Python Programming Challenge

As part of our selection process at Arkera, we kindly ask candidates to complete a short Python task. This will be used to provide a backbone for discussions in one or more of your technical interviews, based on who will be conducting them. Do not worry if you have little or no experience with Python, as what we are primarily looking for is your ability to pick up new ideas / frameworks and run with them.

The task should take no more than 35-40 minutes, so please plan well. The results will be "judged" based on code quality and visual cleanliness.

Find the instructions for the task below:

1. Debugging ~5 mins

Amend the following function to make the given test pass.

```
from unittest import TestCase

def increment_dictionary_values(d, i):
   for k, v in d.items():
       d[k] = v + i
   return d

class TestIncrementDictionaryValues(TestCase):

   def test_increment_dictionary_values(self):
       d = {'a': 1}
       dd = increment_dictionary_values(d, 1)
       ddd = increment_dictionary_values(d, -1)
       self.assertEqual(dd['a'], 2)
       self.assertEqual(ddd['a'], 0)
```

2. Write a function & tests ~10 mins

We want a function which accepts a **very large** list of prices (`pricesLst`) and returns the largest possible loss a client could have made with only a buy transaction followed by a sell

transaction. The largest loss is calculated as `pricesLst[index2] - pricesLst[index1]` where `index1 < index2`.
Please then write tests for this function to ensure it works as expected guarding against all edge cases you can think of.

3. Write a set of classes ~20 mins

We have a sql table with 4 columns (id, url, date, rating). We need a set of classes that allows us to **build a query** which can filter this table across any **combination** of these possibilities:

- id: >, <, =, IN, NOTIN
- url: =
- date: >, <, =
- rating: >, <, =

E.g. we may want all entries with: (2 < rating < 9) **and** (id in list) **and** (date > 1 Jan 2016).
The goal is to have a set of classes which enable easy testing wherever they are used (i.e. the database does not have to be overly mocked every time). We want users of these classes to be able to add filters without having to add to or rewrite tests.