ParTUXza II
Python3 para las Ciencias Naturales

A. Schrapffer[1]

[1]Centro de Investigaciones del Mar y la Atmósfera (CIMA), UBA, CONICET, Buenos Aires, Argentina

6 de junio de 2018

Resumen

Presentar el lenguaje de programación Python3 y dar conocimientos básicos para empezar a manejarlo en el ámbito de las Ciencias Naturales.

Índice

1. Librerías a bajar **3**

2. Introducción **3**

3. Funcionamento **4**

4. Usar Python **5**
- 4.1. Antes de empezar . . . 5
- 4.2. Librarias . . . 6
- 4.3. if and loops . . . 6
- 4.4. Tipos de objetos - básicos . . . 8
  - 4.4.1. String : carácteres . . . 8
  - 4.4.2. Tipos de números . . . 8
  - 4.4.3. Conversión variables . . . 9
  - 4.4.4. Listas . . . 9
  - 4.4.5. Tuples . . . 10
  - 4.4.6. Diccionarios . . . 10
  - 4.4.7. Python y el tiempo . . . 11
- 4.5. Librarias extras . . . 11
  - 4.5.1. Numpy . . . 11
  - 4.5.2. netCDF . . . 12
  - 4.5.3. Otros . . . 13
- 4.6. Uso más avanzado . . . 13
  - 4.6.1. Funciones . . . 13
  - 4.6.2. Llamar a una función desde el terminal . . . 14

5. Graficar con Python **16**
- 5.1. Gráficos básicos . . . 16
  - 5.1.1. 1D . . . 17
  - 5.1.2. Subplots . . . 18
  - 5.1.3. 2D . . . 19
- 5.2. Gráficos sobre mapas . . . 20

6. Conclusión **22**

7. Exercicios **22**
- 7.1. Fibonacci . . . 22
- 7.2. Practicar con NetCDF . . . 22

8. Correcciones **22**
- 8.1. Fibonacci . . . 22
- 8.2. netCDF . . . 23

1. Librerías a bajar

Librerías python3
python3 python3-scipy python3-numpy

Librerías NetCDF
netcdf-bin netcdf-dbg libnetcdf-dev netcdf-doc libnetcdff-dbg libnetcdff-dev libnetcdff-doc libhdf5-dev

Librerías Matplotlib
python3-matplotlib python3-matplotlib-data python3-matplotlib-doc python3-mpltoolkits.basemap python3-mpltoolkits.basemap-data

Spyder
spyder3

2. Introducción

Python fue creado por Guido van Rossum en 1989, su nombre tiene por origin el '*Monty Python's Flying Circus*'. Es un lenguaje de programación orientado objeto, quiere decir que favorece la manipulación de estructuras definidas (ej. serie de caracteres, números enteros..) vía varias herramientas. Python está también en libre uso.

Es un lenguaje de alto nivel, lo que permite abstraerse de los problemas de CPU y de memoria, facilita la parte de programación y la lectura del lenguaje pero puede penalizar la eficiencia y la velocidad.

La última versión es la tercera : python3.x, la cual no es compatible con python2.x. Esta clase se basea en python3.x.

Porque usar Python ?

- **Por el diseño del lenguaje**:
  su simplicidad facilita la implementación de ideas
- **Por la facilidad de uso**:
  es un lenguaje interpretado, no necesita ser compilado solo basta ejecutar el código
- **Por la facilidad de lectura**:
  para poder compartirlo con otras personas, con la comunidad científica
- **Por la alta compatibilidad**:
  puede funcionar con otros tipos de lenguajes de programación (C con cython, Fortran con f2py)
- **Por la estructura de los datos**:
  numerosos tipos de objetos disponibles
- **Por la grande comunidad que lo usa**:
  mucha información en internet, libros, librerias disponibles y actualizadas

3. Funcionamento

Como visto antes, Python es un lenguaje dinámico, no necesita ser compilado antes de usarlo.
Hay varias maneras para usarlo :

- Trabajar en la terminal : ejecutar los commandos en el terminal de Linux

```
linux@partuxza:~\$ python3
>>> $ mycommand
```

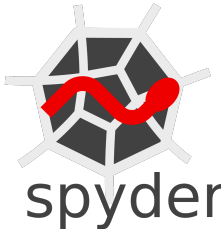
- Ejecutar un script : juntar los comandos en un script (formato script.py)
  - se puede lanzar desde el terminal eligiendo la versión de python que queremos

    ```
    linux@partuxza:~\$ python3 script.py
    ```

  - una buena práctica consiste en agregar en el script la dirección del interpretador que queremos al principio del script con

    ```
    #!/usr/bin/env python3
    ```

- utilizar un entorno de desarollo integrado para lanzar comandos o generar script, por ejemplo con spyder o jupyter, permite ver a las variables, tener una versión del terminal, se puede correr el script por partes ..



4. Usar Python

4.1. Antes de empezar

A saber :

- # es el carácter para comentar

```
# Comentario
```

- Se puede también comentar sobre varias lineas :

```
"""
Muchos Comentarios
"""
```

- después de una condición `if` o de una loop, hay que respectar un espacio, se termina cuando ya no se respecta (no tiene simbolo de salida del `if` o de la loop)

```
>>> if a==1:
...     Instrucciones del if
>>> Instrucciones fuera del if
```

- \ es el caracter para continuación de la linea

```
>>> a = 2 + 3 + 4 \
...     + 5 + 6
```

- El lenguaje es sensible a las mayúsculas y minúsculas
- Se importan las librerías vía el comando `import` (ver próxima parte para más detalles)
- Empieza a contar a 0,
  el primer elemento de una lista es recordado como 0
- La extensión de los script es '.py'
- Para acceder a la posición `i` de una variable `var` se usa

```
>>> var[i]
```

- Para usar el método `mth` de un objeto `obj`, se usa

```
>>> obj.mth()
```

- Otras funciones que no son espécificas de un objeto deben ser llamada de la manera siguiente `fct(a, b, param. opcionales)`
- Si la función o el método tienen en salida uno o varios objetos, se puede guardar los de la manera siguiente :

```
# Si solo hay una variable en salida
c = fct(param, a, b)
#Â Si hay mas variables, por ejemplo 3
c, d, e = fct(param, a, b)
```

- Se puede poner varios comandos en una misma linea, poniendo ; entre cada comándo (para comando que no necesitan ser sobre varias lineas (cf `if`, o loop más tarde)

- Para imprimir elementos en la pantalla se puede usar la función `print`

```
>>> a=4
>>> # Imprimir una variable
>>> print(a)
 4
>>> print("Hola")
 "Hola"
>>> # Imprimir varias variable
>>> print("Hola",a)
 Hola 4
```

- Python3 y sus librerias suelen ser bien documentadas, para acceder a la ayuda se puede usar, x siendo cualquier objeto, método, función etc.

```
>>> help(x)
```

4.2. Librarias

Las librarias adicionales tienen que ser importada antes de usar las funciones que las componen, existen varias maneras para importar las librerias :

```
# 1- Importar una libraria
import library
# 2- Importar una libraria dandole otro nombre
import library as lib
# 3- Importar una funcion de una libraria
from library import funk (as)

# Se puede despues, segun como fue hecha la importacion, usar una funcion de la libreria
#    de las formas siguientes :
library.funk() # caso 1
lib.funk()     # caso 2
funk()         # caso 3
```

Se puede tambien importar a partir de un otro archivo, si está en el mismo directorio solo dando el nombre del archivo sin .py es suficiente

```
# Importar funciones a partir de un script presente en el mismo directorio
# Si el archivo se llama file.py
import file
```

En el caso en el cual el archivo está en un otro repertorio

```
import sys
sys.path.append("/direccion/del/archivo")
import file
```

4.3. if and loops

Condiciones

Existe un tipo de objeto en python que son los booleans, este tipo de objeto puede tomar el valor `True` o `False`. Las condiciones permiten de devolver un boolean, por ejemplo :

```
x == y                  # Da True si x = y
x != y                  # ... x no es igual a y
x > y                   # ... x es mas grande que y
x < y                   # ... x es menos que y
x >= y                  # ... x mas grande o igual y
x <= y                  # ... x menos o igual y
```

También se pueden usar otros operadores como `and`, `or` etc.

if

Para hacer una condición `if`, la estructura es la siguiente, cuidado a no olvidar poner los dos puntos después de la condición.

```
>>> i=3
>>> if i == 2:
...     print("dos")
... elif i ==3:
...     print("tres")
... else:
...     print("otra cosa")
 tres
```

loop

Las loop se hacen vía un elemento tomado entre varios elementos, de una lista por ejemplo. Se puede usar el commando `break` para salir del loop. Básicamente puede ser una liste de números, por eso antes de empezar, vemos una herramienta útil, el `range`:

```
>>> # range(a) es la lista de numeros enteros entre 0 y a (excluido)
>>> print(range(4))
 [0, 1, 2, 3]
>>> # range(a,b) es la lista de numeros enteros entre a y b
>>> print(range(2,5))
 [2,3,4]
>>> # range(a, b, c) es la lista de numeros enteros entre a y b cada c numeros
>>> print(range(0,10,2))
 [0, 2, 4, 6, 8]
```

Después para hacer una loop, se usa lo siguiente :

```
>>> for i in range(2):
...     print(i)
 0
 1
```

También se puede usar una list en general :

```
>>> for i in ["hello",4, 3.5]:
...     print(i)
 hello
 4
 3.5
```

Las loops también se pueden hacer a través de una condición :

```
>>> a = ["azul", "rojo", "amarillo"]
>>> # Buscando al elemento de a conteniendo amarillo
>>> i = 0
... while a[i] != "amarillo":
...     i=i+1
...     if i == len(a):
...         i=-1
...         break # para salir del loop
>>> # Ejemplo de uso del if
>>> if i<len(a):
...     print("La lista no incluye el color amarillo)
... else:
...     print("La lista incluye el color amarillo en posicion",i)
 La lista incluye el color amarillo en posicion 2
```

4.4. Tipos de objetos - básicos

Existen un par de tipos de objetos métodos y funciones ya existentes en python3, después existen una infinidad de tipos de objetos relacionados con librerias importadas, hasta se puede crear nuevas clases de objetos (cf. *https://docs.python.org/3/tutorial/classes.html*)

4.4.1. String : carácteres

Los string son cadenas de carácteres, se pueden definir de tres maneras:

```
>>> a = "hola"
>>> b = 'linux'
>>> c = """ valido sobre multiplas lineas """
>>> d = ''' este tambien '''
```

Las funciones importantes son las siguientes aplicadas al string `str`:

- `str.upper()` and `str.lower()` : para poner todo en mayúscula / minúscula
- `str.count()` : para contar el número de caracteres
- `str.find(seq[, a])` : para encontrar el lugar donde se encuentra la secuencia seq dentro del string partir del índice a (opcional)
- `str.replace(oldseq, newseq[, max])` : reemplaza dentro del string las secuencias oldseq por newseq, limitado a max ocurrencias si dado (opcional)

Existen muchas más funciones más espécificas : *https://docs.python.org/2/library/string.html*

4.4.2. Tipos de números

Existen diferentes tipos de números :

- `int` : número entero entre -2 147 483 648 y 2 147 483 647, ej : `a=2`
- `long` : número entero largo, >2 147 483 647 o <-2 147 483 648
- `float` : número flotante, ej : `a=2.3`
- `complex` : número complejo, ej: `z=2+3j` o `complex(2,3)`

Las funciones pueden variar según el tipo de número :

Operaciones clásicas para float, int, complex

- `x+y` : adición
- `x-y` : sustracción
- `x*y` : multiplicación
- `x/y` : división
- `x//y` : parte entera de la división
- `x%y` : resto de la división
- `x**y` : x al poder y
- `abs(x.)` : valor absoluto de x
- `pow(x,y)` : x al poder y

Operaciones para complex

- `z.real` : parte real
- `z.imag` : parte imaginaria
- `z.conjugate` : conjugado

4.4.3. Conversión variables

Existen funciones para convertir variables de una forma a una otra, ejemplo para un número x :

- `int(x)` : convierte x en número entero
- `long(x)` : convierte x en número entero largo
- `float(x)` : convierte x en número flotante
- `str(x)` : convierte x en cadena de carćter

4.4.4. Listas

Una lista en python, es una variable que puede contener varias variables, de cualquier tipo, los objetos de la lista están ordenados, se puede referir a ellos por sus índices.

Hay diferentes manera de crear un lista :

```
>>> a=[] #crear una lista vacia
>>> b=[1,2,3] #crear una lista ya llena
>>> c=["a", 2, 2.3, [1,2,3]] #lista puede ser llenada con varios tipos de objetos
```

También se puede crear listas de más de una dimensión, pero para facilitarles la vida, les recomiendo numpy para este tipo de lista multidimensional.

Funciones y Operaciones clásicas

- `L.append(4)` : agregar 4 a la lista
- `L1+L2` : agrega L2 después de L1

```
>>> L1=[1,2]; L2+[3,4];
>>> print(L1+L2)
 [1,2,3,4]
```

- `del L[indice]` : suprimir el segundo elemento de L (recuerden empieza por 0), acá se trata del índice

```
>>> del L[1]; print(L)
 [1,3,4]
```

- `L.remove(element)` : suprimir un elemento de una lista, refiriendose a él por su valor

```
>>> L=["a", "b", "c"]; print (L.remove("b"))
 [a, c]
```

- `L.reverse()` : invertir la lista L
- `len(L)` : conocer el número de elementos que contiene la lista L
- `L.count(c)` : contar cuentas veces el elemento c aparece en L
- `L.index(c[, i])` : conocer el index en la lista L del primer elemento c, a partir del indexo i (opcional)
- Para manipular una lista:

```
>>> L = ["a","b","c","d","e"]
>>> print(L[0]) # Primer elemento de L
 a
>>> print(L[-1]) # Ultimo elemento de L
 e
>>> print(L[2:4]) # Elementos entre los indices 2 y 4 (4 excluido)
 ['c', 'd']
>>> print(L[:4]) # Elementos hasta el indice 4 (excluido)
 ['a', 'b', 'c', 'd']
>>> print(L[2:]) # Elemento entre el indice 2 y el ultimo
 ['c', 'd', 'e']
>>> print(L[:]) # Todos los elementos de L
 ['a', 'b', 'c', 'd', 'e']
```

4.4.5. Tuples

Un `tuple` es un objeto parecido a una lista pero que no puede ser cambiado. Se define usando paréntesis

```
>>> tuple = ('a', 'b', 'c', 'd', 'e')
>>> print(tuple[1])
 'b'
>>> tuple[1]=3
 TypeError: 'tuple' object does not support item assignment
```

4.4.6. Diccionarios

Es parecido a una lista, pero los elementos son referenciado no con un índice sino con una otra referencia, que puede ser un número, una cadena de carácter...

```
>>> dico = {} # Creacion de un diccionario vacio
>>> # Agregar un elemento con nombre llave
>>> dico['banana'] = 3
>>> dico['mango'] = 2
>>> dico['manzana'] = 1
```

```
>>> dico['unidad'] = "kilo" # el elemento puede ser de otro tipo
>>> print dico
 {'mango': 2, 'banana': 3, 'manzana': 1, 'unidad': 'kilo'}
```

Funciones clásicas
Las funciones clásicas para un diccionario `dico`

- `del dico[key]` : suprimir el elemento de dico referenciado por key
- `dico.keys()` : acceder a la lista de las referencias de dico
- `dico.values()` : acceder a la lista de los valores de dico
- `len(dico)` : da el numero de elementos de dico
- `key in dico` : da True si key es una referencia de dico False en el caso contrario
- `dico.items()` : otra manera de visualizar un diccionario, devuelve una lista de Tuples
- `dico2 = dico.copy()` : copia el diccionario dico en dico2
- `dico.update(dico2)` : permite agregar a dico las nuevas referencias de dico2 y sus valores, pero tambien actualizar los valores de las referencias ya existentes

4.4.7. Python y el tiempo

Para manejar el tiempo las librerias más comunes son `time` y `datetime`.

La libreria `time` para manejar el tiempo.

```
>>> import time
>>>
>>> a=time.time() # seconds desde el 01/01/1970 a las 00:00:00
>>> b=time.time()
>>> b-a # Numero de segundos entre el momento a y b
>>> time.localtime() # tiempo presente en formato localtime
>>> print(time.localtime())
 time.struct_time(tm_year=2018, tm_mon=6, tm_mday=5, tm_hour=17, tm_min=52, tm_sec=12, tm_wday=1, t
>>>
>>> time.localtime(a) # conversion de a al formato localtime
>>> time.sleep(4) # Pausa durante 4 segundos
```

La libreria `datetime` para manejar las fechas, horas ...
La clase `date` adentro de `datetime` permite manejar fechas y la clase `datetime` adentro de `datetime` para manejar fechas con horas, min, sec ...

```
import datetime
>>> import datetime
>>> date = datetime.date(2010, 12, 25)
>>> print(date)
2010-12-25
>>> datetime.date.today()
 datetime.date(2018, 6, 5)
>>> a = datetime.datetime.now()
 datetime.datetime(2018, 6, 5, 17, 53, 51, 179074)
>>> b = datetime.datetime.now()
datetime.datetime(2018, 6, 5, 17, 56, 52, 18624)
```

```
>>> # Posibilidad de hacer operaciones entre dos momentos
>>> (a-b).seconds
 182
```

4.5. Librarias extras

4.5.1. Numpy

Para la gestión de tabla, lo que se recomienda es Numpy.

```
# Libraria Numpy
import numpy
# Libraria numpy para datos enmascarados
import numpy.ma as ma

np.zeros((x,y)) # Matriz llena de 0, de tamano x*y
np.ones((x,y)) # Matriz llena de 1, de tamano x*y
np.full((5,2),7) # Matriz constante, de valor 7 y tamano 5*2
np.eye(2) # Matriz identidad de tamano 2*2
np.random.random((2,2)) # Matriz aleatoria de tamano 2*2
np.arange(20,30[, 3]) # lista empezando por 20 y terminando por 29, cada 3 (opcional)
np.linspace(30,40, num=20) # Lista de 20 numeros entre 30 y 40 (incluidos), regularmente espaciados
N.shape() # conocer las dimensiones del elemento de numpy N
np.array(list) # convertir una list de 1 o mas dimensiones en array numpy
```

También existen todas las operaciones posibles entre matrices de varias dimensiones y la posibilidad de buscar los elementos que respectan una condición etc.

La libreria `ma` en numpy permite trabajar con array enmascarados. Lo que permite enmascarar parte de los datos para efectuar operaciones sin tomarlos en cuenta con por ejemplo `mx` un array enmascarado, `mx.ma.mean()` calcula el promedio de `mx` sin tomar en cuenta los datos enmascarados.
Con certeza, la libreria tiene mucho más funciones.

4.5.2. netCDF

Network Common Data Form (netCDF) es una interfacie desarollada por UNIDATA para guardar datos en forma de tablas multidimensionales. Las librarias de NetCDF son libres y gratis. Es muy común para la gestión de datos y las salidas de modelos en meteorología y en oceanografía pero también en otras areas como la geologia.

Los documentos netCDf también contienen las informaciones descriptivas de los datos que contienen.

Existe una libreria para netCDF en python3.

Leer NetCDF

```
from netCDF4 import Dataset as NetCDFFile
from netCDF4 import num2date

# Abrir el documento netCDF ubicado en ncdir
ncfile = NetCDFFile(ncdir, 'r')
# Lista de las variables en ncfile
ncfile.variables
# Abrir la variable "var" en ncfile, en formato netCDF (con informacion)
ovar = ncfile.variables["var"]
# Lista de los atributos de la variable
```

```
ovar.ncattrs()
# Ver al valor del atributo attrn de la variable ovar
ovar.attrn

# Si ovar tiene dimension 2, se puede obtener la tabla de datos
# Pero xvar contiene los datos pero no contiene los atributos de la variable
xvar = ovar[:,:]
```

Para convertir el tiempo en un formato `datetime`, por ejemplo si la variable de tiempo se llama time :

```
from netCDF4 import Dataset as NetCDFFile
from netCDF4 import num2date

ncfile = NetCDFFile(ncdir, 'r')
tiempo = ncfile.variables["time"][:]
dtime2 = num2date(tiempo[:], tiempo.units)
```

Crear/modificar un NetCDF

```
# Crear un nuevo archivo netCDF en ncdir
ncnew = NetCDFFile(ncdir, 'w')

# Creacion de las dimensiones
ncnew.createDimension("lon", dimx)
ncnew.createDimension("lat", dimy)
ncnew.createDimension('time', dimt)

# Creacion de las variables
newvar = ncnew.createVariable(varname, type, ovar.dimensions)
# Poner el valor attrv al atributo attrn de la nueva variable
newvar.setncattr(attrn, attrv)
ncnew.sync()
ncnew.close()
```

4.5.3. Otros

Otras librerias pueden ser interesantes para otros usos :

- **panda** : herramientas para gestión de datos y herramientas de análisis de datos
- **xarray** : similar a Panda pero para gestión de datos multidimensionales
- **libraria scipy** : muchas herramientas científicas (optimización, algebra linear, integración, interpolación ...)
- **seaborn** : para la visualización de datos estadísticos
- y mucho más para dominios muy variados : visualización de todo tipo de datos (2D/3D/Animación..), Machine learning, estadísticas, Data mining..

4.6. Uso más avanzado

4.6.1. Funciones

Como dicho al principio, el mayor problema de python es la gestión de la memoria. En un uso 'básico' se guardan todas las variables y para hacer espacio se necesita suprimirlas una a una. Por eso cuando se usa Python, siempre hay que estar pensando en minimizar el uso de la memoria.

La solución a este problema es usar fonciones, las funciones toman en entrada ciertos objetos, y en salidas devuelven ciertos objetos, pero todos los otros objetos creado y utilizado entre tiempo no se guardan em memoria.

Crear una función
Una función se declara con `def` seguido del nombre de la función y entre paréntesis de los parámetros de la función, es posible no imponer paramétros y dar valores por defecto a los parámetros para que si no esten definidos tomen este valor por defecto.

El `return` al final es opcional y sirve para recuperar uno o varios objetos. Además si uno de los objetos en 'entrada' son modificados en la función, se guarda esta modificación.

Otra vez hay que recordar que se respectan unos espacios de diferencias entre el def y el contenido de la dunción, como con el if y los loops.

```
>>> def nombre_funcion(parametros, a, b, c=2):
...     Contenido
...     return d, e
```

Usar funciones

Función simple :

```
>>> def saludar():
...     print("Hola")
>>> saludar()
 Hola
```

Función con parámetros

```
>>> def saludar(name = ""):
...     print("Hola "+str)
>>> saludar(Pablito)
 Hola Pablito
>>> saludar()
 Hola
```

Usar el return :

```
>>> def aggregar2(i):
...     j = i+2
...     return j
>>> aggregar2(4)
 6
>>> out = agregar2(4)
>>> print(out)
 6
```

4.6.2. Llamar a una función desde el terminal

Para llamar a una de las funciones de python desde el terminal se puede usar **OptionParser**.

Por ejemplo si mi script contiene dos funciones, `funcion1` y `funcion2`, que toman como parámetros : `funcion1(indir, outdir)` y `funcion2(indir, outdir, var)`, agrego en el script :

```
# Al principio
from optparse import OptionParser
```

```
...
# Despues de definir las funciones
operation = ["funcion1", "funcion2"]

parser = OptionParser
parser.add_option("-o", "--operation", type='choice', dest="operation", choices=operations, help="
parser.add_option("-i", "--in", dest="indir", help="Input file")
parser.add_option("-w", "--out", dest="outdir", help="Output file")
parser.add_option("-v", "--var", dest="varname", help="Name of the variable")

(opts, args) = parser.parse_args()

if oper == 'funcion1':
    funcion1(opts.indir, opts.outdir)
elif oper == 'funcion2':
    funcion2(opts.indir, opts.outdir, opts.varname)
```

En la terminal me basta escribir :

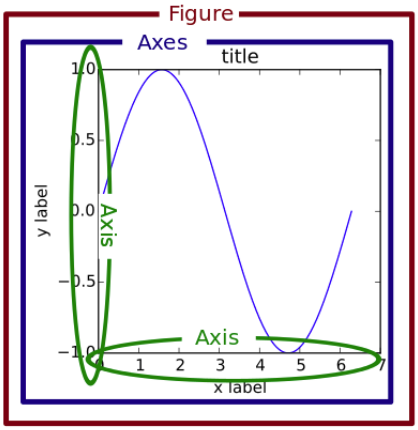
```
python script.py -o funcion2 -i $indir -w $outdir -v $varn
```

5. Graficar con Python

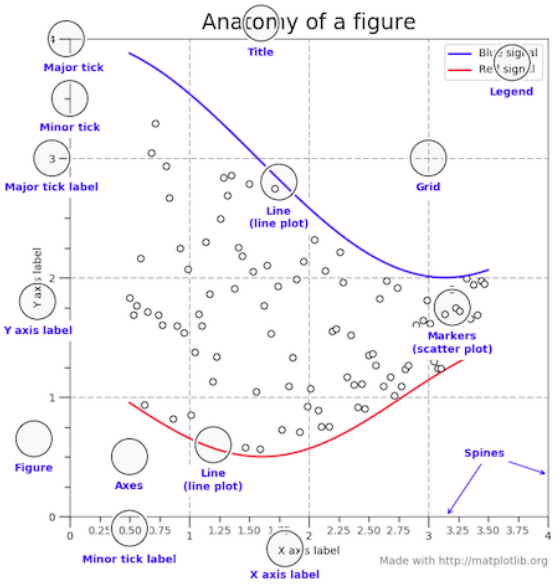
Para graficar con python3, la libreria la más utilizada es matplotlib, muchas otras librerias para graficar están baseadas en matplotlib. Para más informaciones : *https://matplotlib.org/index.html*.

5.1. Gráficos básicos

Matplotlib contiene tres objetos importantes, el más grande es la `figure` que contiene uno o más `Axes`, el segundo objeto, que a su vez contiene `Axis`.



El el siguiente esquema están representados ejemplos de otros objetos, relacionados con uno de los elementos `figure`, `axes` o `axis` que pueden ser modificados.



5.1.1. 1D

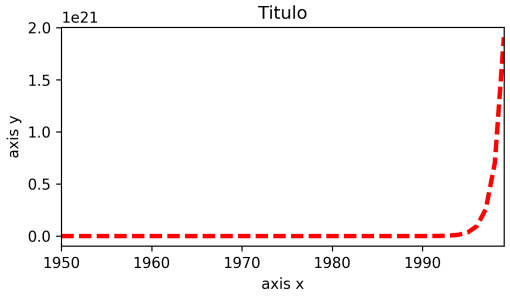
```
import matplotlib.pyplot as plt
import numpy


yr = np.arange(50)
exp = np.exp(rng)
yrs = 1950 + yr

fig, ax = plt.subplots(figsize=(5, 3), dpi = 300)
ax.plot(yrs, exp, color='r', linestyle="--", linewidth=3)

ax.set_title('Titulo')
ax.legend(loc='upper left')
ax.set_ylabel('axis y')
ax.set_xlim(xmin=yrs[0], xmax=yrs[-1])
ax.set_xlabel('axis x')
fig.tight_layout()

#Â Para solo visualizar
plt.show()
# Para guardar el grafico
plt.savefig("/direccion/del/graf/plot1.png", dpi=300)
plt.close()
```



5.1.2. Subplots

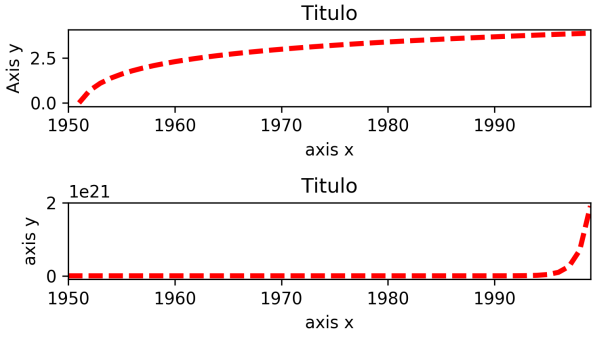
Se puede graficar multiples gráficos en una misma figura gracias a los subplot.

```
fig = plt.figure(figsize=(5, 3), dpi = 300)

fig.add_subplot(211)
plt.plot(yrs, log, color='r', linestyle="--", linewidth=3)
plt.title('Titulo')
plt.legend(loc='upper left')
plt.ylabel('Axis y')
plt.xlim(xmin=yrs[0], xmax=yrs[-1])
plt.xlabel('axis x')

fig.add_subplot(212)
plt.plot(yrs, exp, color='r', linestyle="--", linewidth=3)
plt.title('Titulo')
plt.legend(loc='upper left')
plt.ylabel('axis y')
plt.xlim(xmin=yrs[0], xmax=yrs[-1])
plt.xlabel('axis x')


fig.tight_layout()
plt.savefig("/dir/del/graf/plot2.png", dpi = 300)
plt.show()
plt.close()
```



5.1.3. 2D

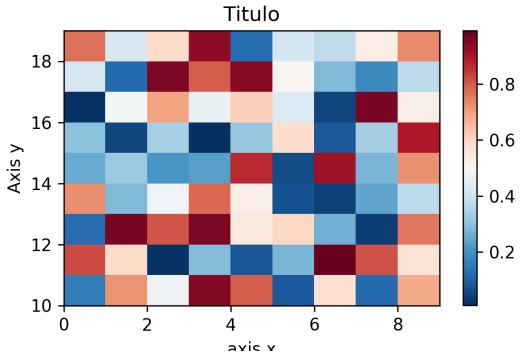
```
A = np.random.random((10,10))
y = np.arange(10,20)
x = np.arange(10)

fig = plt.figure(figsize=(5, 3), dpi = 300)
cs = plt.pcolor(x, y, A, cmap=plt.cm.get_cmap('RdBu_r'))
plt.title('Titulo')
plt.legend(loc='upper left')
plt.ylabel('Axis y')
plt.xlabel('axis x')

cbar=plt.colorbar()

plt.savefig("/dir/del/graph/plot3.png", dpi = 300)

plt.show()
plt.close()
```



5.2. Gráficos sobre mapas

data : es nuestro array 3 dimensiones que depiende de (lat,lon, time) lon, lat : son vectores (caso lonlattype="vect") o pueden ser array de 2 dimensiones lon (lat, lon) y lat (lat, lon) (caso lonlattype=) Se usa Basemap en complemento a matplotlib, para más detalles ver : *https://matplotlib.org/basemap/*

Ej. para una proyección cilíndrica

```
# Libraries
import numpy as np
import matplotlib as mpl
import netCDF4 as nc
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
from netCDF4 import Dataset as NetCDFFile

# Importar los datos
ncfile = NetCDFFile(ncdir, 'r')
olon = ncfile.variables["lon"] #Â si el nombre de la longitud es lon
olat = ncfile.variables["lat"] # si el nombre de la latitud es lat
data = ncfile.variables["var"] #Â si la variable se llama var

# Obtener los minimos de longitud y latitud de nuestra mapa
#(si es una otra proyeccion, ver en matplotlib lo que se necesita)
nlon = np.min(lon)
xlon = np.max(lon)
nlat = np.min(lat)
xlat = np.max(lat)

# Convertir los datos en un array numpy
datatoplot = np.array(data[:,:,:])

# Creacion de la figura con dimension 4*3 y un dpi (resolucion) de 300 (dot per inch)
fig=plt.figure(figsize=(4,3),dpi=300)

# Creacion de la mapa
#Â resolution puede ser empezando por la m\'as baja : "l", "c", "h", "f"
m = Basemap(projection='cyl', resolution="l",llcrnrlon=nlon,
        llcrnrlat=nlat, urcrnrlon=xlon, urcrnrlat=xlat)

# Creacion de los x / y seg\'un el tipo de longitud / latitud
if lonlattype == "vect": #Â si olon y olat son vectores
    print "lonlat: vectors"
    lon, lat = np.meshgrid(olon[:], olat[:])
    xi, yi = m(lon, lat)
else:
    xi, yi = m(olon[:,:], olat[:,:])

# Graficar en la mapa, para mas colores ver en matplotlib los colormaps
cs = m.contourf(xi,yi,datatoplot[:,:,timestep],cmap=plt.get_cmap("rainbow"))
# Posibilidad de utilizar tambien contour, pcolor, pcolormesh etc..
```

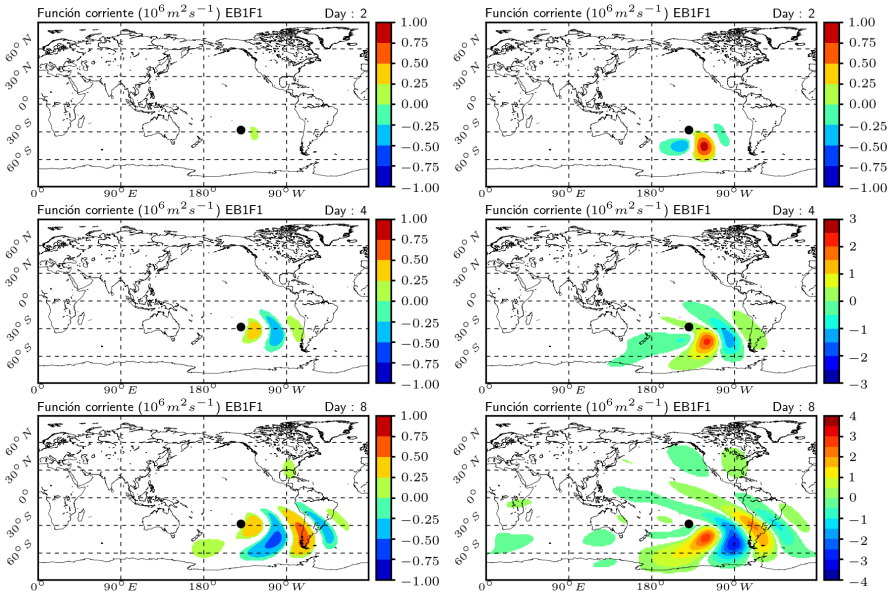
```
# Agregar detalles a la mapa (ver los detalles disponibles en la pagina de basemap)
m.drawparallels(np.arange(-90, 90,45), labels=[1,0,0,0], fontsize=2)
m.drawmeridians(np.arange(-180., 180.,45), labels=[0,0,0,1], fontsize=2)
m.drawcoastlines(linewidth=0.6)
m.drawcountries(color='k',linewidth=0.6)

# Creacion de la barra de color
cbar=m.colorbar(cs,location='right',pad=0.15)
cbar.ax.tick_params(labelsize=10,direction="in",length = 2,pad=4)
plt.setp(cbar.ax.get_xticklabels(), fontsize=18,weight='bold')

# Ajustar la imagen y agregar titulo, espacio a la izquierda / derecha,
#       arriba/abajo, entre los supplot
# left = 0.1 right = 0.9 > la limite izq es a 0.1*Lgrafico de borde izq
#       la der. a 0.9*Lgrafico del borde izq
plt.subplots_adjust(wspace = 0.4, right=0.9,left=0.15,bottom=0.1, top= 0.95,hspace=0.32)
plt.title("Mapa de precipitaciones")

# Para guardar
plt.savefig("/dir/del/graf/plotmap.png",dpi=300)
# o para ver
plt.show()

# Cerrar
plt.close()
```



Ejemplo de mapa hecho con un subplot.

6. Conclusión

Este documento solo presenta las funciones básicas de python3, después a cada uno de ir buscando las funciones, los modules, las clases, o ir creandolo para su propio uso. Python3 es libre y existen muchísimas librerias, sean curiosos para una mejor experiencia con Python3.

7. Exercicios

7.1. Fibonacci

Escribir una función que toma en entrada un entero n y devuelve el elemento n de la suite de Fibonacci (F_n) definida de la manera siguiente :

$F_0 = 0$

$F_1 = 1$

$F_n = F_{n-1} + Fn - 2$

7.2. Practicar con NetCDF

Para practicar a partir de un documento netCDF :

1. Crear una función que permite recuperar una variable desde un documento netCDF
2. Crear una función que recupera la variable de tiempo en un documento netCDF y devuelve el tiempo convertido en datetime
3. Crear una función que encontra en una lista de datetime el índice de una fecha dada
4. Recuperar el promedio de los datos entre dos fechas dadas y graficarlo

8. Correcciones

8.1. Fibonacci

```
def fibo(n):
    if type(n) == int:
        if n<2:
            return n
        else:
            Lo=[0,1]
            j=1
            while j<n:
                Ln=[Lo[1],Lo[0]+Lo[1]]
                Lo=Ln
                j=j+1
            return Ln[1]
    else:
            print("n debe ser entero")
            return
```

```
# llamado por :
fibo(5)
# Y si queremos guardar el resultado en n :
n=fibo(5)
```

8.2. netCDF

```
from netCDF4 import Dataset as NetCDFFile
from netCDF4 import num2date
import numpy as np

def getvar(ncdir, varname):
    ncfile = NetCDFFile(ncdir, 'r')
    var = ncfile.variables[varname]
    return var

def gettime(ncdir, timename)
    ncfile = NetCDFFile(ncdir, 'r')
    var = ncfile.variables[varname]
    dtime = num2date(var[:], var.units)
    return dtime

def finddate(dtime, year, month, day)
    i=0
    while (dtime[i].year != y or dtime[i].month != m or dtime[i].day != d) and i<len(dtime):
        i=i+1
    return i

ncdir="/dire/netcdf/file.nc"
varname="temperature"
timename="time_counter"
year1=2002; year2=2003
month1=2; month2=3
day1=20; day2=10

data = np.array(getvar(ncdir, varname)[:,:,:])
dtime = gettime(ncdir, timename)
ibeg = finddate(dtime, year1, month1, day1)
iend = finddate(dtime, year2, month2, day2)

# considerando que data(lat, lon, time)
datamean = np.mean(data[:,:,ibeg:iend+1], axis=2)

#Para graficar necesitamos lon/lat
#Y despues seguir las instrucciones como visto en el ejemplo
```