Purpose
This instrument is a rubric for coding App Inventor projects. It was developed to serve for a number of purposes. Most importantly, it is
designed to characterize an individual or team student project to illustrate which “mobile computational thinking” (M-CT) ideas are embedded
in the project, and with what degree of sophistication.

Additionally, the rubric may be used to:

- Characterize the extent/type of M-CT in a classroom set of such projects
- Code projects in a database of pedagogical reference projects
- Be a tool for teachers to see different M-CT ideas and how to build towards them
- Be a tool for students to work towards developing better understanding of M-CT ideas

Introduction
The rubric is based on 14 separate and orthogonal features of projects. These features encompass an app’s visual appearance, behaviors,
and underlying implementation (coding). These 14 computational thinking features are a result of combining concepts normally involved in the
programming aspects of computational thinking (e.g. as represented by the computing practice & programming strand of the CSTA K-12
standards) and adding related concepts that are present in mobile computing —e.g., screen design, event-based programming,
location-awareness, and persistent and shared data.

Usage
The rubric assumes that the person doing the scoring has background in computer science. Some of the rows describe advanced ideas that
would only be meaningful to someone with a priori knowledge in the field, or knowledge acquired as a result of successfully completing a CS
Principles-style intro course.

For each of the 15 M-CT features, the rubric provides three orthogonal ways of scoring: (1) What computational ideas are represented in the
project code itself, (2) their appropriateness given the project’s goals and the teacher’s instructional goals, and (3) the correctness of these
ideas. The second and third characteristics must be interpreted with an understanding of the broader instructional context within which the
work was done.
For each feature, up to four categories are provided for characterizing its computational properties. These categories range from “feature is not present” (or “feature is minimally present,” for features that simply must exist given App Inventor) to a description of what constitutes advanced usage of the feature.

After its computational properties are scored, the rater is invited to judge the appropriateness and correctness of the design. These judgments can only be made with the use of broader knowledge about the student’s developmental level, the instructional context, and the particular assignment for which the app being scored was developed. Rater should have this context (e.g., by being the course instructor) in order to make these judgments.

For example, consider the Screen Interface feature. All apps must have a screen, so the minimally-present category is described as *Single screen with five or fewer visual components*. The adjacent category is described as *Single screen with more than five visual components*. Both of these categories describe essentially static interfaces. The third and fourth categories then describe two degrees of complexity of dynamic interfaces: *Single screen with more than five visual components, some of which programmatically change state based on user interaction with the app* and *Two or more screens; screens may be implemented as screen components, or by programmatically changing visibility of groups of visual components*.

After characterizing the properties of the implementation, the rater should judge its appropriateness and correctness. The Screen Interface feature may be characterized as (appropriateness) too simple, just right, and too complex and (correctness) broken (e.g.; buttons not connected to code; interface crashes in some cases), almost works (minor UI problems), and correct.

**Notes:**

Algorithms is removed from this version because (a) we could not find a reasonable definition for them, but more importantly (b) we realized that many of the other dimensions (events, loops, lists, sharing) capture the sophistication of the algorithms embedded in the code.
App Title:

Rater:

Course:

Name(s) of Programmer(s):
(If this was a group project list the members of the group.)

Brief description of what the app is intended to do:

Institution:

Instructor(s):
<table>
<thead>
<tr>
<th>1. Screen Interface</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Screen Interface</td>
<td>Single screen with five or fewer visual components that do not programmatically change state.</td>
<td>Single screen with more than five visual components that do not programmatically change state.</td>
<td>Single screen, where some components programmatically change state based on user interaction with the app.</td>
<td>Two or more screens; screens may be implemented as screen components, or by programmatically changing visibility of groups of visual components.</td>
</tr>
<tr>
<td>Appropriateness:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Correctness of UI implementation:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Naming: Components Variables Procedures</td>
<td>Few or no names were changed from their defaults.</td>
<td>Approximately half of names have been changed from their defaults.</td>
<td>Nearly all of names have been changed from their defaults.</td>
<td>Appropriateness:</td>
</tr>
<tr>
<td>Appropriateness:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Names are descriptive of functionality:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Events</td>
<td>Fewer than two types of event handlers. (Multiple buttons, all with “buttonX.onClick”, are of the same type.)</td>
<td>Two or more types of event handlers. If an event handler modifies label state or sprite position, it’s still in this category.</td>
<td>One event handler modifies state in a way that will change the opportunity for other event handlers to begin (“interacting event handlers”). Such as: enabling a clock; hiding or revealing a sprite.</td>
<td></td>
</tr>
<tr>
<td>Appropriateness:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Correctness:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. Procedural Abstraction</td>
<td>There are no procedures.</td>
<td>There is exactly one procedure, and it is called.</td>
<td>There is more than one procedure: either for code organization (naming chunks of code that are only called once), or code re-use (subroutines used in multiple places), but not both.</td>
<td>There are both procedures for code organization and code re-use.</td>
</tr>
<tr>
<td>Appropriateness:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Correctness:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| 5. Globals with Variables or Text Labels | No data abstraction with globals. Values are hard-coded. | Variables and/or text labels provide names to data, whose values may change as the app runs, or may not (equivalent to a named constant). | Appropriateness: [ ] too simple [ ] just right [ ] too complex  
Correctness: [ ] broken [ ] almost works [ ] correct |
| 6. Component Abstraction | App does not use component abstraction. | App modifies or reads properties of components out of a list using an “any component” block. | Appropriateness: [ ] too simple [ ] just right [ ] too complex  
Correctness: [ ] broken [ ] almost works [ ] correct |
| 7. Loops | No use of while block, for-each block, or for-range block | Simple loop, using a constant-value control values. | Loop is governed by data that may change, dynamic. | Loop uses control values that connect data together, across multiple lists, or addressing multiple structures. | Appropriateness: [ ] too simple [ ] just right [ ] too complex  
Correctness: [ ] broken [ ] almost works [ ] correct |
| 8. Conditionals | No conditionals. | Conditionals use comparison of a variable value to a constant value. | Conditionals use comparison of two variable values. | Appropriateness: [ ] too simple [ ] just right [ ] too complex  
Correctness: [ ] broken [ ] almost works [ ] correct |
| 9. Lists | No lists. | One single-dimensional list. | More than one independent, single-dimensional lists. | A list of tuples, or (equivalently) multiple corresponding lists (a “multi-dimensional list”). | Appropriateness: [ ] too simple [ ] just right [ ] too complex  
Correctness: [ ] broken [ ] almost works [ ] correct |
| 10. Data Persistence | Data are only stored in variables or UI component properties, and do not persist when app is closed. | Data persist beyond a single session of the app. (look for: tinydb, tinywebdb) | | Appropriateness: [ ] too simple [ ] just right [ ] too complex  
Correctness: [ ] broken [ ] almost works [ ] correct |
### 11. Data Sharing

| No data sharing. | Shared data are limited to a single piece of information (such as a high score or address). | Shared data are compound structures (such as a high score with name). | Multiple users of the app read and write the same shared pool of data. |

| Appropriateness: [ ] too simple [ ] just right [ ] too complex |
| Correctness: [ ] broken [ ] almost works [ ] correct |

### 12. Public Web Services

| No web services. | Reads data directly from online data source. | Reads and writes online data source. |

| Appropriateness: [ ] too simple [ ] just right [ ] too complex |
| Correctness: [ ] broken [ ] almost works [ ] correct |

### 13. Accelerometer & Orientation Sensors

| No sensors used. | Accelerometer Shake gesture used to trigger events. | App makes decisions based on sensor data (e.g. controls a sprite). |

| Appropriateness: [ ] too simple [ ] just right [ ] too complex |
| Correctness: [ ] broken [ ] almost works [ ] correct |

### 14. Location Awareness

| No location used. | Accesses location and immediately passes it to built-in features (such as maps) | Accesses location and stores it for later retrieval and use. | Inspects location data numerically, processes this data as a feature. |

| Appropriateness: [ ] too simple [ ] just right [ ] too complex |
| Correctness: [ ] broken [ ] almost works [ ] correct |

### Overall assessment of App:

If this app has notable qualities or contains a good example of some aspect of Mobile Computational thinking, please describe it briefly:
To what extent did the app fulfill its intended purpose?

How would you rate the aesthetics of the app? How much effort was applied to the visual experience?

Additional comments: