Exercise Introduction

- Candidates need to complete all 3 exercises
- The response to the programming exercises can be implemented in javascript or ruby. Feel free to use any OSS library that makes the task easier.

How to Submit

- Send us your source code by email or online repository
- Be sure to include at least one example of how to run your code for each exercise.

Exercise #1

Introduction

Publish a small service on the web that has two endpoints:

- `/messages` takes a message (a string) as a POST and returns the SHA256 hash digest of that message (in hexadecimal format)
- `/messages/<hash>` is a GET request that returns the original message. A request to a non-existent `<hash>` should return a 404 error.

Example

Let’s say you publish to http://mywebsite.com/ (you don’t need a custom domain for this project, any IP address we can access will do):

```
$ curl -X POST -H "Content-Type: application/json" -d '{"mess
age": "foo"}'
http://mywebsite.com/messages
{
 "digest": "b5bb9d8014a0f9b1d61e21e796d78dccdf1352f23cd32812f
4850b878ae4944c"
}
```

You can calculate that your result is correct on the command line:

```
$ echo -n "foo" | shasum -a 256
b5bb9d8014a0f9b1d61e21e796d78dccdf1352f23cd32812f4850b878ae49
44c -
```

You can now query your service for the original message:

```
$ curl
http://mywebsite.com/messages/b5bb9d8014a0f9b1d61e21e796d78dc
cdf1352f23cd32812f4850b878ae4944c

{
 "message": "foo"
}
```

Querying for a message that does not exist returns a 404 status code:

```
$ curl -i
http://mywebsite.com/messages/bar
HTTP/1.0 404 NOT FOUND
Content-Type: application/json
Content-Length: 36
Server: Werkzeug/0.11.5 Python/3.5.1
Date: Wed, 31 Aug 2016 14:21:11 GMT

{
  "err_msg": "Message not found"
}
```

(your specifics may vary, all that matters is that you get a 404)

Exercise #2

Introduction

You have been given a gift card that is about to expire and you want to buy gifts for 2 friends. You want to spend the whole gift card, or if that's not an option as close to it as possible. You have a list of sorted prices for a popular store that you know they both like to shop at. Your task is to find two distinct items in the list whose sum is minimally under (or equal to) the gift card balance.

The file contains two columns:

- A unique identifier of the item. You can assume there are no duplicates.
- The value of that item in cents. It is always a positive integer that represents the price in cents (1000 = $10.00).

Write a program to find the best two items. It takes two inputs:

- A filename with a list of sorted prices
- The balance of your gift card

If no two items have a sum that is less than or equal to the balance on the gift card, print “Not possible”. You don’t have to return every possible pair that is under the balance, just one such pair.

Example

```
$ cat prices.txt
Candy Bar, 500
Paperback Book, 700
Detergent, 1000
Headphones, 1400
Earmuffs, 2000
Bluetooth Stereo, 6000

$ find-pair prices.txt 2500
Candy Bar 500, Earmuffs 2000

```

```

$ find-pair prices.txt 2300
Paperback Book 700, Headphones 1400

$ find-pair prices.txt 10000
Earmuffs 2000, Bluetooth Stereo 6000

$ find-pair prices.txt 1100
Not possible
```

Note: There may be many rows in the file, so be sure to optimize your solution to scale. What is the big O notation for your program?

Exercise #3

Introduction

You are given a string composed of only 1s, 0s, and Xs.

Write a program that will print out every possible combination where you replace the X with both 0 and 1.

Example

```
$ myprogram X0
00

```

```
1

$ myprogram 10X10X0
1001000
1001010
1011000
1011010
```

While your program will take longer to run based on the number of possible combinations, your program shouldn’t crash (or hang) on an input with many Xs.

What is the big O notation for your program?

Evaluation Criteria

Here’s what we’re looking for in successful submissions:

1. Correctness. Your solution should be correct for the example inputs.
2. Test suite. We value developer testing as a way to inform design decisions, provide executable documentation and reduce regression errors, so we’d like to see how you approach testing.
3. Low complexity.