Earnest's Coding Challenge

Thank you for your interest in Earnest. To get a sense of how you write software we would like you to work on a short coding challenge.

Choose your challenge

Below are descriptions for two coding challenges. Please implement a solution for **one** of those challenges. Which one is entirely up to you, and your choice will not factor into our hiring decision.

Choose your tools

At Earnest we primarily work in JavaScript, Scala and Java, but you are welcome to use any language, environment, and development stack you'd like. You are also welcome to leverage 3rd party libraries, as long as your submission still allows us to evaluate your coding capabilities.

Solve the challenge

We respect and value your time. Please don't spend more than 2 or 3 hours on the challenge. Once you are ready, create a zip file and submit your solution using the upload link we sent you via email.

Evaluation Criteria

Our engineers will be assessing your code against 3 broad criteria:

- **Correct code**: Does the program meet the spec? Does it run, does it produce correct results? How does it handle edge cases and bad input?
- **Maintainable code**: Is the code well structured and easy to read? Does it follow language idioms? Is the code well designed such that it could be easily extended or modified? Are there tests, and do they provide appropriate coverage? Is there documentation describing things like design decisions and assumptions around product requirements?
- **Pragmatic, operable code**: Is the code focused on solving the spec or is it over-engineered, providing more functionality than we requested? Are development tasks such as running tests or compiling code automated or do they require manual steps? Is the program easy to configure for different environments? Is there "commodity" functionality in the solution which would have been better provided via a third-party library?

Once you're onsite

Part of our onsite interview process will involve continuing to work on your solution with a couple of our engineers. You should assume that your code will need to meet more requirements in the future.

Blind Review

Our engineers will be performing a "blind" review your solution. Please try to keep your submission anonymized. For example please don't put your name in a file header comment, or in the filename of your zip file.

Let's keep this between us

Please do NOT put your solution on Github or otherwise share the problems or your solution in any other public forum. Thanks!

Good Luck!

Thanks so much for taking the time to show us your skills. We hope you enjoy working on whichever challenge that you pick, and look forward to coding with you soon!

The Battleship Challenge

"Battleship is known worldwide as a pencil and paper game which dates from World War I. It was published by various companies as a pad-and-pencil game in the 1930s, and was released as a plastic board game by Milton Bradley in 1967. The game has spawned electronic versions, video games, smart device apps and a film."

In our simplified version of battleship, a board can be thought of as a grid of potentially attacked positions along with one or more ships each of which occupy one or more positions. Your task is to model a board and implement the ability to attack a position on that board.

Managing multiple players, reading user input, or actually playing a game is beyond the scope of the exercise, which is simply to implement the backend state and logic required to execute an attack.

An attack should result in the new state of the board along with one of the following outcomes:

- 'Hit' if there is a ship occupying the position
- 'Miss' if no ship occupies the position
- 'Already Attacked' if the position has previously been attacked
- 'Sunk' if the attack hits the last remaining position of a ship
- 'Win' if the attack sinks the last remaining ship

Your program will need to manage state from previous attacks in order to supports outcomes such as 'Already Attacked'.

Note that no specific version of battleship is specified for this task (eg: American, Indian, Japanese, etc). You can and should constrain your implementation in any way you see fit given the time recommendation. Interesting features and mechanics from the various rulesets used around the world may be used as enhancement tasks in the live coding session, so we recommend keeping your code simple, flexible, and extensible!

Deriving Fixed Expenses before Education

Your task is to write a program which will take a person's credit report, parse it, and return JSON describing both the parsed contents of the credit report and some derived facts.

Credit Report Format

A Credit Report contains a list of tradelines (essentially a list of any loan or available line of credit that a person has). Each tradeline has a code and subcode which describes the type of liability. For example, all credit cards have a code of '12', a conventional mortgage has a code of '10' and a subcode of '12'. Each tradeline also has a monthly payment and a current balance. Some tradelines may not carry a balance - for example, a fully paid off credit card.

Our credit report input comes in a specific format. Each line represents a tradeline, with the different properties of the tradeline in a specific order, separated by spaces. The properties are listed in the following order: reported date, code, subcode, monthly payment, and current balance. For example, the following line describes a credit card account with a code of 12 and a subcode of 5 reported on May 27, 2015 with a minimum monthly payment of $120.00 and a current balance of $2,113.12:

```
2015-05-27 12 5 $120.00 $2113.12
```

Monetary values may or may not be prefixed by a dollar sign, and may also sometimes have thousands separated by commas. They will never have fractional cents.

Any line with an unexpected number of fields can be safely ignored.

Fixed Expenses Before Education

Your program must calculate a person's Fixed Expenses Before Education based on their credit report. Fixed Expenses Before Education is defined as:
Non-Housing Expenses + Housing Expenses.

Housing Expense is the sum of monthly payments for all mortgage tradelines, where a mortgage tradeline has a code of 10 and a subcode of 12 or 15. If the credit report contains no mortgage tradelines, then the program should assume a housing expense of $1061 (the national average monthly rent).

Non-Housing Expenses is the sum of monthly payments for tradelines which are not mortgages and are not student loan payments. A student loan tradeline has a code of 5.

Any tradelines with zero current balance should not be considered in this Fixed Expenses calculation.

Output Format

Your program should output data about each parsed tradeline plus the calculated Fixed Expenses Before Education in the following format:

```
{
  fixed_expenses_before_education: 412321,
  tradelines: [
    {
      type: 'education',
      monthly_payment: 34131,
      current_balance: 14210021,
    },
    {
      type: 'mortgage',
      monthly_payment: 234412,
      current_balance: 51232121,
    },
    {
      type: 'other',
      monthly_payment: 31241,
      current_balance: 4123,
    },
    {
      type: 'mortgage',
      monthly_payment: 123012,
      current_balance: 21330061,
    }
  ]
}
```

Note that all monetary amounts must be represented in non-fractional cents. For example, $1245.21 must be represented as 124521.

Sample Input and Output

For this input:

```
2015-10-10 10 12 $1470.31 $659218.00
2015-10-10 5 1 $431.98 $51028.00
2015-10-09 8 15 $340.12 $21223.20
2015-10-10 10 15 $930.12 $120413.00
2015-10-09 12 5 $150.50 $6421.21
```

the following output is correct:

```
{
  "fixed_expenses_before_education": 289105,
  "tradelines": [
    {
      "type": "mortgage",
      "monthly_payment": 147031,
      "current_balance": 65921800
    },
    {
      "type": "education",
      "monthly_payment": 43198,
      "current_balance": 5102800
    },
    {
      "type": "other",
      "monthly_payment": 34012,
      "current_balance": 2122320
    },
    {
      "type": "mortgage",
      "monthly_payment": 93012,
      "current_balance": 12041300
    },
    {
      "type": "other",
      "monthly_payment": 15050,
      "current_balance": 642121
    }
  ]
}
```