
AP[®] Computer Science A 2016 Scoring Guidelines

© 2016 The College Board. College Board, Advanced Placement Program, AP, AP Central, and the acorn logo are registered trademarks of the College Board.

Visit the College Board on the Web: www.collegeboard.org.

AP Central is the official online home for the AP Program: apcentral.collegeboard.org.

2016 GENERAL 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., writing to output, failure to compile)
- 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., 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 counter example, note that if the code declares “Bug bug;”, then uses “Bug.move()” instead of “bug.move()”, the context does **not** allow for the reader to assume the object instead of the class.*

2016 SCORING GUIDELINES

Question 1: Random String Chooser

Part (a)	RandomStringChooser	7 points
-----------------	---------------------	-----------------

Intent: *Define implementation of class to choose a random string*

- +1** Uses correct class, constructor, and method headers
- +1** Declares appropriate `private` instance variable(s)
- +1** Initializes all instance variable(s) (*point lost if parameter not used in any initialization*)
- +4** Implements `getNext`
 - +1** Generates a random number in the proper range (*point lost for improper or missing cast*)
 - +1** Chooses a string from instance variable using generated random number
 - +1** Updates state appropriately (*point lost if constructor parameter is altered*)
 - +1** Returns chosen string or `"NONE"` as appropriate

Part (b)	RandomLetterChooser	2 points
-----------------	---------------------	-----------------

Intent: *Define implementation of a constructor of a class that extends `RandomStringChooser`*

- +1** `getSingleLetters(str)`
- +1** `super(getSingleLetters(str));` (*point lost if not first statement in constructor*)

2016 CANONICAL SOLUTIONS**Question 1: Random String Chooser**

Part (a):

```
public class RandomStringChooser
{
    private List<String> words;

    public RandomStringChooser(String[] wordArray)
    {
        words = new ArrayList<String>();

        for (String singleWord : wordArray)
        {
            words.add(singleWord);
        }
    }

    public String getNext()
    {
        if (words.size() > 0)
        {
            return words.remove((int) (Math.random() * words.size()));
        }
        return "NONE";
    }
}
```

Part (b):

```
public RandomLetterChooser(String str)
{
    super(getSingleLetters(str));
}
```

2016 SCORING GUIDELINES

Question 2: Log Messages

Part (a) <code>LogMessage</code> constructor	2 points
---	-----------------

Intent: *Initialize instance variables using passed parameter*

- +1 Locates colon
- +1 Initializes instance variables with correct parts of the parameter

Part (b) <code>containsWord</code>	2 points
---	-----------------

Intent: *Determine whether description properly contains a keyword*

- +1 Identifies at least one properly-contained occurrence of `keyword` in `description`
- +1 Returns `true` if and only if `description` properly contains `keyword`
Returns `false` otherwise (*no bounds errors*)

Part (c) <code>removeMessages</code>	5 points
---	-----------------

Intent: *Remove log messages containing keyword from system log list and return these messages in a new list*

- +1 Accesses all items in `messageList` (*no bounds errors; point lost if no removal attempted*)
- +1 Identifies keyword-containing entry using `containsWord`
- +1 Adds all and only identified entries to new list (*point lost if original order not maintained*)
- +1 Removes all identified entries from `messageList` (*point lost if `messageList` reordered*)
- +1 Constructs and returns new `ArrayList<LogMessage>`

2016 CANONICAL SOLUTIONS

Question 2: Log Messages

Part (a):

```
public LogMessage(String message)
{
    int colon = message.indexOf(":");
    machineId = message.substring(0, colon);
    description = message.substring(colon + 1);
}
```

Part (b):

```
public boolean containsWord(String keyword)
{
    if (description.equals(keyword))
    {
        return true;
    }
    if (description.indexOf(keyword + " ") == 0)
    {
        return true;
    }
    if (description.indexOf(" " + keyword + " ") != -1)
    {
        return true;
    }
    if (description.length() > keyword.length())
    {
        if ((description.substring(description.length() -
                                   keyword.length() - 1).equals(
                                   " " + keyword)))
        {
            return true;
        }
    }
    return false;
}
```

Part (c):

```
public List<LogMessage> removeMessages(String keyword)
{
    List<LogMessage> removals = new ArrayList<LogMessage>();

    for (int i = 0; i < messageList.size(); i++)
    {
        if (messageList.get(i).containsWord(keyword))
        {
            removals.add(messageList.remove(i));
            i--;
        }
    }
    return removals;
}
```

2016 SCORING GUIDELINES

Question 3: Crossword

Part (a)	<code>toBeLabeled</code>	3 points
-----------------	--------------------------	-----------------

Intent: Return a *boolean* value indicating whether a crossword grid square should be labeled with a positive number

- +1** Checks `blackSquares[r][c]`
- +1** Checks for black square/border above and black square/border to the left (*no bounds errors*)
- +1** Returns `true` if square should be labeled with positive number; returns `false` otherwise

Part (b)	Crossword constructor	6 points
-----------------	-----------------------	-----------------

Intent: Initialize each square in a crossword puzzle grid to have a *color* (*boolean*) and an *integer label*

- +1** `puzzle = new Square[blackSquares.length][blackSquares[0].length];`
(or equivalent)
- +1** Accesses all locations in `puzzle` (*no bounds errors*)
- +1** Calls `toBeLabeled` with appropriate parameters
- +1** Creates and assigns new `Square` to location in `puzzle`
- +1** Numbers identified squares consecutively, in row-major order, starting at 1
- +1** On exit: All squares in `puzzle` have correct color and number (*minor errors covered in previous points ok*)

Question-Specific Penalties

- 2** (p) Consistently uses incorrect name instead of `puzzle`
- 1** (q) Uses `array[].length` instead of `array[num].length`

2016 CANONICAL SOLUTIONS**Question 3: Crossword**

Part (a):

```
private boolean toBeLabeled(int r, int c, boolean[][] blackSquares)
{
    return (!(blackSquares[r][c]) &&
            (r == 0 || c == 0 || blackSquares[r - 1][c] ||
             blackSquares[r][c - 1]));
}
```

Part (b):

```
public Crossword(boolean[][] blackSquares)
{
    puzzle = new Square[blackSquares.length][blackSquares[0].length];
    int num = 1;

    for (int r = 0; r < blackSquares.length; r++)
    {
        for (int c = 0; c < blackSquares[0].length; c++)
        {
            if (blackSquares[r][c])
            {
                puzzle[r][c] = new Square(true, 0);
            }
            else
            {
                if (toBeLabeled(r, c, blackSquares))
                {
                    puzzle[r][c] = new Square(false, num);
                    num++;
                }
                else
                {
                    puzzle[r][c] = new Square(false, 0);
                }
            }
        }
    }
}
```


2016 SCORING GUIDELINES

Question 4: String Formatter

Part (a)	<code>totalLetters</code>	2 points
-----------------	---------------------------	-----------------

Intent: Calculate the total number of letters in a list of words

- +1** Accesses all strings in `wordList` and adds length of each to accumulated count
(no bounds errors)
- +1** Initializes and returns accumulated count

Part (b)	<code>basicGapWidth</code>	2 points
-----------------	----------------------------	-----------------

Intent: Calculate the minimum number of spaces (basic gap width) to be placed between each word in the formatted string

- +1** Calls `totalLetters` correctly and uses result
- +1** Returns correct calculated value

Part (c)	<code>format</code>	5 points
-----------------	---------------------	-----------------

Intent: Return a formatted string consisting of words from `wordList` separated by one or more spaces

- +1** Calls `basicGapWidth` and `leftoverSpaces` correctly and uses results
- +1** Adds all strings in `wordList` to formatted string in original order (no bounds errors)
- +1** Inserts *basicGapWidth* spaces between each pair of words in formatted string
- +1** Inserts one space between first *leftoverSpaces* pairs of words in formatted string
- +1** Initializes and returns formatted string (no extra or deleted characters)

2016 CANONICAL SOLUTIONS**Question 4: String Formatter**

Part (a):

```
public static int totalLetters(List<String> wordList)
{
    int total = 0;

    for (String word : wordList)
    {
        total += word.length();
    }
    return total;
}
```

Part (b):

```
public static int basicGapWidth(List<String> wordList, int formattedLen)
{
    return (formattedLen - totalLetters(wordList)) / (wordList.size()-1);
}
```

Part (c):

```
public static String format(List<String> wordList, int formattedLen)
{
    String formatted = "";
    int gapWidth = basicGapWidth(wordList, formattedLen);
    int leftovers = leftoverSpaces(wordList, formattedLen);

    for (int w = 0; w < wordList.size() - 1; w++)
    {
        formatted = formatted + wordList.get(w);
        for (int i = 0; i < gapWidth; i++)
        {
            formatted = formatted + " ";
        }
        if (leftovers > 0)
        {
            formatted = formatted + " ";
            leftovers--;
        }
    }
    formatted = formatted + wordList.get(wordList.size() - 1);

    return formatted;
}
```