PART 04 - 연구과제

제곱수 판별하기 (Level.0)

https://school.programmers.co.kr/learn/courses/30/lessons/120909?language=java

📌 문제 해결 개요

이 문제는 주어진 정수가 제곱수인지 판별하는 문제입니다. 반복문을 사용하여 특정 정수 `x`가 존재하여 `x * x == n`을 만족하는지 확인하는 방식으로 해결합니다.

🔷 핵심 개념

- 제곱수란 어떤 정수 `x`에 대해 `x * x == n`을 만족하는 수
- `x`를 1부터 증가시키면서 `x * x`가 `n`과 같은지 확인
- `x * x`가 `n`을 초과하면 더 이상 탐색할 필요 없음

🚀 해결 방법

1. `x`를 1부터 증가시키면서 `x * x`가 `n`과 같은지 확인합니다.
2. `x * x == n`이면 1을 반환합니다.
3. `x * x > n`이면 `n`이 제곱수가 아니므로 2를 반환합니다.
4. `x`가 `n`보다 작은 동안 반복하여 탐색합니다.

```
class Solution {
    public int solution(int n) {
        int x = 1; // 제곱수를 찾기 위한 초기값 설정

        while (x * x <= n) { // x^2이 n을 초과하지 않는 동안 반복
            if (x * x == n) { // x^2이 n과 같다면 제곱수
                return 1;
            }
            x++; // x 증가
        }

        return 2; // 제곱수를 찾지 못하면 2 반환
    }

    public static void main(String[] args) {
        Solution sol = new Solution();

        // 테스트 예제 실행
        System.out.println(sol.solution(144)); // 1 (12*12)
        System.out.println(sol.solution(976)); // 2 (제곱수가 아님)
    }
}
```

직각삼각형 출력하기 (Level.0)

https://school.programmers.co.kr/learn/courses/30/lessons/120823?language=java

📌 문제 해결 개요

이 문제는 "*"*을 이용하여 직각 이등변 삼각형을 출력하는 문제입니다.* `n`*의 크기에 따라 줄마다 증가하는* "*" 개수를 출력하는 방식으로 해결합니다.

🔷 핵심 개념

- 반복문을 사용하여 줄 수(`n`)만큼 출력
- 각 줄마다 `i+1`개의 "*" 출력 (`i`는 0부터 시작)
- 중첩된 반복문을 활용하여 "*" 문자열을 생성

🚀 해결 방법

1. `for` 문을 사용하여 1부터 `n`까지 반복합니다.
2. 각 줄마다 해당 줄 번호에 맞는 개수의 "*"를 출력합니다.
3. 출력 형식을 맞추어 삼각형 형태로 나타냅니다.

```
import java.util.Scanner;

class Solution {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt(); // 사용자 입력값 n

        // 삼각형 출력
        for (int i = 1; i <= n; i++) { // i는 1부터 n까지 반복
            for (int j = 0; j < i; j++) { // j는 0부터 i-1까지 반복하며 "*" 출력
                System.out.print("*");
            }
            System.out.println(); // 줄바꿈
        }

        scanner.close();
    }
}
```

짝수의 합 (Level.0)

https://school.programmers.co.kr/learn/courses/30/lessons/120831?language=java

📌 문제 해결 개요

이 문제는 `n` 이하의 짝수를 모두 더하는 문제입니다. 반복문을 사용하여 `2`의 배수를 찾아 누적하는 방식으로 해결합니다.

◈ 핵심 개념

- `for` 문을 활용하여 1부터 `n`까지 순회
- `if` 조건문을 사용하여 `i % 2 == 0`인 경우만 합산
- 최종적으로 짝수들의 합을 반환

🚀 해결 방법

1. 결과를 저장할 변수를 초기화합니다.
2. `for` 문을 사용하여 1부터 `n`까지 반복합니다.
3. 현재 숫자가 짝수(`i % 2 == 0`)인지 확인한 후 합산합니다.
4. 최종적으로 누적된 합을 반환합니다.

```
class Solution {
    public int solution(int n) {
        int sum = 0; // 짝수들의 합을 저장할 변수 초기화

        // 1부터 n까지 반복하며 짝수인 경우만 합산
        for (int i = 1; i <= n; i++) {
            if (i % 2 == 0) { // 짝수인지 확인
                sum += i; // 짝수일 경우 합산
            }
        }

        return sum; // 최종적으로 누적된 짝수의 합 반환
    }

    public static void main(String[] args) {
        Solution sol = new Solution();

        // 테스트 예제 실행
        System.out.println(sol.solution(10)); // 30
        System.out.println(sol.solution(4));  // 6
    }
}
```

팩토리얼 (Level.0)

https://school.programmers.co.kr/learn/courses/30/lessons/120848?language=java

📌 문제 해결 개요

이 문제는 `i! ≤ n`을 만족하는 가장 큰 정수 `i`를 찾는 문제입니다. 팩토리얼을 계산하면서 `n`을 초과하기 전의 최대 `i` 값을 찾는 방식으로 해결합니다.

🔷 핵심 개념

- 팩토리얼(`i!`)은 1부터 `i`까지의 정수 곱 (`i! = i * (i-1) * ... * 1`)
- `i!`을 계산하면서 `n`을 초과하지 않는 가장 큰 `i` 찾기
- `for` 문을 사용하여 순차적으로 팩토리얼을 계산하며 최대 `i` 탐색

🚀 해결 방법

1. `fact` 변수를 1로 초기화하고 `i`를 1부터 증가시키며 `fact`를 계산합니다.
2. `fact` 값이 `n`을 초과하면 종료하고 마지막으로 조건을 만족한 `i`를 반환합니다.

```
class Solution {
    public int solution(int n) {
        int fact = 1; // 팩토리얼 값을 저장할 변수 초기화
        int i = 1; // i값을 1부터 시작

        while (fact <= n) { // fact가 n을 초과하기 전까지 반복
            i++; // i를 증가
            fact *= i; // fact에 i를 곱하여 팩토리얼 값을 계산
        }

        return i - 1; // 초과하기 전의 최대 i 값 반환
    }

    public static void main(String[] args) {
        Solution sol = new Solution();

        // 테스트 예제 실행
        System.out.println(sol.solution(3628800)); // 10
        System.out.println(sol.solution(7)); // 3
    }
}
```

순서쌍의 개수 (Level.0)

https://school.programmers.co.kr/learn/courses/30/lessons/120836?language=java

📌 문제 해결 개요

이 문제는 자연수 `n`의 곱이 되는 순서쌍의 개수를 구하는 문제입니다. 이는 `n`의 약수의 개수를 구하는 것과 동일합니다.

🔷 핵심 개념

- `a * b = n`을 만족하는 `(a, b)`의 개수는 `n`의 약수 개수와 동일
- `for` 문을 활용하여 `n`을 나누어 떨어뜨리는 수의 개수를 카운트
- `i`가 `n`의 약수이면 `(i, n/i)` 형태의 순서쌍이 존재

🚀 해결 방법

1. 결과를 저장할 변수를 초기화합니다.
2. `1`부터 `n`까지 반복하면서 `n`을 나누어 떨어뜨리는 `i`의 개수를 세어 반환합니다.
3. 최적화를 위해 `√n`까지만 탐색하고 `(i, n/i)`를 고려하여 두 배수로 계산할 수도 있음.

```
class Solution {
    public int solution(int n) {
        int count = 0; // 순서쌍 개수를 저장할 변수 초기화

        // 1부터 n까지 반복하면서 약수 개수 카운트
        for (int i = 1; i <= n; i++) {
            if (n % i == 0) { // i가 n의 약수인지 확인
                count++; // 약수이면 순서쌍 개수 증가
            }
        }

        return count; // 최종적으로 구한 순서쌍 개수 반환
    }

    public static void main(String[] args) {
        Solution sol = new Solution();

        // 테스트 예제 실행
        System.out.println(sol.solution(20)); // 6
        System.out.println(sol.solution(100)); // 9
    }
}
```

피자 나눠 먹기 (2) (Level.0)

https://school.programmers.co.kr/learn/courses/30/lessons/120815?language=java

📌 문제 해결 개요

이 문제는 `n`명의 사람들이 피자를 남기지 않고 똑같이 나눠 먹을 수 있도록 최소한의 피자 판 수를 구하는 문제입니다. 각 판에는 6조각이 있으며, `n`명이 나눠 먹을 수 있도록 최소한의 피자 판 수를 구해야 합니다.

🔷 핵심 개념

- 피자 `answer`판을 시켰을 때, `answer * 6`이 `n`으로 나누어 떨어져야 함
- 즉, `(answer * 6) % n == 0`이 되는 최소한의 `answer`를 찾는 것이 목표
- 최소공배수(LCM, Least Common Multiple)를 활용하면 최적화 가능

🚀 해결 방법

1. `answer`를 1부터 시작하여 `6 * answer % n == 0`을 만족하는 최소값을 찾는다.
2. 이를 만족하는 `answer`를 찾을 때까지 `answer`를 증가시킨다.
3. `answer`를 반환하여 최소한으로 필요한 피자 판 수를 구한다.

```
class Solution {
    public int solution(int n) {
        int answer = 1; // 최소 피자 판 수 초기화

        // 최소한의 피자 판 수를 찾을 때까지 반복
        while ((answer * 6) % n != 0) {
            answer++; // 조건을 만족할 때까지 answer 증가
        }

        return answer; // 계산된 최소 피자 판 수 반환
    }

    public static void main(String[] args) {
        Solution sol = new Solution();

        // 테스트 예제 실행
        System.out.println(sol.solution(6));  // 1
        System.out.println(sol.solution(10)); // 5
        System.out.println(sol.solution(4));  // 2
    }
}
```

합성수 찾기 (Level.0)

📌 문제 해결 개요

자연수 n 이하의 합성수 개수를 구하는 문제다. 합성수는 약수가 3개 이상인 수를 의미한다.
각 숫자의 약수 개수를 구하여 3개 이상이면 카운트하는 방식으로 해결할 수 있다.

🔷 핵심 개념

- **약수의 개수**: 특정 수의 약수를 찾는 방법
- **반복문 활용**: 1부터 n까지 각 수의 약수 개수를 구함
- **합성수 판별**: 약수 개수가 3개 이상인지 확인

🚀 해결 방법

1. `answer` 변수를 초기화하여 합성수의 개수를 저장한다.
2. 1부터 `n`까지 모든 수에 대해 반복한다.
3. 현재 수 `i`의 약수 개수를 구하기 위해 다시 1부터 `i`까지 반복한다.
4. `i`가 `j`로 나누어 떨어지면 약수 개수를 증가시킨다.
5. 약수 개수가 3개 이상이면 합성수이므로 `answer`를 증가시키고 반복을 중단한다.
6. 최종적으로 `answer` 값을 반환하여 합성수의 개수를 얻는다.

```
public class Solution {

    // 메인 메서드 - 테스트 실행
    public static void main(String[] args) {
        System.out.println(solution(10)); // 5
        System.out.println(solution(15)); // 8
    }

    // 합성수 개수를 구하는 메서드
    public static int solution(int n) {
        // 합성수의 개수를 저장할 변수 초기화
        int answer = 0;

        // 1부터 n까지 반복하면서 각 숫자가 합성수인지 판별
        for (int i = 1; i <= n; i++) {
            // i의 약수 개수를 세는 변수 초기화
            int count = 0;

            // 1부터 i까지 반복하면서 약수 개수 확인
            for (int j = 1; j <= i; j++) {
                // i가 j로 나누어 떨어지면 약수
                if (i % j == 0) {
                    // 약수 개수 증가
                    count++;

                    // 약수의 개수가 3개 이상이면 합성수
                    if (count >= 3) {
                        // 합성수 개수 증가
```

```
                        answer++;
                        // 더 이상 검사할 필요 없으므로 반복 종료
                        break;
                    }
                }
            }
        }

        // 계산된 합성수 개수 반환
        return answer;
    }
}
```

분수의 덧셈 (Level.0)

https://school.programmers.co.kr/learn/courses/30/lessons/120808?language=java

📌 문제 해결 개요

이 문제는 두 개의 분수를 더한 후 **기약 분수** 형태로 변환하는 문제입니다.

🔷 핵심 개념

- **분수의 덧셈 공식**:

$$\frac{a}{b} + \frac{c}{d} = \frac{(a \times d) + (c \times b)}{b \times d}$$

- **기약 분수 변환**:
  - 분자와 분모의 **최대공약수(GCD)**를 구해 분수를 약분

🚀 해결 방법

1. 두 분수를 더하기 위해 공통 분모를 구하여 새로운 분자를 계산한다.
2. 분자와 분모의 최대공약수(GCD)를 구하여 기약분수 형태로 변환한다.
3. 변환된 분자와 분모를 배열로 반환한다.

```
class Solution {
    public int[] solution(int numer1, int denom1, int numer2, int denom2) {
        // 분자 계산: (a * d) + (c * b)
        int numer3 = numer1 * denom2 + numer2 * denom1;
        // 분모 계산: b * d
        int denom3 = denom1 * denom2;

        // 최대공약수(GCD) 구하기
        int gcd = gcd(numer3, denom3);

        // 기약 분수로 변환
        return new int[]{numer3 / gcd, denom3 / gcd};
    }

    // 유클리드 호제법을 이용한 최대공약수(GCD) 계산
    private int gcd(int a, int b) {
        while (b != 0) {
            int temp = b;
            b = a % b;
            a = temp;
        }
        return a;
    }

    public static void main(String[] args) {
        Solution sol = new Solution();
```

```
        // 테스트 예제 실행
        System.out.println(Arrays.toString(sol.solution(1, 2, 3, 4))); // [5,
4]
        System.out.println(Arrays.toString(sol.solution(9, 2, 1, 3))); // [29,
6]
    }
}
```