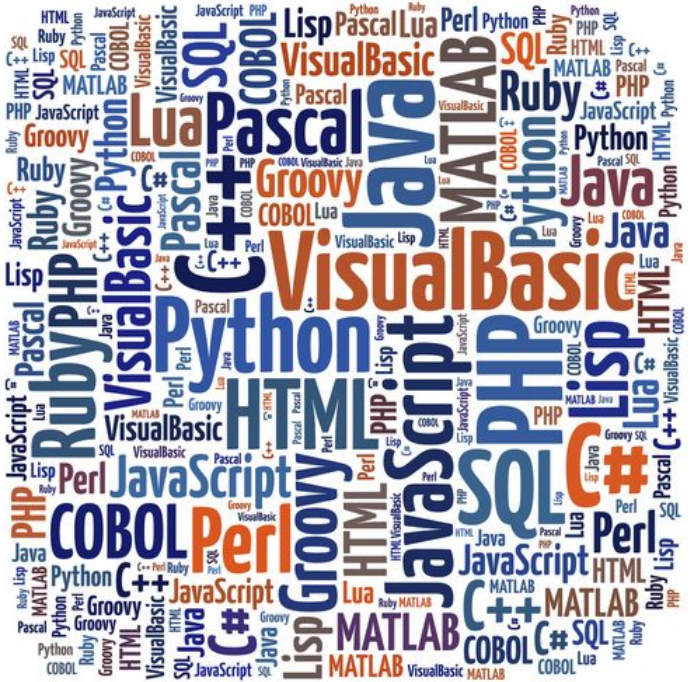
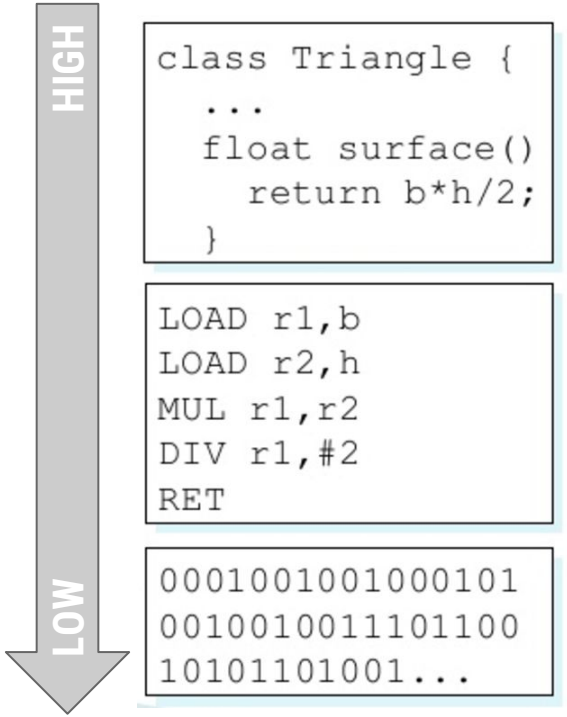
What is a programming language?

- Linguistic framework to describe computation
- Human-readable and machine-readable
- English has a vocabulary and a grammar, just like any programming language
- Why are there so many different languages?
- What makes a language “good”?



Levels of programming languages

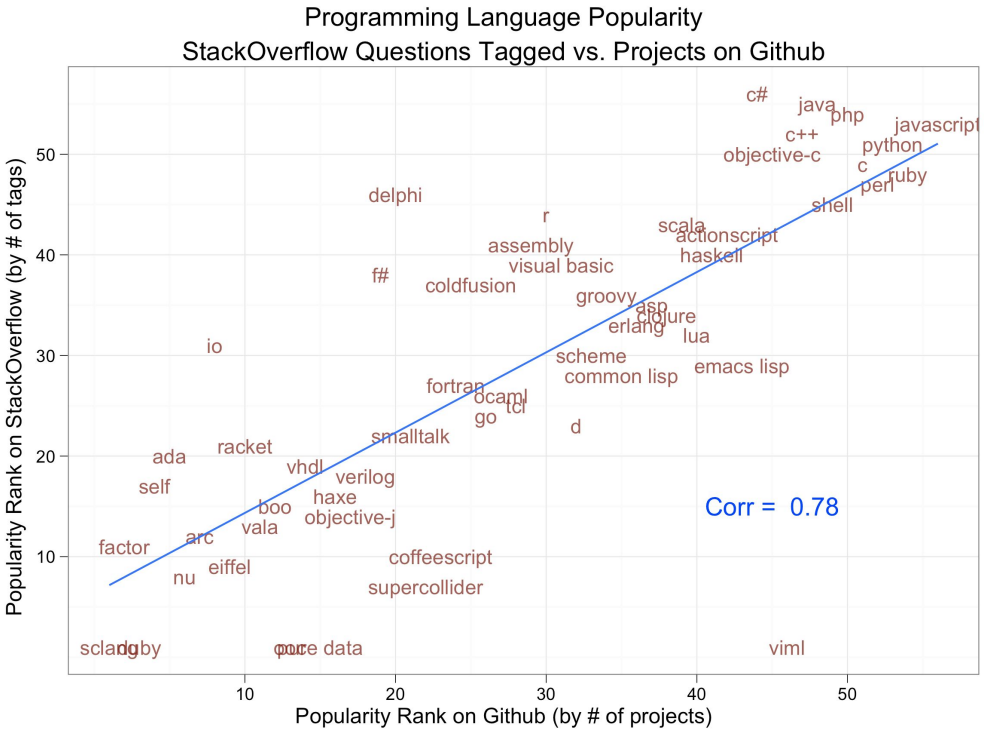


- Trade-off: raw performance vs. breadth and readability
- Nowadays speed is rarely a limitation: high-level languages with access to many libraries are preferred

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"

if __name__ == "__main__":
    app.run()
```

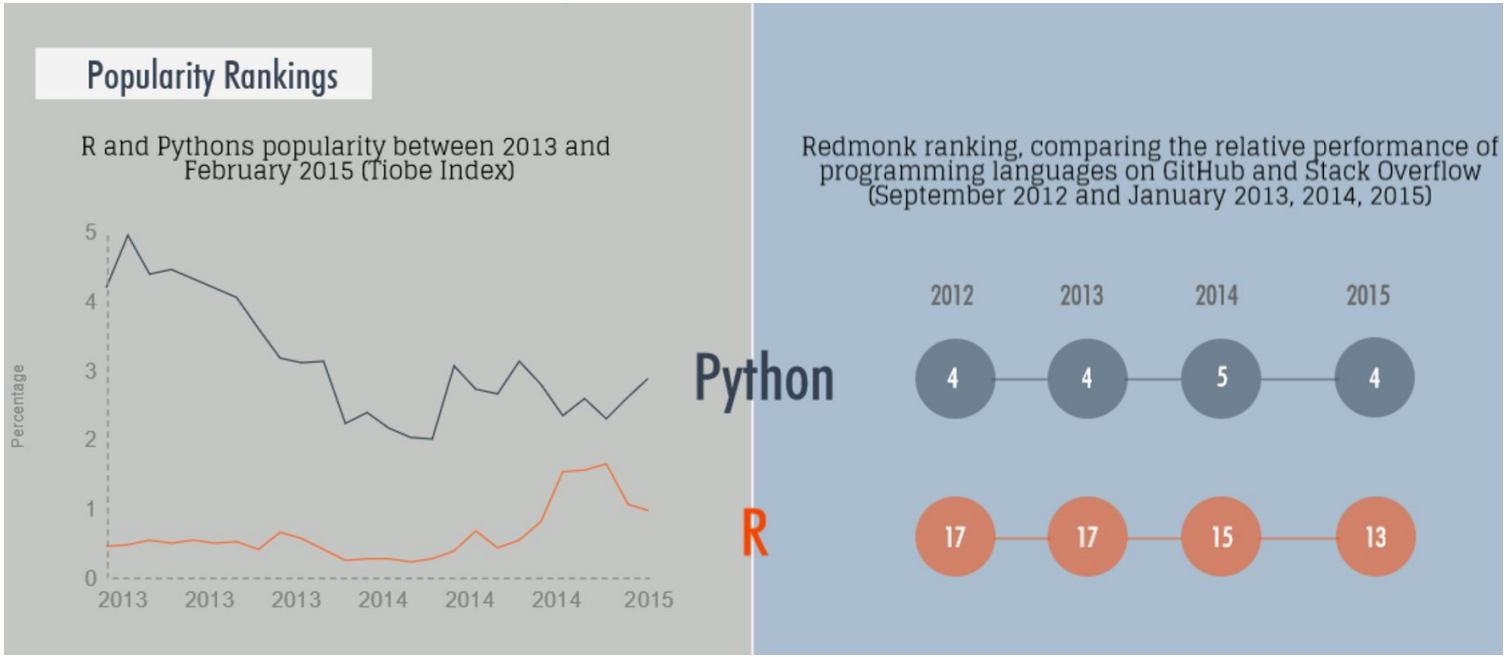
A website in 7 lines of Python!

Popularity of programming languages



- The choice of a language always depends on the application
- What matters is the penetration of a language in your target community, and the maturity of the ecosystem
- JavaScript is the best language for the web (runs in a browser)
- C++ still reigns for high-performance graphic computing (e.g. games), Java for enterprise software
- Python is probably the best all-rounder

R vs Python for data science



- No “best” choice
- R is great for data exploration and quick prototyping
- The closer you get to engineering, the better Python gets

Introduction to Python

- Multi-purpose (GUI, web, backend, scripting, etc.)
- Object-oriented: everything is an object
- Imperative syntax supported
- Interpreted
- Focus on readability and productivity

Panda3D

Python history

- Created in 1989 by Guido Van Rossum
- 1.0 released in 1994
- 2.0 released in 2000
- 3.0 released in 2008
- Current schism between Python 2 (2.7) and 3 (3.6+)

Python syntax

Python indentation

```
/* Bogus C code */
if (foo)
    if (bar)
        baz(foo, bar);
else
    qux();
```

- Most languages do not care about indentation
- Most humans do
- Here the else actually belongs to the second if statement

Python indentation

```
# Python code
if foo:
    if bar:
        baz(foo, bar)
    else:
        qux()
```

- Python uses indentation to denote structure
- Any indentation works (spaces or tabs), but it needs to be consistent

Python comments

```
# A traditional one line comment

"""
Any string not assigned to a variable is
considered a comment.
This is an example of a multi-line comment.
"""

"This is a single line comment"
```

Python data types: strings

```
# This is a string
name = "Nowell Strite"

# This is also a string
home = 'Huntington, VT'

# This is a multi-line string
sites = '''You can find me online
on sites like GitHub and Twitter.'''

# This is also a multi-line string
bio = """If you don't find me online
you can find me outside."""
```

Python data types: numbers

```
# Integers Numbers
year = 2010
year = int("2010")

# Floating Point Numbers
pi = 3.14159265
pi = float("3.14159265")

# Fixed Point Numbers
from decimal import Decimal
price = Decimal("0.02")
```

Python data types: lists

```
# Lists can be heterogeneous
favorites = []

# Appending
favorites.append(42)

# Extending
favorites.extend(["Python", True])

# Equivalent to
favorites = [42, "Python", True]
```

Python data types: lists

```
numbers = [1, 2, 3, 4, 5]

len(numbers)
# 5

numbers[0]
# 1

numbers[0:2]
# [1, 2]

numbers[2:]
# [3, 4, 5]
```

```
>>> fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
>>> fruits.count('apple')
2
>>> fruits.count('tangerine')
0
>>> fruits.index('banana')
3
>>> fruits.index('banana', 4)  # Find next banana starting a position 4
6
>>> fruits.reverse()
>>> fruits
['banana', 'apple', 'kiwi', 'banana', 'pear', 'apple', 'orange']
>>> fruits.append('grape')
>>> fruits
['banana', 'apple', 'kiwi', 'banana', 'pear', 'apple', 'orange', 'grape']
>>> fruits.sort()
>>> fruits
['apple', 'apple', 'banana', 'banana', 'grape', 'kiwi', 'orange', 'pear']
>>> fruits.pop()
'pear'
```

Python data types: tuples

```
#!/usr/bin/python

tup1 = ('physics', 'chemistry', 1997, 2000);
tup2 = (1, 2, 3, 4, 5, 6, 7 );

print "tup1[0]: ", tup1[0]
print "tup2[1:5]: ", tup2[1:5]
```

- Tuples are immutable

Python data types: sets

```
>>> a = [1, 2, 1, 3, 0, 0, 4, 7, 8, 6]
>>> b = [5, 5, 7, 8, 7, 9, 6, 1, 1, 2]
>>> s1 = set(a) # Unique numbers in s1
>>> s2 = set(b) # Unique numbers in s2
>>> s1
{0, 1, 2, 3, 4, 6, 7, 8}
>>> s2
{1, 2, 5, 6, 7, 8, 9}
>>> s1 - s2  # numbers in s1 but not in s2
{0, 3, 4}
>>> s1 | s2 # numbers in either s1 or s2
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
>>> s1 & s2 # numbers in both s1 and s2
{8, 1, 2, 6, 7}
>>> s1 ^ s2 # numbers in s1 or s2 but not both
{0, 3, 4, 5, 9}
```

- Sets are highly efficient data structures

Python data types: dictionaries

```
person = {}

# Set by key / Get by key
person['name'] = 'Nowell Strite'

# Update
person.update({
    'favorites': [42, 'food'],
    'gender': 'male',
    })

# Any immutable object can be a dictionary key
person[42] = 'favorite number'
person[(44.47, -73.21)] = 'coordinates'
```

Python data types: dictionary methods

```
person = {'name': 'Nowell', 'gender': 'Male'}

person['name']
person.get('name', 'Anonymous')
# 'Nowell Strite'

person.keys()
# ['name', 'gender']

person.values()
# ['Nowell', 'Male']

person.items()
# [['name', 'Nowell'], ['gender', 'Male']]
```

Python data types: booleans

```
# This is a boolean
is_python = True

# Everything in Python can be cast to boolean
is_python = bool("any object")

# All of these things are equivalent to False
these_are_false = False or 0 or "" or {} or []
or None

# Most everything else is equivalent to True
these_are_true = True and 1 and "Text" and
{'a': 'b'} and ['c', 'd']
```

Python arithmetic

```
a = 10          # 10
a += 1          # 11
a -= 1          # 10

b = a + 1       # 11
c = a - 1       # 9

d = a * 2       # 20
e = a / 2       # 5
f = a % 3       # 1
g = a ** 2      # 100
```

String manipulation

```
animals = "Cats " + "Dogs "
animals += "Rabbits"
# Cats Dogs Rabbits

fruit = ', '.join(['Apple', 'Banana', 'Orange'])
# Apple, Banana, Orange

date = '%s %d %d' % ('Sept', 11, 2010)
# Sept 11 2010

name = '%(first)s %(last)s' % {
    'first': 'Nowell',
    'last': 'Strite'}
# Nowell Strite
```

Logical and arithmetic comparison

```
# Logical And
a and b

# Logical Or
a or b

# Logical Negation
not a

# Compound
(a and not (b or c))
```

```
# Ordering
a > b
a >= b
a < b
a <= b

# Equality/Difference
a == b
a != b
```

Control flow: conditionals

```
grade = 82
if grade >= 90:
    if grade == 100:
        print 'A+'
    else:
        print "A"
elif grade >= 80:
    print "B"
elif grade >= 70:
    print "C"
else:
    print "F"

# B
```

Control flow: loops

```
for x in range(10): #0-9
    print x
```

```
fruits = ['Apple', 'Orange']

for fruit in fruits:
    print fruit
```

```
states = {
    'VT': 'Vermont',
    'ME': 'Maine',
    }

for key, value in states.items():
    print '%s: %s' % (key, value)
```

```
x = 0
while x < 100:
    print x
    x += 1
```

List comprehensions (replace simple for loops)

```
odds = []
for x in range(50):
    if x % 2:
        odds.append(x)
```

```
odds = [ x for x in range(50) if x % 2 ]
```

Python functions

```
def my_function():
    """Function Documentation"""
    print "Hello World"
```

Python functions

```
# Positional
def add(x, y):
    return x + y

# Keyword
def shout(phrase='Yipee!'):
    print phrase

# Positional + Keyword
def echo(text, prefix=''):
    print '%s%s' % (prefix, text)
```

```
def fib(n):
    """Return Fibonacci up to n."""
    results = []
    a, b = 0, 1
    while a < n:
        results.append(a)
        a, b = b, a + b
    return a
```

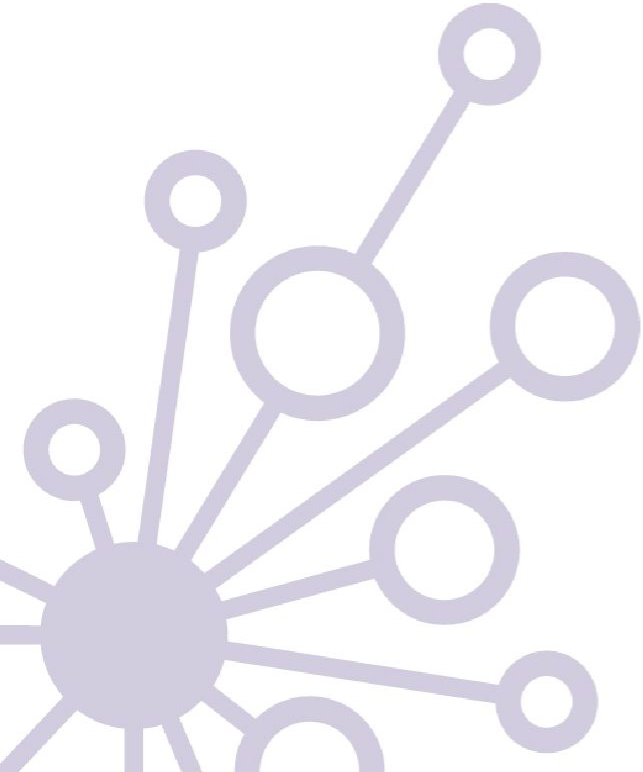
Python imports

```
# Imports the datetime module into the
# current namespace
import datetime
datetime.date.today()
datetime.timedelta(days=1)

# Imports datetime and addes date and
# timedelta into the current namespace
from datetime import date, timedelta
date.today()
timedelta(days=1)
```

```
# Renaming imports
from datetime import date
from my_module import date as my_date

# This is usually considered a big No-No
from datetime import *
```

Python error handling

```
>>> def divide(x, y):
...     try:
...         result = x / y
...     except ZeroDivisionError:
...         print "division by zero!"
...     else:
...         print "result is", result
...     finally:
...         print "executing finally clause"
...
>>> divide(2, 1)
result is 2
executing finally clause
>>> divide(2, 0)
division by zero!
executing finally clause
>>> divide("2", "1")
executing finally clause
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in divide
TypeError: unsupported operand type(s) for /: 'str' and 'str'
```



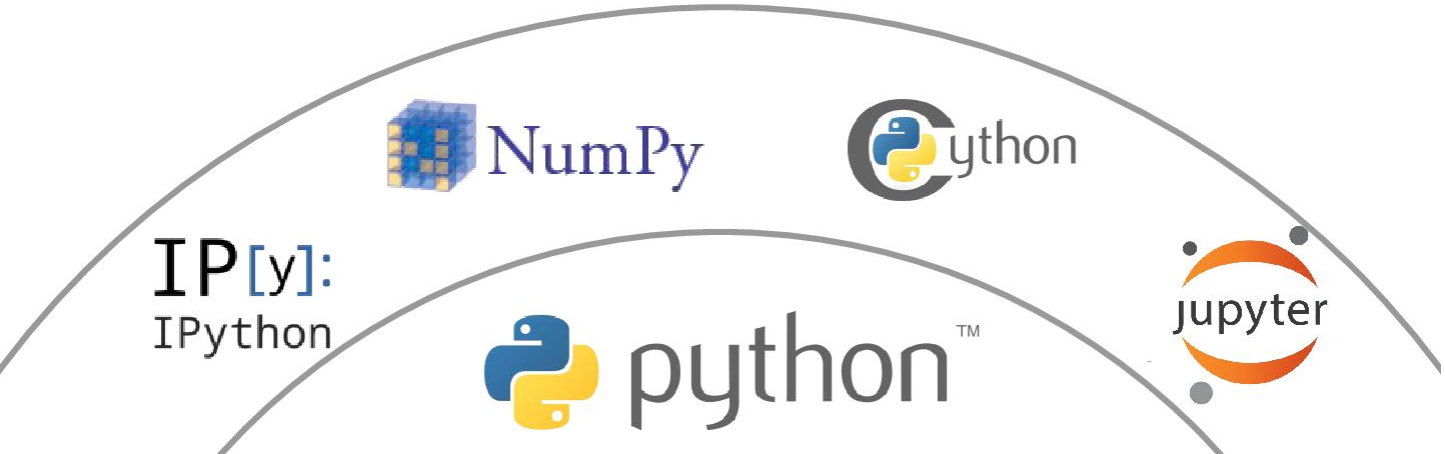
{Propulsion}

Python's Data Science Stack

Jake VanderPlas @jakevdp

JSM, July 31, 2016

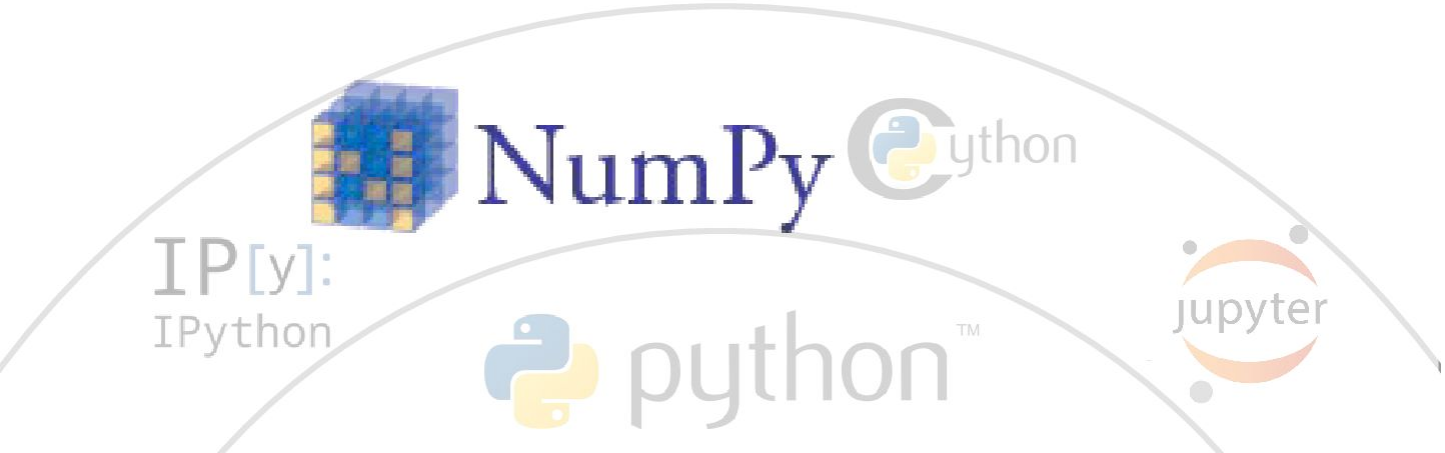
Python's Data Science Stack


NumPy
ython
IP[y]:
IPython
python™
jupyter

NumPy = Numerical Python

{Propulsion}

Efficient array storage, manipulation, and computation

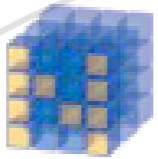


NumPy = Numerical Python

{Propulsion}

Efficient array storage, manipulation, and computation

```
In [1]: import numpy as np

        # Create a 5x5 uniform random matrix
        M = np.random.rand(5, 5)

        # Compute the SVD
        U, S, VT = np.linalg.svd(M)
        print(S)

        [ 2.46102945  0.94542853  0.53550015  0.20705388  0.13071452]
```



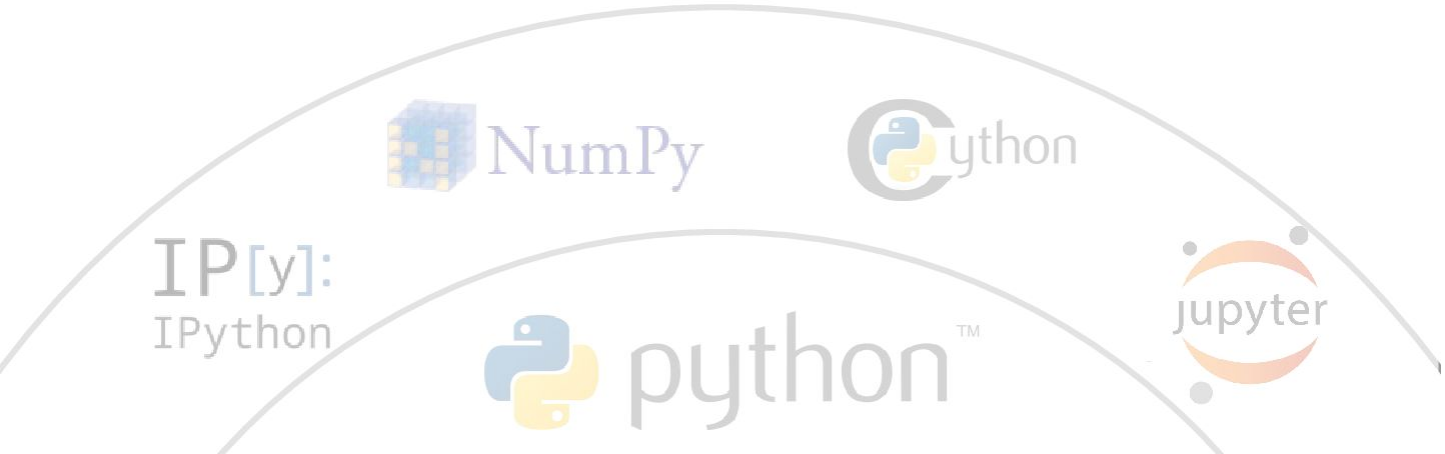
NumPy

Python

IP[y]:
IPython

python™

jupyter

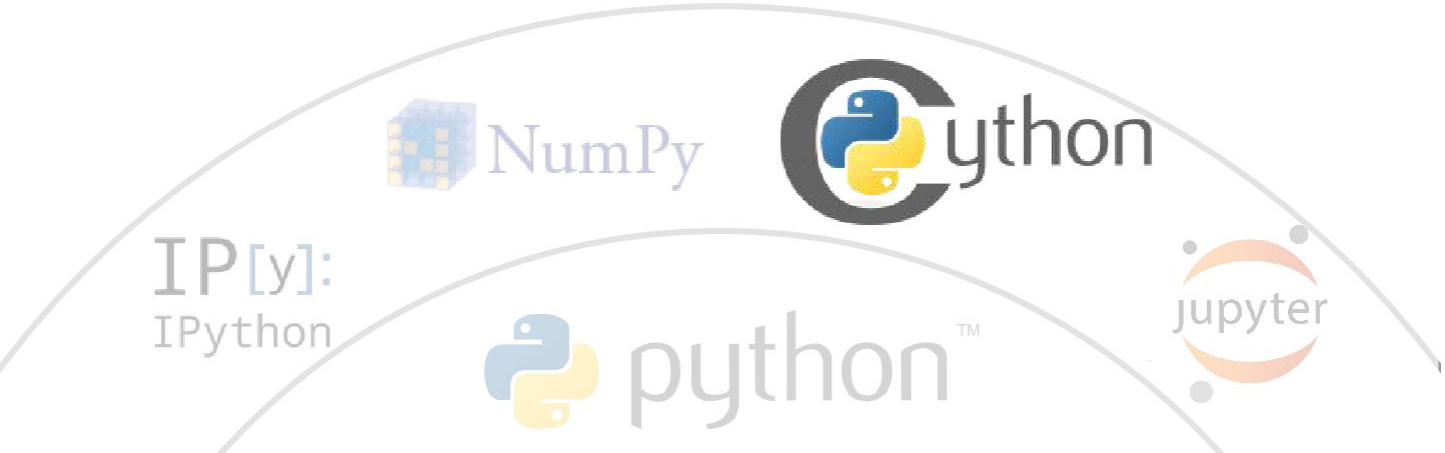

NumPy
ython
IP[y]:
IPython
python™
jupyter

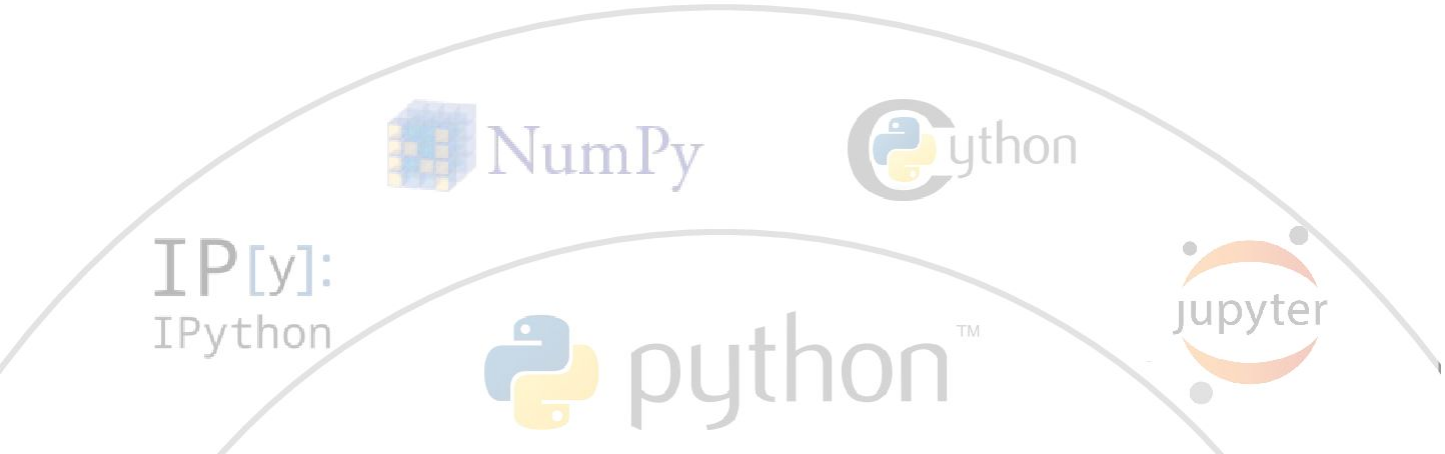
Cython = C + Python

{Propulsion}

Super-set of the Python language that allows easy interfacing with C & Fortran libraries (e.g. BLAS, LAPACK, etc.) and also fast Python code.

Drives many of the packages in the data science stack.

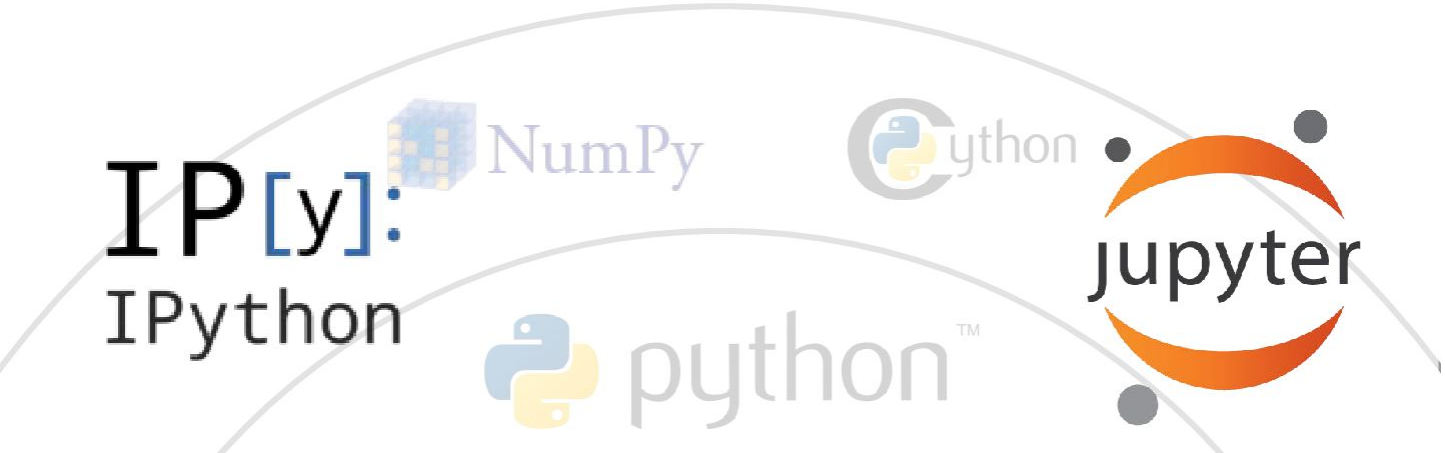



NumPy
ython
IP[y]:
IPython
python™
jupyter

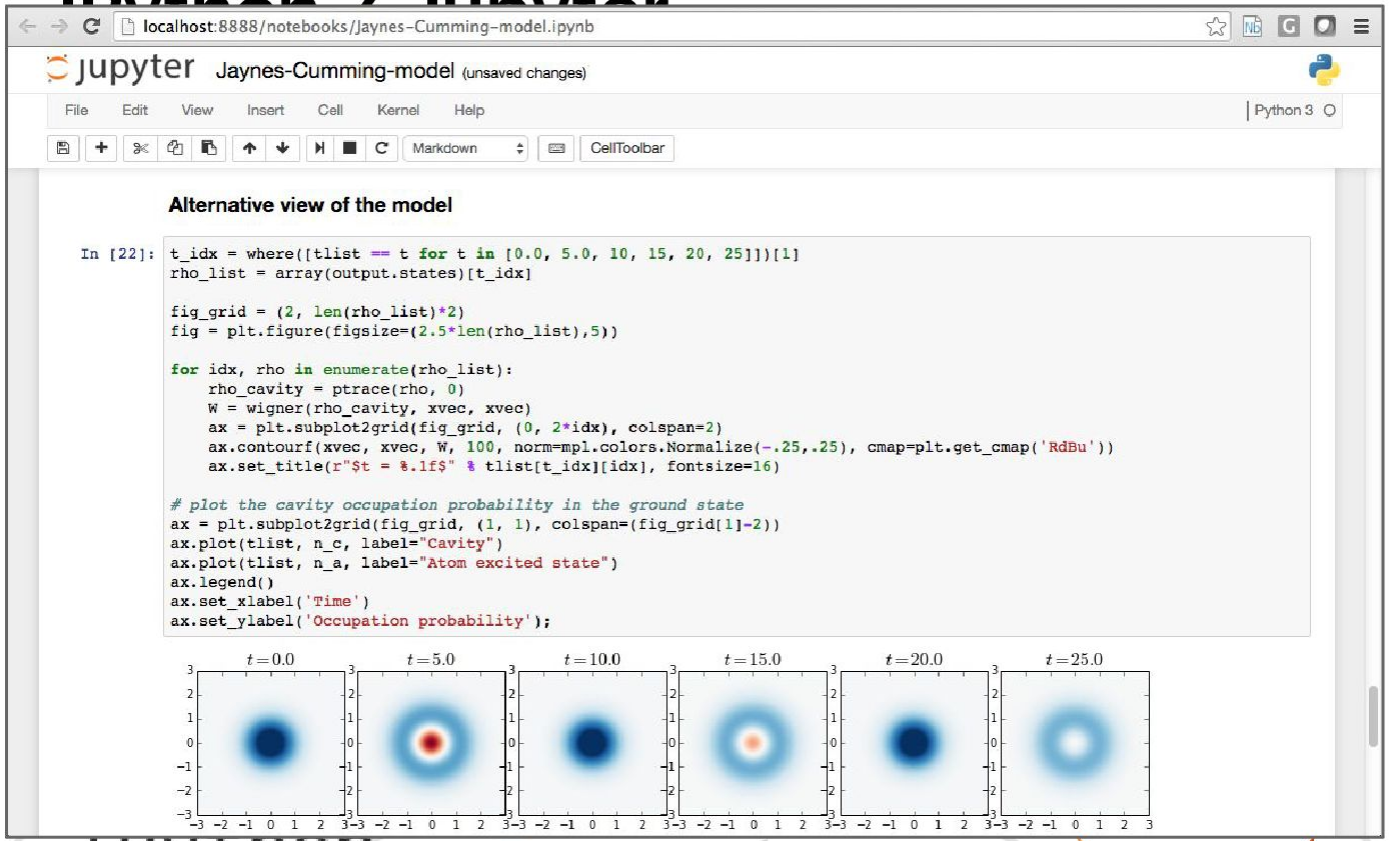
IPython / Jupyter

{Propulsion}

Terminal, development environment, Notebooks, and more for efficient use of Python in day-to-day work



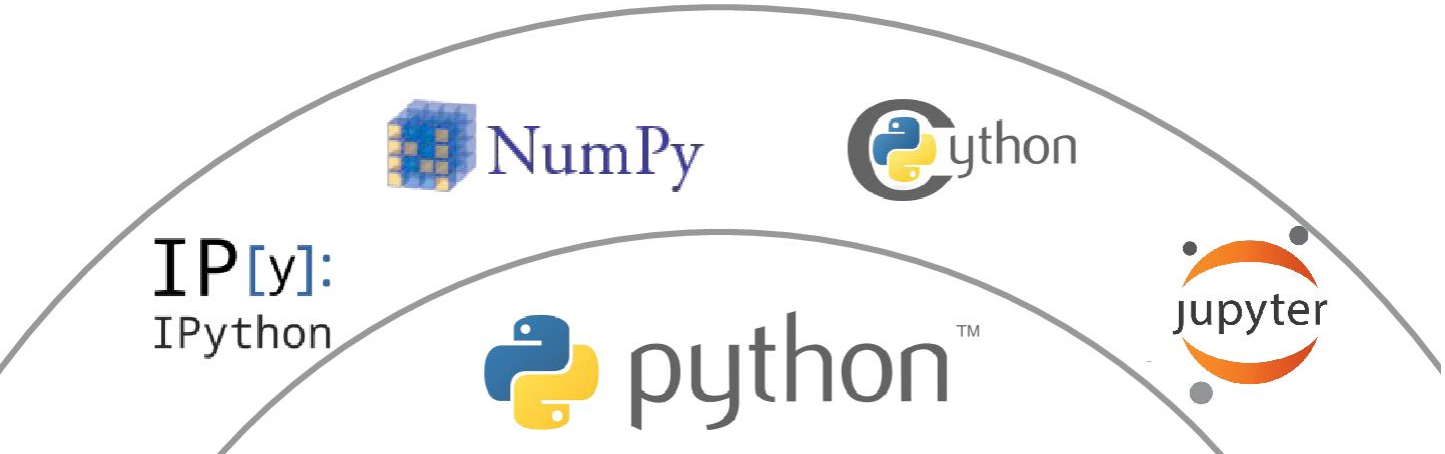
IPython / Jupyter

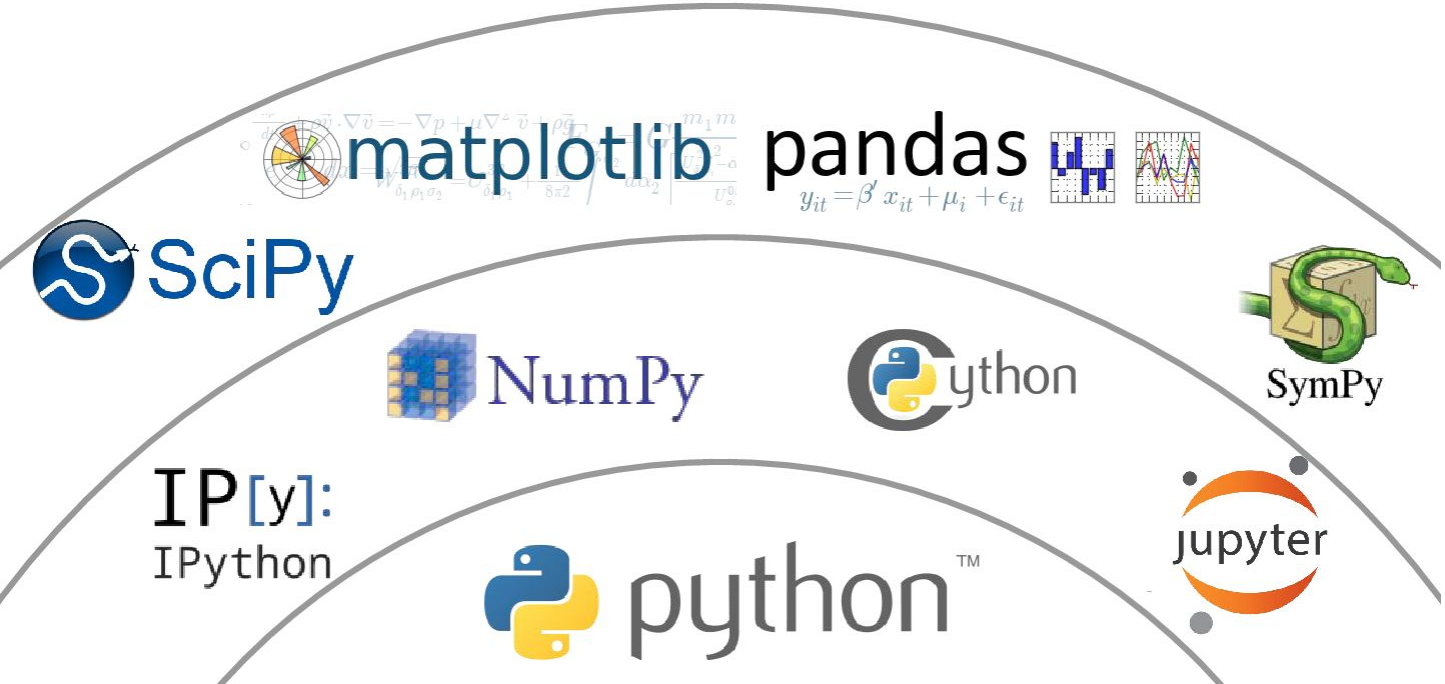


{Propulsion}

IPython

python™

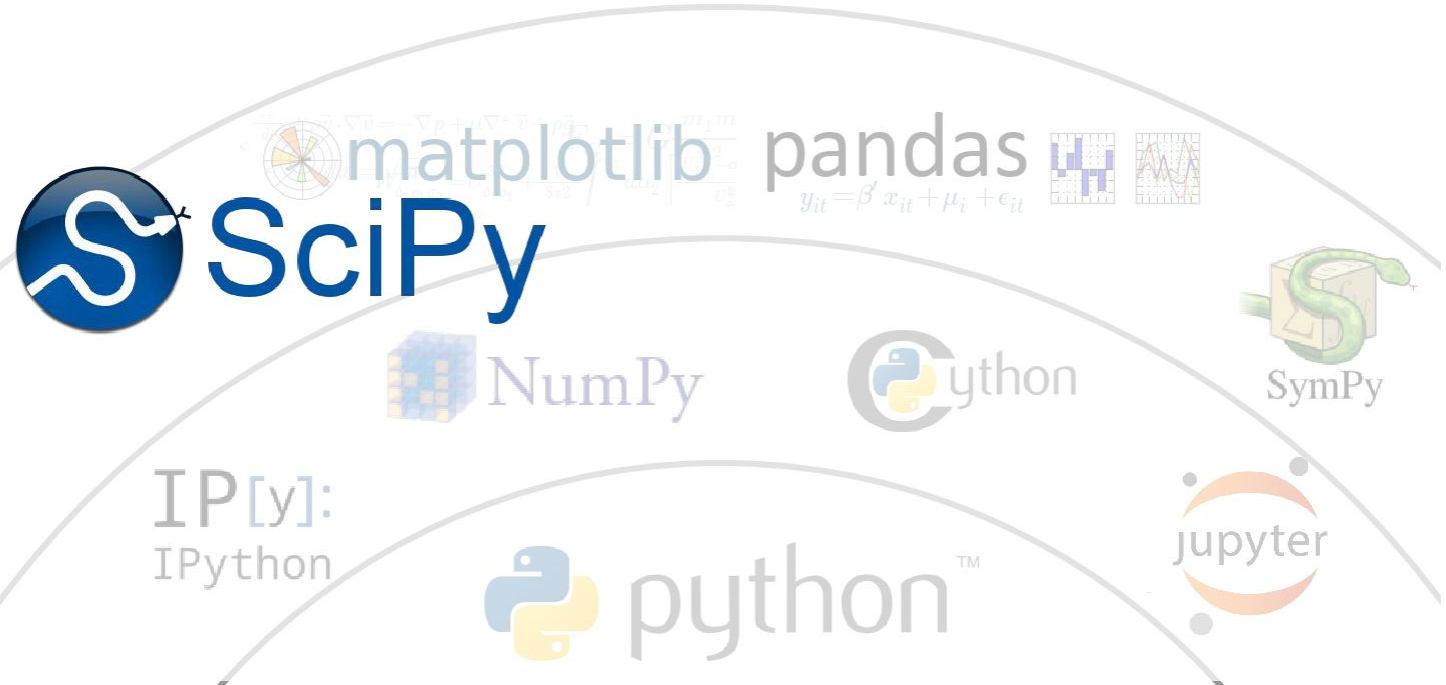

NumPy
ython
IP[y]:
IPython
python™
jupyter


matplotlib
pandas
$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$
SciPy
NumPy
ython
SymPy
IP[y]:
IPython
python™
jupyter

SciPy

{Propulsion}

Provides an interface to common scientific computing Tasks, including wrappers of many NetLib packages.



SciPy

Provi
Tasks

List from http://docs.scipy.org/doc/scipy/reference/

- Special functions (**scipy.special**)
- Integration (**scipy.integrate**)
- Optimization (**scipy.optimize**)
- Interpolation (**scipy.interpolate**)
- Fourier Transforms (**scipy.fftpack**)
- Signal Processing (**scipy.signal**)
- Linear Algebra (**scipy.linalg**)
- Sparse Eigenvalue Problems with ARPACK
- Compressed Sparse Graph Routines (**scipy.sparse.csgraph**)
- Spatial data structures and algorithms (**scipy.spatial**)
- Statistics (**scipy.stats**)
- Multidimensional image processing (**scipy.ndimage**)
- File IO (**scipy.io**)

{Propulsion}

IP[y]: IPython

python™

jupyter


matplotlib
pandas
SciPy
NumPy
Cython
SymPy
IP[y]:
IPython
python
jupyter

Sympy

{Propulsion}

Library for symbolic computation: algebraic operations, differentiation & integration, optimization, etc.

Sympy

{Propulsion}

Library for symbolic computation: algebraic operations, differentiation & integration, optimization, etc.

Polynomials and rational functions

SymPy does not expand brackets automatically. The function expand is used for this.

In [6]:
```
a=(x+y-z)**6
a
```

Out[6]: $(x + y - z)^6$

In [7]:
```
a=expand(a)
a
```

Out[7]: $$x^6 + 6x^5y - 6x^5z + 15x^4y^2 - 30x^4yz + 15x^4z^2 + 20x^3y^3 - 60x^3y^2z + 60x^3yz^2 - 20x^3z^3 + 15x^2y^4 - 60x^2y^3z + 90x^2y^2z^2 - 60x^2yz^3 + 15x^2z^4 + 6xy^5 - 30xy^4z + 60xy^3z^2 - 60xy^2z^3 + 30xyz^4 - 6xz^5 + y^6 - 6y^5z + 15y^4z^2 - 20y^3z^3 + 15y^2z^4 - 6yz^5 + z^6$$


matplotlib
pandas
SciPy
NumPy
Cython
SymPy
IP[y]:
IPython
python
jupyter

matplotlib

{Propulsion}

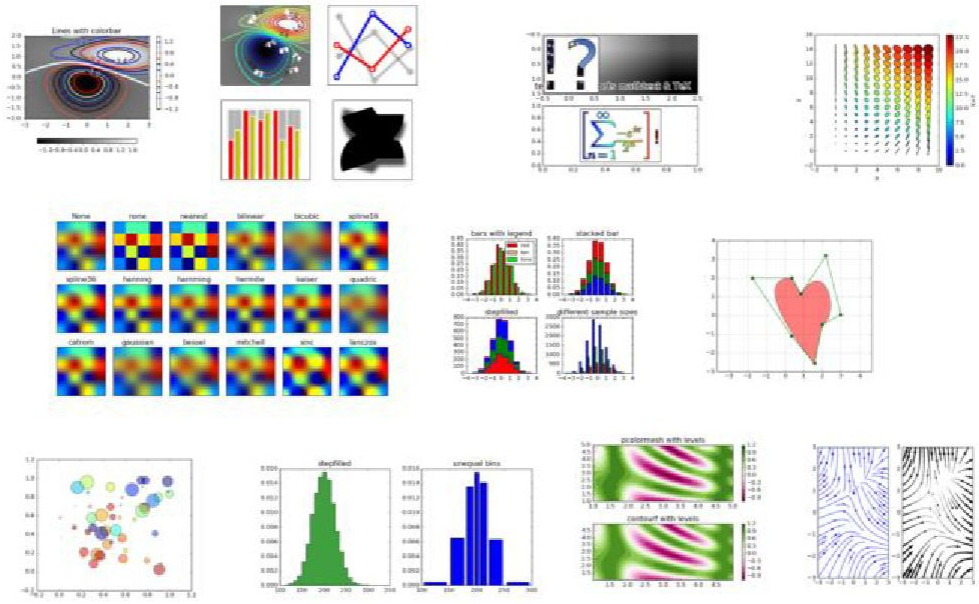
Matlab-inspired plotting and visualization

matplotlib

{Propulsion}

Matlab-inspired plotting and visualization

From http://matplotlib.org/gallery.html

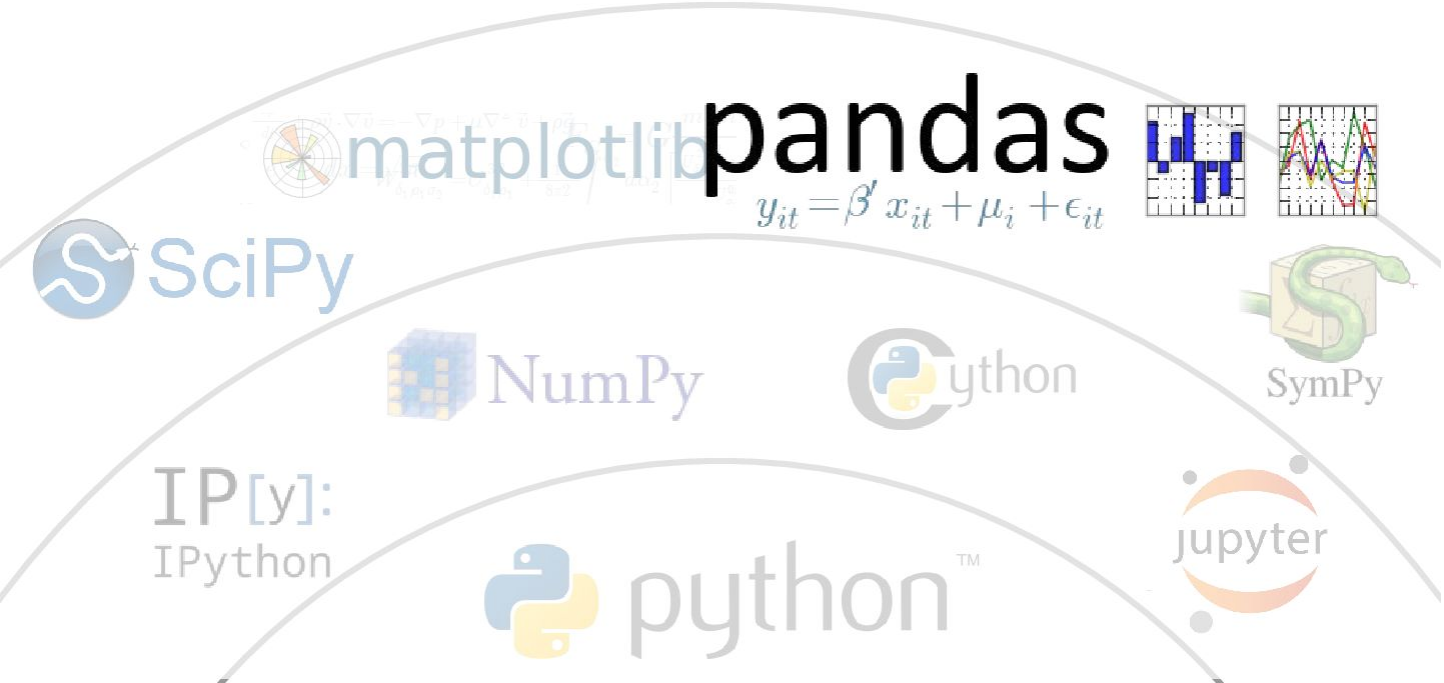



matplotlib
pandas
SciPy
NumPy
Cython
SymPy
IP[y]:
IPython
python
jupyter

Pandas

{Propulsion}

R-inspired DataFrames & associated functionality
(data munging & cleaning, group-by & transformations,
and much more)



Pa

{Propulsion}

In [1]:
```
import pandas as pd
data = pd.read_csv('iris.csv')
data.head()
```

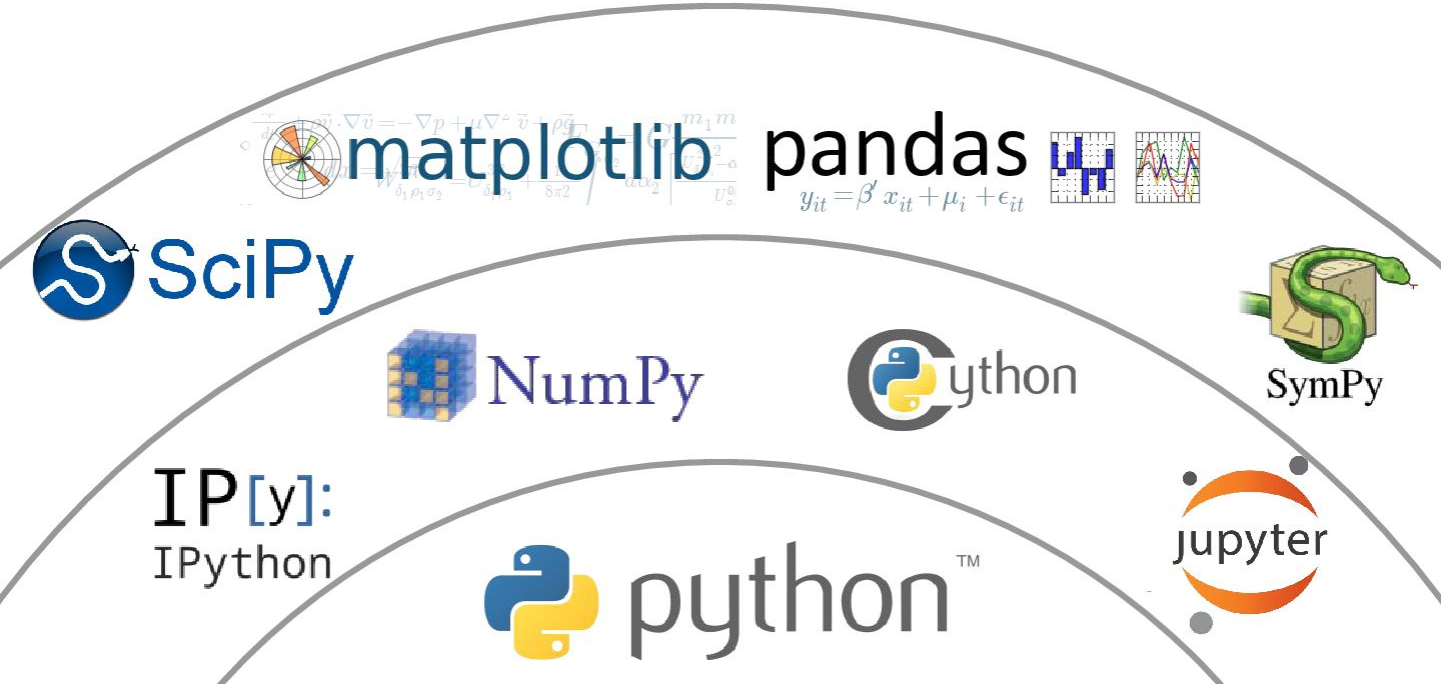
Out[1]:

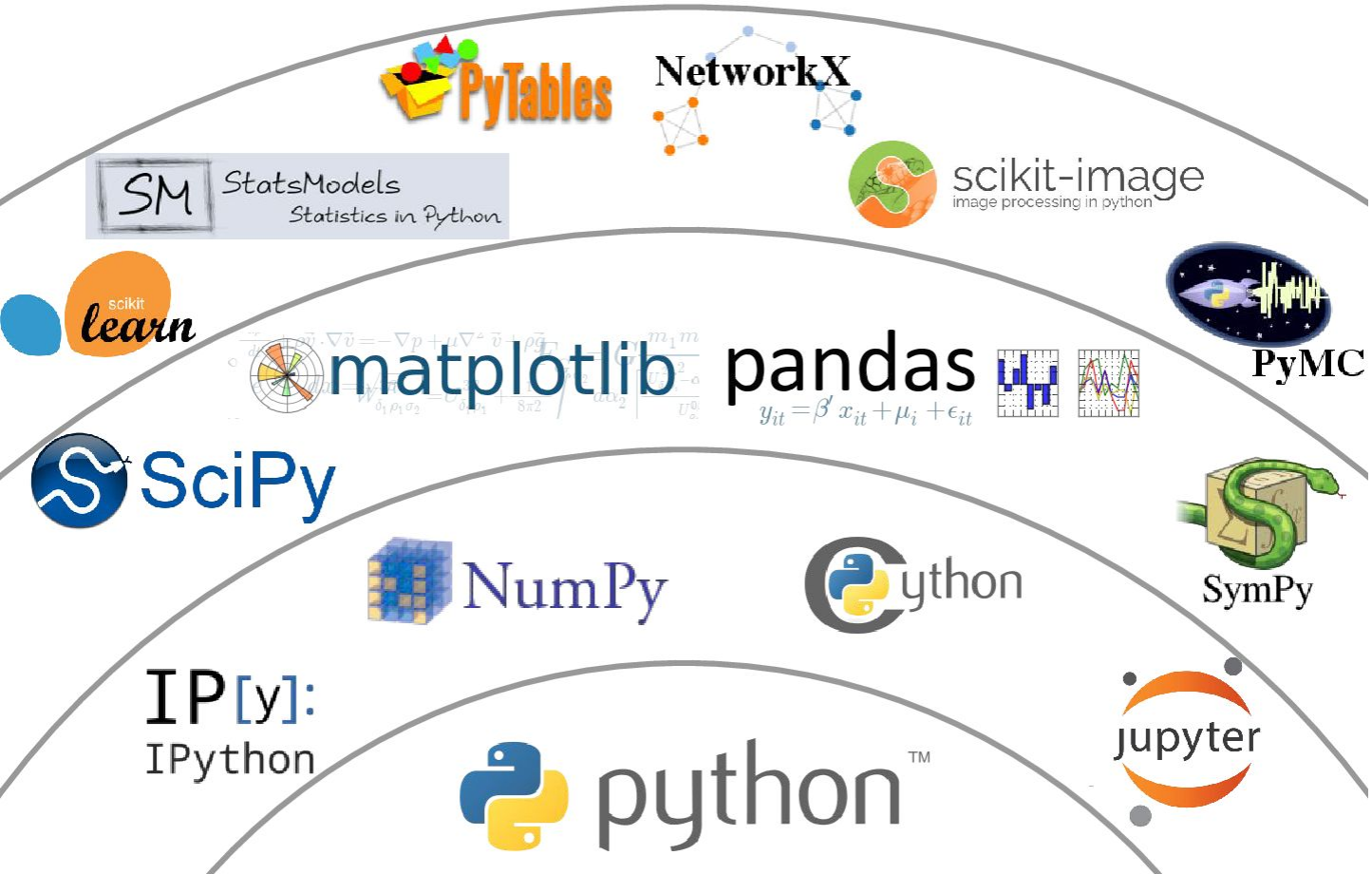
	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [2]:
```
data.groupby('Species').mean()
```

Out[2]:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
Species				
setosa	5.006	3.428	1.462	0.246
versicolor	5.936	2.770	4.260	1.326
virginica	6.588	2.974	5.552	2.026

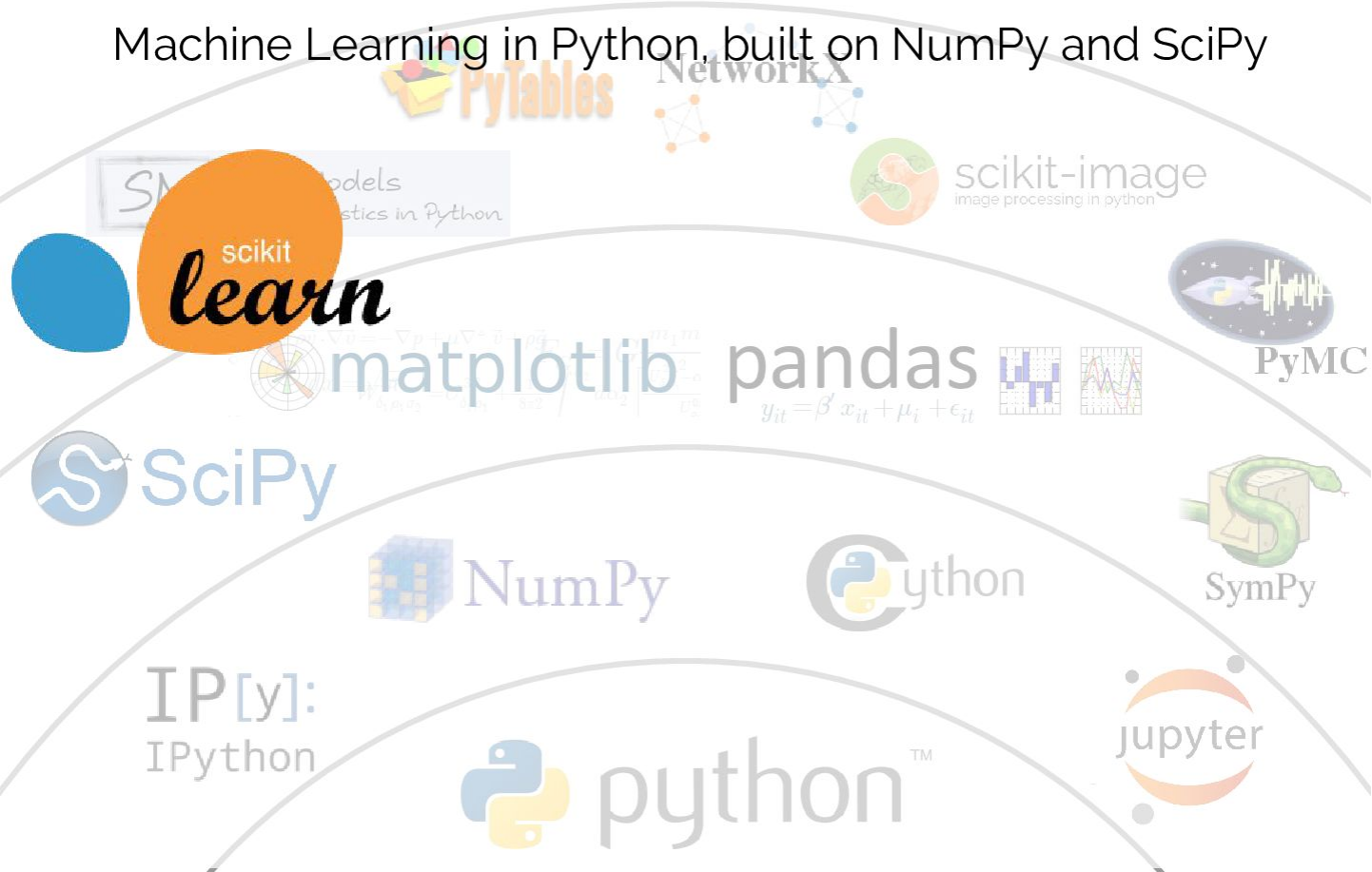

matplotlib
pandas
$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$
SciPy
NumPy
ython
SymPy
IP[y]:
IPython
python™
jupyter


PyTables
NetworkX
SM
StatsModels
Statistics in Python
scikit-image
image processing in python
scikit
learn
matplotlib
pandas
$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$
PyMC
SciPy
NumPy
ython
SymPy
IP[y]:
IPython
python™
jupyter

Scikit-Learn

{Propulsion}

Machine Learning in Python, built on NumPy and SciPy

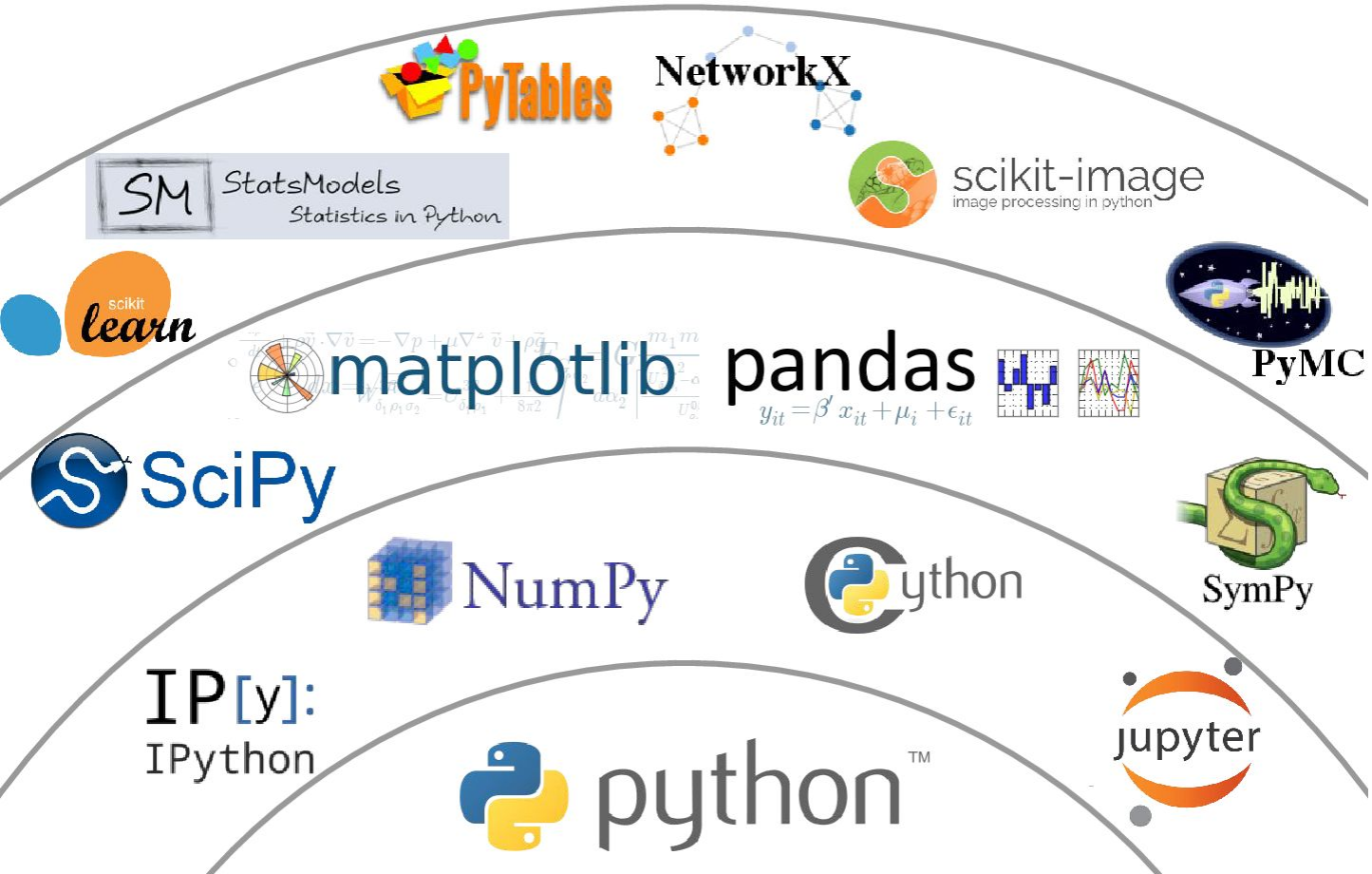


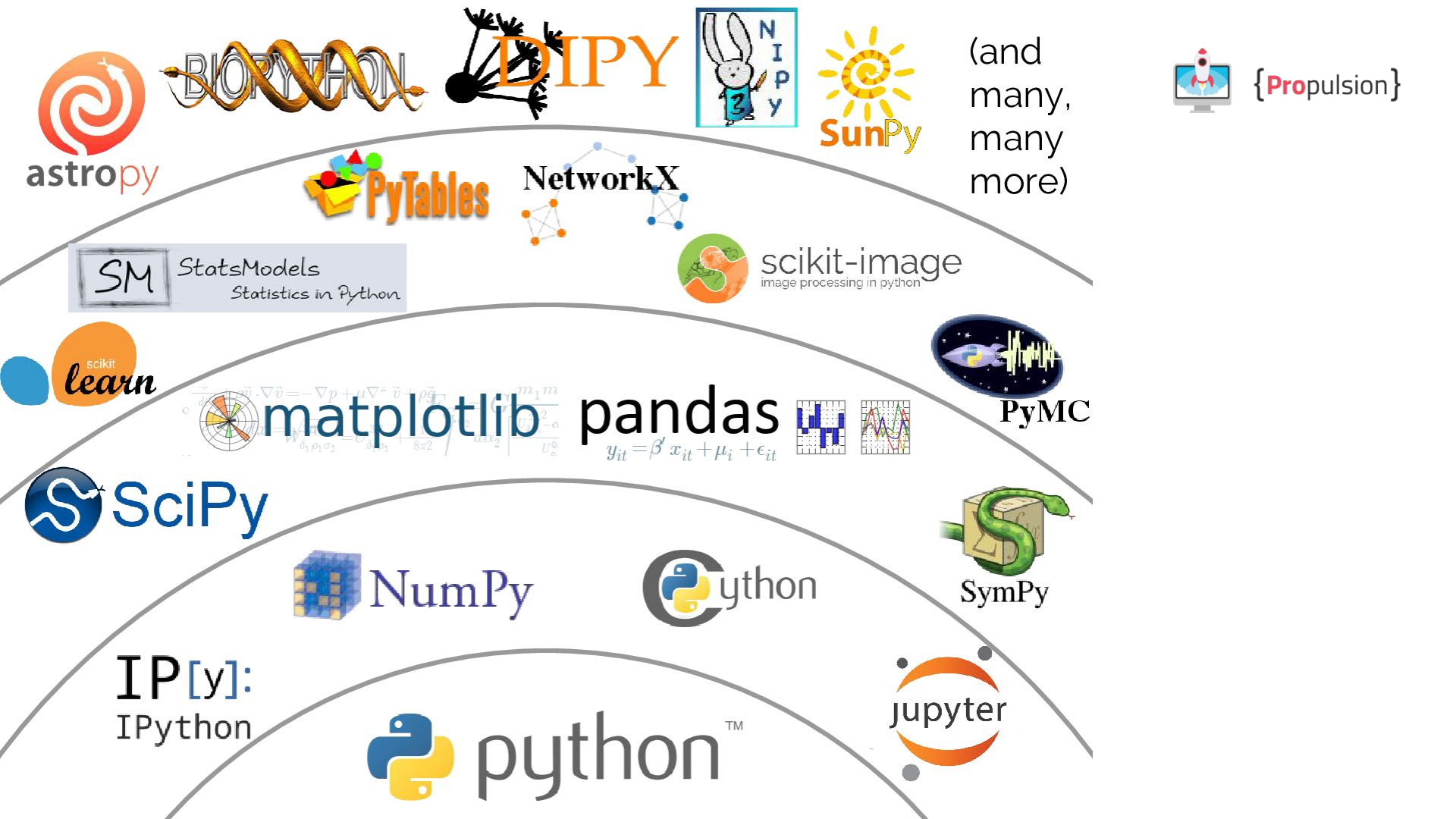
Scikit-Learn

{Propulsion}

Machine Learning in Python, built on NumPy and SciPy

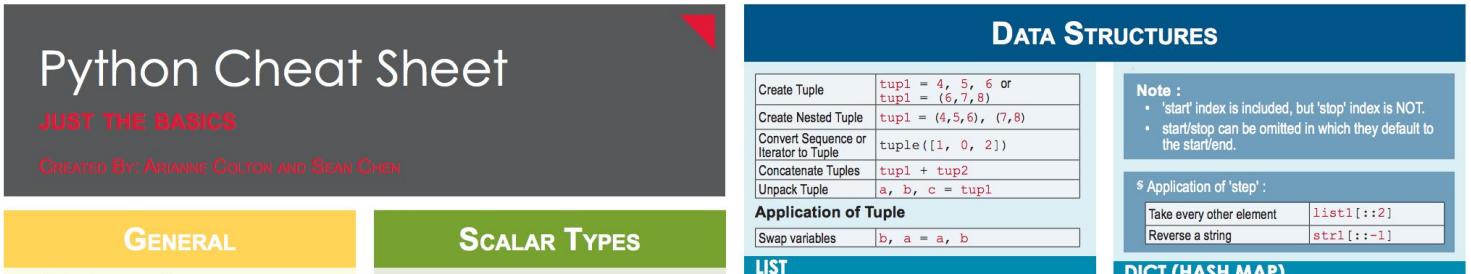
```
In [3]: from sklearn.ensemble import RandomForestClassifier

        features = data.drop('Species', axis=1)
        labels = data['Species']

        model = RandomForestClassifier()
        model.fit(features, labels)

        predicted = model.predict(features.iloc[:5])
        print(predicted)

        ['setosa' 'setosa' 'setosa' 'setosa' 'setosa']
```


PyTables
NetworkX
SM
StatsModels
Statistics in Python
scikit-image
image processing in python
scikit
learn
matplotlib
pandas
$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$
PyMC
SciPy
NumPy
ython
SymPy
IP[y]:
IPython
python™
jupyter


astropy
BIOPYTHON
DIPY
NIPY
SunPy
(and many, many more)
{Propulsion}
PyTables
NetworkX
SM StatsModels Statistics in Python
scikit-image image processing in python
scikit learn
matplotlib
pandas
$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$
PyMC
SciPy
NumPy
Cython
SymPy
IP[y]: IPython
python™
jupyter

Let's dive right into it

- Install Python, iPython, Jupyter, pip
- Single distribution with all important packages: docs.continuum.io/anaconda
- pip/virtualenv tutorial: tiny.cc/propacad-ds-pipvenv
- Jupyter tutorial: tiny.cc/propacad-ds-jupyter
- Python cheat sheet: datasciencefree.com/python.pdf
- Online exercises: tiny.cc/propacad-ds-python



Programming challenges

{Propulsion}

1. Print the difference between two UNIX Timestamps as human-readable time

```
Enter timestamps: 1493560132 1493565412
1 hour 28 minutes
```

2. For any string, print which words appear within which other words

```
Enter sentence: There is a palm tree on the island.
The word is appears in the word island
```

3. Print all words in a string that are anagrams of each other: “How can you *listen* if you are not *silent*?”
Bonus: how do we get all existing anagrams of a given word?

4. Write an email validating **function**, returning True or False depending on whether an email address is correctly formatted (e.g. nicolas@propulsion-academy.com works, not nicolas@test). No input should lead to an exception.

5. Write a password validation function: the input should contain at least one lowercase letter, at least one uppercase letter, at least one digit, and at least one symbol from [$#/\@*]. Unmet criteria or any other character should lead to an exception.

6. Write the following console-based number guessing game:
Bonus: reverse it so the computer guesses the number you’ve thought of.

```
Hello! What is your name?
Albert
Well, Albert, I am thinking of a number between 1 and 20.
Take a guess.
10
Your guess is too high.
Take a guess.
2
Your guess is too low.
Take a guess.
4
Good job, Albert! You guessed my number in 3 guesses!
```