

## W3 Lab: Perception

In this lab, we will learn basic usage of `pandas` library and then perform a small experiment to test the perception of length and area.

In [ ]:

```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

## Vega datasets

Before going into the perception experiment, let's first talk about some handy datasets that you can play with.

It's nice to have clean datasets handy to practice data visualization. There is a nice small package called `vega-datasets`, from the [altair project](#).

You can install the package by running

```
$ pip install vega-datasets
```

or

```
$ pip3 install vega-datasets
```

Once you install the package, you can import and see the list of datasets:

In [118]:

```
#installing vega-datasets
!pip install vega-datasets
```

```
Requirement already satisfied: vega-datasets in /usr/local/lib/python3.7/dist-packages (0.9.0)
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages (from vega-datasets) (1.1.5)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas->vega-datasets) (2.8.1)
Requirement already satisfied: numpy>=1.15.4 in /usr/local/lib/python3.7/dist-packages (from pandas->vega-datasets) (1.19.5)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packages (from pandas->vega-datasets) (2018.9)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.7.3->pandas->vega-datasets) (1.15.0)
```

In [119]:

```
from vega_datasets import data

data.list_datasets()
```

Out[119]:

```
['7zip',
 'airports',
 'annual-precip',
 'anscombe',
 'barley',
 'birdstrikes',
 'budget',
 'budgets',
 'burtin',
```

```

'cars',
'climate',
'co2-concentration',
'countries',
'crimea',
'disasters',
'driving',
'earthquakes',
'ffox',
'flare',
'flare-dependencies',
'flights-10k',
'flights-200k',
'flights-20k',
'flights-2k',
'flights-3m',
'flights-5k',
'flights-airport',
'gapminder',
'gapminder-health-income',
'gimp',
'github',
'graticule',
'income',
'iowa-electricity',
'iris',
'jobs',
'la-riots',
'londonBoroughs',
'londonCentroids',
'londonTubeLines',
'lookup_groups',
'lookup_people',
'miserables',
'monarchs',
'movies',
'normal-2d',
'obesity',
'ohlc',
'points',
'population',
'population_engineers_hurricanes',
'seattle-temps',
'seattle-weather',
'sf-temps',
'sp500',
'stocks',
'udistrict',
'unemployment',
'unemployment-across-industries',
'uniform-2d',
'us-10m',
'us-employment',
'us-state-capitals',
'volcano',
'weather',
'webball26',
'wheat',
'windvectors',
'world-110m',
'zipcodes']

```

**or you can work with only smaller, local datasets.**

In [120]:

```

from vega_datasets import local_data
local_data.list_datasets()

```

Out[120]:

```

['airports',

```

```
['airports',
 'anscombe',
 'barley',
 'burtin',
 'cars',
 'crimea',
 'driving',
 'iowa-electricity',
 'iris',
 'la-riots',
 'ohlc',
 'seattle-temps',
 'seattle-weather',
 'sf-temps',
 'stocks',
 'us-employment',
 'wheat']
```

Ah, we have the `anscombe` data here! Let's see the description of the dataset.

In [ ]:

```
local_data.anscombe.description
```

Out[ ]:

"Anscombe's Quartet is a famous dataset constructed by Francis Anscombe [1]\_. Common summary statistics are identical for each subset of the data, despite the subsets having vastly different characteristics."

## Anscombe's quartet dataset

How does the actual data look like? Very conveniently, calling the dataset returns a Pandas dataframe for you.

In [ ]:

```
df = local_data.anscombe()
df.head()
```

Out[ ]:

	Series	X	Y
0	I	10	8.04
1	I	8	6.95
2	I	13	7.58
3	I	9	8.81
4	I	11	8.33

**Q1: can you draw a scatterplot of the dataset "I"? You can filter the dataframe based on the `Series` column and use `plot` function that you used for the Snow's map.**

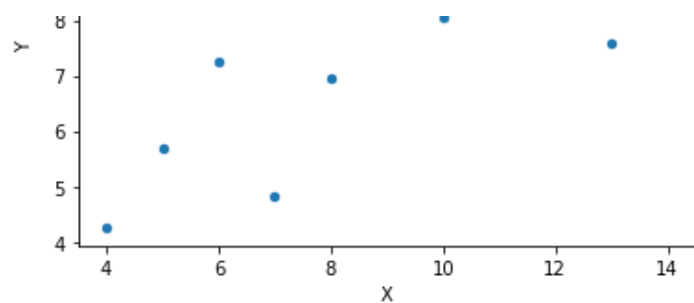
In [59]:

```
# TODO: put your code here
df[df.Series=="I"].plot(x='X', y='Y', kind='scatter')
```

Out[59]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7f9ec6829450>





# Some histograms with pandas

Let's look at a slightly more complicated dataset.

In [ ]:

```
car_df = local_data.cars()
car_df.head()
```

Out [ ]:

	Name	Miles_per_Gallon	Cylinders	Displacement	Horsepower	Weight_in_lbs	Acceleration	Year	Origin
0	chevrolet chevelle malibu	18.0	8	307.0	130.0	3504	12.0	1970-01-01	USA
1	buick skylark 320	15.0	8	350.0	165.0	3693	11.5	1970-01-01	USA
2	plymouth satellite	18.0	8	318.0	150.0	3436	11.0	1970-01-01	USA
3	amc rebel sst	16.0	8	304.0	150.0	3433	12.0	1970-01-01	USA
4	ford torino	17.0	8	302.0	140.0	3449	10.5	1970-01-01	USA

Pandas provides useful summary functions. It identifies numerical data columns and provides you with a table of summary statistics.

In [ ]:

```
car_df.describe()
```

Out [ ]:

	Miles_per_Gallon	Cylinders	Displacement	Horsepower	Weight_in_lbs	Acceleration
count	398.000000	406.000000	406.000000	400.000000	406.000000	406.000000
mean	23.514573	5.475369	194.779557	105.082500	2979.413793	15.519704
std	7.815984	1.712160	104.922458	38.768779	847.004328	2.803359
min	9.000000	3.000000	68.000000	46.000000	1613.000000	8.000000
25%	17.500000	4.000000	105.000000	75.750000	2226.500000	13.700000
50%	23.000000	4.000000	151.000000	95.000000	2822.500000	15.500000
75%	29.000000	8.000000	302.000000	130.000000	3618.250000	17.175000
max	46.600000	8.000000	455.000000	230.000000	5140.000000	24.800000

If you ask to draw a histogram, you get all of them. :)

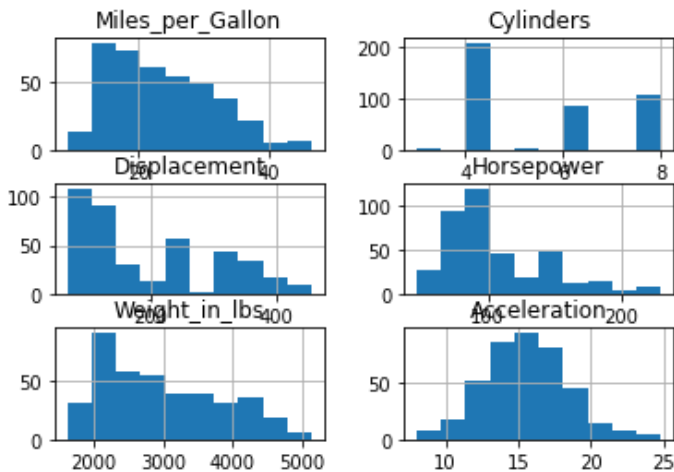
In [ ]:

```
car_df.hist()
```

Out [ ]:

Out[ ]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f9ec7bc7950>,  
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f9ec76f3290>],  
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f9ec76ffa10>,  
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f9ec770f750>],  
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f9ec771f550>,  
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f9ec76aed10>]],  
      dtype=object)
```

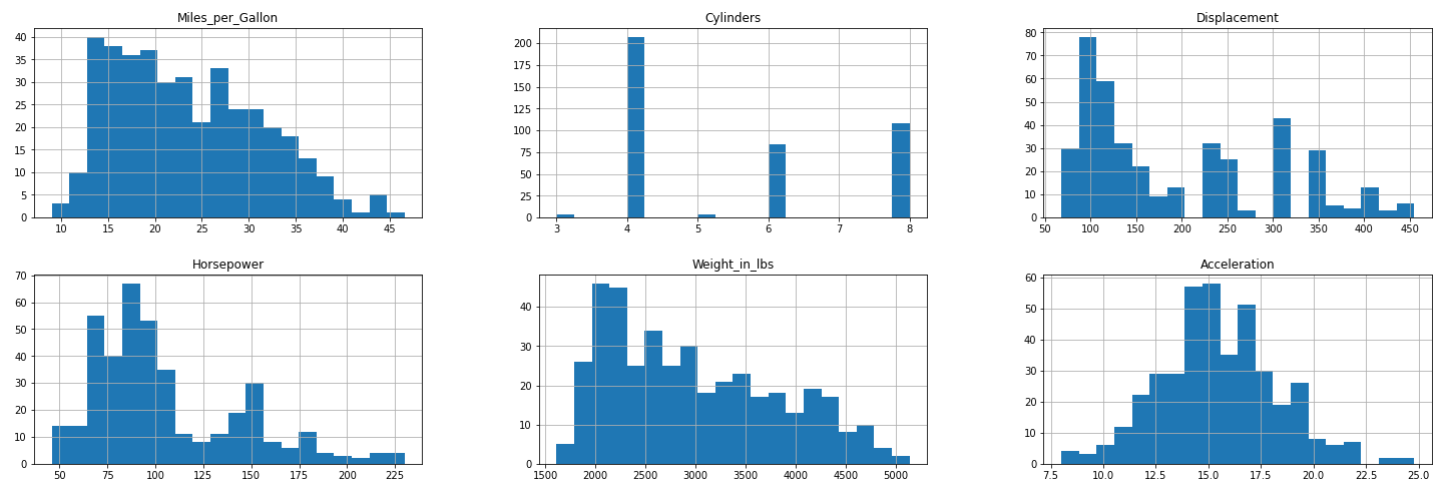


Well this is too small. You can check out [the documentation](#) and change the size of the figure.

**Q2: by consulting the documentation, can you make the figure larger so that we can see all the labels clearly? And then make the layout 2 x 3 not 3 x 2, then change the number of bins to 20?**

In [63]:

```
# TODO: put your code here  
fig = car_df.hist(figsize = (25,8), layout=(2,3), bins=20)
```



## Your own psychophysics experiment!

Let's do an experiment! The procedure is as follows:

1. Generate a random number between [1, 10];
2. Use a horizontal bar to represent the number, i.e., the length of the bar is equal to the number;
3. Guess the length of the bar by comparing it to two other bars with length 1 and 10 respectively;
4. Store your guess (perceived length) and actual length to two separate lists;
5. Repeat the above steps many times;
6. How does the perception of length differ from that of area?.

First, let's define the length of a short and a long bar. We also create two empty lists to store perceived and actual length.

In [ ]:

```
import random
import time
import numpy as np

l_short_bar = 1
l_long_bar = 10

perceived_length_list = []
actual_length_list = []
```

## Perception of length

Let's run the experiment.

The `random` module in Python provides various random number generators, and the `random.uniform(a,b)` function returns a floating point number in `[a,b]`.

We can plot horizontal bars using the `pyplot.barh()` function. Using this function, we can produce a bar graph that looks like this:

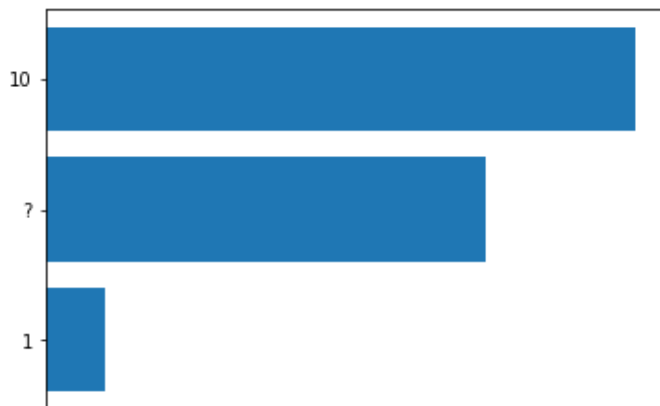
In [ ]:

```
mystery_length = random.uniform(1, 10) # generate a number between 1 and 10. this is the
                                         *actual* length.

plt.barh(np.arange(3), [l_short_bar, mystery_length, l_long_bar], align='center')
plt.yticks(np.arange(3), ('1', '?', '10'))
plt.xticks([]) # no hint!
```

Out [ ]:

([], <a list of 0 Text major ticklabel objects>)



Btw, `np.arange` is used to create a simple integer list `[0, 1, 2]`.

In [ ]:

```
np.arange(3)
```

Out [ ]:

```
array([0, 1, 2])
```

Now let's define a function to perform the experiment once. When you run this function, it picks a random number between 1.0 and 10.0 and show the bar chart. Then it asks you to input your estimate of the length of the middle bar. It then saves that number to the `perceived_length_list` and the actual answer to the `actual_length_list`.

In [ ]:

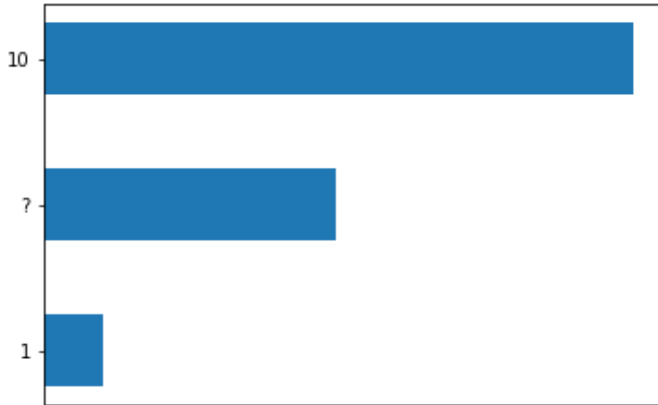
```
def run_exp_once():
    mystery_length = random.uniform(1, 10) # generate a number between 1 and 10.
```

```
plt.barh(np.arange(3), [l_short_bar, mystery_length, l_long_bar], height=0.5, align='center')
plt.yticks(np.arange(3), ('1', '?', '10'))
plt.xticks([]) # no hint!
plt.show()

perceived_length_list.append( float(input()) )
actual_length_list.append(mystery_length)
```

In [ ]:

```
run_exp_once()
```



1

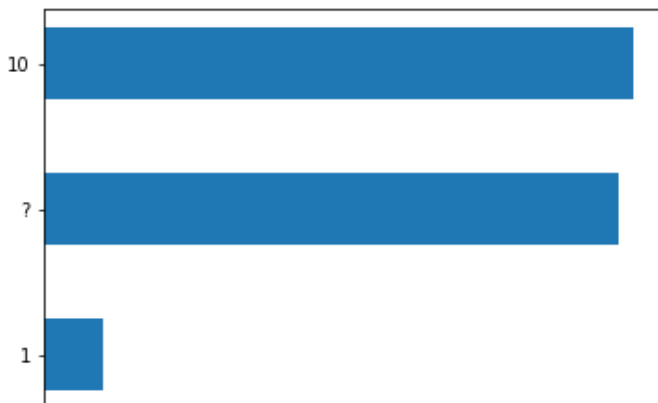
**Now, run the experiment many times to gather your data. Check the two lists to make sure that you have the proper dataset. The length of the two lists should be the same.**

In [64]:

```
# TODO: Run your experiment many times here

perceived_length_list = []
actual_length_list = []
run_exp_once()
run_exp_once()
run_exp_once()
run_exp_once()
run_exp_once()

print("Perceived " + str(len(perceived_length_list)))
print("Actual " + str(len(actual_length_list)))
# Total should be 6 after running the above lines once and the list of both perceived and actual are the same length
```

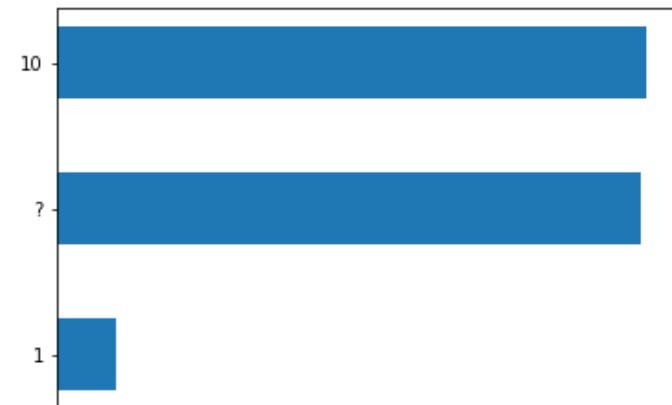


6

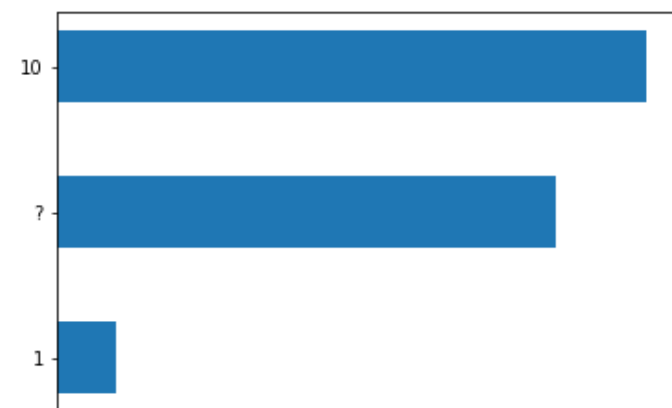




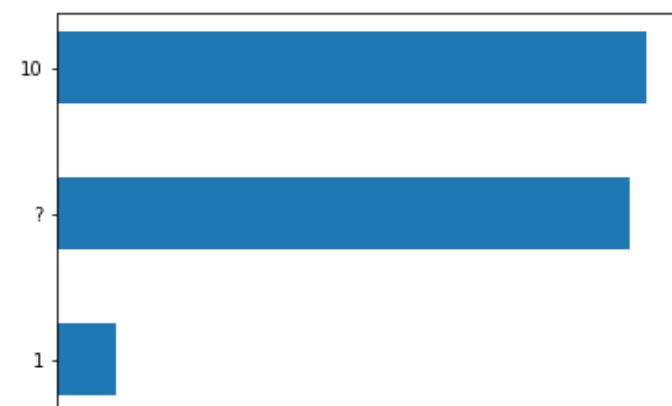
4



8



6



3

Perceived 5  
Actual 5

## Plotting the result

Now we can draw the scatter plot of perceived and actual length. The `matplotlib`'s `scatter()` function will do this. This is the backend of the pandas' scatterplot. Here is an example of how to use `scatter`:

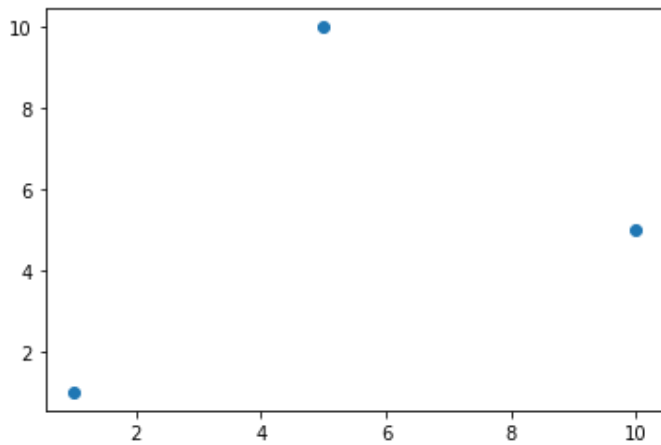


```
In [ ]:
```

```
plt.scatter(x=[1,5,10], y=[1,10, 5])
```

```
Out[ ]:
```

```
<matplotlib.collections.PathCollection at 0x7f9ec6ebf1d0>
```



**Q3: Now plot your result using the `scatter()` function. You should also use `plt.title()`, `plt.xlabel()`, and `plt.ylabel()` to label your axes and the plot itself.**

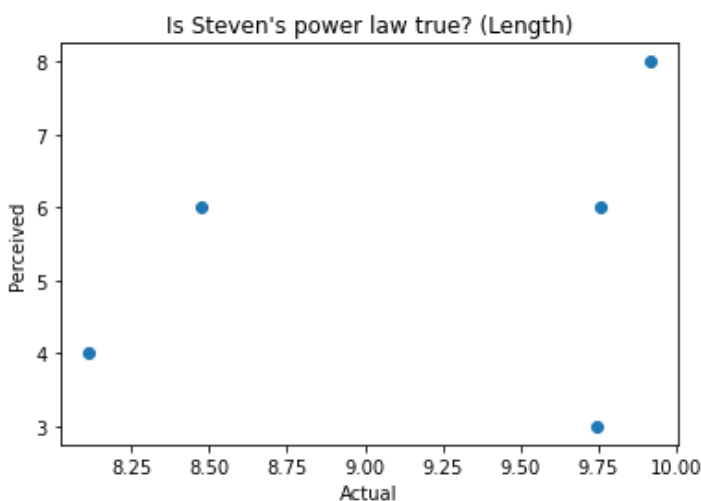
```
In [65]:
```

```
# TODO: put your code here
x = actual_length_list
y = perceived_length_list

plt.scatter(x, y)
plt.title("Is Steven's power law true? (Length)")
plt.xlabel('Actual')
plt.ylabel('Perceived')
```

```
Out[65]:
```

```
Text(0, 0.5, 'Perceived')
```



After plotting, let's fit the relation between actual and perceived lengths using a polynomial function. We can easily do it using `curve_fit(f, x, y)` in Scipy, which is to fit  $x$  and  $y$  using the function  $f$ . In our case,  $f = a * x^b$ . For instance, we can check whether this works by creating a fake dataset that follows the exact form:

```
In [ ]:
```

```
from scipy.optimize import curve_fit

def func(x, a, b, c):
```

```

    return a * np.power(x, b) + c

x = np.arange(20) # [0,1,2,3, ..., 19]
y = np.power(x, 2) # [0,1,4,9, ... ]

popt, pcov = curve_fit(func, x, y)
print('{:.2f} x^{:.2f} + {:.2f}'.format(*popt))

1.00 x^2.00 + -0.00

```

In [ ]:

**Q4: Now fit your data! Do you see roughly linear relationship between the actual and the perceived lengths? It's ok if you don't!**

In [69]:

```

#I do not see a relationship as preceived input is based on user input and the preceived
input could be anything between 1-10 for any iteration.
# TODO: your code here
#Using the above function and x the actual length and y the perceived length the output f
unction can be calculated
#The output is subject to change if started over as the list values(x,y) would be differe
nt meaning the relationship is only associated with this round of data
def func(x, a, b):
    return a * x + b

x = actual_length_list
y = perceived_length_list
popt, pcov = curve_fit(func, x, y)
print('{:.2f} * x + {:.2f}'.format(*popt))

0.70 * x + -1.08

```

## Perception of area

Similar to the above experiment, we now represent a random number as a circle, and the area of the circle is equal to the number.

First, calculate the radius of a circle from its area and then plot using the `Circle()` function.

`plt.Circle((0,0), r)` will plot a circle centered at (0,0) with radius `r`.

In [ ]:

```

n1 = 0.005
n2 = 0.05

radius1 = np.sqrt(n1/np.pi) # area = pi * r * r
radius2 = np.sqrt(n2/np.pi)
random_radius = np.sqrt(n1*random.uniform(1,10)/np.pi)

plt.axis('equal')
plt.axis('off')
circ1 = plt.Circle( (0,0), radius1, clip_on=False )
circ2 = plt.Circle( (4*radius2,0), radius2, clip_on=False )
rand_circ = plt.Circle((2*radius2,0), random_radius, clip_on=False )

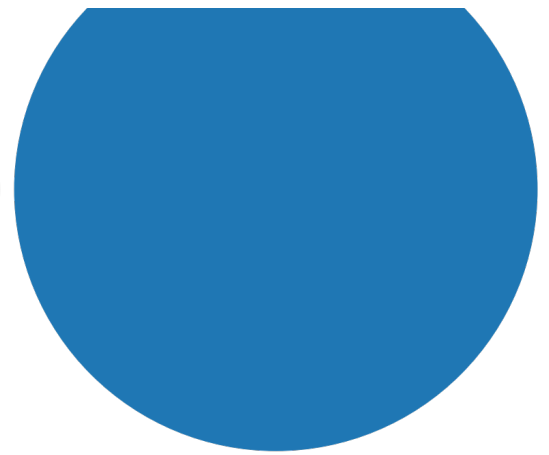
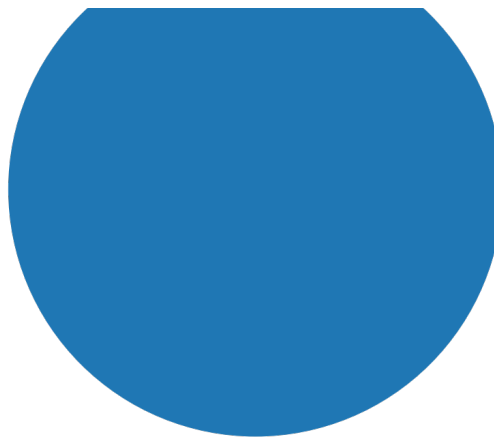
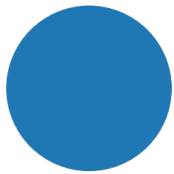
plt.gca().add_artist(circ1)
plt.gca().add_artist(circ2)
plt.gca().add_artist(rand_circ)

```

Out[ ]:

<matplotlib.patches.Circle at 0x7f9ec6d91dd0>





Let's have two lists for this experiment.

In [166]:

```
perceived_area_list = []  
actual_area_list= []
```

And define a function for the experiment.

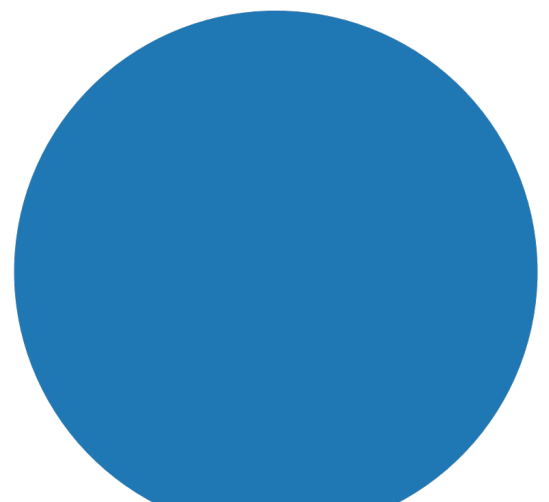
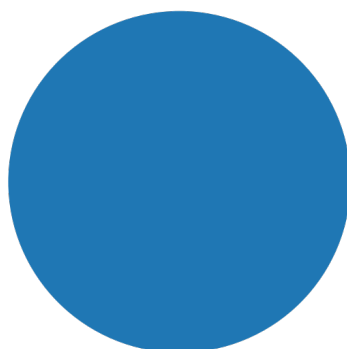
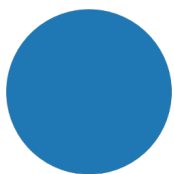
In [167]:

```
def run_area_exp_once(n1=0.005, n2=0.05):  
    radius1 = np.sqrt(n1/np.pi) # area = pi * r * r  
    radius2 = np.sqrt(n2/np.pi)  
  
    mystery_number = random.uniform(1,10)  
    random_radius = np.sqrt(n1*mystery_number/math.pi)  
  
    plt.axis('equal')  
    plt.axis('off')  
    circ1 = plt.Circle( (0,0), radius1, clip_on=False )  
    circ2 = plt.Circle( (4*radius2,0), radius2, clip_on=False )  
    rand_circ = plt.Circle((2*radius2,0), random_radius, clip_on=False )  
    plt.gca().add_artist(circ1)  
    plt.gca().add_artist(circ2)  
    plt.gca().add_artist(rand_circ)  
    plt.show()  
  
    perceived_area_list.append( float(input()) )  
    actual_area_list.append(mystery_number)
```

**Q5: Now you can run the experiment many times, plot the result, and fit a power-law curve!**

In [180]:

```
# TODO: put your code here. You can use multiple cells.  
run_area_exp_once()
```



3

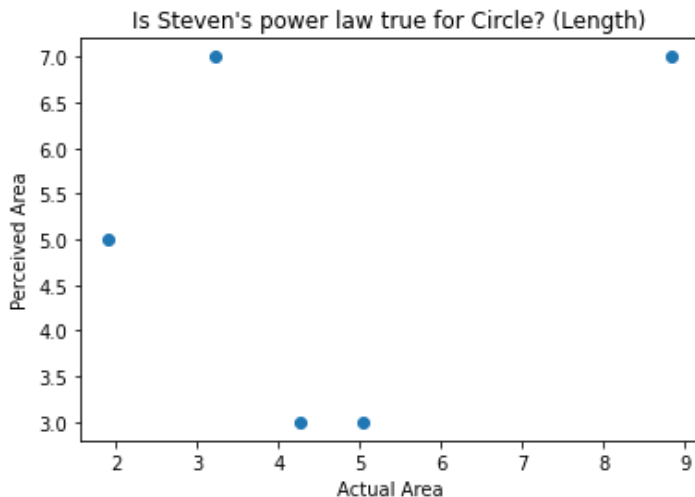
In [181]:

```
# TODO: put your code here
x = actual_area_list
y = perceived_area_list

plt.scatter(x, y)
plt.title("Is Steven's power law true for Circle? (Length)")
plt.xlabel('Actual Area')
plt.ylabel('Perceived Area')
```

Out[181]:

Text(0, 0.5, 'Perceived Area')



In [189]:

```
from scipy.optimize import curve_fit

def func(x, a, b, c):
    return a * np.power(x, b) + c

p0=[50000,-1,0]

popt, pcov = curve_fit(func, actual_area_list, perceived_area_list, p0)

print('{:.2f} x^{:.2f} + {:.2f}'.format(*popt))
```

1.40 x<sup>-2.89</sup> + 4.94

In [196]:

```
perceived_area_list = []
actual_area_list= []
```

In [204]:

```
#Question 5 2nd experiment
# TODO: put your code here. You can use multiple cells.
run_area_exp_once()
```

8

In [211]:

```
print(perceived_area_list)
print(actual_area_list)
```

```
[4.0, 6.0, 8.0]
[4.370584866878281, 5.709135263037376, 9.15443573954767]
```

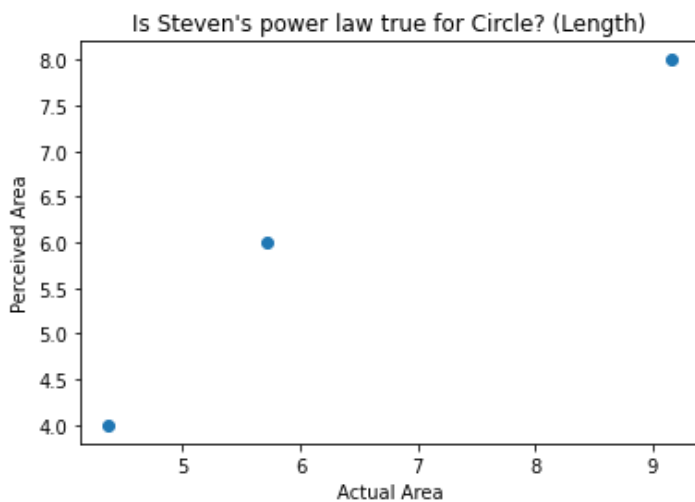
In [206]:

```
# TODO: put your code here
x = actual_area_list
y = perceived_area_list

plt.scatter(x, y)
plt.title("Is Steven's power law true for Circle? (Length)")
plt.xlabel('Actual Area')
plt.ylabel('Perceived Area')
```

Out[206]:

Text(0, 0.5, 'Perceived Area')



In [212]:

```
from scipy.optimize import curve_fit

def func(x, a, b, c):
    return a * np.power(x, b) + c

p0=[50000,-1,0]

popt, pcov = curve_fit(func, actual_area_list, perceived_area_list, p0)

print('{:.2f} x^{:.2f} + {:.2f}'.format(*popt))
```

-59.89 x<sup>-1.58</sup> + 9.80

/usr/local/lib/python3.7/dist-packages/scipy/optimize/minpack.py:808: OptimizeWarning: Covariance of the parameters could not be estimated  
category=OptimizeWarning)

**What is your result? How are the exponents different from each other?**

The output for question 5 using `curve_fit()` was:  $1.40 x^{-2.89} + 4.94$  and  $-59.89 x^{-1.58} + 9.80$

The above were the results that I attained, the exponents are different as this is a different plot using a specialized function and is found to be negative for both of them vs just the lengths, where linear regression is used. Three points were needed for `curve_fit()`. Adding `p0` alleviated `RuntimeError : Optimal parameters not found: Number of calls to function has reached maxfev = 1000`, which may have effected the function output.