Assignment #03: Ruby Games

This assignment is about writing Ruby scripts of games to play on the command line. The three games you're going to write are Pig Latin, Hangman, and Tic Tac Toe.

Learning Goals:

Pseudocode

As a programmer, it is important that you and other stakeholders clearly understand the problem you are trying to solve and the specifications related to the solution of that problem. There are several effective tools every programmer should be armed with: user stories, wireframes, and pseudocode. For now we are focused on writing proper **pseudocode**.

To be able to understand how to read these models and translate them into a functional program, pseudocode is used to communicate the purpose of an algorithm without getting slowed down by language-specific syntax. A good programmer can read well-written pseudocode and translate it into functional code in the language of his choice.

Ruby

Through this project you will learn about different Ruby **data types, built-in Ruby methods**, how to **define methods**, how to **pass parameters into methods**, what an **implicit and explicit return** are, how to use **flow control statements**, **loops** and **terminal I/O** (input/output).

Debugging

As a programmer, you will spend many hours debugging your programs when they break. **Debugging** is a skill that gets better with practice. In order to save you time in the future, it is important to first develop a mental framework for solving tough problems. *How to Solve it* by George Polya gives us a methodology to break problems down in four steps:

1. First, you have to understand the problem.
    a. What are you asked to find or show?
    b. Can you restate the problem in your own words?
    c. Can you think of a picture or a diagram that might help you understand the problem?
    d. Is there enough information to enable you to find a solution?
    e. Do you understand all the words used in stating the problem?
    f. Do you need to ask a question to get the answer?
2. After understanding, then make a plan.
    a. Solve a simpler problem
    b. Use direct reasoning

    c. Look for a pattern
    d. Draw a picture
    e. Work backward
    f. Use a model
    g. Make an orderly list
    h. Eliminate possibilities
    i. Guess and check
    j. Use symmetry
    k. Consider special cases
    l. Solve an equation
    m. Use a formula
    n. Be creative

3. Carry out the plan.
4. Look back on your work. How could it be better?

- *Wikipedia*

External Resources:

- Ruby - Constants
- Ruby - Class and Instance Methods
- Ruby - Boolean Expressions
- Ruby - If..else
- Ruby - Method returns
- Ruby - Hashes
- Ruby - Default Parameters
- Ruby - File I/O
- Ruby - Exceptions
- Ruby - Regular Expressions
- Ruby - Attr_reader, writer and accessor

Pig Latin

Description:

Pig Latin is a language game where words in English are altered according to a simple set of rules. Pig Latin takes the first consonant of an English word, moves it to the end of the word and suffixes an ay, for example, pig yields igpay, banana yields ananabay, and truck yields ucktray. If the word starts with a vowel, don't change it.

The objective is to conceal the meaning of the words from others not familiar with the rules. The reference to Latin is a deliberate misnomer, as it is simply a form of jargon, used only for its English connotations as a "strange and foreign-sounding language." -*Wikipedia*

Suggested Process:

Build with Pseudocode

1. Write the entire program in pseudocode, making sure to fully document each step (however small) you would take to solve the problem.

Implement the .convert method

2. Write the program in Ruby to achieve the following result

Here are some sample inputs and the expected output for the method `PigLatinConverter.convert`:

```
#   INPUT              OUTPUT
# -------------------------------
#   pig               igpay
#   computer          omputercay
#   freedom           eedomfray
#   string            ingstray
#   String            ingstray
#   StrinG            ingstray
#   <EMPTY STRING>    <EMPTY STRING>
#   nil               nil
```

Github repo to clone for setup. (If and only if you want!)

Hangman

Description:

Hangman is a paper and pencil guessing game for two or more players. One player thinks of a word, phrase, or sentence and the other tries to guess it by suggesting letters or numbers. The word to guess is represented by a row of dashes, representing the amount of letters and numbers.

If the guessing player suggests a letter or number which occurs in the word, the other player writes it in all its correct positions. If the suggested letter or number does not occur in the word, the other reduces the amount of guesses left. The game is over when either the amount of guesses left reaches zero or the word is guessed. - *Wikipedia*

Suggested Process:

Implement Launching the Game

In our version of Hangman the game master is the computer.

The game starts off with the computer choosing a word and displaying that word with every letter replaced by an underscore.

For example, if the word is *lemonade* the computer displays _ _ _ _ _ _ _ _ _. To make the computer choose a random word, we recommend that you build an array of preset words and choose a random word out of that array.

We also want to show the player the number of chances they have to guess the word. Since we're just starting the game, that number is 8.

Implement Guessing Logic

Each turn the player guesses a letter.

If the player guessed right, all occurrences of the letter are revealed. For example, if the hidden word is *lemonade* and the player's first guess is e, the word becomes _ e _ _ _ _ _ e. The number of chances does not change.

If the player guessed wrong:

- the number of chances is reduced by 1
- the letter is added to the list of guessed letters, which is displayed every turn

Implement Game Ending

Player Wins:

The game ends if the player successfully guessed all letters in the game with chances to spare.

The program should print out a message saying the player won.

Player Loses:

The game ends as soon as the player used up their last chance. The program should print out a message saying the player lost.

Github repo to clone for setup. (Again, if and only if you want!)

Tic-Tac-Toe

Description:

Tic-Tac-Toe (or Noughts and crosses, Xs and Os) is a pencil-and-paper game for two players, X and O, who take turns marking the spaces in a 3×3 grid. The player who succeeds in placing three respective marks in a horizontal, vertical, or diagonal row wins the game. - *Wikipedia*

Suggested Process:

Implement Launching the Game

At the beginning of the game a tic-tac-toe board is drawn. The board has 3 columns and 3 rows and each square is numbered 1 through 9. Like this:

```
1 | 2 | 3
---------
4 | 5 | 6
---------
7 | 8 | 9
```

Implement Playing Turns

Player 1 starts off the game. To play a square they will type its number, for example, if they want to play in the center they will type 5. Once they play a square the number will be replaced by the player's mark. After a square has been played it must be checked if the player has won or tied the game. If neither have occurred, then the board is drawn again and the next player plays their turn, following the same steps. The players alternate turns until the game has been won or tied.

Implement Game Ending

Player Wins

If a player gets three of their pieces in a row they win. This means if they get three pieces *diagonally, vertically, or horizontally*.

Players Draw

When a square is played its number is replaced with a mark, which is a string (i.e. "x" or "o"). Therefore we know the game has been tied when the board has no more numbers because all of the squares will have been played.