Codedle() Guidelines

The following are a list of rules/guidelines that all daily code segments follow in order to ensure a more balanced and fair game. The following rules can serve 2 purposes: helping players determine how code is structured to optimize their guesses or for helping code contributors determine how their code should be submitted.

- "Errors/Exceptions": there should **not** be any explicit errors/exceptions or a language's equivalent in any of the code segments for any day, but if there is a possibility for an error and it is not covered, it is fine. If you think there might be an error for a day's code, please fill out this form
  - Example: this Java code is **not acceptable**:
    `int num= 5/0;`
  - Example: this Java code is **acceptable** even if a division by zero exception can occur:
    `int input= 5/scanner.nextInt();`

- "Scope": the scope of any day's code segment can vary. Therefore, a variable may or may not be declared within the code segment because any scope is allowed.
  - Example: Let's say we have the following Java code:

```
public class Student {
      private int gradeLevel;
      private int[] grades;
      public double calcAvgGrade(){
            int tot=0;
            for (int grade : grades){
                  tot+=grade;
            }
            return (double)tot/grades.length;
      }
}
```

    In the example above, both the `Student` class as well as the `calcAvgGrade()` code could appear as a code segment.

    This means if `calcAvgGrade()` was used as the daily code segment, `grades` would not be declared within the given code segment for the day, which would not break any rules.

- "Naming Scheme": the naming scheme of variables and members might **not always** follow the typical naming conventions of a language, so that should not narrow down your guess. However, it is **highly encouraged** for contributors to submit code that does follow the language's naming scheme to reduce confusion.
    - Example. `int level_count` in Java or `let HtmlResult` in JavaScript would not break any rules

- "Language Feel": daily code segments should match the "feel" of the target programming language as best as possible. That does not mean, however, that features of a language that do not match its "feel" might not appear.
    - Example: C# code should not use get() and set() methods like Java since C# has properties that allow you to do: `public int VarName {get; private set;}`

    - Example: you can use `printf()` in C++ code, but should be avoided most of the time to maintain the C++ "feel" with `std::cout`

- "Language Use": daily code segments are **not always** limited to the typical use cases of a programming language and it is **highly encouraged** to submit interesting and varied segments that do not fall into the typical use cases.
    - Example: Web pages with Python

- "Identification": each code segment must have a feature that identifies it as the target language and no other with at **least 1** part of the code that exists to that language only. This means each segment should have either a keyword, language feature or other notable aspect that only that language has. It does not mean it can not have common features.
    - Example: the following code can be applicable to C, C++, C#, Java (and probably others too)

```
int count=0;
for (int i=0; i<5; i++)
{
      if (i%2==0)
      {
            count+=i;
      }
}
```

- "Feature Support": any feature for a programming language may appear, whether it is widely used or not. It does not matter if a language still supports a feature or not, it can be used as long as that feature works in the version used for the code segment. The entire code segment should be written in one consistent version.
    - Example: using C++ `nullptr` feature from C++11 or using the older `NULL` for null pointers are **both acceptable**

- "Implementation": daily code segments might **not always** have the best time complexity, space complexity or may have the most optimal way of achieving the task.
    - Example: C implementation of Bubble Sort does not break any rules

- "Style": the color coding for a language may **not** be consistent and therefore should not be used to determine a guess for a language.
    - Example: C# `using` keyword should be blue, but is pink for a daily code segment.