AP® Computer Science A
Scoring Guidelines
Apply the question assessment rubric first, which always takes precedence. Penalty points can only be deducted in a part of the question that has earned credit via the question rubric. No part of a question (a, b, c) may have a negative point total. A given penalty can be assessed only once for a question, even if it occurs multiple times or in multiple parts of that question. A maximum of 3 penalty points may be assessed per question.

1-Point Penalty

v) Array/collection access confusion ([[] get)

w) Extraneous code that causes side-effect (e.g., printing to output, incorrect precondition check)

x) Local variables used but none declared

y) Destruction of persistent data (e.g., changing value referenced by parameter)

z) Void method or constructor that returns a value

No Penalty

o Extraneous code with no side-effect (e.g., valid precondition check, no-op)

o Spelling/case discrepancies where there is no ambiguity*

o Local variable not declared provided other variables are declared in some part

o private or public qualifier on a local variable

o Missing public qualifier on class or constructor header

o Keyword used as an identifier

o Common mathematical symbols used for operators (× • ÷ < > ≠)

o [ ] vs. () vs. <>

o = instead of == and vice versa

o length/size confusion for array, String, List, or ArrayList; with or without ()

o Extraneous [] when referencing entire array

o [i,j] instead of [i][j]

o Extraneous size in array declaration, e.g., int[size] nums = new int[size];

o Missing ; where structure clearly conveys intent

o Missing {} where indentation clearly conveys intent

o Missing () on parameter-less method or constructor invocations

o Missing () around if or while conditions

*Spelling and case discrepancies for identifiers fall under the “No Penalty” category only if the correction can be unambiguously inferred from context, for example, “ArayList” instead of “ArrayList”. As a counterexample, note that if the code declares “int G=99, g=0;”, then uses “while (G < 10)” instead of “while (g < 10)”, the context does not allow for the reader to assume the use of the lower-case variable.
Question 1: Calendar

<table>
<thead>
<tr>
<th>Part (a)</th>
<th>numberOfLeapYears</th>
<th>5 points</th>
</tr>
</thead>
</table>

**Intent:** Return the number of leap years in a range

- +1 Initializes a numeric variable
- +1 Loops through each necessary year in the range
- +1 Calls `isLeapYear` on some valid year in the range
- +1 Updates count based on result of calling `isLeapYear`
- +1 Returns count of leap years

<table>
<thead>
<tr>
<th>Part (b)</th>
<th>dayOfWeek</th>
<th>4 points</th>
</tr>
</thead>
</table>

**Intent:** Return an integer representing the day of the week for a given date

- +1 Calls `firstDayOfYear`
- +1 Calls `dayOfYear`
- +1 Calculates the value representing the day of the week
- +1 Returns the calculated value

**Question-Specific Penalties**

- -1 (t) Static methods called with `this`. 
### Question 1: Scoring Notes

<table>
<thead>
<tr>
<th>Part (a)</th>
<th>numberOfLeapYears</th>
<th>5 points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Points</td>
<td>Rubric Criteria</td>
<td>Responses earn the point even if they...</td>
</tr>
<tr>
<td>+1</td>
<td>Initializes a numeric variable</td>
<td></td>
</tr>
<tr>
<td>+1</td>
<td>Loops through each necessary year in the range</td>
<td></td>
</tr>
<tr>
<td>+1</td>
<td>Calls <code>isLeapYear</code> on some valid year in the range</td>
<td>do not use a loop</td>
</tr>
<tr>
<td>+1</td>
<td>Updates count based on result of calling <code>isLeapYear</code></td>
<td>do not use a loop</td>
</tr>
<tr>
<td>+1</td>
<td>Returns count of leap years</td>
<td>loop from <code>year1</code> to <code>year2</code> incorrectly</td>
</tr>
<tr>
<td></td>
<td></td>
<td>do not initialize the counter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>do not use a loop</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Part (b)</th>
<th>dayOfWeek</th>
<th>4 points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Points</td>
<td>Rubric Criteria</td>
<td>Responses earn the point even if they...</td>
</tr>
<tr>
<td>+1</td>
<td>Calls <code>firstDayOfYear</code></td>
<td></td>
</tr>
<tr>
<td>+1</td>
<td>Calls <code>dayOfYear</code></td>
<td></td>
</tr>
<tr>
<td>+1</td>
<td>Calculates the value representing the day of the week</td>
<td></td>
</tr>
<tr>
<td>+1</td>
<td>Returns the calculated value</td>
<td>return the value from calling <code>firstDayOfYear</code> or <code>dayOfYear</code></td>
</tr>
</tbody>
</table>
Question 1: Calendar

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;
}
### Question 2: Step Tracker

<table>
<thead>
<tr>
<th>Class:</th>
<th>StepTracker</th>
</tr>
</thead>
</table>

**Intent:** Define implementation of a class to record fitness data

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>+1</strong></td>
<td>Declares all appropriate <code>private</code> instance variables</td>
</tr>
<tr>
<td><strong>+2</strong></td>
<td>Constructor</td>
</tr>
<tr>
<td></td>
<td><strong>+1</strong> Declares header: <code>public StepTracker(int ___)</code></td>
</tr>
<tr>
<td></td>
<td><strong>+1</strong> Uses parameter and appropriate values to initialize instance variables</td>
</tr>
<tr>
<td><strong>+3</strong></td>
<td><code>addDailySteps</code> method</td>
</tr>
<tr>
<td></td>
<td><strong>+1</strong> Declares header: <code>public void addDailySteps(int ___)</code></td>
</tr>
<tr>
<td></td>
<td><strong>+1</strong> Identifies active days and increments count</td>
</tr>
<tr>
<td></td>
<td><strong>+1</strong> Updates other instance variables appropriately</td>
</tr>
<tr>
<td><strong>+1</strong></td>
<td><code>activeDays</code> method</td>
</tr>
<tr>
<td></td>
<td><strong>+1</strong> Declares and implements <code>public int activeDays()</code></td>
</tr>
<tr>
<td><strong>+2</strong></td>
<td><code>averageSteps</code> method</td>
</tr>
<tr>
<td></td>
<td><strong>+1</strong> Declares header: <code>public double averageSteps()</code></td>
</tr>
<tr>
<td></td>
<td><strong>+1</strong> Returns calculated <code>double average number of steps</code></td>
</tr>
</tbody>
</table>
## Question 2: Scoring Notes

<table>
<thead>
<tr>
<th>Class</th>
<th>StepTracker</th>
</tr>
</thead>
<tbody>
<tr>
<td>Points</td>
<td>Rubric Criteria</td>
</tr>
</tbody>
</table>
| +1 | Declares all appropriate private instance variables | | • omit keyword `private`  
• declare variables outside the class |
| +2 | Constructor | | |
| +1 | Declares header:  
`public StepTracker(int ___)` | • omit keyword `public` | • declare method `private` |
| +1 | Uses parameter and appropriate values to initialize instance variables | • initialize primitive instance variables to default values when declared | • fail to use the parameter to initialize some instance variable  
• fail to declare instance variables  
• initialize local variables instead of instance variables  
• assign variables to parameters |
| +3 | `addDailySteps` method | | |
| +1 | Declares header:  
`public void addDailySteps(int ___)` | • omit keyword `public` | • declare method `private` |
| +1 | Identifies active days and increments count | • put valid comparison erroneously in some other method | • fail to use the parameter as part of the comparison  
• fail to increment a count of active days  
• fail to increment an instance variable  
• compare parameter to some numeric constant |
| +1 | Updates other instance variables appropriately | | • update another instance variable only on active days  
• update another instance variable inappropriately  
• fail to update appropriate instance variable  
• update a local variable |
| +1 | `activeDays` method | | |
| +1 | Declares and implements `public int activeDays()` | • return appropriate count of active days where the instance variables were updated improperly in `addDailySteps` or `activeDays` | • declare method `private`  
• return value that is not the number of active days  
• fail to return a value |
### Question 2: Scoring Notes (continued)

<table>
<thead>
<tr>
<th>Points</th>
<th>Rubric Criteria</th>
<th>Responses earn the point even if they...</th>
<th>Responses will not earn the point if they...</th>
</tr>
</thead>
<tbody>
<tr>
<td>+2</td>
<td>averageSteps method</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+1</td>
<td>Declares header: public double averageSteps()</td>
<td>• omit keyword public</td>
<td>• declare method private</td>
</tr>
</tbody>
</table>
| +1     | Returns calculated double average number of steps | • maintain instance variables improperly but calculate appropriate average | • use integer division
|        |                  | • fail to handle the special case where no days are tracked | • calculate something other than steps divided by days
|        |                  |                                           | • fail to return |
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;
        }
    }
}

These canonical solutions serve an expository role, depicting general approaches to solution. Each reflects only one instance from the infinite set of valid solutions. The solutions are presented in a coding style chosen to enhance readability and facilitate understanding.
Question 3: Delimiters

<table>
<thead>
<tr>
<th>Part (a)</th>
<th>getDelimitersList</th>
<th>4 points</th>
</tr>
</thead>
</table>

**Intent:** Store delimiters from an array in an ArrayList

+1 Creates ArrayList<String>

+1 Accesses all elements in array tokens (no bounds errors)

+1 Compares strings in tokens with both instance variables (must be in the context of a loop)

+1 Adds delimiters into ArrayList in original order

<table>
<thead>
<tr>
<th>Part (b)</th>
<th>isBalanced</th>
<th>5 points</th>
</tr>
</thead>
</table>

**Intent:** Determine whether open and close delimiters in an ArrayList are balanced

+1 Initializes accumulator(s)

+1 Accesses all elements in ArrayList delimiters (no bounds errors)

+1 Compares strings in delimiters with instance variables and updates accumulator(s) accordingly

+1 Identifies and returns appropriate boolean value to implement one rule

+1 Identifies and returns appropriate boolean values for all cases
# Question 3: Scoring Notes

<table>
<thead>
<tr>
<th>Part (a)</th>
<th>getDelimitersList</th>
<th>4 points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Points</td>
<td>Rubric Criteria</td>
<td>Responses earn the point even if they...</td>
</tr>
<tr>
<td>+1</td>
<td>Creates ArrayList&lt;String&gt;</td>
<td>• omit &lt;String&gt;</td>
</tr>
<tr>
<td>+1</td>
<td>Accesses all elements in array <code>tokens</code> (no bounds errors)</td>
<td>• return incorrectly inside the loop</td>
</tr>
<tr>
<td>+1</td>
<td>Compares strings in <code>tokens</code> with both instance variables (must be in the context of a loop)</td>
<td>• access elements of <code>tokens</code> as if from an ArrayList (e.g., <code>tokens.get(i)</code>)</td>
</tr>
<tr>
<td>+1</td>
<td>Adds delimiters into ArrayList in original order</td>
<td>• add a delimiter by accessing <code>tokens</code> incorrectly (e.g., <code>tokens.get(i)</code>)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Part (b)</th>
<th>isBalanced</th>
<th>5 points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Points</td>
<td>Rubric Criteria</td>
<td>Responses earn the point even if they...</td>
</tr>
<tr>
<td>+1</td>
<td>Initializes accumulator(s)</td>
<td>• initialize inside the loop</td>
</tr>
<tr>
<td>+1</td>
<td>Accesses all elements in <code>ArrayList delimiters</code> (no bounds errors)</td>
<td>• return incorrectly inside the loop</td>
</tr>
<tr>
<td>+1</td>
<td>Compares strings in <code>delimiters</code> with instance variables and updates accumulator(s) accordingly</td>
<td>• access elements of <code>delimiters</code> as if from an array (e.g., <code>delimiters[i]</code>)</td>
</tr>
<tr>
<td></td>
<td>Identifies and returns appropriate boolean value to implement one rule</td>
<td>• check for more closing delimiters (inside a loop) and return <code>false</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• return <code>true</code> if the number of open and close delimiters is the same, and <code>false</code> otherwise (after a loop)</td>
</tr>
<tr>
<td>+1</td>
<td>Identifies and returns appropriate boolean values for all cases</td>
<td>• have correct logic with the exception of a loop bounds error, accessing elements as if from an array, or using <code>==</code> for string comparison</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• fail to check for more closing delimiters inside a loop</td>
</tr>
</tbody>
</table>
**AP® COMPUTER SCIENCE A**  
**2019 SCORING GUIDELINES**

**Question 3: Delimiters**

**Part (a)**

```java
public ArrayList<String> getDelimitersList(String[] tokens)
{
    ArrayList<String> d = new ArrayList<String>();
    for (String str : tokens)
    {
        if (str.equals(openDel) || str.equals(closeDel))
        {
            d.add(str);
        }
    }
    return d;
}
```

**Part (b)**

```java
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;
    }
    if (openCount == closeCount)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

These canonical solutions serve an expository role, depicting general approaches to solution. Each reflects only one instance from the infinite set of valid solutions. The solutions are presented in a coding style chosen to enhance readability and facilitate understanding.

© 2019 The College Board.  
Visit the College Board on the web: collegeboard.org.
### Question 4: Light Board

**Part (a) LightBoard 4 points**

**Intent:** Define implementation of a constructor that initializes a 2D array of lights

+1 Creates a new boolean[numRows][numCols] and assigns to instance variable lights

+1 Accesses all elements in the created 2D array (no bounds errors)

+1 Computes the 40% probability

+1 Sets all values of 2D array based on computed probability

**Part (b) evaluateLight 5 points**

**Intent:** Evaluate the status of a light in a 2D array of lights

+1 Accesses an element of lights as a boolean value in an expression

+1 Traverses specified col of a 2D array (no bounds errors)

+1 Counts the number of true values in the traversal

+1 Performs an even calculation and a multiple of three calculation

+1 Returns true or false according to all three rules

---

**Question-Specific Penalties**

-1 (z) Constructor returns a value

-1 (y) Destruction of persistent data
### AP® COMPUTER SCIENCE A
#### 2019 SCORING GUIDELINES

**Question 4: Scoring Notes**

<table>
<thead>
<tr>
<th>Part (a)</th>
<th>LightBoard</th>
<th><strong>4 points</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Points</td>
<td>Rubric Criteria</td>
<td>Responses earn the point even if they...</td>
</tr>
<tr>
<td>+1</td>
<td>Creates a new boolean[numRows] [numCols] and assigns to instance variable lights</td>
<td></td>
</tr>
<tr>
<td>+1</td>
<td>Accesses all elements in the created 2D array <em>(no bounds errors)</em></td>
<td>• fail to create lights but assume lights[numRows][numCols]</td>
</tr>
<tr>
<td>+1</td>
<td>Computes the 40% probability</td>
<td>• use Math.random() (\leq .4)</td>
</tr>
<tr>
<td>+1</td>
<td>Sets all values of 2D array based on computed probability</td>
<td>• only assign true values</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Part (b)</th>
<th>evaluateLight</th>
<th><strong>5 points</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Points</td>
<td>Rubric Criteria</td>
<td>Responses earn the point even if they...</td>
</tr>
<tr>
<td>+1</td>
<td>Accesses an element of lights as a boolean value in an expression</td>
<td></td>
</tr>
<tr>
<td>+1</td>
<td>Traverses specified col of a 2D array <em>(no bounds errors)</em></td>
<td></td>
</tr>
<tr>
<td>+1</td>
<td>Counts the number of true values in the traversal</td>
<td>• access too many or too few items in a single column • access a single row instead of a single column</td>
</tr>
<tr>
<td>+1</td>
<td>Performs an even calculation and a multiple of three calculation</td>
<td></td>
</tr>
<tr>
<td>+1</td>
<td>Returns true or false according to all three rules</td>
<td>• have an incorrect column count but use the correct logic</td>
</tr>
</tbody>
</table>

© 2019 The College Board.
Visit the College Board on the web: collegeboard.org.
Question 4: Light Board

Part (a)

```java
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)

```java
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)
    {
        return true;
    }

    return lights[row][col];
}
```

These canonical solutions serve an expository role, depicting general approaches to solution. Each reflects only one instance from the infinite set of valid solutions. The solutions are presented in a coding style chosen to enhance readability and facilitate understanding.