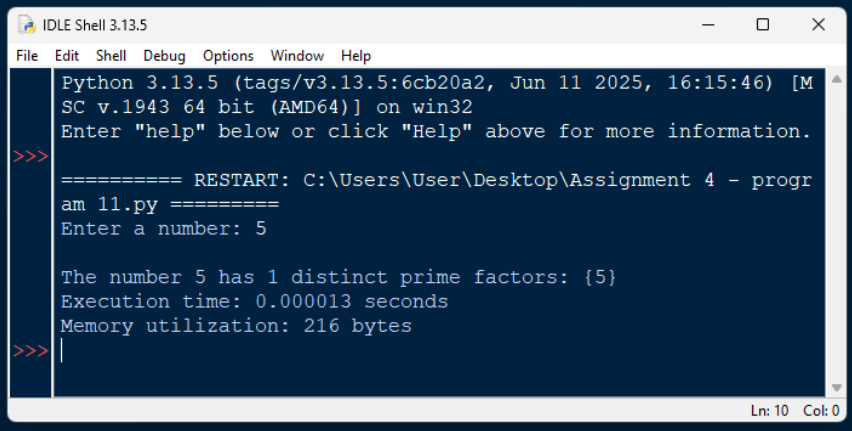
```python
#Write a function in python is_pronic(n) that checks if a number is
#the product of two consecutive integers.

def is_pronic(n):
    if n < 0:
        return False
    for i in range(int(n**0.5) + 1):
        if i * (i + 1) == n:
            return True
    return False
n=int(input("Enter the number:"))
print(is_pronic(n))
```



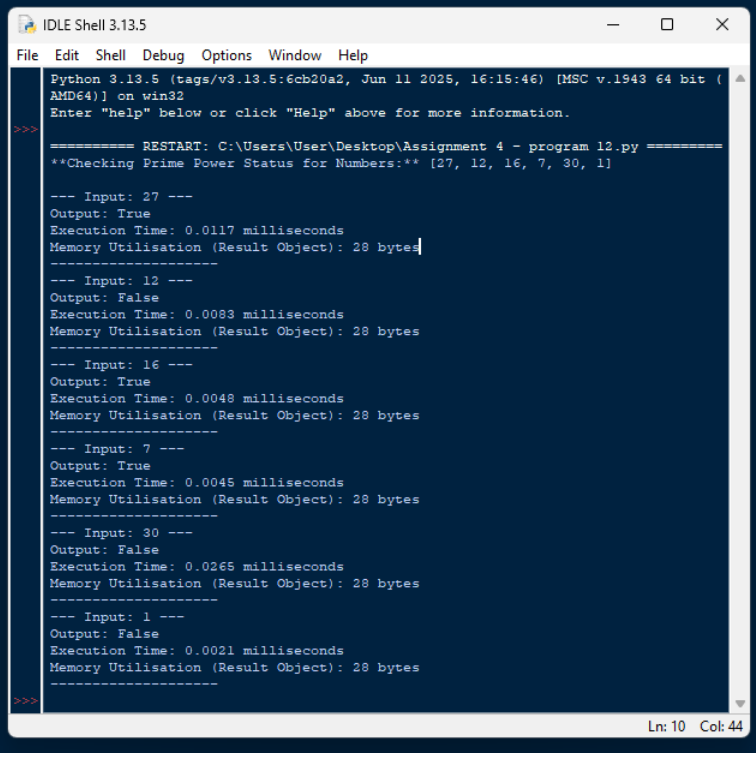
```python
import time
import sys
var=1
start_time=time.time()

def is_automorphic(n):
    square = n * n
    return str(square).endswith(str(n))

num = int(input("Enter a number: "))

if is_automorphic(num):
    print(num, "is an automorphic number.")
else:
    print(num, "is not an automorphic number.")

end_time=time.time()
execution_time=end_time-start_time
print("Execution time:", execution_time)
print(sys.getsizeof(var))
```



```python
import time
import sys
import math

def distinct_prime_factors(n):
    factors = set()
    i = 2
    while i * i <= n:
        while n % i == 0:
            factors.add(i)
            n //= i
        i += 1
    if n > 1:
        factors.add(n)
    return factors

number = int(input("Enter a number: "))

start_time = time.time()
factors = distinct_prime_factors(number)
end_time = time.time()

print(f"\nThe number {number} has {len(factors)} distinct prime factors: {factors}")
print(f"Execution time: {end_time - start_time:.6f} seconds")
print(f"Memory utilization: {sys.getsizeof(factors)} bytes")
```



```python
import time
import sys

def is_prime_power(n):

    if n <= 1:
        return False

    temp_n = n
    prime_factor = -1

    if temp_n % 2 == 0:
        prime_factor = 2
        while temp_n % 2 == 0:
            temp_n //= 2

    i = 3
    while i * i <= temp_n:
        if temp_n % i == 0:

            if prime_factor != -1:
                return False

            prime_factor = i

            while temp_n % i == 0:
                temp_n //= i

        i += 2

    if temp_n > 1:
        if prime_factor != -1 and prime_factor != temp_n:
            return False

        prime_factor = temp_n

    # If a single prime factor was successfully identified, it is a prime power.
    return prime_factor != -1

# --- Execution and Output Section ---

test_numbers = [27, 12, 16, 7, 30, 1]
print(f"**Checking Prime Power Status for Numbers:** {test_numbers}\n")

for n in test_numbers:
    # 1. Start Timer
```



```python
import time
import sys

def is_prime_power_base(n):

    if n <= 1:
        return False
    if n == 2 or n == 3:
        return True
    if n % 2 == 0 or n % 3 == 0:
        return False

    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6
    return True

def is_mersenne_prime(p):

    if not is_prime_power_base(p):
        print(f"Error: Input p={p} must be a prime number.")
        return False

    if p == 2:
        # M_2 = 2^2 - 1 = 3 (Prime)
        return True




    M = 1
    i = 0
    while i < p:
        M = M + M  # Equivalent to M * 2
        i += 1
    M = M - 1 # M = 2^p - 1

    # Initialize the Lucas-Lehmer sequence
    s = 4
```

IDLE Shell 3.13.5

File Edit Shell Debug Options Window Help

```
Python 3.13.5 (tags/v3.13.5:6cb20a2, Jun 11 2025, 16:15:46) [MSC v.1943 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
========== RESTART: C:\Users\User\Desktop\Assignment 4-program 13.py ==========
🔍 **Checking Mersenne Prime Status (2^p - 1) for various prime 'p':**

--- Input p: 3 (M_p = 7) ---
Output (M_p is prime?): **True**
Execution Time: 7.63 microseconds
Memory Utilisation (Result Object): 28 bytes
--------------------------------------------------
--- Input p: 5 (M_p = 31) ---
Output (M_p is prime?): **True**
Execution Time: 9.54 microseconds
Memory Utilisation (Result Object): 28 bytes
--------------------------------------------------
--- Input p: 7 (M_p = 127) ---
Output (M_p is prime?): **True**
Execution Time: 11.21 microseconds
Memory Utilisation (Result Object): 28 bytes
--------------------------------------------------
--- Input p: 11 (M_p = 2047) ---
Output (M_p is prime?): **False**
Execution Time: 94.65 microseconds
Memory Utilisation (Result Object): 28 bytes
--------------------------------------------------
>>>
```

```python
import time
import sys

var=1

start_time=time.time()

def is_prime(n):

    if n <= 1:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False

    i = 3

    while i * i <= n:
        if n % i == 0:
            return False
        i = i + 2
    return True

def twin_primes(limit):

    twin_pairs = []


    p = 3
    while p <= limit - 2:

        if is_prime(p):
            p_plus_2 = p + 2

            if is_prime(p_plus_2):
                twin_pairs.append((p, p_plus_2))
```

IDLE Shell 3.13.5

File Edit Shell Debug Options Window Help

```
Python 3.13.5 (tags/v3.13.5:6cb20a2, Jun 11 2025, 16:15:46) [MSC v.1943 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>

========== RESTART: C:\Users\User\Desktop\Assignment 4 - program 14.py =========
Twin prime pairs up to 100:
[(3, 5), (5, 7), (11, 13), (17, 19), (29, 31), (41, 43), (59, 61), (71, 73)]
Execution time: 0.025059223175048828
28
>>>
```

```python
import time
import tracemalloc

def count_divisors(n):
    count = 0
    i = 1
    while i * i <= n:
        if n % i == 0:
            if i * i == n:
                count += 1
            else:
                count += 2
        i += 1
    return count
num = int(input("Enter a number: "))
tracemalloc.start()
start_time = time.time()
result = count_divisors(num)
end_time = time.time()
current, peak = tracemalloc.get_traced_memory()
tracemalloc.stop()
print(f"Number of divisors of {num}: {result}")
print(f"Execution time: {end_time - start_time:.8f} seconds")
print(f"Current memory usage: {current / 1024:.4f} KB")
print(f"Peak memory usage: {peak / 1024:.4f} KB")
```

IDLE Shell 3.13.5

File Edit Shell Debug Options Window Help

```
Python 3.13.5 (tags/v3.13.5:6cb20a2, Jun 11 2025, 16:15:46) [MSC v.1943 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
========== RESTART: C:\Users\User\Desktop\Assignment 4-program 15.py ==========
Enter a number: 5
Number of divisors of 5: 2
Execution time: 0.00036573 seconds
Current memory usage: 0.7500 KB
Peak memory usage: 0.7500 KB
>>>
```

```python
import time
import sys

var=1

start_time=time.time()

def aliquot_sum(n):
    1 is 0.
    if n <= 0:
        return 0

    total_sum = 0


    i = 1

    while i < n:
        if n % i == 0:
            total_sum = total_sum + i

        i = i + 1

    return total_sum

n=int(input("Enter the number:"))
print(aliquot_sum(n))
end_time=time.time()
execution_time=end_time-start_time
print("Execution time:", execution_time)
print(sys.getsizeof(var))
```

IDLE Shell 3.13.5

File Edit Shell Debug Options Window Help

```
Python 3.13.5 (tags/v3.13.5:6cb20a2, Jun 11 2025, 16:15:46) [MSC v.1943 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.

>>>
Warning (from warnings module):
  File "C:\Users\User\Desktop\Assignment 5 - program 1.py", line 9
    1 is 0.
SyntaxWarning: "is" with 'int' literal. Did you mean "=="?
>>>
>>>
========== RESTART: C:\Users\User\Desktop\Assignment 5 - program 1.py ==========
Enter the number:5
1
Execution time: 0.7570810317993164
28
>>>
```

```python
import time
import sys

var=1

start_time=time.time()
def aliquot_sum(n):

    if n <= 0:
        return 0

    total_sum = 0
    i = 1 # Start checking for divisors from 1

    while i < n:
        if n % i == 0:
            total_sum = total_sum + i

        i = i + 1

    return total_sum

def are_amicable(a, b):

    if a == b:
        return False

    if a <= 0 or b <= 0:
        return False

    sum_a = aliquot_sum(a)

    sum_b = aliquot_sum(b)

    is_amicable = (sum_a == b) and (sum_b == a)
```

IDLE Shell 3.13.5

File Edit Shell Debug Options Window Help

```
Python 3.13.5 (tags/v3.13.5:6cb20a2, Jun 11 2025, 16:15:46) [MSC v.1943 64 bit
AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
========== RESTART: C:\Users\User\Desktop\Assignment 5 - program 2.py ========
Enter the number:5
Enter the first number:4
Enter the other number:8
False
Execution time: 1.4737658500671387
28
>>>
```

```python
import time
import sys

var=1
start_time=time.time()
def multiplicative_persistence(n):

    if n < 10:
        return 0

    persistence_count = 0

    while n >= 10:
        persistence_count = persistence_count + 1

        next_n = 1

        current_num = n

        while current_num > 0:
            digit = current_num % 10


            next_n = next_n * digit

            current_num = current_num // 10


        n = next_n

    return persistence_count


n=int(input("Enter the number:"))
print(multiplicative_persistence(n))
end_time=time.time()
```

IDLE Shell 3.13.5

File Edit Shell Debug Options Window Help

```
Python 3.13.5 (tags/v3.13.5:6cb20a2, Jun 11 2025, 16:15:46) [MSC v.1943 64 bit
AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
========== RESTART: C:\Users\User\Desktop\Assignment 5 - program 3.py ==========
Enter the number:5
0
Execution time: 1.0727691650390625
28
>>>
```

```python
import time
import sys

var=1

start_time=time.time()
def count_divisors(n):

    if n <= 0:
        return 0

    if n == 1:
        return 1

    divisor_count = 0
    i = 1 # Start checking divisors from 1

    while i <= n:
        if n % i == 0:
            divisor_count = divisor_count + 1

        i = i + 1

    return divisor_count

def is_highly_composite(n):

    if n <= 0:
        return False

    if n == 1:
        return True

    n_divisors = count_divisors(n)

    # Start checking all numbers 'i' smaller than 'n
```

IDLE Shell 3.13.5

File Edit Shell Debug Options Window Help

```
Python 3.13.5 (tags/v3.13.5:6cb20a2, Jun 11 2025, 16:15:46) [MSC v.1943 64
AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
========== RESTART: C:\Users\User\Desktop\Assignment 5 - program 4.py =====
Enter the number:5
False
Execution time: 1.0870742797851562
28
>>>
```

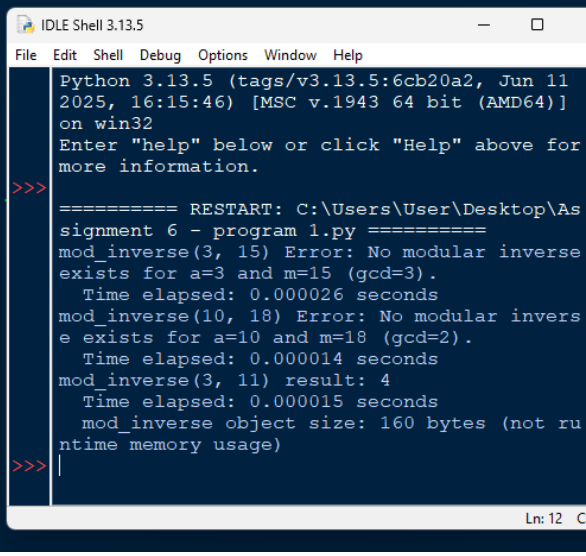
```python
import time
import sys

var=1

start_time=time.time()

def mod_exp(base, exponent, modulus):


    if modulus == 1:
        return 0

    res = 1

    base = base % modulus

    current_exponent = exponent

    while current_exponent > 0:

        if current_exponent % 2 == 1:
            res = (res * base) % modulus

        base = (base * base) % modulus

        current_exponent = current_exponent // 2

    return res

base=int(input("Enter the base:"))
exponent=int(input("Enter the exponent:"))
modulus=int(input("Enter the modulus:"))
print(mod_exp(base, exponent, modulus))
end_time=time.time()
execution_time=end_time-start_time
```

IDLE Shell 3.13.5

File Edit Shell Debug Options Window Help

```
Python 3.13.5 (tags/v3.13.5:6cb20a2, Jun 11 2025, 16:15:46) [MSC v.1943 64
AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
========== RESTART: C:\Users\User\Desktop\Assignment 5 - program 5.py ====
Enter the base:5
Enter the exponent:2
Enter the modulus:10
5
Execution time: 5.930783748626709
28
>>>
```

```python
import time
import sys
# import tracemalloc # for more detailed memory snapshots

def mod_inverse(a: int, m: int) -> int:
    if not isinstance(a, int) or not isinstance(m, int):
        raise TypeError("Both a and m must be integers.")
    if m <= 1:
        raise ValueError("Modulus m must be greater than 1.")

    def extended_gcd(x, y):
        if y == 0:
            return x, 1, 0
        gcd, x1, y1 = extended_gcd(y, x % y)
        return gcd, y1, x1 - (x // y) * y1

    gcd, x, _ = extended_gcd(a, m)

    if gcd != 1:
        raise ValueError(f"No modular inverse exists for a={a} and m={m} (gcd={gcd})")

    return x % m

def run_test(a, m):
    start_time = time.perf_counter()



    try:
        function_size_bytes = sys.getsizeof(mod_inverse)

        result = mod_inverse(a, m)
        end_time = time.perf_counter()

        print(f"mod_inverse({a}, {m}) result: {result}")
        print(f"  Time elapsed: {end_time - start_time:.6f} seconds")
```



```python
import time
import sys


def extended_gcd(a, b):

    if a == 0:
        return (b, 0, 1)

    g, x1, y1 = extended_gcd(b % a, a)

    x = y1 - (b // a) * x1
    y = x1

    return (g, x, y)

def mod_inverse(a, m):

    g, x, y = extended_gcd(a, m)

    if g != 1:
        raise ValueError(f"Modular inverse does not exist (gcd({a}, {m}) = {g} != 1). Moduli must be pairwise coprime for this simple

    return x % m

# --- Main CRT Solver ---

def crt(remainders, moduli):

    if len(remainders) != len(moduli):
        raise ValueError("The number of remainders must match the number of moduli.")

    # 1. Calculate N, the product of all moduli (M in the formula)
    N = 1
    for m in moduli:
        N *= m
```

```
File  Edit  Shell  Debug  Options  Window  Help
>>> ========== RESTART: C:\Users\User\Desktop\Assignment 6 - program 2.py ==========
--- Chinese Remainder Theorem Solver ---
System of congruences: x ≡ 2 (mod 3), x ≡ 3 (mod 5), x ≡ 2 (mod 7)

[Result]
The unique solution modulo N=105 is: x = 23

[Verification]
  23 ≡ 2 (mod 3) - OK
  23 ≡ 3 (mod 5) - OK
  23 ≡ 2 (mod 7) - OK
The solution is verified to be correct.

[Performance Metrics]
Time taken: 0.022100 milliseconds
Memory allocated for solution (approx.): 28 bytes
>>>
                                                                Ln: 20  Col: 0
```

```python
import time
import sys

def is_quadratic_residue(a, p):

    if not isinstance(p, int) or p <= 0:
        raise ValueError("Modulus 'p' must be a positive integer (ideally a prime for this context).")

    if a % p == 0:
        return True


    if p == 2:
        return True # Since a % p != 0 case is handle

    exponent = (p - 1) // 2

    legendre_symbol_value = pow(a, exponent, p)


    if legendre_symbol_value == 1:
        return True

    elif legendre_symbol_value == p - 1:
        return False


    else:
                return False



if __name__ == "__main__":
    p = 7
```

IDLE Shell 3.13.5

File Edit Shell Debug Options Window Help

```
Python 3.13.5 (tags/v3.13.5:6cb20a2, Jun 11 2025, 16:15:46) [MSC v.194
3 64 bit (AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
========== RESTART: C:\Users\User\Desktop\Assignment 6 - program 3.py
==========
--- Quadratic Residue Checker (using Euler's Criterion) ---
Checking residues modulo P = 7

[Test Results]
  x^2 ≡ 1 (mod 7) is Residue (Time: 0.0074 ms)
  x^2 ≡ 2 (mod 7) is Residue (Time: 0.0080 ms)
  x^2 ≡ 3 (mod 7) is Non-Residue (Time: 0.0074 ms)
  x^2 ≡ 4 (mod 7) is Residue (Time: 0.0070 ms)
  x^2 ≡ 5 (mod 7) is Non-Residue (Time: 0.0082 ms)
  x^2 ≡ 6 (mod 7) is Non-Residue (Time: 0.0063 ms)
  x^2 ≡ 0 (mod 7) is Residue (Time: 0.0030 ms)

[Performance Metrics Summary]
Total test cases run: 7
Average time per check: 0.006757 milliseconds
Approximate memory for result object (per case): 28 bytes (for the boo
lean result)
```

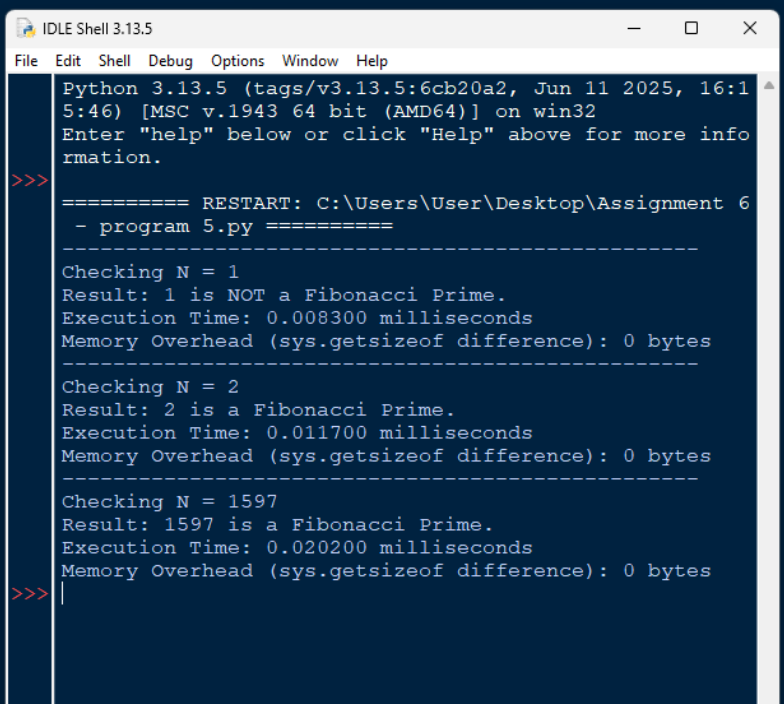
```python
import time
import tracemalloc

def gcd(x, y):
    while y:
        x, y = y, x % y
    return x

def order_mod(a, n):
    if gcd(a, n) != 1:
        return None

    tracemalloc.start()
    start = time.time()

    k = 1
    value = a % n

    while value != 1:
        value = (value * a) % n
        k += 1
    current, peak = tracemalloc.get_traced_memory()
    tracemalloc.stop()
    exec_time = time.time() - start
    mem_used = peak / 1024

    print(f"Execution Time: {exec_time:.6f} seconds")
    print(f"Memory Used: {mem_used:.2f} KB")

    return k

a = int(input("Enter a: "))
n = int(input("Enter n: "))

result = order_mod(a, n)
```

IDLE Shell 3.13.5

File Edit Shell Debug Options Window Help

```
Python 3.13.5 (tags/v3.13.5:6cb20a2, Jun 11 2025, 16:15:46) [MSC v
.1943 64 bit (AMD64)] on win32
Enter "help" below or click "Help" above for more information.

>>>

========== RESTART: C:\Users\User\Desktop\Assignment 6 - program 4
.py ==========
Enter a: 5
Enter n: 9
Execution Time: 0.001730 seconds
Memory Used: 0.00 KB
Order of 5 mod 9 is 6.
>>>
```

```python
import time
import sys

# --- Core Logic Functions (No built-in functions/modules used) ---

def is_prime(n):

    if n <= 1:
        return False
    if n <= 3:
        return True

    if n % 2 == 0 or n % 3 == 0:
        return False

    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6

    return True

def is_fibonacci(n):

    if n < 0:
        return False

    a, b = 0, 1

    if n == 0 or n == 1:
        return True

    while b < n:
        a, b = b, a + b
```



```python
import time
import sys

def lucas_sequence(n):

    if not isinstance(n, int):
        raise TypeError("n must be an integer.")
    if n < 0:
        raise ValueError("n must be a non-negative integer.")

    # Base cases
    if n == 0:
        return []
    elif n == 1:
        return [2]
    elif n == 2:
        return [2, 1]

    lucas_nums = [2, 1]

    for _ in range(2, n):
        lucas_nums.append(lucas_nums[-1] + lucas_nums[-2])

    return lucas_nums

if __name__ == "__main__":
    terms = 15

    print(f"--- Lucas Sequence Generation (First {terms} terms) ---")

    try:
        start_time = time.perf_counter()

        sequence = lucas_sequence(terms)
```

IDLE Shell 3.13.5

File Edit Shell Debug Options Window Help

```
Python 3.13.5 (tags/v3.13.5:6cb20a2, Jun 11 2025, 16:15:46)
[MSC v.1943 64 bit (AMD64)] on win32
Enter "help" below or click "Help" above for more informati
on.
>>>
========== RESTART: C:\Users\User\Desktop\Assignment 7 - pr
ogram 1.py ==========
--- Lucas Sequence Generation (First 15 terms) ---

[Result]
Lucas numbers: [2, 1, 3, 4, 7, 11, 18, 29, 47, 76, 123, 199
, 322, 521, 843]

[Performance Metrics]
Terms calculated: 15
Time taken: 0.018800 milliseconds
Memory allocated for list object (approx.): 184 bytes
>>>
```

```python
import time
import sys

def integer_power(base, exponent):

    if exponent == 0:
        return 1

    result = 1
    for _ in range(exponent):
        result *= base
    return result

def integer_isqrt(n):

    if n < 0:
        raise ValueError("Cannot compute square root of negative number.")
    if n == 0 or n == 1:
        return n

    low = 1

    high = n // 2 + 1

    result = 1

    while low <= high:
        mid = (low + high) // 2
        mid_sq = mid * mid

        if mid_sq == n:
            return mid
        elif mid_sq < n:
            result = mid # Store potential answer
            low = mid + 1
        else:
```

IDLE Shell 3.13.5

File Edit Shell Debug Options Window Help

```
Python 3.13.5 (tags/v3.13.5:6cb20a2, Jun 11 2025, 16:15:
) [MSC v.1943 64 bit (AMD64)] on win32
Enter "help" below or click "Help" above for more inform
ion.
>>>
========== RESTART: C:\Users\User\Desktop\Assignment 7 -
rogram 2.py ==========
--- Perfect Power Checker (No built-in math functions) -

[Test Results]
  4: Perfect Power (Time: 0.013400 ms)
  1: Perfect Power (Time: 0.005600 ms)
  1000000: Perfect Power (Time: 0.020100 ms)
  1000001: Not a Perfect Power (Time: 0.216500 ms)

[Performance Metrics Summary]
Total numbers tested: 4
Average time per check: 0.063900 milliseconds
Approximate memory for result object (per case): 28 byte
(for the boolean result)
>>>
```

```python
import time
import sys

def collatz_length(n):

    if n <= 0:

        raise ValueError("Input must be a positive integer.")

    current = n
    steps = 0

    while current != 1:

        if current % 2 == 0:
            current = current // 2  # Integer division is used
        else:
            current = 3 * current + 1

        steps += 1

        if steps > 100000:

            print(f"Warning: Reached max steps (100000) for starting number
            return -1 # Return -1 to indicate failure or max limit reached

    return steps


if __name__ == "__main__":

    test_numbers = [6, 100, 27]

    print("--- Collatz Sequence Length Calculation ---")

    for number in test_numbers:
```

IDLE Shell 3.13.5

File Edit Shell Debug Options Window Help

```
Python 3.13.5 (tags/v3.13.5:6cb20a2, Jun 11 2025, 16
:15:46) [MSC v.1943 64 bit (AMD64)] on win32
Enter "help" below or click "Help" above for more in
formation.
>>>
========== RESTART: C:\Users\User\Desktop\Assignment
7 - program 3.py ==========
--- Collatz Sequence Length Calculation ---
-------------------------------------------------
Checking N = 6
Steps required to reach 1: 8
Execution Time: 0.016400 milliseconds
Memory Overhead (sys.getsizeof difference): 0 bytes
-------------------------------------------------
Checking N = 100
Steps required to reach 1: 25
Execution Time: 0.022100 milliseconds
Memory Overhead (sys.getsizeof difference): 0 bytes
-------------------------------------------------
Checking N = 27
Steps required to reach 1: 111
Execution Time: 0.021900 milliseconds
Memory Overhead (sys.getsizeof difference): 0 bytes
-------------------------------------------------
>>>
```

Ln: 9 Col:

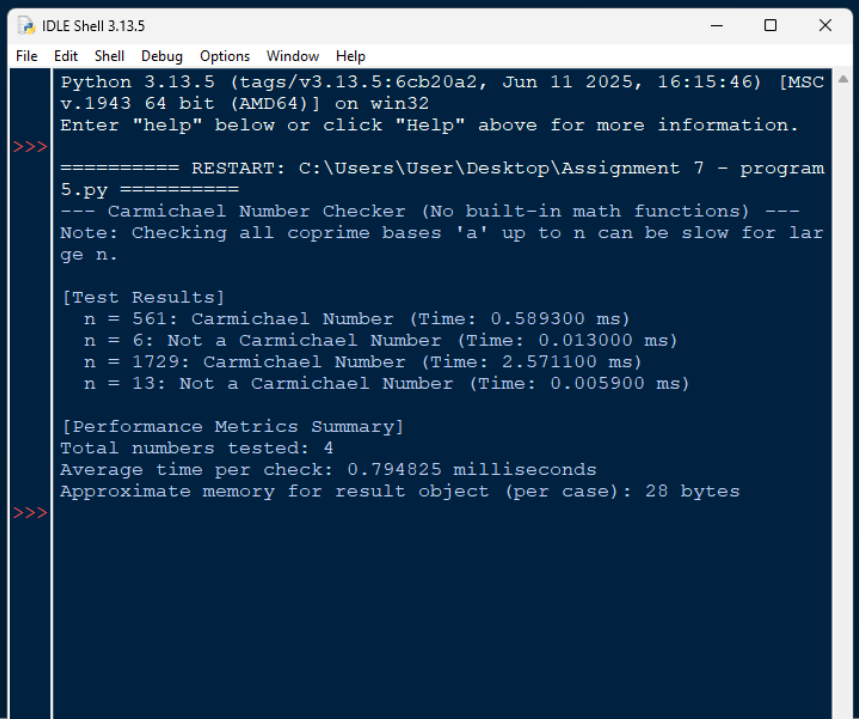
```python
import time
import tracemalloc

def polygonal_number(s, n):
    return ((s - 2) * n * n - (s - 4) * n) // 2
s = int(input("Enter s (number of sides): "))
n = int(input("Enter n (term number): "))
tracemalloc.start()
start = time.time()
result = polygonal_number(s, n)
current, peak = tracemalloc.get_traced_memory()
tracemalloc.stop()
exec_time = time.time() - start

print(f"\n{n}-th {s}-gonal number = {result}")
print(f"Execution Time: {exec_time:.6f} seconds")
print(f"Memory Used: {peak/1024:.2f} KB")
```

IDLE Shell 3.13.5

File Edit Shell Debug Options Window Help

```
Python 3.13.5 (tags/v3.13.5:6cb20a2, Jun 11 2025, 16:15:46) [MSC v.1943 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
========== RESTART: C:\Users\User\Desktop\Assignment 7 - program 4.py ==========
Enter s (number of sides): 5
Enter n (term number): 6

6-th 5-gonal number = 51
Execution Time: 0.000033 seconds
Memory Used: 0.75 KB
>>>
```

```python
import time
import sys

def gcd(a, b):

    while b:
        a, b = b, a % b
    return a

def modular_exponentiation(base, exponent, modulus):

    base %= modulus
    result = 1

    while exponent > 0:
        if exponent % 2 == 1:
            result = (result * base) % modulus

        base = (base * base) % modulus

        exponent //= 2

    return result

def is_composite(n):

    if n <= 3:
        return False
    if n % 2 == 0 or n % 3 == 0:
        return True

    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return True
        i += 6
```



```python
import time
import sys

def modular_exponentiation(base, exponent, modulus):

    if modulus == 1:
        return 0
    base %= modulus
    result = 1

    while exponent > 0:
        if exponent % 2 == 1:
            result = (result * base) % modulus

        base = (base * base) % modulus

        exponent //= 2

    return result

DETERMINISTIC_BASES = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37]

def is_prime_miller_rabin(n: int, k: int = 5) -> bool:

    if not isinstance(n, int):
        raise TypeError("n must be an integer.")

    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0:
        return False

    r, d = 0, n - 1
    while d % 2 == 0:
```

IDLE Shell 3.13.5

File Edit Shell Debug Options Window Help

```
Python 3.13.5 (tags/v3.13.5:6cb20a2, Jun 11 2025, 16:15:46) [MSC
v.1943 64 bit (AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
========== RESTART: C:\Users\User\Desktop\Assignment 8 - program
1.py ==========
--- Miller-Rabin Primality Checker (Custom Implementation) ---
Testing with k=10 deterministic bases: [2, 3, 5, 7, 11, 13, 17,
19, 23, 29]

[Test Results]
  2: Likely Prime (Time: 0.004100 ms)
  7: Likely Prime (Time: 0.014800 ms)
  999983: Likely Prime (Time: 0.045600 ms)
  1000000007: Likely Prime (Time: 0.064500 ms)

[Performance Metrics Summary]
Total numbers tested: 4
Average time per check: 0.032250 milliseconds
Approximate memory for result object (per case): 28 bytes
>>>
```

```python
import time
import sys

def gcd(a, b):

    a = abs(a)
    b = abs(b)

    if a < 0: a = -a
    if b < 0: b = -b

    while b:
        a, b = b, a % b
    return a

def is_prime_simple(n):

    if n <= 1: return False
    if n <= 3: return True
    if n % 2 == 0 or n % 3 == 0: return False

    i = 5
    while i * i <= n and i < 20:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6


    return True

def pollard_rho(n):

    if n <= 1: return None
    if n % 2 == 0: return 2
    if is_prime_simple(n): return n # Skip for probable primes
```

IDLE Shell 3.13.5

File Edit Shell Debug Options Window Help

```
Python 3.13.5 (tags/v3.13.5:6cb20a2, Jun 11 2025, 16:15:46) [MSC
 v.1943 64 bit (AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
========== RESTART: C:\Users\User\Desktop\Assignment 8 - program
 2.py ==========
--- Pollard's Rho Factorization ---
--------------------------------------------------
Factoring N = 91
Result: Factor found: 7. Other factor: 13.
Execution Time: 0.021500 milliseconds
Memory Overhead (sys.getsizeof difference): 0 bytes
--------------------------------------------------
Factoring N = 10403
Result: No non-trivial factor found (N is likely prime: 10403).
Execution Time: 0.012000 milliseconds
Memory Overhead (sys.getsizeof difference): 0 bytes
--------------------------------------------------
Factoring N = 863
Result: No non-trivial factor found (N is likely prime: 863).
Execution Time: 0.008400 milliseconds
Memory Overhead (sys.getsizeof difference): 0 bytes
--------------------------------------------------
Factoring N = 29017
Result: No non-trivial factor found (N is likely prime: 29017).
Execution Time: 0.014300 milliseconds
Memory Overhead (sys.getsizeof difference): 0 bytes
--------------------------------------------------
>>>
```

```python
import time
import sys


def integer_power(base, exponent):

    if exponent == 0:
        return 1.0

    result = 1.0
    float_base = float(base)

    for _ in range(exponent):
        result *= float_base
    return result

def zeta_approx(s, terms):

    if not isinstance(s, int) or not isinstance(terms, int):
        raise TypeError("s and terms must be integers for this constrained implementation.")
    if s <= 1:
        raise ValueError("s must be greater than 1 for series convergence.")
    if terms <= 0:
        return 0.0

    zeta_sum = 0.0
    for n in range(1, terms + 1):
        n_to_s = integer_power(n, s)

        term = 1.0 / n_to_s

        zeta_sum += term

    return zeta_sum
```

IDLE Shell 3.13.5

File Edit Shell Debug Options Window Help

```
Python 3.13.5 (tags/v3.13.5:6cb20a2, Jun 11 2025, 16:15:46) [MSC v.1943 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
========== RESTART: C:\Users\User\Desktop\Assignment 8 - program 3.py ==========
--- Riemann Zeta Approximation ---

[Result: ζ(2) Approximation]
Terms summed (n): 10000
Approximated value: 1.6448340718

[Performance Metrics]
Time taken: 3.469700 milliseconds
Memory allocated for result (approx.): 24 bytes
```

Ln: 14 Col: 0

```python
import time
import sys

def partition_function(n):

    if not isinstance(n, int) or n < 0:
        raise ValueError("n must be a non-negative integer.")

    if n == 0:
        return 1

    dp = [0] * (n + 1)
    dp[0] = 1 # p(0) = 1 (empty partition)

    g_k_list = []
    k = 1
    while True:
        g_k = k * (3 * k - 1) // 2
        g_minus_k = k * (3 * k + 1) // 2

        if g_k <= n:
            g_k_list.append(g_k)
        if g_minus_k <= n:
            g_k_list.append(g_minus_k)

        if g_k > n and g_minus_k > n:
            break

        k += 1

    for i in range(1, n + 1):
        sign_index = 0 # Used to track the sign (+1, +1, -1, -1, +1, +1, ...)
        current_sum = 0

        for g_k in g_k_list:
            if i - g_k < 0:
```