1. Sound it out!

Grey and Gray. Colour and Color. Words like these have been the cause of many heated arguments between Brits and Americans. Accents (and jokes) aside, there are many words that are pronounced the same way but have different spellings. While it is easy for us to realize their equivalence, basic programming commands will fail to equate such two strings.

More extreme than word spellings are names because people have more flexibility in choosing to spell a name in a certain way. To some extent, tradition sometimes governs the way a name is spelled, which limits the number of variations of any given English name. But if we consider global names and their associated English spellings, you can only imagine how many ways they can be spelled out.

One way to tackle this challenge is to write a program that checks if two strings sound the same, instead of checking for equivalence in spellings. We'll do that here using fuzzy name matching.

In [162]:
```
# Importing the fuzzy package
import fuzzy

# Exploring the output of fuzzy.nysiis
fuzzy.nysiis('dufole')

# Testing equivalence of similar sounding words
fuzzy.nysiis('holiday') == fuzzy.nysiis('hailday')
```

Out[162]:
```
False
```

In [163]:
```
%%nose
import sys

def test_fuzzy_is_loaded():
    assert 'fuzzy' in sys.modules, \
    'The fuzzy module should be loaded'
```

Out[163]:
```
1/1 tests passed
```

2. Authoring the authors

The New York Times puts out a weekly list of best-selling books from different genres, and which has been published since the 1930's. We'll focus on Children's Picture Books, and analyze the gender distribution of authors to see if there have been changes over time. We'll begin by reading in the data on the best selling authors from 2008 to 2017.

In [164]:
```
# Importing the pandas module
import pandas as pd

# Reading in datasets/nytkids_yearly.csv, which is semicolon delimited.
author_df = pd.read_csv('datasets/nytkids_yearly.csv', delimiter=';')

# Looping through author_df['Author'] to extract the authors first names
first_name = []
for name in author_df['Author']:
    first_name.append(name.split()[0])

# Adding first_name as a column to author_df
author_df['first_name']= first_name

# Checking out the first few rows of author_df
author_df.head()
```

Out[164]:

	Year	Book Title	Author	Besteller this year	first_name
0	2017	DRAGONS LOVE TACOS	Adam Rubin	49	Adam
1	2017	THE WONDERFUL THINGS YOU WILL BE	Emily Winfield Martin	48	Emily
2	2017	THE DAY THE CRAYONS QUIT	Drew Daywalt	44	Drew
3	2017	ROSIE REVERE, ENGINEER	Andrea Beaty	38	Andrea
4	2017	ADA TWIST, SCIENTIST	Andrea Beaty	28	Andrea

In [165]:
```
%%nose

def test_check_authors():
    len_auth = len(author_df['first_name'])
    all_names = list(author_df['first_name'])
    assert ('Shel' in all_names and len_auth==603), \
    'first_name column does not contan the correct first names of author
s'
```

Out[165]:
```
1/1 tests passed
```

3. It's time to bring on the phonics... *again*!

When we were young children, we were taught to read using phonics; sounding out the letters that compose words. So let's relive history and do that again, but using python this time. We will now create a new column or list that contains the phonetic equivalent of every first name that we just extracted.

To make sure we're on the right track, let's compare the number of unique values in the `first_name` column and the number of unique values in the nysiis coded column. As a rule of thumb, the number of unique nysiis first names should be less than or equal to the number of actual first names.

In [166]:
```
# Importing numpy
import numpy as np

# Looping through author's first names to create the nysiis (fuzzy) equi
valent
nysiis_name = []
for firstname in author_df['first_name']:
    nysiis_name.append(fuzzy.nysiis(firstname))

# Adding nysiis_name as a column to author_df
author_df['nysiis_name'] = nysiis_name

# Printing out the difference between unique firstnames and unique nysii
s_names:
diff_names = len(np.unique(author_df.first_name)) - \
    len(np.unique(author_df.nysiis_name))
print('There are ' + str(diff_names) +
      ' more unqiue values for first_name than nysiis_name')
```

```
There are 25 more unqiue values for first_name than nysiis_name
```

In [167]:
```
%%nose

import numpy as np

def test_check_nysiis_list():
    assert len( np.unique(author_df['nysiis_name']) ) == 145, \
        'The nysiis_name column does not contan the correct entries'
```

Out[167]:
```
1/1 tests passed
```

4. The inbetweeners

We'll use `babynames_nysiis.csv` , a dataset that is derived from [the Social Security Administration's baby name data (https://www.ssa.gov/oact/babynames/limits.html)](https://www.ssa.gov/oact/babynames/limits.html), to identify author genders. The dataset contains unique NYSIIS versions of baby names, and also includes the percentage of times the name appeared as a female name (`perc_female`) and the percentage of times it appeared as a male name (`perc_male`).

We'll use this data to create a list of `gender` . Let's make the following simplifying assumption: For each name, if `perc_female` is greater than `perc_male` then assume the name is female, if `perc_female` is less than `perc_male` then assume it is a male name, and if the percentages are equal then it's a "neutral" name.

In [168]:
```
# Reading in datasets/babynames_nysiis.csv, which is semicolon delimite
d.
babies_df = pd.read_csv('datasets/babynames_nysiis.csv', delimiter=';')

# Looping through the rows of babies_df to and filling up gender
gender = []
for idx in range(len(babies_df['babynysiis'])):
    if babies_df.perc_female[idx] > babies_df.perc_male[idx]:
        gender.append('F')
    elif babies_df.perc_female[idx] < babies_df.perc_male[idx]:
        gender.append('M')
    else:
        gender.append('N')

# Adding a gender column to babies_df
babies_df['gender'] = gender

# Printing out the first few rows of babies_df
babies_df.head()
```

Out[168]:

	babynysiis	perc_female	perc_male	gender
0	NaN	62.50	37.50	F
1	RAX	63.64	36.36	F
2	ESAR	44.44	55.56	M
3	DJANG	0.00	100.00	M
4	PARCAL	25.00	75.00	M

In [169]:
```
%%nose

def test_gender_distribution():
    assert len([i for i, x in enumerate(babies_df['gender']) if x ==
 'N']) == 1170,\
        'gender column does not contain the correct number of Male, Fema
le and Neutral names, which are 7031, 8939 and 1170 respectively'
```

Out[169]:
```
1/1 tests passed
```

5. Playing matchmaker

Now that we have identified the likely genders of different names, let's find author genders by searching for each author's name in the `babies_df` DataFrame, and extracting the associated gender.

In [170]:
```
# This function returns the location of an element in a_list.
# Where an item does not exist, it returns -1.
def locate_in_list(a_list, element):
    loc_of_name = a_list.index(element) if element in a_list else -1
    return(loc_of_name)

# Looping through author_df['nysiis_name'] and appending the gender of e
ach
# author to author_gender.
author_gender = []
for name in author_df['nysiis_name']:
    nloc = locate_in_list(list(babies_df['babynysiis']), name)
    if nloc == -1:
        author_gender.append('Unknown')
    else:
        author_gender.append(babies_df['gender'][nloc])

# Adding author_gender to the author_df
author_df['author_gender'] = author_gender

# Counting the author's genders
author_df['author_gender'].value_counts()
```

Out[170]:
```
F          395
M          191
Unknown      9
N            8
Name: author_gender, dtype: int64
```

In [171]:
```
%%nose

def len_authors():
    return len(author_df[author_df.author_gender == "M"])

def test_num_males():
    assert len_authors() == 191, \
        'The number of Males (M) and Females (F) appear to be wrong. The
se are 191 and 395 respectively'
```

Out[171]:
```
1/1 tests passed
```

6. Tally up

From the results above see that there are more female authors on the New York Times best seller's list than male authors. Our dataset spans 2008 to 2017. Let's find out if there have been changes over time.

In [172]:
```
# Creating a list of unique years, sorted in ascending order.
years = list(np.unique(author_df.Year))

# Intializing lists
males_by_yr = []
females_by_yr = []
unknown_by_yr = []

# Looping through years to find the number of male, female and unknown a
uthors per year
for yr in years:
    males_by_yr.append(
        len(author_df[(author_df["author_gender"] == 'M') & (author_df[
"Year"] == yr)]))
    females_by_yr.append(
        len(author_df[(author_df["author_gender"] == 'F') & (author_df[
"Year"] == yr)]))
    unknown_by_yr.append(len(
        author_df[(author_df["author_gender"] == 'Unknown') & (author_df
["Year"] == yr)]))

# Printing out yearly values to examine changes over time
data = np.array([males_by_yr, females_by_yr, unknown_by_yr])
headers = ['males', 'females', 'unknowns']
pd.DataFrame(data, headers, years)
```

Out[172]:

	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017
males	8	19	27	21	21	11	21	18	25	20
females	15	45	48	51	46	51	34	30	32	43
unknowns	1	3	0	1	0	2	1	0	0	1

In [173]:
```
%%nose

def test_years():
    correct_years = list(np.unique(author_df.Year))
    assert list(years) == correct_years, \
    'years should be the unique years in author_df["Year"] sorted in asc
ending order.'

def test_gender_by_yr():
    assert sum(males_by_yr)==191, \
    'At least one of the lists (males_by_yr, females_by_yr, unknown_by_y
r) contains an incorrect value.'
```

Out[173]:
```
2/2 tests passed
```

7. Foreign-born authors?

Our gender data comes from social security applications of individuals born in the US. Hence, one possible explanation for why there are "unknown" genders associated with some author names is because these authors were foreign-born. While making this assumption, we should note that these are only a subset of foreign-born authors as others will have names that have a match in `baby_df` (and in the social security dataset).

Using a bar chart, let's explore the trend of foreign-born authors with no name matches in the social security dataset.

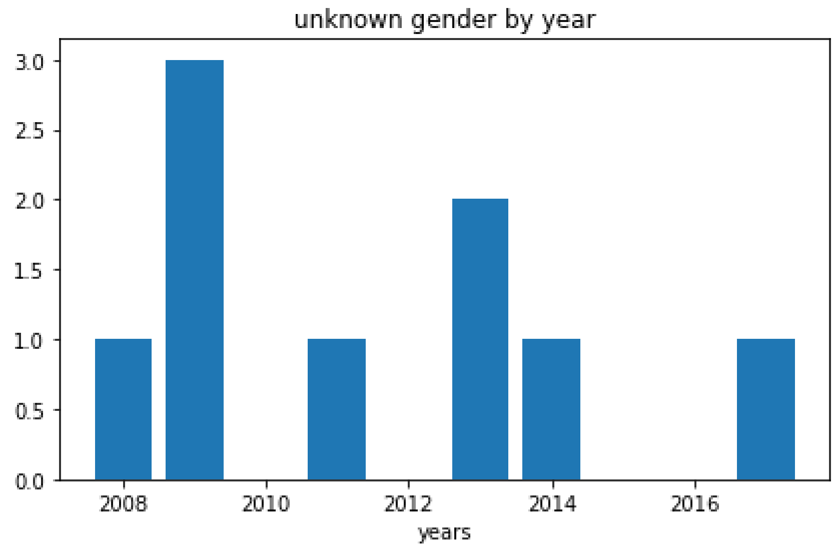
In [174]:
```
# Importing matplotlib
import matplotlib.pyplot as plt

# This makes plots appear in the notebook
%matplotlib inline

# Plotting the bar chart
plt.bar(years, unknown_by_yr)

# [OPTIONAL] - Setting a title, and axes labels
plt.title('unknown gender by year')
plt.xlabel('years')
```

Out[174]:
```
Text(0.5,0,'years')
```



In [175]:
```
%%nose

# It's hard to test plots.
def test_nothing():
    assert True, ""

#def test_pos():
#    assert  pos ==list(range(len(unknown_by_yr))) or pos== range(len(un
known_by_yr)) or pos==years, \
#    'pos should be a list containing integer values with the same lengt
h as unknown_by_yr '
```

Out[175]:
```
1/1 tests passed
```

8. Raising the bar

What's more exciting than a bar chart is a grouped bar chart. This type of chart is good for displaying *changes* over time while also *comparing* two or more groups. Let's use a grouped bar chart to look at the distribution of male and female authors over time.

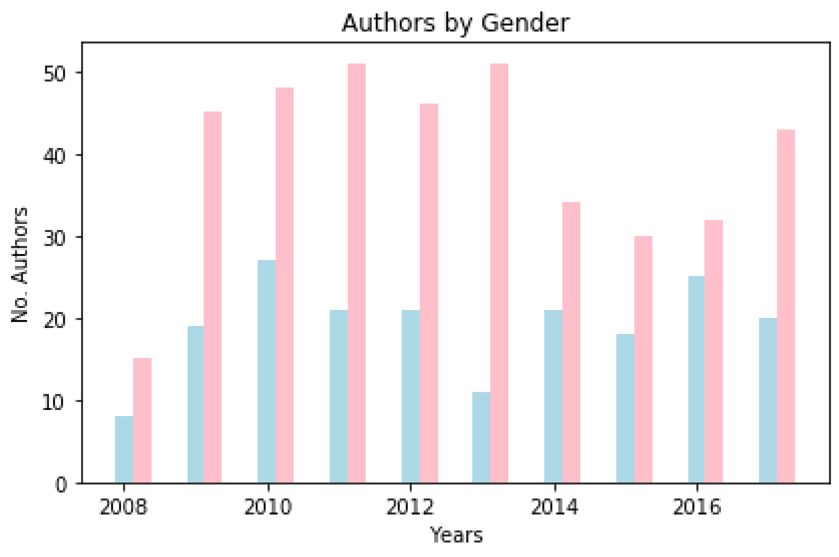
In [176]:
```
# Creating a new list, where 0.25 is added to each year
years_shifted = [year + 0.25 for year in years]

# Plotting males_by_yr by year
plt.bar(years, males_by_yr, width = 0.25, color = 'lightblue')

# Plotting females_by_yr by years_shifted
plt.bar(years_shifted, females_by_yr, width = 0.25, color = 'pink')

# [OPTIONAL] - Adding relevant Axes labels and Chart Title
plt.xlabel('Years')
plt.ylabel('No. Authors')
plt.title('Authors by Gender')
```

Out[176]:
```
Text(0.5,1,'Authors by Gender')
```



In [177]:
```
%%nose

def test_years_shifted():
    correct_years_shifted = [year + 0.25 for year in years]
    assert list(years_shifted) == correct_years_shifted, \
    'years_shifted should be like years but with 0.25 added to each yea
r.'
```

Out[177]:
```
1/1 tests passed
```