Chapter 2

Examples and Exercises from Think Stats, 2nd Edition

http://thinkstats2.com

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

In [2]:
```
from os.path import basename, exists


def download(url):
    filename = basename(url)
    if not exists(filename):
        from urllib.request import urlretrieve

        local, _ = urlretrieve(url, filename)
        print("Downloaded " + local)


download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/thinkstats2.py
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/thinkplot.py")
```

```
Downloaded thinkstats2.py
Downloaded thinkplot.py
```

Given a list of values, there are several ways to count the frequency of each value.

In [3]:
```
t = [1, 2, 2, 3, 5]
```

You can use a Python dictionary:

In [4]:
```
hist = {}
for x in t:
    hist[x] = hist.get(x, 0) + 1

hist
```

Out[4]:
```
{1: 1, 2: 2, 3: 1, 5: 1}
```

You can use a `Counter` (which is a dictionary with additional methods):

In [5]:
```
from collections import Counter
counter = Counter(t)
counter
```

Out[5]:

```
Counter({1: 1, 2: 2, 3: 1, 5: 1})
```

Or you can use the `Hist` object provided by `thinkstats2`:

In [6]:

```
import thinkstats2
hist = thinkstats2.Hist([1, 2, 2, 3, 5])
hist
```

Out[6]:

```
Hist({1: 1, 2: 2, 3: 1, 5: 1})
```

`Hist` provides `Freq`, which looks up the frequency of a value.

In [7]:

```
hist.Freq(2)
```

Out[7]:

```
2
```

You can also use the bracket operator, which does the same thing.

In [8]:

```
hist[2]
```

Out[8]:

```
2
```

If the value does not appear, it has frequency 0.

In [9]:

```
hist[4]
```

Out[9]:

```
0
```

The `Values` method returns the values:

In [10]:

```
hist.Values()
```

Out[10]:

```
dict_keys([1, 2, 3, 5])
```

So you can iterate the values and their frequencies like this:

In [11]:

```
for val in sorted(hist.Values()):
    print(val, hist[val])
```

```
1 1
2 2
3 1
5 1
```

Or you can use the `Items` method:

In [12]:

```
for val, freq in hist.Items():
     print(val, freq)
```

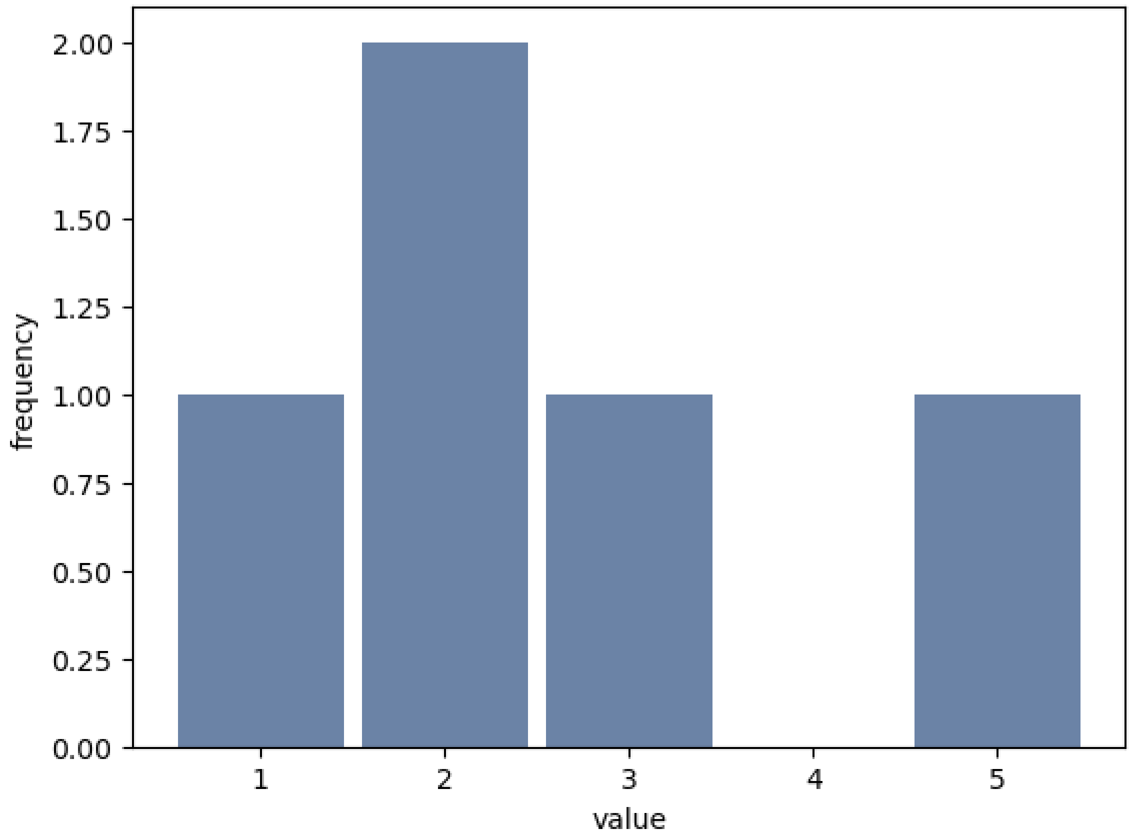
```
1 1
2 2
3 1
5 1
```

`thinkplot` is a wrapper for `matplotlib` that provides functions that work with the objects in `thinkstats2`.

For example `Hist` plots the values and their frequencies as a bar graph.

`Config` takes parameters that label the x and y axes, among other things.

In [13]:
```
import thinkplot
thinkplot.Hist(hist)
thinkplot.Config(xlabel='value', ylabel='frequency')
```



As an example, I'll replicate some of the figures from the book.

First, I'll load the data from the pregnancy file and select the records for live births.

In [14]:
```
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/nsfg.py")

download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/2002FemPreg.dc
download(
    "https://github.com/AllenDowney/ThinkStats2/raw/master/code/2002FemPreg.dat.gz"
)
```

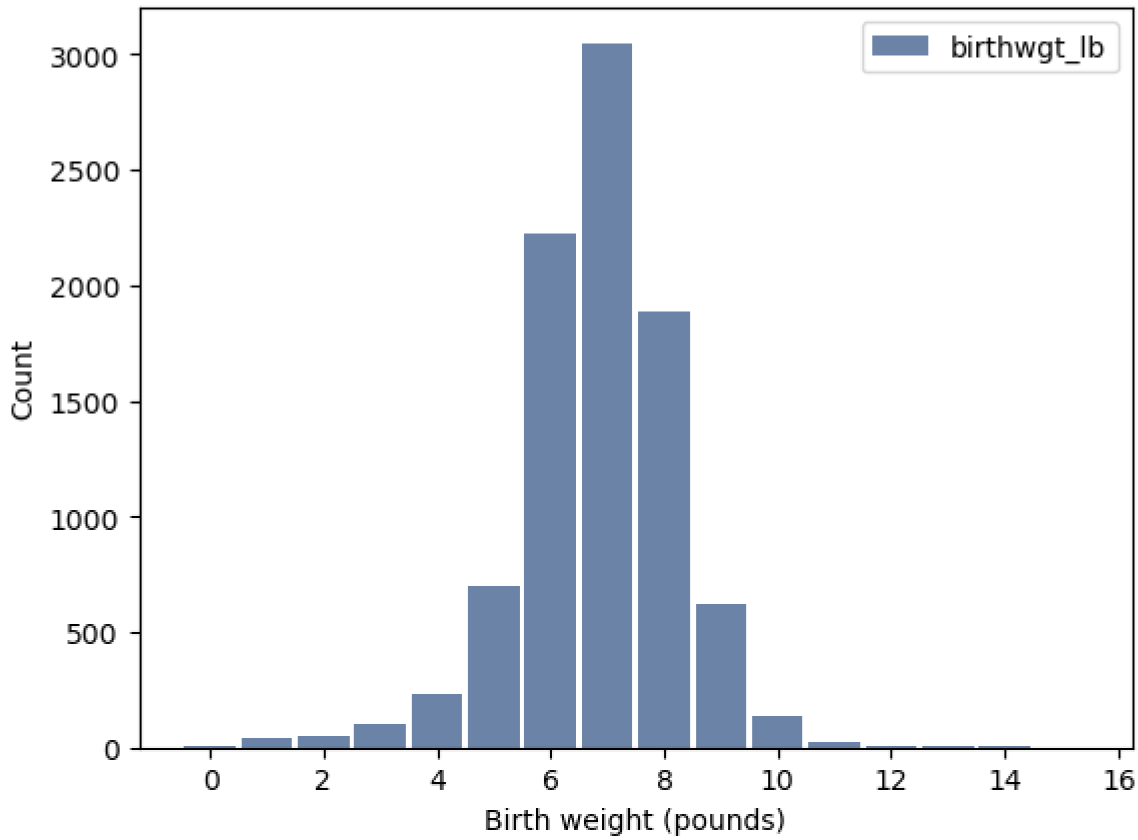
```
Downloaded nsfg.py
Downloaded 2002FemPreg.dct
Downloaded 2002FemPreg.dat.gz
```

In [15]:
```
import nsfg
```

In [16]:
```
preg = nsfg.ReadFemPreg()
live = preg[preg.outcome == 1]
```

Here's the histogram of birth weights in pounds. Notice that `Hist` works with anything iterable, including a Pandas Series. The `label` attribute appears in the legend when you plot the `Hist`.

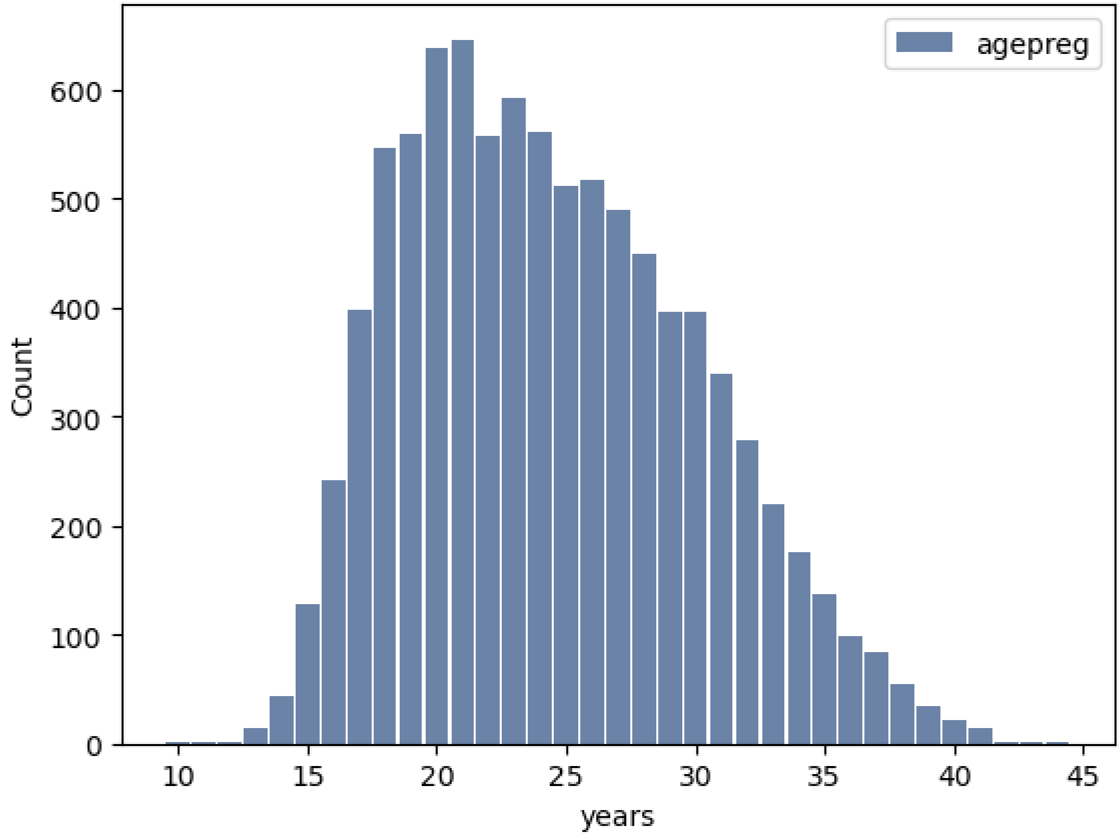
In [17]:
```
hist = thinkstats2.Hist(live.birthwgt_lb, label='birthwgt_lb')
thinkplot.Hist(hist)
thinkplot.Config(xlabel='Birth weight (pounds)', ylabel='Count')
```



Before plotting the ages, I'll apply `floor` to round down:

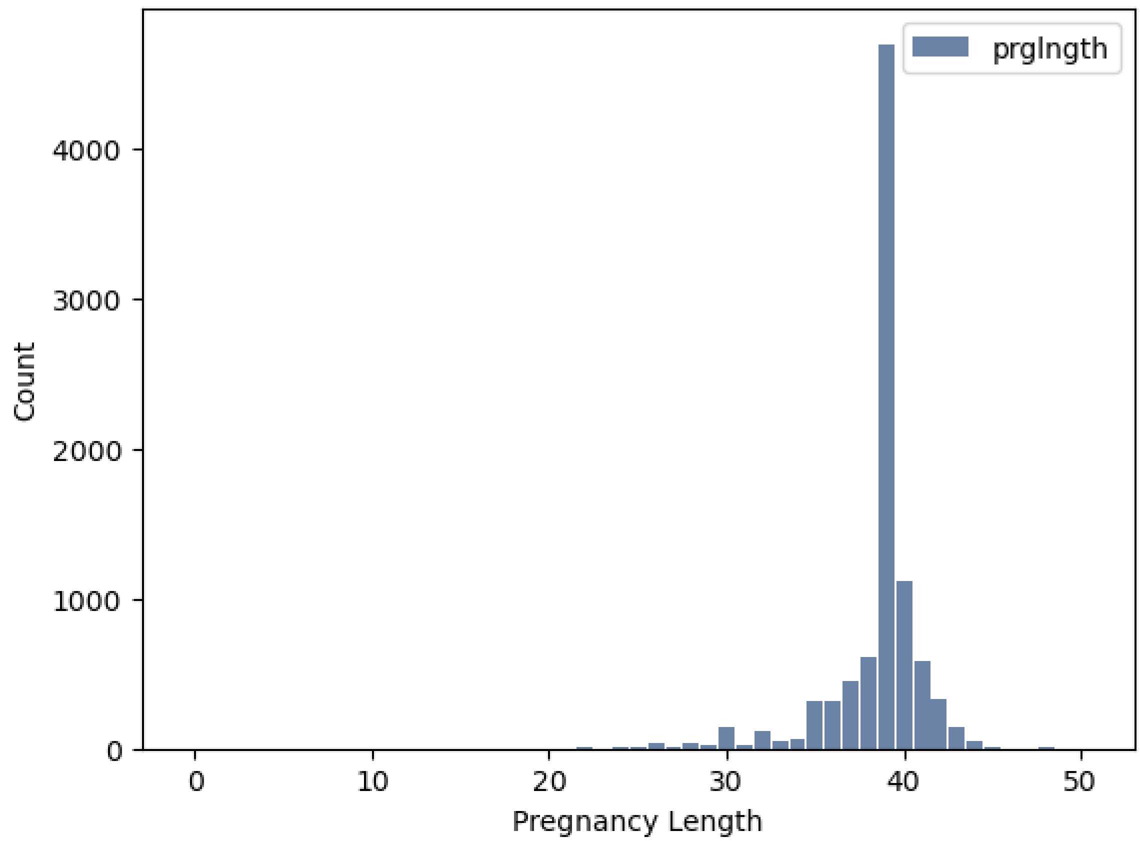
In [18]:
```
ages = np.floor(live.agepreg)
```

In [19]:
```
hist = thinkstats2.Hist(ages, label='agepreg')
thinkplot.Hist(hist)
thinkplot.Config(xlabel='years', ylabel='Count')
```



As an exercise, plot the histogram of pregnancy lengths (column `prglngth`).

In [20]:
```
hist = thinkstats2.Hist(live.prglngth, label='prglngth')
thinkplot.Hist(hist)
thinkplot.Config(xlabel = 'Pregnancy Length', ylabel = 'Count')
```



`Hist` provides smallest, which select the lowest values and their frequencies.

In [21]:
```
for weeks, freq in hist.Smallest(10):
    print(weeks, freq)
```

```
0 1
4 1
9 1
13 1
17 2
18 1
19 1
20 1
21 2
22 7
```

Use `Largest` to display the longest pregnancy lengths.

In [22]:
```
for weeks, freq in hist.Largest(10):
    print(weeks, freq)
```

```
50 2
48 7
47 1
46 1
45 10
44 46
43 148
42 328
41 587
40 1116
```

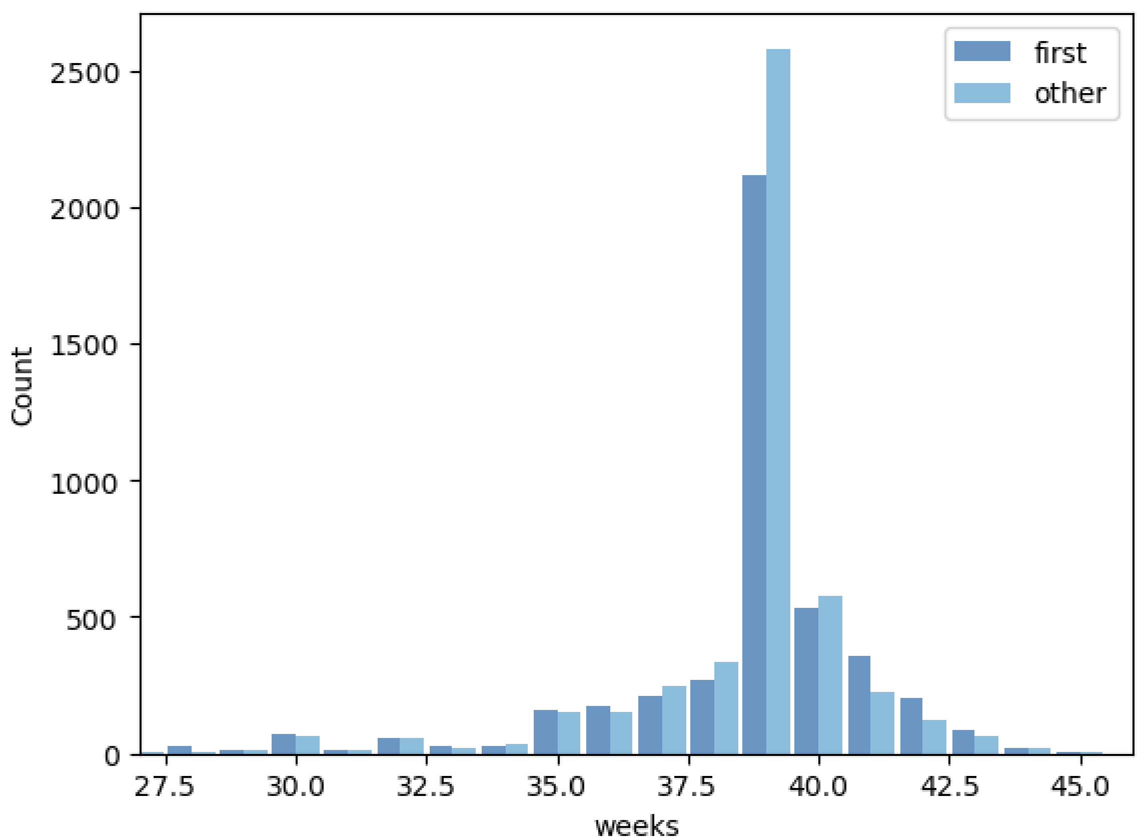
From live births, we can select first babies and others using `birthord`, then compute histograms of pregnancy length for the two groups.

In [23]:
```
firsts = live[live.birthord == 1]
others = live[live.birthord != 1]

first_hist = thinkstats2.Hist(firsts.prglngth, label='first')
other_hist = thinkstats2.Hist(others.prglngth, label='other')
```

We can use `width` and `align` to plot two histograms side-by-side.

In [24]:
```
width = 0.45
thinkplot.PrePlot(2)
thinkplot.Hist(first_hist, align='right', width=width)
thinkplot.Hist(other_hist, align='left', width=width)
thinkplot.Config(xlabel='weeks', ylabel='Count', xlim=[27, 46])
```



`Series` provides methods to compute summary statistics:

In [25]:
```
mean = live.prglngth.mean()
var = live.prglngth.var()
std = live.prglngth.std()
```

Here are the mean and standard deviation:

In [26]:
```
mean, std
```

Out[26]:
```
(38.56055968517709, 2.702343810070593)
```

As an exercise, confirm that `std` is the square root of `var`:

In [27]:
```
std == np.sqrt(var)
```

Out[27]:
```
True
```

Here's are the mean pregnancy lengths for first babies and others:

In [28]:
```
firsts.prglngth.mean(), others.prglngth.mean()
```

Out[28]:
```
(38.60095173351461, 38.52291446673706)
```

And here's the difference (in weeks):

In [29]:
```
firsts.prglngth.mean() - others.prglngth.mean()
```

Out[29]:
```
0.07803726677754952
```

This functon computes the Cohen effect size, which is the difference in means expressed in number of standard deviations:

In [30]:
```
def CohenEffectSize(group1, group2):
    """Computes Cohen's effect size for two groups.

    group1: Series or DataFrame
    group2: Series or DataFrame

    returns: float if the arguments are Series;
             Series if the arguments are DataFrames
    """
    diff = group1.mean() - group2.mean()

    var1 = group1.var()
    var2 = group2.var()
    n1, n2 = len(group1), len(group2)

    pooled_var = (n1 * var1 + n2 * var2) / (n1 + n2)
    d = diff / np.sqrt(pooled_var)
    return d
```

Compute the Cohen effect size for the difference in pregnancy length for first babies and others.

In [31]:
```
CohenEffectSize(firsts.prglngth, others.prglngth)
```

Out[31]:
```
0.028879044654449883
```

Exercises

Using the variable `totalwgt_lb`, investigate whether first babies are lighter or heavier than others.

Compute Cohen’s effect size to quantify the difference between the groups. How does it compare to the difference in pregnancy length?

In [32]:
```
firsts.totalwgt_lb.mean() - others.totalwgt_lb.mean()
print("First babies on an average are 0.124lbs lighter than others.")
```

```
First babies on an average are 0.124lbs lighter than others.
```

In [33]:
```
CohenEffectSize(firsts.totalwgt_lb, others.totalwgt_lb)
```

Out[33]:
```
-0.088672927072602
```

For the next few exercises, we'll load the respondent file:

In [ ]:
```
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/2002FemResp.dc
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/2002FemResp.da
```

In [ ]:
```
resp = nsfg.ReadFemResp()
```

Make a histogram of `totincr` the total income for the respondent's family. To interpret the codes see the codebook.

In [ ]:
```
hist = thinkstats2.Hist(resp.totincr, label='totincr')
thinkplot.Hist(hist)
thinkplot.Config(xlabel = 'Total Income Category', ylabel = 'Count')
```

In [ ]:
```
test = resp[resp.totincr < 5000]
sum(test.totincr)
test.count(), resp.totincr.count(), resp.totincr.min(), resp.totincr.max()
```

Make a histogram of `age_r`, the respondent's age at the time of interview.

In [ ]:
```
hist = thinkstats2.Hist(resp.age_r, label='age_r')
thinkplot.Hist(hist)
thinkplot.Config(xlabel = 'Age', ylabel = 'Count')
```

Make a histogram of `numfmhh`, the number of people in the respondent's household.

In [ ]:
```
hist = thinkstats2.Hist(resp.numfmhh, label='numfmhh')
```

```
thinkplot.Hist(hist)
thinkplot.Config(xlabel = 'Number of People', ylabel = 'Count')
```

Make a histogram of `parity`, the number of children borne by the respondent. How would you describe this distribution?

In [ ]:
```
hist = thinkstats2.Hist(resp.parity, label='parity')
thinkplot.Hist(hist)
thinkplot.Config(xlabel = 'Parity', ylabel = 'Count')
"Live births and number of children are inversely proprotional. As the number of ch
```

Use Hist.Largest to find the largest values of `parity`.

In [ ]:
```
hist.Largest()
```

In [ ]:
```
# hist.Smallest()
```

Let's investigate whether people with higher income have higher parity. Keep in mind that in this study, we are observing different people at different times during their lives, so this data is not the best choice for answering this question. But for now let's take it at face value.

Use `totincr` to select the respondents with the highest income (level 14). Plot the histogram of `parity` for just the high income respondents.

In [ ]:
```
highincome = resp[resp.totincr == 14]
hist = thinkstats2.Hist(highincome.parity, label='parity')
thinkplot.Hist(hist)
thinkplot.Config(xlabel = 'Parity', ylabel = 'Count')
```

Find the largest parities for high income respondents.

In [ ]:
```
hist.Largest()
```

Compare the mean `parity` for high income respondents and others.

In [ ]:
```
otherincome  = resp[resp.totincr != 14]
#otherincome.count()
highincome.parity.mean(), otherincome.parity.mean()
```

Compute the Cohen effect size for this difference. How does it compare with the difference in pregnancy length for first babies and others?

In [ ]:
```
CohenEffectSize(highincome.parity, otherincome.parity)
```

In [ ]:
```
"This effect is about 10 times stronger than the difference in pregnancy length. Bu
 0.028879044654449883
-0.1251185531466061
```