r/ThingsYouDidntKnow • 2 mo. ago
TheStocksGuy

Understanding Sophie Germain Primes and My Mathematical Approach

I've been working on an intriguing mathematical concept involving carried numbers and iterative multiplications to better understand and generate Sophie Germain primes. Here's a breakdown of my approach:

Step-by-Step Calculations

1. **Initial Multiplication**:
   - Start with (2 \times 3 = 6).
   - This gives us a carry value of 6, which we will use in subsequent steps.
2. **Next Multiplication**:
   - Calculate (2 \times 5 = 10).
   - Add the carried value of 1 to get (10 + 1 = 11).
3. **Doubling the Result**:
   - Multiply (11 \times 2 = 22).
   - Add the carried value of 1 to get (22 + 1 = 23).
4. **Adding the Carried Value**:
   - Add 6 to (23) to get (23 + 6 = 29).
5. **Further Multiplication and Adjustment**:
   - Add (3 \times 6) to (29) to get (29 + 18 = 47).
   - Subtract 6 from 47 to get (47 - 6 = 41).
6. **Final Adjustment**:
   - Add 12 to (41) to get (41 + 12 = 53).

Extending the Calculations

To extend this process and generate more numbers:

1. **Start from the Last Result**:
   - Begin with (53 + 6 = 59).
2. **Next Steps**:
   - Add (3 \times 6) to (59) to get (59 + 18 = 77).
   - Subtract 6 from 77 to get (77 - 6 = 71).
   - Add 6 to (71) to get (71 + 6 = 77).
   - Add (3 \times 6) to (77) to get (77 + 18 = 95).
   - Subtract 6 from 95 to get (95 - 6 = 89).
   - Finally, add 12 to (89) to get (89 + 12 = 101).

1) are both prime.

Python Implementation

Here's a Python script to automate this process and verify the primes:

```
import sympy  # Import sympy for prime checking

def is_sophie_germain_prime(p):
    return sympy.isprime(p) and sympy.isprime(2 * p + 1)

def generate_sophie_germain_primes(count, start=100):
    primes = []
    p = start  # Start from 100
    while len(primes) < count:
        if is_sophie_germain_prime(p):
            primes.append(p)
        p += 1
    return primes

# Generate Sophie Germain primes past 100
sophie_germain_primes_past_100 = generate_sophie_germain_primes(30, 100)
print(sophie_germain_primes_past_100)

# Applying the steps to each Sophie Germain prime in the list
for prime in sophie_germain_primes_past_100:
    # Example transformations
    carry_6 = prime * 3
    result_1 = prime * 5 + 1
    result_2 = result_1 * 2 + 1
    print(f"Prime: {prime}, Carried 6: {carry_6}, Result 1: {result_1}, Result 2: {result_2}")
```

Results

By following these steps, I can generate and verify Sophie Germain primes, ensuring that each transformation aligns with the properties of these special primes. Here are some results:

- **Prime: 131**
  - Carried 6: 393
  - Result 1: 656
  - Result 2: 1313
- **Prime: 173**
  - Carried 6: 519

- **Prime: 179**
  - Carried 6: 537
  - Result 1: 896
  - Result 2: 1793
- **Prime: 191**
  - Carried 6: 573
  - Result 1: 956
  - Result 2: 1913
- **Prime: 233**
  - Carried 6: 699
  - Result 1: 1166
  - Result 2: 2333

By iterating through these transformations, we can observe how carried numbers and specific multiplications help in generating primes and aligning them with Sophie Germain prime properties.