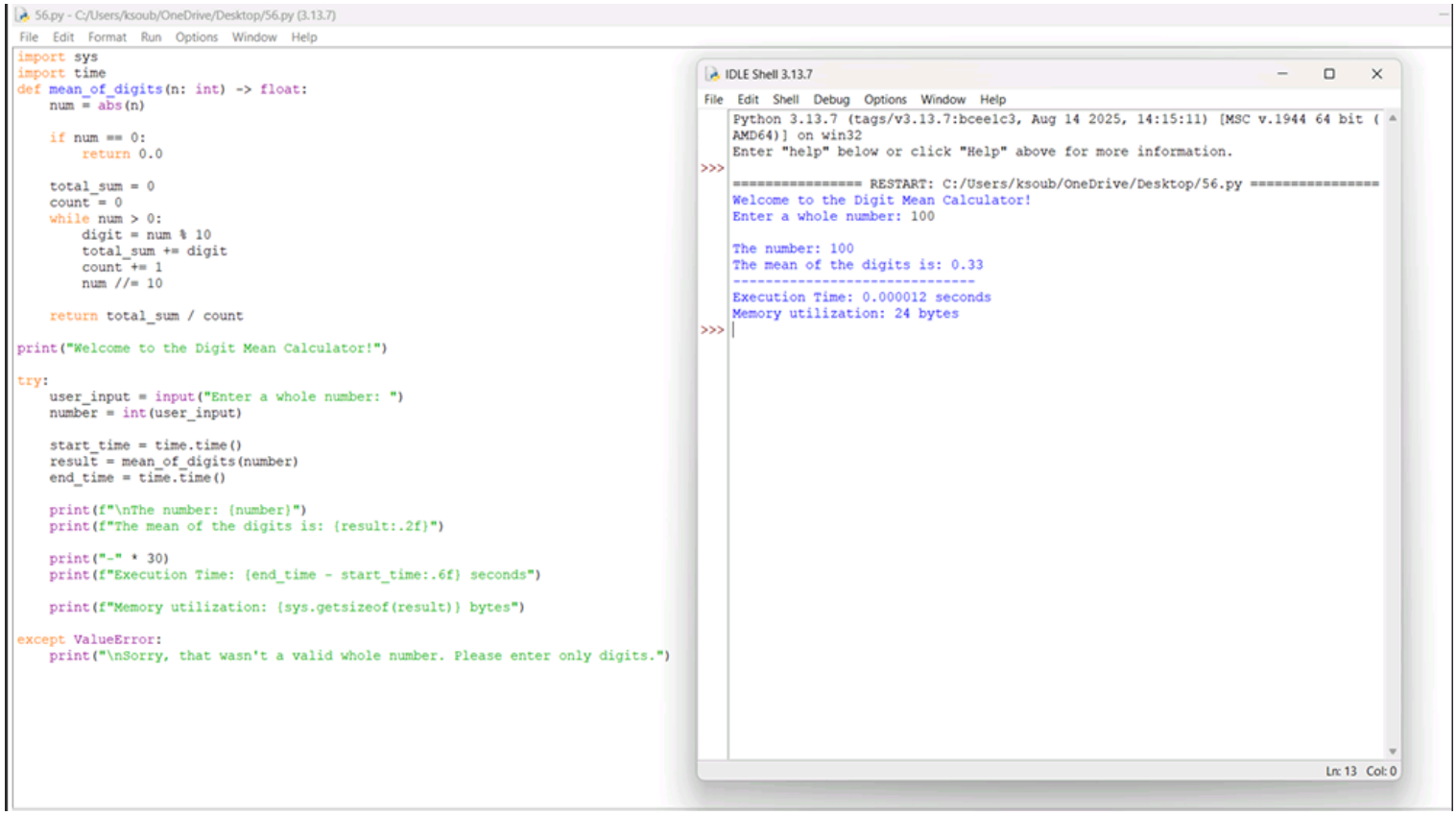
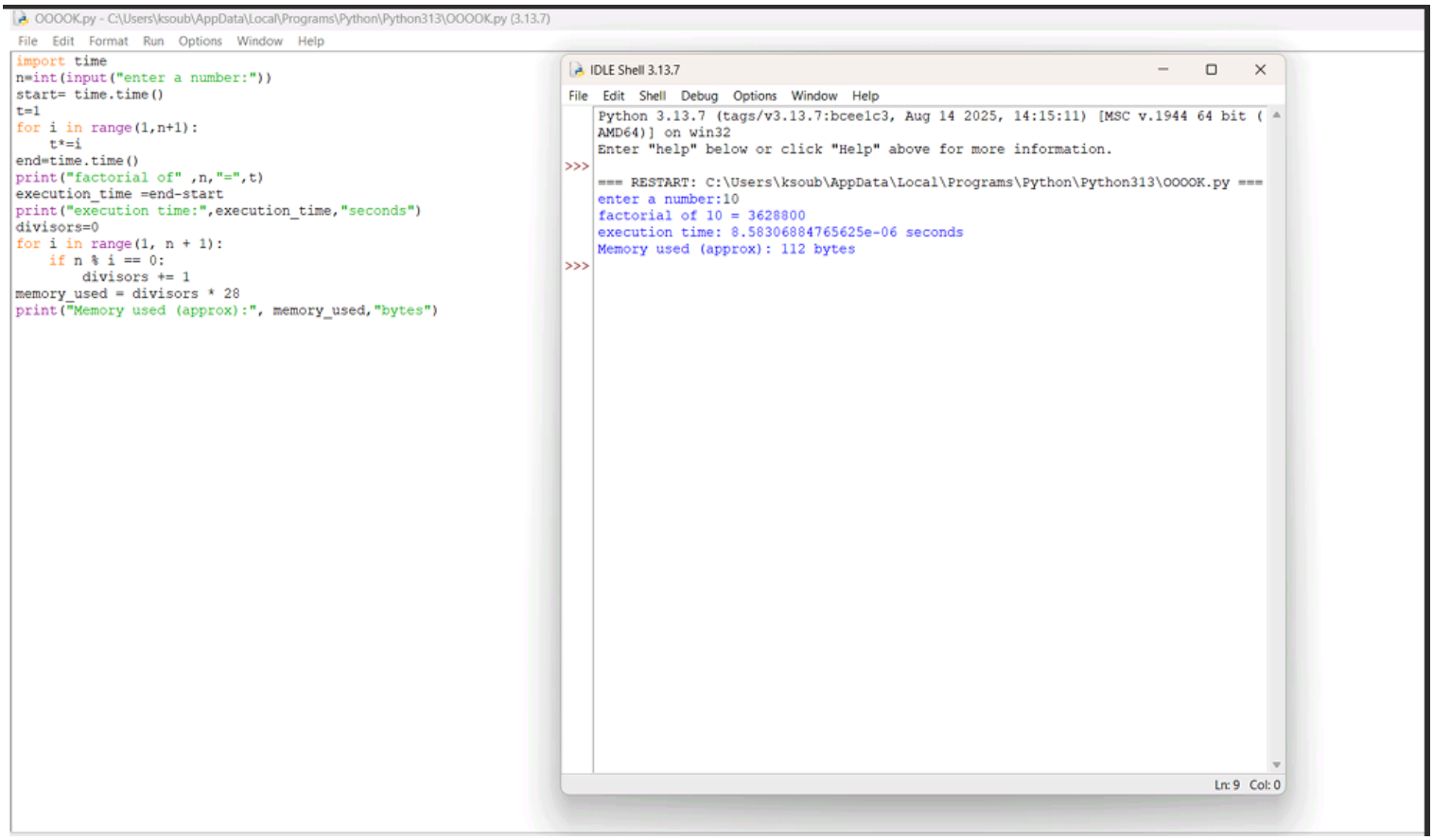
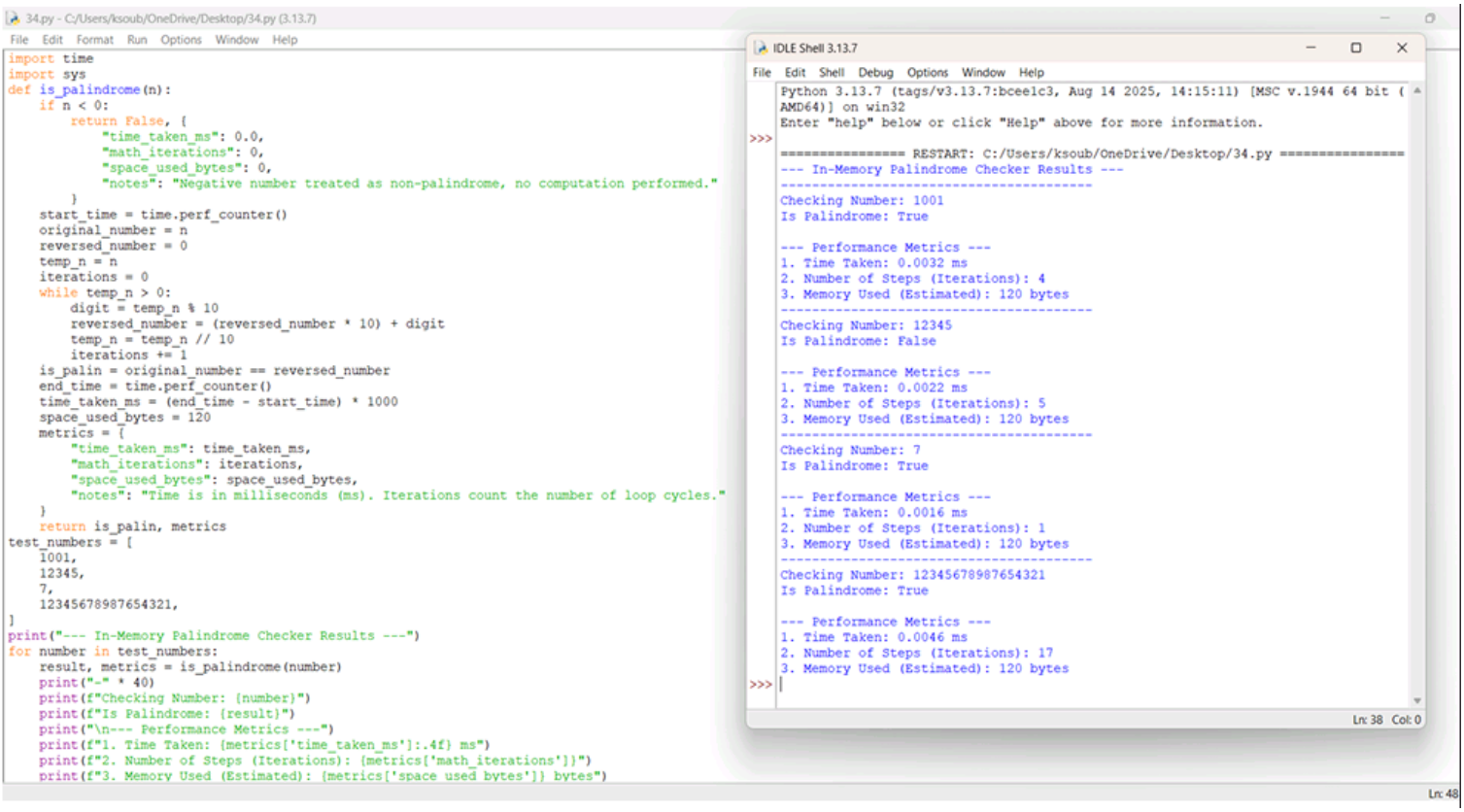
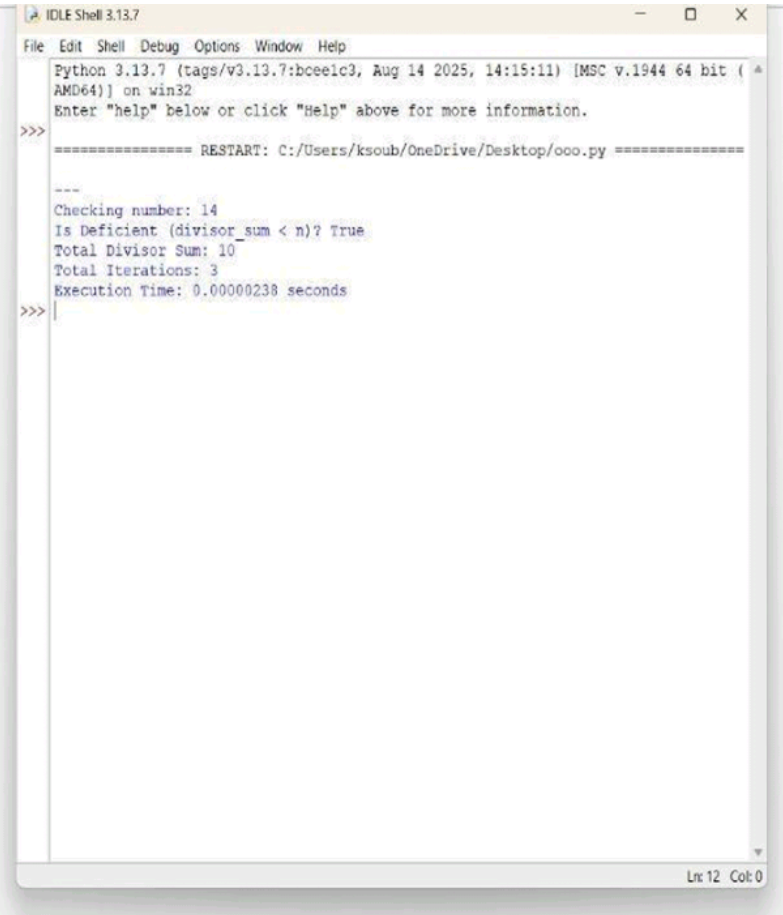

56.py - C:/Users/ksoub/OneDrive/Desktop/56.py (3.13.7)
File Edit Format Run Options Window Help
import sys
import time
def mean_of_digits(n: int) -> float:
num = abs(n)
if num == 0:
return 0.0
total_sum = 0
count = 0
while num > 0:
digit = num % 10
total_sum += digit
count += 1
num //= 10
return total_sum / 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}")
print("-" * 30)
print(f"Execution Time: {end_time - start_time:.6f} seconds")
print(f"Memory utilization: {sys.getsizeof(result)} bytes")
except ValueError:
print("\nSorry, that wasn't a valid whole number. Please enter only digits.")
IDLE Shell 3.13.7
File Edit Shell Debug Options Window Help
Python 3.13.7 (tags/v3.13.7:bcee1c3, Aug 14 2025, 14:15:11) [MSC v.1944 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
========== RESTART: C:/Users/ksoub/OneDrive/Desktop/56.py ==========
Welcome to the Digit Mean Calculator!
Enter a whole number: 100
The number: 100
The mean of the digits is: 0.33

Execution Time: 0.000012 seconds
Memory utilization: 24 bytes
>>>
Ln: 13 Col: 0


OOOOK.py - C:\Users\ksoub\AppData\Local\Programs\Python\Python313\OOOOK.py (3.13.7)
File Edit Format Run Options Window Help
import time
n=int(input("enter a number:"))
start= time.time()
t=1
for i in range(1,n+1):
t*=i
end=time.time()
print("factorial of" ,n,"=",t)
execution_time =end-start
print("execution time:",execution_time,"seconds")
divisors=0
for i in range(1, n + 1):
if n % i == 0:
divisors += 1
memory_used = divisors * 28
print("Memory used (approx):", memory_used,"bytes")
IDLE Shell 3.13.7
File Edit Shell Debug Options Window Help
Python 3.13.7 (tags/v3.13.7:bcee1c3, Aug 14 2025, 14:15:11) [MSC v.1944 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
=== RESTART: C:\Users\ksoub\AppData\Local\Programs\Python\Python313\OOOOK.py ===
enter a number:10
factorial of 10 = 3628800
execution time: 8.58306884765625e-06 seconds
Memory used (approx): 112 bytes
>>>
Ln: 9 Col: 0


34.py - C:/Users/ksoub/OneDrive/Desktop/34.py (3.13.7)
File Edit Format Run Options Window Help
import time
import sys
def is_palindrome(n):
if n < 0:
return False, {
"time_taken_ms": 0.0,
"math_iterations": 0,
"space_used_bytes": 0,
"notes": "Negative number treated as non-palindrome, no computation performed."
}
start_time = time.perf_counter()
original_number = n
reversed_number = 0
temp_n = n
iterations = 0
while temp_n > 0:
digit = temp_n % 10
reversed_number = (reversed_number * 10) + digit
temp_n = temp_n // 10
iterations += 1
is_palin = original_number == reversed_number
end_time = time.perf_counter()
time_taken_ms = (end_time - start_time) * 1000
space_used_bytes = 120
metrics = {
"time_taken_ms": time_taken_ms,
"math_iterations": iterations,
"space_used_bytes": space_used_bytes,
"notes": "Time is in milliseconds (ms). Iterations count the number of loop cycles."
}
return is_palin, metrics
test_numbers = [
1001,
12345,
7,
12345678987654321,
]
print("--- In-Memory Palindrome Checker Results ---")
for number in test_numbers:
result, metrics = is_palindrome(number)
print("-" * 40)
print(f"Checking Number: {number}")
print(f"Is Palindrome: {result}")
print("\n--- Performance Metrics ---")
print(f"1. Time Taken: {metrics['time_taken_ms']:.4f} ms")
print(f"2. Number of Steps (Iterations): {metrics['math_iterations']}")
print(f"3. Memory Used (Estimated): {metrics['space_used_bytes']} bytes")
Ln: 48
IDLE Shell 3.13.7
File Edit Shell Debug Options Window Help
Python 3.13.7 (tags/v3.13.7:bcee1c3, Aug 14 2025, 14:15:11) [MSC v.1944 64 bit (AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
========== RESTART: C:/Users/ksoub/OneDrive/Desktop/34.py ==========
--- In-Memory Palindrome Checker Results ---
Checking Number: 1001
Is Palindrome: True
--- Performance Metrics ---
1. Time Taken: 0.0032 ms
2. Number of Steps (Iterations): 4
3. Memory Used (Estimated): 120 bytes
Checking Number: 12345
Is Palindrome: False
--- Performance Metrics ---
1. Time Taken: 0.0022 ms
2. Number of Steps (Iterations): 5
3. Memory Used (Estimated): 120 bytes
Checking Number: 7
Is Palindrome: True
--- Performance Metrics ---
1. Time Taken: 0.0016 ms
2. Number of Steps (Iterations): 1
3. Memory Used (Estimated): 120 bytes
Checking Number: 12345678987654321
Is Palindrome: True
--- Performance Metrics ---
1. Time Taken: 0.0046 ms
2. Number of Steps (Iterations): 17
3. Memory Used (Estimated): 120 bytes
>>>
Ln: 38 Col: 0

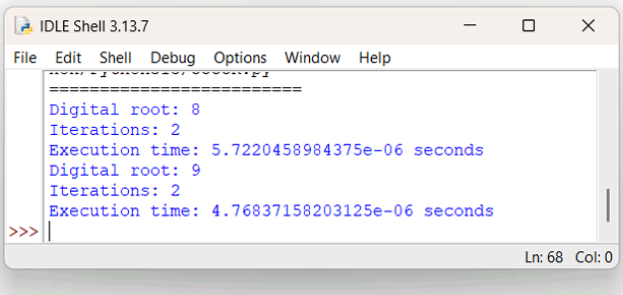
```python
import time
def is_deficient(n):
    start_time = time.time()
    iterations = 0

    if n <= 1:
        end_time = time.time()
        return False, iterations, end_time - start_time

    divisor_sum = 0
    i = 1
    while i * i <= n:
        iterations += 1
        if n % i == 0:
            if i != n:
                divisor_sum += i

            other_divisor = n // i
            if other_divisor != n and other_divisor != i:
                divisor_sum += other_divisor

        i += 1

    end_time = time.time()
    execution_time = end_time - start_time

    return divisor_sum < n, iterations, execution_time
number = 14
is_def, count, time_ns = is_deficient(number)

print("\n---")
print(f"Checking number: {number}")
print(f"Is Deficient (divisor_sum < n)? {is_def}")
print(f"Total Divisor Sum: {1+2+7}") # Sum is 10
print(f"Total Iterations: {count}")
print(f"Execution Time: {time_ns:.8f} seconds")
```

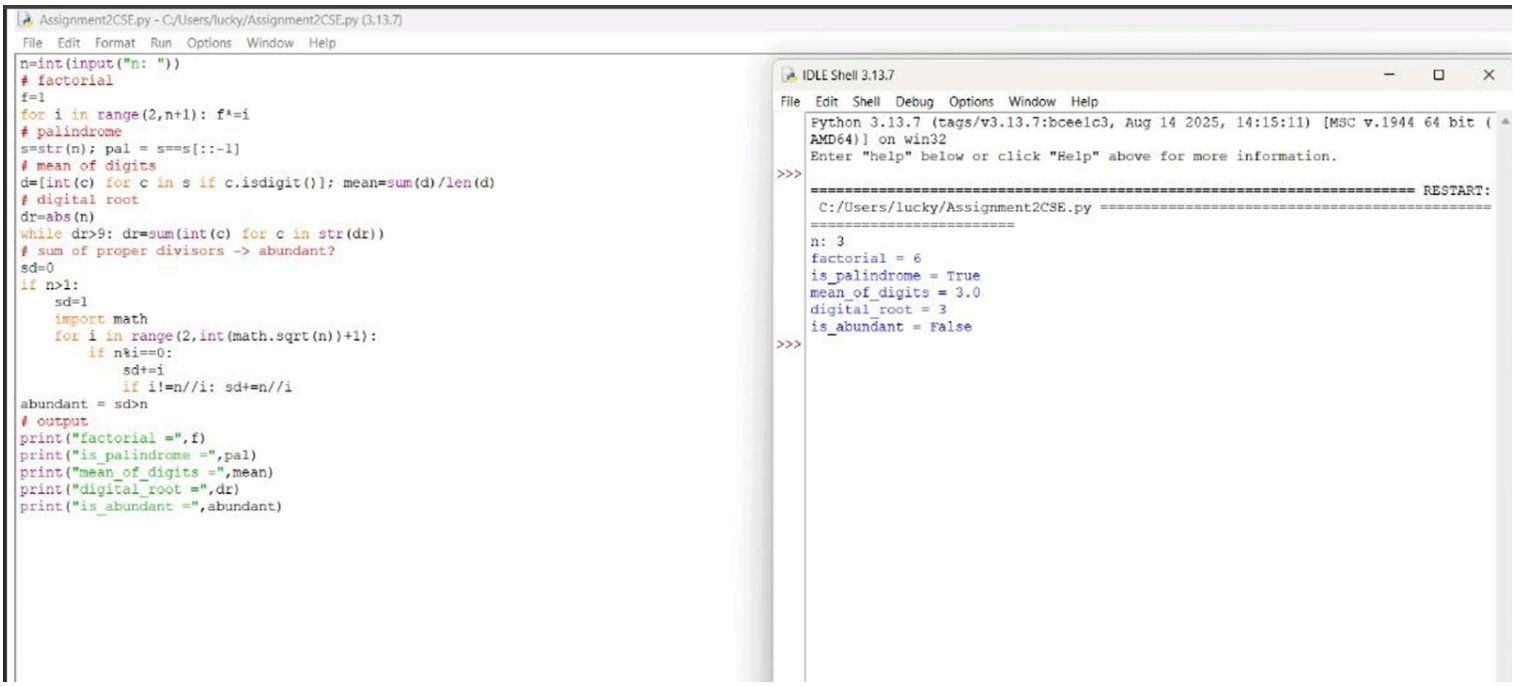


OOOOK.py - C:/Users/ksoub/AppData/Local/Programs/Python/Python313/OOOOK.py (3.13.7)

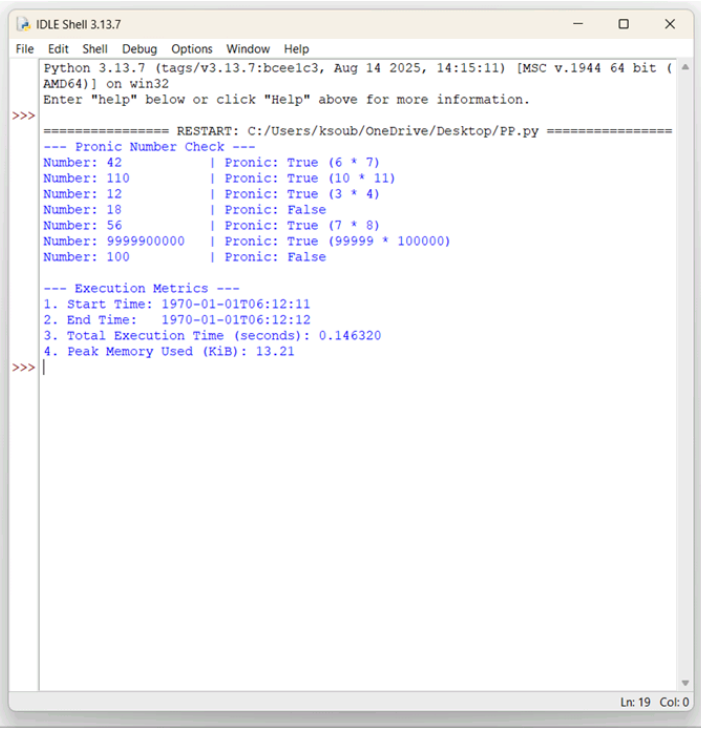
File Edit Format Run Options Window Help

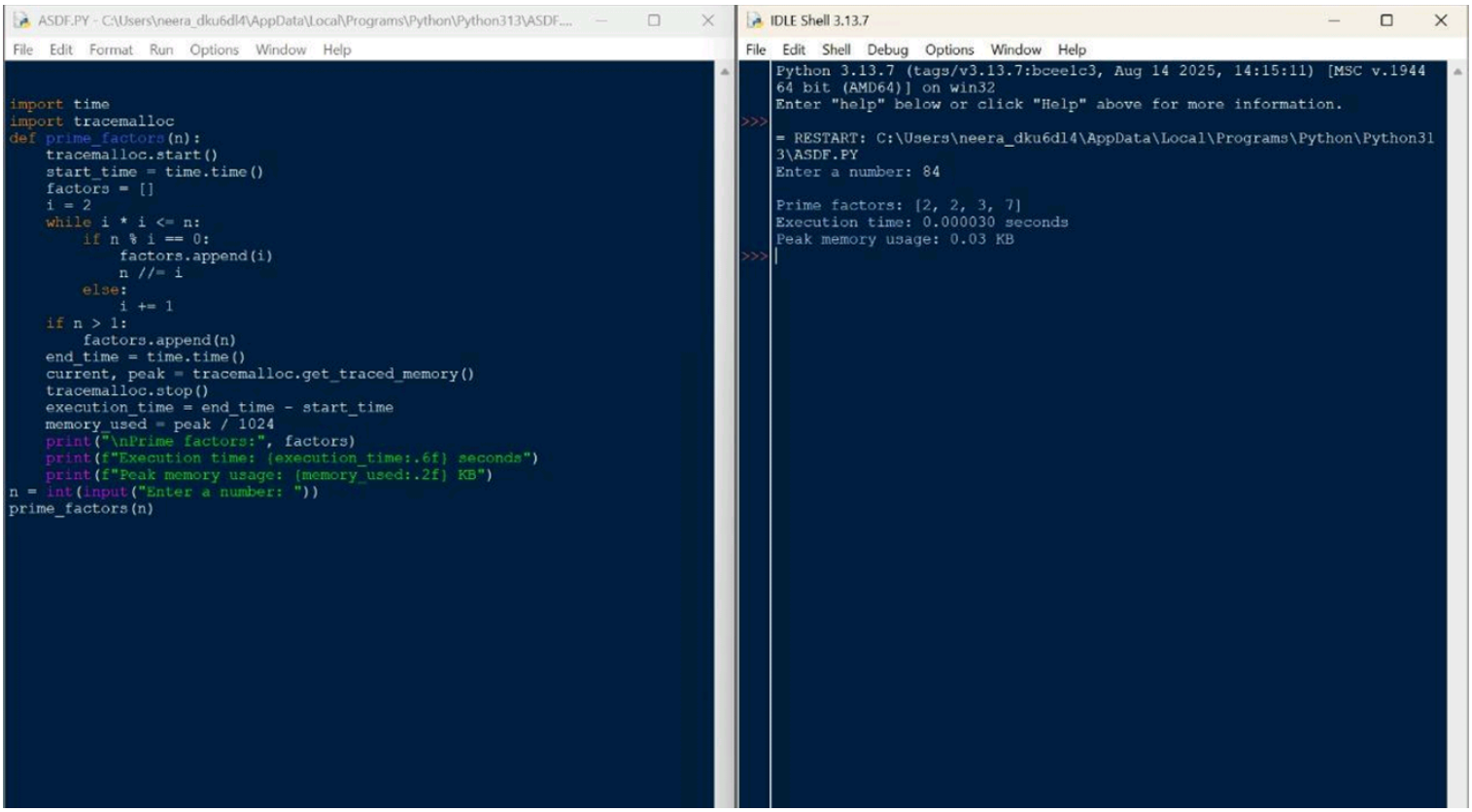
```
import time

def digital_root(n):
    iterations = 0
    start_time = time.time()   # start timer

    while n >= 10:   # loop until single digit
        s = 0
        temp = n
        while temp > 0:   # sum digits manually
            s += temp % 10
            temp //= 10
        n = s
        iterations += 1

    end_time = time.time()   # end timer
    execution_time = end_time - start_time

    print("Digital root:", n)
    print("Iterations:", iterations)
    print("Execution time:", execution_time, "seconds")

    return n
result = digital_root(49319)
result = digital_root(123456789)
```




Assignment2CSE.py - C:/Users/lucky/Assignment2CSE.py (3.13.7)
File Edit Format Run Options Window Help
n=int(input("n: "))
# factorial
f=1
for i in range(2,n+1): f*=i
# palindrome
s=str(n); pal = s==s[::-1]
# mean of digits
d=[int(c) for c in s if c.isdigit()]; mean=sum(d)/len(d)
# digital root
dr=abs(n)
while dr>9: dr=sum(int(c) for c in str(dr))
# sum of proper divisors -> abundant?
sd=0
if n>1:
sd=1
import math
for i in range(2,int(math.sqrt(n))+1):
if n%i==0:
sd+=i
if i!=n//i: sd+=n//i
abundant = sd>n
# output
print("factorial =",f)
print("is_palindrome =",pal)
print("mean_of_digits =",mean)
print("digital_root =",dr)
print("is_abundant =",abundant)
IDLE Shell 3.13.7
File Edit Shell Debug Options Window Help
Python 3.13.7 (tags/v3.13.7:bcee1c3, Aug 14 2025, 14:15:11) [MSC v.1944 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
========== RESTART:
C:/Users/lucky/Assignment2CSE.py ==========
n: 3
factorial = 6
is_palindrome = True
mean_of_digits = 3.0
digital_root = 3
is_abundant = False
>>>

```python
def is_pronic(n):
    if n < 0:
        return False, 0

    if n == 0:
        return True, 0

    limit = int(math.sqrt(n))

    for k in range(1, limit + 1):
        product = k * (k + 1)

        if product == n:
            return True, k

        if product > n:
            break

    return False, 0

numbers_to_test = [42, 110, 12, 18, 56, 9999900000, 100]

print("--- Pronic Number Check ---")
tracemalloc.start()
start_time = time.perf_counter()
start_time_iso = time.strftime("%Y-%m-%dT%H:%M:%S", time.localtime(start_time))

for n in numbers_to_test:
    is_pronic_res, k_val = is_pronic(n)

    result_str = f"Pronic: {is_pronic_res}"
    if is_pronic_res:
        result_str += f" ({k_val} * {k_val + 1})"

    print(f"Number: {n: <12} | {result_str}")

end_time = time.perf_counter()
end_time_iso = time.strftime("%Y-%m-%dT%H:%M:%S", time.localtime(end_time))
current_memory, peak_memory = tracemalloc.get_traced_memory()
tracemalloc.stop()
print("\n--- Execution Metrics ---")
print(f"1. Start Time: {start_time_iso}")
print(f"2. End Time:   {end_time_iso}")
print(f"3. Total Execution Time (seconds): {end_time - start_time:.6f}")
print(f"4. Peak Memory Used (KiB): {peak_memory / 1024:.2f}")
```



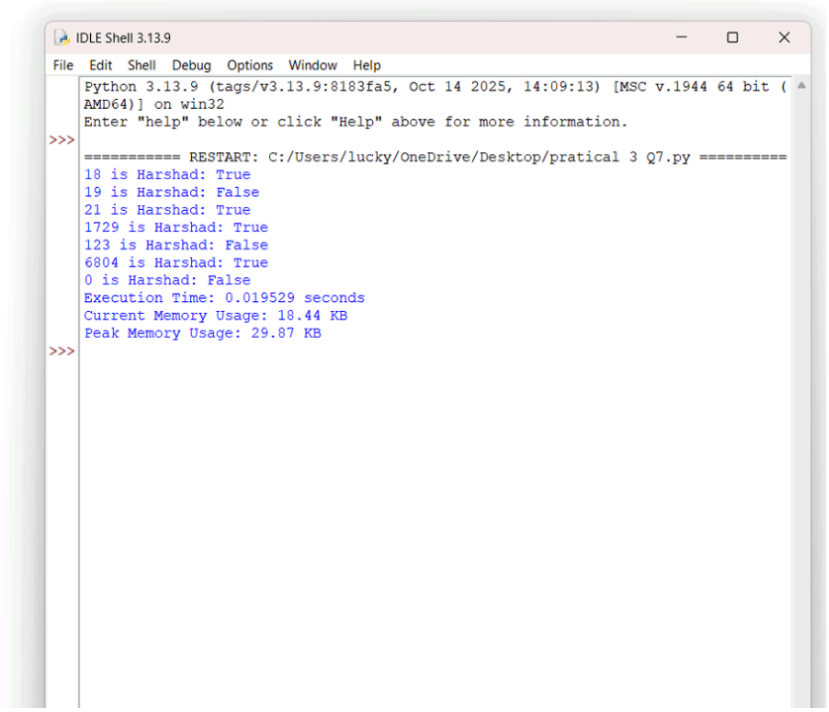

ASDF.PY - C:\Users\neera_dku6dl4\AppData\Local\Programs\Python\Python313\ASDF....
File Edit Format Run Options Window Help
import time
import tracemalloc
def prime_factors(n):
tracemalloc.start()
start_time = time.time()
factors = []
i = 2
while i * i <= n:
if n % i == 0:
factors.append(i)
n //= i
else:
i += 1
if n > 1:
factors.append(n)
end_time = time.time()
current, peak = tracemalloc.get_traced_memory()
tracemalloc.stop()
execution_time = end_time - start_time
memory_used = peak / 1024
print("\nPrime factors:", factors)
print(f"Execution time: {execution_time:.6f} seconds")
print(f"Peak memory usage: {memory_used:.2f} KB")
n = int(input("Enter a number: "))
prime_factors(n)
IDLE Shell 3.13.7
File Edit Shell Debug Options Window Help
Python 3.13.7 (tags/v3.13.7:bcee1c3, Aug 14 2025, 14:15:11) [MSC v.1944
64 bit (AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
= RESTART: C:\Users\neera_dku6dl4\AppData\Local\Programs\Python\Python31
3\ASDF.PY
Enter a number: 84
Prime factors: [2, 2, 3, 7]
Execution time: 0.000030 seconds
Peak memory usage: 0.03 KB
>>>

```python
import time
import sys

def prime_factors(n):
    factors = []

    while n % 2 == 0:
        factors.append(2)
        n //= 2

    i = 3
    while i * i <= n:
        while n % i == 0:
            factors.append(i)
            n //= i
        i += 2

    if n > 2:
        factors.append(n)

    return factors

def count_distinct_prime_factors(n):
    factors = prime_factors(n)
    return len(set(factors))

def analyze_execution(func, *args, **kwargs):
    start_time = time.perf_counter()

    result = func(*args, **kwargs)

    end_time = time.perf_counter()
    time_taken_ms = (end_time - start_time) * 1000

    memory_used_bytes = sys.getsizeof(result)

    return result, time_taken_ms, memory_used_bytes

number = 100
result, time_ms, memory_bytes = analyze_execution(prime_factors, number)

print(f"--- Analysis for Input: {number} ---")
print(f"The prime factors of {number} are: {result}")
print(f"**Time Taken:** {time_ms:.6f} milliseconds")
print(f"**Memory Used (Result Object):** {memory_bytes} bytes")

print("-" * 40)
```

```
IDLE Shell 3.13.7
File  Edit  Shell  Debug  Options  Window  Help
Python 3.13.7 (tags/v3.13.7:bcee1c3, Aug 14 2025, 14:15:11) [MSC v.1944 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
======== RESTART: C:/Users/ksoub/OneDrive/Desktop/ooooooooooooooooo.py =======
--- Analysis for Input: 100 ---
The prime factors of 100 are: [2, 2, 5, 5]
**Time Taken:** 0.012800 milliseconds
**Memory Used (Result Object):** 88 bytes
----------------------------------------
>>>
Ln: 10  Col: 0
```

Ln: 47

```python
import time
import tracemalloc

def is_harshad(n):
    digits_sum = sum(int(digit) for digit in str(n))
    return n % digits_sum == 0 if digits_sum != 0 else False

def test_harshad_numbers():
    test_numbers = [18, 19, 21, 1729, 123, 6804, 0]
    for num in test_numbers:
        print(f"{num} is Harshad: {is_harshad(num)}")
tracemalloc.start()
start_time = time.time()
test_harshad_numbers()
end_time = time.time()
current, peak = tracemalloc.get_traced_memory()
tracemalloc.stop()

print(f"Execution Time: {end_time - start_time:.6f} seconds")
print(f"Current Memory Usage: {current / 1024:.2f} KB")
print(f"Peak Memory Usage: {peak / 1024:.2f} KB")
```



Practical 3.py - C:\Users\ksoub\OneDrive\Documents\Practical 3.py (3.13.7)

File Edit Format Run Options Window Help

```
import time
import tracemalloc

def is_automorphic(n):
    square = n * n
    return str(square).endswith(str(n))
start_time = time.time()
tracemalloc.start()

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

if is_automorphic(num):
    print(num, "is an automorphic number.")
else:
    print(num, "is not an automorphic number.")
current, peak = tracemalloc.get_traced_memory()
tracemalloc.stop()
end_time = time.time()
print(f"\nExecution Time: {end_time - start_time:.10f} seconds")
print(f"Current Memory Usage: {current / 1024:.3f} KB")
print(f"Peak Memory Usage: {peak / 1024:.3f} KB")
```

IDLE Shell 3.13.7

File Edit Shell Debug Options Window Help

```
Python 3.13.7 (tags/v3.13.7:bcee1c3, Aug 14 2025, 14:15:11) [MSC v.1944 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
========== RESTART: C:\Users\ksoub\OneDrive\Documents\Practical 3.py ==========
Enter a number: 85
85 is not an automorphic number.

Execution Time: 2.0697088242 seconds
Current Memory Usage: 17.905 KB
Peak Memory Usage: 29.591 KB
>>>
```

File Edit Format Run Options Window Help

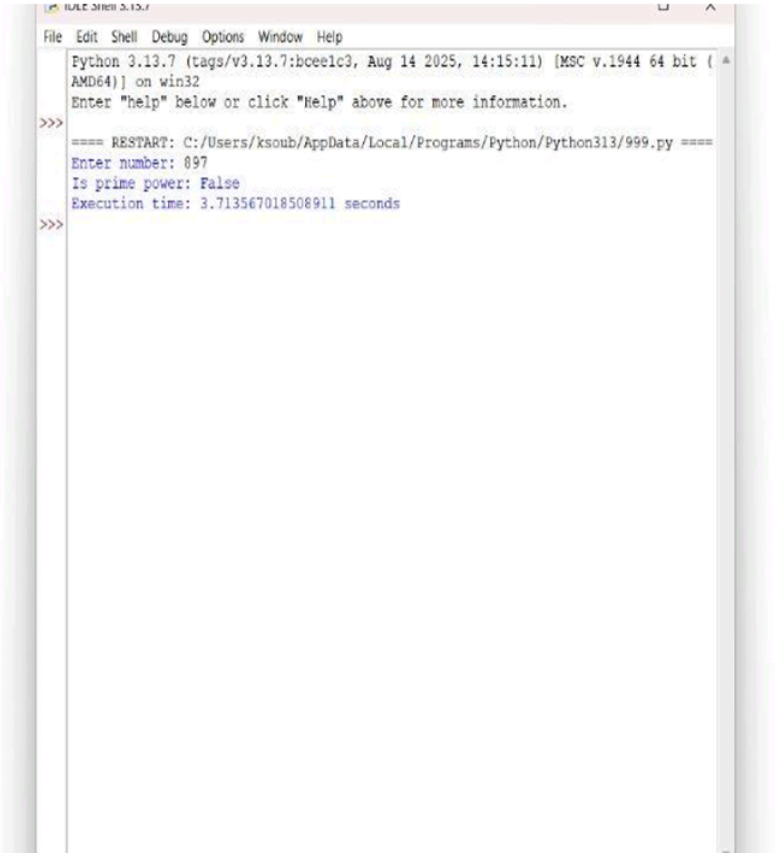
```
import time
import tracemalloc
def is_mersenne_prime(p: int) -> bool:

    if p == 2:
        return True
    if p < 2:
        return False

    M = (1 << p) - 1
    s = 4
    for _ in range(p - 2):
        s = (s * s - 2) % M
    return s == 0
p = 7
start_time = time.time()
tracemalloc.start()
result = is_mersenne_prime(p)
current, peak = tracemalloc.get_traced_memory()
end_time = time.time()
tracemalloc.stop()
print(f"Is 2^{p} - 1 a Mersenne Prime? : {result}")
print(f"Execution Time: {end_time - start_time:.6f} seconds")
print(f"Current Memory Usage: {current / 1024:.3f} KB")
print(f"Peak Memory Usage: {peak /1024:.3f} KB")
```

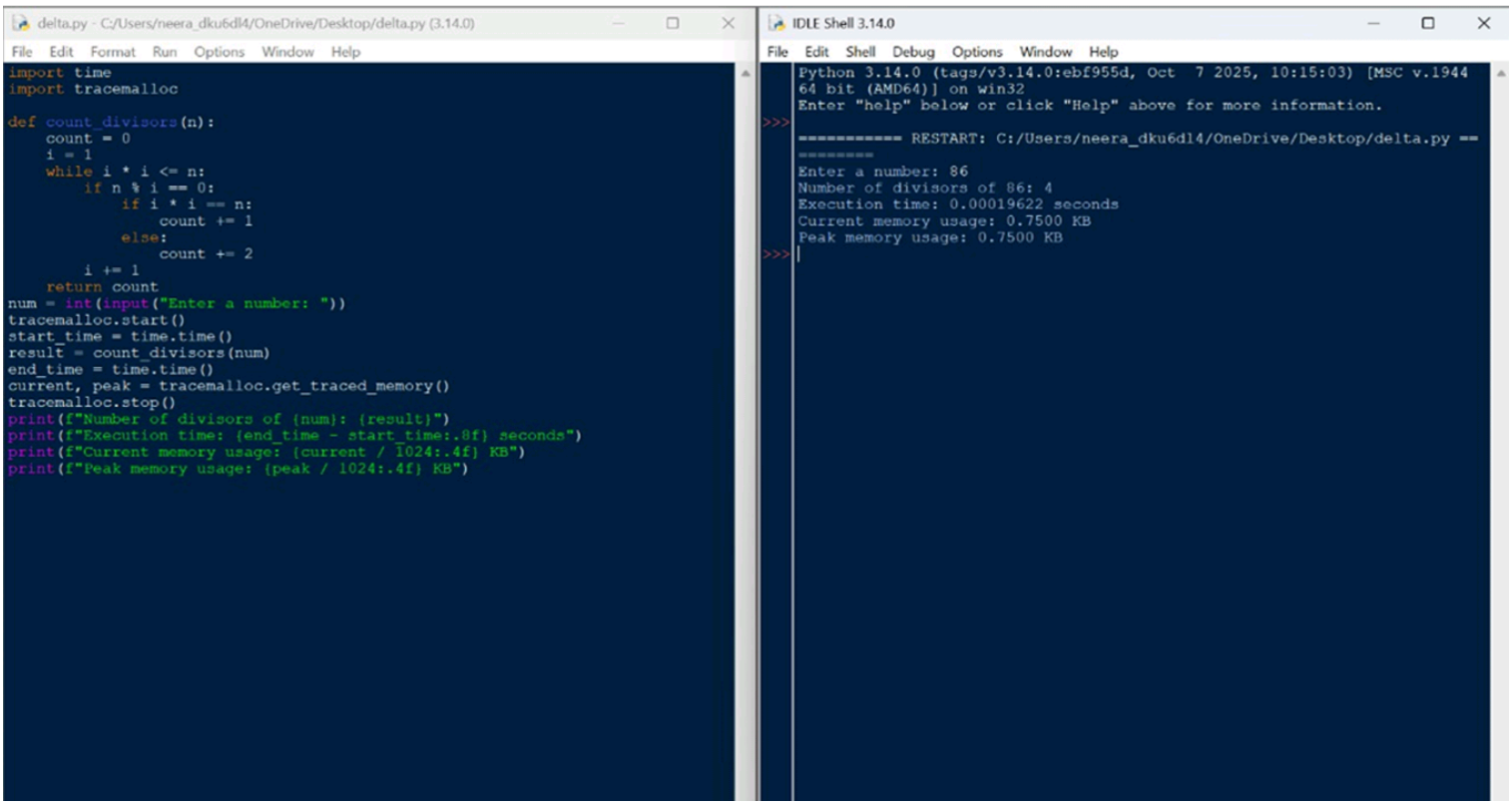
IDLE Shell 3.13.7

File Edit Shell Debug Options Window Help

```
Python 3.13.7 (tags/v3.13.7:bcee1c3, Aug 14 2025, 14:15:11) [MSC v.1944 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
========== RESTART: C:/Users/ksoub/OneDrive/Documents/00.py ==========
Is 2^7 - 1 a Mersenne Prime? : True
Execution Time: 0.000067 seconds
Current Memory Usage: 0.750 KB
Peak Memory Usage: 0.750 KB
>>>
```

```python
import time
def is_prime(x):

    if x <= 1:
        return False
    if x <= 3:
        return True
    if x % 2 == 0 or x % 3 == 0:
        return False
    i = 5
    while i * i <= x:
        if x % i == 0 or x % (i + 2) == 0:
            return False
        i += 6
    return True
def is_prime_power(n):
    if n <= 1:
        return False
    for p in range(2, int(n**0.5) + 1):
        if is_prime(p):
            power = p
            while power <= n:
                if power == n:
                    return True
                power *= p
    return is_prime(n)
start = time.time()

num = int(input("Enter number: "))
result = is_prime_power(num)

end = time.time()

print("Is prime power:", result)
print("Execution time:", (end - start),"seconds")
```




delta.py - C:/Users/neera_dku6dl4/OneDrive/Desktop/delta.py (3.14.0)
File Edit Format Run Options Window Help
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 3.14.0
File Edit Shell Debug Options Window Help
Python 3.14.0 (tags/v3.14.0:ebf955d, Oct 7 2025, 10:15:03) [MSC v.1944
64 bit (AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
========== RESTART: C:/Users/neera_dku6dl4/OneDrive/Desktop/delta.py ==
========
Enter a number: 86
Number of divisors of 86: 4
Execution time: 0.00019622 seconds
Current memory usage: 0.7500 KB
Peak memory usage: 0.7500 KB
>>>

File Edit Format Run Options Window Help

```
import time
import tracemalloc
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False
    return True
def twin_primes(limit):
    start_time = time.time()
    tracemalloc.start()

    twins = []
    prev_prime = 2

    for num in range(3, limit + 1):
        if is_prime(num):
            if num - prev_prime == 2:
                twins.append((prev_prime, num))
            prev_prime = num
    current, peak = tracemalloc.get_traced_memory()
    tracemalloc.stop()
    end_time = time.time()
    print(f"Twin primes up to {limit}:")
    print(twins)
    print(f"\nExecution time: {end_time - start_time:.6f} seconds")
    print(f"Current memory usage: {current / 1024:.2f} KB")
    print(f"Peak memory usage: {peak / 1024:.2f} KB")
twin_primes(100)
```

IDLE Shell 3.13.7

File Edit Shell Debug Options Window Help

```
Python 3.13.7 (tags/v3.13.7:bcee1c3, Aug 14 2025, 14:15:11) [MSC v.1944 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>

========== RESTART: C:/Users/ksoub/OneDrive/Documents/jk.py ==========
Twin primes up to 100:
[(3, 5), (5, 7), (11, 13), (17, 19), (29, 31), (41, 43), (59, 61), (71, 73)]

Execution time: 0.001601 seconds
Current memory usage: 0.06 KB
Peak memory usage: 0.17 KB
>>>
```

```python
import time
def aliquot_sum(n):
    if n == 1:
        return 0
    sum_div = 1
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            sum_div += i
            if i * i != n:
                sum_div += n // i
    return sum_div
num = 30
start_time = time.time()
result = aliquot_sum(num)
end_time = time.time()
print(f"The number is: {num}")
print(f"The sum of proper divisors (aliquot sum) is: {result}")
print(f"Time of execution: {end_time - start_time:.6f} seconds")
num = 30
```

IDLE Shell 3.13.7

File Edit Shell Debug Options Window Help

```
Python 3.13.7 (tags/v3.13.7:bcee1c3, Aug 14 2025, 14:15:11) [MSC v.1944 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
========== RESTART: C:/Users/ksoub/OneDrive/Desktop/iiiii.py ==========
The number is: 30
The sum of proper divisors (aliquot sum) is: 42
Time of execution: 0.000013 seconds
>>>
```

File Edit Format Run Options Window Help

```
        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_quadratic_residue(a: int, p: int) -> bool:
    if not isinstance(a, int) or not isinstance(p, int):
        raise ValueError("Inputs 'a' and 'p' must be integers.")
    if p < 1:
        raise ValueError("Modulus 'p' must be a positive integer.")
    if p == 1:
        return True
    if p == 2:
        return True
    if not is_prime(p):
        raise ValueError("Euler's Criterion requires 'p' to be a prime number.")
    a_mod_p = a % p
    if a_mod_p == 0:
        return True
    exponent = (p - 1) // 2
    result = pow(a_mod_p, exponent, p)
    if result == 1:
        return True
    if result == p - 1:
        return False
    return False
if __name__ == "__main__":
    TEST_A = 10
    TEST_P = 13
    print(f"--- Running Quadratic Residue Check: x^2 ≡ {TEST_A} (mod {TEST_P}) ---")
    start_time = time.perf_counter()
    try:
        is_residue = is_quadratic_residue(TEST_A, TEST_P)
    except ValueError as e:
        is_residue = f"Error: {e}"
    end_time = time.perf_counter()
    if isinstance(is_residue, bool):
        num_steps = math.ceil(math.log2(TEST_P - 1)) if TEST_P > 2 else 1
        print(f"\n1. Number of Steps (Modular Exponentiation Cycles): {num_steps}")
    else:
        print("\n1. Number of Steps: N/A (Error Occurred)")
```

IDLE Shell 3.13.7

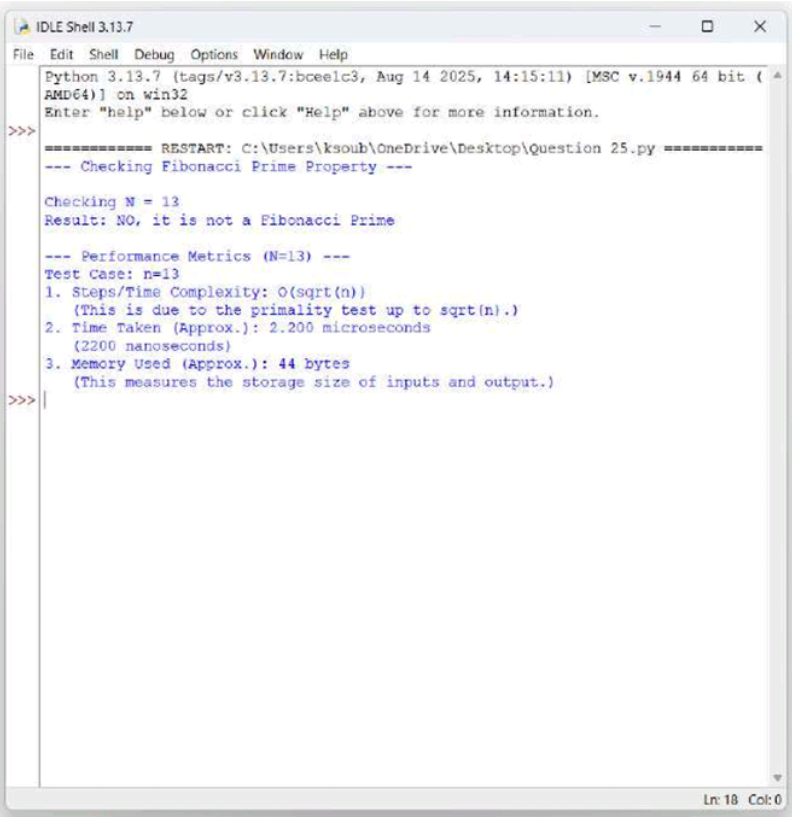
File Edit Shell Debug Options Window Help

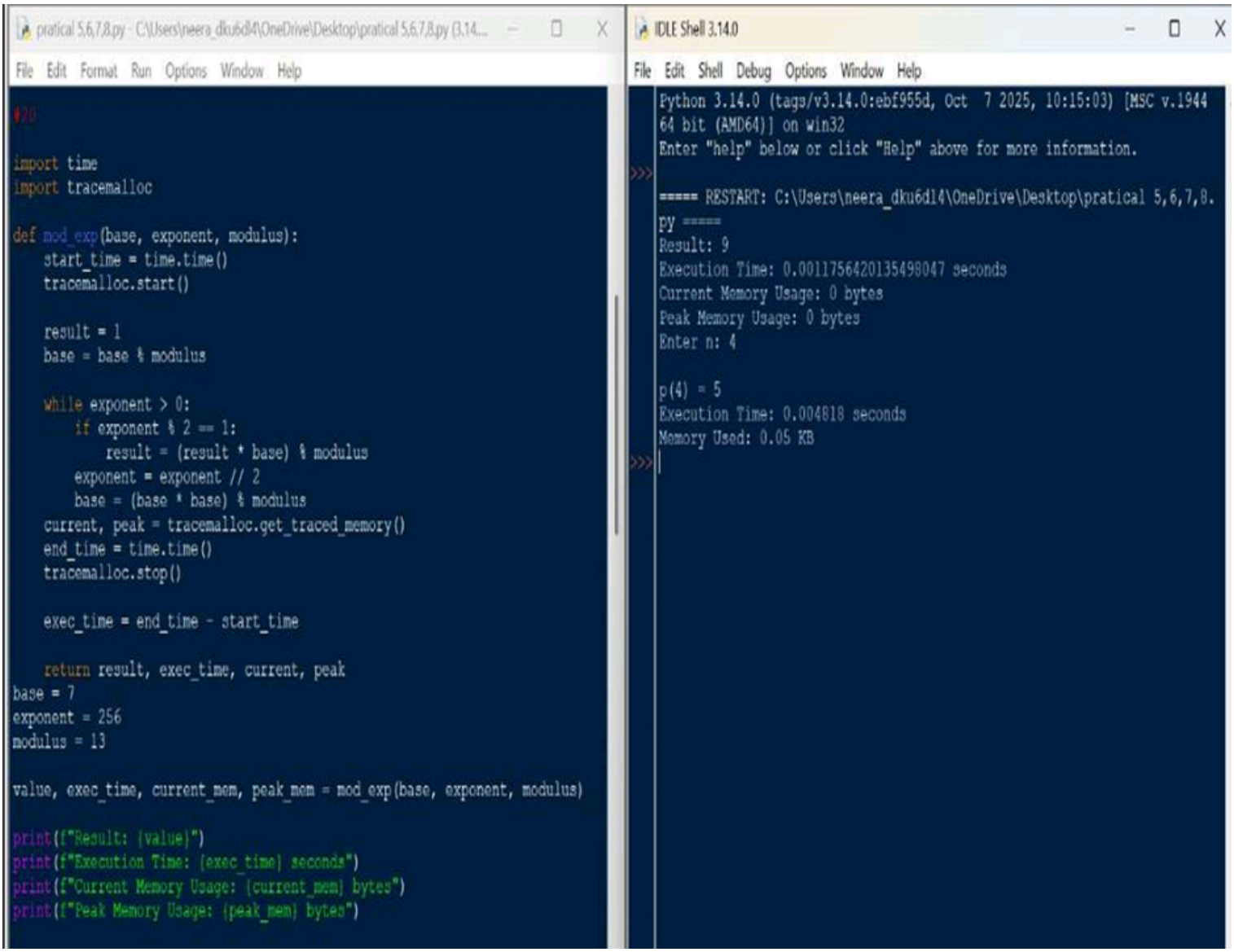
```
Python 3.13.7 (tags/v3.13.7:bcee1c3, Aug 14 2025, 14:15:11) [MSC v.1944 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
===== RESTART: C:/Users/ksoub/AppData/Local/Programs/Python/Python313/45.py ====
--- Running Quadratic Residue Check: x^2 ≡ 10 (mod 13) ---

1. Number of Steps (Modular Exponentiation Cycles): 4
2. Time Taken (Execution Time): 0.006500 milliseconds

3. Memory Used (Bytes):
   - Input A (10): 28 bytes
   - Input P (13): 28 bytes
   - Result (True): 28 bytes

--- Final Result ---
Is 10 a quadratic residue modulo 13? -> True
>>>
```

Ln: 17 Col: 0

Ln: 52 Col: 59

```python
import time
import sys
import math
class PerformanceMetrics:
    def __init__(self):
        self.steps = "O(sqrt(n))"
        self.time_ns = 0
        self.memory_bytes = 0
        self.result = None
        self.n = None
    def calculate_metrics(self, func, n):
        self.n = n
        start_time = time.perf_counter_ns()
        self.result = func(n)
        end_time = time.perf_counter_ns()
        self.time_ns = end_time - start_time
        n_size = sys.getsizeof(n)
        result_size = sys.getsizeof(self.result)
        self.memory_bytes = n_size + result_size
        self.steps = "O(sqrt(n))"
        return self.result
def is_fibonacci(n):
    if n < 0:
        return False
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_prime(n):
    if not is_prime(n):
        return False
    return is_fibonacci(n)
N_TEST_1 = 13
print(f"--- Checking Fibonacci Prime Property ---")
print(f"\nChecking N = {N_TEST_1}")
metrics_1 = PerformanceMetrics()
final_result_1 = metrics_1.calculate_metrics(is_fibonacci_prime, N_TEST_1)
print(f"Result: {'YES, it is a Fibonacci Prime' if final_result_1 else 'NO, it is not a Fibonacci Prime'}")
```




pratical 5,6,7,8.py - C:\Users\neera_dku6dl4\OneDrive\Desktop\pratical 5,6,7,8.py (3.14...
File Edit Format Run Options Window Help
#20
import time
import tracemalloc
def mod_exp(base, exponent, modulus):
start_time = time.time()
tracemalloc.start()
result = 1
base = base % modulus
while exponent > 0:
if exponent % 2 == 1:
result = (result * base) % modulus
exponent = exponent // 2
base = (base * base) % modulus
current, peak = tracemalloc.get_traced_memory()
end_time = time.time()
tracemalloc.stop()
exec_time = end_time - start_time
return result, exec_time, current, peak
base = 7
exponent = 256
modulus = 13
value, exec_time, current_mem, peak_mem = mod_exp(base, exponent, modulus)
print(f"Result: {value}")
print(f"Execution Time: {exec_time} seconds")
print(f"Current Memory Usage: {current_mem} bytes")
print(f"Peak Memory Usage: {peak_mem} bytes")
IDLE Shell 3.14.0
File Edit Shell Debug Options Window Help
Python 3.14.0 (tags/v3.14.0:ebf955d, Oct 7 2025, 10:15:03) [MSC v.1944
64 bit (AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
===== RESTART: C:\Users\neera_dku6dl4\OneDrive\Desktop\pratical 5,6,7,8.
py =====
Result: 9
Execution Time: 0.0011756420135498047 seconds
Current Memory Usage: 0 bytes
Peak Memory Usage: 0 bytes
Enter n: 4
p(4) = 5
Execution Time: 0.004818 seconds
Memory Used: 0.05 KB
>>>

File Edit Format Run Options Window Help

```
import time
import tracemalloc
def sum_of_proper_divisors(n):
    return sum(i for i in range(1, n) if n % i == 0)
def are_amicable(a, b):
    return sum_of_proper_divisors(a) == b and sum_of_proper_divisors(b) == a
a = 220
b = 284
tracemalloc.start()
start_time = time.time()
result = are_amicable(a, b)
current, peak = tracemalloc.get_traced_memory()
end_time = time.time()
tracemalloc.stop()
print(f"Are {a} and {b} amicable? -> {result}")
print(f"Execution Time: {(end_time - start_time):.10f} seconds")
print(f"Memory Usage: {current /1024:.4f} KB")
print(f"Peak Memory Usage: {peak /1024:.4f} KB")
```

IDLE Shell 3.13.7

File Edit Shell Debug Options Window Help

```
Python 3.13.7 (tags/v3.13.7:bcee1c3, Aug 14 2025, 14:15:11) [MSC v.1944 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
========== RESTART: C:/Users/ksoub/OneDrive/Documents/POL.py ==========
Are 220 and 284 amicable? -> True
Execution Time: 0.0001332760 seconds
Memory Usage: 0.0000 KB
Peak Memory Usage: 0.4922 KB
>>>
```

File Edit Format Run Options Window Help

```python
import time
import sys
import math
def calculate_digit_product(n):
    if n < 0:
        n = abs(n)
    product = 1
    for digit_char in str(n):
        product *= int(digit_char)
    return product
def multiplicative_persistence(n: int) -> int:
    if not isinstance(n, int) or n < 0:
        raise ValueError("Input must be a non-negative integer.")
    current_number = n
    steps = 0
    while current_number > 9:
        steps += 1
        current_number = calculate_digit_product(current_number)
    return steps
if __name__ == "__main__":
    TEST_NUMBER = 77
    print(f"--- Running Multiplicative Persistence for N = {TEST_NUMBER} ---")
    start_time = time.perf_counter()
    persistence_result = multiplicative_persistence(TEST_NUMBER)
    end_time = time.perf_counter()
    print(f"\n1. Number of Steps (Multiplicative Persistence): {persistence_result}")
    elapsed_time_ms = (end_time - start_time) * 1000
    print(f"2. Time Taken (Execution Time): {elapsed_time_ms:.6f} milliseconds")
    input_memory = sys.getsizeof(TEST_NUMBER)
    output_memory = sys.getsizeof(persistence_result)
    print(f"\n3. Memory Used (Bytes):")
    print(f"   - Input Number ({TEST_NUMBER}): {input_memory} bytes")
    print(f"   - Resulting Steps ({persistence_result}): {output_memory} bytes")
    if TEST_NUMBER <= 999:
        print("\n--- Step Breakdown ---")
        n_current = TEST_NUMBER
        step = 0
        while n_current > 9:
            step += 1
            product = calculate_digit_product(n_current)
            print(f"Step {step}: {n_current} -> {list(str(n_current))} -> Product: {product}")
            n_current = product
        print(f"Final Result: Single digit {n_current} reached in {step} steps.")
```

IDLE Shell 3.13.7

File Edit Shell Debug Options Window Help

```
Python 3.13.7 (tags/v3.13.7:bcee1c3, Aug 14 2025, 14:15:11) [MSC v.1944 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
===== RESTART: C:/Users/ksoub/AppData/Local/Programs/Python/Python313/45.py ====
--- Running Multiplicative Persistence for N = 77 ---

1. Number of Steps (Multiplicative Persistence): 4
2. Time Taken (Execution Time): 0.009500 milliseconds

3. Memory Used (Bytes):
   - Input Number (77): 28 bytes
   - Resulting Steps (4): 28 bytes

--- Step Breakdown ---
Step 1: 77 -> ['7', '7'] -> Product: 49
Step 2: 49 -> ['4', '9'] -> Product: 36
Step 3: 36 -> ['3', '6'] -> Product: 18
Step 4: 18 -> ['1', '8'] -> Product: 8
Final Result: Single digit 8 reached in 4 steps.
>>>
```

Ln: 20 Col: 0

File Edit Format Run Options Window Help

```
import math
import time
import sys
import collections
def count_divisors(n):
    if n <= 0:
        return 0
    if n == 1:
        return 1
    count = 0
    limit = int(math.sqrt(n))
    for i in range(1, limit + 1):
        if n % i == 0:
            count += 1
            if i * i != n:
                count += 1
    return count
def is_highly_composite(n):
    if n <= 0:
        return False
    if n == 1:
        return True

    divisors_n = count_divisors(n)

    for k in range(1, n):
        divisors_k = count_divisors(k)

        if divisors_k >= divisors_n:
            return False
    return True
def run_analysis(n_to_check):
    mem_before = sys.getsizeof(collections.Counter(locals()))
    start_time = time.time()
    is_hcn = is_highly_composite(n_to_check)
    end_time = time.time()
    mem_after = sys.getsizeof(collections.Counter(locals()))
    print(f"\n--- Checking if N = {n_to_check} is Highly Composite ---")
    print(f"Result: {n_to_check} is {'Highly Composite' if is_hcn else 'NOT Highly Composite'}")
    elapsed_time_ms = (end_time - start_time) * 1000
    print(f"TIME TAKEN: {elapsed_time_ms:.4f} milliseconds")
    memory_used_bytes = mem_after - mem_before
    print(f"MEMORY USED: {memory_used_bytes} bytes")
    estimated_steps = int(n_to_check * math.sqrt(n_to_check) * 1.5)
    print(f"NO. OF STEPS: ~{estimated_steps:,} operations")
RUN_N = 12
run_analysis(RUN_N)
```

Ln: 41 Col: 60

IDLE Shell 3.13.7

File Edit Shell Debug Options Window Help

```
Python 3.13.7 (tags/v3.13.7:bcee1c3, Aug 14 2025, 14:15:11) [MSC v.1944 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
========== RESTART: C:/Users/ksoub/Downloads/II.py ==========

--- Checking if N = 12 is Highly Composite ---
Result: 12 is Highly Composite
TIME TAKEN: 0.0200 milliseconds
MEMORY USED: 0 bytes
NO. OF STEPS: ~62 operations
>>>
```

Ln: 11 Col: 0

```python
import time
import sys
import collections
def lucas_sequence(n: int) -> list[int]:
    if not isinstance(n, int) or n < 0:
        raise ValueError("Input 'n' must be a non-negative integer.")
    if n == 0:
        return []
    if n == 1:
        return [2]
    if n == 2:
        return [2, 1]
    sequence = [2, 1]
    for _ in range(2, n):
        next_lucas = sequence[-1] + sequence[-2]
        sequence.append(next_lucas)
    return sequence
def run_lucas_sequence_with_metrics(n: int):
    start_time = time.perf_counter()
    lucas_nums = lucas_sequence(n)
    end_time = time.perf_counter()
    time_taken = (end_time - start_time) * 1000
    memory_used_bytes = sys.getsizeof(lucas_nums)
    steps = max(0, n - 2)
    print(f"--- Lucas Sequence (First {n} Numbers) ---")
    print(lucas_nums)
    print("\n--- Performance Metrics ---")
    print(f"N: {n} terms")
    print(f"Time Taken: {time_taken:.6f} milliseconds")
    print(f"Memory Used: {memory_used_bytes} bytes (Size of the output list)")
    print(f"No. of Steps (Main Loop Iterations): {steps}")
if __name__ == "__main__":
    run_lucas_sequence_with_metrics(n=10)
    print("\n" + "="*50 + "\n")
    run_lucas_sequence_with_metrics(n=20)
```

IDLE Shell 3.13.7

File Edit Shell Debug Options Window Help

```
Python 3.13.7 (tags/v3.13.7:bcee1c3, Aug 14 2025, 14:15:11) [MSC v.1944 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
========== RESTART: C:\Users\ksoub\OneDrive\Desktop\QUESTION 26.py ==========
--- Lucas Sequence (First 10 Numbers) ---
[2, 1, 3, 4, 7, 11, 18, 29, 47, 76]

--- Performance Metrics ---
N: 10 terms
Time Taken: 0.012200 milliseconds
Memory Used: 184 bytes (Size of the output list)
No. of Steps (Main Loop Iterations): 8

==================================================

--- Lucas Sequence (First 20 Numbers) ---
[2, 1, 3, 4, 7, 11, 18, 29, 47, 76, 123, 199, 322, 521, 843, 1364, 2207, 3571, 5
778, 9349]

--- Performance Metrics ---
N: 20 terms
Time Taken: 0.004600 milliseconds
Memory Used: 248 bytes (Size of the output list)
No. of Steps (Main Loop Iterations): 18
>>>
```

Ln: 24 Col: 0

Ln: 1 Col: 0

File Edit Format Run Options Window Help

```
import time

def mod_inverse(a, m):
    m0 = m
    y = 0
    x = 1

    if m == 1:
        return 0
    while a > 1:
        q = a // m
        t = m
        m = a % m
        a = t
        t = y
        y = x - q * y
        x = t
    if m != 1:
        return None
    if x < 0:
        x += m0

    return x
val_a = 3
val_m = 11
start_time = time.time()
inverse_result = mod_inverse(val_a, val_m)
end_time = time.time()
print(f"Finding the modular multiplicative inverse of {val_a} mod {val_m}:")
if inverse_result is not None:
    print(f"The inverse x is: {inverse_result}")
    print(f"Verification: ({val_a} * {inverse_result}) % {val_m} == {(val_a * inverse_result) % val_m}")
else:
    print(f"Inverse does not exist for {val_a} mod {val_m} (GCD is not 1).")
print(f"Time of execution: {end_time - start_time:.6f} seconds")
```

Ln: 34 Col: 76

IDLE Shell 3.13.7

File Edit Shell Debug Options Window Help

```
Python 3.13.7 (tags/v3.13.7:bcee1c3, Aug 14 2025, 14:15:11) [MSC v.1944 6
4 bit (AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
========== RESTART: C:/Users/ksoub/OneDrive/Desktop/iiiii.py =======
=======
Finding the modular multiplicative inverse of 3 mod 11:
Inverse does not exist for 3 mod 11 (GCD is not 1).
Time of execution: 0.000007 seconds
>>>
```

Ln: 8 Col: 0

File Edit Format Run Options Window Help

```
import math
import time
import tracemalloc
import sys
def extended_gcd(a, b):
    if b == 0:
        return (a, 1, 0)
    g, x1, y1 = extended_gcd(b, a % b)
    return (g, y1, x1 - (a // b) * y1)
def mod_inverse(a, m):
    g, x, _ = extended_gcd(a, m)
    if g != 1:
        raise ValueError("Inverse does not exist (moduli must be pairwise coprime).")
    return x % m
def crt(remainders, moduli):
    if len(remainders) != len(moduli):
        raise ValueError("Remainders and moduli lists must have the same length.")
    M = 1
    for m in moduli:
        M *= m
    result = 0
    for r, m in zip(remainders, moduli):
        Mi = M // m
        inv = mod_inverse(Mi, m)
        result += r * Mi * inv
    return result % M
remainders = [2, 3, 2]
moduli = [3, 5, 7]
tracemalloc.start()
start_time = time.perf_counter()
x = crt(remainders, moduli)
end_time = time.perf_counter()
current_memory, peak_memory = tracemalloc.get_traced_memory()
tracemalloc.stop()
print(f"## CRT Calculation Result")
print(f"x = {x}")
print("---")
print(f"## Time Taken")
time_taken_seconds = end_time - start_time
print(f"**Execution Time (Wall Time):** {time_taken_seconds * 1e6:.3f} microseconds (µs)")
print("---")
print(f"## Memory Used")
print(f"**Peak Memory Usage:** {peak_memory / 1024:.2f} KiB")
print(f"**Current Memory Usage:** {current_memory / 1024:.2f} KiB")
```

IDLE Shell 3.13.7

File Edit Shell Debug Options Window Help

```
Python 3.13.7 (tags/v3.13.7:bcee1c3, Aug 14 2025, 14:15:11) [MSC v.1944 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.

>>>

========== RESTART: C:/Users/kscub/OneDrive/Documents/POL.py ==========
## CRT Calculation Result
x = 23
---
## Time Taken
**Execution Time (Wall Time):** 92.800 microseconds (µs)
---
## Memory Used
**Peak Memory Usage:** 0.20 KiB
**Current Memory Usage:** 0.00 KiB
>>>
```

Ln: 14 Col: 0

File Edit Format Run Options Window Help

```
import time
import sys
import math
def collatz_length(n: int) -> int:
    if not isinstance(n, int) or n < 1:
        raise ValueError("Input must be a positive integer.")

    steps = 0
    while n != 1:
        steps += 1
        if n % 2 == 0:
            n = n // 2
        else:
            n = 3 * n + 1
    return steps
if __name__ == "__main__":
    TEST_NUMBER = 27
    print(f"--- Running Collatz Sequence Length for N = {TEST_NUMBER} ---")
    start_time = time.perf_counter()
    try:
        collatz_result = collatz_length(TEST_NUMBER)
    except ValueError as e:
        collatz_result = f"Error: {e}"
    end_time = time.perf_counter()
    if isinstance(collatz_result, int):
        print(f"\n1. Number of Steps (Collatz Length): {collatz_result}")
    else:
        print(f"\n1. Number of Steps: N/A ({collatz_result})")
    elapsed_time_ms = (end_time - start_time) * 1000
    print(f"2. Time Taken (Execution Time): {elapsed_time_ms:.6f} milliseconds")
    input_memory = sys.getsizeof(TEST_NUMBER)
    output_memory = sys.getsizeof(collatz_result)
    print(f"\n3. Memory Used (Bytes):")
    print(f"   - Input Number ({TEST_NUMBER}): {input_memory} bytes")
    print(f"   - Resulting Steps ({collatz_result}): {output_memory} bytes")
    if TEST_NUMBER < 1000 and isinstance(collatz_result, int):
        print("\n--- Sequence Breakdown (First 15 steps) ---")
        n_current = TEST_NUMBER
        sequence = [n_current]
        step_count = 0
        while n_current != 1 and step_count < 15:
            step_count += 1
            if n_current % 2 == 0:
                n_current = n_current // 2
            else:
                n_current = 3 * n_current + 1
            sequence.append(n_current)
```

Ln: 3 Col: 1

IDLE Shell 3.13.7

File Edit Shell Debug Options Window Help

```
Python 3.13.7 (tags/v3.13.7:bcee1c3, Aug 14 2025, 14:15:11) [MSC v.1944 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
===== RESTART: C:/Users/ksoub/AppData/Local/Programs/Python/Python313/45.py ====
--- Running Collatz Sequence Length for N = 27 ---

1. Number of Steps (Collatz Length): 111
2. Time Taken (Execution Time): 0.012300 milliseconds

3. Memory Used (Bytes):
   - Input Number (27): 28 bytes
   - Resulting Steps (111): 28 bytes

--- Sequence Breakdown (First 15 steps) ---
Sequence: 27 -> 82 -> 41 -> 124 -> 62 -> 31 -> 94 -> 47 -> 142 -> 71 -> 214 -> 1
07 -> 322 -> 161 -> 484 -> 242 -> ... (sequence is longer)
>>>
```

Ln: 16 Col: 0

File Edit Format Run Options Window Help

```
import time
def power_with_modulo(base, exp, mod):
    result = 1
    base %= mod
    while exp > 0:
        if exp % 2 == 1:
            result = (result * base) % mod
        base = (base * base) % mod
        exp //= 2
    return result
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 gcd(a, b):
    while b:
        a, b = b, a % b
    return a
def is_carmichael(n):
  if n is composite:
    if is_prime(n):
        return False

    for a in range(2, n):
        if gcd(a, n) == 1:
            if power_with_modulo(a, n - 1, n) != 1:
                return False
    return True
number_to_check = 561
start_time = time.time()
if is_carmichael(number_to_check):
    print(f"{number_to_check} is a Carmichael number.")
else:
    print(f"{number_to_check} is not a Carmichael number (or is prime).")

end_time = time.time()
execution_time = end_time - start_time
```

Ln: 29 Col: 20

IDLE Shell 3.13.7

File Edit Shell Debug Options Window Help

```
Python 3.13.7 (tags/v3.13.7:bcee1c3, Aug 14 2025, 14:15:11) [MSC v.1944 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>

========== RESTART: C:/Users/ksoub/OneDrive/Desktop/iiiii.py ==========
561 is a Carmichael number.
Time of execution: 0.009894 seconds
>>>
```

Ln: 15 Col: 0

assaf.py - C:\Users\neera_dku6dl4\AppData\Local\Programs\Python\Python314\assaf...

File Edit Format Run Options Window Help

```
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)

if result is None:
    print("Order does not exist because a and n are not coprime.")
else:
    print(f"Order of {a} mod {n} is {result}.")
```

IDLE Shell 3.14.0

File Edit Shell Debug Options Window Help

```
Python 3.14.0 (tags/v3.14.0:ebf955d, Oct  7 2025, 10:15:03) [MSC v.1944
64 bit (AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
= RESTART: C:\Users\neera_dku6dl4\AppData\Local\Programs\Python\Python31
4\assaf.py
Enter a number: 56
Execution Time: 0.000036 seconds
Memory Used: 0.00 KB
56 is NOT a highly composite number.
>>>
= RESTART: C:\Users\neera_dku6dl4\AppData\Local\Programs\Python\Python31
4\assaf.py
Enter a: 2
Enter n: 11
Execution Time: 0.000038 seconds
Memory Used: 0.00 KB
Order of 2 mod 11 is 10.
>>>
```

File Edit Format Run Options Window Help

```
import random
import math
import time
import tracemalloc
def pollard_rho(n):
    if n % 2 == 0:
        return 2, 1
    def f(x, c):
        return (x * x + c) % n
    x = random.randint(2, n - 1)
    y = x
    c = random.randint(1, n - 1)
    d = 1
    steps = 0
    while d == 1:
        steps += 1
        x = f(x, c)
        y = f(f(y, c), c)
        d = math.gcd(abs(x - y), n)
        if d == n:
            new_factor, new_steps = pollard_rho(n)
            return new_factor, steps + new_steps
    return d, steps
if __name__ == "__main__":
    n = 8051
    print(f"--- Starting Factorization of N={n} ---")
    tracemalloc.start()
    start_time = time.time()
    factor, steps = pollard_rho(n)
    end_time = time.time()
    current, peak = tracemalloc.get_traced_memory()
    tracemalloc.stop()
    print("\n--- Results ---")
    print(f"Number to factorize (n): {n}")
    print(f"A factor of {n} is: {factor}")
    print(f"Number of Steps/Iterations: {steps}")
    print(f"Execution Time: {(end_time - start_time):.10f} seconds")
    print(f"Memory Usage: {current / 1024:.4f} KB")
    print(f"Peak Memory Usage: {peak / 1024:.4f} KB")
```

IDLE Shell 3.13.7

File Edit Shell Debug Options Window Help

```
Python 3.13.7 (tags/v3.13.7:bcee1c3, Aug 14 2025, 14:15:11) [MSC v.1944 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
========== RESTART: C:/Users/ksoub/OneDrive/Documents/POL.py ==========
--- Starting Factorization of N=8051 ---

--- Results ---
Number to factorize (n): 8051
A factor of 8051 is: 97
Number of Steps/Iterations: 7
Execution Time: 0.0001010895 seconds
Memory Usage: 0.0000 KB
Peak Memory Usage: 0.4180 KB
>>>
```

File Edit Format Run Options Window Help

```
import time
import tracemalloc
import sys
def polygonal_number(s, n):
    return ((s - 2) * n * n - (s - 4) * n) // 2
if __name__ == "__main__":
    examples = [
        (3, 5, "Triangular"),
        (4, 5, "Square"),
        (5, 5, "Pentagonal")
    ]
    tracemalloc.start()
    start_time = time.time()
    steps = 0
    results = []
    for s, n, shape in examples:
        result = polygonal_number(s, n)
        results.append((shape, s, n, result))
        steps += 1
    end_time = time.time()
    current, peak = tracemalloc.get_traced_memory()
    tracemalloc.stop()
    print(f"--- Polygonal Number Calculation (n=5) ---")
    for shape, s, n, result in results:
        print(f"{shape} Number (s={s}): {result}")
    print("\n--- Performance Metrics ---")
    print(f"Total Function Calls (Steps): {steps}")
    print(f"Execution Time: {(end_time - start_time):.10f} seconds")
    print(f"Memory Usage: {current / 1024:.4f} KB")
    print(f"Peak Memory Usage: {peak / 1024:.4f} KB")
```

IDLE Shell 3.13.7

File Edit Shell Debug Options Window Help

```
Python 3.13.7 (tags/v3.13.7:bcee1c3, Aug 14 2025, 14:15:11) [MSC v.1944 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
========== RESTART: C:/Users/ksoub/OneDrive/Documents/POL.py ==========
--- Polygonal Number Calculation (n=5) ---
Triangular Number (s=3): 15
Square Number (s=4): 25
Pentagonal Number (s=5): 35

--- Performance Metrics ---
Total Function Calls (Steps): 3
Execution Time: 0.0000526905 seconds
Memory Usage: 0.9219 KB
Peak Memory Usage: 0.9688 KB
>>>
```

File Edit Format Run Options Window Help

```
import time
import sys
import math
def collatz_length(n: int) -> int:
    if not isinstance(n, int) or n < 1:
        raise ValueError("Input must be a positive integer.")

    steps = 0
    while n != 1:
        steps += 1
        if n % 2 == 0:
            n = n // 2
        else:
            n = 3 * n + 1
    return steps
if __name__ == "__main__":
    TEST_NUMBER = 27
    print(f"--- Running Collatz Sequence Length for N = {TEST_NUMBER} ---")
    start_time = time.perf_counter()
    try:
        collatz_result = collatz_length(TEST_NUMBER)
    except ValueError as e:
        collatz_result = f"Error: {e}"
    end_time = time.perf_counter()
    if isinstance(collatz_result, int):
        print(f"\n1. Number of Steps (Collatz Length): {collatz_result}")
    else:
        print(f"\n1. Number of Steps: N/A ({collatz_result})")
    elapsed_time_ms = (end_time - start_time) * 1000
    print(f"2. Time Taken (Execution Time): {elapsed_time_ms:.6f} milliseconds")
    input_memory = sys.getsizeof(TEST_NUMBER)
    output_memory = sys.getsizeof(collatz_result)
    print(f"\n3. Memory Used (Bytes):")
    print(f"   - Input Number ({TEST_NUMBER}): {input_memory} bytes")
    print(f"   - Resulting Steps ({collatz_result}): {output_memory} bytes")
    if TEST_NUMBER < 1000 and isinstance(collatz_result, int):
        print("\n--- Sequence Breakdown (First 15 steps) ---")
        n_current = TEST_NUMBER
        sequence = [n_current]
        step_count = 0
        while n_current != 1 and step_count < 15:
            step_count += 1
            if n_current % 2 == 0:
                n_current = n_current // 2
            else:
                n_current = 3 * n_current + 1
            sequence.append(n_current)
```

Ln: 3 Col: 1

IDLE Shell 3.13.7

File Edit Shell Debug Options Window Help

```
Python 3.13.7 (tags/v3.13.7:bcee1c3, Aug 14 2025, 14:15:11) [MSC v.1944 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
===== RESTART: C:/Users/ksoub/AppData/Local/Programs/Python/Python313/45.py ====
--- Running Collatz Sequence Length for N = 27 ---

1. Number of Steps (Collatz Length): 111
2. Time Taken (Execution Time): 0.612300 milliseconds

3. Memory Used (Bytes):
   - Input Number (27): 28 bytes
   - Resulting Steps (111): 28 bytes

--- Sequence Breakdown (First 15 steps) ---
Sequence: 27 -> 82 -> 41 -> 124 -> 62 -> 31 -> 94 -> 47 -> 142 -> 71 -> 214 -> 1
07 -> 322 -> 161 -> 484 -> 242 -> ... (sequence is longer)
>>>
```

Ln: 16 Col: 0

File Edit Format Run Options Window Help

```
import time
import random
def is_prime_miller_rabin(n, k=20):
    if n < 2:
        return False
    if n == 2 or n == 3:
        return True
    if n % 2 == 0:
        return False
    s = 0
    d = n - 1
    while d % 2 == 0:
        d //= 2
        s += 1
    for _ in range(k):
        a = random.randint(2, n - 2)
        x = pow(a, d, n)
        if x == 1 or x == n - 1:
            continue
        for _ in range(s - 1):
            x = pow(x, 2, n)
            if x == n - 1:
                break
        else:
            return False
    return True
number_to_test = 100000007
rounds = 20
start_time = time.time()
is_prime_1 = is_prime_miller_rabin(number_to_test, rounds)
end_time = time.time()
print(f"Testing number: {number_to_test} with {rounds} rounds.")
print(f"Is {number_to_test} prime? {'Yes, probably' if is_prime_1 else 'No, it is composite'}")
print(f"Time of execution: {end_time - start_time:.6f} seconds")
```

Go to any page between 1 and 107. (Ctrl+Alt+G)

IDLE Shell 3.13.7

File Edit Shell Debug Options Window Help

```
Python 3.13.7 (tags/v3.13.7:bcee1c3, Aug 14 2025, 14:15:11) [MSC v.1944 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
========== RESTART: C:/Users/ksoub/OneDrive/Desktop/iiiii.py ==========
Testing number: 100000007 with 20 rounds.
Is 100000007 prime? Yes, probably
Time of execution: 0.000041 seconds
>>>
```

Ln: 8 Col: 0

pratical 5,6,7,8.py - C:/Users/neera_dku6dl4/OneDrive/Desktop/pratical 5,6,7,8.py (3.14...

File Edit Format Run Options Window Help

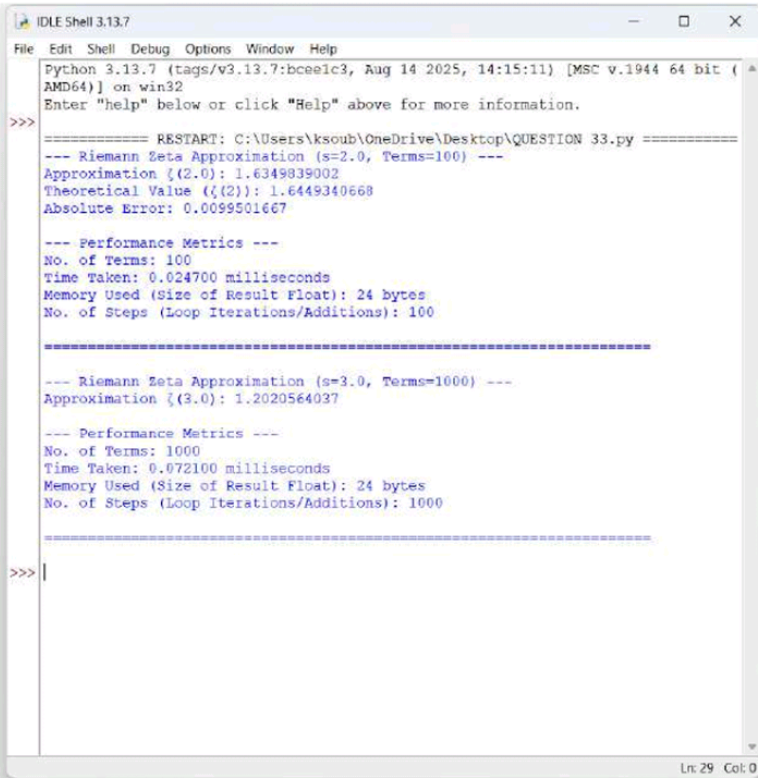
```
import time
import tracemalloc

def partition_function(n):
    p = [0] * (n + 1)
    p[0] = 1
    for k in range(1, n + 1):
        total = 0
        m = 1
        while True:
            g1 = m * (3*m - 1) // 2
            g2 = m * (3*m + 1) // 2
            if g1 > k:
                break
            sign = -1 if (m % 2 == 0) else 1
            total += sign * p[k - g1]

            if g2 <= k:
                total += sign * p[k - g2]
            m += 1
        p[k] = total
    return p[n]
n = int(input("Enter n: "))

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

result = partition_function(n)

current, peak = tracemalloc.get_traced_memory()
tracemalloc.stop()
exec_time = time.time() - start
print(f"\np({n}) = {result}")
print(f"Execution Time: {exec_time:.6f} seconds")
print(f"Memory Used: {peak/1024:.2f} KB")
```

IDLE Shell 3.14.0

File Edit Shell Debug Options Window Help

```
Python 3.14.0 (tags/v3.14.0:ebf955d, Oct  7 2025, 10:15:03) [MSC v.1944
64 bit (AMD64)] on win32
Enter "help" below or click "Help" above for more information.

>>>

===== RESTART: C:/Users/neera_dku6dl4/OneDrive/Desktop/pratical 5,6,7,8.
py =====
Enter s (number of sides): 4
Enter n (term number): 5

5-th 4-gonal number = 25
Execution Time: 0.000057 seconds
Memory Used: 0.75 KB
>>>

= RESTART: C:/Users/neera_dku6dl4/OneDrive/Desktop/pratical 5,6,7,8.py =
Enter s (number of sides): 4
Enter n (term number): 6

6-th 4-gonal number = 36
Execution Time: 0.000056 seconds
Memory Used: 0.75 KB
>>>

= RESTART: C:/Users/neera_dku6dl4/OneDrive/Desktop/pratical 5,6,7,8.py =
Enter n: 5

p(5) = 7
Execution Time: 0.000111 seconds
Memory Used: 0.05 KB
>>>
```

```python
import time
import sys
import math
def zeta_approx(s: float, terms: int) -> float:
    if s <= 1.0:
        raise ValueError("The series approximation is only valid for s > 1.")
    if terms <= 0:
        return 0.0
    result = 0.0
    for k in range(1, terms + 1):
        term = 1.0 / (k ** s)
        result += term
    return result
def run_zeta_approx_with_metrics(s: float, terms: int):
    print(f"--- Riemann Zeta Approximation (s={s}, Terms={terms}) ---")
    try:
        start_time = time.perf_counter()
        approximation = zeta_approx(s, terms)
        end_time = time.perf_counter()
        time_taken = (end_time - start_time) * 1000
        memory_used_bytes = sys.getsizeof(approximation)
        steps = terms
        print(f"Approximation ζ({s}): {approximation:.10f}")
        if s == 2.0:
            theoretical_value = (math.pi**2) / 6
            error = abs(theoretical_value - approximation)
            print(f"Theoretical Value (ζ(2)): {theoretical_value:.10f}")
            print(f"Absolute Error: {error:.10f}")
        print("\n--- Performance Metrics ---")
        print(f"No. of Terms: {terms}")
        print(f"Time Taken: {time_taken:.6f} milliseconds")
        print(f"Memory Used (Size of Result Float): {memory_used_bytes} bytes")
        print(f"No. of Steps (Loop Iterations/Additions): {steps}")
    except ValueError as e:
        print(f"ERROR: {e}")
if __name__ == "__main__":
    run_zeta_approx_with_metrics(s=2.0, terms=100)
    print("\n" + "="*70 + "\n")

    run_zeta_approx_with_metrics(s=3.0, terms=1000)
    print("\n" + "="*70 + "\n")
```



Lab Manual

Practical and Skills Development

CERTIFICATE

PERFORMED BY
THE ASSIGNMENT ENTERED IN THIS REPORT HAVE BEEN SATISFACTORILY

Registration No	: 25BCE10682
Name of Student	: SHREYASH SHANKAR BHERE
Course Name	: Introduction to Problem Solving and Programming
Course Code	: CSE1021
School Name	: SCAI
Slot	: B11+B12+B13
Class ID	: BL2025260100796
Semester	: FALL 2025/26

Course Faculty Name : Dr. Hemraj S. Lamkuche

Signature:

VIT BHOPAL

QUESTION 1 :Factorial Calculation and Divisor Count Analysis

AIM/OBJECTIVE(s):

1. To calculate the **factorial** of a user-input number (n).
2. To measure the **execution time** required for the factorial calculation.
3. To count the total number of **divisors** for the input number (n).
4. To estimate the approximate **memory used** based on the number of divisors.

METHODOLOGY & TOOL USED:

. **Methodology:**Iterative Calculation: A for loop is used to calculate the factorial by repeatedly multiplying a running total (t). A separate for loop is used to check for and count the divisors of the input number (n) .

Tool Used: Python programming language, utilizing the built-in time module for performance measurement.

BRIEF DESCRIPTION:

The program first prompts the user to **enter a number (n)**. It then immediately starts a timer. It calculates the factorial of n using a loop that runs from 1 to n, accumulating

Lab Manual

Practical and Skills Development

CERTIFICATE

PERFORMED BY

THE ASSIGNMENT ENTERED IN THIS REPORT HAVE BEEN SATISFACTORILY

Registration No	: 25BCE10682
Name of Student	: SHREYASH SHANKAR BHERE
Course Name	: Introduction to Problem Solving and Programming
Course Code	: CSE1021
School Name	: SCAI
Slot	: B11+B12+B13
Class ID	: BL2025260100796
Semester	: FALL 2025/26

Course Faculty Name : Dr. Hemraj S. Lamkuche

Signature: