# Scientific Programming in Python

Inteligencia Artificial en los Sistemas de Control Autónomo
Máster Universitario en Ingeniería Industrial

Departamento de Automática

Universidad de Alcalá
ISG

## Objectives

1. Introduce some Python tools for scientific programming.
2. Motivate the need of efficient matrix manipulation.
3. Handle matrices and dataframes in Python.
4. Basic data visualization with Python.

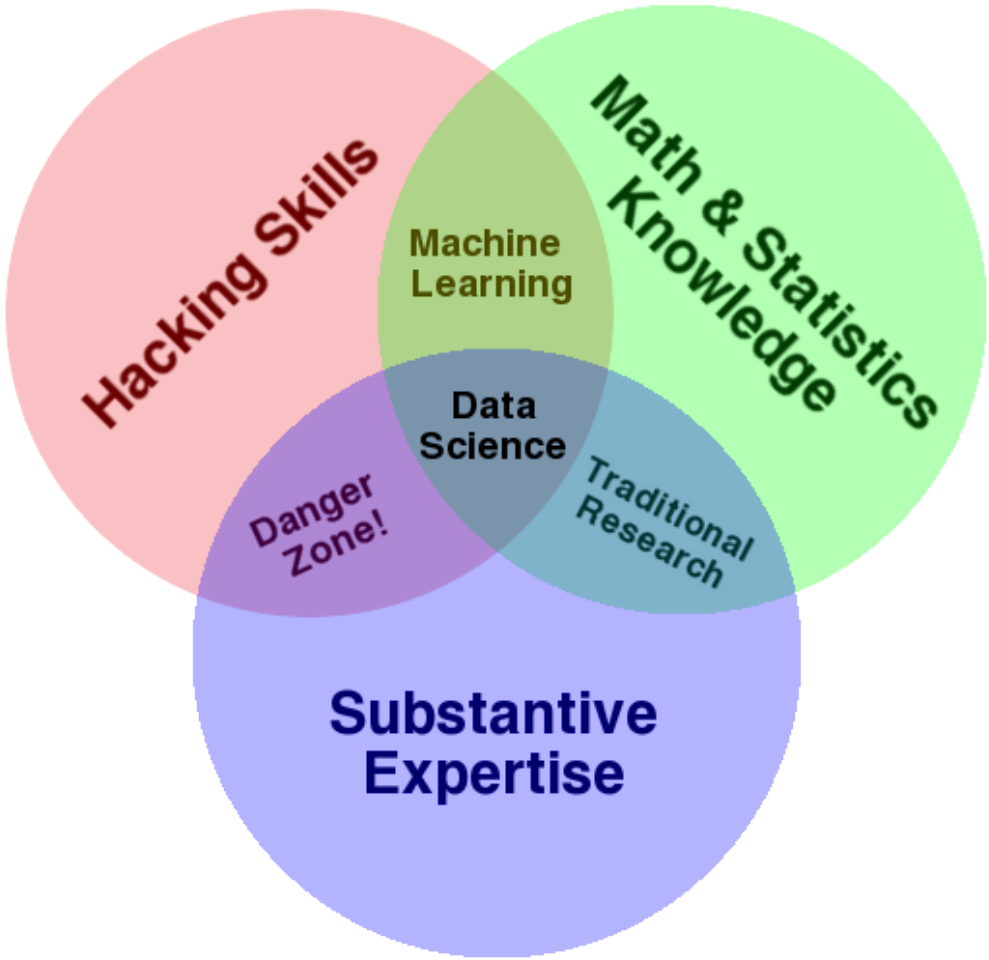
## Bibliography

Jake VanderPlas. *Python Data Science Handbook*. Chapters 1, 2, 3 and 4. O'Reilly. (Link).

# Table of Contents

# Overview

## Data Science



# Data Science

## The data scientist tookit (I)

Data science is about manipulating data

- Need of specialized tools
- Two main languajes: R and Python

Python is a general purpose programming language

- Easy integration
- Huge ecosystem of packages and tools

Need of data-oriented tools

- Features provided by third-party tools

# Data Science

## The data scientist tookit (II)

| Tool | Type | Description |
|---|---|---|
| `iPython` | Software | Advaced Python interpreter |
| `Jupiter` | Software | Python notebooks (Python interpreter) |
| `Numpy` | Package | Efficient array operations |
| `Pandas` | Package | Dataframe support |
| `Matplotlib` | Package | Data visualization |
| `Seaborn` | Package | Data visualization with dataframes |
| `Scikit-learn` | Package | AI/ML package for Python |

# Data Science

## Anaconda

All those tools are packaged in `Anaconda`

- Python distribution for Data Science

Anaconda provides `Spyder`

- Python IDE designed for Data Science

Other tools provided by Anaconda

- Conda: Packages management tool
- TensorFlow: Deep Learning
- Many others


# Data Science

## Python IDEs for Data Science (I)

### iPython

iPython = Interactive Python

- Extended funcionality
- Enhanced UI
- External editor

Running iPython:

```
$ ipython
```

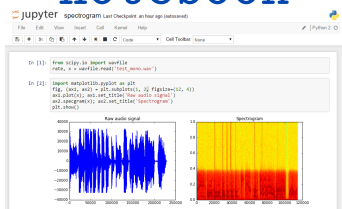
### Jupyter

Python notebooks

- Web-based IDE
- Documentation
- Integration with GitHub
- Uses iPython

Running Jupyter:

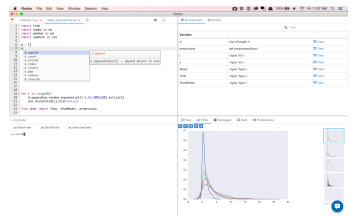
```
$ jupyter notebook
```



### Rodeo

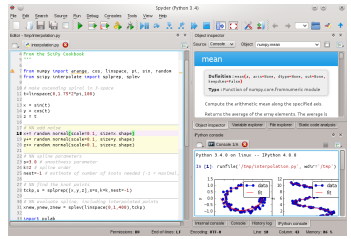
Python version of RStudio

- Good for R developers
- Not included in Anaconda
- Uses iPython



### Spyder

Matlab-like IDE



# Data Science

## Python IDEs for Data Science (II)

### Exercises

Write a Python script that shows the multiplication table of the number 5. Write the script using each one of the following environments:

1. iPython + text editor of your choice.
2. Jupiter.
   - Bonus track: Publish the notebook in GitHub.
3. Spyder.
4. Rodeo.

# iPython

## Basics (I)

In regular Python ...

- most objects come with a docstring attribute
- docstring accesible thorugh `help()`

iPython provides `?', a shortcut to `help()`

- `len?`, `list?`, `list.append?`
- Try to type just `?'

Easy access to source code with `??'

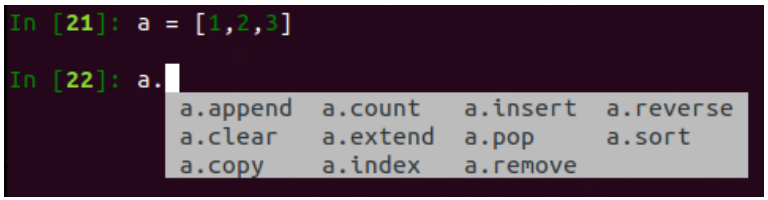
- Does not work with most buildin functions!

# iPython

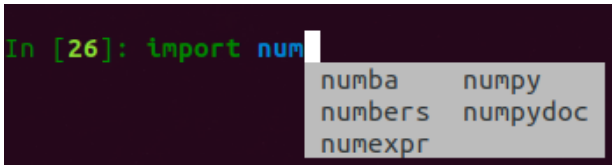
## Basics (II)

Press `<tab>` to complete almost everything

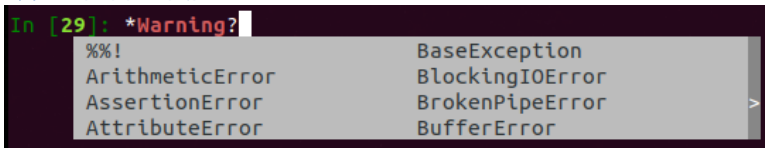
- Object contents




- Packages




- Wildcards



# iPython

## Basics (III): Keyboard shortcuts

Navigation

| Keystroke | Action |
|---|---|
| `Ctrl-a` | Move cursor to the beginning of the line |
| `Ctrl-e` | Move cursor to the end of the line |
| `Ctrl-b` | Move cursor back one character |
| `Ctrl-f` | Move cursor forward one character |

History

| Keystroke | Action |
|---|---|
| `Ctrl-p` (↑) | Previous command |
| `Ctrl-n` (↓) | Next command |
| `Ctrl-r` | Reverse-search |

Text entry

| Keystroke | Action |
|---|---|
| `Ctrl-d` | Delete next character in line |
| `Ctrl-k` | Cut text from cursor to end of line |
| `Ctrl-u` | Cut text from beginning of line to cursor |
| `Ctrl-y` | Yank (paste) previously cut text |

# iPython

## iPython magic commands

Magic commands: iPython extension of Python syntax

- Not valid in regular Python
- Provides handly features
- Widely used in DS and ML

Two flavours

- % prefix: Line magics - single line
- % % prefix: Cell magics - several lines

Help available

- `%magic`: Magic commands
- `%lsmagic`: List of magic commands

# iPython

## Pasting code blocks: %paste and %cpaste

Pasting code in Python is troublesome

- %paste: Paste one time
- %%cpaste: Paste several times

```
def donothing(x):
    return x
```

### %paste

```
In [20]: %paste
   def donothing(x):
    return x

## -- End pasted text --
```

### %cpaste

```
In [25]: %cpaste
Pasting code; enter '--' alone on the line
to stop or use Ctrl-D.
:       def donothing(x):
            return x:
:--
```

# iPython

## Running external code: %run and %timeit

%run: Execute script

- Many optional arguments
- Checkout %run?

```
In [40]: %run donothing.py

In [41]: donothing(10)
Out[41]: 10
```

%timeit: Computes execution time

- Executes a single line
- Automatic adjustment of runs
- Shows basic statistics

```
In [33]: %timeit [n ** 2 for n in range(200)]
71.6 µs ± 1.84 µs per loop
(mean ± std. dev. of 7 runs, 10000 loops each)

In [34]: %timeit [n ** 2 for n in range(2000)]
753 µs ± 16.2 µs per loop
(mean ± std. dev. of 7 runs, 1000 loops each)
```

%%timeit: Several lines

# iPython

## Input and output history (I)

iPython stores its history as objects

- `In`: Input commands
  - List storing commands
- `Out`: Commands output
  - Dictionary storing outputs
  - Not all commands have outputs

```
In [1]: import math
In [2]: math.sin(2)
Out[2]: 0.9092974268256817
In [3]: math.cos(2)
Out[3]: -0.4161468365471424
In [4]: Out[2] ** 2 + Out[3] ** 2
Out[4]: 1.0
```

# iPython

## Input and output history (II)

Fast access to history: Underscore (_)

- Variable containing the last output
- Example: `print(_)`

Double and triple underscores

- Example: `print(__)`
- Example: `print(___)`

Trick: Shortcut to access (_n)

- Out[n] = _n, with n=number
- Example: `print(_2)`

Magic command to show history

- `%history`

Supressing command output (;)

- Example: `4 * 2;`

# iPython

## iPython shell commands

iPython provides easy interaction with the shell

- Execution of shell commands from iPython
- Use prefix `!'
- Example: `!ls`, `!pwd`

Save shell output in Python variables

- Example: `files = !ls`

Use Python variables in shell

- Example: `!echo {files}`

# iPython

## Automagic

Problems with some shell commands

```
In [23]: !pwd
/repositorios/pythonCourse
In [24]: !cd ..
In [25]: !pwd
/repositorios/pythonCourse
```

Some magic commands here to help

- `%cd`, `%ls`, `%mkdir`, `%pwd`, ...

Those magics are regularly used ...

- ... so common that % is no longer required (automagic)
- Working with iPython is almost like working with a Unix-like shell

**Automagic commands**

`cat`, `cp`, `env`, `ls`, `man`, `mkdir`, `more`, `mb`, `pwd`, `rm` and `rmdir`

# NumPy

## Understanding Data Types in Python (I)

**Static typing**

```
/* C code */
int result = 0;
for(int i=0; i<100; i++){
    result += i;
}
```

- Data types must be declared
- Data types cannot change
- Error detection in compilation
- Variables names are, basicly, labels

**Dynamic typing**

```
# Python code
result = 0
for i in range(100):
   result += i
```

- Data types are not declared
- Data types can change
- Error detection in run-time
- Variables are complex data structures (even for simple types)

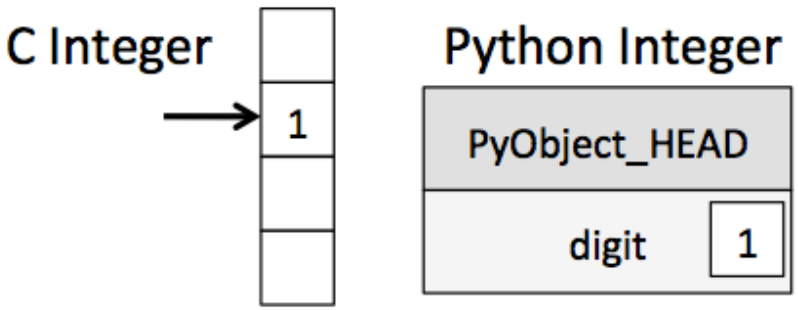
# NumPy

## Understanding Data Types in Python (II)

Dynamic typing must be implemented somewhere ...

**Python 3.4 source code**

```
struct _longobject {
    long ob_refcnt;
    PyTypeObject *ob_type;
    size_t ob_size;
    long ob_digit[1];
};
```

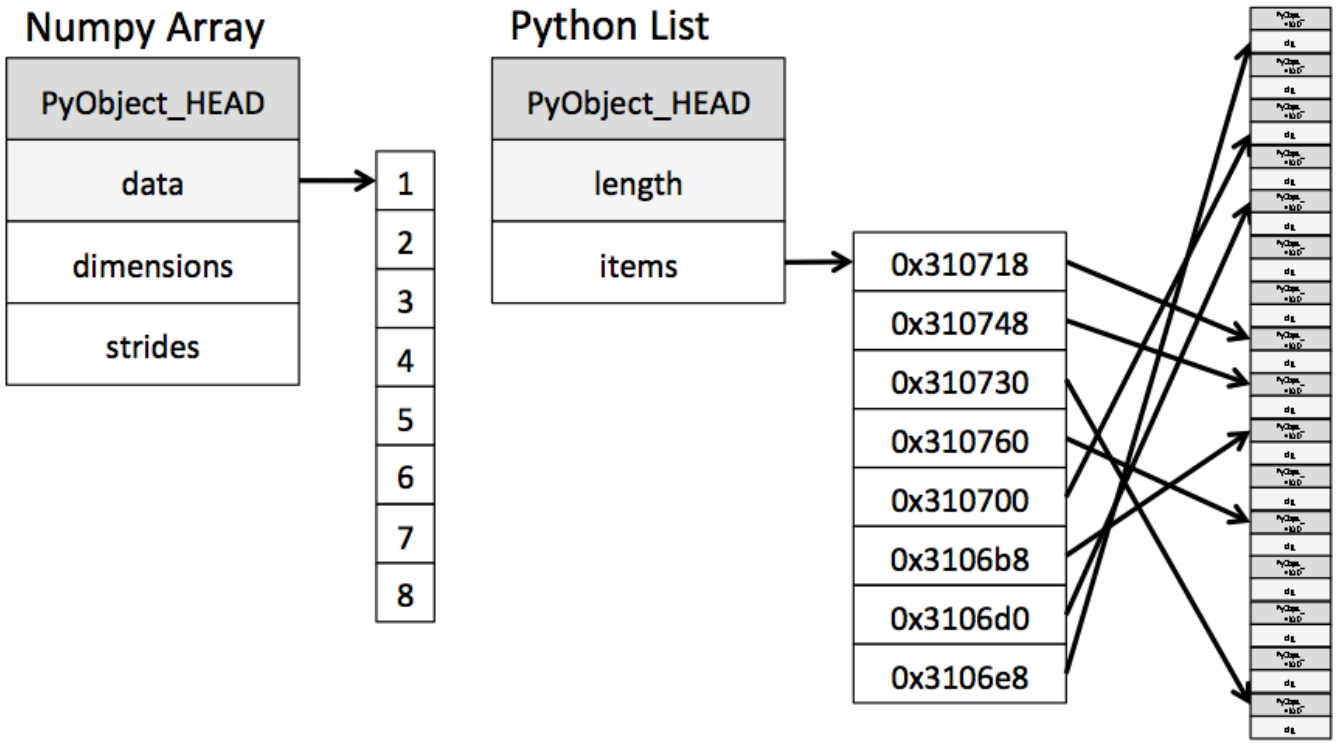


# NumPy

## Understanding Data Types in Python (III)

A Python list may contain different types

```
In [1]: L3 = [True, "2", 3.0, 4]
   ...: [type(item) for item in L3]
Out[1]: [bool, str, float, int]
```



# NumPy

## Understanding Data Types in Python (IV)

Standard Python data types are powerful and flexible

- Flexibility has a price: Reduced performance
- Not an big issue in generic programming
- A big issue in scientific programming
- We require efficient data manipulation mechanisms: NumPy

NumPy: Python package for numeric computation

- Efficient array implementation
- Fast mathematical functions
- Random numbers generation
- Static data types: Less flexibility

Most Python modules for AI/ML depend on NumPy, in particular

- Pandas (dataframes), Scikit-learn (ML), Seaborn (data visualization)

# NumPy

## Introduction

NumPy must be imported in order to be available

- Remember, you can use `np?` or `np.<TAB>`

The main component of NumPy is ndarray

- Python object
- Efficient matrix representation
- Homogeneus elements

**Convention**

```
import numpy as np
```

**Convention**

```
In [1]: array = np.array
    ([1,2,3])
In [2]: array
Out[1]: array([1, 2, 3])
```

# NumPy

## Matrix creation

NumPy functions for array creation from lists

- Lists must contain the same type, NumPy will upcast if needed
- `np.array([1, 4, 2, 5, 3])`
- `np.array([1, 2, 3, 4], dtype='float32')`: Explicit data type
- `np.array([3.14, 4, 2, 3])`: Upcast

NumPy functions for array creation from scratch

- `np.zeros(10, dtype=int)`: All zeros
- `np.ones((3, 5), dtype=float)`: All ones
- `np.full((3, 5), 3.14)`: Fill matrix
- `np.arange(0, 20, 2)`: Similar to Python's range()
- `np.linspace(0, 1, 5)`: Evenly spaced numbers
- `np.random.random((3, 3))`: Random numbers
- `np.random.normal(0, 1, (3, 3))`: Random normal numbers
- `np.random.randint(0, 10, (3, 3))`: Random integers
- `np.eye(3)`: Identity matrix
- `np.empty(3)`: Empty matrix

# NumPy

## NumPy data types

Python is implemented in C

- Data types in NumPy are based on those in C

Two styles to declare types

- String:
  `np.zeros(10, dtype='int16')`
- NumPy object:
  `np.zeros(10, dtype=np.int16)`

| Data type | Description |
|---|---|
| `bool_` | Boolean (True or False) stored as a byte |
| `int_` | Default integer type |
| `intc` | Identical to C |
| `intp` | Integer used for indexing |
| `int8` | Byte |
| `int16` | Integer |
| `int32` | Integer |
| `int64` | Integer |
| `uint8` | Unsigned integer |
| `uint16` | Unsigned integer |
| `uint32` | Unsigned integer |
| `uint64` | Unsigned integer |
| `float_` | Shorthand for float64 |
| `float16` | Half precision float |
| `float32` | Single precision float |
| `float64` | Double precision float |
| `complex_` | Shorthand for complex128 |
| `complex64` | Complex number |
| `complex128` | Complex number |