Python Tutorial:
Python for Data Science

By Fahad Kamran
Overview

- Introduction
- Variables and Types
- Operators
- Containers
- Functions
- Control Flow
- Packages
  - Numpy
  - Pandas
  - Matplotlib
Overview

- Introduction
- Variables and Types
- Operators
- Containers
- Functions
- Control Flow
- Packages
  - Numpy
  - Pandas
  - Matplotlib
- Easy to learn.
- Elegant syntax.
- Lots of scientific computing resources.
- Very user friendly.
print("Hello World")

Hello World
Overview

- Introduction
- **Variables and Types**
- Operators
- Containers
- Functions
- Control Flow
- Packages
  - Numpy
  - Pandas
  - Matplotlib
Variables

- Put data in memory, and give it a name.
- Created using the assignment operator =

```python
x = 3
print(x)
3

x = 3 + 3
print(x)
6
```
Variables

\[ x = 3 \]  \textbf{Int:} Integers

\[ \text{pi} = 3.14 \]  \textbf{Float:} Real numbers

\text{day} = \text{“Monday”}  \textbf{String:} Characters, \textit{different} than a variable.

\text{is_monday} = \text{True}  \textbf{Boolean:} True/False.

\# Objects.  \textbf{Comment:} Notes for programmers, doesn’t get executed.
Overview

- Introduction
- Variables and Types
- **Operators**
- Containers
- Functions
- Control Flow
- Packages
  - Numpy
  - Pandas
  - Matplotlib
Operators

- Symbols that carry out computation.
- PEMDAS

```
x = 15
y = 4
print(x + y)  # Addition.
19

print(y - x)  # Subtraction.
-11

print(y * x)  # Multiplication.
60
```
Operators

```python
x = 15
y = 4

print(x / y)  # Division.
3.75

print(x // y)  # Floor Division.
3

print(x % y)  # Modulus (remainder).
3

print(y ** y)  # Exponent.
256
```
Operators

x = 15
y = 4

print(x == y)  # Equality.
False

print(x > y)  # Greater than.
True

print(x <= y)  # Less than or Equal.
False

print(x != y)  # Not Equal.
True
Operators

```python
print(True and True)
True

print(True and False)
False

print(True or False)
True

print(False or False)
False
```
Operators

w = "Hello "
x = "World!"
y = 1
z = 1.1

print(w + x)
Hello World!

print(y + z)
2.1

print(w + y)
Error!
Operators

Arithmetic Operators

+  
-  
*  
/  
//  
%  
**

Logical Operators

==  
>  
>=  
!=  
<  
<=  
and  
or  
not

Logical operators are evaluated after arithmetic.
Demo
Overview

- Introduction
- Variables and Types
- Operators
- Containers
- Functions
- Control Flow
- Packages
  - Numpy
  - Pandas
  - Matplotlib
Class/Object

- Combine data and *behaviors* related to the class’ data.
- Making our own is out of the scope of this course.
- We will use them!
List

- Series of values.

a = [] # Empty list.

b = [1, 2, 3, 4, 5] # List containing 5 elements.
List

- Series of values.

a = []  # Empty list.

b = [1, 2, 3, 4, 5]  # List containing 5 elements.

b[0]  # Zero-indexed!
1
List

- Series of values.

a = [] # Empty list.

b = [1, 2, 3, 4, 5] # List containing 5 elements.

b[0] # Zero-indexed!
1

b[3]
b[3]
4
List

0 1 2 3 4  # Index.

b = [1, 2, 3, 4, 5]

- We can get parts of the list with the slice operator:
- Optionally, define a range of indices, excludes last element.

b[:1]
[1]

b[1:3]
[2, 3]
List

- Access class `methods` using the `dot` operator.

```python
x = [1]
print(x)
[1]

x.append(2)  # Add element to end of list.
[1, 2]

x.pop()    # Get last element in list and remove from list.
2
print(x)
[1]
```
List

- Reassign parts of a list

```python
x = [1]
print(x)
[1]
x[0] = 'new element!'
print(x)
['new element!']
```
5. Data Structures

This chapter describes some things you've learned about already in more detail, and adds some new things as well.

5.1. More on Lists

The list data type has some more methods. Here are all of the methods of list objects:

list.append(x)
Add an item to the end of the list. Equivalent to a[len(a):] = [x].

list.extend(iterable)
Extend the list by appending all the items from the iterable. Equivalent to a[len(a):] = iterable.

list.insert(i, x)
Insert an item at a given position. The first argument is the index of the element before which to insert, so a.insert(0, x) inserts at the front of the list, and a.insert(len(a), x) is equivalent to a.append(x).

list.remove(x)
Remove the first item from the list whose value is x. It is an error if there is no such item.

list.pop(i)
Remove the item at the given position in the list, and return it. If no index is specified, a.pop() removes and returns the last item in the list. (The square brackets around the i in the method signature denote that the parameter is optional, not that you should type square brackets at that position. You will see this notation frequently in the Python Library Reference.)

list.clear()
Remove all items from the list. Equivalent to del a[:].

list.index(x, [start], [end])
Return zero-based index in the list of the first item whose value is x. Raises a ValueError if there is no such item.

The optional arguments start and end are interpreted as in the slice notation and are used to limit the search to a particular subsequence of the list. The returned index is computed relative to the
Dictionary / Map

- List of key-value pairs.
- Lists but access elements with anything!

```python
a = {}  # Empty dict.

a["key"] = "value"

print(a)
{"key": "value"}
```
Dictionary / Map

```python
person = {
    "name": "Jim",
    "family_name": "Harbaugh"
}

person["name"]
Jim

person["age"] = 55

print(person)
{"name": "Jim", "family_name": "Harbaugh", "age": 55}
```
Other Containers

Containers not covered include:

- Sets
- Tuples
- Generators
Creating Copies

- What does an assignment statement do?

```python
x = [1, 2, 3]
y = x  # Make a copy of x.
y[2] = 100  # Reassign part of y.
print(x)
```
Creating Copies

- What does an assignment statement do?

```python
x = [1, 2, 3]
y = x  # Make a copy of x.
y[2] = 100  # Reassign part of y.
print(x)
[1, 2, 100]
```
Creating Copies

- What does an assignment statement do?

```python
x = [1, 2, 3]
y = x  # Make a copy of x.
y[2] = 100  # Reassign part of y.
print(x)
[1, 2, 100]
```

**Changing y still changes x**
Creating Copies

- What does an assignment statement do?

```python
x = [1, 2, 3]
y = x  # Make a copy of x.
y[2] = 100  # Reassign part of y.
print(x)
[1, 2, 100]
```

Changing \( y \) still changes \( x \)
Creating Copies

- Fixing the assignment issue

```python
x = [1, 2, 3]
y = x[:]  # Make a new copy of x.
y[2] = 100  # Reassign part of y.
print(x)
```
Creating Copies

- Fixing the assignment issue

```python
x = [1,2,3]
y = x[:]
# Make a new copy of x.
y[2] = 100
# Reassign part of y.
print(x)
[1, 2, 3]
```
Creating Copies

- Fixing the assignment issue

```python
x = [1, 2, 3]
y = x[:]  # Make a new copy of x.
y[2] = 100  # Reassign part of y.
print(x)
[1, 2, 3]
```

Frames

- Global frame
- first_list
- second_list

Objects

- list
  - 0: 1
  - 1: 2
  - 2: 3
  - 3: 3
Exercise #1

- What are the outputs of each line of code?

```python
fib = ["hello", "hi", "hey", "hola", "heyo", "sup", "howdy"]

fib[1:5]
# Q1.

fib[3:]
# Q2.

fib[:3]
# Q3.
```
Demo
Overview

- Introduction
- Variables and Types
- Operators
- Containers
- Functions
- Control Flow
- Packages
  - Numpy
  - Pandas
  - Matplotlib
Functions

- Group related code that performs a task.
- Reusable blocks of code.
- Take in *arguments*.
Functions

n_hours = 181 // 60
n_minutes = 181 % 60
print(n_hours)
print(n_minutes)

n_hours = 72 // 60
n_minutes = 72 % 60
print(n_hours)
print(n_minutes)

n_hours = 451 // 60
n_minutes = 451 % 60
print(n_hours)
print(n_minutes)
Functions

n_hours = 181 // 60
n_minutes = 181 % 60
print(n_hours)
print(n_minutes)

n_hours = 72 // 60
n_minutes = 72 % 60
print(n_hours)
print(n_minutes)

n_hours = 451 // 60
n_minutes = 451 % 60
print(n_hours)
print(n_minutes)
Functions

```python
n_hours = 181 // 60
n_minutes = 181 % 60
print(n_hours)
print(n_minutes)

n_hours = 72 // 60
n_minutes = 72 % 60
print(n_hours)
print(n_minutes)

n_hours = 451 // 60
n_minutes = 451 % 60
print(n_hours)
print(n_minutes)

def hours_and_minutes(minutes):
    n_hours = minutes // 60
    n_minutes = minutes % 60
    return n_hours, n_minutes

print(hours_and_minutes(181))
print(hours_and_minutes(72))
print(hours_and_minutes(451))
```
Functions

def hours_and_minutes(minutes):
    n_hours = minutes // 60
    n_minutes = minutes % 60
    return n_hours, n_minutes
def hours_and_minutes(minutes):
    n_hours = minutes // 60
    n_minutes = minutes % 60
    return n_hours, n_minutes
```python
def hours_and_minutes(minutes):
    n_hours = minutes // 60
    n_minutes = minutes % 60
    return n_hours, n_minutes
```
Functions

```python
def hours_and_minutes(minutes):
    n_hours = minutes // 60
    n_minutes = minutes % 60
    return n_hours, n_minutes
```

Code the function performs.

Whitespace/tabs matter!

Define scope
def hours_and_minutes(minutes):
    n_hours = minutes // 60
    n_minutes = minutes % 60
    return n_hours, n_minutes

n_hours = 10
ret_hours, ret_minutes = hours_and_minutes(81)
print(n_hours)
Functions

def hours_and_minutes(minutes):
    n_hours = minutes // 60
    n_minutes = minutes % 60
    return n_hours, n_minutes

n_hours = 10
ret_hours, ret_minutes = hours_and_minutes(81)
print(n_hours)
10
Functions

def hours_and_minutes(minutes):
    n_hours = minutes // 60
    n_minutes = minutes % 60
    print(n_hours, n_minutes)

ret_hours, ret_minutes = hours_and_minutes(81)
1,21
print(ret_hours)
None
ret_hours + 1
Error
Exercise #2

```python
def fahrenheit_to_celsius(f):
    # Converts from °F to °C.
    # Where: °C = (°F - 32) * 5/9

    # Your code here.

    return c

fahrenheit_to_celsius(98.6)
```

37.0
Demo
Overview

- Introduction
- Variables and Types
- Operators
- Containers
- Functions
- **Control Flow**
- Packages
  - Numpy
  - Pandas
  - Matplotlib
If
- Only execute a block of code if a **condition** is True.

```python
if age > 21:
    print("Come in!")
elif age == 20:
    print("Try again next year bud.")
else:
    print("You’ve got some time!")
```

**Condition Statement.** Must result in True/False

**If/Elif Block.** Evaluated if condition is true.

**Else Block.** Evaluated if no conditions are true.
For-Loop

- Perform operations using item(s) in a container.
- *Cannot modify items.*

container = [1, 2, 3]
For-Loop

- Perform operations using item(s) in a container.
- *Cannot modify items.*

container = [1, 2, 3]

```python
for x in container:
    print(x)
```

1
2
3
For-Loop

`range(n)` function creates container of size \( n \) useful for repeating an action \( n \) times

```python
for x in range(5):
    print('Hello world!')
```

Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
List Comprehensions

lst = [1, 2, 3, 4, 5]
squared_lst = []
for num in lst:
    squared_lst.append(num**2)
squared_list
[1, 4, 9, 16, 25]
List Comprehensions

```
lst = [1,2,3,4,5]
squared_lst = []
for num in lst:
    squared_lst.append(num**2)
squared_list
[1,4,9,16,25]
squared_lst2 = [num**2 for num in lst]
squared_list
[1,4,9,16,25]
```
List Comprehensions

```
lst = [1,2,3,4,5]
squared_even_lst = []
for num in lst:
    if num % 2 == 0:
        squared_even_lst.append(num**2)
squared_even_lst
[4,16]

squared_even_lst2 = [num**2 for num in lst if num % 2 == 0]
squared_even_lst2
[4,16]
```
While-Loop

- Perform operations until condition is met.

```python
i = 0
while i < 3:
    print(i)
    i += 1
```

0 1 2
Exercise #3

Please sort the following data, into two new lists: one for words and the other numbers.

data = ["michigan", 1, "stats", 3.33, "hello"]

# Your code here.

At the end we’ll have two lists one containing “michigan”, “stats”, and “hello”; the other list containing 1, 3.33.
Exercise #4

Define a function `only_odd`, which takes in a dictionary, and returns a new dictionary with the key:value pairs where the value is an odd number

dct = {“Janice”: 3, “Fred”: 2, “Gregg”: 8, “Gloria”: 1}

# Your code here.

`only_odd(dct)`

{“Janice”: 3, “Gloria”: 1}
Demo
Overview

- Introduction
- Variables and Types
- Operators
- Containers
- Functions
- Control Flow
- Packages
  - Numpy
  - Pandas
  - Matplotlib
Module

- Collect variables, functions, classes into a module.
  - Sometimes called: library or package.
Module

- Collect variables, functions, classes into a module.
  - Sometimes called: library or package.

```python
import math
```
Module

- Collect variables, functions, classes into a module.
  - Sometimes called: library or package.

```python
import math
math.sqrt(16)
4
```
Module

- Collect variables, functions, classes into a module.
  - Sometimes called: library or package.

```python
import math
math.sqrt(16)
4

from math import sqrt
sqrt(16)
4
```
NumPy

- Tensor/matrix operation library.
- Lists, but more dimensions, and faster.
- **NOTE:** Normally you would need to install this library.

```python
import numpy as np
dx = np.array([1, 2, 3])
```
NumPy

- Tensor/matrix operation library.
- Lists, but more dimensions, and faster.
- **NOTE:** Normally you would need to install this library.

```python
import numpy as np
x = np.array([1, 2, 3])

np.dot([1, 2], [1, 2])  # Dot Product.
4
```
NumPy Array with Operators

```python
x = np.array([1, 2, 3])

x + 1
[2, 3, 4]

x + x
[2, 4, 6]

x ** 2
[2, 4, 9]
```
NumPy Array with Operators

```python
x = np.array([1, 2, 3])

x > 1
[False, True, True]

x == 1
[True, False, False]
```
NumPy Statistics

np.sum([[0, 1], [2, 3]])
6

np.sum([[0, 1], [2, 3]], axis=1)
[1, 5]

np.max([[0, 1], [2, 3]])
3

np.max([[0, 1], [2, 3]], axis=0)
[2, 3]
NumPy Statistics

np.sum([[0, 1], [2, 3]])
6

np.sum([[0, 1], [2, 3]], axis=1)
[1, 5]

np.max([[0, 1], [2, 3]])
3

np.max([[0, 1], [2, 3]], axis=0)
[2, 3]

https://docs.scipy.org/doc/numpy-1.13.0/reference/routines.statistics.html
Creating Sequences in NumPy

```
np.arange(3)
[0, 1, 2]

np.arange(0, 4, 2)  # (start, stop, step).
[0, 2]
```
Creating Arrays in NumPy

np.zeros(5)
arr([0, 0, 0, 0, 0])

np.ones(5)*3
arr([3, 3, 3, 3, 3])

np.random.random((2,2))
arr([[.810, .081], [.687, .541]])
Array Indexing

```
a = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
a[0,0]
1

a[1,:]
arr([5, 6, 7, 8])

a[1:,2:]
arr([[7, 8], [11, 12]])
```
Array Indexing

```python
import numpy as np

a = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
a[0, 0] = 4
a[1] = [4, 8, 9, 1]
print(a)
```

```python
print(a)
```

```python
arr([[4, 2, 3, 4], [4, 8, 9, 1], [9, 10, 11, 12]])
```
Array Indexing

```python
a = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
b = a[:2,1:3]
print(b)
arr([[2, 3], [6, 7]])
b[0,0] = 1
print(a)
arr([[1,1,3,4],[5,6,7,8],[9,10,11,12]])
```
Array Indexing

```python
a = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
b = np.array(a[:2,1:3])
print(b)
arr([[2, 3], [6, 7]])

b[0,0] = 1
print(a)
arr([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
```
Demo
Pandas

- Database package.
- Import data from:
  - Excel (csv, tsv, etc.)
  - Stata, sas, matlab
  - SQL,
  - Etc.

import pandas as pd
Pandas

- This will be a high-level summary of the package.
- We’ll look at stuff that you can follow along with our data.
```python
[24] s = pd.Series([1, 3, 4, np.nan, 6, 8])
print(s)
```

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.0</td>
</tr>
<tr>
<td>1</td>
<td>3.0</td>
</tr>
<tr>
<td>2</td>
<td>4.0</td>
</tr>
<tr>
<td>3</td>
<td>NaN</td>
</tr>
<tr>
<td>4</td>
<td>6.0</td>
</tr>
<tr>
<td>5</td>
<td>8.0</td>
</tr>
</tbody>
</table>

dtype: float64
```python
In [8]: df = pd.DataFrame(np.random.randn(6,4), index=dates, columns=list('ABCD'))

In [9]: df
Out[9]:
         A       B       C      D
2013-01-01  0.469112 -0.282863 -1.509059 -1.135632
2013-01-02  1.212112 -0.173215  0.119209 -1.044236
2013-01-03 -0.861849 -2.104569 -0.494929  1.071804
2013-01-04  0.721555 -0.706771 -1.039575  0.271860
2013-01-05 -0.424972  0.567020  0.276232 -1.087401
2013-01-06 -0.673690  0.113648 -1.478427  0.524988
```
**Pandas**

```python
In [10]: df2 = pd.DataFrame({
    'A' : 1.,
    # Other columns...
    'B' : pd.Timestamp('20130102'),
    'C' : pd.Series(1,index=list(range(4)),dtype='float32'),
    'D' : np.array([3] * 4,dtype='int32'),
    'E' : pd.Categorical(['test','train','test','train']),
    'F' : 'foo'
})

In [11]: df2
Out[11]:
<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1.0</td>
<td>2013-01-02</td>
<td>1.0</td>
<td>3</td>
<td>test</td>
<td>foo</td>
</tr>
<tr>
<td>1</td>
<td>1.0</td>
<td>2013-01-02</td>
<td>1.0</td>
<td>3</td>
<td>train</td>
<td>foo</td>
</tr>
<tr>
<td>2</td>
<td>1.0</td>
<td>2013-01-02</td>
<td>1.0</td>
<td>3</td>
<td>test</td>
<td>foo</td>
</tr>
<tr>
<td>3</td>
<td>1.0</td>
<td>2013-01-02</td>
<td>1.0</td>
<td>3</td>
<td>train</td>
<td>foo</td>
</tr>
</tbody>
</table>
```
Pandas

[27] df2.columns

Index(['A', 'B', 'C', 'D', 'E', 'F'], dtype='object')
## Pandas

**In [14]:** `df.head()`  
**Out[14]:**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>2013-01-01</td>
<td>0.469112</td>
<td>-0.282863</td>
<td>-1.509059</td>
</tr>
<tr>
<td>2013-01-02</td>
<td>1.212112</td>
<td>-0.173215</td>
<td>0.119209</td>
</tr>
<tr>
<td>2013-01-03</td>
<td>-0.861849</td>
<td>-2.104569</td>
<td>-0.494929</td>
</tr>
<tr>
<td>2013-01-04</td>
<td>0.721555</td>
<td>-0.706771</td>
<td>-1.039575</td>
</tr>
<tr>
<td>2013-01-05</td>
<td>-0.424972</td>
<td>0.567020</td>
<td>0.276232</td>
</tr>
</tbody>
</table>

**In [15]:** `df.tail(3)`  
**Out[15]:**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>2013-01-04</td>
<td>0.721555</td>
<td>-0.706771</td>
<td>-1.039575</td>
</tr>
<tr>
<td>2013-01-05</td>
<td>-0.424972</td>
<td>0.567020</td>
<td>0.276232</td>
</tr>
<tr>
<td>2013-01-06</td>
<td>-0.673690</td>
<td>0.113648</td>
<td>-1.478427</td>
</tr>
</tbody>
</table>
### Pandas

```py
In [19]: df.describe()
Out[19]:
```

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>6.0000000</td>
<td>6.0000000</td>
<td>6.0000000</td>
<td>6.0000000</td>
</tr>
<tr>
<td>mean</td>
<td>0.073711</td>
<td>-0.431125</td>
<td>-0.687758</td>
<td>-0.233103</td>
</tr>
<tr>
<td>std</td>
<td>0.843157</td>
<td>0.922818</td>
<td>0.779887</td>
<td>0.973118</td>
</tr>
<tr>
<td>min</td>
<td>-0.861849</td>
<td>-2.104569</td>
<td>-1.509059</td>
<td>-1.135632</td>
</tr>
<tr>
<td>25%</td>
<td>-0.611510</td>
<td>-0.600794</td>
<td>-1.368714</td>
<td>-1.076610</td>
</tr>
<tr>
<td>50%</td>
<td>0.022070</td>
<td>-0.228039</td>
<td>-0.767252</td>
<td>-0.386188</td>
</tr>
<tr>
<td>75%</td>
<td>0.658444</td>
<td>0.041933</td>
<td>-0.034326</td>
<td>0.461706</td>
</tr>
<tr>
<td>max</td>
<td>1.212112</td>
<td>0.567020</td>
<td>0.276232</td>
<td>1.071804</td>
</tr>
</tbody>
</table>
### Pandas

**In [22]:** `df.sort_values(by='B')`

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>2013-01-03</td>
<td>-0.861849</td>
<td>-2.104569</td>
<td>-0.494929</td>
<td>1.071804</td>
</tr>
<tr>
<td>2013-01-04</td>
<td>0.721555</td>
<td>-0.706771</td>
<td>-1.039575</td>
<td>0.271860</td>
</tr>
<tr>
<td>2013-01-01</td>
<td>0.469112</td>
<td>-0.282863</td>
<td>-1.509059</td>
<td>-1.135632</td>
</tr>
<tr>
<td>2013-01-02</td>
<td>1.212112</td>
<td>-0.173215</td>
<td>0.119209</td>
<td>-1.044236</td>
</tr>
<tr>
<td>2013-01-06</td>
<td>-0.673690</td>
<td>0.113648</td>
<td>-1.478427</td>
<td>0.524988</td>
</tr>
<tr>
<td>2013-01-05</td>
<td>-0.424972</td>
<td>0.567020</td>
<td>0.276232</td>
<td>-1.087401</td>
</tr>
</tbody>
</table>
### Pandas

**In [23]:** df['A']

**Out[23]:**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2013-01-01</td>
<td>0.4691112</td>
</tr>
<tr>
<td>2013-01-02</td>
<td>1.212112</td>
</tr>
<tr>
<td>2013-01-03</td>
<td>-0.861849</td>
</tr>
<tr>
<td>2013-01-04</td>
<td>0.721555</td>
</tr>
<tr>
<td>2013-01-05</td>
<td>-0.424972</td>
</tr>
<tr>
<td>2013-01-06</td>
<td>-0.673690</td>
</tr>
</tbody>
</table>

Freq: D, Name: A, dtype: float64
In [24]: df[0:3]

Out[24]:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>2013-01-01</td>
<td>0.469112</td>
<td>-0.282863</td>
<td>-1.509059</td>
<td>-1.135632</td>
</tr>
<tr>
<td>2013-01-02</td>
<td>1.212112</td>
<td>-0.173215</td>
<td>0.119209</td>
<td>-1.044236</td>
</tr>
<tr>
<td>2013-01-03</td>
<td>-0.861849</td>
<td>-2.104569</td>
<td>-0.494929</td>
<td>1.071804</td>
</tr>
</tbody>
</table>
In [39]: df[df.A > 0]

Out[39]:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>2013-01-01</td>
<td>0.469112</td>
<td>-0.282863</td>
<td>-1.509059</td>
<td>-1.135632</td>
</tr>
<tr>
<td>2013-01-02</td>
<td>1.212112</td>
<td>-0.173215</td>
<td>0.119209</td>
<td>-1.044236</td>
</tr>
<tr>
<td>2013-01-04</td>
<td>0.721555</td>
<td>-0.706771</td>
<td>-1.039575</td>
<td>0.271860</td>
</tr>
</tbody>
</table>
In [51]: df

Out[51]:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>2013-01-01</td>
<td>0.000000</td>
<td>0.000000</td>
<td>-1.509059</td>
<td>5</td>
<td>NaN</td>
</tr>
<tr>
<td>2013-01-02</td>
<td>1.212112</td>
<td>-0.173215</td>
<td>0.119209</td>
<td>5</td>
<td>1.0</td>
</tr>
<tr>
<td>2013-01-03</td>
<td>-0.861849</td>
<td>-2.104569</td>
<td>-0.494929</td>
<td>5</td>
<td>2.0</td>
</tr>
<tr>
<td>2013-01-04</td>
<td>0.721555</td>
<td>-0.706771</td>
<td>-1.039575</td>
<td>5</td>
<td>3.0</td>
</tr>
<tr>
<td>2013-01-05</td>
<td>-0.424972</td>
<td>0.567020</td>
<td>0.276232</td>
<td>5</td>
<td>4.0</td>
</tr>
<tr>
<td>2013-01-06</td>
<td>-0.673690</td>
<td>0.113648</td>
<td>-1.478427</td>
<td>5</td>
<td>5.0</td>
</tr>
</tbody>
</table>
In [51]: df
Out[51]:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>2013-01-01</td>
<td>0.000000</td>
<td>0.000000</td>
<td>-1.509059</td>
<td>5.000000</td>
<td>NaN</td>
</tr>
<tr>
<td>2013-01-02</td>
<td>1.212112</td>
<td>-0.173215</td>
<td>0.119209</td>
<td>5.000000</td>
<td>1.000000</td>
</tr>
<tr>
<td>2013-01-03</td>
<td>-0.861849</td>
<td>-2.104569</td>
<td>-0.494929</td>
<td>5.000000</td>
<td>2.000000</td>
</tr>
<tr>
<td>2013-01-04</td>
<td>0.721555</td>
<td>-0.706771</td>
<td>-1.039575</td>
<td>5.000000</td>
<td>3.000000</td>
</tr>
<tr>
<td>2013-01-05</td>
<td>-0.424972</td>
<td>0.567020</td>
<td>0.276232</td>
<td>5.000000</td>
<td>4.000000</td>
</tr>
<tr>
<td>2013-01-06</td>
<td>-0.673690</td>
<td>0.113648</td>
<td>-1.478427</td>
<td>5.000000</td>
<td>5.000000</td>
</tr>
</tbody>
</table>

In [52]: df2 = df.copy()

In [53]: df2[df2 > 0] = -df2

In [54]: df2
Out[54]:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>2013-01-01</td>
<td>0.000000</td>
<td>0.000000</td>
<td>-1.509059</td>
<td>-5.000000</td>
<td>NaN</td>
</tr>
<tr>
<td>2013-01-02</td>
<td>-1.212112</td>
<td>-0.173215</td>
<td>-0.119209</td>
<td>-5.000000</td>
<td>-1.000000</td>
</tr>
<tr>
<td>2013-01-03</td>
<td>-0.861849</td>
<td>-2.104569</td>
<td>-0.494929</td>
<td>-5.000000</td>
<td>-2.000000</td>
</tr>
<tr>
<td>2013-01-04</td>
<td>-0.721555</td>
<td>-0.706771</td>
<td>-1.039575</td>
<td>-5.000000</td>
<td>-3.000000</td>
</tr>
<tr>
<td>2013-01-05</td>
<td>-0.424972</td>
<td>-0.567020</td>
<td>-0.276232</td>
<td>-5.000000</td>
<td>-4.000000</td>
</tr>
<tr>
<td>2013-01-06</td>
<td>-0.673690</td>
<td>-0.113648</td>
<td>-1.478427</td>
<td>-5.000000</td>
<td>-5.000000</td>
</tr>
</tbody>
</table>
Useful Functions

df.values # converts a pandas table to a 2D numpy array

df.iloc[[0,2], [1,3]] # returns rows 1-2, columns 1-3 of df

df.loc[df['A'] > 4] # returns rows where A is greater than 4
Pandas

https://pandas.pydata.org/pandas-docs/stable/getting_started/10min.html

import pandas as pd
dataset = pd.read_csv("nbaallelo.csv")
dataset.describe()
Matplotlib

- Package for plotting data.

import matplotlib.pyplot as plt
Matplotlib

- Package for plotting data.

- Collection of functions that make changes to a figure.
- The package keeps track of the current figure,
- Therefore changes are all to the same figure.
Matplotlib

- Package for plotting data.

```python
import matplotlib.pyplot as plt

x = np.arange(5)
y = x ** 2
plt.plot(x, y)
```
Matplotlib

- Package for plotting data.

```python
import matplotlib.pylab as plt

plt.figure()  # Let's start plotting on a new figure.
```
Matplotlib

- Package for plotting data.

```python
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(5)
y = x ** 2
plt.plot(x, y)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Matplotlib Demo!")
```

Matplotlib Demo!
Matplotlib

- Package for plotting data.

```python
import matplotlib.pyplot as plt

x = np.arange(5)
y = x ** 2
plt.scatter(x, y)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Matplotlib Demo!")
```
Matplotlib

- For most functions that plot data,
- They accept a **format string** after the data.

```python
plt.plot(x, y, "b-")
```
Matplotlib

- For most functions that plot data,
- They accept a **format string** after the data.

```python
plt.plot(x, y, "b-")
```
- "b-" is the default.
- The pattern of the string is:
  - Color (b: blue),
  - Pattern (-: straight line).
Matplotlib

`plt.plot(x, y, "b-")`

<table>
<thead>
<tr>
<th>character</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>'b'</td>
<td>blue</td>
</tr>
<tr>
<td>'g'</td>
<td>green</td>
</tr>
<tr>
<td>'r'</td>
<td>red</td>
</tr>
<tr>
<td>'c'</td>
<td>cyan</td>
</tr>
<tr>
<td>'m'</td>
<td>magenta</td>
</tr>
<tr>
<td>'y'</td>
<td>yellow</td>
</tr>
<tr>
<td>'k'</td>
<td>black</td>
</tr>
<tr>
<td>'w'</td>
<td>white</td>
</tr>
</tbody>
</table>
Matplotlib

```python
plt.plot(x, y, "b-")
```

<table>
<thead>
<tr>
<th>character</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>'b'</td>
<td>blue</td>
</tr>
<tr>
<td>'g'</td>
<td>green</td>
</tr>
<tr>
<td>'r'</td>
<td>red</td>
</tr>
<tr>
<td>'c'</td>
<td>cyan</td>
</tr>
<tr>
<td>'m'</td>
<td>magenta</td>
</tr>
<tr>
<td>'y'</td>
<td>yellow</td>
</tr>
<tr>
<td>'k'</td>
<td>black</td>
</tr>
<tr>
<td>'w'</td>
<td>white</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>character</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'-'</td>
<td>solid line style</td>
</tr>
<tr>
<td>'--'</td>
<td>dashed line style</td>
</tr>
<tr>
<td>'-.'</td>
<td>dash-dot line style</td>
</tr>
<tr>
<td>':'</td>
<td>dotted line style</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>character</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'.'</td>
<td>point marker</td>
</tr>
<tr>
<td>','</td>
<td>pixel marker</td>
</tr>
<tr>
<td>'o'</td>
<td>circle marker</td>
</tr>
<tr>
<td>'v'</td>
<td>triangle_down marker</td>
</tr>
<tr>
<td>'^'</td>
<td>triangle_up marker</td>
</tr>
<tr>
<td>'&lt;'</td>
<td>triangle_left marker</td>
</tr>
<tr>
<td>'&gt;'</td>
<td>triangle_right marker</td>
</tr>
<tr>
<td>'1'</td>
<td>tri_down marker</td>
</tr>
<tr>
<td>'2'</td>
<td>tri_up marker</td>
</tr>
</tbody>
</table>
Colors

- The world is more diverse than a Crayola 8-pack.
Colors

- **HTML Color Codes**: method for describing colors used in websites.
- Specify how much of each primary color to use in mixed color (R, G, B).
Colors

- **HTML Color Codes**: method for describing colors used in websites.
- Specify how much of each primary color to use in mixed color (R, G, B).
- **Slang**: hex, hex-code.

```
"#RRGGBB"
```
Colors

- **HTML Color Codes**: method for describing colors used in websites.
- Specify how much of each primary color to use in mixed color (R, G, B).
- **Slang**: hex, hex-code.

```
"#RRGGBB"
```

[0, 255]
Colors

- **HTML Color Codes**: method for describing colors used in websites.
- Specify how much of each primary color to use in mixed color (R, G, B).
- *Slang*: hex, hex-code.

```
```
```
# Flat Design Color Chart

<table>
<thead>
<tr>
<th>Color Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#F9323F</td>
<td>Coral Red</td>
</tr>
<tr>
<td>#F9522D</td>
<td>Tomato Red</td>
</tr>
<tr>
<td>#F9D22D</td>
<td>Peach</td>
</tr>
<tr>
<td>#F9F122</td>
<td>Light Pink</td>
</tr>
<tr>
<td>#FAB5D0</td>
<td>Medium Orchid</td>
</tr>
<tr>
<td>#F200D0</td>
<td>Red Orange</td>
</tr>
<tr>
<td>#F90808</td>
<td>Dark Red</td>
</tr>
<tr>
<td>#F96C6C</td>
<td>Medium Pink</td>
</tr>
<tr>
<td>#F9B1B1</td>
<td>Light Purple</td>
</tr>
<tr>
<td>#F9C5CC</td>
<td>White Smoke</td>
</tr>
<tr>
<td>#F9D0D0</td>
<td>Silver</td>
</tr>
<tr>
<td>#F9D2D2</td>
<td>Light Gray</td>
</tr>
<tr>
<td>#F9D3D3</td>
<td>Gray</td>
</tr>
<tr>
<td>#F9D4D4</td>
<td>Dark Gray</td>
</tr>
<tr>
<td>#F9D5D5</td>
<td>Black</td>
</tr>
</tbody>
</table>

**16.7m colors**
Matplotlib

- Package for plotting data.

```python
import matplotlib.pyplot as plt
x = np.arange(5)
y = x ** 2
plt.plot(x, y,
    color="#D7BDE2")
```
Matplotlib

- Package for plotting data.

```python
import matplotlib.pyplot as plt

x = np.arange(5)
y = x ** 2
plt.plot(x, y,
         color="#D7BDE2",
         label="Purple")
plt.legend()
```
Matplotlib

import matplotlib.pylab as plt

x = np.arange(5)
y = x ** 2
plt.plot(x, y,
         color="#D7BDE2",
         label="Purple")
plt.plot(x, x**3,
         label="Blue")
plt.legend()
Matplotlib

import matplotlib.pylab as plt

x = np.arange(5)
y = x ** 2
plt.plot(x, y,
        color="#D7BDE2",
        label="Purple")

plt.figure()

plt.plot(x, x**3,
        label="Blue")
plt.legend()
Demo
Other Useful Packages

- **SciPy**
  - Scientific computing package often used in conjunction with the others

- **Scikit-Learn**
  - Machine learning packages for classification, regression, model selection, etc.
  - Algorithms include linear regression, k-nearest neighbors, support vector machines, etc.

- **PyTorch**
  - User-friendly deep learning python package
We’re done! Now what?

- Learning Python for 3 hours through slides and demos is not enough
- Go home and **practice**!
- Continue learning what different packages have to offer
- Try Python out to analyze some datasets you might have
  - Get comfortable with pandas, numpy, and matplotlib