1. **(12 points) Expressions**

   (a) **(10 pt)** An array of integers named `ca` contains the (estimated) population of California every 10 years. It has 11 items. The first item is the population of California in 1900. The last is the population in 2000.

   ```
   array([ 1485053,  2377549,  3426861, ..., 23667902, 29760021, 33871648])
   ```

   Write a Python expression below each of the following descriptions that computes its value. The first one is provided as an example. Do not include numbers above (e.g., 1485053) in your solutions.

   - The population in 1900.

     ```
     ca.item(0)
     ```

   - The population change from 1940 to 1960, expressed as a number of persons (not a proportion).

     ```
     ca.item(6) - ca.item(4)
     ```

   - Whether the population ever grew by less than 500000 in a decade represented by `ca`. (`True` or `False`)

     ```
     min(np.diff(ca)) < 500000
     ```

   - The *annual* (yearly) growth rate from 1920 to 1930.

     ```
     (ca.item(3)/ca.item(2)) ** (1/10) - 1
     ```

   - The population in 1924, assuming a fixed exponential annual growth rate from 1920 to 1930. *You may use the name* `g` *for the annual growth rate from 1920 to 1930 (computed above).*

     ```
     ca.item(2) * (1+g)**4
     ```

   - The number of items in `ca` that are at least twice as large as the population in 1960.

     ```
     np.count_nonzero(ca >= 2 * ca.item(6))
     ```

   (b) **(2 pt)** You have 1000 different shirts in your huge closet, but only 2 of them are red. Each morning, you pick one uniformly at random, wear it, then put it back at night. Write a Python expression to compute the chance that, during a 30-day month, you *never* wear a red shirt.

   ```
   (998/1000)**30
   ```

2. (10 points) Tables

(a) (2 pt) The table named `twins` has a row for each pair of twins that contains the height of each twin in inches. The following code computes the average absolute difference in heights among the twins.

```
def diff(height1, height2):
    return abs(height1 - height2)
diffs = Table(['Absolute Differences'])
for i in np.arange(twins.num_rows):
    diffs.append([diff(twins.column(0).item(i), twins.column(1).item(i))])
np.mean(diffs.column(0))
```

Complete the expression below to compute the same result **without calling `diff` or using a `for` statement**.

```
np.mean(np.abs(twins.column(0) - twins.column(1)))
```

(b) (8 pt) Each row of the `trip` table from lecture describes a single bicycle rental in the San Francisco area. Durations are integers representing times in seconds. The first three rows out of 338343 appear below.

Start	**End**	**Duration**
Ferry Building	SF Caltrain	765
San Antonio Shopping Center	Mountain View City Hall	1036
Post at Kearny	2nd at South Park	307

Write a Python expression below each of the following descriptions that computes its value. You *may* use up to two lines and introduce variables.

- The average duration of a rental that lasted more than 2 minutes.

```
two_mins = trip.where('Duration', are.above(120))
two_mins.column('Duration').sum() / two_mins.num_rows
```

- The number of rentals that started at the `SF Caltrain` station.

```
trip.where('SF Caltrain', 0).num_rows
```

- The name of the station where the most rentals ended (assume no ties).

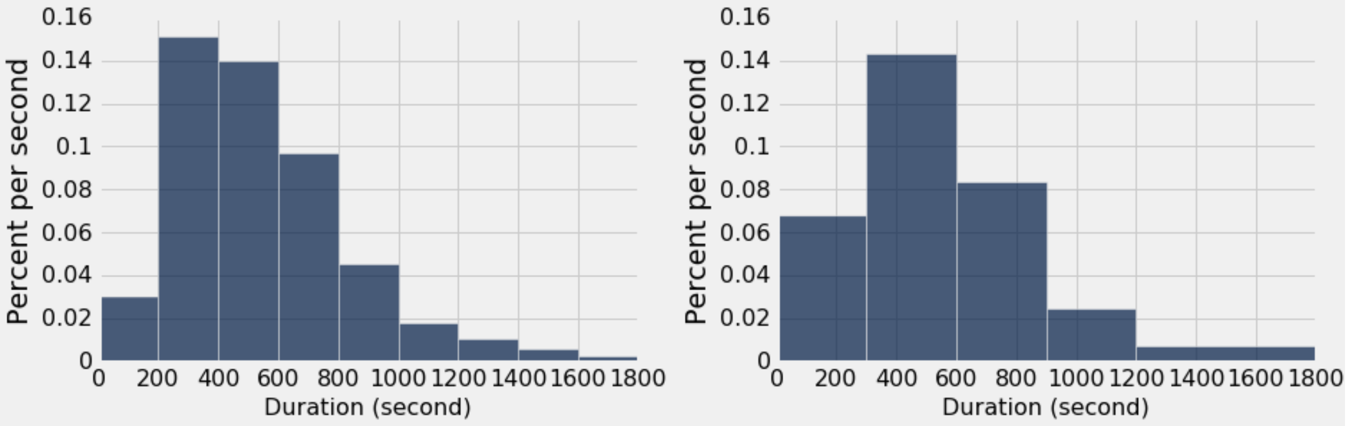
```
trip.group('End').sort('count', descending=True).column(0).item(0)
```

- The number of stations for which the average duration ending at that station was at least 300 seconds.

```
np.count_nonzero(trip.select(1, 2).group(0, np.mean).column(1) >= 300)
```

3. (11 points) Distributions

The two histograms of bike trip durations below were both generated by `trip.hist(...)` using different bins.



(a) (8 pt) Write the proportion of trips that fall into each range of durations below. *Show your work.* If it is not possible to tell from the histograms, instead write **Not enough information**.

- Between 200 (inclusive) and 400 (exclusive) seconds

  0.0015 * 200 == 0.30 or 30%

- Between 300 (inclusive) and 900 (exclusive) seconds

  0.0014 * 300 + 0.0008 * 300 == 0.66 or 66%

- Between 400 (inclusive) and 900 (exclusive) seconds

  0.0014 * 200 + 0.0008 * 300 == 0.52 or 52%

- Between 200 (inclusive) and 300 (exclusive) seconds

  0.0015 * 200 + 0.0014 * 200 - 0.0014 * 300 == 0.16 or 16%
  Alternatively: 0.0007 * 300 - 0.0003 * 200 == 0.15 or 15%

(b) (3 pt) A study followed 369 people with cardiovascular disease, randomly selected from hospital patients. A year later, those who owned a dog were four times more likely to be alive than those who didn't.

- Circle *True* or *False*: This study is a randomized controlled experiment. Answer: False; the experimenters did not control who owned a dog.

- Circle *True* or *False*: This study shows that dog owners live longer than cat owners on average. Answer: False; the experiment compares those who owned a dog to those who didn't (no mention of cats)

- Circle *True* or *False*: This study shows that for someone with cardiovascular disease, adopting a dog will probably cause them to live longer. Answer: False; an observational study does not show causation.

(10 points) Probability

(a) (4 pt) We roll two fair six-sided dice. If the sum is not 7, we re-roll just the second die (once). After rolling the two dice and possibly re-rolling the second, what is the probability that the sum of the two dice is now 7? It's OK to leave your answer unsimplified. *Examples*: **(A)** We roll 2 and 5: the sum is 2+5=7. **(B)** We roll 1 and 4, so we re-roll the second die and it comes up 6: the sum is 1+6=7.

$1/6 + 5/6 \times 1/6 = 11/36$: The chance of totaling 7 the first time (6 of 36) + the chance of not totaling 7 the first time (30 of 36) and then totaling 7 the second time (1 of 6).

or

$1 - (5/6 * 5/6) = 11/36$: The chance that the following does not happen: Not totaling 7 the first time (30 of 36) and not totaling 7 the second time either (5 of 6).

(b) (6 pt) Implement the **estimate** function, which returns an estimate of this probability generated by simulating the above process repeatedly, **trials** times, and returning the fraction of trials where the sum is 7. In each trial, we simulate rolling two fair six-sided dice and re-rolling the second if necessary.

```
def estimate(trials):
    sevens = 0
    faces = np.arange(1, 7)

    for i in np.arange(trials):
        first = np.random.choice(faces)
        second = np.random.choice(faces)

        if first + second != 7:
            second = np.random.choice(faces)

        if first + second == 7:
            sevens = sevens + 1

    return sevens / trials
```