(http://www.pieriandata.com)

Copyright by Pierian Data Inc.
For more information, visit us at www.pieriandata.com (http://www.pieriandata.com)

Text Methods

A normal Python string has a variety of method calls available:

In [2]:
```
mystring = 'hello'
```

In [3]:
```
mystring.capitalize()
```

Out[3]:
```
'Hello'
```

In [4]:
```
mystring.isdigit()
```

Out[4]:
```
False
```

In [5]:
```
help(str)
```

```
Help on class str in module builtins:

class str(object)
 |  str(object='') -> str
 |  str(bytes_or_buffer[, encoding[, errors]]) -> str
 |
 |  Create a new string object from the given object. If encoding or
 |  errors is specified, then the object must expose a data buffer
 |  that will be decoded using the given encoding and error handler.
 |  Otherwise, returns the result of object.__str__() (if defined)
 |  or repr(object).
 |  encoding defaults to sys.getdefaultencoding().
 |  errors defaults to 'strict'.
 |
 |  Methods defined here:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __contains__(self, key, /)
```

Pandas and Text

Pandas can do a lot more than what we show here. Full online documentation on things like advanced string indexing and regular expressions with pandas can be found here: https://pandas.pydata.org/docs/user_guide/text.html (https://pandas.pydata.org/docs/user_guide/text.html)

Text Methods on Pandas String Column

In [6]:
```
import pandas as pd
```

In [7]:
```
names = pd.Series(['andrew','bobo','claire','david','4'])
```

In [8]:
```
names
```

Out[8]:
```
0    andrew
1      bobo
2    claire
3     david
4         4
dtype: object
```

In [9]:
```
names.str.capitalize()
```

Out[9]:
```
0    Andrew
1      Bobo
2    Claire
3     David
4         4
dtype: object
```

In [10]:
```
names.str.isdigit()
```

Out[10]:
```
0    False
1    False
2    False
3    False
4     True
dtype: bool
```

Splitting , Grabbing, and Expanding

In [14]:
```
tech_finance = ['GOOG,APPL,AMZN','JPM,BAC,GS']
```

In [15]:
```
len(tech_finance)
```

Out[15]:
```
2
```

In [16]:
```
tickers = pd.Series(tech_finance)
```

In [17]:
```
tickers
```

Out[17]:
```
0    GOOG,APPL,AMZN
1        JPM,BAC,GS
dtype: object
```

In [18]:
```
tickers.str.split(',')
```

Out[18]:
```
0    [GOOG, APPL, AMZN]
1        [JPM, BAC, GS]
dtype: object
```

In [19]:
```
tickers.str.split(',').str[0]
```

Out[19]:
```
0    GOOG
1     JPM
dtype: object
```

In [21]:
```
tickers.str.split(',',expand=True)
```

Out[21]:

	0	1	2
0	GOOG	APPL	AMZN
1	JPM	BAC	GS

Cleaning or Editing Strings

In [22]:
```
messy_names = pd.Series(["andrew  ","bo;bo","  claire  "])
```

In [27]:
```
# Notice the "mis-alignment" on the right hand side due to spacing in "andr
messy_names
```

Out[27]:
```
0      andrew
1         bo;bo
2      claire
dtype: object
```

In [28]:
```
messy_names.str.replace(";","")
```

Out[28]:
```
0      andrew
1          bobo
2      claire
dtype: object
```

In [29]:
```
messy_names.str.strip()
```

Out[29]:
```
0    andrew
1     bo;bo
2    claire
dtype: object
```

In [31]:
```
messy_names.str.replace(";","").str.strip()
```

Out[31]:
```
0    andrew
1      bobo
2    claire
dtype: object
```

In [32]:
```
messy_names.str.replace(";","").str.strip().str.capitalize()
```

Out[32]:
```
0    Andrew
1      Bobo
2    Claire
dtype: object
```

Alternative with Custom apply() call

In [33]:
```
def cleanup(name):
    name = name.replace(";","")
    name = name.strip()
    name = name.capitalize()
    return name
```

In [34]:
```
messy_names
```

Out[34]:
```
0      andrew
1         bo;bo
2      claire
dtype: object
```

In [35]:
```
messy_names.apply(cleanup)
```

Out[35]:
```
0    Andrew
1      Bobo
2    Claire
dtype: object
```

Which one is more efficient?

In [43]:
```
import timeit

# code snippet to be executed only once
setup = '''
import pandas as pd
import numpy as np
messy_names = pd.Series(["andrew  ","bo;bo","  claire  "])
def cleanup(name):
    name = name.replace(";","")
    name = name.strip()
    name = name.capitalize()
    return name
'''

# code snippet whose execution time is to be measured
stmt_pandas_str = '''
messy_names.str.replace(";","").str.strip().str.capitalize()
'''

stmt_pandas_apply = '''
messy_names.apply(cleanup)
'''

stmt_pandas_vectorize='''
np.vectorize(cleanup)(messy_names)
'''
```

In [44]:
```
timeit.timeit(setup = setup,
                    stmt = stmt_pandas_str,
                    number = 10000)
```

Out[44]: 3.931618999999955

In [45]:
```
timeit.timeit(setup = setup,
                    stmt = stmt_pandas_apply,
                    number = 10000)
```

Out[45]: 1.2268500999999787

In [46]:
```
timeit.timeit(setup = setup,
                    stmt = stmt_pandas_vectorize,
                    number = 10000)
```

Out[46]: 0.28283379999993485

Wow! While .str() methods can be extremely convienent, when it comes to performance, don't forget about np.vectorize()! Review the "Useful Methods" lecture for a deeper discussion on np.vectorize()