Code Challenges

Please select **one** of the following challenges to complete. Select the challenge you find most interesting. We expect the project to take between 1 and 2 hours.

For each challenge we expect you to complete the following:

1. Create a solution that abstracts the principal functionality of the challenge behind a clear interface. We are not expecting you to develop a perfect service/API, but the interface should show a good understanding of encapsulation.
2. Develop **either:** a set of unit tests to exercise the module you develop or a console/web-client that utilizes it. If you choose the latter please write a few bullets on what your testing strategy would be.
3. Prepare a minimal readme file explaining how to run your project (i.e. what commands if any to build, which to run)

Things we evaluate all challenges on:

1. Idiomatic and clear use of language/framework chosen to develop the solution
2. Appropriate use of data structures and algorithms (these are not brain teasers but also develop code that would not grind to a halt on a big data set)
3. Code is well-factored and readable (should look like well-crafted code not a quick throw away script)
4. Documentation as appropriate
5. Is the code extensible (i.e. if more variations on given parts of the challenge were added how easy would it be to accommodate them)

Challenge 1: Consumer Cart Suggestions

Design a service that can ingest a user's current shopping cart and return a list of suggestions for discounts, deals, or upsells that the front-end can then decide to show or not based on various A|B testing strategies.

Requirements:

- Create a module which takes a JSON input consisting of *customer* and *cart*.
    - Return a JSON response of the cart annotated with a list of *suggestions* which will be read by the caller.
    - It is possible for a single cart to result in multiple suggestions.
- The decision of which suggestion to use will be made by the caller, so no need to worry about that here.
- *suggestion*s should include the following as appropriate:
    - Indication if it is an *upsell*, *discount*, or *freeShipping*.
- (If appropriate) data required to process the suggestion (i.e. what product to upsell, what discount to apply, etc.).

- (If appropriate) what line item the suggestion relates to (i.e. which item are we discounting, what product prompted the upsell).

Suggestions to provide:

- Offer free shipping on orders over $100.
- If the customer's cart contains a Card Pack, upsell a Lovepop Note 5-Pack.
- If the cart contains any products tagged *holiday*, upsell the Santa Biker T-Shirt.
- Lovepop wants to run a "5 for $50!" sale. If there are 5 or more Greeting Cards in the Cart, discount all Greeting Cards to $10.
    - Note: only applies to Greeting Card products.

Extra Credit:

- Don't allow the "5 for $50!" discount on Licensed cards.
- If the customer is a Lovepop Prime member, offer free shipping and discount all Greeting Cards to $10.
- If the customer's cart contains only one Greeting Card and it has a matching T-Shirt, upsell that product

See the included *challenge_1_product_data.json* and *challenge_1_cart_data.json* for sample data.

Challenge 2: SVG File Variation

A centerpiece of our 3D Custom Card Design system is an engine that allows us to parameterize design files by parsing those files and outputting "variables", which represent content in the file that can be substituted. For this challenge, we would like you to develop a simple module to perform basic variable extraction and substitution for an SVG file.

Requirements:

- Create a module which performs two major functions:
    - takes an SVG file as input, extracts variables for *fill* and *rotate* attributes, and returns the set of identifiers for those variables.
    - takes an SVG file and a JSON data structure containing the set of variable identifiers and the values they should be substituted with, and returns a new SVG file with the variables substituted.
- The output of the first function and the input of the second function should use consistent identifiers. That is, if the output of the first function returns **fill1** as a variable identifier, the second function should accept a new value for **fill1** and substitute it in the same variable instance.

See the included *challenge_2_sample.svg* for sample data.

Challenge 3: Realtime order processing

For this challenge you will build a system to ensure appropriate facilitation of order fulfillment given a set of prioritization logic.

Requirements:

- Assume that we can only fulfill **N** orders per day (this can be a fixed constant in your solution) we want to build a module that at any time can return the N most important orders to fill.
- Orders can be simple JSON structures (e.g. {id: 1, priority: 'high', ship_date: 2019-01-01})
- To make this a realtime approach the interface you develop should accept orders one or a few at a time and not in bulk (e.g. it should be able to update the set of N orders to fill on an ongoing basis.
- There should be a method that returns the list of orders to fulfill at any time as a JSON list (they need not be prioritized)

Order prioritization:

- **Ship Date -** We *must* ship orders if today is there ship date and there is still capacity, we *may* ship orders if it is within 3 days of their ship date).
- **Priority -** Low, Medium, High - if capacity constrained we should prioritize high over medium and medium over low orders