sesion1

February 4, 2020

1 SESION 1

- Introducion al curso
- Breve Historia
- Entornos: Interprete, scripts, Jupyter
- Instalacion Anaconda
- Numpy: Arreglos

1.1 Instalacion y Configuracion de Ananconda

The open-source Anaconda Distribution is the easiest way to perform Python/R data science and machine learning on Linux, Windows, and Mac OS X. With over 19 million users worldwide, it is the industry standard for developing, testing, and training on a single machine, enabling individual data scientists to: * Quickly download 7,500+ Python/R data science packages * Manage libraries, dependencies, and environments with Conda * Develop and train machine learning and deep learning models with scikit-learn, TensorFlow, and Theano * Analyze data with scalability and performance with Dask, NumPy, pandas, and Numba * Visualize results with Matplotlib, Bokeh, Datashader, and Holoviews

1.2 Python Version

Some previous versions of the documentation remain available online. Use the list below to select a version to view.

For unreleased (in development) documentation, see In Development Versions.

- Python 3.8.1, documentation released on 18 December 2019.
- Python 3.8.0, documentation released on 14 October 2019.
- Python 3.7.6, documentation released on 18 December 2019.
- Python 3.7.5, documentation released on 15 October 2019.
- Python 3.7.4, documentation released on 08 July 2019.
- Python 3.7.3, documentation released on 25 March 2019.
- Python 3.7.2, documentation released on 24 December 2018.
- Python 3.7.1, documentation released on 20 October 2018.
- Python 3.7.0, documentation released on 27 June 2018.
- Python 3.3.7, documentation released on 19 September 2017.
- Python 3.3.6, documentation released on 12 October 2014.
- Python 3.3.5, documentation released on 9 March 2014.
- Python 3.2.2, documentation released on 4 September 2011.

- Python 3.0.1, documentation released on 13 February 2009.
- Python 3.0, documentation released on 3 December 2008.
- Python 2.7.17, documentation released on 19 October 2019
- Python 2.7.16, documentation released on 02 March 2019
- Python 2.7.15, documentation released on 30 April 2018
- Python 2.7.7, documentation released on 31 May 2014.
- Python 2.7.6, documentation released on 10 November 2013.
- Python 2.6.9, documentation released on 29 October 2013.
- Python 2.6.8, documentation released on 10 April 2012.
- Python 2.6, documentation released on 1 October 2008.
- Python 2.5.4, documentation released on 23 December 2008.
- Python 2.2.3, documentation released on 30 May 2003.
- Python 2.2.2, documentation released on 14 October 2002.
- Python 2.2.1, documentation released on 10 April 2002.
- Python 2.0.1, documentation released on 22 June 2001.
- Python 2.0, documentation released on 16 October 2000.
- Python 1.6, documentation released on 5 September 2000.
- Python 1.4, documentation released on 25 October 1996.

1.3 Instalacion de librerias

conda y pip son herramientas para manejar y desplegar aplicaciones, ambientes y paquetes.

Se puede usar el comando `conda` para instalar librerias asi:

```
conda install seaborn
```

Tambien se puede usar el comando `pip` para instalar librerias asi:

```
pip install seaborn
```

Se puede forzar la instalacion de paquetes de la siguiente forma:

```
conda install -c conda-forge statsmodels
```

1.4 Importar librerias

La forma mas directa de importar una librerias es:

```
import sys
import os
import numpy
```

Otra forma que puede ser mas util, es instalar y dar un alias al nombre de la libreria:

```
import numpy as np
```

Igualmente se puede instalar un submodulo en vez de toda la libreria:

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

O se puede importar una o varias funciones de la libreria:

```
from random import random, uniform, gauss
```

Para revisar si un modulo esta instalado:

```
'numpy' in sys.modules
'scipy' in sys.modules
```

Se puede revisar si un modulo esta instalado asi:

```
'numpy' in sys.modules
'scipy' in sys.modules
```

1.5 Python Kernel

Referencia The kernel is the server that enables Python programmers to run cells within Notebook. You typically see the kernel commands in a separate command or terminal window.

Each entry shows the time the kernel executed the task, which application the command executed, the task it performed, and any resources affected. In most cases, you don't need to do anything with this window, but viewing it can be helpful when you run into problems because you often see error messages that can help you resolve an issue.

You control the kernel in a number of ways. For example, saving a file issues a command to the kernel, which carries the task out for you. However, you also find some kernel-specific commands on the Kernel menu, which are described in the following list:

- **Interrupt**: Causes the kernel to stop performing the current task without actually shutting the kernel down. You can use this option when you want to do something like stop processing a large dataset.
- **Restart**: Stops the kernel and starts it again. This option causes you to lose all the variable data. However, in some cases, this is precisely what you need to do when the environment has become dirty with old data.
- **Restart & Clear Output**: Stops the kernel, starts it again, and clears all the existing cell outputs.
- **Restart & Run All**: Stops the kernel, starts it again, and then runs every cell starting from the top cell and ending with the last cell. When Notebook reaches the bottom, it selects the last cell but doesn't insert a new one.
- **Reconnect**: Recreates the connection to the kernel. In some cases, environmental or other issues could cause the application to lose its connection, so you use this option to reestablish the connection without loss of variable data.
- **Shutdown**: Shuts the kernel down. You may perform this step in preparation for using a different kernel.
- **Change Kernel**: Selects a different kernel from the list of kernels you have installed. For example, you may want to test an application using various Python versions to ensure that it runs on all of them.

1.6 SHORTCUTS

- New cell below: "Esc+b"
- New cell above: "Esc+a"
- Delete cell: "Esc+D+D"
- Comment region:"Crtl+/" or "Ctrl+}"
- SHIFT + M = It merges multiple selected cells into one cell.
- CTRL + SHIFT + – = It splits the current cell into two cells from where your cursor is.

1.7 Magics

```
%ls
%dirs
%lsmagic
%matplotlib inline ## It allows the output of plotting command to be displayed inline i.e. in Ju
%who ## will list all variables that exist in the global scope.
%time ## will give you information about the time taken in a single run of the code in your cell
```

1.8 Tipos de Variable

- Tipos de Variables: Enteros, flotantes, cadenas de caracteres, boleanos, bytes, flotantes, complejos

1.8.1 Enteros y Flotantes

```
In [8]: a=1
        b=3
        a+b
```

```
Out[8]: 4
```

```
In [3]: a+b;
```

```
In [4]: print(a+b)
```

```
4
```

```
In [5]: a-1
```

```
Out[5]: 0
```

```
In [6]: c=a*b
```

```
Out[6]: 3
```

```
In [16]: 2**(1/2.)
```

```
Out[16]: 1.4142135623730951
```

```
In [8]: int(5.5)
```

```
Out[8]: 5
```

```
In [9]: float(5)
```

```
Out[9]: 5.0
```

1.8.2 Cadenas de caracteres

- Referencia 1
- Referencia 2

Particularmente las cadenas de caracteres "strings" en python son clases. Sobre los "strings" se pueden realizar multiples operaciones:

```
In [3]: "abc"+"def"

Out[3]: 'abcdef'

In [2]: 3*"abc"

Out[2]: 'abcabcabc'

In [3]: 3*"abc"+"def"

Out[3]: 'abcabcabcdef'

In [1]: "hola este es el curso de programacion en python".split(" ")

Out[1]: ['hola', 'este', 'es', 'el', 'curso', 'de', 'programacion', 'en', 'python']

In [5]: "hola este es el curso de programacion en python".upper()

Out[5]: 'HOLA ESTE ES EL CURSO DE PROGRAMACION EN PYTHON'

In [6]: "hola este es el curso de programacion en python".upper().lower()

Out[6]: 'hola este es el curso de programacion en python'

In [8]: "hola este es el curso de programacion en python".replace("python","java")

Out[8]: 'hola este es el curso de programacion en java'

In [9]: len("hola este es el curso de programacion en python")

Out[9]: 47

In [10]: type("hola este es el curso de programacion en python")

Out[10]: str
```

1.9 Tipos de Secuencias

Referencia

En python existen diferentes tipos de estructuras que permiten el almacenamiento y manipulacion de conjuntos de datos, A continuacion se mencionaran algunas de las estructuras mas basicas: Listas, Tuplas, Conjuntos

1.9.1 Tuplas

The items of a tuple are arbitrary Python objects. Tuples of two or more items are formed by comma-separated lists of expressions. A tuple of one item (a 'singleton') can be formed by affixing a comma to an expression (an expression by itself does not create a tuple, since parentheses must be usable for grouping of expressions). An empty tuple can be formed by an empty pair of parentheses.

tupla1 = ("hola este es el curso de","python", "y", "R") print(tupla1)

```
In [2]: tupla1[0]

Out[2]: 'hola este es el curso de'

In [3]: tupla1[-1]

Out[3]: 'R'

In [6]: tupla1[1:4]

Out[6]: ('python', 'y', 'R')

In [7]: len(tupla1)

Out[7]: 4

In [15]: type(tupla1)

Out[15]: tuple
```

1.9.2 Listas

The items of a list are arbitrary Python objects. Lists are formed by placing a comma-separated list of expressions in square brackets. (Note that there are no special cases needed to form lists of length 0 or 1.)

```
In [11]: utiles_inutiles = ["Lapiz","Borrador","Cuaderno","libro","sacapuntas","colores"]

In [12]: len(utiles_inutiles)

Out[12]: 6

In [13]: utiles_inutiles[-2]

Out[13]: 'sacapuntas'

In [14]: type(utiles_inutiles)

Out[14]: list

In [ ]: # Secuencias

In [25]: range(1,10,1)

Out[25]: range(1, 10)
```

1.10 Conjuntos

Python also includes a data type for sets. A set is an unordered collection with no duplicate elements. Basic uses include membership testing and eliminating duplicate entries. Set objects also support mathematical operations like union, intersection, difference, and symmetric difference.

```
In [ ]: a = {6,8,3,"hola"}

In [ ]: type(a)

In [32]: a = set('abracadabra')

In [33]: a

Out[33]: {'a', 'b', 'c', 'd', 'r'}
```

1.11 Tipos de Mapeo

These represent finite sets of objects indexed by arbitrary index sets. The subscript notation a[k] selects the item indexed by k from the mapping a; this can be used in expressions and as the target of assignments or del statements. The built-in function len() returns the number of items in a mapping.

1.11.1 Diccionarios

These represent finite sets of objects indexed by nearly arbitrary values. The only types of values not acceptable as keys are values containing lists or dictionaries or other mutable types that are compared by value rather than by object identity, the reason being that the efficient implementation of dictionaries requires a key's hash value to remain constant. Numeric types used for keys obey the normal rules for numeric comparison: if two numbers compare equal (e.g., 1 and 1.0) then they can be used interchangeably to index the same dictionary entry.

```
In [34]: dict_utils = {"cuadernos":12,"colores":24,"lapiz":1,"borrador":2,"libros":12}

In [35]: dict_utils["cuadernos"]

Out[35]: 12

In [36]: dict_utils.items()

Out[36]: dict_items([('cuadernos', 12), ('colores', 24), ('lapiz', 1), ('borrador', 2), ('libros

In [37]: dict_utils.keys()

Out[37]: dict_keys(['cuadernos', 'colores', 'lapiz', 'borrador', 'libros'])
```

1.12 Example (standard input):

Read two integers from STDIN and print three lines where:

- The first line contains the sum of the two numbers.
- The second line contains the difference of the two numbers (first - second).
- The third line contains the product of the two numbers.

```
In [ ]: if __name__ == '__main__':
            a = int(input())
            b = int(input())
```

1.13 Task

Read two integers and print two lines. The first line should contain integer division, // . The second line should contain float division, / .

Note: You don't need to perform any rounding or formatting operations.

```
In [ ]: if __name__ == '__main__':
            a = int(input())
            b = int(input())
```

1.14 Built in Functions

abs(), delattr(), hash(), memoryview(), t(), l(), ct(), lp(), min(), setattr(), any(), dir(), hex(), next(), slice(), ascii(), divmod(), id(), object(), rted(), n(), enumerate(), input(), oct(), staticmethod(), bool(), eval(), int(), open(), str(), breakpoint(), exec(), isinstance(), ord(), sum(), bytearray(), filter(), issubclass(), pow(), super(), bytes(), float(), iter(), print(), tuple(), callable(), format(), len(), property(), type(), chr(), frozenset(), list(), range(), vars(), classmethod(), getattr(), locals(), repr(), zip(), compile(), globals(), map(), reversed(), **import**(), complex(), hasattr(), max(), round()

```
In [17]: type(c)

Out[17]: float

In [15]: d=a/b

In [18]: type(d)

Out[18]: float

In [24]: word='hola mundo!'
         print(word)

hola mundo!


In [25]: type(word)

Out[25]: str

In [12]: str(a)

Out[12]: '1'

In [13]: float(a)

Out[13]: 1.0
```

1.15 Numpy Library

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities
- Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

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

1.16 Arrays

NumPy provides an N-dimensional array type, the ndarray, which describes a collection of "items" of the same type. The items can be indexed using for example N integers.

All ndarrays are homogenous: every item takes up the same size block of memory, and all blocks are interpreted in exactly the same way. How each item in the array is to be interpreted is specified by a separate data-type object, one of which is associated with every array. In addition to basic types (integers, floats, etc.), the data type objects can also represent data structures.

An item extracted from an array, e.g., by indexing, is represented by a Python object whose type is one of the array scalar types built in NumPy. The array scalars allow easy manipulation of also more complicated arrangements of data.

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

Out[2]: array([1, 2, 3])

In [3]: x=np.array([[1,2,3],[5,6,7],[8,9,10]])
        x.view()

Out[3]: array([[ 1,  2,  3],
               [ 5,  6,  7],
               [ 8,  9, 10]])

In [4]: y=np.arange(0,9)
        y.view()

Out[4]: array([0, 1, 2, 3, 4, 5, 6, 7, 8])

In [9]: np.zeros(5)

Out[9]: array([ 0.,  0.,  0.,  0.,  0.])

In [10]: np.ones(10)

Out[10]: array([ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.])
```

```
In [13]: np.eye(4)

Out[13]: array([[1., 0., 0., 0.],
                [0., 1., 0., 0.],
                [0., 0., 1., 0.],
                [0., 0., 0., 1.]])
```

1.16.1 Array: Atributos y Metodos

```
In [39]: y.size

Out[39]: 9

In [40]: y.shape

Out[40]: (9,)

In [41]: y.reshape((3,3))

Out[41]: array([[0, 1, 2],
                [3, 4, 5],
                [6, 7, 8]])

In [43]: y = np.arange(0,9).reshape((3,3))
         y.view()

Out[43]: array([[0, 1, 2],
                [3, 4, 5],
                [6, 7, 8]])
```

1.16.2 Array: Operaciones Basicas

```
In [44]: x+y

Out[44]: array([[ 1,  3,  5],
                [ 8, 10, 12],
                [14, 16, 18]])

In [47]: y/x

Out[47]: array([[0.        , 0.5       , 0.66666667],
                [0.6       , 0.66666667, 0.71428571],
                [0.75      , 0.77777778, 0.8       ]])

In [48]: np.matmul(x,y)

Out[48]: array([[ 24,  30,  36],
                [ 60,  78,  96],
                [ 87, 114, 141]])
```

2 Numpy Routines

Numpy include different modules like: random, testing, linalg, fft, matlib

```
In [2]: np.sin(0)

Out[2]: 0.0

In [3]: np.cos(0)

Out[3]: 1.0

In [4]: np.exp(0)

Out[4]: 1.0

In [5]: np.linspace(0,10,30)

Out[5]: array([  0.        ,   0.34482759,   0.68965517,   1.03448276,
                 1.37931034,   1.72413793,   2.06896552,   2.4137931 ,
                 2.75862069,   3.10344828,   3.44827586,   3.79310345,
                 4.13793103,   4.48275862,   4.82758621,   5.17241379,
                 5.51724138,   5.86206897,   6.20689655,   6.55172414,
                 6.89655172,   7.24137931,   7.5862069 ,   7.93103448,
                 8.27586207,   8.62068966,   8.96551724,   9.31034483,
                 9.65517241,  10.        ])

In [8]: np.sin(np.linspace(0,2*np.pi,50))

Out[8]: array([  0.00000000e+00,   1.27877162e-01,   2.53654584e-01,
                 3.75267005e-01,   4.90717552e-01,   5.98110530e-01,
                 6.95682551e-01,   7.81831482e-01,   8.55142763e-01,
                 9.14412623e-01,   9.58667853e-01,   9.87181783e-01,
                 9.99486216e-01,   9.95379113e-01,   9.74927912e-01,
                 9.38468422e-01,   8.86599306e-01,   8.20172255e-01,
                 7.40277997e-01,   6.48228395e-01,   5.45534901e-01,
                 4.33883739e-01,   3.15108218e-01,   1.91158629e-01,
                 6.40702200e-02,  -6.40702200e-02,  -1.91158629e-01,
                -3.15108218e-01,  -4.33883739e-01,  -5.45534901e-01,
                -6.48228395e-01,  -7.40277997e-01,  -8.20172255e-01,
                -8.86599306e-01,  -9.38468422e-01,  -9.74927912e-01,
                -9.95379113e-01,  -9.99486216e-01,  -9.87181783e-01,
                -9.58667853e-01,  -9.14412623e-01,  -8.55142763e-01,
                -7.81831482e-01,  -6.95682551e-01,  -5.98110530e-01,
                -4.90717552e-01,  -3.75267005e-01,  -2.53654584e-01,
                -1.27877162e-01,  -2.44929360e-16])
```

2.0.1 Numpy: Linear Algebra

```
In [50]: np.linalg.inv(x)
```

```
Out[50]: array([[-1.0293942e+16,  2.4019198e+16, -1.3725256e+16],
                [ 2.0587884e+16, -4.8038396e+16,  2.7450512e+16],
                [-1.0293942e+16,  2.4019198e+16, -1.3725256e+16]])
```

```
In [52]: np.linalg.det(x)
```

```
Out[52]: 2.914335439641041e-16
```

2.0.2 linalg.det: The linalg.det tool computes the determinant of an array.

```
In [ ]: np.linalg.det([[1 , 2], [2, 1]])        #Output : -3.0
```

2.0.3 linalg.eig: The linalg.eig computes the eigenvalues and right eigenvectors of a square array.

```
In [ ]: vals, vecs = np.linalg.eig([[1 , 2], [2, 1]])
        print(vals)                                          #Output : [ 3. -1.]
        print(vecs)                                          #Output : [[ 0.70710678 -0.70710678]
                                                             #         [ 0.70710678  0.70710678]]
```

2.0.4 linalg.inv: The linalg.inv tool computes the (multiplicative) inverse of a matrix.

```
In [5]: np.linalg.inv([[1 , 2], [2, 1]])        #Output : [[-0.33333333  0.66666667]
                                                 #         [ 0.66666667 -0.33333333]]
```

```
Out[5]: array([[-0.33333333,  0.66666667],
               [ 0.66666667, -0.33333333]])
```

```
In [ ]: np.linalg.solve()
```

2.1 Task:

Solve the following operations: $\mathbf{A}+\mathbf{B}$, $\mathbf{A}-\mathbf{B}$, $\mathbf{AB}$, $\mathbf{A}(\mathbf{BC})$, $(\mathbf{AB})\mathbf{C}$,

$$A=\begin{pmatrix}0 & 1 & -2\\ 3 & 4 & 5\\ -6 & 7 & 15\end{pmatrix} \quad ,B=\begin{pmatrix}0 & -5 & 3\\ 5 & 2 & -1\\ -4 & 2 & 0\end{pmatrix} \quad ,C=\begin{pmatrix}6 & -2 & -3\\ 2 & 0 & 1\\ 0 & 5 & 7\end{pmatrix}$$

2.2 Task:

Solve the following equation systems

$$2x_1-5x_2=3 \tag{1}$$
$$5x_1+8x_2=5 \tag{2}$$

Solve the following equation systems

$$2x_1 + 2x_2 - x_3 = 2 \tag{3}$$
$$x_1 - 3x_2 + x_3 = 0 \tag{4}$$
$$3x_1 + 4x_2 - x_3 = 1 \tag{5}$$

```
In [9]: #A = np.array([[0, 1, -2],[3, 4, 5],[-6, 7, 15]])
        A = np.array([[2, 2,-1],[1,-3,1],[3,4,-1]])
        b = np.array([2,0,1])
        A.view()

Out[9]: array([[ 2,  2, -1],
               [ 1, -3,  1],
               [ 3,  4, -1]])

In [6]: np.linalg.det(A)

Out[6]: -164.99999999999994

In [11]: x=np.linalg.solve(A,b)

In [12]: x.view()

Out[12]: array([ 0.42857143, -0.71428571, -2.57142857])
```