

Python Tutorial: Python for Data Science

By Fahad Kamran



Overview

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

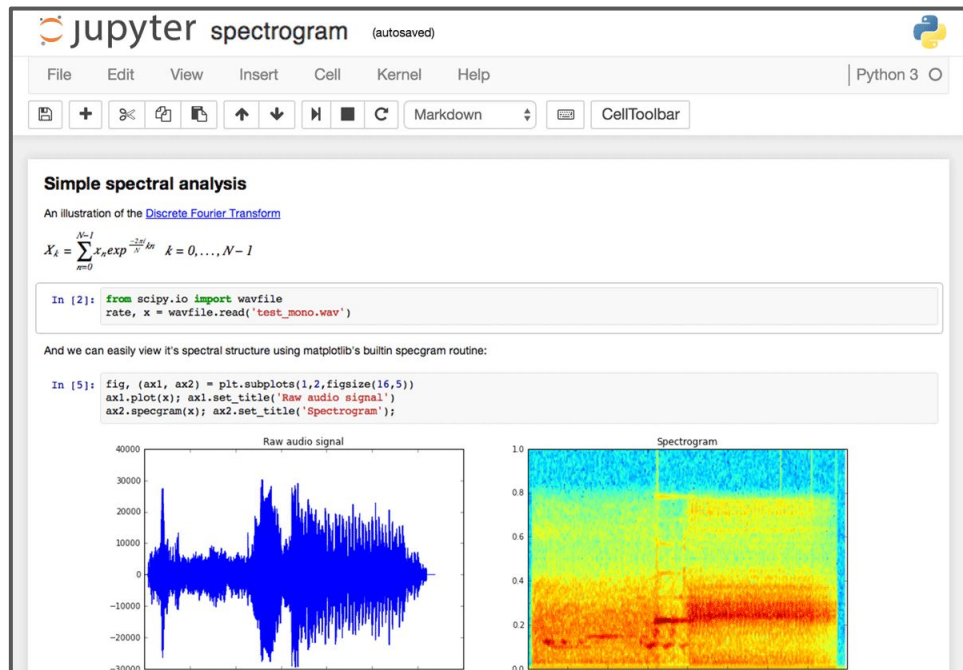
Overview

- **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.

Jupyter Notebook



```
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 =

```
x = 3
```

```
print(x)
```

3

```
x = 3 + 3
```

```
print(x)
```

6

Variables

`x = 3` **Int:** Integers

`pi = 3.14` **Float:** Real numbers

`day = "Monday"` **String:** Characters, *different* than a variable.

`is_monday = True` **Boolean:** True/False.

`# Objects.` **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

```
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

```
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]
```

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 .

```
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

```
x = [1]
```

```
print(x)
```

```
[1]
```

```
x[0] = 'new element!'
```

```
print(x)
```

```
['new element!']
```


docs.python.org/3/tutorial/datastructures.html

Python » English » 3.6.6rc1 » Documentation » The Python Tutorial »

Table Of Contents

- 5. Data Structures
 - 5.1. More on Lists
 - 5.1.1. Using Lists as Stacks
 - 5.1.2. Using Lists as Queues
 - 5.1.3. List Comprehensions
 - 5.1.4. Nested List Comprehensions
 - 5.2. The `del` statement
 - 5.3. Tuples and Sequences
 - 5.4. Sets
 - 5.5. Dictionaries
 - 5.6. Looping Techniques
 - 5.7. More on Conditions
 - 5.8. Comparing Sequences and Other Types

Previous topic

4. More Control Flow Tools

Next topic

6. Modules

This Page

[Report a Bug](#)
[Show Source](#)

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!

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

```
a["key"] = "value"
```

```
print(a)
```

```
{"key": "value"}
```

Dictionary / Map

```
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?

```
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?

```
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?

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

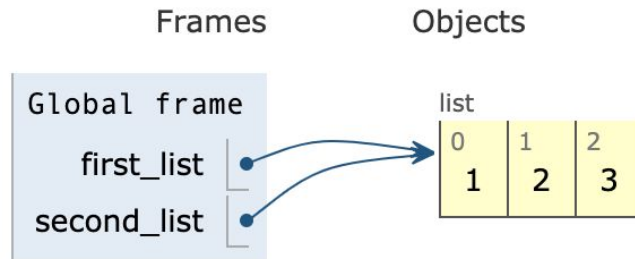
Changing *y* still changes *x*

Creating Copies

- What does an assignment statement do?

```
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

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

Creating Copies

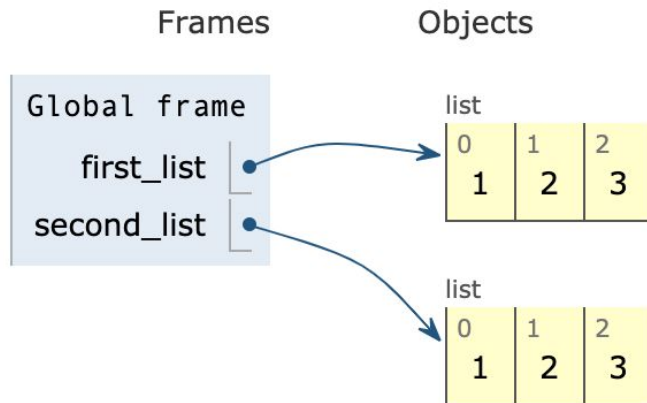
- Fixing the assignment issue

```
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

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



Exercise #1

- What are the outputs of each line of code?

```
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

```
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

About to define a function



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

Functions

Name of the function.



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

Functions

Parameters: Stuff you may need to compute with.



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

Functions

```
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`

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)
```

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

```
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.

```
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.

COLONS

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]
```

```
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

```
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]
```

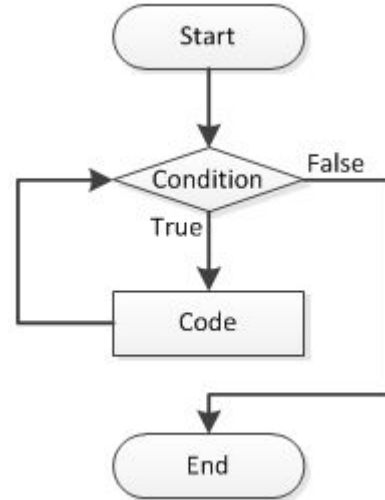
While-Loop

- Perform operations until condition is met.

```
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.

```
import math
```

Module

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

```
import math
```

```
math.sqrt(16)
```

```
4
```

Module

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

```
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.

```
import numpy as np  
x = 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.

```
import numpy as np
```

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

```
np.dot([1, 2], [1, 2])  # Dot Product.
```

NumPy Array with Operators

```
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

```
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/num
py-1.13.0/reference/routines.stat
istics.html](https://docs.scipy.org/doc/num
py-1.13.0/reference/routines.stat
istics.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

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

Array Indexing

```
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

```
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.

Pandas

```
[24] s = pd.Series([1, 3, 4, np.nan, 6, 8])  
      print(s)
```

```
0    1.0  
1    3.0  
2    4.0  
3    NaN  
4    6.0  
5    8.0  
dtype: float64
```

Pandas

```
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

```
In [10]: df2 = pd.DataFrame({ 'A' : 1.,
    ....:                      '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]:
```

	A	B	C	D	E	F
0	1.0	2013-01-02	1.0	3	test	foo
1	1.0	2013-01-02	1.0	3	train	foo
2	1.0	2013-01-02	1.0	3	test	foo
3	1.0	2013-01-02	1.0	3	train	foo

Pandas

```
[27] df2.columns
```

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

Pandas

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

```
Out[14]:
```

	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

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

```
Out[15]:
```

	A	B	C	D
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

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

```
Out[19]:
```

	A	B	C	D
count	6.000000	6.000000	6.000000	6.000000
mean	0.073711	-0.431125	-0.687758	-0.233103
std	0.843157	0.922818	0.779887	0.973118
min	-0.861849	-2.104569	-1.509059	-1.135632
25%	-0.611510	-0.600794	-1.368714	-1.076610
50%	0.022070	-0.228039	-0.767252	-0.386188
75%	0.658444	0.041933	-0.034326	0.461706
max	1.212112	0.567020	0.276232	1.071804

Pandas

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

```
Out[22]:
```

	A	B	C	D
2013-01-03	-0.861849	-2.104569	-0.494929	1.071804
2013-01-04	0.721555	-0.706771	-1.039575	0.271860
2013-01-01	0.469112	-0.282863	-1.509059	-1.135632
2013-01-02	1.212112	-0.173215	0.119209	-1.044236
2013-01-06	-0.673690	0.113648	-1.478427	0.524988
2013-01-05	-0.424972	0.567020	0.276232	-1.087401

Pandas

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

```
Out[23]:
```

```
2013-01-01    0.469112
```

```
2013-01-02    1.212112
```

```
2013-01-03   -0.861849
```

```
2013-01-04    0.721555
```

```
2013-01-05   -0.424972
```

```
2013-01-06   -0.673690
```

```
Freq: D, Name: A, dtype: float64
```

Pandas

```
In [24]: df[0:3]
```

```
Out[24]:
```

	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

Pandas

```
In [39]: df[df.A > 0]
```

```
Out[39]:
```

	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
2013-01-04	0.721555	-0.706771	-1.039575	0.271860

In [51]: df

Out[51]:

	A	B	C	D	F
2013-01-01	0.000000	0.000000	-1.509059	5	NaN
2013-01-02	1.212112	-0.173215	0.119209	5	1.0
2013-01-03	-0.861849	-2.104569	-0.494929	5	2.0
2013-01-04	0.721555	-0.706771	-1.039575	5	3.0
2013-01-05	-0.424972	0.567020	0.276232	5	4.0
2013-01-06	-0.673690	0.113648	-1.478427	5	5.0

In [51]: df

Out[51]:

	A	B	C	D	F
2013-01-01	0.000000	0.000000	-1.509059	5	NaN
2013-01-02	1.212112	-0.173215	0.119209	5	1.0
2013-01-03	-0.861849	-2.104569	-0.494929	5	2.0
2013-01-04	0.721555	-0.706771	-1.039575	5	3.0
2013-01-05	-0.424972	0.567020	0.276232	5	4.0
2013-01-06	-0.673690	0.113648	-1.478427	5	5.0

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

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

In [54]: df2

Out[54]:

	A	B	C	D	F
2013-01-01	0.000000	0.000000	-1.509059	-5	NaN
2013-01-02	-1.212112	-0.173215	-0.119209	-5	-1.0
2013-01-03	-0.861849	-2.104569	-0.494929	-5	-2.0
2013-01-04	-0.721555	-0.706771	-1.039575	-5	-3.0
2013-01-05	-0.424972	-0.567020	-0.276232	-5	-4.0
2013-01-06	-0.673690	-0.113648	-1.478427	-5	-5.0

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()
```

	gameorder	game_id	lg_id	_iscopy	year_id	date_game	seasongame	is_playoffs	team_id	fran_id	...	win_equiv	opp_id	opp_fran	opp_pts	opp_elo_i	opp_elo_n	game_loc
1	1	194611010TRH	NBA	1	1947	11/1/1946	1	0	NYK	Knicks	...	41.705170	TRH	Huskies	66	1300.0000	1293.2767	
3	2	194611020CHS	NBA	1	1947	11/2/1946	2	0	NYK	Knicks	...	40.692783	CHS	Stags	63	1300.0000	1309.6521	
5	3	194611020DTF	NBA	1	1947	11/2/1946	1	0	WSC	Capitols	...	43.135952	DTF	Falcons	33	1300.0000	1279.6189	
6	4	194611020PRO	NBA	1	1947	11/2/1946	1	0	BOS	Celtics	...	40.459381	PRO	Steamrollers	59	1300.0000	1305.1542	
8	5	194611020STB	NBA	1	1947	11/2/1946	1	0	PIT	Ironmen	...	40.507980	STB	Bombers	56	1300.0000	1304.6908	
10	6	194611030CLR	NBA	1	1947	11/3/1946	2	0	TRH	Huskies	...	39.548164	CLR	Rebels	71	1300.0000	1307.1233	
13	7	194611040PIT	NBA	1	1947	11/4/1946	2	0	WSC	Capitols	...	44.946995	PIT	Ironmen	56	1295.3092	1277.9456	
15	8	194611050BOS	NBA	1	1947	11/5/1946	2	0	CHS	Stags	...	42.686188	BOS	Celtics	55	1294.8458	1288.4139	
16	9	194611050DTF	NBA	1	1947	11/5/1946	2	0	STB	Bombers	...	42.347137	DTF	Falcons	49	1279.6189	1271.4624	
18	10	194611070GSW	NBA	1	1947	11/7/1946	3	0	PIT	Ironmen	...	38.201401	PHW	Warriors	81	1300.0000	1304.6670	
21	11	194611070PRO	NBA	1	1947	11/7/1946	3	0	CHS	Stags	...	42.020947	PRO	Steamrollers	73	1305.1542	1311.5032	
23	12	194611070STB	NBA	1	1947	11/7/1946	3	0	NYK	Knicks	...	41.767715	STB	Bombers	63	1312.8473	1302.5988	
25	13	194611080TRH	NBA	1	1947	11/8/1946	3	0	DTF	Falcons	...	37.697380	TRH	Huskies	73	1286.1534	1289.1691	
26	14	194611090CHS	NBA	1	1947	11/9/1946	4	0	TRH	Huskies	...	39.286152	CHS	Stags	62	1309.7350	1315.2522	
28	15	194611090DTF	NBA	1	1947	11/9/1946	3	0	BOS	Celtics	...	38.410091	DTF	Falcons	69	1268.4468	1281.5840	
31	16	194611090PRO	NBA	1	1947	11/9/1946	4	0	PIT	Ironmen	...	37.595333	PRO	Steamrollers	76	1311.5032	1317.3143	
33	17	194611090STB	NBA	1	1947	11/9/1946	3	0	WSC	Capitols	...	44.621101	STB	Bombers	70	1302.5988	1305.7323	
35	18	194611100CLR	NBA	1	1947	11/10/1946	4	0	WSC	Capitols	...	43.162102	CLR	Rebels	92	1307.1233	1321.1034	
36	19	194611110NYK	NBA	1	1947	11/11/1946	5	0	CHS	Stags	...	44.073521	NYK	Knicks	68	1307.3197	1293.2161	
39	20	194611110PIT	NBA	1	1947	11/11/1946	4	0	PRO	Steamrollers	...	41.757576	PIT	Ironmen	84	1267.4674	1277.5587	
40	21	194611130CHS	NBA	1	1947	11/13/1946	4	0	BOS	Celtics	...	37.847424	CHS	Stags	71	1329.3558	1334.7465	

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

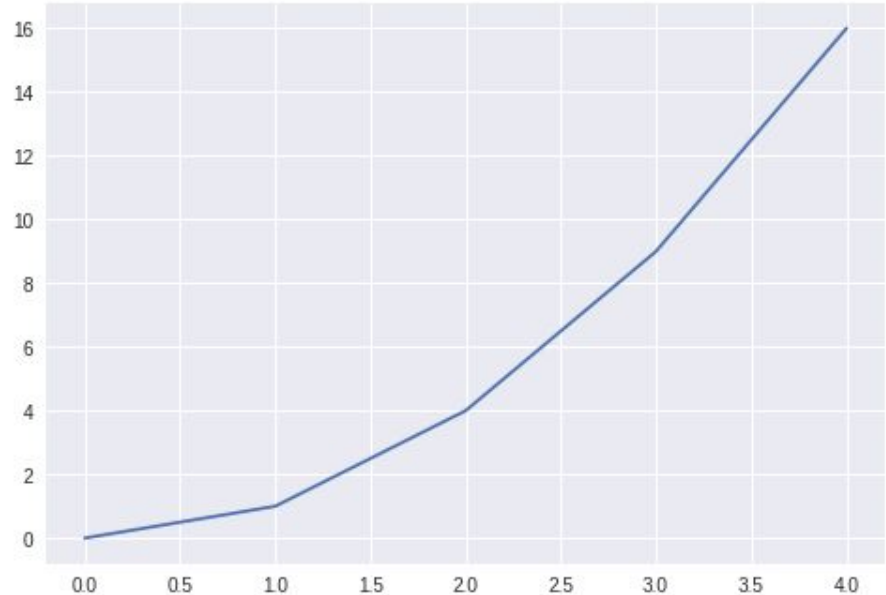
- Package for plotting data.

```
import matplotlib.pyplot as plt
```

```
x = np.arange(5)
```

```
y = x ** 2
```

```
plt.plot(x, y)
```



Matplotlib

- Package for plotting data.

```
import matplotlib.pyplot as plt
```

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

Matplotlib

- Package for plotting data.

```
import matplotlib.pyplot as plt
```

```
x = np.arange(5)
```

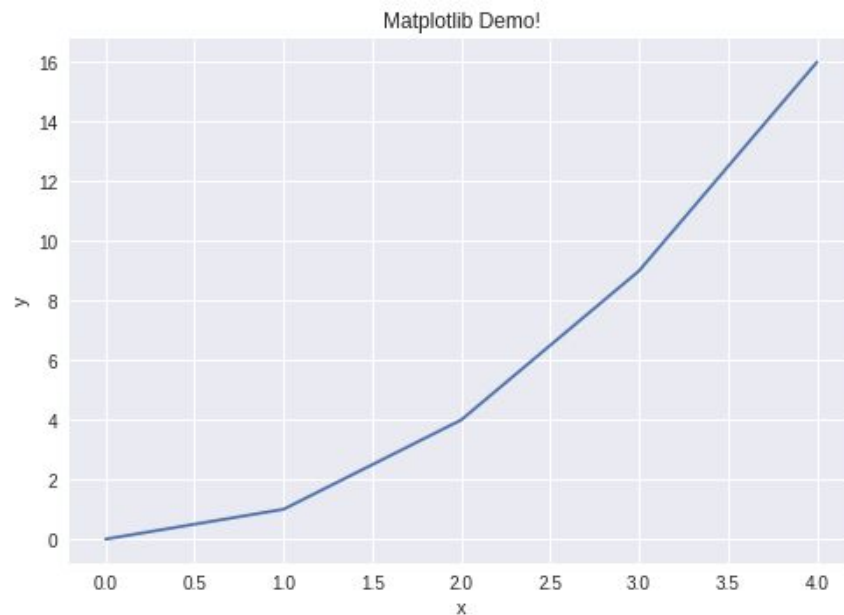
```
y = x ** 2
```

```
plt.plot(x, y)
```

```
plt.xlabel("x")
```

```
plt.ylabel("y")
```

```
plt.title("Matplotlib Demo!")
```



Matplotlib

- Package for plotting data.

```
import matplotlib.pyplot as plt
```

```
x = np.arange(5)
```

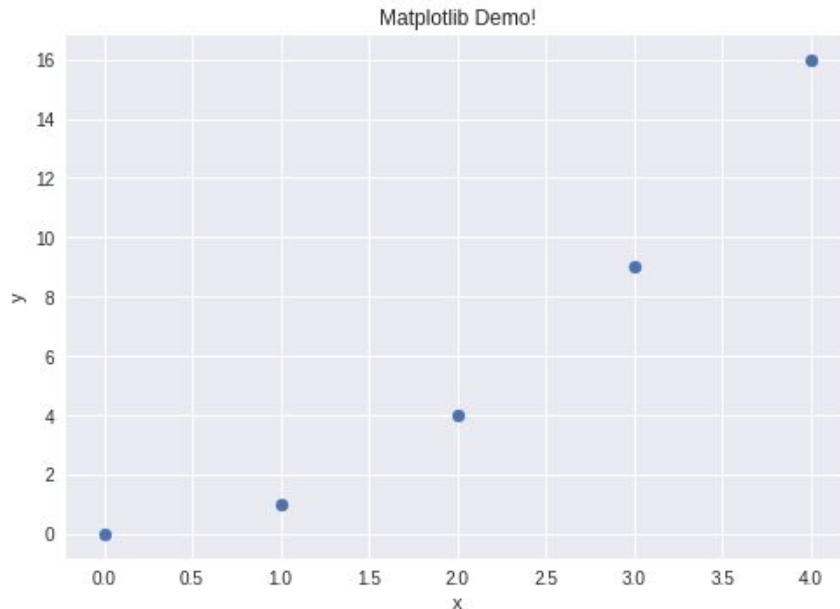
```
y = x ** 2
```

```
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.

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


Matplotlib

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

```
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-")
```

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

Matplotlib

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

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

character	description
'_'	solid line style
'--'	dashed line style
'-.'	dash-dot line style
':'	dotted line style

Markers

character	description
'.'	point marker
','	pixel marker
'o'	circle marker
'v'	triangle_down marker
'^'	triangle_up marker
'<'	triangle_left marker
'>'	triangle_right marker
'1'	tri_down marker
'2'	tri_up marker

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.

"#RRGGBB"

↑
~~[0, 255]~~
[0, FF]

Decimal number	Hexadecimal number
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F



HTML Color Codes
HTMLCOLORCODES.COM

Flat Design Color Chart

#F9BE8A rgb(249, 190, 138)	#F0DEEC rgb(240, 222, 236)	#F5EEF8 rgb(245, 238, 248)	#FAECF7 rgb(249, 234, 255)	#FAF2F8 rgb(250, 242, 248)	#EBF5FB rgb(235, 246, 253)	#EBF9F5 rgb(236, 248, 245)	#EBF6F3 rgb(235, 246, 245)	#E9F7EF rgb(233, 242, 250)	#EAF4F1 rgb(238, 245, 249)	#FEF9E7 rgb(254, 249, 239)	#FEF5E7 rgb(254, 245, 239)	#FDF2E9 rgb(253, 242, 238)	#FBEED6 rgb(253, 238, 235)	#F0F0F0 rgb(240, 240, 240)	#F8F9F9 rgb(248, 249, 249)	#F4F6F6 rgb(244, 246, 246)	#F2F4F4 rgb(242, 244, 244)	#EBEDEF rgb(233, 238, 238)	#EAECEE rgb(234, 236, 236)
#F2D7D5 rgb(242, 215, 215)	#FADBD8 rgb(252, 218, 218)	#EBDEF0 rgb(232, 232, 243)	#EBDAEF rgb(232, 218, 239)	#D4E6F1 rgb(212, 232, 241)	#D6EAF8 rgb(216, 234, 248)	#D1F2EB rgb(209, 242, 231)	#D0EE7 rgb(209, 236, 231)	#D4EFD7 rgb(212, 239, 223)	#D5F5E3 rgb(213, 245, 237)	#FCF3CF rgb(252, 243, 207)	#F0EBD0 rgb(245, 235, 218)	#FAE5D3 rgb(250, 229, 217)	#F6D0CC rgb(246, 221, 204)	#F8F0FC rgb(248, 232, 252)	#F2F3F4 rgb(242, 243, 244)	#EAEDED rgb(234, 237, 237)	#E5E8E8 rgb(229, 232, 232)	#D6DBDF rgb(216, 223, 223)	#D5D8DC rgb(216, 220, 220)
#E6B0AA rgb(230, 176, 176)	#F5B7B1 rgb(245, 183, 177)	#D7BDE2 rgb(215, 189, 230)	#D2B4DE rgb(210, 182, 222)	#A9CCE3 rgb(169, 204, 227)	#AED6F1 rgb(174, 214, 248)	#A3E4D7 rgb(163, 228, 215)	#A2D9CE rgb(165, 221, 193)	#A9D0FB rgb(173, 208, 255)	#ABEBC6 rgb(171, 235, 198)	#F9F79F rgb(249, 239, 159)	#FAD7A0 rgb(250, 215, 160)	#F5CBA7 rgb(245, 203, 165)	#EDB899 rgb(237, 187, 153)	#F7F9F9 rgb(248, 249, 249)	#E5E7E9 rgb(229, 232, 233)	#D5D8DB rgb(216, 218, 218)	#CCD1D1 rgb(204, 209, 209)	#AEB6B8 rgb(173, 182, 183)	#ABB2B9 rgb(171, 180, 180)
#D98880 rgb(217, 136, 136)	#F1948A rgb(241, 148, 138)	#C39B03 rgb(195, 157, 3)	#B8BCE rgb(189, 183, 204)	#7FB3D5 rgb(127, 179, 213)	#85CE9 rgb(135, 192, 239)	#76D7C4 rgb(118, 215, 188)	#73C6B6 rgb(115, 198, 182)	#7DCEA0 rgb(126, 206, 160)	#82E0AA rgb(130, 224, 170)	#F7DC6F rgb(247, 222, 111)	#F8C471 rgb(248, 198, 112)	#F0B27A rgb(240, 178, 122)	#E59866 rgb(229, 162, 102)	#F4F6F7 rgb(248, 246, 245)	#D7DBD0 rgb(215, 219, 212)	#BFC9CA rgb(191, 201, 202)	#B2BAB8 rgb(176, 186, 183)	#85929E rgb(133, 146, 158)	#808996 rgb(128, 139, 150)
#CD6155 rgb(205, 97, 85)	#EC7063 rgb(236, 112, 99)	#AF7AC5 rgb(175, 122, 187)	#A569BD rgb(165, 105, 189)	#5499C7 rgb(84, 155, 199)	#5DAE2 rgb(93, 173, 229)	#48C9B0 rgb(72, 201, 176)	#45B39D rgb(69, 179, 163)	#52B8E0 rgb(82, 192, 181)	#58D68D rgb(88, 214, 141)	#F4D03F rgb(244, 208, 63)	#F5B041 rgb(245, 176, 66)	#E9B84E rgb(233, 182, 78)	#DC7633 rgb(220, 118, 51)	#F0F3F4 rgb(242, 243, 244)	#CACFD2 rgb(202, 207, 203)	#AAB7B8 rgb(170, 183, 184)	#99A3A4 rgb(153, 163, 164)	#5D6D7E rgb(93, 109, 129)	#566573 rgb(86, 101, 111)
#CD3028 rgb(205, 48, 40)	#E74C3C rgb(231, 76, 60)	#9E5986 rgb(158, 89, 134)	#8E44AD rgb(142, 68, 170)	#2980B9 rgb(41, 129, 185)	#3498DB rgb(52, 152, 205)	#1ABC9C rgb(26, 186, 159)	#16A085 rgb(22, 160, 133)	#27AE60 rgb(39, 174, 96)	#2ECC71 rgb(46, 204, 115)	#F1C40F rgb(241, 196, 15)	#F39C12 rgb(243, 157, 46)	#E67E22 rgb(230, 126, 35)	#D35400 rgb(211, 84, 0)	#ECF0F1 rgb(236, 240, 241)	#BDC3C7 rgb(189, 189, 189)	#95A5A6 rgb(149, 165, 166)	#7F8C8D rgb(127, 140, 140)	#34495E rgb(52, 73, 94)	#2C3E50 rgb(44, 62, 80)
#A93226 rgb(169, 50, 38)	#CB4335 rgb(203, 67, 53)	#8B4EA0 rgb(139, 78, 160)	#7D3C98 rgb(125, 60, 152)	#2471A3 rgb(36, 113, 163)	#2E86C1 rgb(46, 134, 193)	#17A589 rgb(23, 165, 137)	#13B075 rgb(19, 176, 117)	#229954 rgb(34, 155, 86)	#2B8463 rgb(43, 132, 100)	#D4AC0D rgb(212, 173, 13)	#D68910 rgb(214, 133, 16)	#CA6F1E rgb(202, 111, 30)	#BA4A00 rgb(186, 74, 0)	#D0D3D4 rgb(208, 211, 213)	#A6ACAF rgb(166, 171, 173)	#839192 rgb(131, 144, 145)	#707B7C rgb(112, 123, 124)	#2E4053 rgb(46, 64, 83)	#273746 rgb(39, 55, 70)
#992821 rgb(159, 40, 33)	#B03A2E rgb(176, 58, 46)	#76448A rgb(118, 68, 138)	#6C3483 rgb(108, 52, 131)	#1F61BD rgb(31, 97, 189)	#2874A6 rgb(40, 116, 166)	#1ABF77 rgb(26, 193, 119)	#117A65 rgb(17, 124, 101)	#1E8449 rgb(30, 132, 73)	#229B56 rgb(34, 155, 86)	#B7950B rgb(183, 158, 11)	#B9770E rgb(189, 119, 14)	#AF601A rgb(175, 96, 26)	#A04000 rgb(160, 64, 0)	#B3B6B7 rgb(183, 183, 183)	#909497 rgb(144, 148, 151)	#717D7E rgb(113, 125, 126)	#616A6B rgb(97, 106, 107)	#283747 rgb(40, 55, 71)	#212F3D rgb(33, 47, 61)
#7B241C rgb(123, 36, 28)	#943126 rgb(148, 49, 38)	#633974 rgb(99, 57, 116)	#5B2C6F rgb(91, 44, 111)	#1A5276 rgb(26, 82, 118)	#21618C rgb(33, 97, 140)	#117864 rgb(17, 124, 101)	#0E6655 rgb(14, 102, 85)	#196F3D rgb(25, 114, 61)	#1D8348 rgb(29, 131, 72)	#9A7D0A rgb(154, 125, 10)	#9C640C rgb(156, 102, 12)	#935116 rgb(148, 81, 22)	#873600 rgb(135, 54, 0)	#979A9A rgb(151, 154, 154)	#797D7F rgb(121, 127, 127)	#5F6A6A rgb(95, 106, 106)	#515A5A rgb(81, 90, 90)	#212F3C rgb(33, 47, 60)	#1C2833 rgb(28, 40, 58)
#641E16 rgb(100, 30, 22)	#78281F rgb(120, 40, 31)	#512E5F rgb(81, 46, 95)	#4A235A rgb(74, 35, 90)	#1B4360 rgb(27, 67, 96)	#1B4F72 rgb(27, 79, 114)	#0E6251 rgb(14, 98, 81)	#0B5345 rgb(11, 83, 69)	#145A32 rgb(20, 90, 50)	#186A3B rgb(24, 106, 59)	#7D6608 rgb(126, 102, 8)	#7E5109 rgb(126, 81, 9)	#784212 rgb(120, 66, 18)	#6E2C00 rgb(110, 44, 0)	#7B7D7D rgb(123, 125, 125)	#626567 rgb(98, 101, 101)	#4D5656 rgb(77, 86, 86)	#424949 rgb(66, 73, 73)	#1B2631 rgb(27, 38, 49)	#17202A rgb(23, 32, 42)

16.7m colors

Matplotlib

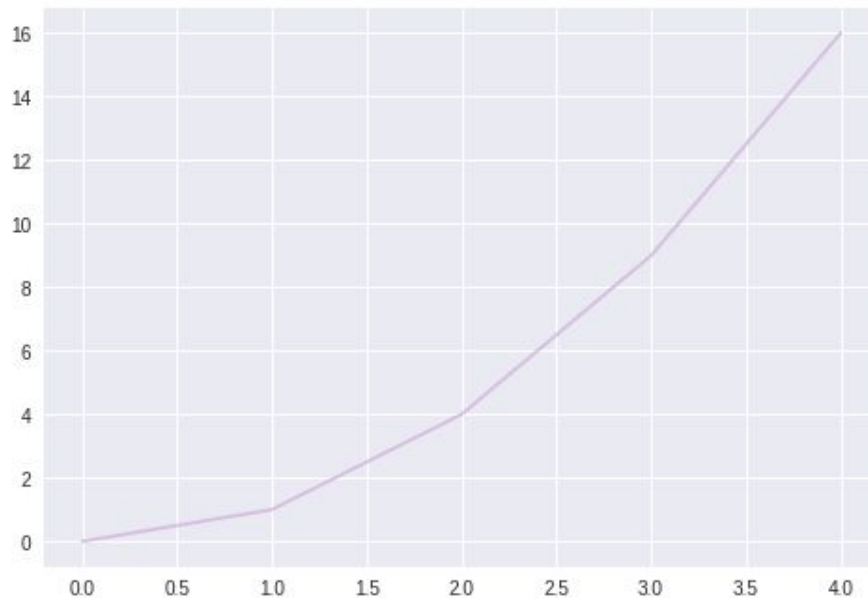
- Package for plotting data.

```
import matplotlib.pyplot as plt
```

```
x = np.arange(5)
```

```
y = x ** 2
```

```
plt.plot(x, y,  
         color="#D7BDE2")
```



Matplotlib

- Package for plotting data.

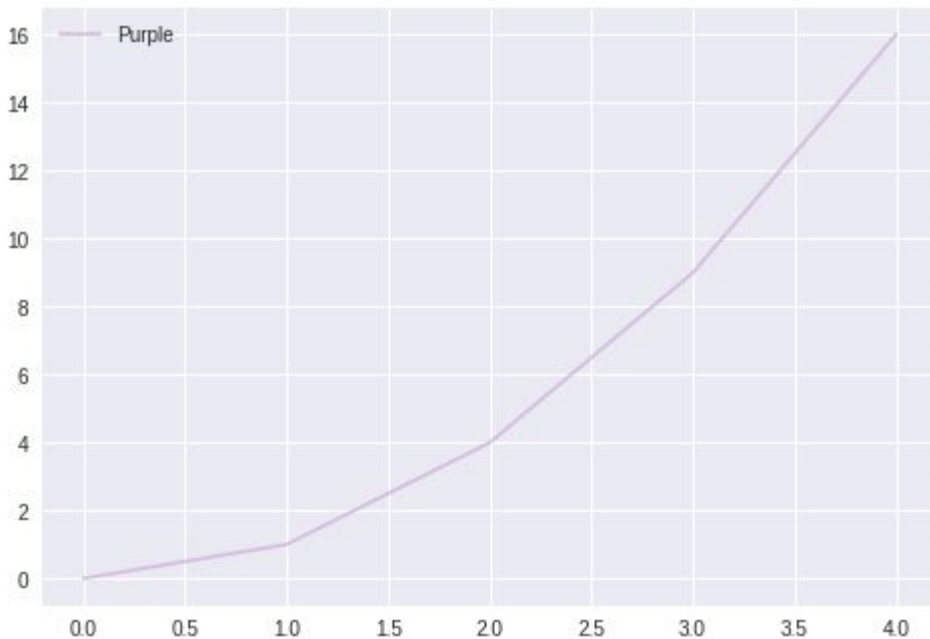
```
import matplotlib.pyplot as plt
```

```
x = np.arange(5)
```

```
y = x ** 2
```

```
plt.plot(x, y,  
         color="#D7BDE2",  
         label="Purple")
```

```
plt.legend()
```



Matplotlib

```
import matplotlib.pyplot as plt
```

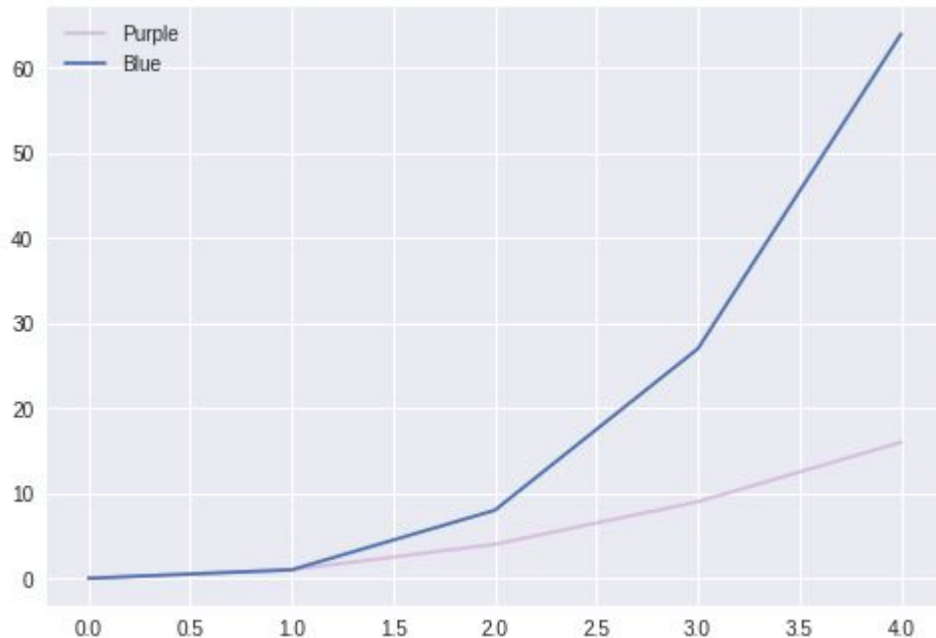
```
x = np.arange(5)
```

```
y = x ** 2
```

```
plt.plot(x, y,  
         color="#D7BDE2",  
         label="Purple")
```

```
plt.plot(x, x**3,  
         label="Blue")
```

```
plt.legend()
```



Matplotlib

```
import matplotlib.pyplot as plt
```

```
x = np.arange(5)
```

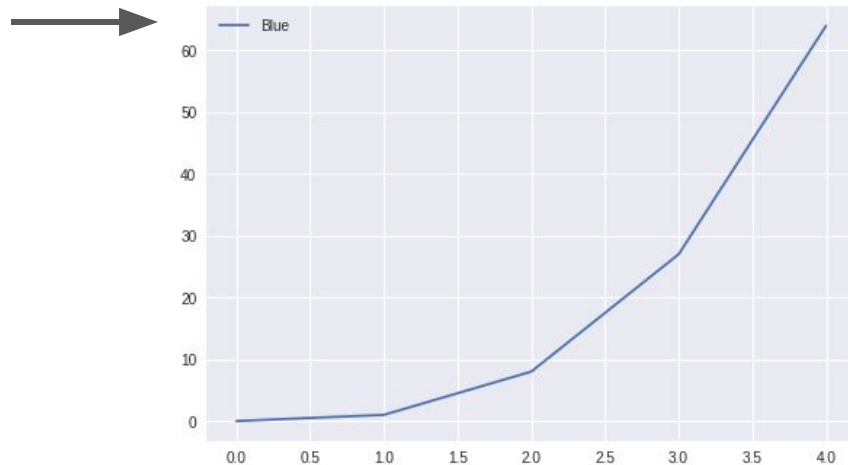
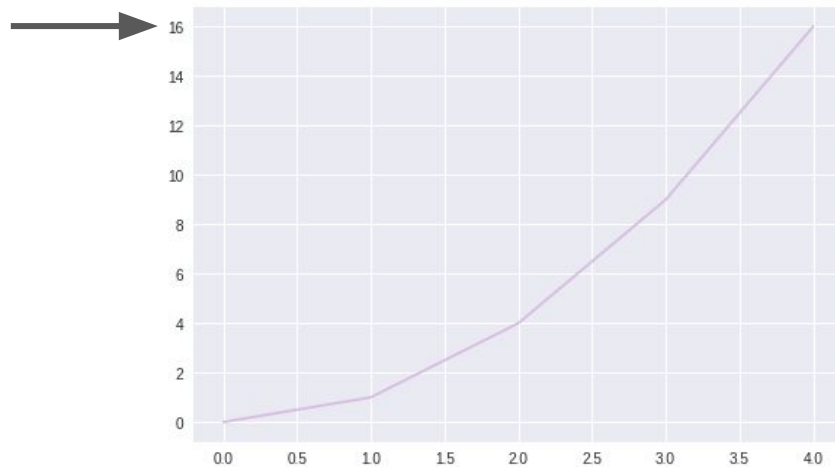
```
y = x ** 2
```

```
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