code challenges

#Coding Program - Ruby Foundations - Exercises

The purpose of these exercises is to help you apply and practice some of the basics which you have covered in the Treehouse videos and exercises. A good understanding of the basic principles of programming and the foundations of Ruby will help you to progress with the more complicated parts of the course.

We would like you to work on these exercises in pairs initially, and where possible, to compare your final solutions with those of the other pairs in the group - discuss the differences between your solutions and the reasons for them!

Each of these exercises starts simple, but is open ended, so that you can progress as far as you are able to - returning to a working program and adding new features is an crucial part of development! Feel free to add your own features if you want to continue after you finish an exercise.

Oh, and you’re probably going to get stuck at some point.. we deliberately haven’t told you everything..!

There's a good Ruby cheat-sheet here: http://www.pragtob.info/rails-beginner-cheatsheet/ (ignore the 'Rails' bits for now!)

Some items are in italics. Treat these as ‘homework’ exercises if you want to dig a deeper. Please try to complete EITHER 1, 2 or 3 first and then move on to one of the games (4+).

The key foundations you should aim to build a solid understanding of include:

• Flow control (if, else, elsif, unless, case, conditional expressions using ?)

• Loops (while-until, for-in, iterators - each, times, map, upto)

• Arithmetic - add/subtract/multiple/divide

• Arrays & Hashes

• Strings (printing, reading, manipulating and changing)

• Objects, Classes, Inheritance

A good understanding of Object Orientated programming is really important, so where possible, try to take an ‘OO’ approach to solving these exercises. Ask for help if you need it!

Testing is optional. If you’d prefer to write tests and are comfortable setting up your environment then feel free to do so but don’t feel obligated.

• Hello, World!

• Create a program which prints “Hello, World!” to the screen

• Update your program so that it asks the user for their name and greets them

• Update your program so that it will only greet it’s creators!

• Make the output look “nicer” - for example, put the text in a frame, one word per line:

\*\*\*\*\*\*

\*Like\*

\*This\*

\*\*\*\*\*\*

• Add the current date and time to the output

• Times Tables

• Create a program which outputs the 2 times table ( 1 x 2 = 2, 2 x 2 = 4, etc )

• Amend your program so that it asks for a number and outputs that table (e.g. 2, 3, etc)

• Amend your program so that if -1 is given above, it will output all tables from 1 to 12

• Have you repeated code within your program? Refactor your code to use methods to avoid this.

• Create a class which generates a Times Table and use this to produce the output

• SOS

• Create a program which prints the Morse Code alphabet, from A to Z, using . and - (http://en.wikipedia.org/wiki/Morse\_code)

• Convert your program so that it works using a “range” and a “loop” and a method to produce the same output (less code for the same output)

• Instead of outputting the alphabet, ask the user to input some text and then convert it to Morse code

• Add functionality to your program to convert input in Morse code back into English

• Snakeeyes!

• Create a program which outputs a random number between 1 and 6, simulating the roll of a die

• Make your program ask the user how many dice s/he would like to roll. Display the result as individual numbers (e.g. 4, 6, 2 if three dice were rolled)

• Add the total to the output above

• Create a class which represents a die and re-structure your program to use this

• Create a child class of your ‘die’ which outputs one of the values used in the game ‘Crown and Anchor’ - http://en.wikipedia.org/wiki/Crown\_and\_Anchor

• Create a new program, using your die class above which mimics the game ‘Crown and Anchor’. Ask the user which area they would like to bet on (and how much), simulate the dice roll, and tell them how much they would have won (or lost)!

• Create a new program, using your die class. Using the rules of Yahtzee, allow the user to make multiple throws of the dice and record their score

• I’m thinking of a number…

• Write a program which picks a random number between 1 and 100. Ask the user to guess the number. Tell them when they get it right (and then exit)

• Instead of exiting, ask the user if they would like to play again

• Give the user a hint - “Higher!” or “Lower!” depending on how close their guess was

• If the user is close (within 5 or 10 - you decide), then give them more help - “Almost! But Lower!”, etc

• Keep track of the number of guesses the user has to make and tell them how many it took at the end

• Track how long (in seconds or minutes) it takes the user to guess correctly and let them know when they get it right

• Allow the user to set the upper and lower limits when they start the game (wrap your logic in a class, so that you pass these values in when starting up)

• Write the same program, but the other way around! Make a program which will try to guess a number you are thinking of. Make it react to feedback such as “lower, higher”, etc!

• Rock, Paper, Scissors

• Create a program which will play RPS with you. It should ask you for your choice, then tell you it’s choice and who won

• Make the program keep score as to who has won how many rounds

• Make it announce a winner on a “best out of 3” basis, then start again

• Add functionality to ask the user if they want to make it “best out of 5” if they lose (and then best out of 7, etc, so there is the option to never admit defeat)

• Refactor your program so that you have a parent (base) class for valid choices - and one specific class each for Rock, Paper and Scissors. Have a ‘Beats’ or ‘IsBeatenBy’ method which checks who has won.

• Can you make your program win more than half the time by making it a little more intelligent? Are people predictable when they play RPS?

• Alternatively, add another type of choice to your game e.g. Rock Paper, Scissors, Lizard, Spock (http://en.wikipedia.org/wiki/Rock-paper-scissors-lizard-Spock)

• Word games

• Create a program which takes string input from the user and then prints it reversed

• Modify your program so that it scrambles the word randomly (turning it into an anagram)

• Allow the user to guess the anagram and tell them if they get it right (or not), before letting them guess again

• Now try things the other way around - ask the user for an anagram which the program will try to solve. To start with, print a list of 5 words and ask the user to scramble one of them.

• Write a routine which goes through each word in the list and checks if it could be the answer to the anagram. Remember:

• It must be the same length

• It must contain all the same letters

• It must contain the letters the right number of times

• Wrap the code which performs these tests in a class, so that when you are given an anagram, you create an instance of ‘Anagram’ (or however you have named it), then call methods or properties on this to test candidates words to see if they match

• Find an online word-list file (e.g. this one http://wordlist.aspell.net/), download it and load it into your program so that it can solve anagrams without limiting the user’s choice!

• Fizzbuzz

• https://en.wikipedia.org/wiki/Fizz\_buzz