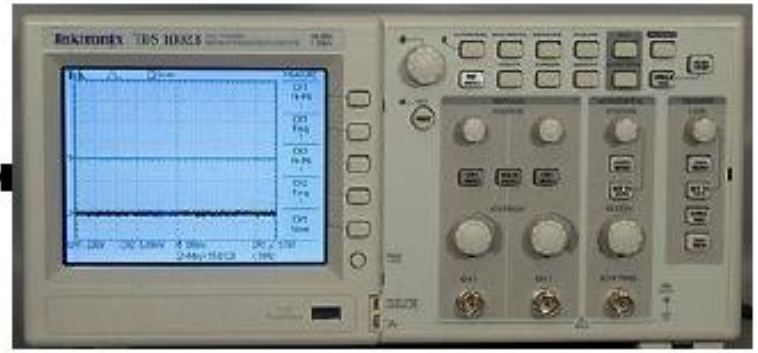
Comunicación con instrumentos

USB

GPIB

Osciloscopio TDS1002b, interface USB



EXP.

Python

Sesión VISA

Objeto USB-VISA

Instrucción específica del osciloscopio (`DATA:SOURCE CH1`)

Objeto GPIB-VISA

Instrucción específica del multímetro (`MEAS:CURR:DC?`)

Hago algo con los datos

SCPI

Multímetro de banco Agilent 34401A, interface GPIB

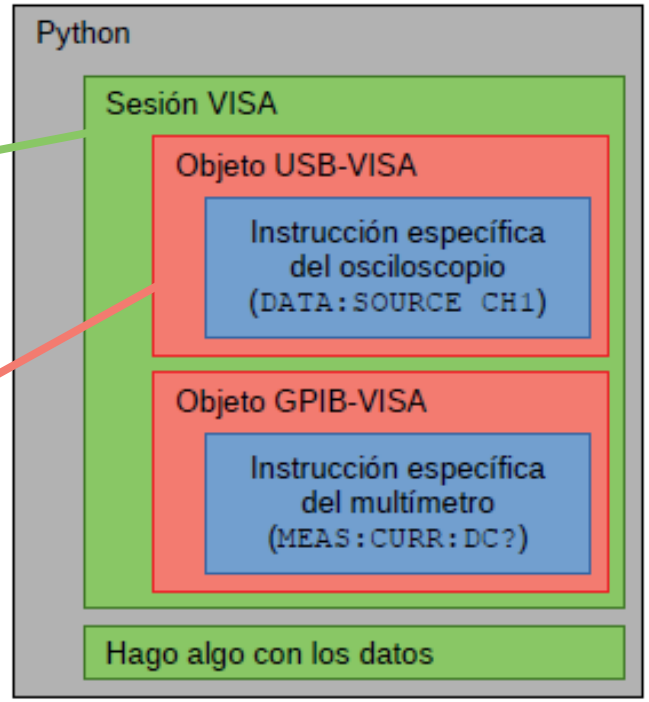
EXP.

Comunicándonos con Python

Usamos el paquete PyVisa
([https://pyvisa.readthedocs.io/en/latest/)](https://pyvisa.readthedocs.io/en/latest/)

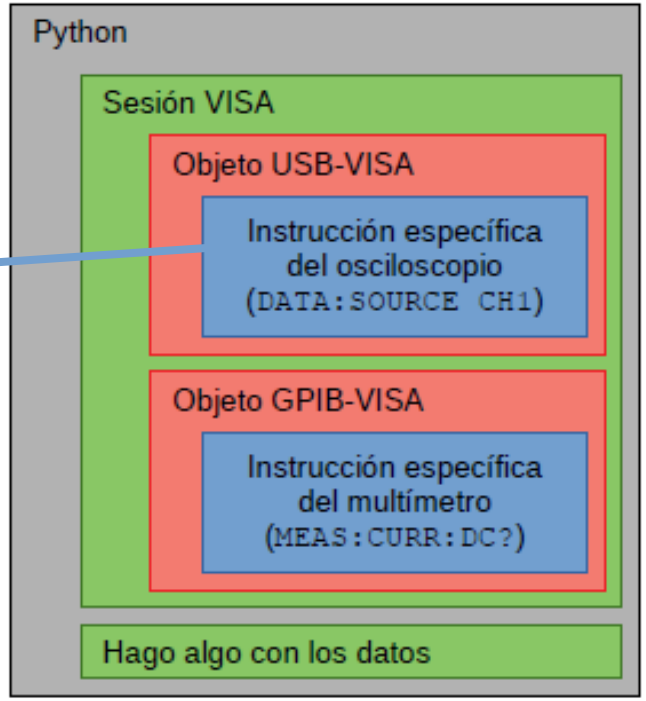
Ejemplo de comunicación con un instrumento

```
>> import pyvisa as visa

>> rm = visa.ResourceManager()

>> instrumentos = rm.list_resources()
   print(instrumentos)
       ('USB0::0x0699::0x0363::C065093::INSTR', 'ASRL4::INSTR',
       'ASRL5::INSTR')

   #Con ese nombre abro el vinculo con el osciloscopio

>> osc=rm.open_resource(instrumentos[0])
   #osc=rm.open_resource('USB0::0x0699::0x0363::C065093::INSTR')
```



osc es un **objeto** que representa a la comunicación con el instrumento

¿Qué comandos entiende VISA?



write: le enviamos un mensaje al instrumento

```
In [13]: osc.write('MEASU:MEAS3:SOURCE CH2')
Out[13]: 24

In [14]: osc.write('MEASU:MEAS3:TYPE CRMs')
Out[14]: 23
```

read: si el instrumento mandó un mensaje, lo leemos
(si no mandó nada, obtendremos un timeout)

```
In [16]: osc.read
Out[16]: <bound method MessageBasedResource.read of
<'USBInstrument'('USB0::0x0699::0x0363::C065093::0::INSTR')>>
```

query: enviar mensaje y leer respuesta luego (**write** y **read** combinados)

```
In [17]: osc.query('*IDN?')
Out[17]: 'TEKTRONIX,TDS 1002B,C065093,CF:91.1CT FV:v22.11\n'
```

¿Qué comandos entiende VISA?

```
In [18]: osc.query('MEASU:MEAS1:VAL?')
Out[18]: '7.99999982E-2\n'
```

Podemos especificar el encoding

- **query_ascii_values**
- **query_binary_values**

```
In [19]: osc.query_ascii_values('MEASU:MEAS3:VAL?')
Out[19]: [0.0282842703]

In [20]: CRMS1=osc.query_ascii_values('MEASU:MEAS3:VAL?')

In [21]: CRMS1?
Type:         list
String form: [0.0163299311]
Length:       1
```

¿Qué comandos entiende el instrumento?

¡LEER SU MANUAL DE PROGRAMACIÓN!

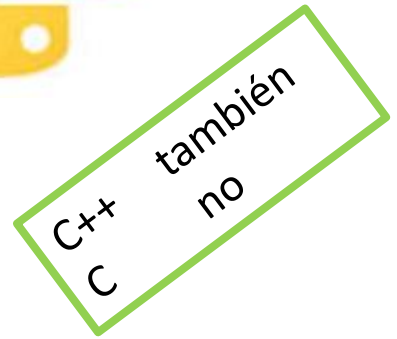
MEASUrement:IMMed:SOUrce[1]	Set or query the channel for immediate measurement

Ej.: osc.write('MEAS:INM:SOU CH1')

MEASUrement:IMMed:TYPe	Set or query the immediate measurement to be taken
MEASUrement:IMMed:UNIts?	Return the immediate measurement units
MEASUrement:IMMed:VALue?	Return the immediate measurement result

http://materias.df.uba.ar/l4a2021c2/files/2021/02/TBS1000-B-EDU-TDS2000-B-C-TDS1000-B-C-EDU-TDS200-TPS2000-Programmer_EN-US-RevA.pdf

“Clases” en Python (intro no exhaustiva!)



- Python es un lenguaje de programación **orientado a objetos**

  Object Oriented Programming (OOP)

¿Qué significa?

Que tiene funcionalidades que le dan ciertas ventajas para organizar el código.

Sirve para vincular **datos** con comportamiento/**funcionalidad** (veremos ejemplos más adelante)

https://docs.python.org/3/tutorial/classes.html

¿De verdad está orientado a objetos?

¡Desde el principio!

```
>>> a = 1
>>> type(a)
<class 'int'>
```

```
>>> b = [1,2,3]
>>> type(b)
<class 'list'>
```

O sea, tanto a como b son **instancias** de las **clases** "Entero" y "Lista" respectivamente

→ Eso es lo que define a a y b como **objetos**

OBJETO = INSTANCIA DE UNA CLASE

Por ejemplo, puedo tener una clase "Instrumento", que uso para crear los objetos "Osciloscopio" o "Generador"

En esencia, si hay Clases, entonces está orientado a objetos

Una Clase por dentro

- Cuando creo una clase, creo un nuevo **TIPO de objeto**.

- Una clase es una estructura más **compleja** que un simple tipo de dato.
- Agrupa **atributos** (datos y métodos) en una unidad conceptual.

```
7   class Perro:
8       """Este es el docstring"""
9       edad = 5
10
11      def crecer(self):
12          self.edad += 1
13          return "Me pediste que haga esto."
14
```

Attributes

Dato (*attribute* o *property* o *data o variable*) "Variables"

Método (*method*) "Funciones"

DATOS

FUNCIONALIDAD

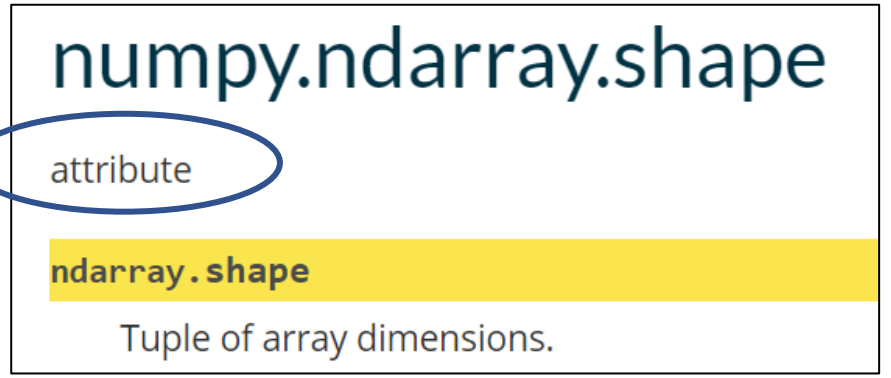
```
15  pupy = Perro()
16  print(pupy.__doc__)
17  print(pupy.edad)
18  print(pupy.crecer())
19  print(pupy.edad)
```

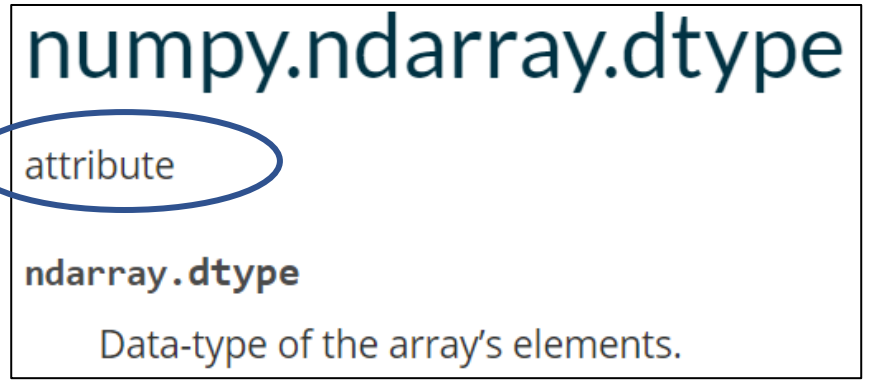
```
Este es el docstring
5
Me pediste que haga esto.
6
```

Accedo a los atributos con un PUNTO: >> objeto.dato **o bien** >> objeto.metodo()

Un ejemplo: ndarray

```
>>> x = np.array([[1, 2, 3], [4, 5, 6]], np.int32)
>>> type(x)
<class 'numpy.ndarray'>
>>> x.shape
(2, 3)
>>> x.dtype
dtype('int32')
```





Tomado de
https://numpy.org/doc/stable/reference/arrays.ndarray.html#the-n-dimensional-array-ndarray

Cuando creo un objeto, llamo a una clase (*instantiate an object*).
En Python, las clases suelen tener un método __init__ que es invocado por defecto en el momento en el que se crea el objeto.

→ Eso convierte a esta función un *constructor*.

```
1 class Customer:
2     def __init__(self, name, membership_type):
3         self.name = name
4         self.membership_type = membership_type
5         print("customer created")
6
7
8 c = Customer("Caleb", "Gold")
9 print(c.name, c.membership_type)
10
```

```
customer created
Caleb Gold
```

Algunos conceptos extra

- **INHERITANCE**
  - Cuando creo una clase dentro de una clase ("Subclass"), esta comparte los métodos de la clase madre (relación *child/parent*).

    Ejemplos: un electrón es un tipo de partícula subatómica

Clase Partícula Subatómica
Atributos: masa, carga, spin
Método: cambiar energía
Clase Quark
Métodos: formar núcleo
Clase Leptón

Clase Sustancia
Atributos: número atómico, composición
Métodos: calentar
Clase Gas
Métodos: condensar
Clase Líquido
Métodos: Solidificar

- **POLYMORPHISM**
  - Los métodos pueden ser adaptados a cada sub clase (cambian de forma) pero conservan mucho del método original.

Resumen/glosario

- Clase
- Objeto
- Instancia
- Atributo
- Método
- “Instanciar” instantiate
- __init__

Ejemplo: docs de PyVISA

```
>>> import pyvisa
>>> rm = pyvisa.ResourceManager()
>>> rm.list_resources()
('ASRL1::INSTR', 'ASRL2::INSTR', 'GPIB0::14::INSTR')
>>> my_instrument = rm.open_resource('GPIB0::14::INSTR')
>>> print(my_instrument.query('*IDN?'))
```

Este ejemplo ya muestra los dos principales objetivos de diseño de PyVISA: preferir la simplicidad a la generalidad y hacerlo de forma orientada a objetos.

Después de importar `pyvisa`, creamos un `ResourceManager` objeto. Si se llama sin argumentos, PyVISA preferirá el servidor predeterminado (IVI) que intenta encontrar la biblioteca compartida de VISA por usted. Si falla, volverá a pyvisa-py si está instalado. Puede verificar qué backend se usa y la ubicación de la biblioteca compartida utilizada, si es relevante, simplemente haciendo lo siguiente:

```
>>> print(rm)
<ResourceManager('/path/to/visa.so')>
```

Ejemplo: docs de PyVISA

Nota

En algunos casos, PyVISA no puede encontrar la biblioteca por usted, lo que genera un archivo `OSError`. Para solucionarlo, encuentre la ruta de la biblioteca usted mismo y pásela al constructor de ResourceManager. También puede especificarlo en un archivo de configuración como se explica en Configuración del backend .

Una vez que tenga un `ResourceManager`, puede enumerar los recursos disponibles utilizando el `list_resources` método. El resultado es una tupla que enumera los nombres de los recursos de VISA . Puede usar una sintaxis de expresión regular dedicada para filtrar los instrumentos descubiertos por este método. La sintaxis se describe en detalle en `list_resources()`. El valor predeterminado es '?*::INSTR', lo que significa que, de manera predeterminada, solo se enumeran los instrumentos cuyo nombre de recurso termina con '::INSTR' (en particular, los recursos USB RAW y los recursos TCPIP SOCKET no se enumeran). Para listar todos los recursos presentes, pase '?*' a `list_resources` .

En este caso, hay un instrumento GPIB con el número de instrumento 14, por lo que solicita `ResourceManager` que abra "'GPIB0::14::INSTR'" y asigne el objeto devuelto a *my_instrument* .

Aviso `open_resource` le ha dado una instancia de `GPIBInstrument` clase (una subclase de la más genérica `Resource`).

Aplicación: adquirir V(t) del Osciloscopio

La forma que ya vimos:

```
import pyvisa as visa
# Cargamos el Resource Manager. El manejador de recursos VISA
rm = visa.ResourceManager()
osci = rm.open_resource('USB0::0x0699::0x0363::C065089::INSTR')
print(osci.query('*IDN?'))

# Le pido algunos parametros de la pantalla, para poder escalear adecuadamente
xze, xin, yze, ymu, yoff = osci.query_ascii_values('WFMPRE:XZE?;XIN?;YZE?;YMU?;YOFF?;',
                                                   separator=';')

# Modo de transmisión: Binario
osci.write('DAT:ENC RPB')
osci.write('DAT:WID 1')

# Adquiere los datos del canal 1 y los devuelve en un array de numpy
data = osci.query_binary_values('CURV?', datatype='B', container=np.array)

voltaje =(data-yoff)*ymu+yze;
tiempo = xze + np.arange(len(data)) * xin
```

Aplicación: adquirir V(t) del Osciloscopio

La nueva (creamos una Clase TDS1002B):

TDS1002B es una Clase en el archivo instrumental.py de ese directorio

```
7   from instrumental import TDS1002B
8
9   #osciloscopio
10  osci = TDS1002B('USB0::0x0699::0x0363::C102223::INSTR')
11  osci.get_time()
12  osci.set_time(scale = 1e-3)
13  osci.set_channel(1,scale = 2)
14  tiempo, data = osci.read_data(channel = 1)
15
```

Acá se ve el poder de haber creado una clase específica:

→ Está escondido cómo se implementa la comunicación para obtener los valores (**ENCAPSULATION**)

→ Por ejemplo, los comandos de control, las cuentas para obtener el voltaje y el tiempo, etc.

→ Es más fácil de leer y programar

→ PERO require conocer la clase TDS1002B para poder usarla

Aplicación: adquirir V(t) del Osciloscopio

La Clase TDS1002B

```python
class TDS1002B:
    """Clase para el manejo osciloscopio TDS2000 usando PyVISA de interfaz
    """

    def __init__(self, name):
        self._osci = visa.ResourceManager().open_resource(name)
        self._osci.query("*IDN?")

        #Configuración de curva

        # Modo de transmision: Binario positivo.
        self._osci.write('DAT:ENC RPB')
        # 1 byte de dato. Con RPB 127 es la mitad de la pantalla
        self._osci.write('DAT:WID 1')
        # La curva mandada inicia en el primer dato
        self._osci.write("DAT:STAR 1")
        # La curva mandada finaliza en el último dato
        self._osci.write("DAT:STOP 2500")

        #Adquisición por sampleo
        self._osci.write("ACQ:MOD SAMP")

        #Seteo de canal
```

(...SIGUE...)

En la pág. de la materia (link): Scripts para control de instrumentos > python > instrumentos.py

Aplicación: adquirir V(t) del Osciloscopio

```
import pyvisa as visa
# Cargamos el Resource Manager. El manejador de recurs
rm = visa.ResourceManager()
osci = rm.open_resource('USB0::0x0699::0x0363::C065089
print(osci.query('*IDN?'))

# Le pido algunos parametros de la pantalla, para pode
xze, xin, yze, ymu, yoff = osci.query_ascii_values('W
                                                   sep

# Modo de transmisión: Binario
osci.write('DAT:ENC RPB')
osci.write('DAT:WID 1')

# Adquiere los datos del canal 1 y los devuelve en un
data = osci.query_binary_values('CURV?', datatype='B',

voltaje =(data-yoff)*ymu+yze;
tiempo = xze + np.arange(len(data)) * xin
```

```
class TDS1002B:
    """Clase para el manejo osciloscopio TDS2000 usando
    """

    def __init__(self, name):
        self._osci = visa.ResourceManager().open_resourc
        self._osci.query("*IDN?")

        #Configuración de curva

        # Modo de transmision: Binario positivo.
        self._osci.write('DAT:ENC RPB')
        # 1 byte de dato. Con RPB 127 es la mitad de la
        self._osci.write('DAT:WID 1')
        # La curva mandada inicia en el primer dato
        self._osci.write("DAT:STAR 1")
        # La curva mandada finaliza en el último dato
        self._osci.write("DAT:STOP 2500")

        #Adquisición por sampleo
        self._osci.write("ACQ:MOD SAMP")

        #Seteo de canal
```

(...SIGUE...)