

Scientific Programming in Python

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

Departamento de Automática

Objectives

1. Motivate the need of efficient matrix representations.
2. Introduce some Python scientific tools.
3. Handle data representations in Python.
4. Basic data visualization with Python.
5. Provide a background for scientific programming.

Bibliography

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

Table of Contents

1. Overview

- Data Science
- The data scientist toolkit
- Anaconda
- Python IDEs for Data Science

2. Basics

3. Numpy

4. Pandas

5. Matplotlib

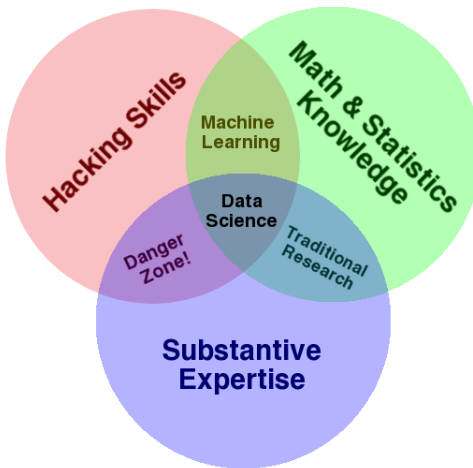
6. Seaborn

7. Classes in Python

- Syntax
- Class objects
- Constructors
- More about methods
- Solved exercises
- Approach to a final problem

Overview

Data Science



Data Science

The data scientist toolkit (I)

Data science is about manipulating data

- Need of specialized tools
- Two main languages: 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 toolkit (II)

Tool	Type	Description
iPython	Software	Advanced Python interpreter
Jupyter	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 functionality
- Enhanced UI
- External editor

Running iPython:
\$ ipython

Jupyter

Python notebooks

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

Running Jupyter:
\$ jupyter notebook



Rodeo

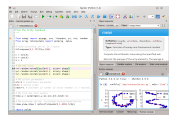
Python version of RStudio

- 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 thorough `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 builtin functions!

iPython

Basics (II)

Press <tab> to complete almost everything

- Object contents

```
In [21]: a = [1,2,1]
In [22]: a.
a.append  a.count  a.insert  a.reverse
a.clear   a.extend  a.pop     a.sort
a.copy    a.index   a.remove
```

- Packages

```
In [26]: import num
numba      numpy
numbers    numpydoc
numexpr
```

- Wildcards

```
In [29]: *Warning?
%%!
ArithmeticError      BaseException
AssertionError        BlockingIOError
AttributeError         BrokenPipeError
BufferError
```

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

Characteristics

Polymorphism (I)

Polymorphism

Mechanism of object-oriented programming that allows to invoke a method whose implementation will depend on the object that does it.

Figura 1: Example of polymorphism. Obtained from: <http://virtual.uaeh.edu.mx>

Characteristics

Polymorphism (II)

Figura 2: Example of polymorphism. Obtained from: <http://datateca.unad.edu.co>

Characteristics

Abstraction and encapsulation (I)

Abstraction

Mechanism that allows the isolation of the not relevant information to a level of knowledge.

- A driver does not need to know how the carburetor works.
- To talk on the phone does not need to know how the voice is transferred.
- To use a computer do not need to know the internal composition of their materials.

Characteristics

Abstraction and encapsulation (II)

Encapsulation

Mechanism use to provide an access level to methods and attributes for avoiding unexpected state changes. This mechanism is used to limit the visibility of the attributes and to create methods controlling them (`set()` y `get()`).

The most common access levels are:

- **public:** visible for everyone [default level in Python].
- **private:** visible for the creator class [start with a double underscore and does not end in the same manner].
- **protected:** visible for the creator class and its descendents [not exist in Python].

Characteristics

Abstraction and encapsulation (III). Example 1

Figura 3: Example of abstraction and encapsulation. Obtained from: <https://binalparekh.wordpress.com>

Characteristics(III)

Abstraction and encapsulation (IV). Example 2

Figura 4: Example of abstraction and encapsulation. Obtained from: <http://www.onlinebuff.com>

Classes in Python

Syntax (I)

- **Class:** Start with the word **class** followed by class name written in **capital letter** and a colon [Substantives].
- **Attributes:** A lowercase noun.
 - There is no need to declare attributes.
- **Inherited class:** Similar to a class but the class name followed by the class father in brackets.
- **Instance:** Object in lower case followed by the class assignment.

```
coche.py
```

Classes in Python

Syntax (II)

- **Method:** Start with the word `def`, and later the method, a verb, in lower case is written. Next, the parameter in brackets and a colon (`print_name()`).
 - Methods receive automatically a reference to the object (usually named `self`).
- **Constructor:** Method whose name is `__init__()`, the first attribute is `self` and then the class attributes are written.
- **main:** Method defined with `def main() :`. In it, the wished commands are specified and after it, an exit condition is created. The `sys` module is required to be imported at the beginning.
- All methods and attributes are public.
 - By convention, private members begin with double underscore (`__varName`, `__method_name()`)

Classes in Python

Syntax (III). Example 1

```
main.py
```

Classes in Python

Syntax (IV). Example 2

```
bicicleta.py
```

Classes in Python

Syntax (V). Example 3

Time.py

Classes in Python

Class objects

Two operations on classes

Attribute references

Accesses an attribute value
Standard dot syntax

```
obj.name
```

Example

```
time.hour = 4  
print(time.hour)  
hour = time.hour
```

Instantiation

Creates a new object
Standard functional notation

```
x = MyClass()
```

Example

```
time = Time()
```


Constructors (I)

Instantiation creates empty objects

- We usually need to initialize attributes
- Initialization operations

Constructor: Method called when an object is created

- In Python, it is the `__init__()`
- A constructor can get arguments

Constructors (II)

Time.py with constructor

Other special methods

In addition to special method `__init__`, there are several others, including:

- `__str__(self)` It should return a string with **self** information. When `print()` is invoked with the object, if the method `__str__()` is defined, Python shows the result of running this method on the object.
- `__len__(self)` It should return the length or “size” of object (number of elements if is a set or queue).
- `__add__(self, otro_obj)` It allows to apply the addition operator (+) to objects of the class in which it is defined.
- `__mul__(self, otro_obj)` It allows to apply the multiplication operator (*) to objects of the class in which it is defined.
- `__comp__(self, otro_obj)` It allows to apply the comparison operators (<, >, <=, >=, ==, !=) to objects of the class in which it is defined. It should return 0 if they are equal, -1 if **self** is smaller than **other_obj** and 1 if **self** is greater than **other_obj**.

Overriding methods (I)

Often we need to adapt an inheritance method: **Overriding**

Overriding example

Overriding methods (II)

Still possible to get superclass' method with `super()`

`super()` example

Exercise statement

Animal class

1. Create the `animal` class.
2. Create the constructor. The class will have the attributes `tipo` and `patas`.
3. Create the get methods from both attributes which receive like own parameter the animal through `self` and return respectively the `tipo` and `patas`.
4. Create two instances of animals using the constructor.
5. Print the attributes of both instances.

Solved exercise

Animal class

```
animales.py
```

Solved exercise

Animal class

1. Create a `gato` class in the same file which inherits from the `animal` class.
2. Create the constructor and add the `sonido` attribute.
3. Create the method `maullar` which prints the sound MIAU.
4. Create a instance and check the methods.

Solved exercise

Class Animals

animales.py

Exercise statement

Class Parcela

1. Create a script containing the class `Parcela`.
2. Create the constructor. The class will have the attributes `uso_suelo` and `valor`.
3. Create the `valoracion` method to calculate the tax associated with the parcel as follows:
 - For single-family residential: $tasa = 0.05 * valor$
 - For multifamily residential: $tasa = 0.04 * valor$
 - For all other land uses: $tasa = 0.02 * valor$
4. Use the class from another script named `tasaparcela.py` which you create una instance of `Parcela` named `miparcela` using the constructor.
5. Print the attribute `uso_suelo` of the instance.
6. Use the method `valoracion` of `Parcel` to calculate the assessment of `miparcela`.

Solved exercise

Class Parcela

```
claseparcela.py
```

Solved exercise

Use of Parcela

```
tasaparcels.py
```

Source

Solved exercise. Serializando objetos Parcela

tasaparcela_pickle.py

Exercise statement

Rio class

1. Create the `Rio` class.
2. Create the constructor and add the `nombre` and `longitud` attributes.
3. `Longitud` attribute must be private.
4. Create the `setLongitud` method which receives `self` and `longitudR` and allows the set of any value for `longitud`.
5. Create the `getNombre` method which obtains the name of the river.
6. Create the `getLongitud` method which obtains the river length.
7. Create an instance and check the methods.
8. Try to do an assignment of `rio.nombre` and other assignment with `rio.longitud`. What happens? It is correct to invoke the method named `rio.getLongitud()` out of the classes? How do you explain that?

Exercise statement

Establishment of hierarchies from `Rio` class

1. Add to the `Rio` class the attribute `caudal` and the method `trasvasar` which receives two rivers and transfers 5 liters from the first to the second.
2. Create the `Afluente` class which inherits from `Rio`.
3. Create the method `__init__` of `Afluente` which initializes its `nombre` and `longitud` and, also, `afluenteDeRio`, new attribute initialized with the name of the river which the affluent starts.
4. Is there any polymorphism in this sample?
5. Create the main and exit condition and try it. Does the main position affect to the application?
6. Experiment now with conditions and iterative structures limiting when a river can transfer water or try to do some transfer at the same time.

Y más...

Aprende más: [4]

Bibliographic references I



[1] Lenguajes de programacion, capítulo 1.

Lenguajes de programacion.

http:

[//rua.ua.es/dspace/bitstream/10045/4030/1/tema01.pdf](http://rua.ua.es/dspace/bitstream/10045/4030/1/tema01.pdf)



[2] Downey, A and Elkner, J and MEYER, C.

Aprenda a Pensar como un Programador con Python, capitulos 14 y 16.

Green Tea Press, 2002.



[3] G. van Rossum, Jr. Fred L. Drake.

Python Tutorial Release 3.2.3, chapter 9.

Python Software Foundation, 2012.



[4] Dusty Phillips

Python 3 Object Oriented Programming.

Packt Publishing, 2010.