INTRODUCCIÓN AL ENTORNO DE TRABAJO CON LA DAQ

Carlos Tomás Medrano Sánchez, Juan Carlos García López, Guillermo Palacios Navarro, Inmaculada Plaza García.

Objetivos: El objetivo de la práctica es conocer el entorno Linux / Python que nos va a permitir trabajar con las tarjetas de adquisición de datos.

0. Previo.

Se trabaja en una máquina virtual donde, en la siguiente práctica, podremos ejecutar un simulador de la tarjeta.

Para ello tenemos que acceder a **https://aulainformatica.unizar.es/** con nuestro NIP y contraseña; luego dentro de las máquinas de EUPT tendrás que acceder a la máquina "instrumentación" (usuario y contraseña: consultar con el profesor).

En Moodle tienes enlaces a una serie de vídeos para saber cómo trabajar con la máquina virtual y seguir esta práctica. Puedes seguir el guión, si bien los vídeos son un recurso más visual en este caso.

1. Comandos básicos en Linux.

Consola: Vamos a aprender a trabajar desde la consola. Para abrir una consola, suele estar en el menú dentro de *sistema*, o bien usar *lxterminal* (acceso en la barra inferior). Otra opción para gestionar tus ficheros es abrir el programa *pcmanfm* (es probable que tengas ya el acceso desde la barra inferior). Este navegador permite acceder también a los archivos de tu cuenta.

Nota: Otras versiones de linux poseen diferentes programas para acceder a la consola (terminal) y para gestionar los ficheros de forma gráfica.

Una vez que hayas abierto la consola, los comandos más básicos son:

- *cd* (change directory): permite cambiar el directorio en el que se está trabajando. Por ejemplo: *cd mis_practicas*. Hay que tener en cuenta que en linux el punto . representa el directorio actual, mientras que dos puntos .. representan el directorio inmediatamente superior. Para indicar rutas de varios niveles se usa la barra /.
- *pwd* (present working directory): devuelve el directorio actual.
- *ls* (list): permite ver todos los ficheros y subdirectorios del directorio actual o de la ruta que indiques como argumento.
- *man* (manual): permite ir a una página de información de un comando. Para salir de ella, pulsar la tecla 'q' (quit). Ejemplo: *man cd*, informará sobre el comando

cd. No obstante, actualmente es fácil encontrar información en internet sobre los comandos.

- *mkdir* (make directory): permite crear un directorio. Ejemplo: *mkdir practica1*. Se recomienda que los nombres de ficheros y directorios no contengan espacios ni acentos, etc.
- *cp* (copia): permite copiar un fichero con otro nombre (y en otra ruta). Ejemplo: *cp foto1.jpg misfotos/foto_amigos.jpg* .
- *mv* (move): permite mover un fichero de un lugar a otro, o cambiarlo de nombre. Ejemplo: *mv foto1.jpg foto_vieja.jpg*.
- *rm* (remove): permite eliminar ficheros. Ejemplo: *rm foto1.jpg*. Existe una opción (-rf) que permite borrar todo un directorio y su contenido completo. Ejemplo: *rm -rf direc1* borrará el directorio *direc1* y todo su contenido. Cuidado, es un comando peligroso.
- *rmdir* (remove directory): borra un directorio. El directorio debe estar vacío.
- cat (concatenate): sirve para mostrar por pantalla un fichero. Ejemplo: *cat leeme.txt*
- En muchos comandos, el símbolo * equivale a cualquier cadena de caracteres. Por ejemplo, *rm a*.pdf* borrará todos los archivos pdf que empiecen por la letra *a*.
- Las aplicaciones también se pueden lanzar desde la consola. Basta con escribir el nombre de la aplicación. Por ejemplo: *firefox, geany* (editor de textos). Si utilizas el símbolo & después del nombre, la aplicación se ejecuta en el background y vuelves a tener la consola activa sin necesidad de terminar el programa que hayas lanzado.
  A las aplicaciones también se puede acceder desde el botón izquierdo de la barra inferior.
- Algunas aplicaciones útiles:
  firefox (navegador web)
  pcmanfm (navegador web y del sistema de archivos)
  geany (editor de textos simple).

Ejercicio 1.1: Edita un fichero de texto con *geany* (es probable que tenga ya el acceso desde la barra inferior). Guárdalo e intenta desde la consola crear un subdirectorio y copiarlo en él con otro nombre.

2. Introducción a Python.

Python es una lenguaje de programación tipo script. Se puede pensar en él como algo similar a Matlab. Se pueden ir ejecutando comandos tanto desde una consola como escribir un fichero de texto con funciones y llamarlas desde la consola.

Para empezar a trabajar con Python entraremos en el entorno escribiendo *ipython* en una consola. IPython es un entorno para escribir comandos que hace más manejable su uso. Para entrar en él teclearemos *ipython*. Para salir haremos *exit()*.

Nota: cuando escribimos el símbolo '$' nos referimos a una consola de linux (en una consola puede aparecer algo similar a 'usuario@nombrePC$'). Cuando aparezca el símbolo 'In[]' o 'Out[]' es que estamos ya dentro de la consola ipython, que indica de esta manera las entradas y salidas (también entre corchetes aparece su número)

$ *ipython3*

Una vez en el entorno de *ipython*, podemos empezar a escribir cualquier instrucción de Python. Por ejemplo, podemos definir dos variables y multiplicarlas:

In []: *a=5.2*
In []: *b=4*
In []: *print(a*b)*

En Python podemos tener valores numéricos flotantes, enteros, booleanos (*True, False*). Los operadores aritméticos son los mismos que en C con alguna excepción (la exponenciación es **: por ejemplo 5^2 se escribe 5**2).

COMENTARIOS

Los comentarios empiezan con el carácter '#', o bien todo lo que sigue hasta el final de línea es un comentario:

In []: *a=5.2 # Variable a*
In []: *b=4 # Variable b*
In []: #Y ahora vamos a multiplicarlas
In []: *print(a*b)*
En ficheros de texto con funciones, podemos poner varias líneas seguidas de comentarios encerrados entre tres comillas seguidas:

"""
Comentario 1
Comentario 2
Comentario 3
"""

MATH y NUMPY:

Lo interesante de Python es la cantidad de módulos que permiten ampliar sus funcionalidades. Para utilizar un módulo es necesario importarlo. Por ejemplo, vamos a importar el módulo *math* que permite acceder a funciones matemáticas:

In[]: *import math*
In[]: *print(math.sqrt(4))*
In[]: *print(math.sin(math.pi/4))*

Por su parte el módulo numpy permita trabajar con vectores, matrices y diversas utilidades de cálculo numérico. Es convencional importarlo asociándole un nombre más corto (*np*)

In[]: *import numpy as np*

Creamos una matriz 2x3:

In[]: *m = np.array([[1, 2, 3], [4, 5, 6]])*

In[]: *print(m)*

Podemos acceder a un elemento fila, columna. La numeración empieza en 0.

In[]: *print(m[0,1])*

En Python, las variables son en su mayoría objetos (Python es un lenguaje orientado a objetos). Podemos acceder a variables de los objetos o a sus métodos usando la notación del punto. Por ejemplo, prueba a escribir *m.* en la consola y pulsa el tabulador. Te mostrará todas las posibilidades:

In[]: *m.* (pulsa el tabulador)

Por ejemplo, el método *sum* calcula la suma de todos los elementos de la matriz.

In[]: *print(m.sum())*

LISTAS:

Es muy común en Python trabajar con listas, es decir un conjunto ordenado de objetos. Por ejemplo, vamos a crear una lista de 3 enteros:

In[]: *lis = [1,2,5]*

Podemos acceder a los elementos de una lista con su índice (empiezan en 0):

In[]: *print(lis[0])*
In[]: *lis[0]*
In[]: *lis[0]=8*
In[]: *print(lis)*

y encontrar el número de elementos de la lista con la función *len()*:

In[]: print(len(lis))

Las listas tienen una serie de métodos para trabajar con ellas: *append* (añadir un elemento), *insert* (insertar), *remove* (eliminar) etc.

AYUDA

Para obtener ayuda usamos el comando *help*. Por ejemplo, para saber lo que hace el método append de la lista que acabamos de definir, podemos hacer:

In[]: *help(lis.append)*

Para SALIR de la ayuda pulsaremos la tecla 'q' (de quit).

En el entorno *ipython* funciona el autocompletado, de forma que si pulsas el tabulador, apareceran todas las opciones que tienes. Por ejemplo, si escribes *lis.* y pulsas el tabulador, te aparecerán todas las variables y métodos asociados a esa variable.

DICCIONARIOS:

Los diccionarios son objetos que permiten almacenar valores a los que se puede acceder mediante palabras clave (keywords). Por ejemplo:

In []: *dic={} # Crea un diccionario vacio*
In []: *carlos={}*
In []: *carlos['telefono']='666777888' # Añade una entrada al diccionario*
In []: *print(carlos)*
In []: *carlos['edad']=30 # Añade otro campo*
In []: juan={'telefono':555444333, 'edad':58} # Creamos otro diccionario desde cero

Ejercicio 2.1: Utiliza el autocompletado para ver los métodos asociados a una variable de tipo diccionario. Utiliza la ayuda para comprobar qué hace alguno de ellos.

CADENAS DE CARACTERES

Las cadenas de caracteres se pueden definir entre comillas simples o dobles. Por ejemplo:

In[]: *mensaje='Error en el programa'*
In[]: *print(mensaje)*

Podemos acceder a un caracter por su indice (mensaje[3] es la letra 'o'). El operador de suma concatena caracteres:

In[]: *print(mensaje + '. Parece grave')*

ESTRUCTURAS DE CONTROL: CONDICIONALES

En Python, cualquier estructura de control tiene un sangrado obligatorio (en general cualquier estructura que acabe en el carácter ':' tiene sangrado para la parte que le afecta). Esto permite saber directamente qué parte del programa se ejecuta en una condicional por ejemplo. Dentro de ipython, el propio entorno realiza automáticamente el sangrado. Cuando queramos acabar, pulsaremos la tecla 'intro' dos veces seguidas.

La condicional tiene este aspecto:

```
if (condición) :
  hacer esto
else:
 hacer lo otro
```

Trata de introducir estos comandos desde ipython y observa el resultado (para la línea *else* deberás quitar el sangrado que aparece por defecto):

In []: *x=5*
In []: *if(x<1):*

```
....:     print('x es menor que 1')
....: else:
....:     print('x es mayor o igual que 1')
....:
....:
```

ESTRUCTURAS DE CONTROL: BUCLES WHILE

Contamos también con la estructura *while*:

```
while (condición):
  hacer algo
```

Ejercicio 2.2: prueba a escribir un bucle que imprima los números del 1 al 10.

ESTRUCTURAS DE CONTROL: BUCLES FOR

Los bucles for son un poco distintos de otros programas. Se realizan sobre objetos que sean iterables, es decir, que puedan ir tomando un valor u otro como en una lista. Por ejemplo, una lista es iterable. Para los bucles más habituales podemos usar range: *range(n1,n2)* devuelve una lista que va desde n1 hasta n2-1. Así para escribir los números del 90 al 99 haremos:

```
In [25]: for n in range(90,100):
  ....:     print(n)
  ....:
  ....:
```

ESTRUCTURAS DE CONTROL: SALIDA Y CONTINUACIÓN

Dentro de cualquier bucle, el comando *break* sale del bucle más interno, mientras que *continue* pasa a la siguiente iteración del bucle. Son similares a las que existen en C.

DEFINICIÓN DE FUNCIONES

Las funciones se define con *def*. Por ejemplo, para definir una función que sume dos números prueba a hacer esto:

```
In []: def misuma(a,b):
  ....:     res=a+b
  ....:     return res
  ....:
```

Desde este momento la función *misuma* está definida y puede ser llamada:

```
In []: mysuma(8.0, 5.8)
```

FUNCIONES DEFINIDAS EN FICHEROS

Lógicamente, no resulta muy útil tener que definir cada vez las funciones para utilizarlas. Para ello, podemos escribirlas en un fichero (se pueden definir varias en cada fichero) e importarlas al principio. Para ello:

- Edita un fichero con este texto (por ejemplo con el programa *geany*) teniendo cuidado de mantener el sangrado, y guárdalo en el directorio de trabajo como *ejemplo.py*. Fíjate también en que las líneas que empiezan con '#' son comentarios. Si dentro de un fichero necesitas usar un módulo, puedes importarlo al principio del fichero con *import*.

```
def suma_multiplica(a,b):
    # Esta es la suma
    s=a+b
    # Esta es la multiplicacion
    m=a*b
    # Lo devolvemos como una lista
    return [s,m]
```

IMPORTANTE: Puedes usar 2, 4 espacios o el tabulador para realizar el sangrado, pero mantén siempre el mismo tipo de sangrado en un mismo fichero o cuando copies y pegues código de algún lugar. Algunos editores de texto realizan el sangrado automáticamente, al detectar que se trata de un código Python, y se puede configurar. Como norma general, utilizaremos 4 espacios para el sangrado

- Entonces, desde la consola de ipython, puedes importar el módulo que has creado y utilizar esa función:

```
In []: import ejemplo
In []: resul = ejemplo.suma_multiplica(5,6)
In []: print(resul)
[5, 6]
```

IMPORTANTE: Para que encuentre el módulo, el fichero .py debe estar en el directorio de trabajo desde el que has lanzado *ipython*. Si no, dentro de la consola ipython también puedes usar el comando *cd* para acceder al directorio donde esté el fichero.

IMPORTANTE: si realizas un cambio en *ejemplo.py*, debes guardarlo y volverlo a cargar para que el cambio sea visible desde la consola. Esto se hace con *reload*. OJO la propia función *reload* debe cargarse antes en la sesión:

```
In[]: from importlib import reload
In[]: reload (ejemplo)
```

Ejercicio 2.3: incluye en el fichero *ejemplo.py* una función que reciba dos variables. La primera, *n*, es un número entero. La otra, *b*, una variable booleana. Si la variable booleana es cierta, debe devolver la suma de todos los enteros hasta *n*. En otro caso, la

función debe devolver el factorial de *n* (el producto de todos los números desde 1 hasta n).

Ejercicio 2.4: Realiza también una función que recibe un número n y devuelve una lista desde 0 hasta n-1

Ejercicio 2.5: Realiza una función que recibe un número n e imprime todos los números de 0 a n de 2 en 2, y luego hace lo mismo pero hacia atrás. Entre cada salida por pantalla espera 0.5 s
Nota: el módulo time de python permite que el programa realice una espera.

```
import time
time.sleep(1.5) # espera 1.5 s
```

El módulo también tiene una función que devuelve el tiempo actual en segundos (time.time()). Puede servir para saber cuanto tiempo ha transcurrido:

```
t1= time.time()
time.sleep(2.0)
t2 = time.time()
print('He estado durmiendo', t2-t1, ' segundos')
```

Ejercicio 2.6: Realiza una función que reciba un número, s, y realice una espera activa durante s segundos, es decir un bucle del que no salga hasta que hayan pasado s segundos.

Ejercicio 2.7: Realiza una función que reciba un número n (se supone entero) y que devuelva n puntos de una función sinusoidal repartidos equitativamente entre 0 y 2π.