Introducción a Python

1 Python

Python es un lenguaje de programación interpretado (no hay que compilarlo), de software libre (no cuesta) y orientado a objetos (Object-Oriented Programming), esto quiere decir que trabaja y realiza funciones con objetos. Un objeto es simplemente un conjunto de datos o variables a los cuales se les puede aplicar un conjunto de funciones. Dependiendo del tipo de objeto que sea nuestra variable se le podrá aplicar un conjunto de funciones.

De tal forma podemos asignar valores a variables/objetos. En el primer ejemplo tendremos objetos con valores numéricos que podrán ser tratados como variables matemáticas.

R es otro lenguaje de programación, también de software libre pero ha sido diseñado para hacer análisis estadístico y gráficas. Es desarrollado por la Fundación para computo estadístico (R). Y puede incluirse en la programación de python dentro de Jupyter Notebook. R también es un lenguaje interpretado, es decir, no hay que compilar el programa para ejecutarlo.

JuPyter es un proyecto de software libre que nació de IPython y evolucionó para dar soporte interactivo a la programación de *ciencia de datos* y *cómputo científico*. Jupyter permite mezclar varios lenguajes de programación como Julia, Python y R.

Jupyter Notebook es una aplicación que permite crear y compartir –con otras personas– documentos que contienen código, ecuaciones, visualización y comentarios de texto. Esta plataforma es ideal para generar el prototipo de un programa y compartir trabajos, incluyendo su visualización. Sin embargo es importante señalar que JuPyter Notebook no se *ejecuta* o *corre* de manera muy eficiente.

Cuando se dice que se programa en python o R normalmente se asume que se está escribiendo código en un archivo de texto para después ejecutarlo en la terminal. En Jupyter Notebook también se escribe código, sin embargo el código se escribe sobre

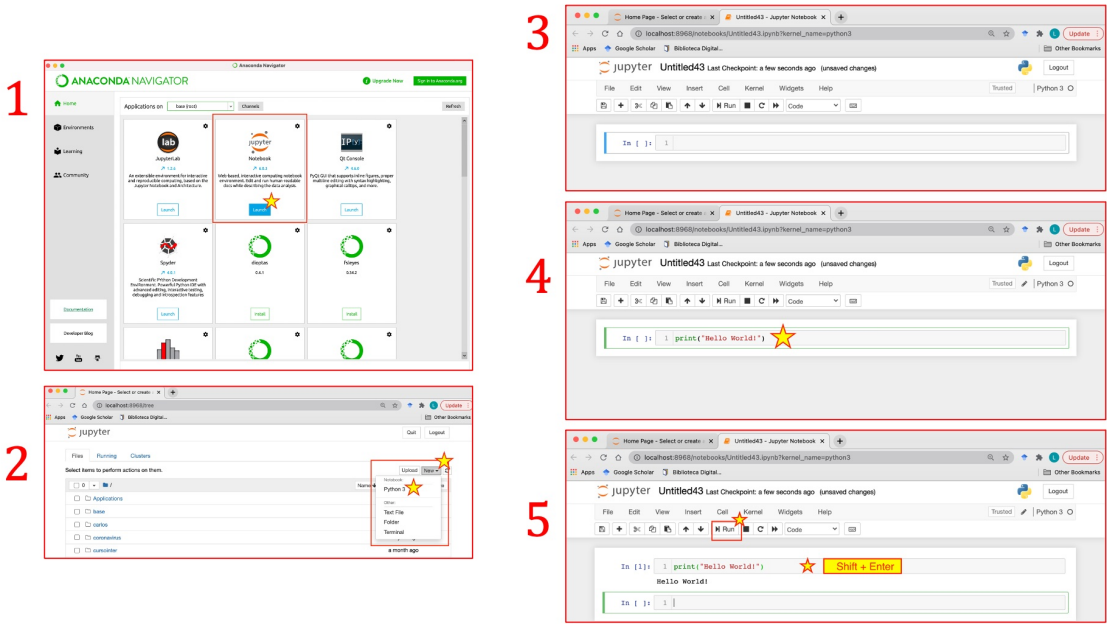
un conjunto de celdas que podrán ser ejecutadas independientemente por *cachos* o *bloques* que fueron programados o escritos en una celda.

Anaconda es una distribución de python y R que tiene el objetivo de simplificar la instalación de paqueterías científicas de python y R. La distribución de anaconda ya contiene librerías como numpy, scipy y matplotlib que vamos a utilizar en este curso.

1.1 Instalación de Anaconda y Jupyter-Notebook

La instalación de Anaconda es es sencilla, hay que seguir los siguientes pasos:

- Descargar e instalar el programa Anaconda según su sistema operativo (Windows, iOS o Linux) de la siguiente liga (https://www.anaconda.com/products/individual
- Inicar el programa Anaconda Navigator (Figura con el número 1)
- Iniciar Jupyter Notebook. Al abrir Jupyter-Notebook se abre una ventana en su navegador (u otro tab en su navegador ya abierto) como se muestra en la siguiente imagen. En el navegador se pueden ver las carpetas de su computadora. (Figura con el número 2)
- Abrir una nueva ventana de Python3 (New, Python3). Al abrir la nueva celda de Python3 se abre otro tab en su navegador como se muestra en la siguiente imagen. En el navegador se pueden ubicar las celdas de programación (Figura con el número 3)
- Empezar a programar escribiendo en la celda ln[1]. (Figura con el número 4)
- Ejecutar la celda con (Shift + Enter) o bien picarle con el mouse a (Play). (Figura con el número 5)



Recomendaciones para Linux

En el sistema operativo linux no es necesario instalar la interfaz Anaconda (al menos yo no lo recomiendo) esto porque se puede instalar jupyter notebook directamente con el programa *pip*, de tal forma nos ahorramos la interfaz. Cada librería hay que instalarla independiente y Jupyter notebook estará ejecutando todas las librerías que se encuentren instaladas en su computadora.

```
> laural$ pip install notebook
```

y dentro de la terminal ejecutar

```
> jupyter notebook
```

1.2 Funcionamiento general de Jupyter Notebook

La interfaz de un notebook es muy familiar, es parecida a un procesador de texto y tiene un conjunto de botones para picarle con el mouse. Hay dos botones particulares, las *celdas* y el *kernel*.

- El *kernel* es el motor computacional que ejecutará el código contenido dentro del notebook y va a utilizar python o R según le hayamos indicado al inicio.
- Una celda es el contenedor del texto de programación que será ejecutada por el *kernel*

Celdas

Podemos identificar las celdas ya que tienen un recuadro sombreado. Y una numeración a la izquierda $In[\ ]$. Cuando ejecutamos la celda con $shift + enter$ la *salida* del código que se escribió en la celda se despliega abajo de la misma. Al momento de ejecutar una celda cambia la numeración de $In[\ ]$ a $In[1]$ así podemos saber si ya ejecutamos una celda y también llevar la cuenta de cuál es la última línea que ejecutamos y en qué orden.

Comentarios o Markdowns

Podemos convertir una celda de código en una celda de texto o $Markdown$. Una celda de este tipo no ejecuta código python, simplemente contiene texto. Lo bonito de estas celdas tipo Markdown es que pueden procesarse con LaTex —programa que procesa texto— para que las ecuaciones se vean bonitas. Una celda $Markdown$ sirve para que nuestro notebook tenga comentarios, aclaraciones, títulos etc. Que justamente es el propósito de un notebook (cuaderno/apunte).

Archivo .ipynb

El archivo .ipynb es un notebook. Cada vez que se crea un notebook nuevo, se genera un nuevo archivo .ipynb. El archivo .ipynb contiene toda la información del notebook en un formato llamado $json$. Tiene información de cada celda y su contenido, incluidas imágenes, attachments etc.

Este archivo es diferente a un archivo .py que únicamente contiene el texto de la programación.

Otras monerías

Para facilitar la escritura de código, un notebook tiene varias funciones o *shortcuts*

- $Esc + Enter$ cambiar de editar la propiedad de una celda a editar el contenido de una celda (en modo escritura).
- A o B inserta una celda arriba o abajo de la celda activa
- M transforma una celda a celda tipo $Markdown$
- D elimina una celda
- Z regresa la celda barrada
- $Ctl + Shift + -$ Divide la celda activa en la posición del cursor

Kernel

Atrás de cada notebook hay un kernel. Cuando ejecutas o corres el código de la celda, ese código es ejecutado dentro de un kernel. Este kernel persiste en el tiempo y entre las celdas, contiene la ejecución del documento completo y no de las celdas individuales. Digamos se acuerda y contiene lo que se ejecutó en las primeras celdas que ejecutaron o corrieron.

Por ejemplo, cuando se definió una variable o se importó una función de una librería la información se va al kernel y se todo tiene ahí guardado. El kernel va a funcionar sin importar el orden de ejecución de las celdas en un notebook. Si se ejecuta una celda en la parte superior de un notebook en donde se redefinen variables, pues permanece esta redefinición si no se ejecutan celdas siguientes.

Nos va a pasar muchas veces que trabemos nuestra computadora escribiendo un código erróneo, por ejemplo con un loop infinito. O bien que perdamos la cuenta de qué ya se ejecutó o no. Para esto podemos reiniciar el kernel, que sería equivalente a borrón y cuenta nueva. En ese caso hay que recordar que tenemos que volver a ejecutar todas las líneas de nuestro notebook.

2 Iniciando con Python

2.1 Variables u objetos en Python

Tenemos diferentes tipos de variables/objetos en python:

1. **Numéricas:** Estas variables/objetos contienen números y con ellas se podrán ejecutar funciones características para el tipo de número que se define. A su vez, las variables numéricas se dividen en tres tipos:

   - **Integer (int)** son números enteros, no tienen un límite de qué tan grande o chico (negativo) puede ser este número —aparte de la capacidad de memoria de tu computadora.

   ```
   >>> x=7
   >>> y=-5
   ```

   - **Float (float)** o número real con punto decimal es distinto al número entero dado que python necesita alocar memoria de forma distinta para este tipo de números. Hay que tomar en cuenta que en los números reales o floats sí tendremos un valor máximo superior (1.79×10^{308}) y mínimo más cercano a cero (5.0×10^{-234}).

   ```
   >>> x=5.5
   >>> y=-5.75
   ```

   - **Complex (complex)** son números complejos, que tienen una parte real y una parte imaginaria. Las operaciones que se pueden realizar con estos números todas aquellas características de los números complejos. La forma más sencilla para definir un número complejo es la siguiente:

   ```
   >>> x=5+7j
   >>> y=3-8j
   ```

2. **Strings (str):** Estas variables/objetos consisten en una secuencia de caracteres y se les pueden aplicar funciones propias para los strings. Para asignar a

un objeto un string se utilizan comillas, todo lo que vaya dentro de las comillas será parte del objeto, incluidos espacios y guiones:

```
>>> x="Hola "
>>> y="Mundo!"
```

3. **Listas(list)**: Estas variables son secuencias ordenadas de elementos. Es una de las variables más utilizadas. Los elementos de un lista no necesariamente deben ser del mismo tipo, pueden ser strings o variables numéricas. Para declarar un lista se deben agregar cada elemento dentro de corchetes cuadrados [ ] separados por comas.

```
>>> x=[ 1, 7, 5, 8, 9, 2 ]
>>> y=[ 1, 4, 2.2, "Hola ", 7.5 ]
```

Para extraer un elemento de una lista, o un conjunto de elementos, se utiliza el operador *slicing* [ ]. Por ejemplo:

```
>>> x=[ 1, 7, 5, 8, 9, 2 ]
>>> x[2]
5
```

Y un detalle muy importante es que **python inicia a contar desde cero**, de tal forma el primer elemento de la lista es el x[0]=1, el segundo es el primero x[1]=7, x[2]=5, etc, etc.

Para extraer varios elementos de la lista:

```
>>> x=[ 1, 7, 5, 8, 9, 2 ]
>>> x[2:5]
[5, 8, 9]
```

4. **Tuplas:** son —al igual que las listas— secuencias ordenadas de elementos. La diferencia principal que tienen con las listas es que una vez que se crea un tupla no se podrá modificar. Esta particularidad de las tuplas tiene el objetivo de proteger datos. Para definir una tupla se agregan elementos separados

por comas dentro de paréntesis ( ), podemos utilizar el operador *slicing* para extraer datos pero no para modificarlos:

```
>>> x=( 1, 7, 5, 8, 9, 2 )
>>> x[2:5]
[5, 8, 9]
```

5. **Diccionarios (dict)** son conjunto de datos no ordenados de pares nombre-valor (key-value). Los diccionarios pueden almacenar un monton de datos y están optimizados para obtener o extraer datos y necesitamos saber el nombre para obtener los datos que queremos extraer. Para definirlos se utilizan las llaves y unos dos puntos para asignar a cada nombre su valor (key:value). Los valores pueden ser de cualquier tipo, por ejemplo, strings, variables numéricas o listas!

```
>>> d={ "uno":"rojo", "dos":"naranja", "tres":"amarillo",
"cuatro":"verde", 5:"azul", 6:"morado" }
>>> d["dos"]
"naranja"

>>>d[5]
"azul"
```

   En el ejemplo anterior se puede ver que lo que se utiliza para extraer un elemento del diccionario haciendo un *slicing* es el nombre de mi elemento, no el número de elemento como lo hicimos en el caso de las listas o tuplas

6. **Boleanos (bool):** este tipo de variables pueden tener únicamente dos valores verdadero (True) o falso(False):

```
>>> a=True
>>> type(a)
<class 'bool'>
```

2.2 Reasignar el tipo a una variable

Podemos asignar un valor numérico a una variable, preguntar de qué tipo de variable se trata con el comando

```
type(x)
```

y además convertir variables de un tipo a otro —siempre que sea posible. Esto se hace escribiendo la abreviación del tipo de variable a la que queremos convertir seguido del nombre de la variable entre paréntesis

```
float(x), int(x), complex(x)
```

```
>>> x=5
>>> type(x)
<class 'int'>

>>> float(x)
>>> print(x)
5

>>>type(x)
 <class 'float'>

>>> y=7.2
>>> type(y)
<class 'float'>

>>>int(y)
>>>y
7

>>>type(y)
<class 'int'>

>>>float("2.7")
2.7

>>str(y)
```

```
        "2.7"

        >>>complex(y)
        (2.7+0j)
```

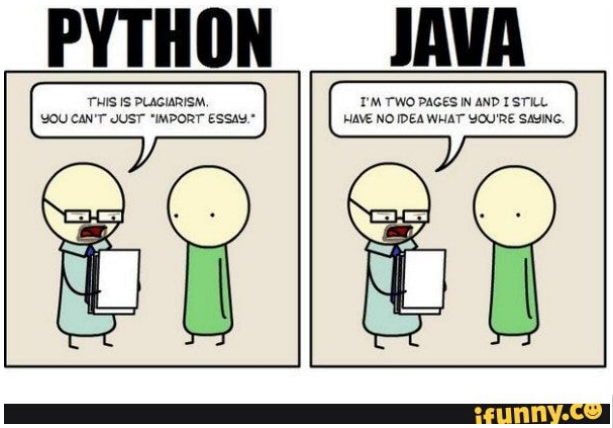
Cuidado! para REASIGNAR el tipo de una variable no basta con ejecutar el comando

```
int(x)
```

tenemos que redefinir a la variable de la siguiente manera:

```
x=int(x)
```

```
>>> y="5.4" #aquí tenemos un string
>>> y=float(y) # redefinimos nuestra variable
>>> print(y) # la imprimimos
5.4
```



[h]

3 Librerías de Python

Python tiene grandes ventajas sobre algunos otros lenguajes de programación. Una de esas ventajas es que tiene un conjunto de librerías enfocadas a ciencias. Por ejemplo tenemos librerías de python numérico (NumPy), python científico (SciPy), librerías para jugar con bases de datos (pandas) librerías para graficar (matplotlib) o para hacer matemáticas simbólicas (SymPy).

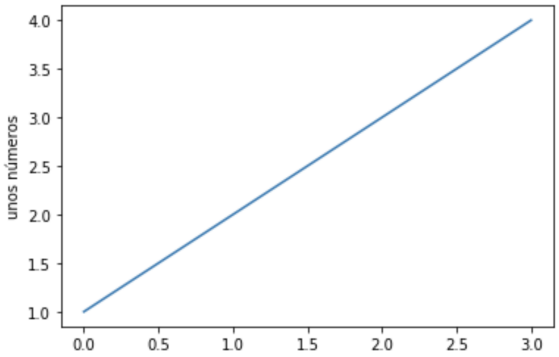
Gráficas, Matplotlib

Matplotlib.pyplot es una librería que integra un conjunto de funciones parecidas a MATLAB. Cada función de pyplot realiza un comando a una figura, por ejemplo crea una figura, crea un área para graficar en esa figura, gráfica líneas en el área para graficar, decora la gráfica con títulos, mueve los ejes, etc.

Hacer gráficas en Python con la libreria Matplotlib.pyplot es sencillo pero hay que conocer cómo se deben ejecutar las diferentes funciones o comandos. De forma general, los pasos para hacer una gráfica son los siguientes:

- Importamos la librería matplotlib.pyplot (import matplotlib.pyplot as plt)
- Se genera la gráfica (plt.plot([1,2,3,4]))
- Podemos cambiar las etiquetas de la gráfica (plt.ylabel(etiqueta para eje y))
- Pedirle a python que nos muestre la gráfica (plt.show(), si con todo y los paréntesis)

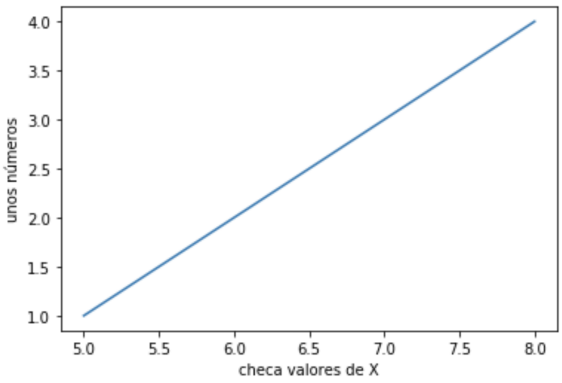
```
import matplotlib.pyplot as plt # importa la libreria en el programa
plt.plot([1,2,3,4]) # el conjunto de datos que vamos a graficar,
        #es una lista de números!
plt.ylabel(unos números}) #modifico el nombre del eje y
plt.show() #le pido a python que me muestre la gráfica
```



Valores en x y y

Al ejecutar mi gráfica, se puede ver que el eje x de mi gráfica tiene números del 0 al 3. Si le damos una lista de números a matplotlib para graficar, python asume que se trata de una secuencia de valores de y y automáticamente python genera valores de x. Como ya se mencionó, python empieza la numeración desde el 0, por eso la gráfica inicia en cero. También podemos mandar los valores de x en otra lista para graficarlos pero CUIDADO! es necesario que dos listas tengan el mismo número de puntos, o que las listas tengan el mismo tamaño.

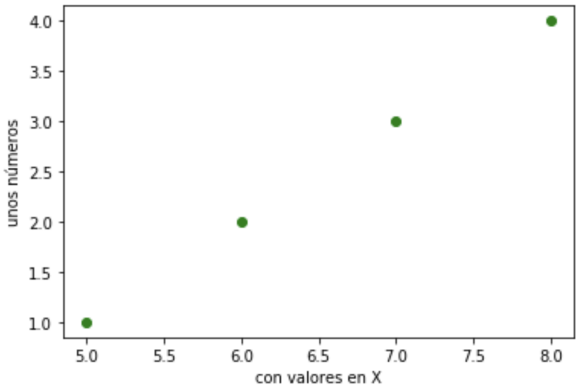
```
plt.ylabel("unos números")
plt.xlabel("checa valores de X")
plt.plot([5,6,7,8],[1,2,3,4])
```



Cambiamos el formato de los puntos

El formato de la gráfica se envía a la hora de graficar. Automáticamente Matplotlib utiliza el formato -b que es una línea sólida. Pero por ejemplo se pueden utilizar otros formatos, como go que va a mostrar círculos verdes (green o, circulos verdes):

```
plt.ylabel("unos números")
plt.xlabel("con valores en X")
plt.plot([5,6,7,8],[1,2,3,4], "go")
```

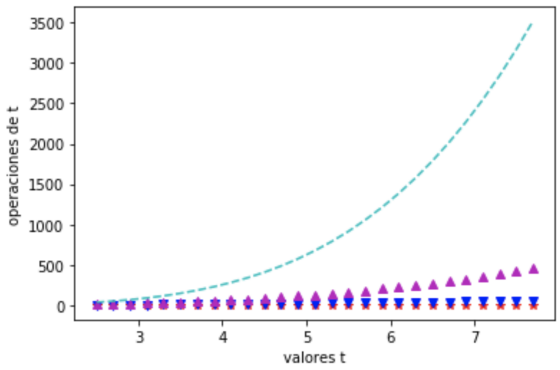


Operaciones dentro de la gráfica

Podemos generar un conjunto de valores con la función *arange* de numpy (np.arange(a,b,c)). Esta función de numpy genera un conjunto de números entre a y b con un espaciado de c. Podemos incluir estos datos dentro de la misma función para graficar, se

pueden a su vez generar nuevos números y hacer operaciones. Por ejemplo el cuadrao de t, cubo, a la cuarta y graficar el conjunto de varios datos:

```
import numpy as np  #importamos la librería de  python numérico
t=np.arange(2.5,7.8,0.2) #donde a, b y c son números reales
plt.plot(t,t, "r*", t, t**2, "bv", t,t**3, "m^",t,t**4, "c--") #operaciones!
```



Gráficas a partir de bases de datos

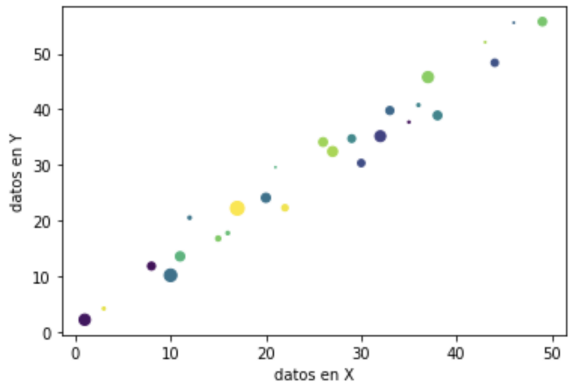
Podemos también graficar datos que se leen de una base de datos (para más detalle ver la siguiente sección). Esto porque típicamente tendremos listas de variables. Para el caso de este ejemplo vamos a generar un conjunto de datos aleatorios y los vamos a acumular en la variable *data*. La siguiente gráfica tiene dos cosas interesantes, el color de los puntos (c=c, color=c) tienen una escala de valores de acuerdo al valor de la lista de datos en la variable $s = data[3] * 35$. Y el tamaño de los puntos (s=data[3]*35) también dependen de los datos de una de las listas de *data*. Ojo también con los comandos de numpy para generar números aleatorios:

```
np.arange(50); generamos 50 números entre el 0 y el 50,
        incluimos el 0 y no el 50.
np.random.randint(0,50,50); generamos 50 números aleatorios
        enteros entre el 0 y el 50. C
np.random.randn(50); generamos 50 números aleatorios entre -1 y 1
```

La escala de colores y el tamaño de puntos

Dentro de la función plot, se puede especificar el color de los puntos con el comando $c = color$. De la misma forma, el tamaño de los puntos de la gráfica se especifican con el comando $s = 4$. Ambas variables de las gráficas (color (c) y size (s)) pueden depender del conjunto de datos, es decir, pueden variar. Para hacerlo se especifican en función del conjunto de datos, por ejemplo:

```
c=data[2], s=data[3]*35
```



Para generar la gráfica de esta sección tenemos el siguiente código:

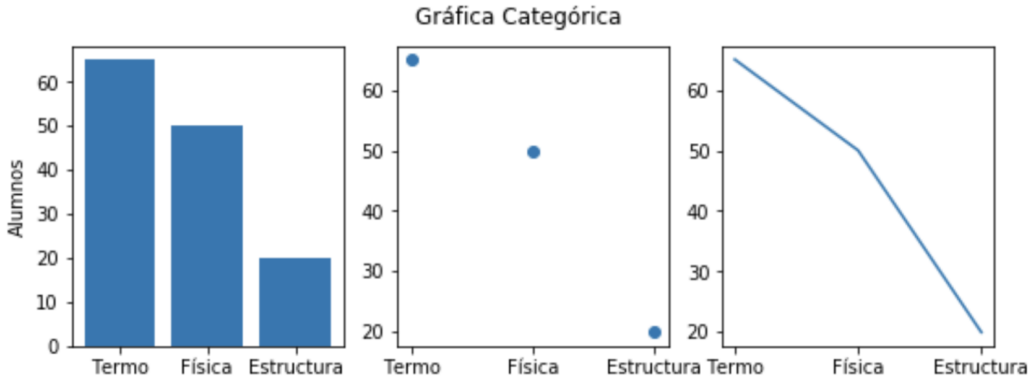
```
a=np.arange(50)
b=a+10*np.random.rand(50)
c=np.random.randint(0,50,50)
d=np.random.randn(50)

#podemos checar los datos que se generan imprimiento las listas de datos
#print("a=",a)
#print("b=",b)
#print("c=",c)
#print("d=",d)

data = [a,b,c,d] #generamos una lista de listas de datos
plt.xlabel("datos en X")
plt.ylabel("datos en Y")
plt.scatter(data[0],data[1],c=data[2], s=data[3]*35)
```

Varias gráficas en una figura con variables categóricas

Se pueden graficar variables categóricas, por ejemplo si tenemos una lista de número de alumnos para cada materia.

En el siguiente ejemplo vamos a graficar en una sola figura tres veces los mismos datos, es decir, vamos a hacer tres gráficas. Los pasos son los siguientes:

- En las líneas 1 y 2 definimos nuestras variables (checa cómo las listas tienen el mismo número de elementos)
- En la línea 3 vamos a definir el tamaño de la figura. El default es en inches, pero se puede especificar en centímetros también (15*cm, 5*cm)
- En la línea 6 especificamos un ARREGLO para acomodar las tres gráficas con los valores (131) el primer número dice cuántas gráficas vamos a acomodar en el eje vertical (1 en este caso) el segundo número cuantas gráficas vamos a mandar en el eje horizontal (2 en este caso) y finalmente especificamos que la subgráfica de la que estamos hablando es la número 1 (131).
- Las siguientes instrucciones van a aplicar únicamente para la primer gráfica por eso únicamente aparece la etiqueta del eje y en la primera figura de acuerdo a lo que se especifíca en la línea número 7 del código $plt.ylabel(\text{"}Alumnos\text{"})$
- En la línea 8 hacemos la primera gráfica en formato de barras (bar)
- En la línea 10 especificamos que la siguiente instrucción que aplica para la segunda gráfica (132), luego graficamos
- lo mismo sucede en las líneas 13 y 14.
- Es importante notar que una vez que se manda a graficar, por ejemplo con los comandos plt.plot, plt.bar, plt.scatter termina las instrucciones que iniciaron en plt.subplot()
- La última instrucción en la línea 15indica el título superior de la figura entera.




```
materia=["Termo", "Física", "Estructura"]
alumnos=[65, 50, 20]

plt.figure(figsize=(9,3)) #default en inches

plt.subplot(131)
plt.ylabel("Alumnos")
plt.bar(materia, alumnos)

plt.subplot(132)
plt.scatter(materia, alumnos)

plt.subplot(133)
plt.plot(materia, alumnos)

plt.subtitle('Gráfica Categórica')

plt.show()
```

Modificar otros detalles de las gráficas

Desde luego se pueden modificar todos los aspectos de las gráficas y para eso lo ideal es revisar el manual general de matplotlib, esta únicamente es una guía para iniciar. Entonces, de los detalles que a mi me gusta modificar están los siguientes:

- linewidth=2.0; cambia el grosor de la línea: plt.plot(x, y, linewidth=2.0)

- color='r' especifica el color de la línea
- marker='+' cambia el marcador de los puntos
- plt.grid() le pone grid a la gráfica
- Podemos especificar dentro de las etiquetas el tamaño de la letra y su color. plt.xlabel('Mis datos', fontsize=14, color='red')
- Hay una infinidad de detalles que se pueden modificar. Leer el manual

3.1 Trabajando con datos, Pandas

Trabajar con bases de datos grandes o enormes *es lo de hoy*. Vivimos en los tiempos del *big data*. Eso en parte se debe a que las computadoras son extraordinarias para guardar mucha información y procesarla.

El ejemplo más interesante que se me ocurre el día de hoy son los datos internacionales del número de personas contagiadas de COVID-19 y que nos tiene encerrados en casa tomando clases en línea.