UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES
Centro de e-Learning

UNIDAD DIDÁCTICA VI

DIPLOMATURA EN PYTHON

Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Módulo VI – Nivel Avanzado

Unidad VIII – Socket y visualización de datos.

Presentación:

En esta unidad, comenzaremos trabajando con el módulo socket, el cual nos permite entre otras cosas, abrir una comunicación entre un cliente y un servidor remoto a partir de una dirección IP. De esta forma veremos los pasos básicos para enviar y recibir datos, creando un servidor a partir de unas pocas líneas de código.

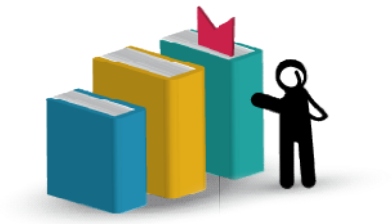
Luego presentaremos una herramienta de gran potencial en la visualización de datos 2D y 3D que nos permite trabajar con algebra lineal, transformadas de Fourier, ecuaciones diferenciales, desarrollos a partir de C /C++ y Fortran. Esta herramienta que podemos incluir dentro de una ventana de tkinter se llama Matplotlib.

Objetivos:

Que los participantes:

Comprendan los pasos necesarios para abrir una conexión entre dos dispositivos mediante ip.

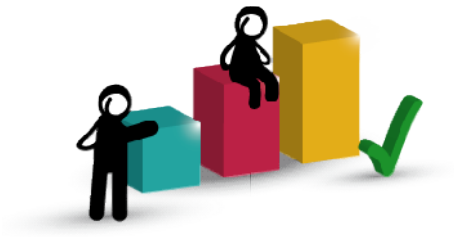
Incorporen una herramienta de visualización de datos 2D y 3D.



Bloques temáticos:

1.- Comunicación punto a punto.

2.- Visualización de datos con Matplotlib.



Consignas para el aprendizaje colaborativo

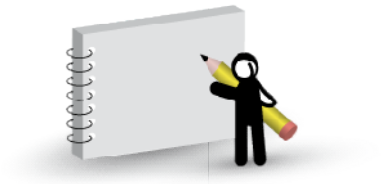
En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.



Tomen nota

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.

1.- Comunicación punto a punto.

Cuando necesitamos comunicarnos entre dos computadoras, podemos utilizar el módulo "socket" el cual viene por defecto instalado en el núcleo de la versión de python utilizada. Este módulo nos permite obtener datos de nuestro sistema operativo y de otras máquinas, así como iniciar una conversación entre dos dispositivos diferentes.

Cada dispositivo vinculado a internet, posee una dirección IP, esta dirección es como la dirección de cada dispositivo en la red, la cual es considerada como un nodo de la red. Hasta hace poco el protocolo utilizado para establecer las direcciones era el IPv4, el cual establece a cada dirección como una secuencia de cuatro grupos de tres números, los cuales en definitiva son un número de 32 bits segmentado en grupos de ocho bits (xxx.xxx.xxx.xxx). Una ipv4 podría verse de la siguiente forma:

Ipv4 = 192.168.0.192

Esta ipv4 pasada a bytes de 8 bits se vería así:

Byte 1								Byte 2								Byte 3								Byte 4							
192								168								0								192							
1	1	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0

Al trabajar con ivp4 tenemos la posibilidad de asignar direcciones ip en el orden de los mil millones, existiendo actualmente unos 4 mil millones de direcciones, las cuales por más que parezcan muchas se han agotado.

Actualmente el nuevo protocolo de reemplazo, el ipv6 permite asignar una cantidad de 1 setillón de direcciones. Si comparamos la cantidad de valores asignables con cada protocolo, veremos que la diferencia es impresionante:

																								1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

El porqué de este número gigantesco lo encontramos principalmente en dos puntos:

1. La idea de que cada persona se pueda conectar a la red con su propio dispositivo - Bring your own device.

2. El pensamiento de que cada objeto sobre el planeta tierra, desde una persona a una planta o cualquier otro objeto se encuentren anexados a la red.

En las redes de datos, los dispositivos se etiquetan con direcciones IP numéricas para enviar y recibir datos a través de las redes, sin embargo recordar una dirección ip de memoria es difícil, por lo que en internet se utiliza el protocolo DNS, el cual define un servicio automatizado que hace coincidir un nombre preestablecido con una dirección de red numérica. De esta forma si queremos acceder a la página oficial de python es más fácil de recordar el nombre de la página:

www.python.org

Que su dirección IPv4:

151.101.216.223

Al nombre asociado a la ip se lo denomina dominio, el cual se convierte en una ruta que escribimos en el explorador de nuestro dispositivo. Las rutas se llaman url, y en el caso de que sean amigables se llaman uri.

El paquete socket nos provee de un par de métodos que podemos utilizar para recuperar la dirección ip de una determinada url como podemos ver a continuación:

Datos de maquina externa

La función “gethostbyname()” también nos permite acceder a la ip de un host remoto si tenemos el nombre de dominio:

socket1.py

```
1  import socket
2
3  host_remoto = 'www.python.org'
4  try:
5      print("Dirección IP: %s" %socket.gethostbyname(host_remoto))
6  except socket.error:
7      print("Error de host remoto")
```

Retorna

```
Dirección IP: 151.101.216.223
```

Como podemos imaginar, si podemos obtener información de una máquina externa también podemos hacerlo de nuestro dispositivo como sigue:

Datos de maquina

Cómo primer paso utilizaremos las funciones “gethostname()” y “gethostbyname()” para obtener el nombre de nuestro equipo y la dirección ipv4 que tiene asociada.

socket1.py

```
1  import socket
2  nombre_host = socket.gethostname()
3  print("Nombre de Host: %s" %nombre_host)
4  print("Dirección IP: %s" %socket.gethostbyname(nombre_host))
```

Retorna

```
Nombre de Host: LAPTOP-AG43ODG1T
Dirección IP: 192.168.10.02
```

TCP y UDP

Nuestro dispositivo puede tener en ejecución varios procesos, los cuales se ejecutan en un puerto específico, sin poder superponerse, es decir que dos procesos no pueden estar utilizando un mismo puerto al mismo tiempo. A modo de ejemplo cuando nos conectamos a internet y accedemos a una página lo estamos realizando a través del puerto 80, cada puerto es como si fuera una puerta de acceso a nuestro equipo. Podemos tener varios puertos abiertos de forma simultánea en un determinado servidor, en donde cada uno es destinado a una aplicación diferente.

En el tipo de conexión remota TCP podemos distinguir tres pasos:

Paso 1: Un determinado cliente solicita que se habrá una comunicación a un servidor remoto.

Paso 2: El servidor reconoce que un cliente está queriendo abrir una comunicación y solicita a su vez el establecimiento de la conexión del servidor hacia el cliente.

Paso 3: El cliente reconoce la sesión de comunicación servidor – cliente.

El protocolo TCP posee mecanismos de control de flujo de datos para determinar la cantidad de datos que el servidor de destino puede recibir y procesar con confianza.

Mientras el protocolo TCP es confiable, el UDP no implementa por si solo los mecanismos de chequeo de datos y es utilizado por ejemplo para enviar datos de audio o video en una comunicación online. A continuación se pasan varios puertos que de seguro estás utilizando en este momento y consultamos mediante “geservbyport()” el servicio asociado al puerto.

socket3.py

```
1  import socket
2  protocolo_tcp = 'tcp'
3  for puerto in [80, 25, 443, 110]:
4      print("Puerto: %s => Nombre de servicio: %s" %(puerto, socket.
   getservbyport(puerto, protocolo_tcp)))
5  protocolo_udp = 'udp'
6  for puerto in [53]:
7      print("Puerto: %s => Nombre de servicio: %s" %(puerto, socket.
   getservbyport(puerto, protocolo_udp)))
```

Retorna

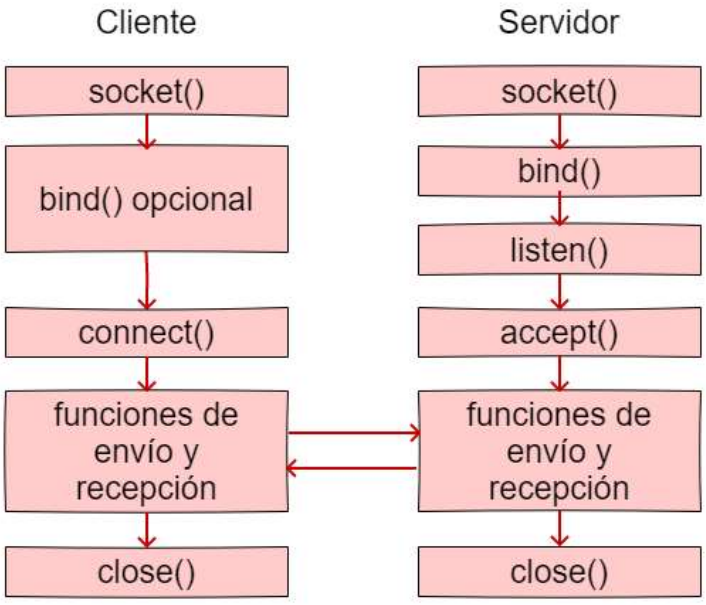
```
Puerto: 80 => Nombre de servicio: http
Puerto: 25 => Nombre de servicio: smtp
Puerto: 443 => Nombre de servicio: https
Puerto: 110 => Nombre de servicio: pop3
Puerto: 53 => Nombre de servicio: domain
```

Si en Windows quiero obtener los puertos activos y en que están ocupados puedo utilizar desde el cmd el comando:

```
netstat -oan
```

Conexión cliente servidor

Una aplicación muy interesante, es pensar en la conexión punto a punto entre dos dispositivos, en el cual uno actúa como un servidor y el otro como un cliente que puede enviarle y recibir información. La capa socket de python nos permite lograr este trabajo con muy pocas líneas de código, siguiendo una serie de pasos simples que pueden ser esquematizados de la siguiente manera:



Crearemos a continuación dos script, uno que utilizaremos como servidor y el otro como cliente y los iremos comentando paso a paso teniendo presente el esquema anterior.

Servidor.

Crearemos un servidor muy simple que cumpla con los pasos básicos, los cuales son:

Paso 1 - Creamos objeto socket mediante el método socket()

```
serversocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Aquí:

socket.AF_INET es el domino del conector. En este caso, un conector IPv4.

socket.SOCK_STREAM es el tipo del conector, y depende del parámetro anterior, no todos los dominios soportan los mismos tipos. En este caso un conector STREAM está utilizando el protocolo TCP con el cual los paquetes de datos llegan en orden descartando los repetidos y/o dañados.

Paso 2 – Una vez que se ha creado el objeto socket, podemos utilizar el método **bind()** para asociar un socket a una dirección de servidor. A este método se le deben pasar una dirección ip y un puerto de acceso.

Paso 3 – A esta altura utilizamos el método **listen()** para pone al socket servidor en modo escucha.

Paso 4 – Ahora que el servidor está escuchando utilizamos el método **accept()** de forma de aceptar una conexión entrante proveniente de un cliente. El método **accept()** nos devuelve una conexión abierta entre el servidor y el cliente, junto con la dirección del cliente. La ip del cliente es el primer componente de una tupla.

Paso 5 – Establecida la conexión, ya podemos crear funciones o rutinas de recepción y envío de datos. Para leer los datos puedo utilizar el método **recv()**, y para transmitir datos el método **sendall()** o **send()**.

Paso 6 – Finalmente para cerrar la conexión utilizamos el método **close()**.

Para ejecutar el servidor que se presenta a continuación ingresamos por línea de comandos y lo ejecutamos.

servidorTCP.py

```
1   import socket
2
3   serversocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
4   host = socket.gethostname() #Esta es la IP del servidor
5   puerto = 456 #Puerto en el cual estoy escuchado
6   print(host)
7
8   serversocket.bind((host, puerto))
9   serversocket.listen(3)
10  while True:
11      #Inicia la conexión
12      clientsocket,address = serversocket.accept()
13      print(type(address))
14      # address es una tupla de dos valores
15      print(0, '---', address[0])    #Dirección IP
```

```
16      print(1, '---', address[1])    #Número de conexión
17
18      print("Recibo la conexión desde: " + str(address[0]))
19      #Mensaje Enviado
20      mensaje = b'Hola Bienvenido a nuestro servidor' + b'\r\n'
21      clientsocket.send(mensaje)
22      clientsocket.close()
```

Cliente

En la creación del cliente realizamos los siguientes pasos:

Paso 1 - Creamos objeto socket

PASO 2 (OPCIONAL) – Utilizamos el método bind()para asociar un socket a una dirección de servidor, pasándole una dirección y un puerto.

Paso 3 – Utilizamos el método connect() de forma de fijar una conexión remota.

Paso 4 – Establecida la conexión, ya podemos crear funciones o rutinas de recepción y envío de datos. Para leer los datos puedo utilizar el método **recv()**, y para transmitir datos el método **sendall()** o **send()**.

Paso 5 – Finalmente para cerrar la conexión utilizamos el método **close()**.

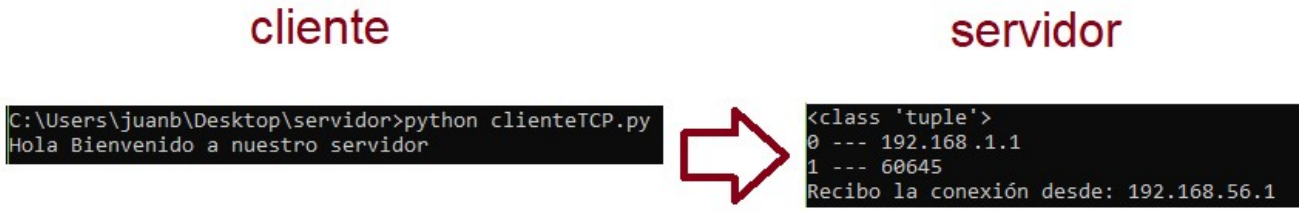
Para ejecutar el servidor que se presenta a continuación ingresamos por línea de comandos.

clienteTCP.py

```
1   import socket
2
3   clientsocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
4   host = '192.168.20.1'
5   #host = socket.gethostname() #esta es la dirección ip del servidor.
6   puerto = 456
7   clientsocket.connect((host, puerto))
8   mensaje = clientsocket.recv(1024)
9   clientsocket.close()
10  print(mensaje.decode('ascii'))
```

Nota: Hay que tener en cuenta que si estamos ejecutando el cliente y el servidor en el mismo sistema operativo es necesario tener abiertas dos terminales.

Nota: Para recuperar la ip de nuestro sistema operativo podemos hacerlo con el comando **ipconfig** si estoy en Windows o **ifconfig** si nos encontramos en Linux.



2.- Visualización de datos con Matplotlib.

Cuando debemos visualizar datos en python tenemos varias opciones, como:

- Pygal: Es una muy buena librería para crear gráficos en 2D.
- Processing: Es una plataforma que en inicio fue desarrollada en JAVA pero que fue extendida a javascript y python. Esta plataforma permite el desarrollo de videojuegos, realidad aumentada, animaciones y un sinfín de aplicaciones utilizadas por desarrolladores gráficos y programadores de todo el mundo.
- Matplotlib: Es una excelente librería para realizar representaciones 2D y 3D.

Nosotros realizaremos una introducción a matplotlib ya que cuenta con la opción de crear gráficos en 2D y 3D, y se encuentra mejor documentada que la opción de processing, cuya documentación es excelente en JAVA pero en python aún está en desarrollo.

Instalación

Para instalar Matplotlib ejecutamos por línea de comandos:

```
python -m pip install -U matplotlib
```

Matplotlib requiere las siguientes dependencias:

Python (>= 3.5)

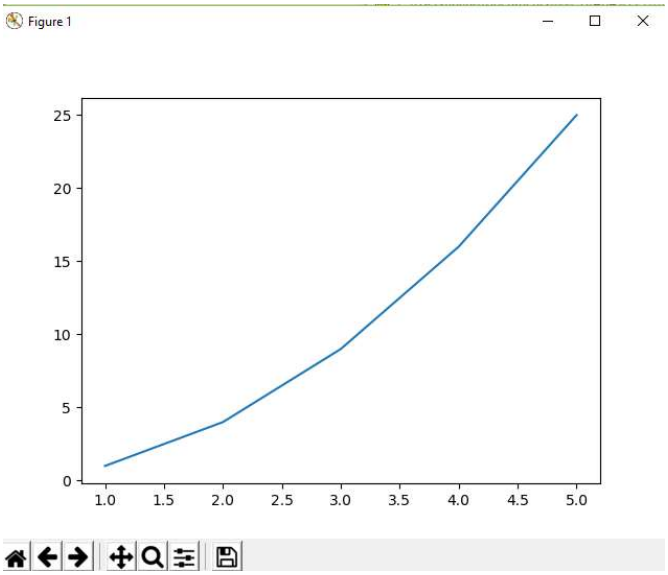
- FreeType (>= 2.3)
- libpng (>= 1.2)
- NumPy (>= 1.10.0)
- setuptools
- cycler (>= 0.10.0)
- dateutil (>= 2.1)
- kiwisolver (>= 1.0.0)
- pyparsing

Ejemplo - Gráfico de líneas

Utilizar la librería es un poco complicado al inicio es conveniente comenzar con un ejemplo simple de gráfico de líneas en el cual utilizamos la función **'plot()'** para graficar sobre un eje cartesiano 5 puntos determinados por pares '**x,y**' en donde las **'x'** vienen dadas por la primer lista y las '**y**' por la segunda. Luego utilizamos la función **'show()'** para visualizar el resultado, el resultado se muestra en una ventana emergente:

matplotlib1.py

```
1  import matplotlib.pyplot as plt
2  plt.plot([1,2,3,4,5],[1, 4, 9, 16, 25])
3  plt.show()
```



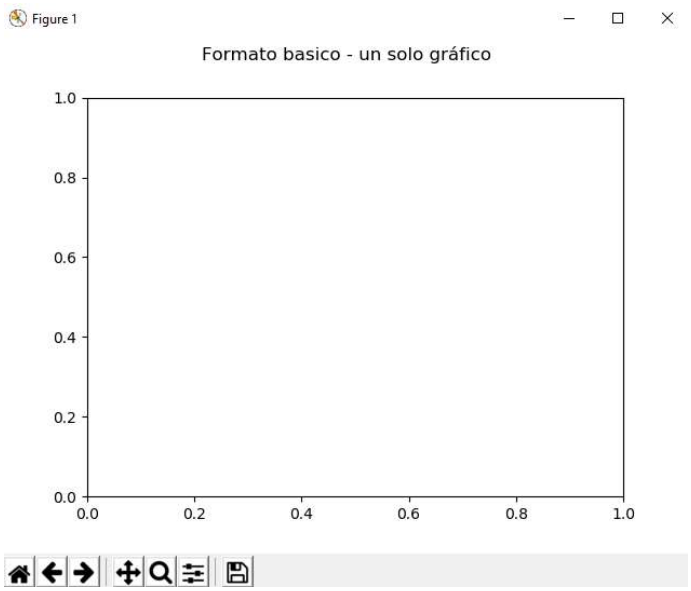
figure

Con figure podemos determinar el lienzo sobre el cual crearemos nuestro gráfico, permitiéndonos agregar datos a los ejes, títulos, subtítulos, etc. La figura permite crear cualquier número de ejes, pero como es obvio debemos especificar por lo menos uno. En los siguientes dos ejemplo creamos los ejes para un gráfico y para cuatro gráficos simultáneamente.

Un gráfico

matplotlib2.py

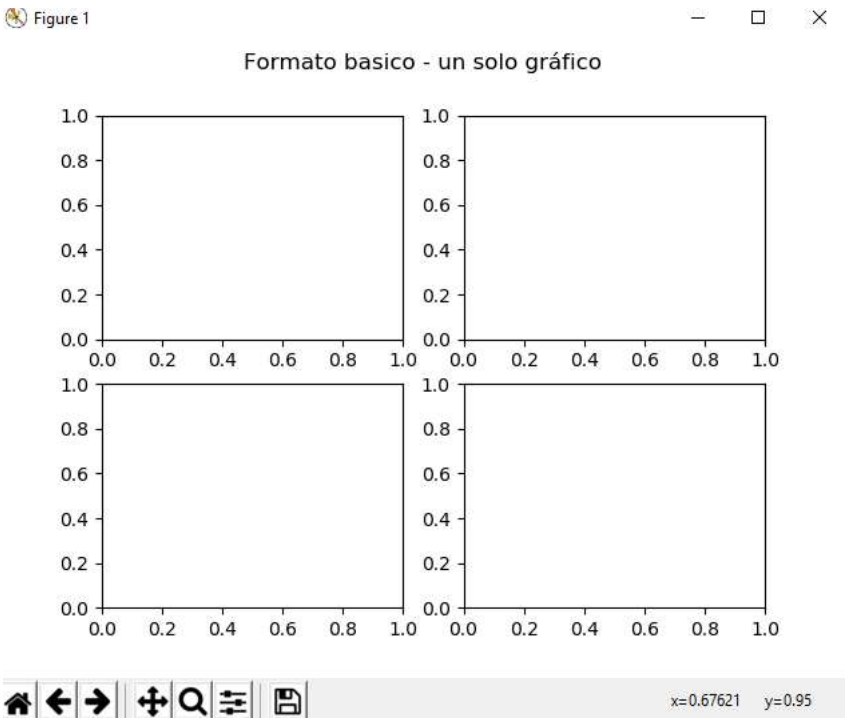
```
1  import matplotlib.pyplot as plt
2
3  fig = plt.figure()
4  fig.suptitle('Formato basico - un solo gráfico')
5  axes = fig.subplots(nrows=1, ncols=1)
6
7  plt.show()
```



Cuatro gráficos

matplotlib2.py

```
1  import matplotlib.pyplot as plt
2
3  fig = plt.figure()
4  fig.suptitle('Formato basico - un solo gráfico')
5  axes = fig.subplots(nrows=2, ncols=2)
6
7  plt.show()
```



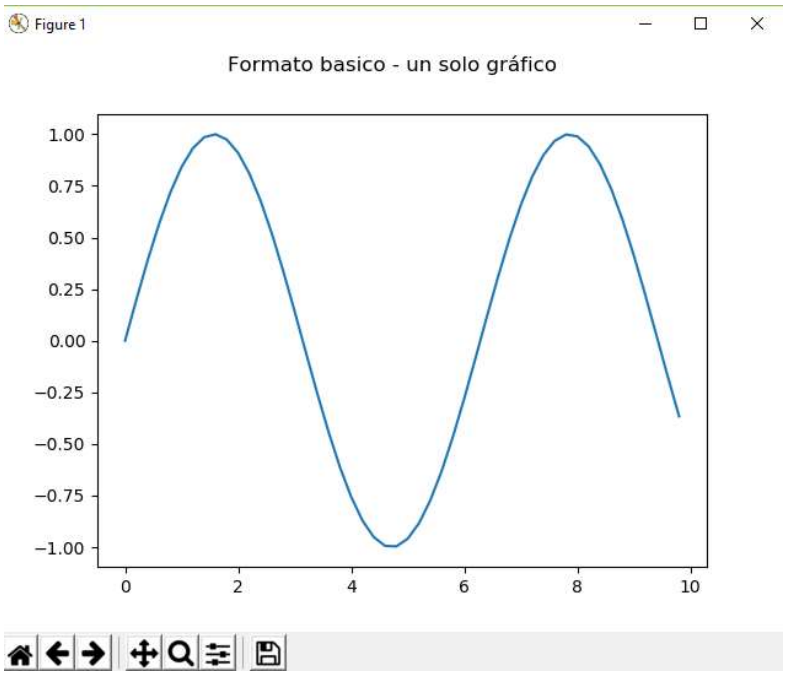
numpy

NumPy es un paquete fundamental para cuando queremos trabajar con función y datos científicos con python y contiene herramientas para integrar código de C / C ++ y Fortran. También entre otras cosas útiles nos permite trabajar con herramientas de algebra lineal y transformadas de Fourier.

En el siguiente ejemplo lo utilizaremos para crear datos para el eje x, especificando que inicie en cero, termine en 10 y genere puntos cada 0,2. En la figura hemos utilizado también la función sin(x).

matplotlib4.py

```
1   import matplotlib.pyplot as plt
2   import numpy as np
3   fig = plt.figure()
4   axes = fig.subplots(nrows=1, ncols=1)
5   fig.suptitle('Formato basico - un solo gráfico')
6   x = np.arange(0, 10, 0.2)
7   y = np.sin(x)
8   axes.plot(x, y)
910 plt.show()
```



Notemos que los siguientes códigos son equivalentes:

```
fig = plt.figure()
axes = fig.subplots(nrows=1, ncols=1)
fig.suptitle('Formato basico - un solo gráfico')
```

Y

```
fig, axes = plt.subplots(nrows=1, ncols=1)
fig.suptitle('Formato basico - un solo gráfico')
```

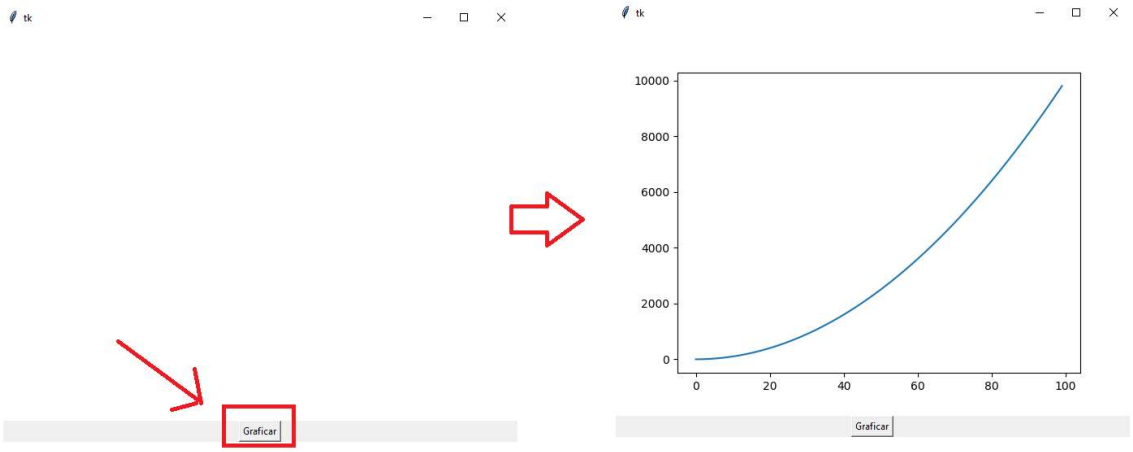
Matplotlib dentro de Tkinter

Algo interesante y muy útil es incorporar matplotlib a tkinter, para lo cual tendremos que hacer unas pequeñas modificaciones en la forma de renderizar el gráfico, utilizando el método de matplotlib FigureCanvasTkAgg para incorporar la figura a tkinter y 'canvas.draw()' para agregar la figura a un canvas.

matplotlib5.py

```
1   import numpy as np
2   import tkinter
3   import matplotlib
4   matplotlib.use('TkAgg')
5   from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
6   import matplotlib.pyplot as plt
7
8   Ventana = tkinter.Tk()
9
10  def GRAFICO():
11      x = np.arange(100)
12      y = x**2
13      plt.plot(x,y)
14      fig.canvas.draw()
15  #creamos la ubicación del grafico
16  fig = plt.figure()
17  FIGURA = FigureCanvasTkAgg(fig, master=Ventana)
18  FIGURA.get_tk_widget().grid(row=0, column=0)
19  #Creamos un boton para realizar el gráfico
20  tkinter.Button(Ventana,text="Graficar", command=GRAFICO).grid(row=1, column=0)
21  Ventana.mainloop()
```

El gráfico es activado presionando un botón en la ventana de tkinter:



Bibliografía utilizada y sugerida

Libros

Programming Python 5th Edition – Mark Lutz – O'Reilly 2013

Programming Python 4th Edition – Mark Lutz – O'Reilly 2011

Manual online

https://docs.python.org/3.7/tutorial/

https://docs.python.org/3.7/library/index.html

https://wiki.python.org/moin/HowTo/Sockets

https://matplotlib.org/tutorials/index.html

Lo que vimos

En esta unidad hemos trabajado con el módulo socket utilizado en comunicación de red y con la aplicación Matplotlib utilizada en la visualización de datos.