1.

a)

```
In [9]: def my_function(num1):
            new_num=num1**3
            return new_num
```

```
In [10]: my_function(5)

Out[10]: 125
```

b)

```
In [50]: list_1=[1,2,3]

         import numpy as np
         array_1=np.array([3,6,9])

         my_tuple=(2,4,6)
```

c)

```
In [52]: list_2=list_1*3
         print (list_2)
         print (my_tuple*3)
         print (array_1*3)

[1, 2, 3, 1, 2, 3, 1, 2, 3]
(2, 4, 6, 2, 4, 6, 2, 4, 6)
[ 9 18 27]
```

Arrays allow numerical calculation like matrix. So every item in the array is multiplied by 3.

d)

```
In [58]: list_1[2]=23
         print (list_1)

         array_1[2]=23
         print (array_1)

         my_tuple[2]=23
         print (my_tuple)

[1, 2, 23]
[ 3  6 23]

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-58-79920d82cc6c> in <module>
      5 print (array_1)
      6
----> 7 my_tuple[2]=23
      8 print (my_tuple)

TypeError: 'tuple' object does not support item assignment
```

It’s because tuples are immutable. The items in it cannot be changed, but items in lists and arrays could be changed.

e)

In [2]:
```
list (range(0, 43, 3) )
```
Out[2]:
```
[0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42]
```

f)

In [12]:
```
from datascience import *
Table().with_columns(
    "Month", make_array("January", "Feburary", "March"),
    "days of using dictionarys", make_array(13,15,16),
    "days of using pandas", make_array(7,9,6),
)
```
Out[12]:

Month	days of using dictionarys	days of using pandas
January	13	7
Feburary	15	9
March	16	6

2.

a)

In [75]:
```
from datascience import *
import numpy as np
import matplotlib
matplotlib.use('Agg', warn=False)
%matplotlib inline
import matplotlib.pyplot as plots
```

In [8]:
```
vec = np.array([80, 78, 72, 79, 75, 72, 72])
percentile(50, vec)
```
Out[8]:
```
75
```

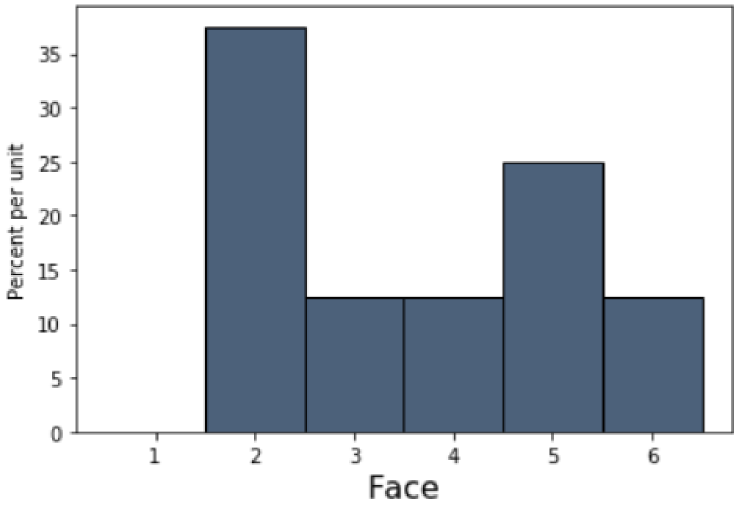
b)

In [19]:
```
percentile(75, vec)
```
Out[19]:
```
79
```

3.

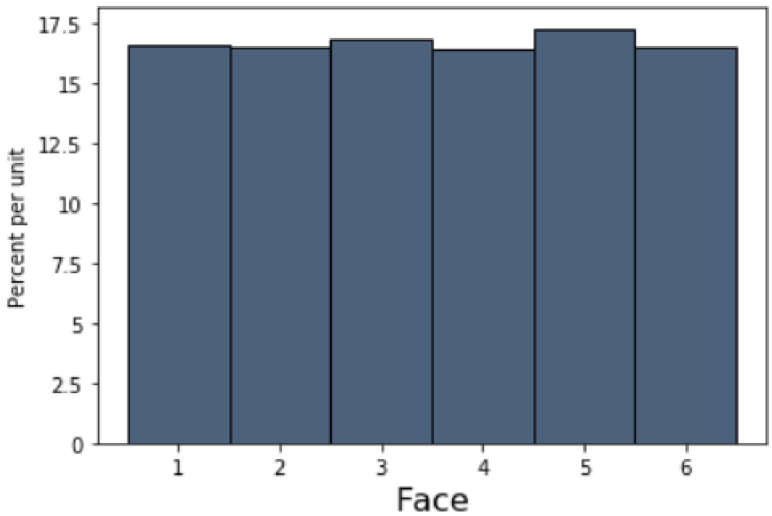
a)

```
In [7]: die = Table().with_column('Face', np.arange(1, 7, 1))
        die_bins = np.arange(0.5, 6.6, 1)
        def empirical_hist_die(n):
            die.sample(n).hist(bins = die_bins)
        empirical_hist_die(8)
```



b)

```
In [23]: empirical_hist_die(5000)
```



The area of each bar gets closer to 16.7%. It shows that theoretically, each appears about 16,7% of the time we row. It's because in the long run, the proportion of times that an event occurs gets closer to the theoretical probability of the event.

4.

a)

In [31]:
```
Regular_Gross_Paid = Table.read_table('nyc_population_salaries.csv')

percentile(50, Regular_Gross_Paid.column('Regular.Gross.Paid'))
```
Out[31]: 52555.43

b)

In [32]:
```
np.mean( Regular_Gross_Paid.column('Regular.Gross.Paid'))
```
Out[32]: 54490.83889002729

c)

In [33]:
```
sample_10 =  Regular_Gross_Paid.sample(10)
print( np.mean(sample_10.column("Regular.Gross.Paid")) )
```
53119.942

d)

In [34]:
```
sample_4000 =  Regular_Gross_Paid.sample(4000)
print( np.mean(sample_4000.column("Regular.Gross.Paid")) )
```
54505.111549999994

e) The second answer (with sample size of 4000) is closer to the population mean because is has a larger sample size.

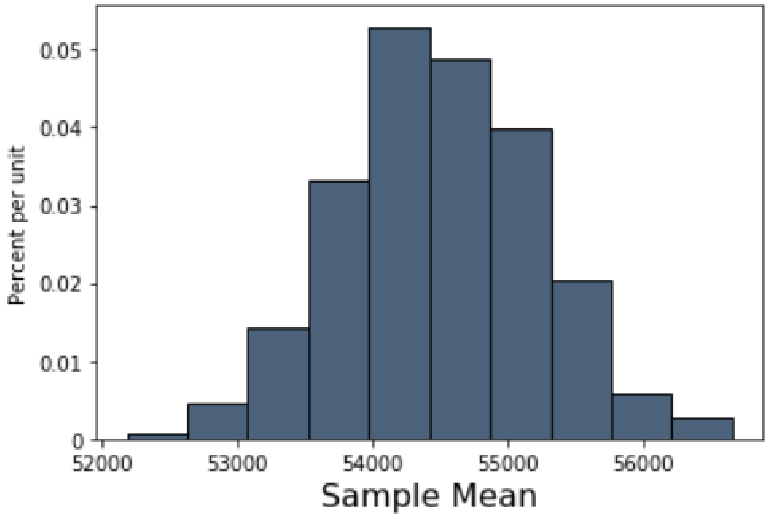
f)

In [35]:
```
def random_samp_mean():
    return np.mean( Regular_Gross_Paid.sample(1800).column('Regular.Gross.Paid'))
```

g)

In [34]:
```
nsims = 700

our_means = make_array()

for i in np.arange(nsims):
    our_means = np.append(our_means, random_samp_mean())
```

In [35]:
```
simulated_means = Table().with_column('Sample Mean', our_means)
simulated_means.hist()
```



h)

In [36]:
```
np.mean(simulated_means.column('Sample Mean'))
```

Out[36]: 54500.18622152381

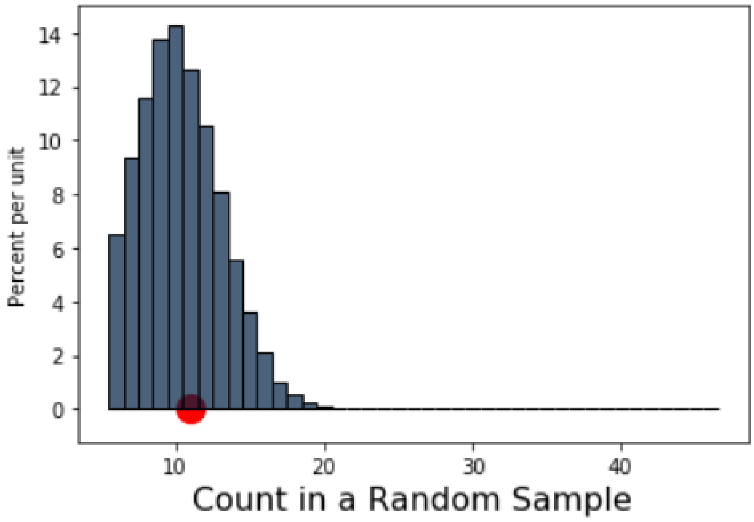
5.

a)

In [38]:
```
eligible_population = [0.10, 0.90]

def one_simulated_count():
    return (100 * sample_proportions(100, eligible_population)).item(0)
```

b)

In [40]:
```
counts = make_array()

repetitions = 15000
for i in np.arange(repetitions):
    counts = np.append(counts, one_simulated_count())

swain_observed = 11

Table().with_column(
    'Count in a Random Sample', counts
).hist(bins = np.arange(5.5, 46.6, 1))
plots.scatter(swain_observed, 0, color='red', s=200);
```

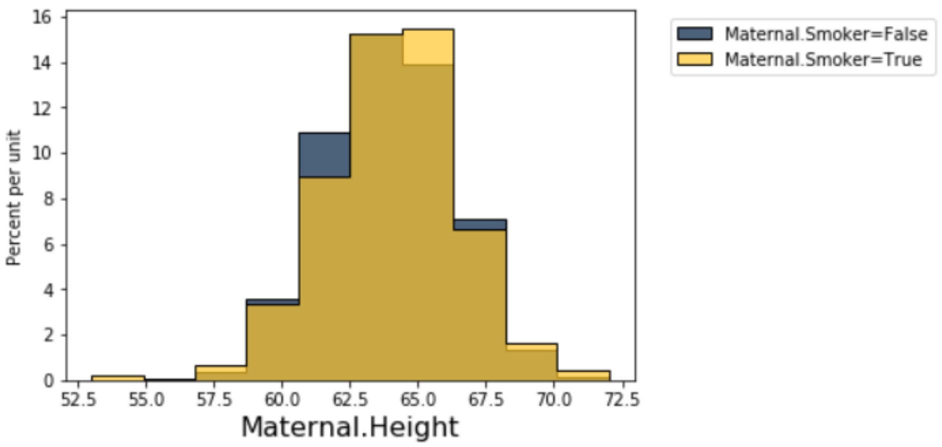


c) Yes.

6.

a)

```
In [43]: births = Table.read_table('baby.csv')
smoking_and_birthweight = births.select('Maternal.Smoker', 'Maternal.Height')
smoking_and_birthweight.hist('Maternal.Height', group = 'Maternal.Smoker')
```



b)

```
In [44]: means_table = smoking_and_birthweight.group('Maternal.Smoker', np.average)
means_table
```

Out[44]:

Maternal.Smoker	Maternal.Height average
False	64.014
True	64.1046

c)

```
In [45]: means = means_table.column(1)
observed_difference = means.item(1) - means.item(0)
observed_difference
```

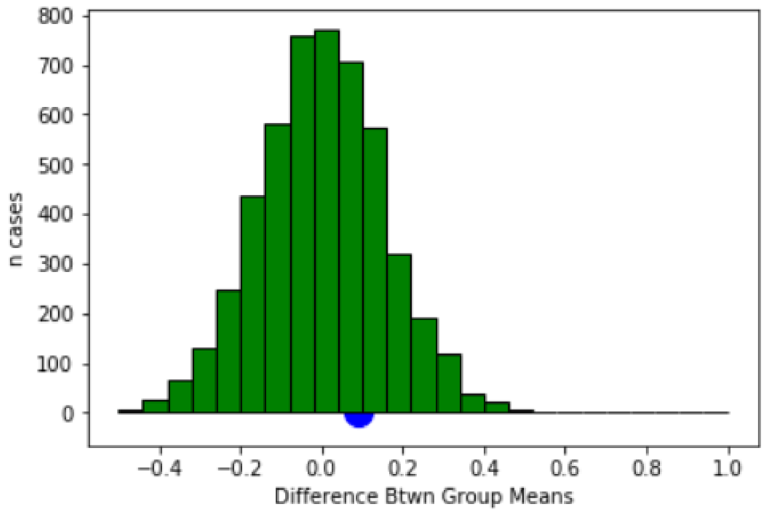
Out[45]: 0.09058914941267915

The observed difference is 0.09058914941267915. Maternal smokers are taller.

d)

In [48]:
```
def difference_of_means(table, label, group_label):
    reduced = table.select(label, group_label)
    means_table = reduced.group(group_label, np.average)
    means = means_table.column(1)
    return means.item(1) - means.item(0)

def one_simulated_difference(table, label, group_label):
    shuffled_labels = table.sample(with_replacement = False
                                                        ).column(group_label)
    shuffled_table = table.select(label).with_column(
        'Shuffled Label', shuffled_labels)
    return difference_of_means(shuffled_table, label, 'Shuffled Label')

differences = make_array()

repetitions = 5000
for i in np.arange(repetitions):
    new_difference = one_simulated_difference(births, 'Maternal.Height', 'Maternal.Smoker')
    differences = np.append(differences, new_difference)

plots.hist(differences, color="green",range=[-0.5, 1], bins=25)
plots.scatter(observed_difference, 0, color='blue', s=200);
plots.xlabel('Difference Btwn Group Means')
plots.ylabel('n cases')
```

Out[42]: Text(0, 0.5, 'n cases')



e)

In [46]:
```
empiricalP_lessthan = np.count_nonzero(differences <= observed_difference) / repetitions
print( "(simulated) P value for mean_smokers <= mean_nonsmokers=",  empiricalP_lessthan)
```

```
(simulated) P value for mean_smokers <= mean_nonsmokers= 0.733
```

The p-value is 0.733. It’s not statistically significant at conventional levels.

f)

```
In [55]: empiricalP_greaterthan = np.count_nonzero(differences >=  observed_difference) / repetitions
         print("(simulated) P value for mean_smokers >=  mean_nonsmokers=", empiricalP_greaterthan)

         (simulated) P value for mean_smokers >=  mean_nonsmokers= 0.2858
```

The p-value is 0.2734. It's statistically significant at conventional levels.

g)

```
In [48]: print( "(simulated) P value for mean_smokers != mean_nonsmokers=",  empiricalP_greaterthan*2)

         (simulated) P value for mean_smokers != mean_nonsmokers= 0.5468
```

0.5468.

7.

a) 22*0.05=1.1
b) 22*0.95=20.9
c) No, it doesn't. Since the experiments that result p-value greater than 0.05 are never published. But actually those could reject the null hypothesis. So there might actually be relationship between X and Y.