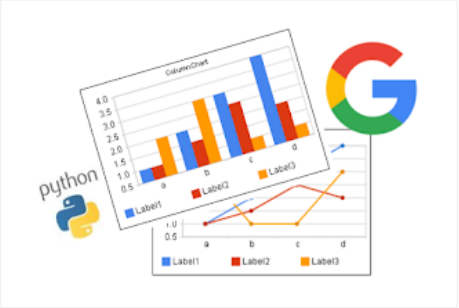
Python 3 para impacientes

"Simple es mejor que complejo" (Tim Peters)

jueves, 23 de junio de 2016

Gráficos con GooPyCharts



GooPyCharts es un módulo desarrollado por **Google** para la creación de gráficos con Python 2.x/3.x. Es muy fácil de utilizar y para el desarrollo de los gráficos más comunes es una opción a considerar que se suma a otras bibliotecas como **matplotlib**.

GooPyCharts permite generar los gráficos (de momento, de líneas, barras, dispersión e histogramas) en archivos **html** comunes que se muestran en el navegador web, o bien, como elementos de un cuaderno (**Notebook**) de la aplicación web **Jupyter**.

Instalación del módulo gpcharts

Para instalar **GooPyCharts** con pip:

En GNU/Linux:

$ sudo pip install gpcharts

En Windows:

c:\> pip install gpcharts

También, se puede descargar el archivo del módulo **gpcharts.py** en el directorio de trabajo o en el directorio donde se encuentren los paquetes Python y utilizarlo realizando la importación correspondiente:

from gpcharts import figure

Por último, para completar la instalación es posible que se tenga que instalar el módulo **future**, que permite que un mismo programa se pueda ejecutar tanto con Python 2.x como con Python 3.x. Si fuera necesaria su instalación, cuando se ejecute un programa que haga uso del módulo **GooPyCharts**, aparecerá un mensaje informando de este requisito. En tal caso, instalar:

En GNU/Linux:

$ sudo pip install future

En Windows:

C:\> pip install future

Crear gráficos en páginas HTML comunes

Los gráficos se pueden generar en archivos **HTML** comunes para visualizarlos directamente en un navegador Internet (recomendado: **Chrome** y **Firefox**). Cuando se ejecute el código fuente de un gráfico se creará el correspondiente archivo **.html** en el directorio de trabajo y, con posterioridad, se abrirá, automáticamente, en el navegador Internet predeterminado. Además del gráfico se mostrarán dos enlaces que permiten la descarga del gráfico en formato **.png** y de los datos utilizados, en formato **.csv**.

Los archivos **.html**, también, se pueden publicar en webs y enviar por correo electrónico como

Buscar

Buscar

Python para impacientes

Python
IPython
EasyGUI
Tkinter
JupyterLab
Numpy

Anexos

Guía urgente de MySQL
Guía rápida de SQLite3

Entradas + populares

Dar color a las salidas en la consola
En Python para dar color a las salidas en la consola (o en la terminal de texto) existen varias posibilidades. Hay un método basado ...

Instalación de Python, paso a paso
Instalación de Python 3.6 A finales de 2016 se produjo el lanzamiento de Python 3.6 . El propósito de esta entrada es mostrar, pas...

Añadir, consultar, modificar y suprimir elementos en Numpy
Acceder a los elementos de un array. [], [,], ... Acceder a un elemento de un array. Para acceder a un elemento se utiliz...

Variables de control en Tkinter
Variables de control Las variables de control son objetos especiales que se asocian a los widgets para almacenar sus valore...

Cálculo con arrays Numpy
Numpy ofrece todo lo necesario para obtener un buen rendimiento cuando se trata de hacer cálculos con arrays. Por como está concebid...

Tkinter: interfaces gráficas en Python
Introducción Con Python hay muchas posibilidades para programar una interfaz gráfica de usuario (GUI) pero Tkinter es fácil d...

Operaciones con fechas y horas. Calendarios
Los módulos datetime y calendar amplían las posibilidades del módulo time que provee funciones para manipular expresiones de ti...

Convertir, copiar, ordenar, unir y dividir arrays Numpy
Esta entrada trata sobre algunos métodos que se utilizan en Numpy para convertir listas en arrays y viceversa; para copiar arrays d...

Tkinter: Tipos de ventanas
Ventanas de aplicación y de diálogos En la entrada anterior tratamos los distintos gestores de geometría que se utilizan para di...

Threading: programación con hilos (I)
En programación, la técnica que permite que una aplicación ejecute

adjuntos de mensajes. La única dependencia que tienen para poder mostrar su contenido es que es necesario contar con acceso a Internet (para acceder a los sitios code.jquery.com y www.gstatic.com).

A continuación, se ofrecen varios ejemplos para mostrar lo sencillo que es utilizar este módulo y, sobre todo, la calidad que tienen los resultados que se obtienen.

Gráfico de una línea: plot()

La clase **figure** se utiliza para crear objetos gráficos de distintos tipos. Tiene un método llamado **plot()** para construir gráficos de líneas.

El siguiente ejemplo crea un gráfico de una línea que representa los valores de una lista.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from gpcharts import figure

grafico1 = figure()
grafico1.plot([3,4,5,4,3])
```

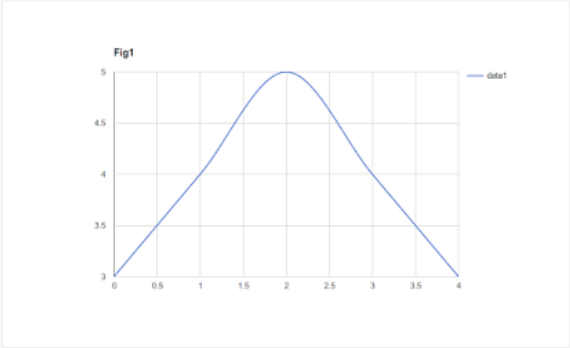


Gráfico de varias líneas

Para crear un gráfico con dos o más líneas los valores se agruparán en listas con tantos valores como líneas a representar, en la lista principal.

```
grafico2 = figure()
grafico2.plot([[3,4],[4,2],[5,1],[4,2],[3,4]])
```

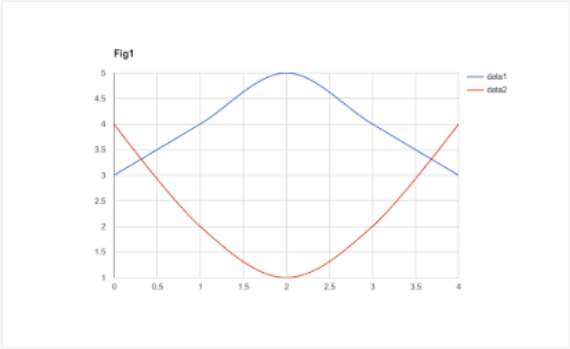


Gráfico de líneas con título, etiquetas de ejes y tamaño determinado

El siguiente ejemplo crea un gráfico de dos líneas de 800x600 que incluye un título y etiquetas para los ejes de coordenadas:

```
grafico3 = figure(title='Promedio horas extras',
                  ylabel='Num Horas',
                  width=800,height=600)
valoresX = ['Dias Semana','Lun','Mar','Mie','Jue','Vie']
valoresY = [['Enero', 'Febrero'],[3,4],[4,2],[5,1],[4,2],[3,4]]
grafico3.plot(valoresX,valoresY)
```

simultáneamente varias operaciones en el mismo espacio de proceso se...

Archivo

junio 2016 (2)

python.org

pypi.org

Sitios

- ActivePython
- Anaconda
- Bpython
- Django
- Flask
- Ipython
- IronPython
- Matplotlib
- MicroPython
- Numpy
- Pandas
- Pillow
- PortablePython
- PyBrain
- PyCharm
- PyDev
- PyGame
- Pypi
- PyPy
- Pyramid
- Python.org
- PyTorch
- SciPy.org
- Spyder
- Tensorflow
- TurboGears

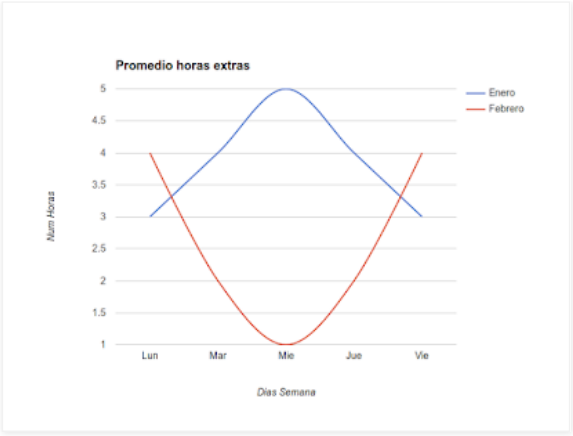


Gráfico de línea con serie temporal

En el eje x de un gráfico se pueden utilizar series temporales pero los valores se deben expresar como una cadena utilizando el siguiente formato '**aaaa-MM-DD HH: MM: SS**'; teniendo en cuenta que la parte '**HH: MM: SS**' es opcional.

A continuación, un ejemplo que crea un gráfico de una línea utilizando una serie temporal, en el que se asignan los valores de los atributos después de la creación del objeto.

```
grafico4 = figure()
grafico4.title='Promedio horas extras'
grafico4.ylabel='Num Horas'
grafico4.xlabel='Dias Semana'
grafico4.width=800
grafico4.height=600
valoresX = [grafico3.xlabel,'Lun','Mar','Mie','Jue','Vie']
valoresY = ['Enero',3,4,5,4,3]
grafico4.plot(valoresX,valoresY)
```

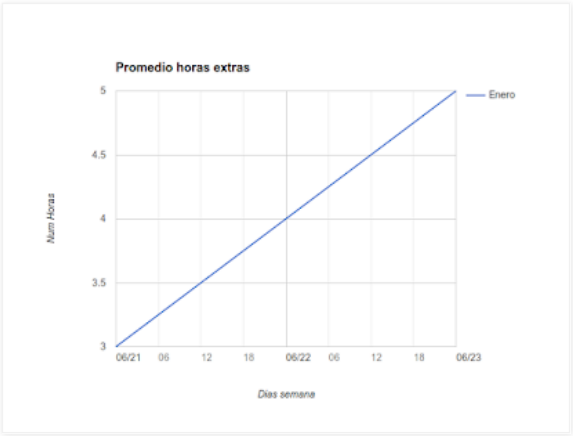


Gráfico de líneas con escala logarítmica

El atributo **logScale** del método **plot()** con el valor **True** se emplea para gráficos que necesitan mostrar valores utilizando escalas logarítmicas.

```
grafico5 = figure(title='Propagacion de virus',
             ylabel='Equipos')
xVals = ['Horas',1,2,3,4]
yVals = [['TipoA', 'TipoB'],[10,30],[20,3000],
        [400,300000],[8000,30000000]]
grafico5.plot(xVals,yVals,logScale=True)
```

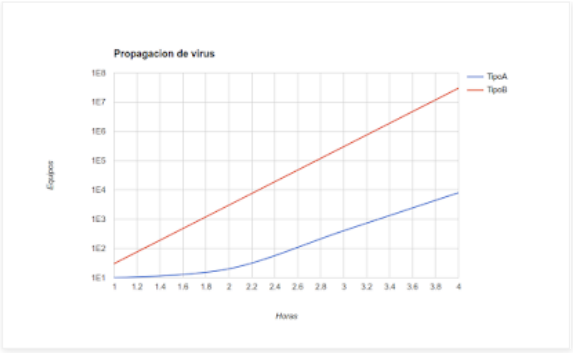


Gráfico de dispersión: scatter()

El método **scatter()** se utiliza para crear gráficos de dispersión. Su atributo **trendline** con el valor **True** indica que se ha de mostrar la línea de tendencia.

```
grafico6 = figure('Evaluaciones y tendencia del alumno')
grafico6.ylabel="Calificaciones"
grafico6.xlabel="Pruebas realizadas"
grafico6.scatter([5,5,6,5,6,7,8,9,5,6], trendline=True)
```

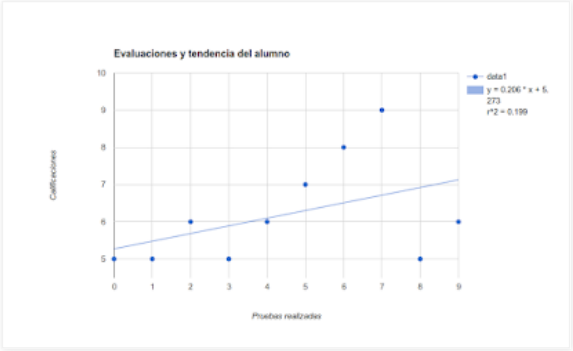
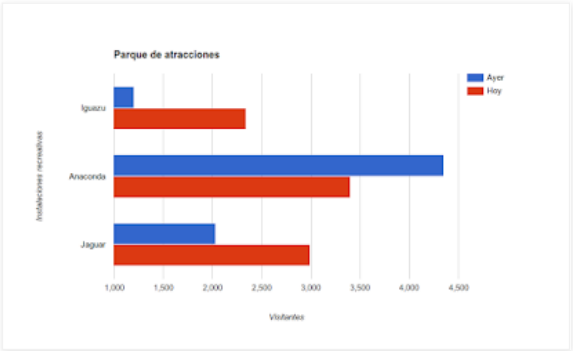


Gráfico de barras: bar()

El método **bar()** permite crear gráficos de barras. Como sucede con los gráficos de líneas, en la lista de valores del eje Y se incluirán listas con tantos valores como barras a representar.

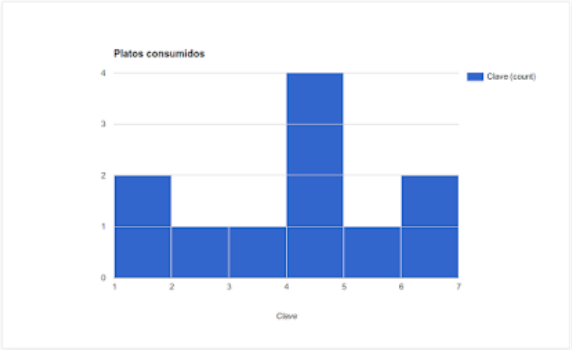
```
grafico7 = figure('Parque de atracciones')
grafico7.ylabel='Instalaciones recreativas'
valoresx=['Visitantes','Iguazu','Anaconda','Jaguar']
valoresy=[['Ayer','Hoy'],
       [1203,2340],[4350,3400],[2030,2990]]
grafico7.bar(valoresx,valoresy)
```



Histograma: hist()

El método **hist()** se utiliza para crear histogramas (representación gráfica de una variable en forma de barras, donde la superficie de cada barra es proporcional a la frecuencia de los valores representados).

```
grafico8 = figure('Platos consumidos',xlabel='Cantidad')
grafico8.xlabel = "Clave"
grafico8.hist([6,4,5,4,4,1,1,2,3,4,6])
```



Crear gráficos en un cuaderno Jupyter Notebook

Jupyter Notebook es una aplicación web que permite crear documentos que pueden contener código Python editable que se puede ejecutar, ecuaciones, textos explicativos, gráficos y otros contenidos multimedia.

Entre las ventajas de utilizar este tipo de cuadernos está la posibilidad de cambiar los contenidos con facilidad y, quizás la más importante, que los cuadernos se pueden compartir con otras personas como se hace con otros documentos.

El módulo **GooPyCharts** cuenta con métodos específicos para crear gráficos en un cuaderno **Jupyter** que son equivalentes a los utilizados con anterioridad: **plot_nb()**, **bar_nb()**, **scatter_nb()** y **hist_nb()**.

Instalar Jupyter Notebook

Para instalar **Jupyter Notebook** y todas sus dependencias:

En GNU/Linux:

$ sudo pip install jupyter

En Windows:

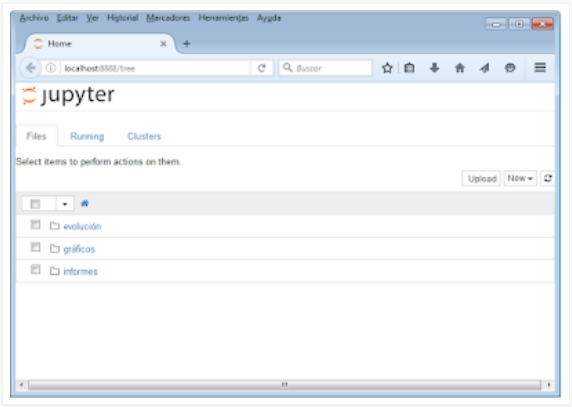
c:\> pip install jupyter

Iniciar Jupyter Notebook

En GNU/Linux y Windows:

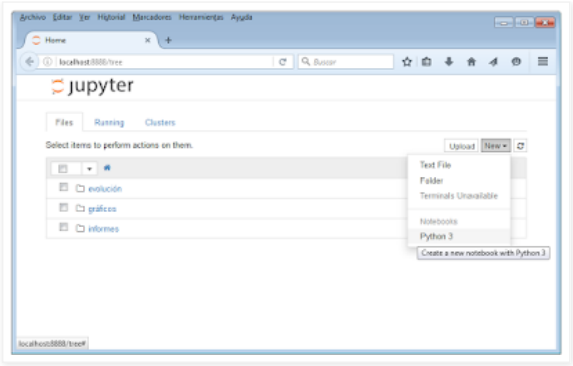
$ jupyter notebook

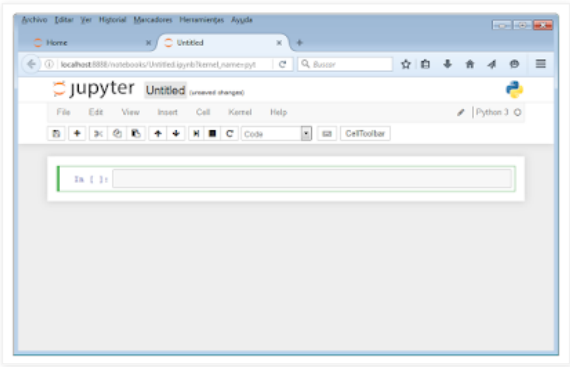
A continuación, se iniciará el navegador internet predeterminado con la aplicación **Jupyter Notebook**:



Crear un cuaderno Jupyter Notebook

Para crear un cuaderno en el directorio actual seleccionar la acción "**New**" y después "**Python 3**". Después de la acción se creará un libro vacío con el nombre "**Untitled1**" que se puede renombrar (p. e. "*Gráficos*") haciendo clic sobre el propio nombre, editando y aceptando el cambio).





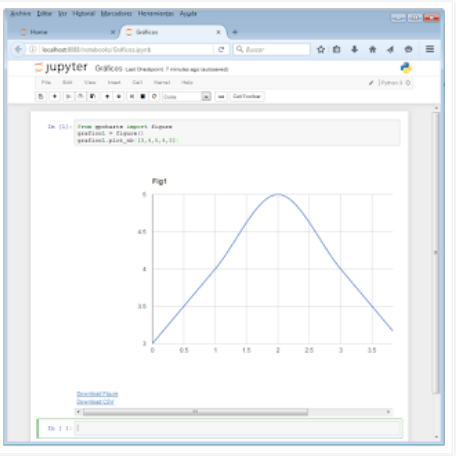
Un cuaderno comienza con un celda vacía identificada con el literal **"In [ ]"** donde, en principio, se puede introducir código Python.

Insertar un gráfico en un cuaderno Jupyter Notebook

Para insertar un gráfico de ejemplo en el cuaderno actual insertar el siguiente código Python en la celda vacía:

```
from gpcharts import figure
grafico1 = figure()
grafico1.plot_nb([3,4,5,4,3])
```

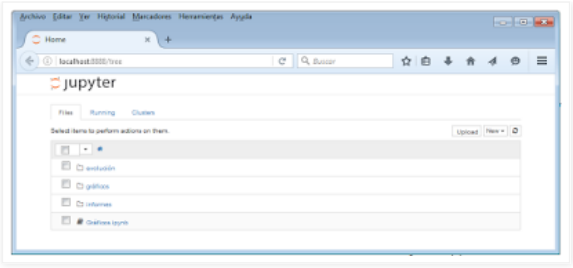
A continuación, para ejecutar el código hacer clic sobre el botón **"Run Cell"**. El resultado, es decir, el gráfico, aparecerá debajo de la celda actual. Después de que aparezca también lo hará una nueva celda (de entrada) vacía **"In []"** que estará lista para "recibir" nuevo código u otros contenidos de otro tipo.



Además, en cualquier momento será posible editar y ejecutar el código de cualquier celda que se vaya agregando al cuaderno actual. Esta es una de las ventajas que ofrece trabajar con este tipo de documentos.

Para cerrar el cuaderno actual hacer clic sobre el botón **"Save"** y después seleccionar en el menú **"File"** la opción **"Close and Halt"**.

Después, si se examina el directorio actual se verá el archivo del cuaderno creado, que se podrá abrir en cualquier momento para seguir trabajando en él, simplemente, haciendo clic sobre su nombre.



Para probar en el cuaderno todos los ejemplos realizados con anterioridad es necesario cambiar los nombres de los métodos de cada tipo de gráfico a los equivalentes: **plot_nb()**, **bar_nb()**, **scatter_nb()** y **hist_nb()**.

Para finalizar la sesión de trabajo en la consola donde se inició **Jupyter Notebook** presionar [**Control+C**]. Después, se puede cerrar el navegador web o las pestañas de **Jupyter Notebook**.

Relacionado:

- Gráficos en IPython

Ir al índice del tutorial de Python

Publicado por Pherkad en 12:21

Etiquetas: Python3

Entrada más reciente Inicio Entrada antigua