Exercise Wednesday

Introduction to Programming (/introduction-to-programming) / Arrays and Looping (/introduction-to-programming/arrays-and-looping) / Practice: Pig Latin

Text

Goal: Focus on breaking your project down into small pieces of functionality, and tackling them one at a time with TDD. Also, as our projects grow in size, continue to practice good organization by keeping business logic and user interface logic distinctly separate.

Warm Up

Let's say we have an application that counts the number of vowels in a string. This application has tests detailing each piece of functionality it must exhibit. However, they're out of order! Reorder the following list of tests from the *simplest possible* functionality to the *most complex* functionality with your partner. Consult with other pairs if necessary.

As you work through this warm up, keep in mind that there could be multiple starting points for the *simplest possible piece of functionality*. There could also be multiple next best steps. Don't get hung up on being right and instead discuss the reasons why you think certain pieces of functionality are more simple than others.

```
Describe: vowelCounter();

Test: "It recognizes vowels in a multiple word sentence reg
ardless of capitalization."
Code: vowelCounter("CATS CATERED THE EVENT");
Expected Output: 7

Test: "It recognizes a single vowel in a word with multiple
characters."
Code: vowelCounter("cat");
Expected Output: 1

Test: "It recognizes a single vowel."
Code: vowelCounter("a");
Expected Output: 1

Test: "It recognizes multiple vowels in a single word."
Code: vowelCounter("cater");
Expected Output: 2

Test: "It recognizes a single vowel regardless of case."
Code: vowelCounter("A");
Expected Output: 1

Test: "It recognizes all vowels in a multiple-word sentence
regardless of inconsistent capitalization."
Code: vowelCounter("CaTS CATEReD ThE EveNT");
Expected Output: 7

Test: "It ignores non-alphabetical characters since they ca
n't be vowels."
Code: vowelCounter("*&$92%");
Expected Output: 0

Test: "It recognizes vowels in a multiple-word sentence."
Code: vowelCounter("cats catered the event");
Expected Output: 7
```

Code

Start by completing the Pig Latin project. If you have additional time, then move on to complete either of the further exploration projects.

Pig Latin

Write a Pig Latin translator or should we say an *"igPay atinLay anslatorTray"*? **Read all instructions carefully before beginning.**

Also note that it's normal to not complete this project. Remember that our goal is always understanding, and not the completion of a project. As long as you practice breaking this project down into small pieces of functionality and tackling them one at a time with TDD, your work on this project is a success!

How Pig Latin Works

Pig Latin is a language game that involves adding imaginary endings to real English words. In this prompt, you'll create an application that turns words into Pig Latin.

Here are the rules of Pig Latin:

- For words beginning with a vowel, add "way" to the end. For Pig Latin, vowels are "a," "e," "i," "o," and "u." Don't treat "y" as a vowel. **Examples:** "away" becomes "awayway" and "okay" becomes "okayway."
- For words beginning with one or more consonants, move all of the first consecutive consonants to the end and add "ay". **Examples:** "code" becomes "odecay" and "move" becomes "ovemay."
- If the first consonants include "qu", move the "q" *and* the "u." Don't forget about words like "squeal" where "qu" doesn't come first! **Examples:** "quick" becomes "ickquay" while "squeal" becomes "quealsay."

Start by Getting Organized

We'll use TDD to create the Pig Latin functionality. With TDD, we typically work with one piece of functionality at a time, writing a test and the corresponding code, and then running the test to verify that our code works as expected. With this project, we'll take a different approach by first brainstorming all of the functionality our Pig Latin project needs, and making a list of that functionality in the form of tests ordered by simplest to most complex. Then, we'll start in on the TDD process.

1. Make a list of tests detailing the functionality that your program will have. Start with the simplest possible piece of functionality, and slowly move up in complexity. We will get you started with a potential first test. Note that there is no one right way to start or to build up your logic — just focus on starting small and gradually work your way up.

```
Describe: pigLatin()

Test: "It will add 'way' to the end of words that begin wit
h a vowel."
Code: pigLatin("a");
Expected Output: "away"
```

1. After you write all of your tests, **have at least one other pair check your tests before you begin coding.** Ensure that every possible functionality is represented by a test, and that they are ordered from simplest to most complex.
2. Place your tests in your project's README.

Use TDD to Write your Code

Now you're ready to start writing (and passing tests). Here's the process:

1. Starting with the first test you wrote for the simplest possible piece of functionality, write the smallest amount of code needed to get the test passing.
2. Test the code in the DevTools console.
3. Once the test is passing, commit your code.

Repeat the cycle above for *each* test. Do not move onto the next test until the previous one passes.

Once you've completed your business logic, feel free to add a user interface so you can run your application in the browser. **Do not add a UI until after you've written and tested all business logic.**

Helpful Hints

When you get to consonants, don't try to solve it all at once. Instead, start with an example of a word that only has one consonant. Next, try a word with two consonants. Finally, work your way up to a word with three consonants. After you've written passing tests for this code, tackle the exceptions to the rule like "qu" and "y". Then, when your application can successfully translate single words, work on translating entire sentences.

Here are a few methods that you may want to consider for solving this problem. Note that there are plenty of other ways to solve this problem, too!

- `String.prototype.slice()` (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/slice)
- `String.prototype.includes()` (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/includes)
- `String.prototype.indexOf()` (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/indexOf)

Take a look at this and other documentation to gather the tools you'll need before you try to fulfill your specs.

Further Exploration with Pig Latin: Refactor with Regular Expressions

We haven't covered regular expressions yet because we want you to primarily focus on looping. But if you want to try out regular expressions and you are comfortable with looping, now is your chance to refactor Pig Latin using regular expressions as described in the lesson Further Exploration: Introduction to Regular Expressions (https://www.learnhowtoprogram.com/introduction-to-programming/arrays-and-looping/further-exploration-introduction-to-regular-expressions).

Note: Once again, do not worry about using regex if you are still trying to understand loops. You do *not* need to use regex on any code reviews!

Further Exploration with Bases

Binary: Write a method to convert numbers from binary to decimal. The input should be a string, and the output an integer. Decimal is the normal system we use for counting. We start at 0, increment until we reach 9, and then reset back to 0 and add another number to the left. In binary, we also start at zero, but we only increment until we reach 1. Then we reset back to zero and add another number to the left. **Write specs, implement the smallest possible piece of functionality, and don't move onto the next test until the previous test is passing.**

Here are some example of numbers in decimal and binary:

```
Decimal  Binary
  0        0
  1        1
  2       10
  3       11
  4      100
...      ...
```

Trinary: You get where I'm going, right?

Hexadecimal: Here's what happens after 9...

```
Decimal  Hexadecimal
...      ...
 9        9
10        a
11        b
12        c
13        d
14        e
15        f
16       10
17       11
...     ...
```

If you get this far, write a method that takes two arguments: the number to be evaluated and the base you would like it to be evaluated in.

Peer Code Review

- Are there tests for each piece of functionality?
- Are all tests passing?
- If business and user interface logic well-separated?
- Does the application work as expected?

Previous (/introduction-to-programming/arrays-and-looping/for-loops-with-text-analyzer)
Next (/introduction-to-programming/arrays-and-looping/further-exploration-introduction-to-regular-expressions)

disable dark mode

(http://www.epicodus.com)