Q1:

```
// Part (a)
public static int numberOfLeapYears(int year1, int year2) {
    int count = 0;
    for (int y = year1; y <= year2; y++) {
        if (isLeapYear(y)) {
            count++;
        }
    }
    return count;
}

// Part (b)
public static int dayOfWeek(int month, int day, int year) {
            int startDay = firstDayOfYear(year);
            int nthDay = dayOfYear(month, day, year);
            int returnDay = (startDay + nthDay - 1) % 7;
            return returnDay;
}
```

Q2:

```
public class StepTracker {
    private int minSteps;
    private int totalSteps;
    private int numDays;
    private int numActiveDays;

    public StepTracker(int threshold) {
        minSteps = threshold;
        totalSteps = 0;
        numDays = 0;
        numActiveDays = 0;
    }

    public void addDailySteps(int steps) {
        totalSteps += steps;
        numDays++;
        if (steps >= minSteps) {
            numActiveDays++;
        }
    }
```

```
    public int activeDays() {
        return numActiveDays;
    }

    public double averageSteps() {
        if (numDays == 0) {
            return 0.0;
        } else {
            return (double) totalSteps / numDays;
        }
    }
}
```

Q3:

```
// Part (a)
public ArrayList<String> getDelimitersList(String[] tokens) {
    ArrayList<String> delimiters = new ArrayList<>();
    for (String str : tokens) {
        if (str.equals(openDel) || str.equals(closeDel)) {
            delimiters.add(str);
        }
```

```
    }
    return delimiters;
}

// Part (b)
public boolean isBalanced(ArrayList<String> delimiters) {
    int openCount = 0;
    int closeCount = 0;

    for (String str : delimiters) {
        if (str.equals(openDel)) {
            openCount++;
        } else {
            closeCount++;
        }

        if (closeCount > openCount) {
            return false;
        }
    }
    return openCount == closeCount;
}
```

Q4:

```
// Part (a)
public class LightBoard {
    private boolean[][] lights;

    public LightBoard(int numRows, int numCols) {
        lights = new boolean[numRows][numCols];
        for (int r = 0; r < numRows; r++) {
            for (int c = 0; c < numCols; c++) {
                double rnd = Math.random();
                lights[r][c] = rnd < 0.4;
            }
        }
    }

    // Part (b)
    public boolean evaluateLight(int row, int col) {
        int numOn = 0;
        for (int r = 0; r < lights.length; r++) {
            if (lights[r][col]) {
                numOn++;
            }
```

```
        }
        if (lights[row][col] && numOn % 2 == 0) {
            return false;
        }
        if (!lights[row][col] && numOn % 3 == 0) { // Assuming you meant % 3 == 0 based on common patterns; replace with the correct condition if needed
            return true;
        }
        return lights[row][col];
    }
}
```

```
_1:
// ____ (_)
______ ______ ___ _________________(___ ____1, ___ ____2) {
    ___ _____ = 0;
    ___ (___ _ = ____1; _ <= ____2; _++) {
        __ (__________(_)) {
            _____++;
        }
    }
    ______ _____;
}

// ____ (_)
______ ______ ___ __________(___ _____, ___ ___, ___ ____) {
            ___ ________ = ______________(____);
            ___ ______ = _________(_____, ___, ____);
            ___ __________ = (________ + ______ - 1) % 7;
            _____ _________;
}
```

```
_2:

______ _____ ___________ {

    _______ ___ ________;

    _______ ___ __________;

    _______ ___ _______;

    _______ ___ _____________;


    ______ ___________(___ _________) {

        ________ = ________;

        __________ = 0;

        _______ = 0;

        _____________ = 0;

    }


    ______ ____ _____________(___ _____) {

        __________ += _____;

        _______++;

        __ (_____ >= ________) {

            _____________++;

        }

    }
```

```
    _____ ___ __________() {
        _____ _____________;
    }

    _____ _____ _____________() {
        __ (_______ == 0) {
            _____ 0.0;
        } ____ {
            _____ (_____) __________ / _______;
        }
    }
}



_3:
// ____ (_)
_____ __________<_____> ___________________(_____[] _____) {
    _________<_____> __________ = ___ _________<>();
    ___ (_____ ___ : _____) {
        __ (___._____(_____) || ___._____(_________)) {
            __________.___(___);
        }
```

```
    }
    ______ __________;
}

// ____ (_)
______ _______ __________(_________ <______ > __________) {
    ___ _________ = 0;
    ___ __________ = 0;

    ___ (______ ___ : __________) {
        __ (___.______(_______)) {
            _________++;
        } ____ {
            __________++;
        }

        __ (___________ > _________) {
            ______ _____;
        }
    }
    ______ _________ == _________;
}
```

```
_4:

// ____ (_)

______ _____ __________ {

    _______ _______[][] ______;


    ______ __________(___ _______, ___ _______) {

        ______  = ___ _______[_______][_______];

        ___ (___ _  = 0; _  < _______; _++) {

            ___ (___ _  = 0; _  < _______; _++) {

                ______ ___  = ____.______();

                ______[_][_] = ___  < 0.4;

            }

        }

    }


    // ____ (_)

    ______ _______ ______________(___ ___, ___ ___) {

        ___ _____  = 0;

        ___ (___ _  = 0; _  < ______.______; _++) {

            __ (______[_][___]) {

                _____++;

            }
```

```
        }
        __ (_____[___][___] && _____ % 2 == 0) {
            _____ _____;
        }
        __ (!_____[___][___] && _____ % 3 == 0) { // _________ ___ _____ %
3 == 0 _____ __ _____ _________; _______ ____ ___ _______ __________ __
_____
            _____ ____;
        }
        _____ _______[___][___];
    }
}
```