

# مقدمه ای بر پایتون برای یادگیری ماشین

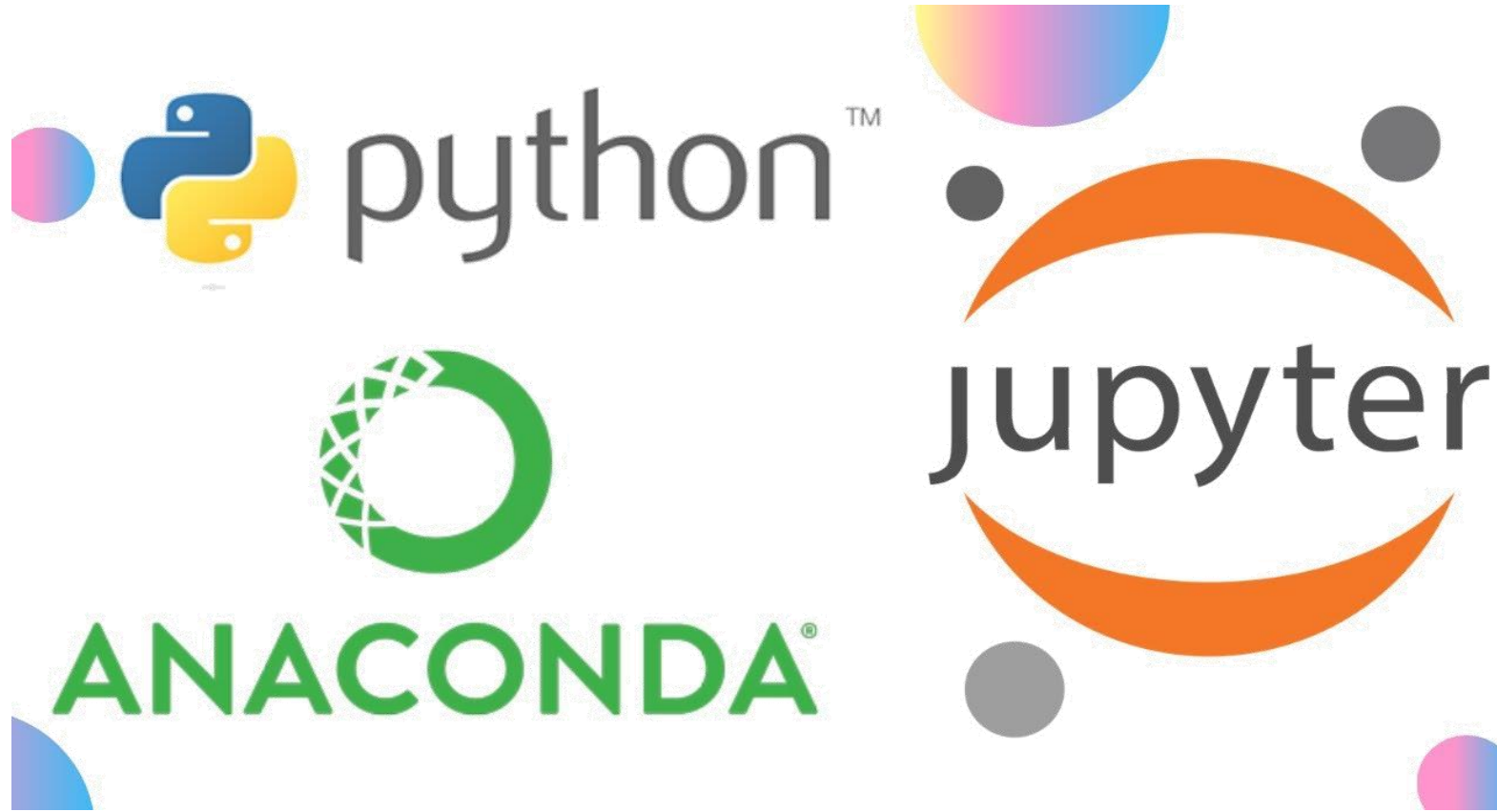
دکتر مهدی شریفزاده  
امیرحسین محمودی

بهمن ۱۴۰۱

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

# Python; Anaconda; Jupyter notebook

---



# Python; Anaconda; Jupyter notebook



ANACONDA.NAVIGATOR

Connect

Home

Environments

Learning


Community

All applications

on

base (root)


Channels



DataSpell

DataSpell is an IDE for exploratory data analysis and prototyping machine learning models. It combines the interactivity of Jupyter notebooks with the intelligent Python and R coding assistance of PyCharm in one user-friendly environment.


Install



Datalore

Online Data Analysis Tool with smart coding assistance by JetBrains. Edit and run your Python notebooks in the cloud and share them with your team.


Launch



IBM Watson Studio Cloud

IBM Watson Studio Cloud provides you the tools to analyze and visualize data, to cleanse and shape data, to create and train machine learning models. Prepare data and build models, using open source data science tools or visual modeling.

Launch




JupyterLab

3.3.2

An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Launch




Jupyter Notebook

6.4.8

Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Launch




PyCharm Community

2022.1.1

An IDE by JetBrains for pure Python development. Supports code completion, listing, and debugging.

Launch




Qt Console

5.3.0

PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

Launch




Spyder

5.1.5

Scientific PYTHON Development Environment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

Launch




VS Code

1.68.1

Streamlined code editor with support for development operations like debugging, task running and version control.

Launch




Glueviz

1.0.0

Multidimensional data visualization across files. Explore relationships within and among related datasets.

Install




Orange 3

3.26.0

Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox.


Install



PyCharm Professional

A full-fledged IDE by JetBrains for both Scientific and Web Python development. Supports HTML, JS, and SQL.

Install



RStudio

1.1.456

A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.

ANACONDA

Secure your software supply chain from the source

Upgrade Now

End-to-end package security, guaranteed

Documentation

Anaconda Blog

YouTube



# Python; Anaconda; Jupyter notebook



**jupyter** Welcome to P

File Edit View Insert Cell Kernel Help Python 3

**Exploring the Lorenz System**

In this Notebook we explore the [Lorenz system](#) of differential equations:

$$\begin{aligned}\dot{x} &= \sigma(y - x) \\ \dot{y} &= \rho x - y - xz \\ \dot{z} &= -\beta z + xy\end{aligned}$$

This is one of the classic systems in non-linear differential equations. It exhibits a range of complex behaviors as the parameters ( $\sigma$ ,  $\beta$ ,  $\rho$ ) are varied, including what are known as *chaotic solutions*. The system was originally developed as a simplified mathematical model for atmospheric convection in 1963.

In [7]: `interact(Lorenz, N=fixed(10), angle=(0.,360.),  
sigma=(0.0,50.0), beta=(0.,5), rho=(0.0,50.0))`

angle 308.2  
max\_time 12  
 $\sigma$  10  
 $\beta$  2.6  
 $\rho$  28



# Setting Up a Python Environment

---



- ✓ Set Up Anaconda Python Environment (also jupyter notebook)

<https://www.anaconda.com/products/distribution>

<https://code.visualstudio.com/download>

Watch video:

<https://drive.google.com/file/d/13J9-1tjYpLJ9jy0U5kGq7NSmDT0DOdE/view>



# Google Colab



✓ <https://colab.research.google.com/>

# colab

**Overview of Collaboratory Features**  
File Edit View Insert Runtime Tools Help

**Table of contents**

- Cells
  - Code cells
  - Text cells
- Adding and moving cells
- Working with python
  - System aliases
  - Magics
  - Automatic completions and exploring code
  - Exception Formatting
  - Rich, interactive outputs
- Integration with Drive
- Commenting on a cell
- Section

**Cells**  
A notebook is a list of cells. Cells contain either explanatory text or executable code and its output. Click a cell to select it.

**Code cells**  
Below is a **code cell**. Once the toolbar button indicates CONNECTED, click in the cell to select it and execute the contents in the following ways:

- Click the **Play icon** in the left gutter of the cell;
- Type **Cmd/Ctrl+Enter** to run the cell in place;
- Type **Shift+Enter** to run the cell and move focus to the next cell (adding one if none exists); or
- Type **Alt+Enter** to run the cell and insert a new code cell immediately below it.

There are additional options for running some or all cells in the **Runtime** menu.

```
[ ] a = 10  
a  
  
10
```

**Text cells**  
This is a **text cell**. You can **double-click** to edit this cell. Text cells use markdown syntax. To learn more, see our [markdown guide](#).  
You can also add math to text cells using [LaTeX](#) to be rendered by [MathJax](#). Just place the statement within a pair of \$ signs. For example



# Python basic functions

---



- ✓ type( ) function
- ✓ input( ) function
- ✓ abs( ) function
- ✓ pow( ) function
- ✓ dir( ) function
- ✓ sorted( ) function
- ✓ max( ) function
- ✓ round( ) function
- ✓ divmod( ) function
- ✓ id( ) function
- ✓ ord( ) function
- ✓ len( ) function
- ✓ sum( ) function
- ✓ help( ) function
- ✓ Str() function
- ✓ Int() function





# Lines and Indentation

---



Python provides no braces to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced.

```
if True:
    print "True"
else:
    print "False"
```



# Multi-Line Statements



Statements in Python typically end with a new line. Python does, however, allow the use of the line continuation character (\) to denote that the line should continue

```
total = item_one + \  
        item_two + \  
        item_three
```

Statements contained within the [], {}, or () brackets do not need to use the line continuation character. For example

```
days = ['Monday', 'Tuesday', 'Wednesday',  
        'Thursday', 'Friday']
```



# Quotation in Python



Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.

The triple quotes are used to span the string across multiple lines.

```
word = 'word'  
sentence = "This is a sentence."  
paragraph = """This is a paragraph. It is  
made up of multiple lines and sentences."""
```



# Comments in Python



A hash sign (#) that is not inside a string literal begins a comment. All characters after the # and up to the end of the physical line are part of the comment and the Python interpreter ignores them.

```
# First comment  
print "Hello, Python!" # second comment
```

# Waiting for the User



A hash sign (#) that is not inside a string literal begins a comment. All characters after the # and up to the end of the physical line are part of the comment and the Python interpreter ignores them.

```
raw_input("\n\nPress the enter key to exit.")
```



# Multiple Statements on a Single Line



The semicolon ( ; ) allows multiple statements on the single line given that neither statement starts a new code block. Here is a sample snip using the semicolon

```
import sys; x = 'foo'; sys.stdout.write(x + '\n')
```

## Multiple Assignment

```
a,b,c = 1,2,"john"
```





# Python reserved keywords



and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield



# Basic Arithmetic Operations



In [2]:

```
a = 12  
b = 2  
  
print(a + b)  
print(a**b)  
print(a/b)
```

14

144

6.0

# Python Libraries



In [3]:

```
import numpy as np

# mathematical constants
print(np.pi)
print(np.e)

# trigonometric functions
angle = np.pi/4
print(np.sin(angle))
print(np.cos(angle))
print(np.tan(angle))
```

```
3.141592653589793
2.718281828459045
0.707106781187
0.707106781187
1.0
```



# Python Lists



A list is a data structure in Python that is a **mutable, or changeable, ordered sequence of elements**. Each element or value that is inside of a list is called an item. Just as strings are defined as characters between quotes, lists are defined by having values between square brackets [ ].

In [4]:

```
xList = [1, 2, 3, 4]  
xList
```

Out[4]: [1, 2, 3, 4]

In [5]:

```
# Concatenation  
x = [1, 2, 3, 4];  
y = [5, 6, 7, 8];  
  
x + y
```

Out[5]: [1, 2, 3, 4, 5, 6, 7, 8]



# Python Lists



Sum a list of numbers:

```
In [6]: np.sum(x)
```

```
Out[6]: 10
```

An element-by-element operation between two lists may be performed with

```
In [7]: print(np.add(x,y))  
        print(np.dot(x,y))
```

```
[ 6  8 10 12]
```

```
70
```



# Python Tuples



Tuples can be thought of as read-only lists

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )
tinytuple = (123, 'john')

print tuple           # Prints the complete tuple
print tuple[0]        # Prints first element of the tuple
print tuple[1:3]       # Prints elements of the tuple starting from 2nd till 3rd
print tuple[2:]        # Prints elements of the tuple starting from 3rd element
print tinytuple * 2    # Prints the contents of the tuple twice
print tuple + tinytuple # Prints concatenated tuples
```

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tuple[2] = 1000      # Invalid syntax with tuple
list[2] = 1000       # Valid syntax with list
```





# Python Dictionary



Dictionaries are used to store data values in key:value pairs.

A dictionary is a collection which is *ordered\**, *changeable* and *do not allow duplicates*.

Dictionaries are written with curly brackets, and have keys and values:

```
In [9]: mw = {'CH4': 16.04, 'H2O': 18.02, 'O2': 32.00, 'CO2': 44.01}
mw
```

```
Out[9]: {'CH4': 16.04, 'CO2': 44.01, 'H2O': 18.02, 'O2': 32.0}
```

Add or change a value:

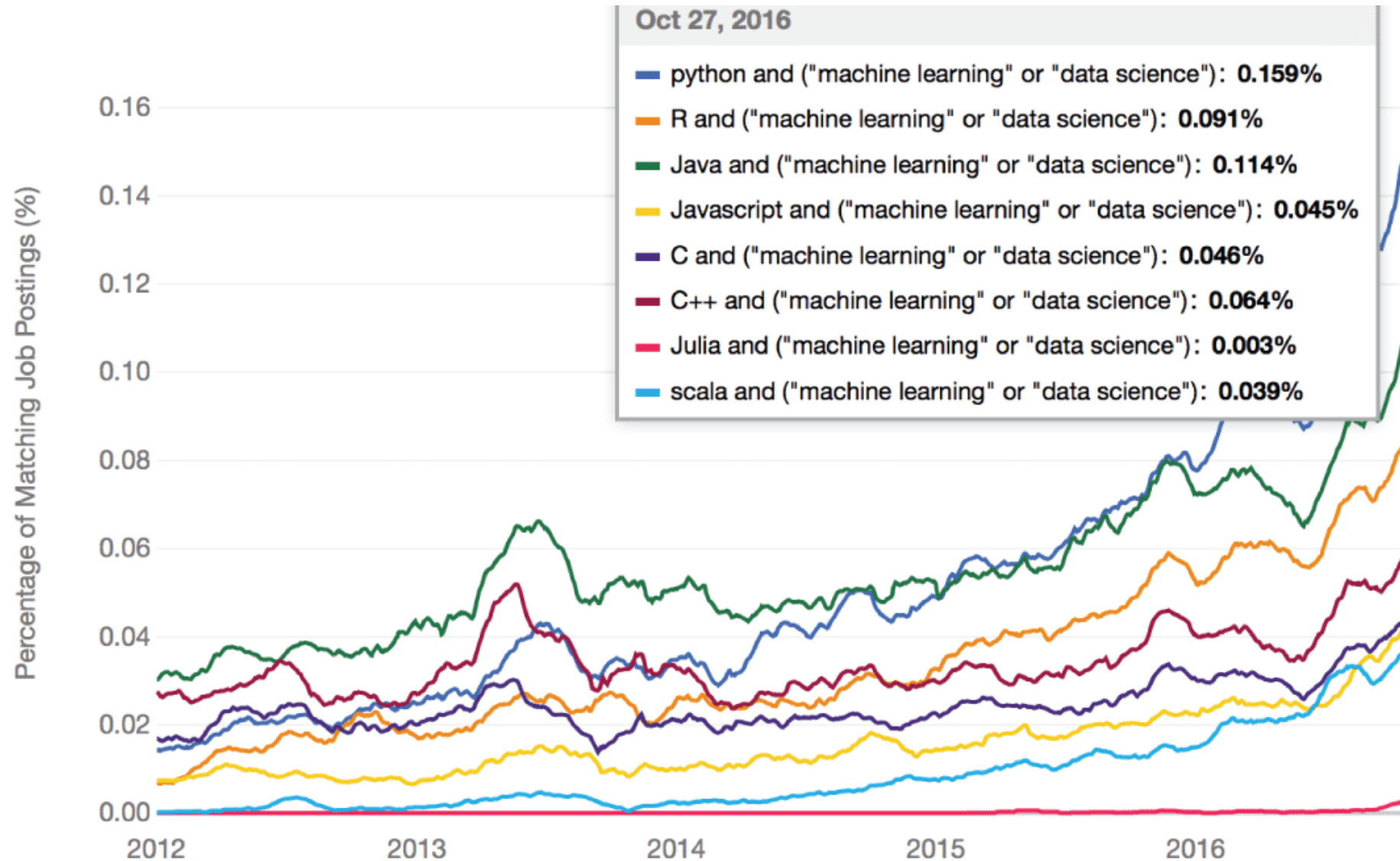
```
In [10]: mw['C8H18'] = 114.23
mw
```

```
Out[10]: {'C8H18': 114.23, 'CH4': 16.04, 'CO2': 44.01, 'H2O': 18.02, 'O2': 32.0}
```

As of Python version 3.7, dictionaries are *ordered*. In Python 3.6 and earlier, dictionaries are *unordered*.



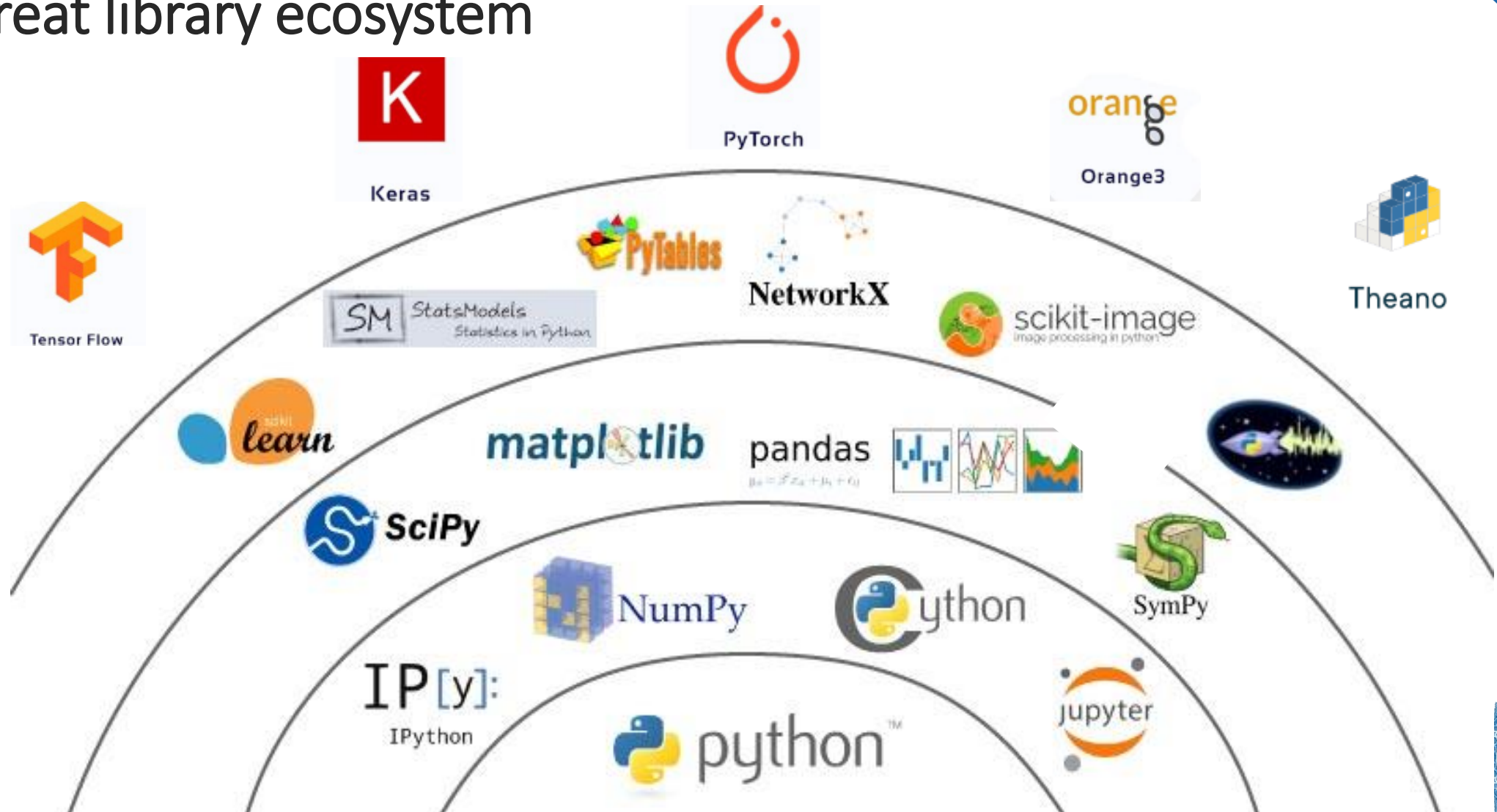
# Why Python for Machine Learning?



# Why Python for Machine Learning?



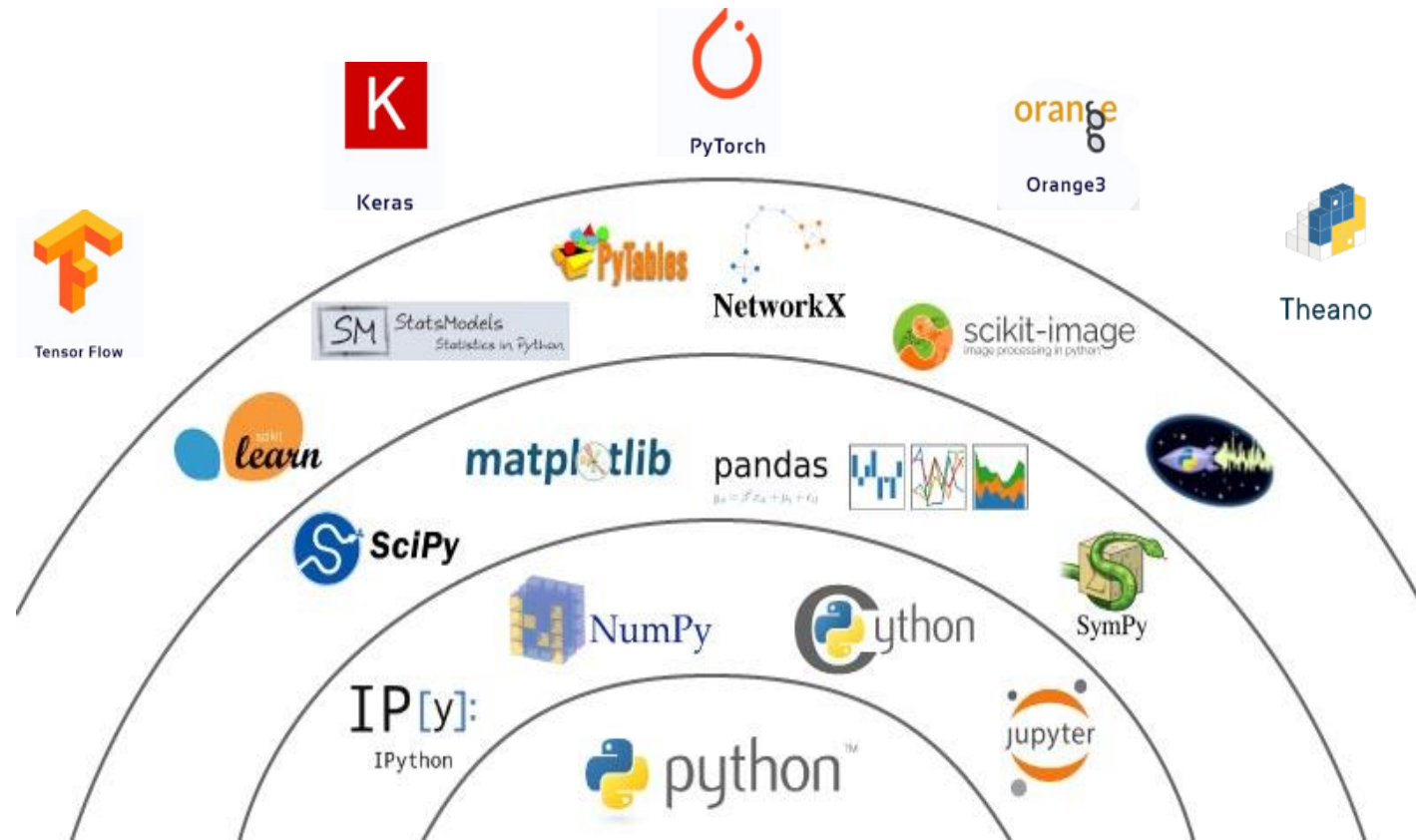
- A great library ecosystem



# Why Python for Machine Learning?



- A great library ecosystem
- A low entry barrier
- Flexibility
- Platform independence
- Good visualization options
- Community support



# References

- Dipanjan Sarkar, Raghav Bali, Tushar Sharma, Practical Machine Learning with Python- A Problem-Solver's Guide to Building Real-World Intelligent Systems, Apress, 2018.

