

Laboratorio Nacional de Modelaje y Sensores Remotos

Taller de Programación con Python

DR. VICTOR M. RODRIGUEZ MORENO

Responsable del Laboratorio Nacional de Modelaje y Sensores Remotos

IIS JORGE E. MAURICIO RUVALCABA. PSP. INIFAP

MC Arturo Corrales Suastegui INIFAP

Breve historia

- Creado por Guido Van Rossum en 1991.
- Es sucesor del lenguaje ABC
- Su nombre esta inspirado en el programa de televisión de la BBC “Monty Python Flying Circus”

¿Qué es?

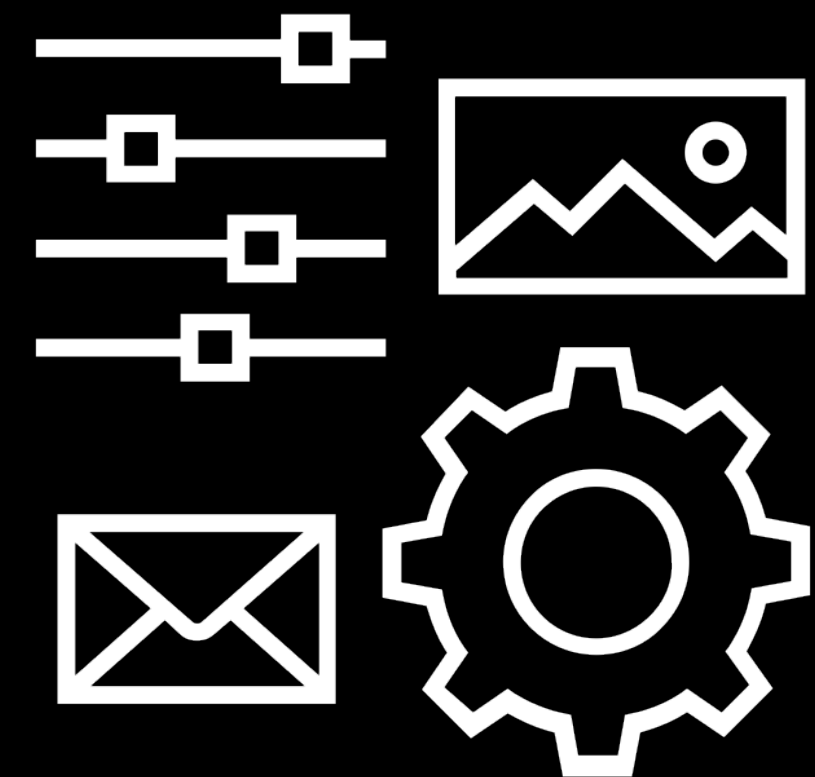
- Interpretado
- Multiparadigma
- Tipado dinámico
- Sintaxis de código legible
- Open source
- Multiplataforma

¿Porqué aprenderlo?

- Curva de aprendizaje baja; esto lo vamos a ver desde el primer ejercicio de este taller. Una sola línea de código nos da el resultado directamente
- Sintaxis legible (simple)
- Utilidad en muchos entornos de trabajo. Servidor, páginas web, aplicaciones desktop, etc; agricultura, algoritmos de estimación de datos perdidos, reconstrucción de series de datos, etc, etc

¿Big Data?

Es un término para conjuntos de datos que son tan grandes o complejos que el software de aplicación de procesamiento de datos tradicional es inadecuado para tratar con ellos.



¿Aplicación?



Detección de fraudes en
tiempo real



Análisis de
sentimientos del
consumidor



Gestión de inteligencia del tráfico

¿Características?

- Volumen: demasiados datos para manejar fácilmente
- Velocidad: la velocidad de entrada y salida de datos dificulta su análisis
- Variedad: el rango y el tipo de fuentes de datos son demasiado grandes para asimilar

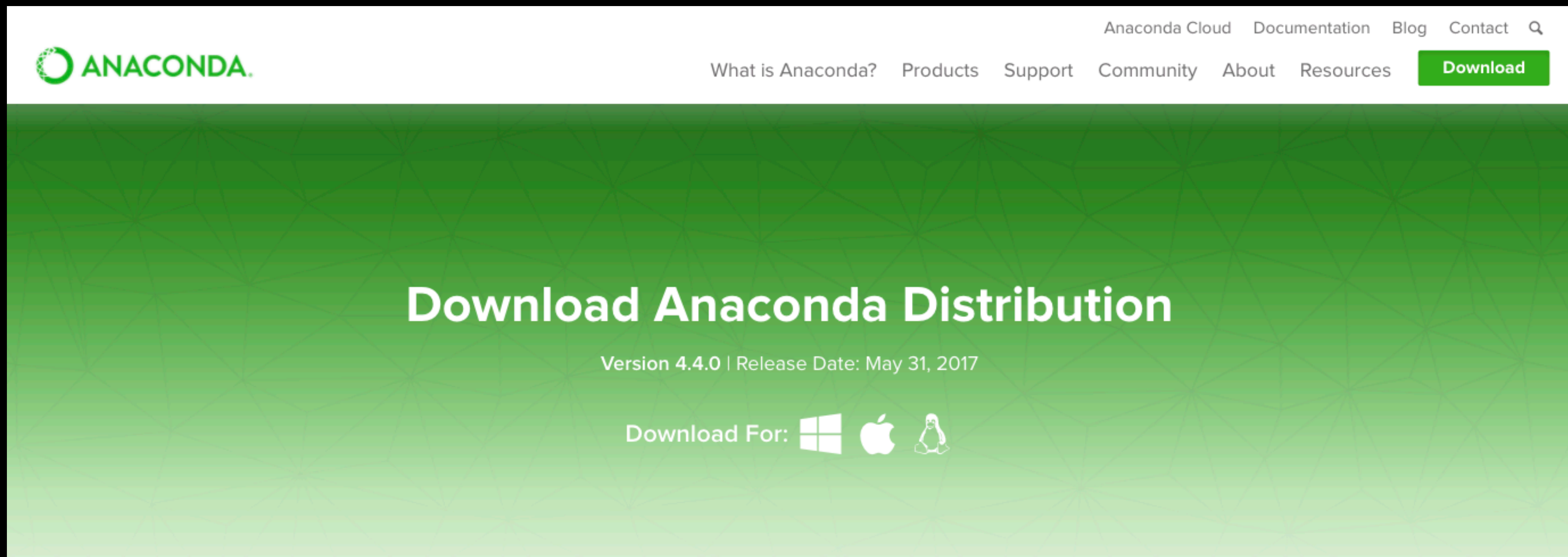
Inicio del taller

Exámen

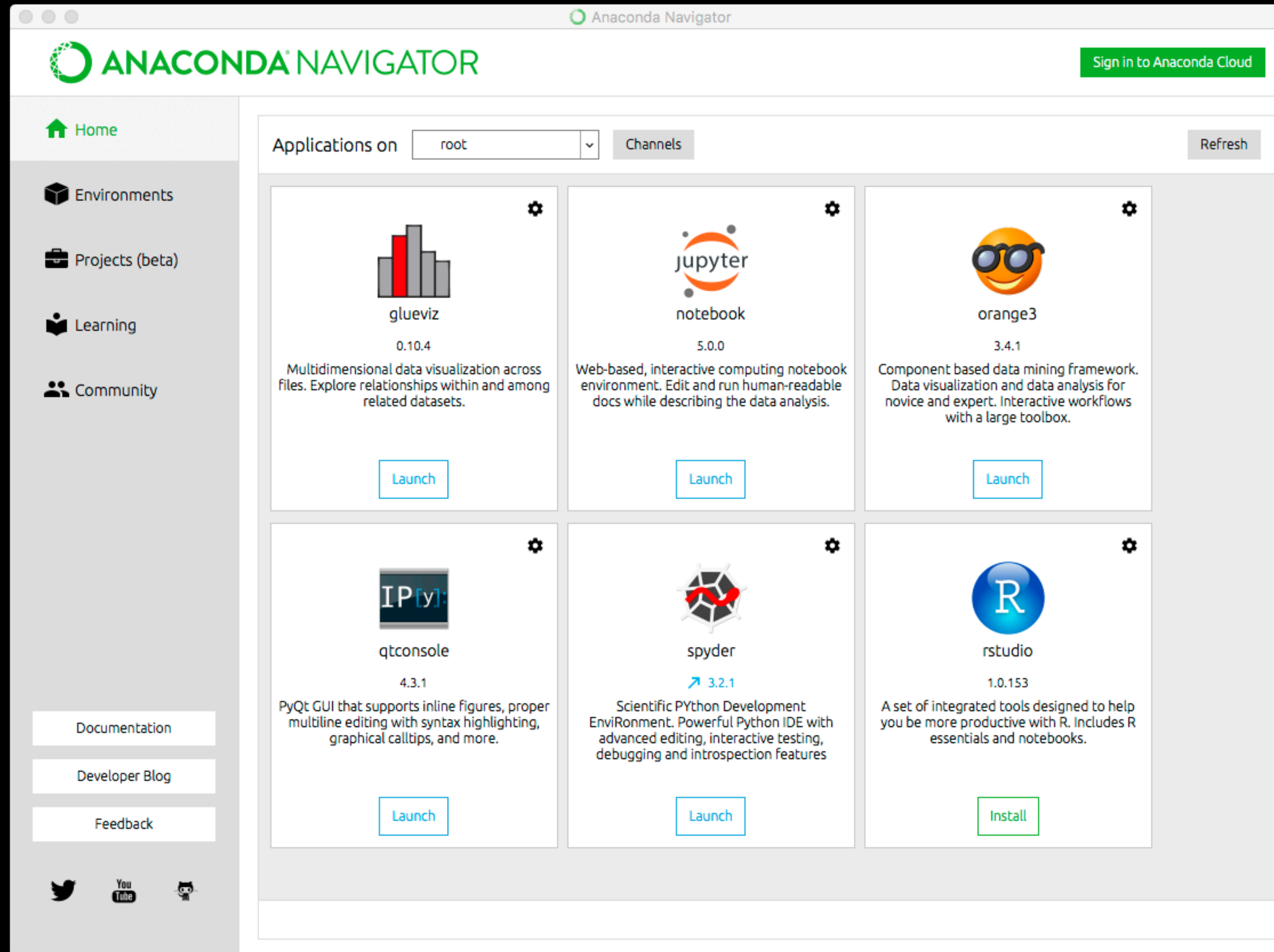
taller_python/ejercicios/passwords.md

Instalación Anaconda

- www.anaconda.com/downloads
- Windows, macOS, Linux
- Windows-macOS: asistente, clic, clic, clic, Finalizar
- Linux: descargar el archivo .sh, ejecutarlo.

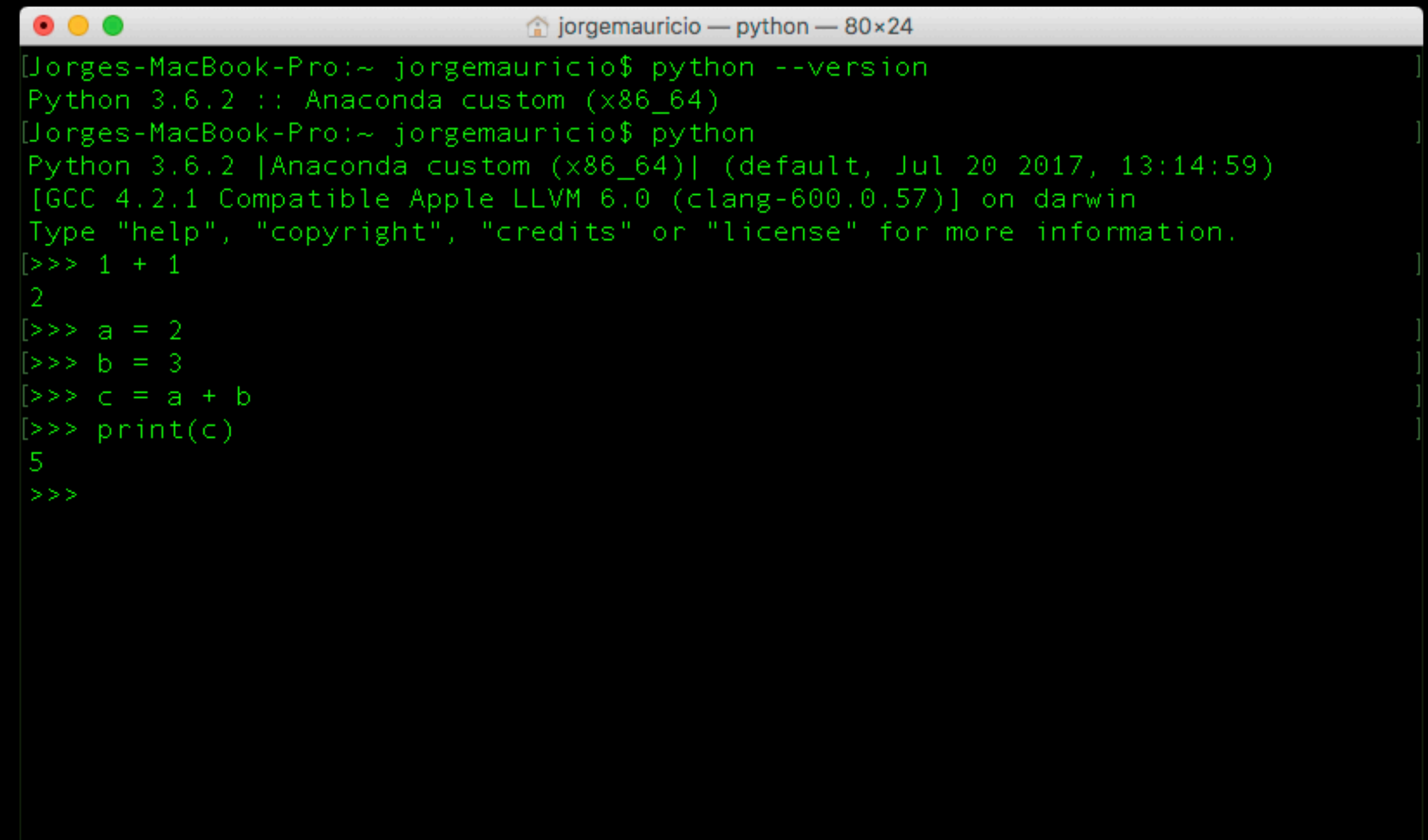


Anaconda Navigator



Modo Interactivo

- Para emplear el modo interactivo, se debe de ingresar el comando **python** en la terminal de comandos. De ahí en adelante las expresiones pueden ser introducidas una a una, pudiendo verse el resultado de su evaluación inmediatamente, lo que da la posibilidad de probar porciones de código en el modo interactivo antes de integrarlo como parte de un programa.

A screenshot of a macOS terminal window titled "jorgemaurocio — python — 80x24". The terminal shows the execution of the Python command-line interface. The user enters "python --version", which outputs "Python 3.6.2 :: Anaconda custom (x86_64)". Then, the user enters "python", which starts the interactive shell. The shell displays the Python version, the environment (Anaconda custom), the date and time, and the compiler information. The user then enters several lines of code: "1 + 1", "a = 2", "b = 3", "c = a + b", and "print(c)". The shell outputs the results of these commands: "2", "3", and "5".

```
jorgemaurocio — python — 80x24
[Jorges-MacBook-Pro:~ jorgemaurocio$ python --version
Python 3.6.2 :: Anaconda custom (x86_64)
[Jorges-MacBook-Pro:~ jorgemaurocio$ python
Python 3.6.2 |Anaconda custom (x86_64)| (default, Jul 20 2017, 13:14:59)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 1 + 1
2
>>> a = 2
>>> b = 3
>>> c = a + b
>>> print(c)
5
>>>
```

Modo IDE

- Existen varios IDE (Interface Development Enviroment), los más populares son:
 - Spyder
 - Jupyter



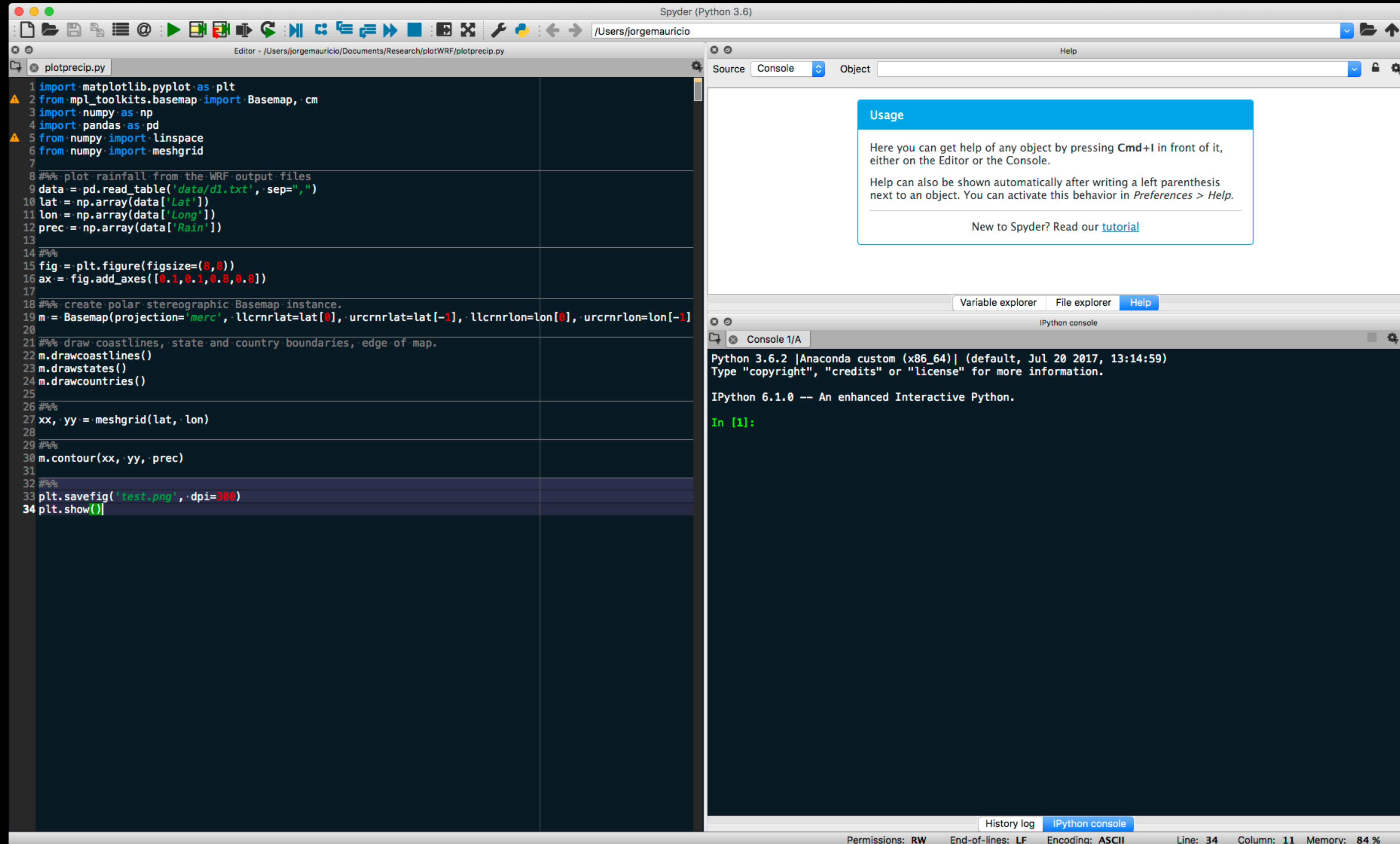
spyder



Spyder

- Entorno de desarrollo interactivo.
- Entorno de computación numérica que permite el uso de:
 - NumPy (álgebra lineal)
 - SciPy (procesamiento de señales e imágenes)
 - Matplotlib (trazado interactivo 2D / 3D)

IDE



Editor

The image shows the Spyder Python IDE interface. The main window is divided into three panes: a code editor on the left, a help panel on the top right, and an IPython console on the bottom right.

Code Editor: The file `plotprecip.py` is open. The code is as follows:

```
1 import matplotlib.pyplot as plt
2 from mpl_toolkits.basemap import Basemap, cm
3 import numpy as np
4 import pandas as pd
5 from numpy import linspace
6 from numpy import meshgrid
7
8 ### plot rainfall from the WRF output files
9 data = pd.read_table('data/d1.txt', sep=",")
10 lat = np.array(data['Lat'])
11 lon = np.array(data['Long'])
12 prec = np.array(data['Rain'])
13
14 ###
15 fig = plt.figure(figsize=(8,8))
16 ax = fig.add_axes([0.1,0.1,0.8,0.8])
17
18 ### create polar stereographic Basemap instance.
19 m = Basemap(projection='merc', llcrnrlat=lat[0], urcrnrlat=lat[-1], llcrnrlon=lon[0], urcrnrlon=lon[-1])
20
21 ### draw coastlines, state and country boundaries, edge of map.
22 m.drawcoastlines()
23 m.drawstates()
24 m.drawcountries()
25
26 ###
27 xx, yy = meshgrid(lat, lon)
28
29 ###
30 m.contour(xx, yy, prec)
31
32 ###
33 plt.savefig('test.png', dpi=300)
34 plt.show()
```

Help Panel: The 'Usage' tab is selected. It contains the following text:

Here you can get help of any object by pressing **Cmd+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our [tutorial](#)

IPython Console: The console shows the following output:

```
Python 3.6.2 |Anaconda custom (x86_64)| (default, Jul 20 2017, 13:14:59)
Type "copyright", "credits" or "license" for more information.

IPython 6.1.0 -- An enhanced Interactive Python.

In [1]:
```

The status bar at the bottom indicates: Permissions: RW, End-of-lines: LF, Encoding: ASCII, Line: 34, Column: 11, Memory: 84 %.

Visualizador de archivos

The image shows the Spyder Python IDE interface. The main window is titled "Spyder (Python 3.6)". It features a top toolbar with icons for file operations, running, and debugging. The central area is divided into three panes: a code editor on the left, a help panel on the right, and an IPython console at the bottom.

The code editor displays a file named `plotprecip.py` with the following Python code:

```
1 import matplotlib.pyplot as plt
2 from mpl_toolkits.basemap import Basemap, cm
3 import numpy as np
4 import pandas as pd
5 from numpy import linspace
6 from numpy import meshgrid
7
8 """ plot rainfall from the WRF output files
9 data = pd.read_table('data/d1.txt', sep=",")
10 lat = np.array(data['Lat'])
11 lon = np.array(data['Long'])
12 prec = np.array(data['Rain'])
13
14 """
15 fig = plt.figure(figsize=(8,8))
16 ax = fig.add_axes([0.1,0.1,0.8,0.8])
17
18 """ create polar stereographic Basemap instance.
19 m = Basemap(projection='merc', llcrnrlat=lat[0], urcrnrlat=lat[-1], llcrnrlon=lon[0], urcrnrlon=lon[-1])
20
21 """ draw coastlines, state and country boundaries, edge of map.
22 m.drawcoastlines()
23 m.drawstates()
24 m.drawcountries()
25
26 """
27 xx, yy = meshgrid(lat, lon)
28
29 """
30 m.contour(xx, yy, prec)
31
32 """
33 plt.savefig('test.png', dpi=300)
34 plt.show()
```

The help panel on the right is titled "Usage" and contains the following text:

Here you can get help of any object by pressing **Cmd+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our [tutorial](#)

The IPython console at the bottom shows the following output:

```
Python 3.6.2 [Anaconda custom (x86_64)] (default, Jul 20 2017, 13:14:59)
Type "copyright", "credits" or "license" for more information.

IPython 6.1.0 -- An enhanced Interactive Python.

In [1]:
```

The bottom status bar displays the following information:

- Permissions: RW
- End-of-lines: LF
- Encoding: ASCII
- Line: 34
- Column: 11
- Memory: 84 %

Consola

The image displays the Spyder Python IDE interface, which is divided into several panels. The main panel on the left is the code editor, showing a Python script named `plotprecip.py`. The script imports `matplotlib.pyplot`, `mpl_toolkits.basemap`, `numpy`, and `pandas`, and uses them to read data from a file, create a polar stereographic map, and plot rainfall data. The right panel is split into two sections. The top section is the 'Help' panel, which displays the 'Usage' of the IPython console, explaining how to get help for objects and how to activate automatic help. The bottom section is the 'IPython console', which shows the Python version (3.6.2) and the IPython version (6.1.0). The console also displays the prompt `In [1]:`, indicating that the first line of code has been executed. The bottom status bar shows the current line (34), column (11), and memory usage (84 %).

```
1 import matplotlib.pyplot as plt
2 from mpl_toolkits.basemap import Basemap, cm
3 import numpy as np
4 import pandas as pd
5 from numpy import linspace
6 from numpy import meshgrid
7
8 ### plot rainfall from the WRF output files
9 data = pd.read_table('data/d1.txt', sep=",")
10 lat = np.array(data['Lat'])
11 lon = np.array(data['Long'])
12 prec = np.array(data['Rain'])
13
14 ###
15 fig = plt.figure(figsize=(8,8))
16 ax = fig.add_axes([0.1,0.1,0.8,0.8])
17
18 ### create polar stereographic Basemap instance.
19 m = Basemap(projection='merc', llcrnrlat=lat[0], urcrnrlat=lat[-1], llcrnrlon=lon[0], urcrnrlon=lon[-1])
20
21 ### draw coastlines, state and country boundaries, edge of map.
22 m.drawcoastlines()
23 m.drawstates()
24 m.drawcountries()
25
26 ###
27 xx, yy = meshgrid(lat, lon)
28
29 ###
30 m.contour(xx, yy, prec)
31
32 ###
33 plt.savefig('test.png', dpi=300)
34 plt.show()
```

Usage

Here you can get help of any object by pressing **Cmd+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our [tutorial](#)

Python 3.6.2 |Anaconda custom (x86_64)| (default, Jul 20 2017, 13:14:59)
Type "copyright", "credits" or "license" for more information.

IPython 6.1.0 -- An enhanced Interactive Python.

In [1]:



History log IPython console

Permissions: RW End-of-lines: LF Encoding: ASCII Line: 34 Column: 11 Memory: 84 %












Jupyter Notebook

- Aplicación web
- Permite código en vivo, visualizaciones y texto
- Soporta más de 40 lenguajes, widgets interactivos y big data
- Los archivos se pueden compartir

IDE

 jupyter 1_Arreglos Numpy (autosaved)  Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

          Code 

```
In [3]: # librerias
import numpy as np
```

Crear Numpy Arrays

De una lista de python

Creamos el arreglo directamente de una lista o listas de python

```
In [4]: my_list = [1,2,3]
my_list
```

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

```
In [5]: np.array(my_list)
```


```
Out[5]: array([1, 2, 3])
```

```
In [6]: my_matrix = [[1,2,3],[4,5,6],[7,8,9]]
my_matrix
```


```
Out[6]: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Métodos

Barra de herramientas






 jupyter






1_Arreglos Numpy (autosaved)

 Logout


FileEditViewInsertCellKernelWidgetsHelp

TrustedPython 3





Code



In [3]:

```
# librerias
import numpy as np
```

Crear Numpy Arrays

De una lista de python

Creamos el arreglo directamente de una lista o listas de python

In [4]:

```
my_list = [1,2,3]
my_list
```

Out[4]:

```
[1, 2, 3]
```

In [5]:

```
np.array(my_list)
```

Out[5]:

```
array([1, 2, 3])
```

In [6]:

```
my_matrix = [[1,2,3],[4,5,6],[7,8,9]]
my_matrix
```

Out[6]:

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Métodos

Línea de código

The screenshot displays a Jupyter Notebook environment. At the top, the header shows "jupyter" with its logo, followed by the file name "1_Arreglos Numpy (autosaved)". On the right side of the header, there is a "Logout" button. Below the header is a menu bar with options: File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. To the right of the menu bar are two buttons: "Trusted" and "Python 3".

Below the menu bar is a toolbar containing icons for saving, adding new files, undo, redo, copy, paste, up/down arrows, previous/next cell navigation, a square icon, a refresh icon, a dropdown menu currently set to "Code", and a keyboard icon.

The main area of the notebook contains three code cells:

- Cell 1:** The prompt is "In [3]:". The code is:

```
# librerias  
import numpy as np
```
- Cell 2:** This cell contains markdown text.

Crear Numpy Arrays

De una lista de python

Creamos el arreglo directamente de una lista o listas de python
- Cell 3:** The prompt is "In [4]:". The code is:

```
my_list = [1,2,3]  
my_list
```

The output for this cell is "Out[4]: [1, 2, 3]".
- Cell 4:** The prompt is "In [5]:". The code is:

```
np.array(my_list)
```

The output for this cell is "Out[5]: array([1, 2, 3])".
- Cell 5:** The prompt is "In [6]:". The code is:

```
my_matrix = [[1,2,3],[4,5,6],[7,8,9]]  
my_matrix
```

The output for this cell is "Out[6]: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]".

At the bottom of the notebook, there is another heading:

Métodos

Línea de resultado

[illegible]

Markdown

The image shows a Jupyter Notebook interface with a dark theme. At the top, there's a header bar with the Jupyter logo, the file name "1_Arreglos Numpy (autosaved)", and a "Logout" button. Below the header is a menu bar with options like File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. A toolbar contains icons for saving, adding new files, undo, redo, and running code. The main area displays three code cells. The first cell has the prompt "In [3]:" followed by "# librerias" and "import numpy as np". The second cell has the prompt "In [4]:" followed by "my_list = [1,2,3]" and "my_list". Its output, "Out[4]:", is "[1, 2, 3]". The third cell has the prompt "In [5]:" followed by "np.array(my_list)". Its output, "Out[5]:", is "array([1, 2, 3])". Below this, another code cell has the prompt "In [6]:" followed by "my_matrix = [[1,2,3],[4,5,6],[7,8,9]]" and "my_matrix". Its output, "Out[6]:", is "[[1, 2, 3], [4, 5, 6], [7, 8, 9]]". At the bottom left, the word "Métodos" is partially visible.

Sintaxis Markdown

Título

Subtítulo

Título normal

**** Negrita ****

* Lista

* ` Código `

* *_ Itálica _*

[link](http://github.com/jorgemauricio)

Título

Subtítulo

Título normal

Negrita

- Lista
- Código
- *Itálica*

[link](#)

Sintaxis

Indentación

El contenido de los bloques de código es delimitado mediante espacios o tabuladores, conocidos como **indentación**.

C (Indentación opcional)

```
int numeroMayor (int x, int y)
{
    if (x == y){
        printf("Los dos valores son iguales");
    }else if (x > y){
        printf("El primer valor es mayor");
    }else (x > y){
        printf("El primer valor es mayor");
    }
}
```

C (Indentación opcional)

```
int numeroMayor (int x, int y)
{
    if (x == y){
        printf("Los dos valores son iguales");
    }else if (x > y){
        printf("El primer valor es mayor");
    }else (x > y){
        printf("El primer valor es mayor");
    }
}
```


Python (Indentación obligatoria)

```
def numeroMayor(x,y):  
    if x == y:  
        print("Los dos valores son iguales")  
    elif x > y:  
        print("El primer valor es mayor")  
    else:  
        print("El segundo valor es mayor")
```

Python (Indentación obligatoria)

```
def numeroMayor(x,y):  
<--> if x == y:  
<-----> print("Los dos valores son iguales")  
<--> elif x > y:  
<-----> print("El primer valor es mayor")  
<--> else:  
<-----> print("El segundo valor es mayor")
```

Comentarios

Existen dos formas. La primera y más apropiada para comentarios largos es utilizando la notación

''' comentario ''', tres apóstrofes de apertura y tres de cierre.

La segunda notación utiliza el símbolo **#**, y se extienden hasta el final de la línea.

```
'''  
Comentario multilínea  
'''
```

```
print("Hola Mundo") # Comentario en línea
```

Imprimir variables

```
a = 2
print(a)
# imprime 2
print("Hola mundo")
# imprime Hola mundo
a = 2
b = 3
print("Multiplicar {} x {} = {}".format(a,b,a*b))
# imprime Multiplicar 2 x 3 = 6
```

Variables

Las variables se definen de forma dinámica, lo que significa que no se tiene que especificar cuál es su tipo de antemano y puede tomar distintos valores en otro módulo, función o proceso, incluso de un tipo diferente al que tenía previamente.

Se utiliza el símbolo “=” para asignar valores

```
x = 1
```

```
x = "Texto"
```

```
# Esto es posible por que los tipos son asignados dinámicamente
```

Operadores lógicos

!	not
	or
&	and

Tipos de datos

Tipo	Clase	Notas	Ejemplo
str	Cadena	Inmutable	Cadena'
unicode	Cadena	Versión Unicode de str	u'Cadena'
list	Secuencia	Mutable, puede contener objetos de diversos tipos	[4.0, 'Cadena', True]
tuple	Secuencia	Inmutable, puede contener objetos de diversos tipos	(4.0, 'Cadena', True)
set	Conjunto	Mutable, sin orden, no contiene duplicados	set([4.0, 'Cadena', True])
frozenset	Conjunto	Inmutable, sin orden, no contiene duplicados	frozenset([4.0, 'Cadena', True])
dict	Mapping	Grupo de pares clave:valor	{'key1': 1.0, 'key2': False}
int	Número entero	Precisión fija, convertido en long en caso de overflow.	42
long	Número entero	Precisión arbitraria	42L ó 456966786151987643L
float	Número decimal	Coma flotante de doble precisión	3.1415927
complex	Número complejo	Parte real y parte imaginaria j.	(4.5 + 3j)
bool	Booleano	Valor booleano verdadero o falso	True o False

Lista

Para declarar una lista se usan los corchetes “[]”.

Para acceder a los elementos de una lista se utiliza un índice entero (empezando por "0", no por "1"). Se pueden utilizar índices negativos para acceder elementos a partir del final.

Las listas se caracterizan por ser mutables, es decir, se puede cambiar su contenido en tiempo de ejecución.

```
>>>> lista = ["abc", 42, 3.1416]

>>>> lista[0] # Acceder a un elemento por su índice

'abc'

>>>> lista[-1] # Acceder a un elemento usando un índice negativo

3.1416

>>>> lista.append(True) # Añadir un elemento al final de la lista

>>>> lista

['abc', 42, 3.1416, True]

>>>> del lista[3] # Borra el elemento número 3 de la lista

>>>> lista[0] = "xyz" # Re-asignar el valor del primer elemento de la lista

>>>> lista[0:2] # Mostrar los elementos de la lista del índice 0 al 2

["xyz", 42]

>>>> lista_anidada = [lista, [True, 42L]]

>>>> lista_anidada

[['xyz', 42, 3.1416], [True, 42L]]

>>>> lista_anidada[1][0] # Acceder a un elemento de una lista dentro de otra lista
```

Diccionarios

Para declarar un diccionario se usan las llaves “{ }”. Contienen elementos separados por comas, donde cada elemento está formado por un par **clave : valor** (el símbolo : separa la clave de su valor correspondiente).

Los diccionarios son mutables, es decir, se puede cambiar el contenido de un valor en tiempo de ejecución.

En cambio, las claves de un diccionario deben ser inmutables. Esto quiere decir, por ejemplo, que no podremos usar ni listas ni diccionarios como claves.

El valor asociado a una clave puede ser de cualquier tipo de dato, incluso un diccionario.

```
>>>> diccionario = {"cadena": "abc", "numero": 42, "lista": [True, 42L]}
# Diccionario que tiene diferentes valores por cada clave
>>>> diccionario["cadena"] # Accede a su valor usando clave
'abc'
>>>> diccionario["lista"][0] # Acceder a un elemento de una lista dentro de un valor
True
>>>> diccionario["cadena"] = "xyz" # Re-asignar el valor de una clave
>>>> diccionario["cadena"]
"xyz"
>>>> diccionario["decimal"] = 3.1416 # Insertar un nuevo elemento clave:valor
>>>> diccionario["decimal"]
3.1416
```

Condicionales

Una sentencia condicional **if** ejecuta su bloque de código interno sólo si se cumple cierta condición. Se define usando la palabra clave `if` seguida de la condición, y el bloque de código.

Condiciones adicionales, si las hay, se introducen usando **elif** seguida de la condición y su bloque de código. Todas las condiciones se evalúan secuencialmente hasta encontrar la primera que sea verdadera, y su bloque de código asociado es el único que se ejecuta.

Opcionalmente, puede haber un bloque final (la palabra clave es **else** seguida de un bloque de código) que se ejecuta sólo cuando todas las condiciones fueron falsas.

```
>>>> verdadero = True
>>>> if verdadero: # No es necesario poner "verdadero == True"
...     print("Verdadero")
... else:
...     print("Falso")
Verdadero
```

```
>>>> lenguaje = "Python"

>>>> if lenguaje == "C": # Lenguaje no es "C", por lo que este bloque se obviará y evaluará la siguiente condición
...     print("Lenguaje de programación: C")
...     elif lenguaje == "Python": # se pueden añadir tantos bloques "elif" como se quiera
...         print("Lenguaje de programación: Python")
...     else: # en caso de que ninguna de las anteriores condiciones fuera cierta, se ejecutaría este bloque
...         print("Lenguaje de programación: Indefinido")
...
...
```

Lenguaje de programación: Python


```
>>> if verdadero and lenguaje == "Python": # Uso de "and" para comprobar que ambas condiciones son verdaderas
...     print("Verdadero y Lenguaje de programación: Python")
...
Verdadero y Lenguaje de programación: Python
```

Ciclos

For

Es similar a **foreach** en otros lenguajes.

Recorre un objeto iterable, como una lista, una tupla o un generador, y por cada elemento del iterable ejecuta el bloque de código interno.

Se define con la palabra clave **for** seguida de un nombre de variable, seguido de **in**, seguido del iterable, y finalmente el bloque de código interno.

En cada iteración, el elemento siguiente del iterable se asigna al nombre de variable especificado

```
>>>> lista = ["a", "b", "c"]
>>>> for i in lista: # iteramos sobre una lista
...     print(i)
...
a
b
c
```

```
>>>> cadena = "abcdef"
>>>> for i in cadena: # iteramos sobre una cadena
...     print(i)
...
a
b
c
d
e
f
```

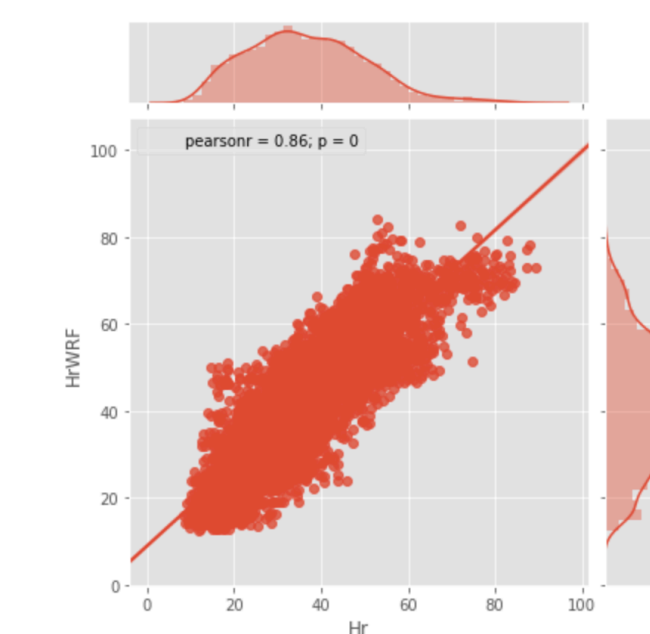
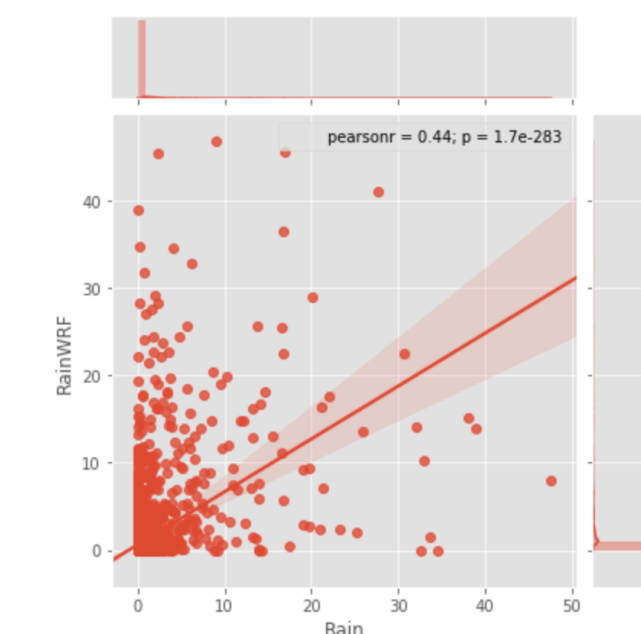
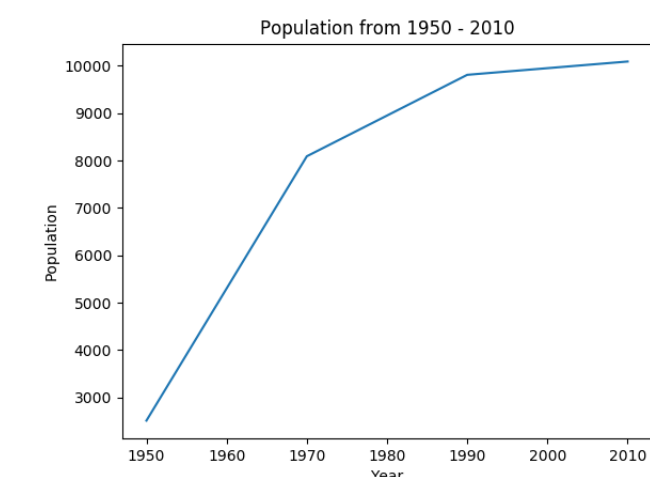
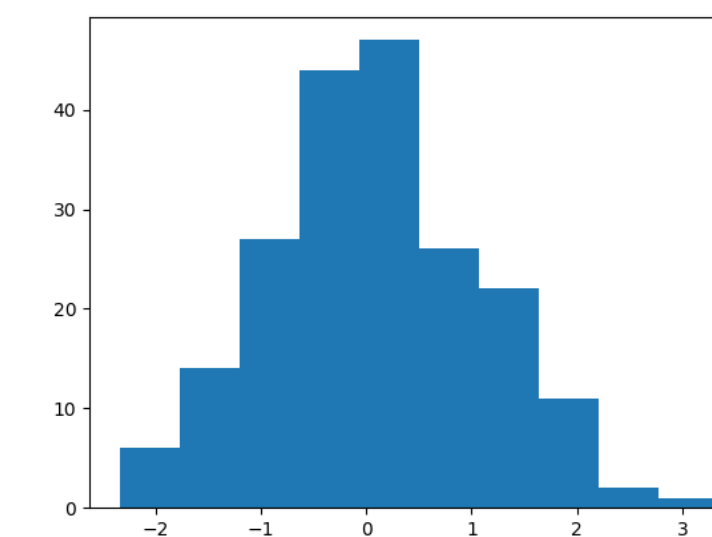
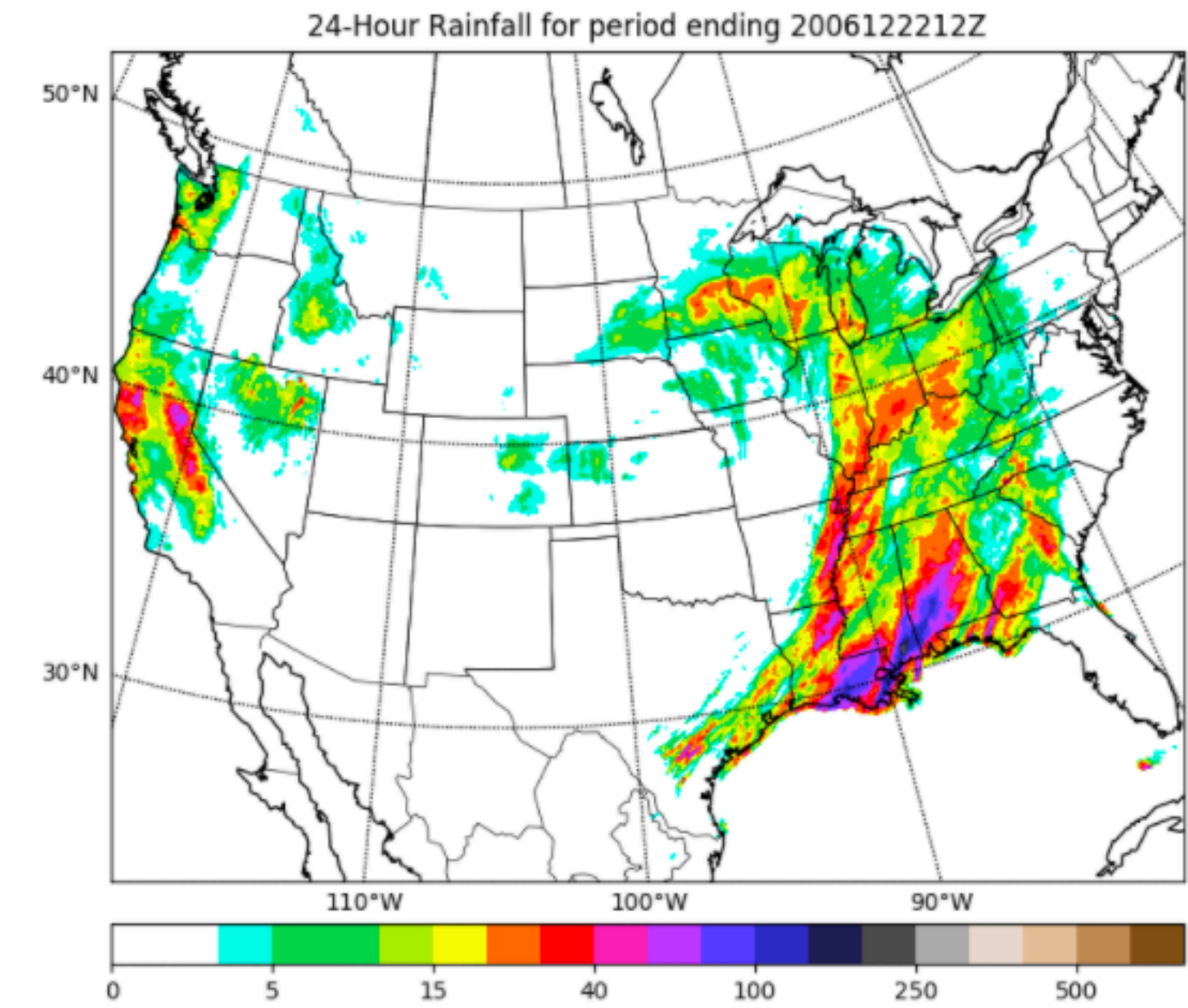
While

Evalúa una condición y, si es verdadera, ejecuta el bloque de código interno. Continúa evaluando y ejecutando mientras la condición sea verdadera. Se define con la palabra clave **while** seguida de la **condición**, y a continuación el bloque de código interno

```
>>>> numero = 0
>>>> while numero < 5:
...     print(numero)
...     numero += 1 # un buen programador modificará las variables
...                 # de control al finalizar el ciclo while
...
0
1
2
3
4
```

Módulos

- Existen muchas propiedades que se pueden agregar al lenguaje importando módulos, que son "minicódigos" (la mayoría escritos también en Python) que proveen de ciertas funciones y clases para realizar determinadas tareas.
- Un ejemplo es el módulo Basemap, que nos permite crear mapas.
- Otro ejemplo es el módulo os, que provee acceso a muchas funciones del sistema operativo. Los módulos se agregan a los códigos escribiendo **import** seguida del nombre del módulo que queremos usar.




```
>>>> import os # módulo que provee funciones del sistema operativo
>>>> os.name # devuelve el nombre del sistema operativo
'posix'
>>>> os.mkdir("tmp/ejemplo") # crea un directorio en la ruta especificada
>>>> import time # módulo para trabajar con fechas y horas
.... time.strftime("%Y-%m-%d %H:%M:%S")
# dándole un cierto formato, devuelve la fecha y/u hora actual
'2017-09-15 10:52:01'
```

ZEN OF PYTHON

Keep in mind Python's philosophy as you code

```
>>> import this
```

The Zen of Python, by Tim Peters

Beautiful is better than ugly.

Explicit is better than implicit.

Simple is better than complex.

Complex is better than complicated.

Flat is better than nested.

Sparse is better than dense.

Readability counts.

Special cases aren't special enough to break the rules.

Although practicality beats purity.

Errors should never pass silently.

Unless explicitly silenced.

In the face of ambiguity, refuse the temptation to guess

...

Contacto

Dr. Victor M. Rodríguez M.
rodriguez.victor@inifap.gob.mx

IIS Jorge Ernesto Mauricio Ruvalcaba
jorge.ernesto.mauricio@gmail.com

github.com/jorgemauricio

Muchas gracias!