main.py

Share Run

```python
def is_prime(n):
    """Check if a number is prime"""
    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 twin_primes(limit):
    """Generate all twin prime pairs up to a given limit"""
    twin_primes = []
    for i in range(2, limit - 1):
        if is_prime(i) and is_prime(i + 2):
            twin_primes.append((i, i + 2))
    return twin_primes

# Example usage:
limit = int(input("Enter the limit: "))
print(twin_primes(limit))
```

Output

```
Enter the limit: 34
[(3, 5), (5, 7), (11, 13), (17, 19), (29, 31)]

=== Code Execution Successful ===
```

main.py

Share Run

```python
def zeta_approx(s, terms):
    zeta = 0
    for n in range(1, terms + 1):
        zeta += 1 / (n ** s)
    return zeta

s = float(input("enter the value of s: "))
terms = int(input("enter the number of terms: "))
print(f"ζ({s}) ≈ {zeta_approx(s, terms)}")
```

Output

```
enter the value of s: 3
enter the number of terms: 4
ζ(3.0) ≈ 1.177662037037037

=== Code Execution Successful ===
```

main.py

```python
from functools import reduce

def crt(remainders, moduli):
    def mul_inv(a, b):
        b0 = b; x0, x1 = 0, 1
        if b == 1: return 1
        while a > 1:
            q = a // b
            a, b = b, a%b
            x0, x1 = x1 - q * x0, x0
        return x1 + b0 if x1 < 0 else x1

    M = reduce(lambda x, y: x*y, moduli)
    return sum(r * M//m * mul_inv(M//m, m) for r, m in zip(remainders, moduli)) % M

remainders = list(map(int, input("Enter the remainders (space-separated): ").split()))
moduli = list(map(int, input("Enter the moduli (space-separated): ").split()))
print(f"Solution to the system of congruences: x ≡ {crt(remainders,moduli)}")
```

Share Run

Output

```
Enter the remainders (space-separated): 3
Enter the moduli (space-separated): 5
Solution to the system of congruences: x ≡ 3

=== Code Execution Successful ===
```

main.py

Share Run

```python
def is_quadratic_residue(a: int, p: int) -> bool:

    if not isinstance(p, int) or p < 2 or not all(p % i for i in range(2, int(p**0.5) + 1)):
        raise ValueError("p must be a prime number")

    # Euler's criterion
    return pow(a, (p - 1) // 2, p) == 1

a = int(input("Enter the number: "))
p = int(input("Enter the prime modulus: "))
print(f"{a} is {'a quadratic residue' if is_quadratic_residue(a, p) else 'not a quadratic residue'} mod {p}")
```

Output

```
Enter the number: 3
Enter the prime modulus: 5
3 is not a quadratic residue mod 5

=== Code Execution Successful ===
```

main.py

Share Run

```python
def is_pronic(n):
    i = 0
    while i * (i + 1) <= n:
        if i * (i + 1) == n:
            return True
        i += 1
    return False

n = int(input("Enter the number: "))
print(f"{n} is {'a pronic number' if is_pronic(n) else 'not a pronic number'}")
```

Output

```
Enter the number: 34
34 is not a pronic number

=== Code Execution Successful ===
```

main.py

Share Run

```python
def aliquot_sum(n: int) -> int:
    if n < 1:
        raise ValueError("Input must be a positive integer.")
    total = 0
    for i in range(1, int(n**0.5) + 1):
        if n % i == 0:
            if n // i == i:
                total += i
            else:
                total += i
                if i != 1:
                    total += n // i
    return total - n

def are_amicable(a: int, b: int) -> bool:
    if a < 1 or b < 1:
        raise ValueError("Inputs must be positive integers.")
    return aliquot_sum(a) == b and aliquot_sum(b) == a

a = int(input("Enter the first number: "))
b = int(input("Enter the second number: "))
print(f"{a} and {b} are {'amicable' if are_amicable(a, b) else 'not amicable'}.")
```

Output

```
Enter the first number: 3
Enter the second number: 4
3 and 4 are not amicable.

=== Code Execution Successful ===
```

main.py

Share Run

```python
import random

def is_prime_miller_rabin(n, k):
    if n < 2:
        return False
    if n == 2 or n == 3:
        return True
    if n % 2 == 0:
        return False

    def check(a, s, d, n):
        x = pow(a, d, n)
        if x == 1 or x == n - 1:
            return True
        for _ in range(s - 1):
            x = pow(x, 2, n)
            if x == n - 1:
                return True
        return False

    s = 0
    d = n - 1
    while d % 2 == 0:
        d >>= 1
        s += 1

    for _ in range(k):
        a = random.randint(2, n - 1)
        if not check(a, s, d, n):
            return False
    return True

n = int(input("Enter the number: "))
k = int(input("Enter the number of rounds: "))
print(f"{n} is {'probably prime' if is_prime_miller_rabin(n, k) else 'composite'}")
```

Output

```
Enter the number: 3
Enter the number of rounds: 4
3 is probably prime

=== Code Execution Successful ===
```

main.py

Share Run

```python
def prime_factors(n):
    factors = []
    i = 2
    while i * i <= n:
        if n % i:
            i += 1
        else:
            n //= i
            factors.append(i)
    if n > 1:
        factors.append(n)
    return factors

num = int(input("Enter a number: "))
print(prime_factors(num))
```

Output

```
Enter a number: 34
[2, 17]

=== Code Execution Successful ==
```

main.py

```python

def polygonal_number(s: int, n: int) -> int:

    if s < 3:
        raise ValueError("Input s must be an integer greater than 2.")
    if n < 1:
        raise ValueError("Input n must be a positive integer.")

    return (n**2 * (s - 2) - n * (s - 4)) // 2

s = int(input("Enter the number of sides of the polygon: "))
n = int(input("Enter the position of the polygonal number: "))
print(f"The {n}-th {s}-gonal number is {polygonal_number(s, n)}")
```

Share Run

Output

```
Enter the number of sides of the polygon: 3
Enter the position of the polygonal number: 4
The 4-th 3-gonal number is 10

=== Code Execution Successful ===
```

main.py

Share Run

```python
import random

def gcd(a, b):
    while b:
        a, b = b, a % b
    return a

def pollard_rho(n):
    if n % 2 == 0:
        return 2
    x = random.randint(1, n - 1)
    y = x
    c = random.randint(1, n - 1)
    g = 1
    while g == 1:
        x = (x * x + c) % n
        y = (y * y + c) % n
        y = (y * y + c) % n
        g = gcd(abs(x - y), n)
    if g == n:
        return none
    return g

n = int(input("enter the number: "))
factor = pollard_rho(n)
if factor:
    print(f"{n} = {factor} * {n // factor}")
else:
    print("failed to find a factor")
```

Output

```
enter the number: 34
34 = 2 * 17

=== Code Execution Successful ===
```

main.py

Share Run

```python

def is_perfect_power(n: int) -> bool:

    if n < 1:
        return False

    for b in range(2, int(n**0.5) + 1):
        a = 2
        while a**b <= n:
            if a**b == n:
                return True
            a += 1

    return False

n = int(input("Enter the number: "))
print(f"{n} is {'a perfect power' if is_perfect_power(n) else 'not a perfect power'}")
```

Output

```
Enter the number: 34
34 is not a perfect power

=== Code Execution Successful ===
```

main.py

Share

Run

```python
def partition_function(n):
    partitions = [0] * (n + 1)
    partitions[0] = 1
    for i in range(1, n + 1):
        for j in range(i, n + 1):
            partitions[j] += partitions[j - i]
    return partitions[n]

n = int(input("Enter the number: "))
print(f"The number of partitions of {n} is {partition_function(n)}")
```

Output

```
Enter the number: 34
The number of partitions of 34 is 12310

=== Code Execution Successful ===
```

main.py

Share Run

```python
def is_palindrome(s: str) -> bool:
    return s ==0

def main():
    s = input("Enter a number: ")
    print(f"{s} is {'a' if is_palindrome(s) else 'not a'} palindrome.")

if __name__ == "__main__":
    main()
```

Output

```
Enter a number: 34
34 is not a palindrome.

=== Code Execution Successful ===
```

main.py

Share

Run

```python
def mod_exp(base, exponent, modulus):
    result = 1
    base = base % modulus
    while exponent > 0:
        if exponent % 2 == 1:
            result = (result * base) % modulus
        exponent = exponent // 2
        base = (base * base) % modulus
    return result

def order_mod(a, n):
    k = 1
    while True:
        if mod_exp(a, k, n) == 1:
            return k
        k += 1

a = int(input("Enter the number: "))
n = int(input("Enter the modulus: "))
print(f"The order of {a} modulo {n} is {order_mod(a, n)}")
```

Output

```
Enter the number: 3
Enter the modulus: 4
The order of 3 modulo 4 is 2

=== Code Execution Successful ===
```

main.py

Share

Run

```python
def multiplicative_persistence(n: int) -> int:
    if n < 0:
        raise ValueError("Input must be a non-negative integer.")
    steps = 0
    while n >= 10:
        product = 1
        for digit in str(n):
            product *= int(digit)
        n = product
        steps += 1
    return steps

num = int(input("Enter a non-negative integer: "))
print(f"Multiplicative persistence: {multiplicative_persistence(num)}")
```

Output

```
Enter a non-negative integer: 34
Multiplicative persistence: 2

=== Code Execution Successful ===
```

main.py

Share Run

```python
def mod_inverse(a: int, m: int) -> int:
    def extended_gcd(a: int, b: int) -> tuple:
        if a == 0:
            return b, 0, 1
        else:
            gcd, x, y = extended_gcd(b % a, a)
            return gcd, y - (b // a) * x, x

    gcd, x, _ = extended_gcd(a, m)
    if gcd != 1:
        raise ValueError("Modular inverse does not exist.")
    return x % m

a = int(input("Enter the number: "))
m = int(input("Enter the modulus: "))
print(f"Modular inverse of {a} mod {m} = {mod_inverse(a, m)}")
```

Output

```
Enter the number: 3
Enter the modulus: 4
Modular inverse of 3 mod 4 = 3

=== Code Execution Successful ===
```

main.py

Share Run

```python
def mod_exp(base: int, exponent: int, modulus: int) -> int:
    if modulus < 1:
        raise ValueError("Modulus must be a positive integer.")
    result = 1
    base = base % modulus
    while exponent > 0:
        if exponent % 2 == 1:
            result = (result * base) % modulus
        exponent = exponent >> 1
        base = (base * base) % modulus
    return result

base = int(input("Enter the base: "))
exponent = int(input("Enter the exponent: "))
modulus = int(input("Enter the modulus: "))
print(f"({base}^{exponent}) % {modulus} = {mod_exp(base, exponent, modulus)}")
```

Output

```
Enter the base: 3
Enter the exponent: 4
Enter the modulus: 34
(3^4) % 34 = 13

=== Code Execution Successful ===
```

main.py

Share Run

```python
import time
import tracemalloc

def mean_of_digits(n):
    try:
        return sum(map(int, str(abs(n)))) / len(str(abs(n)))
    except ZeroDivisionError:
        return 0

try:
    n = int(input('enter your value :'))
    start_time = time.time()
    tracemalloc.start()
    result = mean_of_digits(n)
    end_time = time.time()
    memory, peak = tracemalloc.get_traced_memory()
    tracemalloc.stop()

    print(f"{result:.2f}")
    print(f"{(end_time - start_time) * 1e6:.2f} µs")
    print(f"{peak} bytes")
except ValueError:
    print("Invalid input. Please enter an integer.")
```

Output

```
enter your value :34
3.50
252.25 µs
167 bytes

=== Code Execution Successful ===
```

main.py

Share Run

```python
def lucas_sequence(n):
    sequence = [2, 1]
    while len(sequence) < n:
        sequence.append(sequence[-1] + sequence[-2])
    return sequence[:n]

num = int(input("Enter the number of Lucas numbers to generate: "))
print(lucas_sequence(num))
```

Output

Clear

```
Enter the number of Lucas numbers to generate: 34
[2, 1, 3, 4, 7, 11, 18, 29, 47, 76, 123, 199, 322, 521, 843, 1364, 2207, 3571, 5778, 9349, 15127, 24476, 39603, 64079, 103682, 167761, 271443, 439204, 710647, 1149851, 1860498,
    3010349, 4870847, 7881196]

=== Code Execution Successful ===
```

main.py

Share Run

```python
def count_divisors(n: int) -> int:
    count = 0
    for i in range(1, int(n**0.5) + 1):
        if n % i == 0:
            if n // i == i:
                count += 1
            else:
                count += 2
    return count

def is_highly_composite(n: int) -> bool:
    if n < 1:
        raise ValueError("Input must be a positive integer.")
    max_divisors = 0
    for i in range(1, n):
        max_divisors = max(max_divisors, count_divisors(i))
    return count_divisors(n) > max_divisors

num = int(input("Enter a positive integer: "))
print(f"{num} is {'highly composite' if is_highly_composite(num) else 'not highly composite'}.")
```

Output

```
Enter a positive integer: 34
34 is not highly composite.

=== Code Execution Successful ===
```

main.py

Share

Run

```python
def is_harshad(n):
    sum_digits = sum(int(digit) for digit in str(n))
    return n % sum_digits == 0

n = int(input("Enter the number: "))
print(f"{n} is {'a Harshad number' if is_harshad(n) else 'not a Harshad number'}")
```

Output

```
Enter the number: 34
34 is not a Harshad number

=== Code Execution Successful ===
```

main.py

Share Run

```python
import math

def is_perfect_square(x):
    s = int(math.sqrt(x))
    return s*s == x

def is_fibonacci(n):
    return is_perfect_square(5*n*n + 4) or is_perfect_square(5*n*n - 4)

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):
    return is_fibonacci(n) and is_prime(n)

num = int(input("Enter a positive integer: "))
print(f"{num} is {'a' if is_fibonacci_prime(num) else 'not a'} Fibonacci prime number.")
```

Output

```
Enter a positive integer: 34
34 is not a Fibonacci prime number.

=== Code Execution Successful ===
```

main.py

Share Run

```python
def factorial(n: int) -> int:
    if n < 0:
        raise ValueError("Input must be a non-negative integer.")
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

def main():
    try:
        n = int(input("Enter a non-negative integer: "))
        print(f"The factorial of {n} is: {factorial(n)}")
    except ValueError as e:
        print(f"Error: {e}")

if __name__ == "__main__":
    main()
```

Output

```
Enter a non-negative integer: 34
The factorial of 34 is: 295232799039604140847618609643520000000

=== Code Execution Successful ===
```

main.py

Share Run

```python
def digital_root(n: int) -> int:
    return (n - 1) % 9 + 1 if n > 0 else 0

def main():
    n = int(input("Enter a number: "))
    print(f"The digital root of {n} is {digital_root(n)}.")

if __name__ == "__main__":
    main()
```

Output

```
Enter a number: 34
The digital root of 34 is 7.

=== Code Execution Successful ===
```

main.py

Share Run

```python
def is_deficient(n: int) -> bool:
    """Return True if the sum of proper divisors of n is less than n."""
    if n < 1:
        raise ValueError("Input must be a positive integer.")
    return sum(i for i in range(1, n) if n % i == 0) < n

def main():
    n = int(input("Enter a number: "))
    print(f"{n} is {'deficient' if is_deficient(n) else 'not deficient'}.")

if __name__ == "__main__":
    main()
```

Output

```
Enter a number: 34
34 is deficient.

=== Code Execution Successful ===
```

main.py

Share Run

```python
def count_divisors(n):
    count = 0
    for i in range(1, int(n**0.5) + 1):
        if n % i == 0:
            if n // i == i:
                count += 1
            else:
                count += 2
    return count

num = int(input("Enter a positive integer: "))
if num <= 0:
    print("Please enter a positive integer.")
else:
    print(f"The number of divisors of {num} is: {count_divisors(num)}")
```

Output

```
Enter a positive integer: 3
The number of divisors of 3 is: 2

=== Code Execution Successful ===4
```

main.py

Share Run

```python
def collatz_length(n: int) -> int:

    if n < 1:
        raise ValueError("Input must be a positive integer.")

    length = 1
    while n != 1:
        if n % 2 == 0:
            n = n // 2
        else:
            n = 3 * n + 1
        length += 1

    return length

n = int(input("Enter the starting number: "))
print(f"The Collatz sequence length for {n} is {collatz_length(n)}")
```

Output

```
Enter the starting number: 34
The Collatz sequence length for 34 is 14

=== Code Execution Successful ===
```

main.py

Share Run

```python
import math

def gcd(a, b):
    while b:
        a, b = b, a % b
    return a

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

def is_carmichael(n):
    if is_prime(n):
        return False
    for a in range(2, n):
        if gcd(a, n) == 1 and pow(a, n-1, n) != 1:
            return False
    return True

n = int(input("Enter the number: "))
print(f"{n} is {'a Carmichael number' if is_carmichael(n) else 'not a Carmichael number'}")
```

Output

```
Enter the number: 34
34 is not a Carmichael number

=== Code Execution Successful ===
```

main.py

Share Run

```python
def is_automorphic(n):
    square = str(n ** 2)
    return square.endswith(str(n))

n = int(input("Enter the number: "))
print(f"{n} is {'an automorphic number' if is_automorphic(n) else 'not an automorphic number'}")
```

Output

```
Enter the number: 34
34 is not an automorphic number

=== Code Execution Successful ===
```

main.py

Share Run

```python
def aliquot_sum(n: int) -> int:
    if n < 1:
        raise ValueError("Input must be a positive integer.")
    total = 0
    for i in range(1, int(n**0.5) + 1):
        if n % i == 0:
            if n // i == i:
                total += i
            else:
                total += i
                if i != 1:
                    total += n // i
    return total - n

def are_amicable(a: int, b: int) -> bool:
    if a < 1 or b < 1:
        raise ValueError("Inputs must be positive integers.")
    return aliquot_sum(a) == b and aliquot_sum(b) == a

a = int(input("Enter the first number: "))
b = int(input("Enter the second number: "))
print(f"{a} and {b} are {'amicable' if are_amicable(a, b) else 'not amicable'}.")
```

Output

```
Enter the first number: 3
Enter the second number: 4
3 and 4 are not amicable.

=== Code Execution Successful ===
```

main.py

Share Run

```python
def aliquot_sum(n):
    sum_divisors = 0
    for i in range(1, n):
        if n % i == 0:
            sum_divisors += i
    return sum_divisors

n = int(input("Enter the number: "))
print(f"Aliquot sum of {n} is {aliquot_sum(n)}")
```

Output

```
Enter the number: 34
Aliquot sum of 34 is 20

=== Code Execution Successful ===
```

main.py

Share Run

```python
def is_abundant(n):
    sum_divisors = 0
    for i in range(1, n):
        if n % i == 0:
            sum_divisors += i
    return sum_divisors > n

n = int(input("Enter the number: "))
print(f"{n} is {'abundant' if is_abundant(n) else 'not abundant'}")
```

Output

```
Enter the number: 34
34 is not abundant

=== Code Execution Successful ===
```

main.py

Share Run

```python
def partition_function(n):
    partitions = [0] * (n + 1)
    partitions[0] = 1
    for i in range(1, n + 1):
        for j in range(i, n + 1):
            partitions[j] += partitions[j - i]
    return partitions[n]

n = int(input("Enter the number: "))
print(f"The number of partitions of {n} is {partition_function(n)}")
```

Output

```
Enter the number: 34
The number of partitions of 34 is 12310

=== Code Execution Successful ===
```

main.py

Share Run

```python
def is_palindrome(s: str) -> bool:
    return s ==0

def main():
    s = input("Enter a number: ")
    print(f"{s} is {'a' if is_palindrome(s) else 'not a'} palindrome.")

if __name__ == "__main__":
    main()
```

Output

```
Enter a number: 34
34 is not a palindrome.

=== Code Execution Successful ===
```