Advanced Series & Pattern Problems

Task 1: Factorial Reciprocal Series

Problem Statement: Calculate the sum of the series:

```
1 + 3/2! + 5/4! + 7/6! + ... up to n terms
```

Visualization:

- Term 1: 1
- Term 2: 3/2!
- Term 3: 5/4!
- Term 4: 7/6!
- ...

Input: An integer `n` (if n is odd, run loop till n-1)

Output: Sum of the series up to `n` terms (6 decimal places)

Sample Input:

```
5
```

Sample Output:

```
2.716667
```

Explanation:

```
1 + 3/2 + 5/24 + 7/720 ≈ 2.716667
```

Task 2: Alternating Power Reciprocal Series

Problem Statement: Calculate the sum of the series:

```
1 - 2/2^2 + 3/3^3 - 4/4^4 + ... up to n terms
```

Visualization:

- Term 1: 1
- Term 2: - 2/2^2
- Term 3: + 3/3^3
- Term 4: - 4/4^4
- ...

Input: Integer n

Output: Sum of the series (6 decimal places)

Sample Input:

```
4
```

Sample Output:

```
0.837037
```

Explanation:

```
1 - 2/4 + 3/27 - 4/256 ≈ 0.837037
```

Task 3: Factorial Power Series with x (Input from Previous Output)

Problem Statement: Use the output of Task 2 as x and calculate the sum of the series:

```
1 + 1!*x + 2!*x^2 + 3!*x^3 + ... + n!*x^n
```

Visualization:

- Term 1: 1
- Term 2: 1!*x
- Term 3: 2!*x^2
- Term 4: 3!*x^3
- ...

Input:

- x: Output of Task 2
- Integer n

Output: Sum of the series (6 decimal places)

Sample Input:

```
x = 0.837037
n = 3
```

Sample Output:

```
8.129327
```

Explanation:

```
1 + 1!*0.837037 + 2!*0.700648 + 3!*0.583301 ≈ 8.129327
```

Task 4: Alternating Factorial Series (Chained Input)

Problem Statement: Use the output of Task 3 as `n` (round to nearest integer) and calculate the sum of the series:

```
1 - 1! + 2! - 3! + 4! - ... up to n terms
```

Visualization:

- Term 1: 1
- Term 2: -1!
- Term 3: +2!
- Term 4: -3!
- Term 5: +4!
- ...

Input: Integer `n` (rounded output of Task 3)

Output: Sum of the series

Sample Input:

```
8
```

Sample Output:

```
-405
```

Explanation:

```
1 - 1 + 2 - 6 + 24 - 120 + 720 - 5040 + 40320 ≈ -405 (example)
```

Note:

- Each task's output can be used as input for the next task.
- Use `double` or `float` for precision-based outputs.
- Factorials should be computed carefully to avoid overflow.