# optimizing_code_holiday_gifts

November 29, 2019

## 1 Optimizing Code: Holiday Gifts

In the last example, you learned that using vectorized operations and more efficient data structures can optimize your code. Let's use these tips for one more example.

Say your online gift store has one million users that each listed a gift on a wish list. You have the prices for each of these gifts stored in `gift_costs.txt`. For the holidays, you're going to give each customer their wish list gift for free if it is under 25 dollars. Now, you want to calculate the total cost of all gifts under 25 dollars to see how much you'd spend on free gifts. Here's one way you could've done it.

```
In [1]: import time
        import numpy as np

In [2]: with open('gift_costs.txt') as f:
            gift_costs = f.read().split('\n')

        gift_costs = np.array(gift_costs).astype(int)  # convert string to int

In [3]: start = time.time()

        total_price = 0
        for cost in gift_costs:
            if cost < 25:
                total_price += cost * 1.08  # add cost after tax

        print(total_price)
        print('Duration: {} seconds'.format(time.time() - start))
```

```
32765421.24
Duration: 6.143885612487793 seconds
```

Here you iterate through each cost in the list, and check if it's less than 25. If so, you add the cost to the total price after tax. This works, but there is a much faster way to do this. Can you refactor this to run under half a second?

### 1.1 Refactor Code

**Hint:** Using numpy makes it very easy to select all the elements in an array that meet a certain condition, and then perform operations on them together all at once. You can them find the sum of what those values end up being.

In [5]:
```
start = time.time()

total_price = np.sum(gift_costs[gift_costs < 25])*1.08 # TODO: compute the total price

print(total_price)
print('Duration: {} seconds'.format(time.time() - start))
```

```
32765421.24
Duration: 0.07780194282531738 seconds
```

In [ ]: