19/3/2019

CLASE 1 : git

* git config --global user.name "Guybrush Threepwood"
* git config --global user.email guybrush@example.com

**-configura el usuario realizando los comandos. sirve porque algunas notificaciones respecto de lo que haga el usuario en los repositorios te las manda el git al email indicado.**

* git init

**crea un repositorio local en la carpeta local en donde la terminal esté ubicada.**

* ssh-keygen

**genera 2 archivos:**

**1- nombre-pub es una clave ssh para vincular la terminal local a un servidor git de forma segura, encriptada, debe agregarse a la cuenta de github menu settings>SSH and GPG keys**

**2- otro archivo de mismo nombre sin extensión que es la *private key***

**éste ultimo debe agregarse al agente ssh local mediante los comandos:**

eval $(ssh-agent -s)

**eso devuelve algo como “>Agent pid 59566” donde el nro será distinto en gral.**

ssh-add ~/.ssh/[nombre archivo sin extensión]

**esto último asocia la private key…si éste archivo está en la carpeta actual, ignorar el “**~/.ssh/”

cd ~/.ssh **cambia la ubicación a la subcarpeta que contiene la clave ssh creada**

cat id rsa .pub **muestra la clave creada**

* git clone [URL]

**copia localmente un repositorio remoto que tenga la URL indicada**

Ejercicio (ejemplo): git clone [git@gitlab.com](mailto:git@gitlab.com):talleres-comcom/taller-git-ejercicio.git

* git add [filename]

**pasa el archivo “filename” al estado *staged* en el repositorio actual. o sea se lo presentamos al git.**

Ejemplo: git add archivo.txt

* git commit -m [mensaje]

**confirma todos los cambios de los archivos en estado *staged* y los agrega a un “log” o vitácora con el mensaje indicado entre comillas.**

Ejemplo: git commit -m “cambié tal cosa en tales archivos”

* git commit --amend -m [nuevo mensaje]

**arregla un mensaje mal hecho**

* git revert [hash]

**Buscamos el hash del commit a cambiar usando** git log **y usamos este comando**

* git stash

**Se guarda el estado actual de los archivos**

**modificados y nos deja el directorio limpio.**

**Para volver a mostrar los cambios guardados ejecutamos git stash**

**apply.**

* git rm [archivo]

**preborra un archivo y marca esta acción como staged.**

* git mv [archivo] [nuevo nombre/ubicación]

**pre - mueve/renombra archivo y marca como staged**

* git diff

**Para esto tenemos el comando git diff, que muestra las diferencias**

**entre el estado actual de los archivos y la ´ultima vez que hicimos git add**

**(cambios marcados como staged).**

**Si, en cambio, queremos ver las diferencias entre los cambios marcados**

**como staged y los que confirmamos en el ´ultimo commit, podemos usar**

**git diff --staged**

* git log

**muestra el historial de cambios**

* git branch

**crea una rama en el repositorio**

* git checkout [rama]

**cambia a la rama indicada**

* git merge [nombre de la rama a fusionar]

**fusiona la rama actual la indicada. cambia sólo la rama actual**

* git status

**muestra el estado en el que se encuentran los archivos en el repositorio actual**

ADMINISTRACIÓN DE REPOSITORIOS REMOTOS

* git remote add [nombre repositorio] [URL]

**agrega un repositrio remoto y lo vincula al local.**

* git remote show [repositorio remoto]

**Devuelve información del repositorio remoto indicado, si no se pone nada te dice los nombres de los repositorios remotos asociados.**

* git remote -v

**muestra todos los repositorios remotos asociados**

* git push/pull [remoto] [branch]

**envía/recibe al repositorio remoto los cambios *staged* de archivos locales presentes en el repositorio remoto**

**si se queja porque dice que hay cosas en el remoto que no tenemos y quizá queramos le podemos forzar el push con**

git push -f [repo] [rama]

26/03/19

CLASE 2: COMUNICACIÓN CON INSTRUMENTOS

En el editor raro de mensajes de commit del GitBash, escribir el mensaje después de las líneas con # y rematarlo con :qw o :wq (probar)

Pyvisa **requiere** de NI VISA instalado para funcionar, porque el NI VISA es el que realmente se comunica directamente con los instrumentos

python -m visa info “Para ver los backends instalados mediante línea de comando en la terminal.”

pip install pyvisa “Instala por terminal.”

USB::fabricante::modelo::nroSerie::INSTR “Es el estándar para identificar un dispositivo USB.”

query\_ascii\_values “Comando del paquete pyvisa”

Con este comando tratamos de levantar una tira de datos, este comando debería transformar bien una serie de datos con encoding ascii.

query\_binary\_values

Y luego tratar de obtener la misma tira usando el instrumento con encoding en binario, puesto que recibir los datos así es más eficiente para la transferencia. Este comando de pyvisa lo transforma automáticamente a números de base 10.

(Por ejemplo, en binario, 255 se escribe usando 1 byte y en ascii son 3. Ascii mapea del 1 al 255 un montón de símbolos, letras y números y sirve porque es más legible que el binario…) **Preguntar esto**

\***Preguntar a Hernán**

-por lo que leí el osciloscopio trabaja con bytes de 8 bits….o sea que quizá Hernán dijo que un número cualquiera pesa 1 byte en binario y que se representa con 8 bits, que son 8 zeros y unos…

Pero en ASCII por ejemplo, que significa que trabaje con “datos de 1 byte”, que éstos pueden variar de -128 a 127 ¿?? En ASCII cada carácter es un nro binario de 8 bits, y por eso 123 son 3 bytes? Si es así un “dato de 7 caracteres” pesaría más de 1 byte violando que sean “datos de 1 byte”.

Manual de programación del osciloscopio:

pág 2-40 del manual

“Internally, the oscilloscope uses one 8-bit data byte to represent each waveform data point, regardless of the acquisition mode.

The DATa:WIDth command lets you specify the number of bytes per data point when transferring data to and from an oscilloscope. This provides compatibility with other digitizing oscilloscopes.”

El osciloscopio trabaja internamente con 8 bits = 1 byte = 1 dato.

**ASCII:** “One-byte-wide data ranges from –128 to 127. Two-byte-wide data ranges from –32768 to 32767.”

O sea que datos de 1 byte de 8 bits permiten 2^8 = 256 =128\*2 (incluye el 0) combinaciones o valores distintos del dato.

Y datos de 2-bytes sería eso al cuadrado: 256^2=32768\*2 OK

“Each data value requires two to seven characters. This includes one character for the minus sign if the value is negative, one to five ASCII characters for the waveform value, and a comma to separate data points”

Esto no entiendo, dice que cada dato llega a 7 caracteres...pero 1 caracter = 8 bits , entonces si el osciloscopio trabaja con datos de 1 byte de 8 bits, ¿cómo puede ser que 1 dato puede tener 7 caracteres componentes que pueden valer 2^8 cosas distintas cada uno en principio! particularmente la memoria interna no lo permite: “7 caracteres: 5 son números del 0 al 9; 1 es el signo (2 valores +/- o 1/0); 1 es una coma (todos de 8 bits cada 1)”

=> 10^5\*2+1=200001 valores en total….no me cierra

Pág 2-248

“Y specifies a normal waveform where one ASCII or binary data point is transmitted for each point in the waveform record.

For Y format, the time (absolute coordinate) of a point, relative to the trigger, can be calculated using the following formula. N ranges from 0 to 2499.

**Xn = XZEro + XINcr (n -- PT\_OFf)**

For Y format, the magnitude (usually voltage, relative to ground)

(absolute coordinate) of a point can be calculated:

For Y format, the magnitude (usually voltage, relative to ground)

(absolute coordinate) of a point can be calculated:

**Yn = YZEro + YMUIty (yn -- YOFf)**” **(nos faltaron estos datos para la transformación)**

**XINcr:**

The set form of this command specifies the interval (seconds per point for non-FFT, Hertz per point for FFT) between samples of the reference waveform specified by the DATa:DESTination command.

The oscilloscope uses this value to calculate the seconds/division or Hertz/division units shown in the status bar and cursor readouts when displaying a reference waveform.

**XUNit**

For all model and firmware combinations except the TDS200 series with a TDS2CMA communications module, the set form of this command specifies the horizontal units (”s” for seconds and “Hz” for Hertz) for the reference waveform specified by the DATa:DESTination command. Setting a reference waveform to Hz causes the oscilloscope to display the waveform as an FFT waveform.

**XZEro**

The set form of this command specifies the position, in XUNits, of the first sample of the reference waveform specified by the DATa:DESTination command, relative to the trigger.

**YMUlt** is a value, expressed in YUNits per digitizer level, used to convert waveform record values to YUNit values using the following formula (where dl is digitizer levels):

**value\_in\_YUNits = ((curve\_in\_dl – YOFF\_in\_dl) \* YMUlt) + YZERO\_in\_YUNits**

The set form of this command sets the vertical scale factor of the reference waveform specified by the DATa:DESTination command, expressed in YUNits per digitizing level.

**YOFf** is a value, expressed in digitizer levels, used to convert waveform record values to YUNit values using the following formula (where dl is digitizer levels):

**value\_in\_YUNits = ((curve\_in\_dl – YOFF\_in\_dl) \* YMUlt) + YZERO\_in\_YUNits**

The set form of this command stores a value for the reference waveform specified by the DATa:DESTination command. This value does not affect how the oscilloscope displays the waveform, but does affect the cursor readouts.

**YUNit**

For all model and firmware combinations except the TDS200 series with a TDS2CMA communications module or a TDS2MM measurement module, the set form of this command sets the vertical units for the reference waveform specified by DATa:DESTination.

**YZEro**

For all model and firmware combinations except the TDS210 or TDS220 oscilloscope (firmware below V 2.00) with a TDS2CMA communications module, YZEro is a value, expressed in YUNits, used to convert waveform record values to YUNit values using the

following formula (where dl is digitizer levels):

**value\_in\_YUNits =((curve\_in\_dl – YOFF\_in\_dl) \* YMUlt) + YZERO\_in\_YUNits**

YZEro is used when calculating cursor readouts.

Si el cable tiene binarios y usamos query\_ascii tira un error… **read** lee y transforma o decodifica y si no coincide la codificación no entiende lo q lee.

osci.read\_raw() me tira la lectura de la info que viene en hexadecimal.

En query\_binary\_values los parámetros tienen que ver con cómo se codifican los datos, esto es clave para poder obtener el resultado en valores humanamente legibles… en el manual de programador del osciloscopio están las codificaciones posibles.

Comentarios que había escrito en el script osci-tds1002B.py, los vuelco acá porque sino quedan olvidados en ese archivo que no nos interesará más:

#para escribirle un mensaje al instrumento:

inst.write()

#estos mensajes deben ir con una secuencia de caracteres que lo terminan "terminador" cuyo formato especifican los manuales de los instrumentos

#por ejemplo un terminador de linea puede ser CR "carriage return" o LF "line fill" o también \n

#pero el paquete pyvisa ya hace esto automaticamente

#inst.query('\*IDN?') funciona pero inst.query('\*IDN?\n') porque estaría duplicando el terminador

#query hace un write y un read a la vez

#a veces la memoria de los instrumentos queda llena con cosas pasadas, y esto traba los comandos, entonces probar reiniciar

visa.log\_to\_screen()

#habilita la informacion de que está ocurriendo mientras nos comunicamos y operamos con el aparato para poder hacer debugging por ejemplo

CLASE 3

PROGRAMACION ORIENTADA A OBJETOS

tipos de datos : integer , character

nros siempre son floats

luego tenemos arrays de esos datos

si en un for un array cambia de tamaño, eso es costoso computacionalmente porque en el orden de la memoria fisica cada vez van apareciendo desordenadamente y discontinuamente las partes del array

por eso → pre allocate

las listas tienen 2 indicadores a diferencia de los arrays, el dato y un puntero que dice donde esta en la memoria

las listas pueden contener arrays , son como cells en matlab...lists en python

las listas o tuplas son arrays pero de cosas en gral DISTINTAS

mas inteligente que usar funciones de estruct de objetos es hacer un objeto dinamico capaz de mutar y autoactualizarse--→ prog orientada a objetos

en python es mas eficiente en un for:

for n in lista

que hacer:

for n in range(length(lista))

la 2da es menos eficiente por tener q usar range y averiguar toooodos los datos para obtener el n-esimo

la lista en el 1| caso puede ser cualquier cosa iterable en el 2do no

ser iterable significa:

saber por donde empezar; saber donde seguir; decir cuando termine

siempre conviene ir definiendo variables funciones y clases cada una en su seccion de codigo por separado según esten familiarizados y no definir toda variable al pcipio por ej--→modularidad

\_\_init\_\_(self,contenido) este m{etodo es el constructor de objetos

class milista

def \_\_init\_\_(self,contenido)

self.contenido=contenido #esto dice como funciona el objeto creado

def duplicar(self)

x=MiLista([1,2,3])

print(x.duplicar())

class criculo:

def \_\_init\_\_(self,radio)

self.radio=radio #esto dice como funciona el objeto creado

def area(self):

return pi\*sel.radio\*self.radio

def perim(self):

return formula perim

class cuadrado:

def \_\_init\_\_(self,lado)

self.lado=lado

def area(self):

return self.lado\* self.lado

def perim(self):

return 4\*self.lado

def ratio(self):

return self.area() / self.perim()

si hacia:

class Forma:

def ratio(self)

return self.area() / self.perim()

class cuadrado(Forma):

def \_\_init\_\_(self,lado)

self.lado=lado

def area(self):

return self.lado\* self.lado

def perim(self):

return 4\*self.lado

si corro cuadrado.ratio(2) va a funcionar igual!! porque heredo las propiedaddes de la otra clase definida

en un for el iterador se fija en la lista y va de uno en uno desde el principio

en uno de estos creadores magicos de python itera de forma inteligente, desordenada pero eficicentemente

\_\_len\_\_(self) es el metodo magico para consultar el largo del objeto self

def \_\_len\_\_(self) :

self.largo=largo

return len(self.largo)

al hacer clases se escriben los objetos usando la convencion cammelcase, poniendo mayusculas asi GeneradorFunciones

y los metodos en minusc asi set\_measure usando \_ para separar

----------------------------------------------------------------------------------------------------------

Example:

import visa

rm = visa.ResourceManager()

Class OsciloscopioTDS1002B(Instrumento):

def \_\_inti\_\_(self, serialno):

self.serialno = serialno

self.inst = rm.open\_resource('USB0::0x069::0x0363::{}::INSTR' .format(serialno))

def idn(self):

return self.inst.query('\*IDN?')

def set\_timebase(self, seconds):

self.inst.write('HOR:DEL:SCA {}' .format(seconds))

Class GeneradorFunciones(Instrumento):

def \_\_int\_\_ (self, serialno)

self.serialno = serialno

--------------------------------------------------------------------------------------------------------------

si hay un error irrecuperable es preferible mandarlo al usuario inmediatamente para q se tope co el y no que el programa siga corriendop

usar booleanos en el codigo en vez de opciones en char con ifs….

Granularidad del programa: a veces conviene agrupar muchos metodos familiares en una funcion así no se multiplican definiciiones

--------------------------------------

CLASE 4

\_value es un atributo privado en esta convencion

propiedades o descriptores

atributos que pueden ser observados y alterados por otros objetos/metodos u usuarios

es un atributo con metodos getter y setter

“setter “

def set\_timebase(self, value)

return self.write('HOR:MAIN:SCA []'.format(value))

“getter”

def get\_timebase

return self.query('HOR:MAIN:SCA?')

c.timebase=property(get\_timebase, set\_timebase) …. Acá timebase es una propiedad del obj c con get y set

c.timebase=5 setea el timebase

print(c.timebase) llama el getter y te dice el valor

PREGUNTAR COMO SERíA la forma alternativa de definir get y set para una propiedad

CREO q es :

@property

Def timebase(self)

Return self.\_\_timebase ¿propiedad privada?

@timebase.setter

def timebase(self, value):

self.write('HOR:MAIN:SCA []'.format(value))

decoradores

los decoradores son en sí mismos funciones, que toman como argumento una función y retornan otra función. Aún más, el código anterior cumple el mismo objetivo que el siguiente.

Ejemplo:

def debug(f):

def new\_function(\*args, \*\*kwargs):

print(f"Function {f.\_\_name\_\_}() called!")

return f(\*args, \*\*kwargs)

return new\_function

@debug

def add(a, b):

return a + b

print(add(7, 5))

@deco(set\_value=True) asi recuerda por default el 1er valor asignado a la propiedad

Def func(x):

LANTZ

ver instrucciones de instalacion:

para crear un environment en anaconda:

create -m entorno

Feats (decoradores????)

mfeats. Varias opciones de hacer los sets y gets de manera abreviada, con metodos que reconocen unidades, limites, etc

EJEMPLO: creo una clase de osciloscopio Osci1 que hereda de MessageBasedDriver de la librería de Lantz y luego mas abajo uso el método via\_usb para identificar con el serial el aparato y asi usar los drivers adecuados del lantz

from lantz import MessageBasedDriver, Feat

from lantz.core import mfeats

|  |
| --- |
| class Osci1(MessageBasedDriver): |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| @Feat() |
|  |

|  |
| --- |
| def idn(self): |
|  |

return self.query('\*IDN?')

|  |
| --- |
| def get\_timebase(self): |
|  |

|  |
| --- |
| return float(self.query('HOR:MAIN:SCA?')) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def set\_timebase(self, value): |
|  |

return self.write('HOR:MAIN:SCA {}'.format(value))

|  |
| --- |
| from lantz import ureg |
| Osci1.initialize() |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| with Osci1.via\_usb('C065092') as osci: |
|  |

|  |
| --- |
| print(osci.idn) |
|  |

|  |
| --- |
| print(osci.get\_timebase()) |
|  |

|  |
| --- |
| osci.set\_timebase(0.01) |
|  |

|  |
| --- |
| print(type(osci.timebase)) |
|  |

print((1.0/osci.timebase).to('Hz'))

como lantz no abre el instrumento , entonces se hace esa linea

osci.initialize() abre la comunicacion con el aparato

osci=Osci.via\_usb('serial')

osci.initialize()

print(osci.idn)

osci.finalize() finaliza comunicacion

a veces es necesario finalizar la comunicación para liberar recursos del instrumento, eso se logra con esto

from lantz.core import log

lantz.log to socket() me abre una pantalla donde te dice todo lo q hizo el instrumento

en python: usar excepciones para q los programas nos avisen q algo salio mal o viene mal antes q reviente:

def fun(x):

if x<0:

raise Exception

return 2\*x

hay muchas clases de excepciones

acá Exception es en realidad ValueError('esto q pusiste esta mal')

tmb hay TypeError

ejemplo

if not isinstance(x, (float, int)):

raise TypeError

para tratar excepciones se usa

def external(N):

for n in range(N):

try:

fun(-n)

except ValueError: #notificar{a solo los errores de valor

print('algo paso')