W3 Lab: Perception

In [1]:
```
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. For Python, there is a package called `vega-datasets` (https://github.com/altair-viz/vega_datasets), from the altair project (https://github.com/altair-viz). You can install the package by running

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

or

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

or

```
$ conda install vega-datasets
```

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

In [2]:
```
import altair as alt
```

In [5]:
```
from vega_datasets import data

data.list_datasets()
```

```
Out[5]: ['7zip',
         'airports',
         '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',
         'points',
         'population',
         'population_engineers_hurricanes',
         'seattle-temps',
         'seattle-weather',
         'sf-temps',
         'sp500',
         'stocks',
         'udistrict',
         'unemployment',
         'unemployment-across-industries',
         'us-10m',
         'us-state-capitals',
         'weather',
         'weball26',
```

```
 'wheat',
 'world-110m',
 'zipcodes']
```

or you can work with only smaller, local datasets.

In [3]:
```
from vega_datasets import local_data
local_data.list_datasets()
```

Out[3]:
```
['airports',
 'anscombe',
 'barley',
 'burtin',
 'cars',
 'crimea',
 'driving',
 'iowa-electricity',
 'iris',
 'la-riots',
 'seattle-temps',
 'seattle-weather',
 'sf-temps',
 'stocks']
```

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

In [4]:
```
local_data.anscombe.description
```

Out[4]:
```
"Anscombe's Quartet is a famous dataset constructed by Francis Anscombe [1]_. Common su
mmary statistics are identical for each subset of the data, despite the subsets having
vastly different characteristics."
```

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

In [5]:
```
df = local_data.anscombe()
df.head()
```

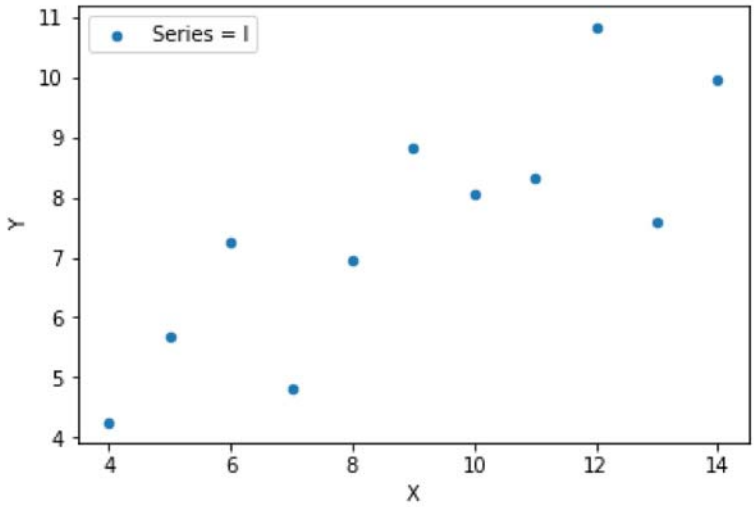
Out[5]:

	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 [6]:
```
# TODO: put your code here
#df.shape #(44, 3)
#df.describe()

%matplotlib inline

#filter Series=I
is_I =df['Series']=='I'

#create df based on is_I list
df_I = df[is_I]

df_I.plot(x='X', y='Y', kind='scatter', label='Series = I')

#check values
print(df_I.head())
```

```
  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
```



Some histograms with pandas

Let's look at a slightly more complicated dataset.

In [7]:
```
car_df = local_data.cars()
car_df.head()
```

Out[7]:

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

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

In [8]:
```
car_df.describe()
```

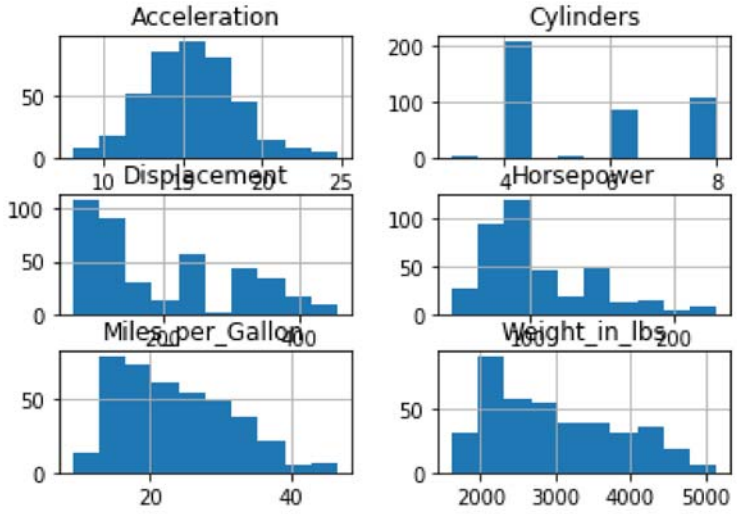
Out[8]:

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

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

```
In [9]: car_df.hist()
```

```
Out[9]: array([[<matplotlib.axes._subplots.AxesSubplot object at 0x0000016AC8BD1128>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x0000016AC8BF0F60>],
               [<matplotlib.axes._subplots.AxesSubplot object at 0x0000016AC8C15D30>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x0000016AC8C6D208>],
               [<matplotlib.axes._subplots.AxesSubplot object at 0x0000016AC8C94518>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x0000016AC8C94550>]],
              dtype=object)
```

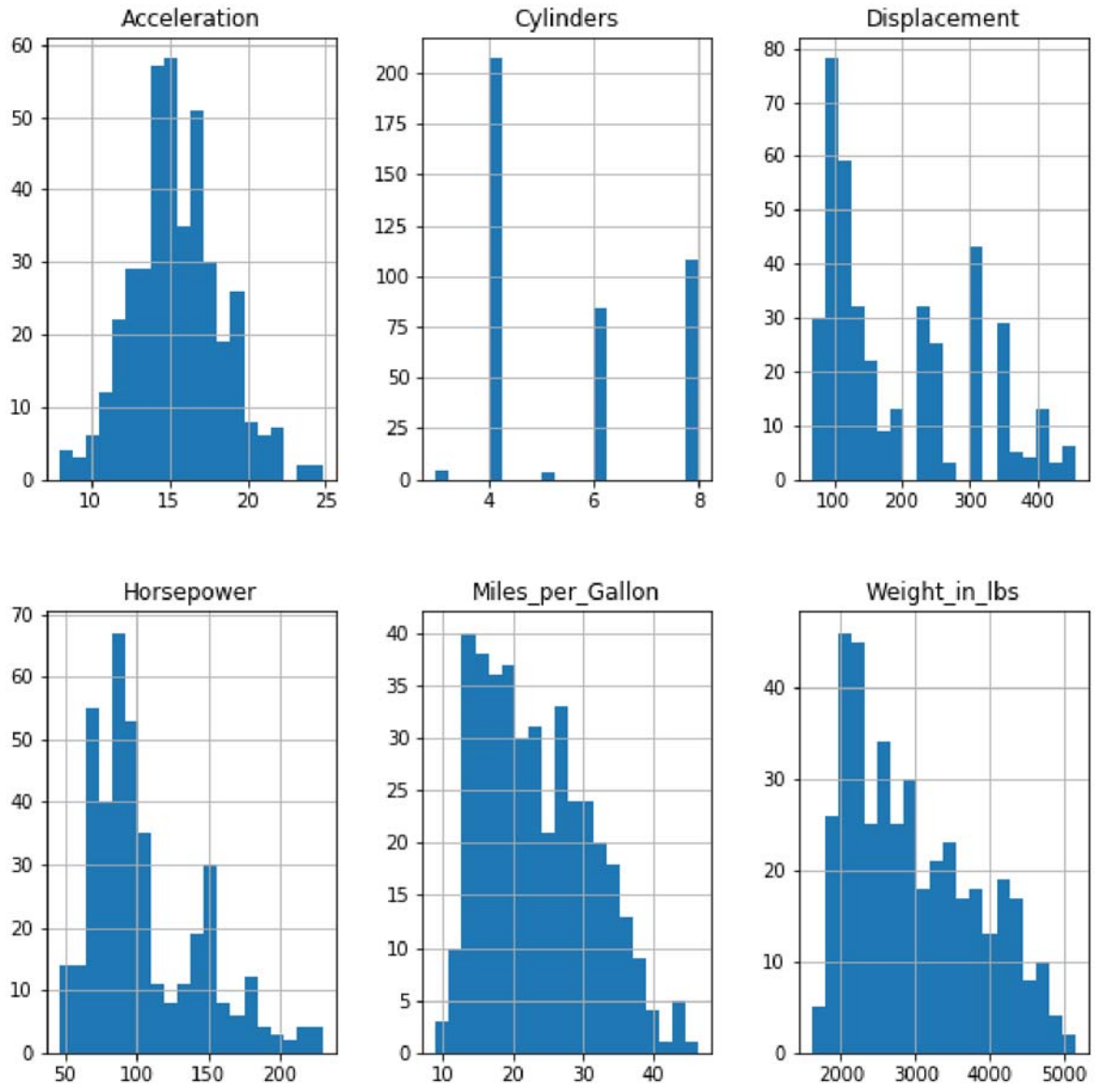


Well this is too small. You can check out the documentation (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.hist.html) and change the size of the figure.

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 [10]:
```
# TODO: put your code here

%matplotlib inline
car_df.hist(layout=(2,3), bins=20, figsize=(10,10))
```

Out[10]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x0000016AC8D51DD8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000016AC8E16FD0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000016AC9E12668>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000016AC9E38CF8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000016AC9E6A3C8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000016AC9E6A400>]],
      dtype=object)
```



Stevens' power-law and 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. Check whether Steven's power-law holds.

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 [13]:
```
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`** (https://docs.python.org/3.6/library/random.html) module in Python provides various random number generators, and the **`random.uniform(a,b)`** (https://docs.python.org/3.6/library/random.html#random.uniform) function returns a floating point number in [a,b].

We can plot horizontal bars using the **`pyplot.barh()`** (http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.barh) function. Using this function, we can produce a bar graph that looks like this:

In [15]:
```
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], align='center')
plt.yticks(np.arange(3), ('1', '?', '10'))
plt.xticks([]) # no hint!
```

Out[15]:
```
([], <a list of 0 Text xticklabel objects>)
```

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

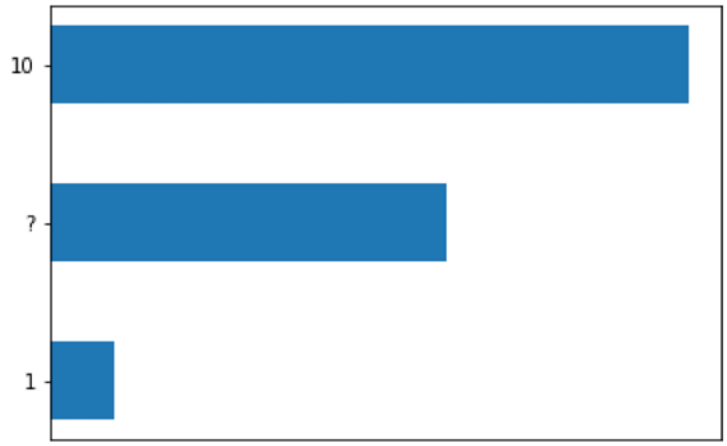
In [16]:
```
np.arange(3)
```

Out[16]:
```
array([0, 1, 2])
```

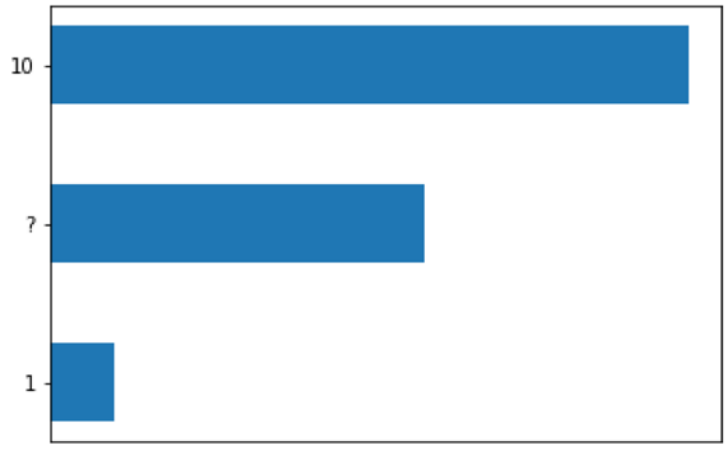
Now let's define a function to perform the experiment once. When you run this function, it picks a random number between 1.0 and 10.0 and show the bar chart. Then it asks you to input your estimate of the length of the middle bar. It then saves that number to the `perceived_length_list` and the actual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

In [17]:
```
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)
```

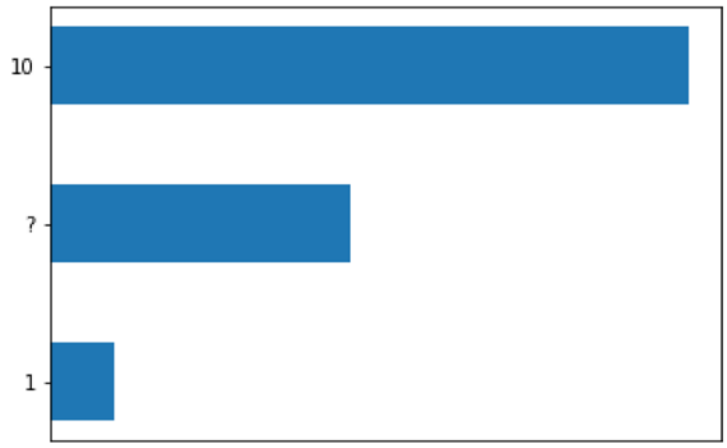
In [18]:
```
for i in range(10):
    run_exp_once()
```



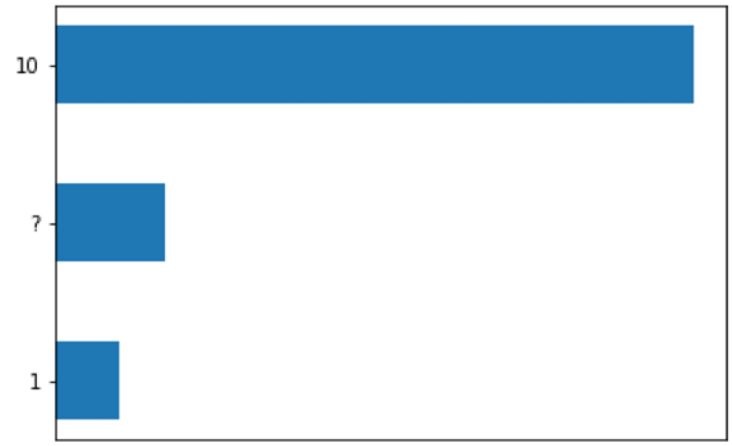
5.90000



5.8700

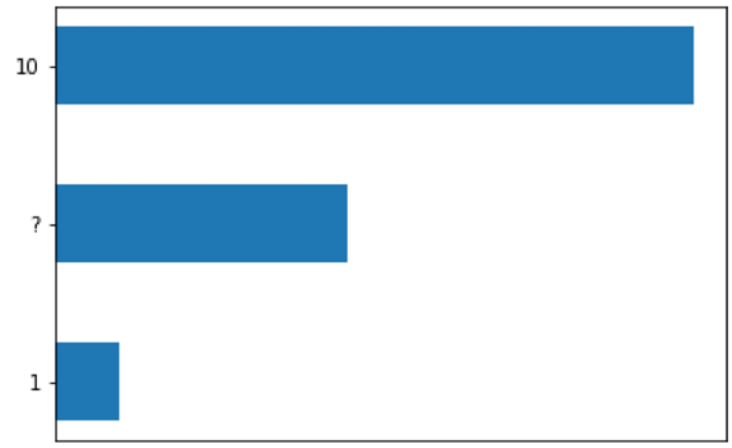


4.5555

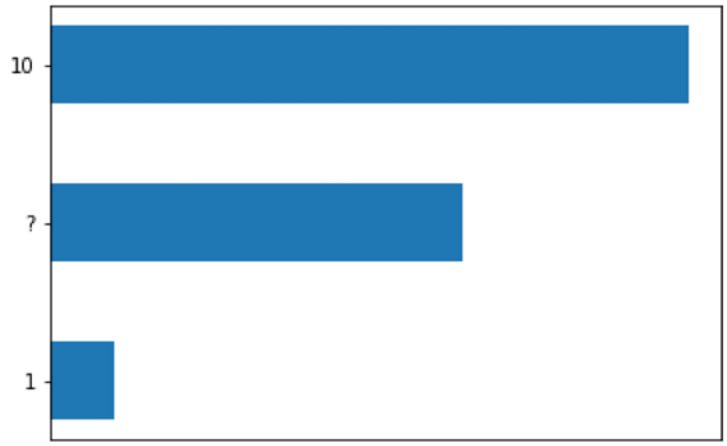


1.73333

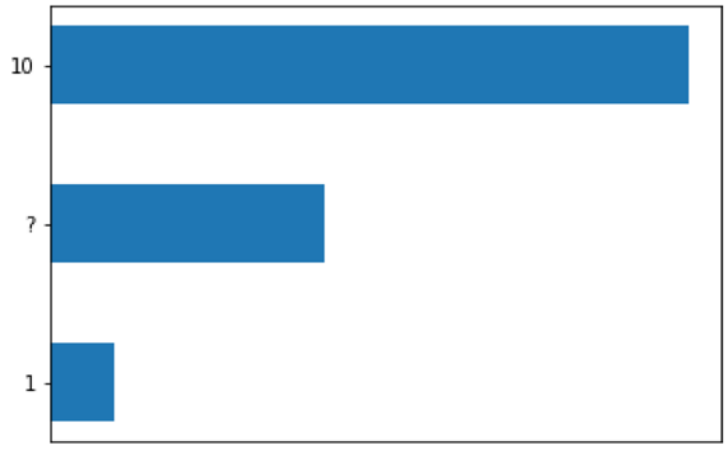
9.56666



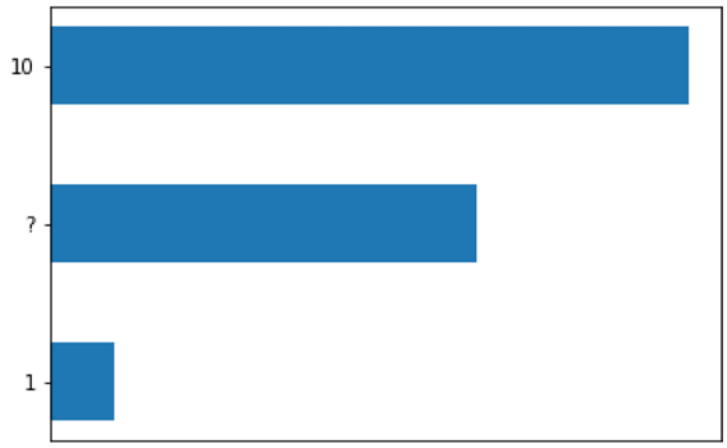
4.53333



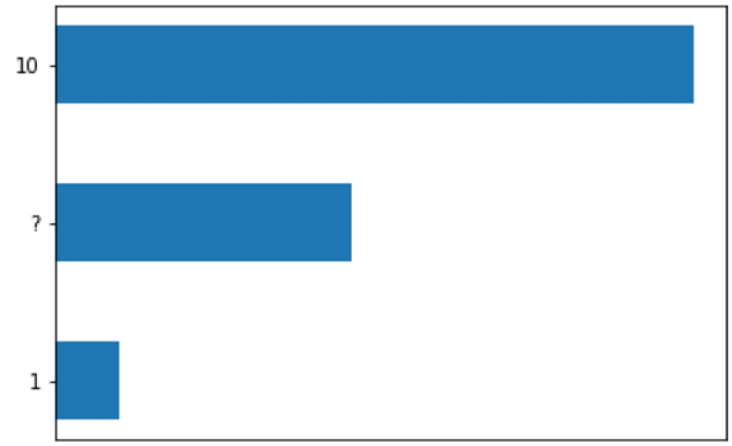
6.7333



4.5444



6.7444




```
4.5333
```

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 [30]:
```
length_perceived=[float(i) for i in perceived_length_list]
```

In [28]:
```
len(perceived_length_list)
print(length_perceived)
```

```
[5.9, 5.87, 4.5555, 1.73333, 9.56666, 4.53333, 6.7333, 4.5444, 6.7444, 4.5333]
```

In [21]:
```
len(actual_length_list)
length_actual=actual_length_list
print(length_actual)
```

```
[6.206411347311613, 5.8776810819388725, 4.6935413708879, 1.7115481396859655, 9.63812872
1537917, 4.56556440696342, 6.45847691208492, 4.28670272471745, 6.683110411695153, 4.628
2945288448545]
```

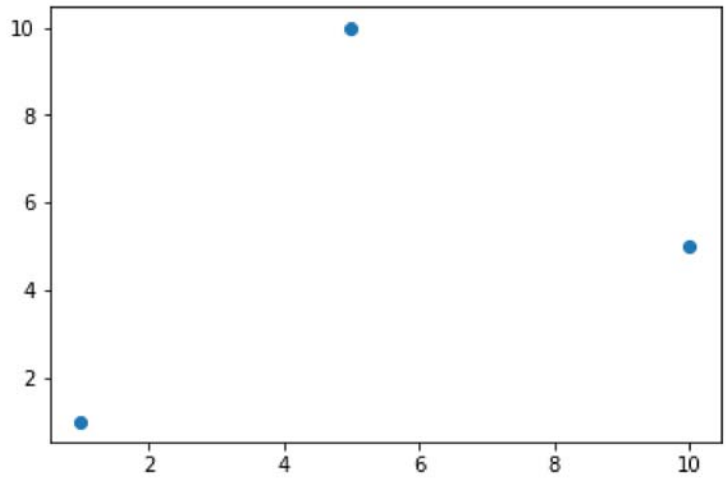
Plotting the result

Run the above two cells many times, namely: (1) generate a random bar graph, (2) put your guess into `length_perceived` while entering the actuall ratio into `length_actual`

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

In [26]:
```
plt.scatter(x=[1,5,10], y=[1,10, 5])
```

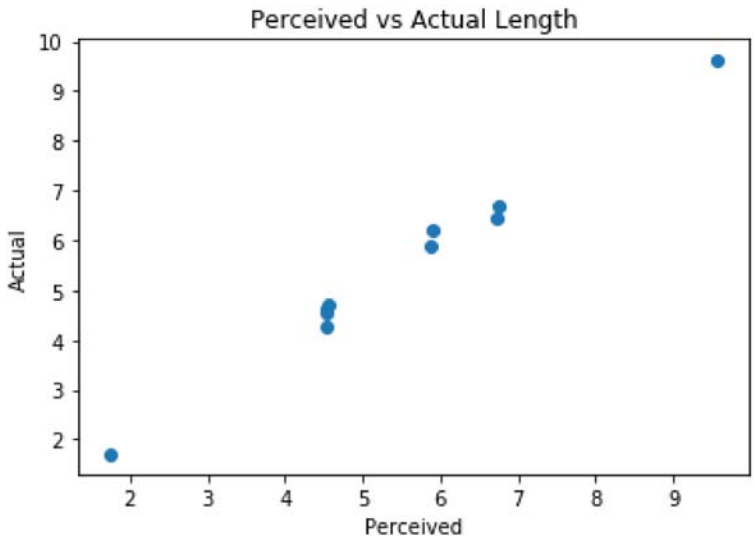
Out[26]:
```
<matplotlib.collections.PathCollection at 0x27470220ba8>
```



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 [36]:
```
# TODO: put your code here
plt.title('Perceived vs Actual Length')
plt.xlabel('Perceived')
plt.ylabel('Actual')
plt.scatter(x=length_perceived, y=length_actual)
```

Out[36]:
```
<matplotlib.collections.PathCollection at 0x1cc42414940>
```



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)`** (http://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html) in Scipy, which is to fit x and y using the function `f`. In our case, $f = a * x^b + c$. For instance, we can check whether this works by creating a fake dataset that follows the exact form:

In [37]:
```
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
```

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 [38]:
```
# TODO: put your code here
popt, pcov = curve_fit(func, length_perceived, length_actual)
print('{:.2f} x^{:.2f} + {:.2f}'.format(*popt))
```

```
1.00 x^1.00 + -0.01
```

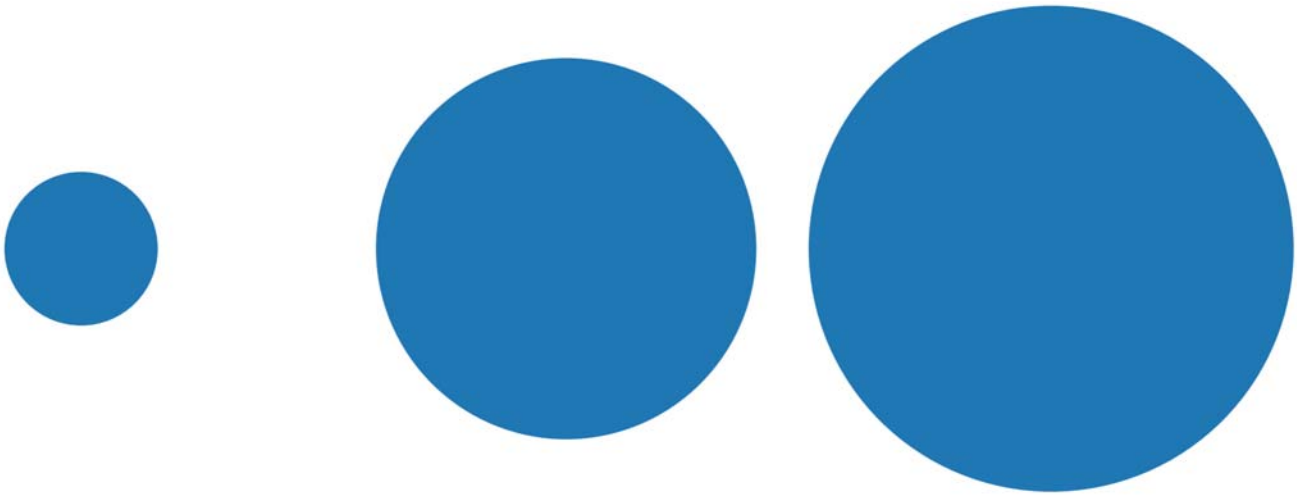
Perception of area

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 [39]:
```
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[39]:
```
<matplotlib.patches.Circle at 0x1cc42e1e780>
```



Let's have two lists for this experiment.

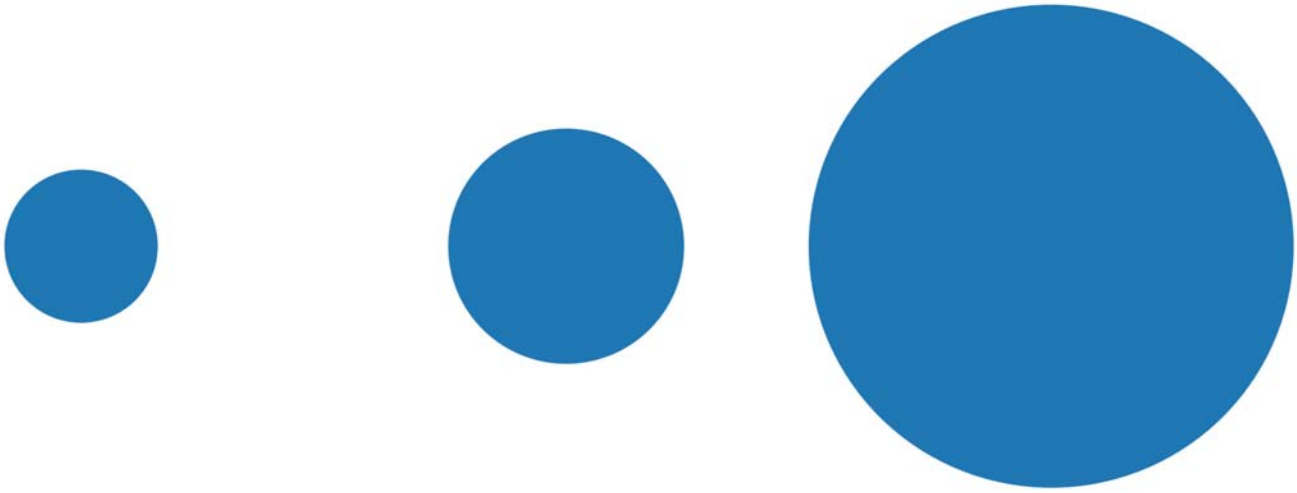
In [40]:
```
perceived_area_list = []
actual_area_list = []
```

And define a function for the experiment.

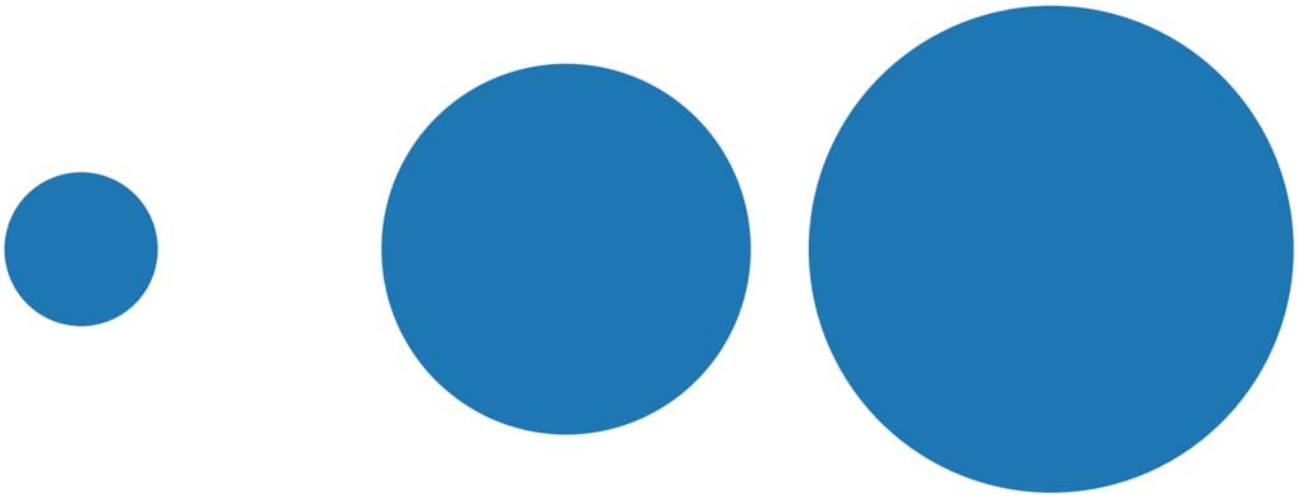
In [43]:
```
import math
def run_area_exp_once(n1=0.005, n2=0.05):
    radius1 = np.sqrt(n1/np.pi) # area = pi * r * r
    radius2 = np.sqrt(n2/np.pi)

    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 to test the Stevens' power-law!

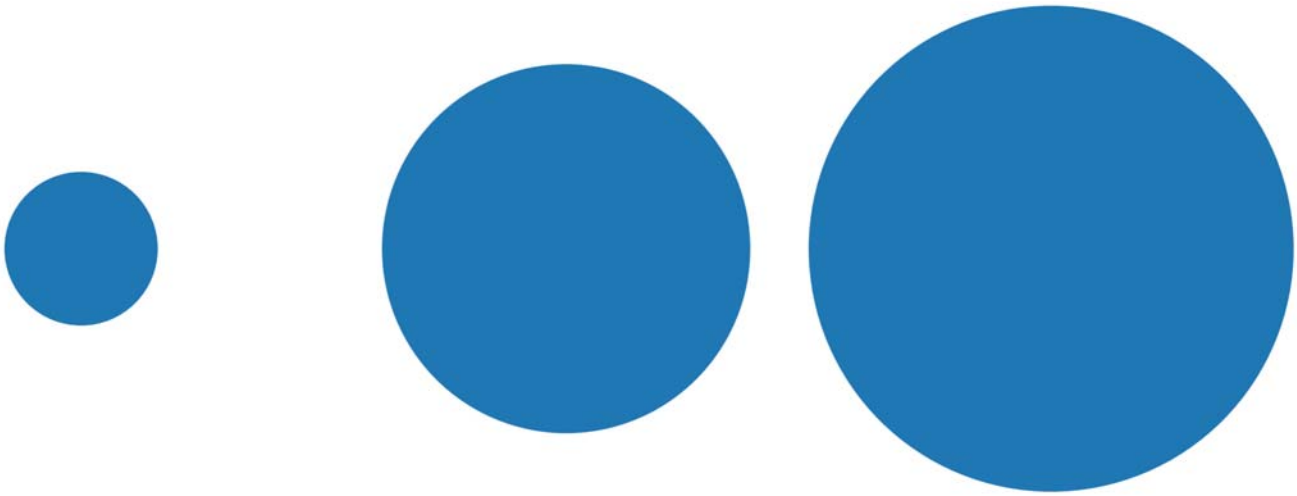
In [44]:
```
# TODO: put your code here. Use multiple cells.
for i in range(5):
    run_area_exp_once()
```



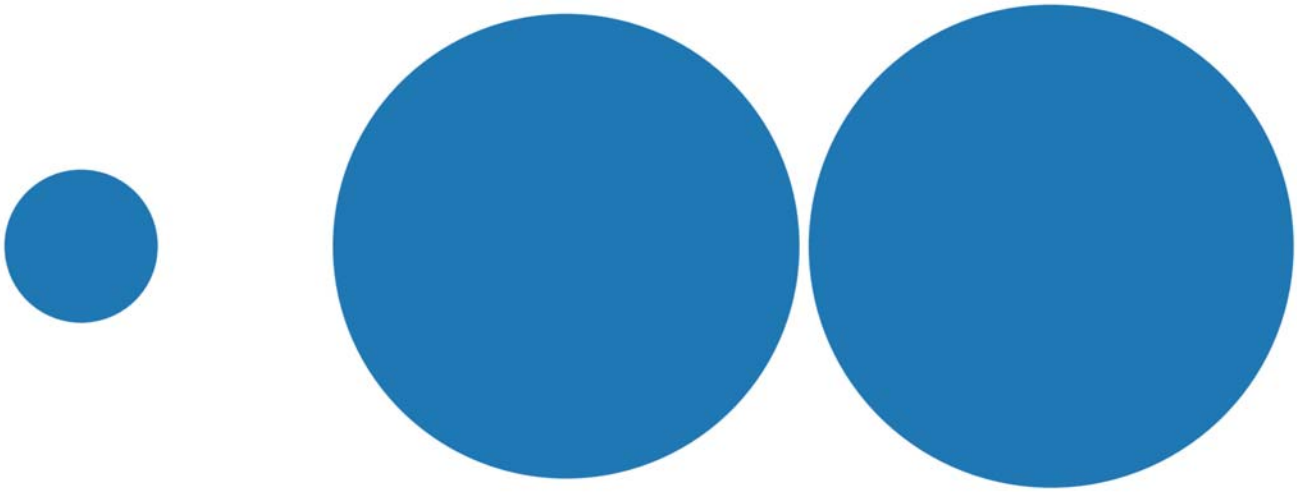
2.5



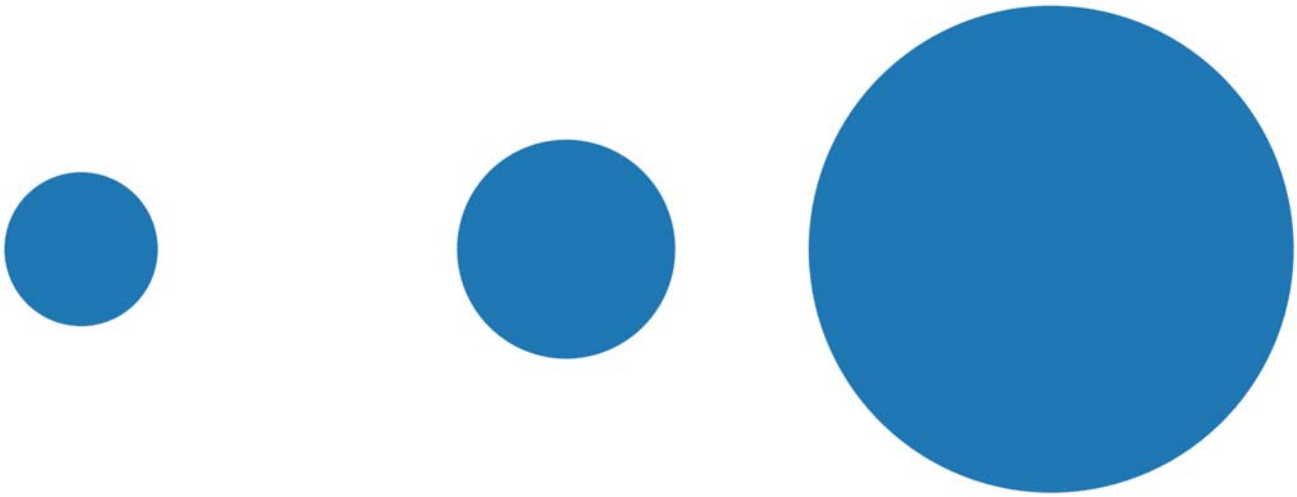
6.3



6.5



9.5



1.8

In [45]:
```
print(perceived_area_list)
```

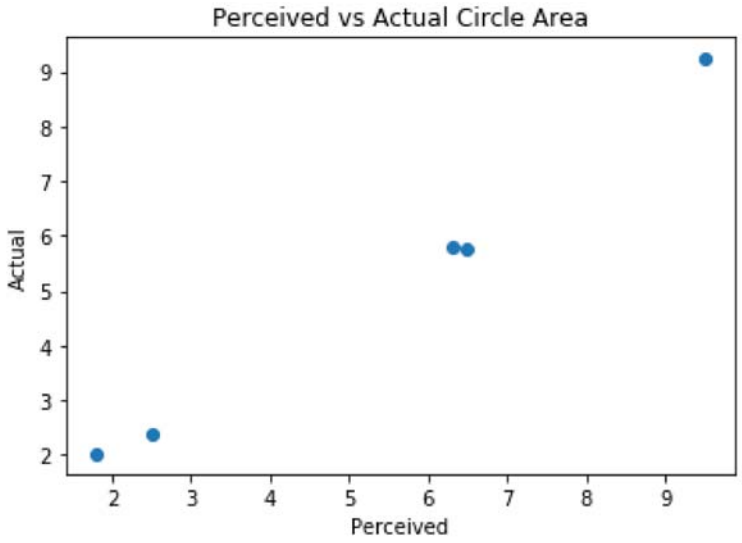
```
[2.5, 6.3, 6.5, 9.5, 1.8]
```

In [46]:
```
print(actual_area_list)
```

```
[2.366161683300528, 5.799733794758007, 5.76675586019363, 9.256586846815965, 2.023225537
875522]
```

In [47]:
```
plt.title('Perceived vs Actual Circle Area')
plt.xlabel('Perceived')
plt.ylabel('Actual')
plt.scatter(x=perceived_area_list, y=actual_area_list)
```

Out[47]: <matplotlib.collections.PathCollection at 0x1cc45617b70>



In [48]:
```
popt, pcov = curve_fit(func, perceived_area_list, actual_area_list)
print('{:.2f} x^{:.2f} + {:.2f}'.format(*popt))
```

```
0.31 x^1.45 + 1.27
```

Have you observed a sublinear relationship?

In [ ]:
```
#Yes, sublinear relationships were observed.
```