

PYTHON + Raspberry Pi

Introducción y Ejercicios Prácticos



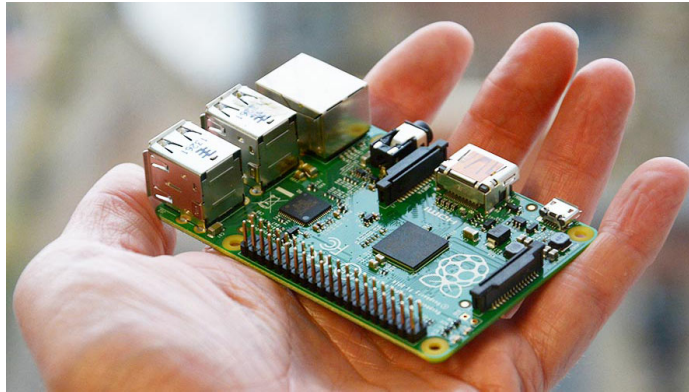
Patricio Reinoso M.
Agosto - 2016

Índice de Contenido

[0] - Introducción.....	3
[1] - Variables y Tipos.....	5
1.1.- Asignación de Variables.....	5
1.2.- Tipos de Variables.....	5
1.3.- Operadores Aritméticos Básicos.....	7
1.4.- Operadores Booleanos.....	8
[2] - Listas, Tuplas y Diccionarios.....	9
2.1.- Listas.....	9
2.2.- Tuplas.....	11
2.3.- Diccionarios.....	11
[3] - Condicionales.....	14
[4] - Bucles.....	17
4.1.- Bucles while.....	17
4.2.- Bucles for.....	18
[5] - Funciones.....	20
[6] - Manejo de Excepciones.....	22
[7] - Manejo de I/O.....	23
7.1.- Entrada desde consola.....	23
7.2.- Salida a consola.....	23
7.3.- Manejo de archivos.....	24
7.3.1.- Lectura de Archivos.....	24
7.3.2.- Escritura en Archivo.....	25
[8] - Ejecución de programas Python desde la consola.....	26
[9] - Python y la Raspberry Pi.....	27
9.1.- GPIO Básico.....	27
9.2.- PWM.....	30
9.3.- Control de Servos mediante PWM.....	31
9.4.- Sensores 1-Wire.....	32
[10] - Conexión con servicios Web.....	35
10.1.- Datos en formato JSON.....	35
[11] - Visualización de Datos.....	38
[12] - Observaciones finales.....	39

[0] - Introducción

La Raspberry Pi, a partir de su lanzamiento (e incluso desde un tiempo antes) se convirtió en un punto de inflexión en la computación actual. Debido a ciertas falencias encontradas en estudiantes de informática de algunas Universidades del Reino Unido, nació la idea de tener un computador de muy bajo costo, que pueda ser utilizado para enseñar programación a estudiantes de escuelas y colegios alrededor del mundo. Esto con el fin de reforzar los conocimientos de programación, y volver a lo básico, tal como sucedió en los 80s, con la masificación del computador personal.



Poder computacional en la palma de su mano

Este computador personal (del que inicialmente no esperaban vender mas de 10000 unidades) ha tenido tal éxito, que hasta la fecha (agosto de 2016) se encuentran muy cerca de alcanzar los 10 millones de unidades vendidas. Se preguntarán a qué se debe tanto éxito? Pues entre algunos parámetros podemos mencionar:

- Realmente es un sistema de bajo costo, con el cual uno puede trabajar sin preocuparse de “romper” un computador costoso
- Una gran comunidad de desarrolladores y de soporte técnico a nivel mundial. Varios millones de usuarios y muchísima gente participando en desarrollo y soporte técnico
- Su sistema operativo principal (Raspbian) es una distribución de Linux amigable con el usuario, y que incluye muchas aplicaciones orientadas a la educación y al desarrollo de software
- Python, que es el lenguaje principal usado en la Raspberry, es facil de aprender y muy poderoso

Lo citado, mas un “boom” en temas como Open Hardware e Internet Of Things, han ocasionado que muchisima gente use esta plataforma de hardware para las tareas mas variadas. Desde servidores web en producción, hasta consolas de juego, o pajareras. Los usos de la Raspberry son ilimitados, quedando todo en la inventiva de sus usuarios.

Este manual se centrará más en una introducción al lenguaje Python, aplicado sobre la Raspberry Pi, que a temas como instalación y configuración de la tarjeta. Para mayor información de como

descargar, grabar y configurar el sistema operativo de la Raspberry, recomiendo visitar la web <http://www.raspberrypi.org> donde se encontrará toda la información necesaria para iniciar con el hardware y el software.

Dado que este es un manual introductorio a Python, para realizar los ejercicios, se usará la consola interactiva de Python. La consola interactiva la podemos iniciar desde una ventana de terminal de Linux, ejecutando el comando:

```
python
```

y obtendremos una consola, como lo presentamos a continuación:

```
Python 2.7.12 (default, Jul 1 2016, 15:12:24)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

desde aquí ejecutaremos los comandos de manera interactiva, obteniendo respuestas inmediatas a los mismos, tal como se irá revisando en los ejercicios posteriores.

Adicional a esto, se pueden grabar los scripts de python a ejecutar como programas independientes, y para su ejecución desde una consola de terminal, basta con ejecutar:

```
python mi_programa.py
```

y obtendremos el resultado de la ejecución del programa en nuestra consola.

Algunas convenciones de formato de texto usadas en este manual, según la tipografía:

- *comando de consola (nótese la italica)*
- programa de python ejecutado en modo interactivo, o guardado en archivo
.py

Para finalizar esta introducción y pasar a temas mas interesantes, me permito mencionar:

- Tomé el 0 como numeral de Introducción, porque el primer elemento de las listas en Python es el elemento [0]
- Este manual fué escrito y revisado sobre una Raspberry Pi 3, usando Vim y LibreOffice
- Este manual, así como algunos ejercicios descritos aquí están disponibles en <http://www.patolin.com>

[1] - Variables y Tipos

1.1.- Asignación de Variables

Los nombres de variables son "Case Sensitive", es decir, diferencian mayúsculas de minúsculas.

Los nombres de variables pueden contener caracteres alfanuméricos (empezando siempre por una letra), y pueden incluir el guión bajo como único carácter especial.

Nombres de variables válidos:

- miVariable
- dato10
- dato_temperatura

Nombres de variables inválidos:

- dato-temperatura
- 10numeros
- mi!variable

Las variables se asignan mediante el operador =:

```
>>> miVariable=10
>>> miVariable
10
>>> miString="Hola mundo!"
>>> miString
'Hola mundo!'
>>> miLista=[1,2,3]
>>> miLista
[1, 2, 3]
```

1.2.- Tipos de Variables

En Python existen 3 tipos de datos básicos:

Numéricos: Enteros, Reales y Complejos

```
>>> miEntero=10
>>> miEntero
10
>>> miReal=12.34
```

```

>>> miReal
12.34
>>> miReal_1=10.0
>>> miReal_1
10.0
>>> miComplejo=2+3j
>>> miComplejo
(2+3j)

```

Nótese la diferencia entre las variables `miEntero` y `miReal_1`. A pesar de tener el mismo valor, se diferencian por la inclusión de la parte decimal en el número. Para convertir valores numéricos a cadenas de caracteres, y poder visualizarlas mediante el comando `print()`, es necesario usar el comando `str()`.

```

>>> a=10
>>> a
10
>>> b=str(a)
>>> b
'10'

```

Cadenas de caracteres

Se puede utilizar la comilla simple y doble para definir cadenas de caracteres. Adicionalmente, se puede usar una triple comilla doble, para definir texto multilínea.

```

>>> miStringSimple='Hola mundo!'
>>> miStringSimple
'Hola mundo!'
>>> miStringDoble="Hola mundo!!"
>>> miStringDoble
'Hola mundo!!'
>>> miStringMultilinea=""" Esto es
... una prueba
... de un string multilinea"""
>>> miStringMultilinea
' Esto es \nuna prueba\nde un string multilinea'

```

Se puede utilizar codificación Unicode o Raw, para las cadenas de caracteres. Esto se realiza colocando `u` ó `r` antes de la cadena. Ejm.

```

>>> a=u'Esto es \n una prueba'
>>> print a
Esto es
  una prueba
>>> b=r'Esto es\n una prueba'
>>> print b
Esto es\n una prueba

```

Nótese la diferencia entre la impresión (mediante el comando print) de las variables creadas. En la variable a, el salto de línea se aplica, al estar la cadena codificada en Unicode, mientras que en la variable b, al estar en raw, toma los valores tal cual se guardan en el string.

Booleanos

Pueden ser True o False.

```
>>> verdadero=True
>>> verdadero
True
>>> falso=False
>>> falso
False
```

1.3.- Operadores Aritméticos Básicos

Python permite el uso de los siguientes operadores aritméticos:

- + Suma
- - Resta
- * Multiplicación
- / División
- // División entera
- ** Exponente
- % Módulo

```
>>> a=3+2
>>> a
5
>>> b=3-2
>>> b
1
>>> c=3*2
>>> c
6
>>> d=3/2
>>> d
1
>>> e=3/2.0
>>> e
1.5
```

Nótese que dependiendo del tipo de dato en cada variable, el resultado de la operación es distinto. Si comparamos los valores de las variables d y e, notamos que en la variable d, la división se realiza entre enteros, por lo que su resultado es un número entero. Para e, uno de los números es un número Real, por lo que el resultado también es Real.

Algunos operadores, por polimorfismo, funcionan de diferente manera según las variables utilizadas como parámetros. El caso mas útil de esto, es el operador de suma, que si se utiliza con Strings, permite concatenación de cadenas de caracteres.

```
>>> a="hola"
>>> b="mundo"
>>> c=a+b
>>> c
'holamundo'

>>> d=a*3
>>> d
'holaholahola'
```

1.4.- Operadores Booleanos

Python cuenta con los siguientes operadores booleanos:

- and Condición Y
- or Condición O
- not Negación
- == Igualdad
- != Desigualdad
- > Mayor que
- < Menor que
- >= Mayor o igual que
- <= Menor o igual que

Ejemplos:

```
>>> True and True
True
>>> True and False
False
>>> True or False
True
>>> not True
False
>>> 'hola'=='hola'
True
```



```
>>> 'hola' != 'hola'
False
>>> 10>20
False
>>> 10<20
True
```

[2] - Listas, Tuplas y Diccionarios

2.1.- Listas

Las listas son colecciones ordenadas de valores, que pueden tener dentro de sus items, valores de cualquier tipo. Se pueden extraer los valores dentro de la lista indicando su índice (que empieza en 0). Se puede definir una lista vacía mediante [], o se pueden incluir sus valores encerrándolos entre [y]. Ejm.

```
>>> a=[]
>>> a
[]
>>> b=[1,2,3,4,5]
>>> b
[1, 2, 3, 4, 5]
>>> c=[1, "dos", 3, "cuatro", 5.0
... ]
>>> c
[1, 'dos', 3, 'cuatro', 5.0]
>>> c[1]
'dos'
>>> c[4]
```

Podemos obtener el tamaño de una lista mediante el comando len(). Ejm.

```
>>> len(c)
5
```

Las variables tipo lista, cuentan con varias funciones que permiten realizar operaciones dentro de ellas, entre las mas comunes podemos mencionar:

- .append(valor) Agrega un elemento al final de la lista
- .count(pos) Indica cuantas veces aparece el valor situado en la posición (pos) dentro de la lista
- .insert(pos, valor) Inserta un elemento (valor) en la posición (pos) indicada
- .pop(pos) Remueve el valor situado en la posición (pos) indicada

Ejemplos:

```
>>> c
[1, 'dos', 3, 'cuatro', 5.0]
>>> c.count(1)
1
>>> c.append(1)
>>> c
[1, 'dos', 3, 'cuatro', 5.0, 1]
>>> c.count(1)
2
>>> c.insert(2, 'dos')
>>> c
[1, 'dos', 'dos', 3, 'cuatro', 5.0, 1]
>>> c.pop(3)
3
>>> c
[1, 'dos', 'dos', 'cuatro', 5.0, 1]
```

Se pueden acceder a los items de la lista, iniciando desde el final, si se usa un valor negativo para el índice.

```
>>> c[-1]
1
>>> c[-2]
5.0
>>> c[-4]
'dos'
```

Al poder las listas contener cualquier tipo de elemento, se pueden crear listas de listas, a manera de matrices bidimensionales.

```
>>> a=[[1,2], [3,4]]
>>> a
[[1, 2], [3, 4]]
>>> a[0][1]
2
```

Podemos también crear una lista numérica en secuencia, mediante la función `range()`, que se puede utilizar de la siguiente manera:

- `range(fin)`
- `range(inicio, fin)`
- `range(inicio,fin,paso)`

```
>>> range(5)
[0, 1, 2, 3, 4]
>>> range(5,10)
[5, 6, 7, 8, 9]
```

Fíjese que cuando usamos `range(inicio, fin)`, el valor de fin es excluido de la lista.

2.2.- Tuplas

Las tuplas son colecciones ordenadas de valores, que mantienen características similares con las listas. Se definen encerrando sus items entre paréntesis, o simplemente listándolos separándolos con comas. Ejm.

```
>>> a=(1,2,3)
>>> a
(1, 2, 3)
>>> b=1,2,3
>>> b
(1, 2, 3)
```

La principal diferencia con las listas, es que las tuplas son inmutables, es decir, una vez definidas, no se puede modificar ni su tamaño, ni los elementos contenidos en ella. Por esta característica, las tuplas consumen menos espacio en memoria, lo que los hace útiles en determinados casos.

Al igual que con las listas, podemos acceder a los items contenidos en una tupla, indicando el índice del mismo. Ejm.

```
>>> a=(1, "dos", 3, "cuatro", 5.0)
>>> a[2]
3
```

Si intentamos modificar un valor contenido en una tupla, nos presentará un error, por lo mencionado anteriormente.

```
>>> a[2]='dos_nuevo'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

2.3.- Diccionarios

Los diccionarios son listas de elementos, pero asociados mediante una estructura llave -> valor. Se definen mediante `{ y }` y sus elementos internos se definen llave:valor, y tanto la llave como el valor, pueden ser de cualquier tipo.

```
>>> a={ 'nombre': 'Patricio', 'valor': 10.0, 1: 20, 2.0: 21.0}
>>> a
{'nombre': 'Patricio', 2.0: 21.0, 1: 20, 'valor': 10.0}
```

Nótese en el ejemplo, que el momento de guardar el diccionario pierde el orden de ingreso de los valores. Esto se debe a que los diccionarios son listas no ordenadas de elementos, y su orden final, dependerá de la disposición de memoria del equipo en ese instante.

Para acceder a los elementos del diccionario, accedemos mediante la llave del valor que deseamos obtener.

```
>>> a['nombre']
'Patricio'
>>> a[1]
20
>>> a[2.0]
21.0
>>> a[2]
21.0
```

Fíjese en el ejemplo, que al usar a[1] ó a[2] no estamos accediendo al 2o o 3r elemento de la lista, sino a los elementos con índice 1 y 2 respectivamente. Si intentáramos usar a[0] para acceder al primer elemento del diccionario, obtendríamos un error del intérprete, porque no existe ningún valor cuya llave sea "0"

```
>>> a[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 0
```

Los diccionarios resultan útiles, cuando necesitamos generar información estructurada.

Ejercicio: Se desea crear un directorio telefónico. Mediante el uso de diccionarios y listas, cree un objeto que contenga nombre y número de teléfono de varias personas.

```
>>> dato1={'nombre':'pedro', 'telefono':'2800111'}
>>> dato2={'nombre':'juan', 'telefono':'2811221'}
>>> dato3={'nombre':'jose', 'telefono':'2899888'}
>>> directorio=[dato1, dato2, dato3]
>>> directorio
[{'nombre': 'pedro', 'telefono': '2800111'}, {'nombre': 'juan', 'telefono': '2811221'}, {'nombre': 'jose', 'telefono': '2899888'}]
>>> dato4={'nombre':'manuel', 'telefono':'2299323'}
>>> directorio.append(dato4)
>>> directorio
[{'nombre': 'pedro', 'telefono': '2800111'}, {'nombre': 'juan', 'telefono': '2811221'}, {'nombre': 'jose', 'telefono': '2899888'}, {'nombre': 'manuel', 'telefono': '2299323'}]
```

Para acceder a los datos del objeto creado, al ser una lista primero debemos indicar el índice del item, y luego la llave del dato:

```
>>> directorio[1]["nombre"]
'juan'
>>> directorio[2]["nombre"]
'jose'
>>> directorio[3]
{'nombre': 'manuel', 'telefono': '2299323'}
```

[3] - Condicionales

Los condicionales nos permiten realizar una u otra tarea dependiendo de una condición preestablecida. En Python, el comando para el condicional es "if", con algunas variaciones que detallaremos a continuación.

Previo al uso del condicional hay que recordar el especial cuidado que se debe tener con la indentación. Al ser Python un lenguaje que no tiene caracteres reservados para la definición de funciones, subrutinas o código que se ejecuta dentro de los bucles o condicionales, la única forma de definir que un bloque de código se ejecuta dentro de una determinada función o bucle, es mediante el uso de indentación.

Volviendo al caso del condicional, tenemos que se define de la siguiente manera:

```
if (condicion):  
    accion1  
    accion2  
    .  
    .  
    accion_n
```

Como observamos todas las "acciones" que se encuentran un nivel hacia la derecha de la línea principal de indentación, serán ejecutadas dentro de la condición. Esto lo podemos visualizar mejor dentro del siguiente ejemplo:

```
>>> a=10  
>>> b=15  
>>> if (a!=b):  
...     print "diferentes"  
...  
diferentes  
>>> if (a==b):  
...     print "iguales"  
...
```

Al operador "if" se lo puede usar con algunas variaciones:

Condición estándar:

```
if (condicion):  
    accion  
    accion  
    .  
    .
```

Condición con acción positiva y negativa:

```
if (condicion):
    accion
    accion
    .
    .
else:
    accion
    accion
    .
    .
```

Condición múltiple:

```
if (condicion1):
    accion
    .
    .
elif (condicion2):
    accion
    .
    .
elif (condicion3):
    .
    .
    .
```

Su uso dependerá de cada aplicación. Pero esto nos muestra la versatilidad de Python en el uso de condicionales.

Un caso especial de uso del "if" es en la asignación de valores a variables. Se puede asignar un valor en función de una condición de acuerdo a la estructura A if C else B. Esto lo podemos visualizar en el siguiente ejemplo:

```
>>> num=1
>>> var="par" if (num%2==0) else "impar"
>>> var
'impar'
>>> num=2
>>> var="par" if (num%2==0) else "impar"
>>> var
'par'
```

Como podemos observar el valor asignado a la variable "par" estará en función si el valor de la variable num es par o impar (calculado mediante el operador módulo %). Esto nos permite escribir código mas expresivo, ya que la misma funcionalidad se podría obtener mediante el siguiente código:

```
>>> if (num%2==0):  
...     var="par"  
... else:  
...     var="impar"  
...
```


[4] - Bucles

En python existen dos funciones para generar bucles:

- while(condicion):
- for valor en lista:

4.1.- Bucles while

El bucle while permite ejecutar un bloque de código mientras se cumpla determinada condición:

```
>>> a=10
>>> while (a>0):
...     print a
...     a=a-1
...
10
9
8
7
6
5
4
3
2
1
```

Podemos también crear bucles infinitos, que pueden ser detenidos mediante el comando break

```
>>> a=10
>>> while (True):
...     print a
...     if (a<3):
...         break
...     a=a-1
...
10
9
8
7
6
5
4
3
2
```

En este ejercicio, observamos que el bucle se ejecutará hasta que la comparación ($a < 3$) sea verdadera, rompiendo el bucle mediante el comando `break`

4.2.- Bucles for

Los bucles `for` permiten iterar sobre cada uno de los elementos de una lista, tupla o diccionario. El formato del comando `for` es el siguiente:

```
for item in lista:
    accion
    accion
    .
    .
```

Donde `item` es el nombre de la variable que almacenará cada uno de los valores resultado de la iteración de consulta de los elementos de lista, siendo lista una lista de elementos, tupla, o diccionario. En el caso de los diccionarios, en `item` se irán almacenando las llaves de cada uno de los elementos del diccionario.

```
>>> a=[1,2,3,4]
>>> for valor in a:
...     print valor
...
1
2
3
4
```

```
>>> dato1={'nombre': 'pedro', 'telefono': '2800111'}
>>> for llave in dato1:
...     print llave
...
nombre
telefono
```

Un uso común de la función `for` es en conjunto con la función `range`, para generar contadores numéricos.

```
>>> for val in range(1,11):
...     print str(val)+"*2="+str(val*2)
...
1*2=2
2*2=4
3*2=6
```

$$4 \times 2 = 8$$

$$5 \times 2 = 10$$

$$6 \times 2 = 12$$

$$7 \times 2 = 14$$

$$8 \times 2 = 16$$

$$9 \times 2 = 18$$

$$10 \times 2 = 20$$

[5] - Funciones

Python, al igual que otros lenguajes de programación, permite la generación de funciones para estructurar sus programas. Para definir una función se usa la palabra reservada `def`, y los parámetros de la función se colocarán entre paréntesis. Para devolver un valor desde una función, se utilizará el comando `return` (valor).

```
>>> def saluda():
...     print "hola a todos"
...
>>> saluda()
hola a todos

>>> def suma(a,b):
...     c=a+b
...     return c
...
>>> suma(10,23)
33
>>> suma(10,13)
23
```

Una característica especial de Python, es la preasignación de valores para las funciones. Esto permite que la llamada a función pueda contener menos valores de los determinados en la función.

Esto lo podemos observar en el siguiente ejercicio:

```
>>> def multiplica(val1=2, val2=3):
...     c=val1*val2
...     return c
...
>>> multiplica(10,2)
20
>>> multiplica(10)
30
>>> multiplica()
6
>>> multiplica(val2=5)
10
```

Si analizamos los resultados, en la primera llamada `multiplica(10,2)` efectivamente la función `multiplica` los 2 valores, y devuelve 20 como resultado. En el segundo caso, pasamos solo el valor 10, que se asigna a `val1` el momento de llamar a la función. Al ser `val2=3`, el resultado es $10*3=30$. En el tercer caso, donde no se pasa ningún parámetro, la función toma los valores predeterminados por defecto, devolviendo $3*2=6$ como resultado. El último caso es el más interesante, ya que estamos pasando como parámetro `val2=5`. Esto indica que el segundo parámetro (`val2`) se asigne

con el valor indicado, teniendo val1 el valor por defecto. Esto nos muestra la versatilidad de las llamadas a función en python.

Otro dato adicional con respecto a la definición de funciones, es que no es necesario indicar el tipo de variables a pasar a la función. Esto puede convertirse en una desventaja, si no se lleva un orden correcto en el proceso a ejecutar en la función y en las operaciones a realizar sobre las variables.

Otra característica adicional de la definición de funciones, es la definición de parámetros indeterminados. Esto se consigue colocando un asterisco (*) antes de la variable indeterminada en la función. Al hacer esto, dicha variable almacenará una tupla con los valores pasados.

```
>>> def prueba(a, *b):  
...     print a  
...     print b  
...  
>>> prueba('hola', 'mundo')  
hola  
( 'mundo', )  
>>> prueba('hola', 'mundo', 'de', 'python')  
hola  
( 'mundo', 'de', 'python' )  
>>> prueba('hola')  
hola  
( )
```

Como observamos en el ejemplo, el primer valor es el obligatorio, tomando el resto de valores pasados a la función y guardándolos en una tupla.

[6] - Manejo de Excepciones

El manejo de excepciones permite que el flujo de ejecución del programa no se vea interrumpido por un error determinado. Esto lo logramos mediante el uso de las funciones try, except y else. En su forma más básica, podemos usar el manejo de excepciones de la siguiente manera:

```
try:
    acciones
    acciones
    .
    .
except:
    accion_en_caso_error
    .
    .
else:
    accion_final
    .
    .
```

Podemos ilustrar el funcionamiento de las excepciones mediante el siguiente ejercicio:

```
>>> def division(a,b):
...     try:
...         return a/b
...     except:
...         print "Error en el calculo"
...
>>> division(10,2)
5
>>> division(10,0)
Error en el calculo
```

Como vemos, en el segundo caso de aplicación de la función, al dividir el valor para 0, nos muestra un mensaje indicando "Error en el cálculo", pero si ejecutamos directamente la división para cero, veremos que nos presenta un error fatal el intérprete:

```
>>> 10/0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: integer division or modulo by zero
```

[7] - Manejo de I/O

7.1.- Entrada desde consola

Python permite la interacción con el usuario desde la consola de terminal mediante un simple comando. el comando `raw_input` (o solo `input` en la versión 3), lee el texto introducido por el usuario desde la consola, y lo devuelve como string al programa.

```
>>> texto=raw_input('ingrese su nombre: ')
ingrese su nombre: patricio
>>> texto
'patricio'
```

7.2.- Salida a consola

Para presentar texto como salida a consola, Python dispone del comando `print()`. Este comando acepta una cadena de caracteres como parámetro a visualizar, permitiendo cierta funcionalidad similar al comando `sprintf` de C/C++.

Para definir formato mediante la función `print`, utilizaremos el siguiente esquema:

```
print formato % (tupla_valores)
```

Para el formato, tenemos los siguientes operadores:

- `%s` Cadena de caracteres
- `%d` Número Entero
- `%o` Numero Entero Octal
- `%x` Número Entero Hexadecimal
- `%f` Número Real

```
>>> print "%s" % ('hola mundo')
hola mundo
>>> print "%s -> %d" % ('hola mundo', 10)
hola mundo -> 10
>>> print "%s -> %x" % ('hola mundo', 10)
hola mundo -> a
>>> print "%s -> %f" % ('hola mundo', 10)
hola mundo -> 10.000000
```

También se usa la función `str()` para convertir valores numéricos a cadenas de caracteres. Esto se puede usar en conjunto con el operador `+` para concatenar cadenas

```
>>> print "hola mundo -> " + str(10)
hola mundo -> 10
>>> print "hola mundo -> " + str(0x10)
hola mundo -> 16
>>> print "hola mundo -> " + str(0xa)
hola mundo -> 10
>>> print "hola mundo -> " + str(10.00)
hola mundo -> 10.0
```

7.3.- Manejo de archivos

Python maneja la entrada y salida de datos hacia archivos de una manera sencilla y eficiente. Esto lo convierte en un lenguaje muy versátil y útil a la hora de procesar archivos de datos.

el comando `open(archivo, tipo)` permite abrir un archivo y almacenarlo en un puntero de memoria, para su acceso. Los tipos de acceso permitidos son:

- `'r'` solo lectura
- `'w'` escritura (borra contenido si ya existía previamente)
- `'a'` adicionar contenido al archivo

Para mantener la consistencia de la información del archivo, es necesario cerrarlo luego de utilizarlo. Esto se realiza mediante el comando `.close()` sobre la variable que apunta al archivo abierto.

7.3.1.- Lectura de Archivos

Una vez que hemos abierto el archivo, podemos leer su contenido mediante dos comandos:

- `read()` Lee todo el contenido y lo almacena como cadena de caracteres en una variable
- `readline()` Lee una línea de contenido y la almacena como cadena de caracteres

Para el caso de que queramos leer todo el contenido del archivo en una sola variable, podemos usar el siguiente código:

```
f=open("archivo.txt", "r")
contenido=f.read()
f.close()
```


Para el caso de `readline()` es necesario crear un bucle que realice una iteración sobre cada una de las líneas del archivo. Esto se logra de la siguiente manera:

```
f=open("archivo.txt", "r")
while True:
    linea = f.readline()
    if not linea: break
    print linea
f.close()
```

7.3.2.- Escritura en Archivo

Al igual que el caso de la lectura de contenido de un archivo, para grabar contenido dentro del mismo, tenemos dos comandos, que se aplican al puntero del archivo abierto previamente:

- `.write(linea)` Graba la línea de texto al archivo
- `.writelines([texto1, texto2..., texton])` Graba cada una de las líneas de la lista en el archivo

El uso de estas funciones es bastante trivial:

```
f=open("archivo.txt", "r")
f.write('hola! esto va al archivo de texto')
f.close()
```

Hay que tener cuidado de cerrar correctamente el archivo, caso contrario, podría darse una pérdida de la información guardada en el mismo.

[8] - Ejecución de programas Python desde la consola.

Como hemos visto a lo largo de los temas anteriores, todos los ejercicios los hemos venido realizando desde la consola interactiva de Python.

Python está pensado como un lenguaje de programación para ejecución de scripts, por lo que permite la posibilidad de generar archivos de scripts para su posterior ejecución. Para esto, nuestro programa debemos almacenarlo en un archivo de texto con extensión .py (usando su editor de texto favorito), y se ejecuta desde una consola de terminal mediante el siguiente comando:

```
python archivo.py
```

Ejercicio:

Como ejercicio final de esta etapa, generaremos un script que obtenga la serie de Fibonacci con un número determinado de valores. El resultado de esta ejecución deberá almacenarse en un archivo de texto.

```
# programa fib.py
# obtiene la serie de fibonacci y guarda el resultado en un archivo de texto
def fib(num):
    lista=[0]
    for i in range(num):
        if (len(lista)>1):
            val=lista[-1]+lista[-2]
        else:
            val=1
        lista.append(val)
    return lista

x=raw_input("numero de datos: ")
lista=fib(int(x))
f=open('salida.txt', 'w')
for item in lista:
    f.write(item)
    print item
f.close
# fin del programa
```

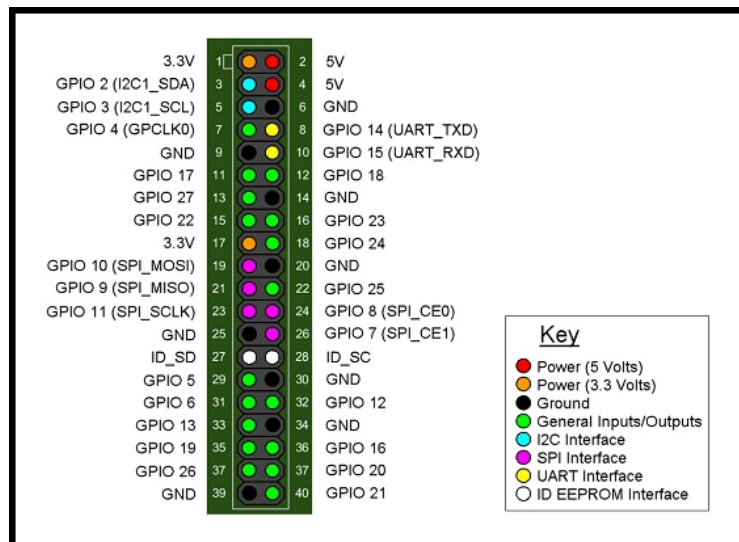
[9] - Python y la Raspberry Pi

La Raspberry Pi toma su nombre de dos partes. En Inglaterra, ha existido una cierta "tradición" de las empresas de poner nombres de frutas a sus equipos (tradición que no tiene nada que ver con Apple en América), así que Raspberry (o mora) es la fruta favorita de uno de los creadores del computador. Pi viene desde Python, que desde un inicio, fué el lenguaje que iba a tener mayor soporte sobre la tarjeta.

Es por esto, que mucho del esfuerzo de desarrollo sobre la Raspberry Pi se ha enfocado a su uso con Python como lenguaje de programación de preferencia. Todas las funcionalidades que revisaremos a continuación nos mostrarán la versatilidad del lenguaje aplicado a este micro computador personal.

9.1.- GPIO Básico

La Raspberry Pi (desde la versión B+ en adelante), cuenta con un conector de 40 pines denominado GPIO, que viene de General Purpose Input Output. Son 40 pines de entrada y salida, donde algunos son programables, y pueden ejecutar varias funciones. Para acceder a estos pines, contamos con algunas librerías que, dependiendo del caso, nos brindarán una u otra funcionalidad sobre el equipo.



Para usar los pines del GPIO como pines de entrada y salida, debemos tomar ciertas consideraciones:

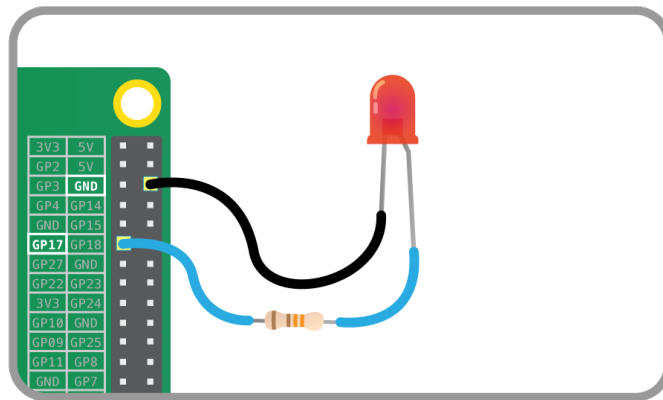
- El voltaje de operación es de 3.3v, y los pines no son tolerantes a 5V. Usar un voltaje mayor a 3.3v podría ocasionar un daño total a la tarjeta
- La corriente de salida es de 5mA para cada pin, por lo que se debe tener cuidado en no sobrecargar los pines
- Debe cuidarse la protección contra estática, ya que los pines del GPIO están directamente conectados al procesador, sin protección alguna.

Antes de usar el GPIO, debemos asegurarnos que tenemos las librerías correctas instaladas. Para esto vamos a ejecutar los siguientes comandos desde una consola de terminal

```
sudo apt-get update
sudo apt-get install python-gpiozero python3-gpiozero
```

Estos dos comandos se encargan de descargar e instalar la librería gpiozero, que usaremos en los siguientes ejercicios.

Para probar la funcionalidad de la librería, vamos a conectar un LED a uno de los pines del GPIO. Esta conexión la realizaremos según el siguiente diagrama:



Y para probar la funcionalidad de la librería GPIO, podemos probar con los siguientes programas:

```
# programa led1.py
# parpadea un led conectado al pin 17 del GPIO
# en bucle infinito
from gpiozero import LED
from time import sleep
```

```
red=LED(17)
while (True):
    red.on()
    sleep(1)
    red.off()
    sleep(1)
#fin del programa
```

```
# programa led2.py
# parpadea un led conectado al pin 17 del GPIO
# usando la funcion blink()
from gpiozero import LED
from signal import pause
```

```
red=LED(17)
red.blink()
```

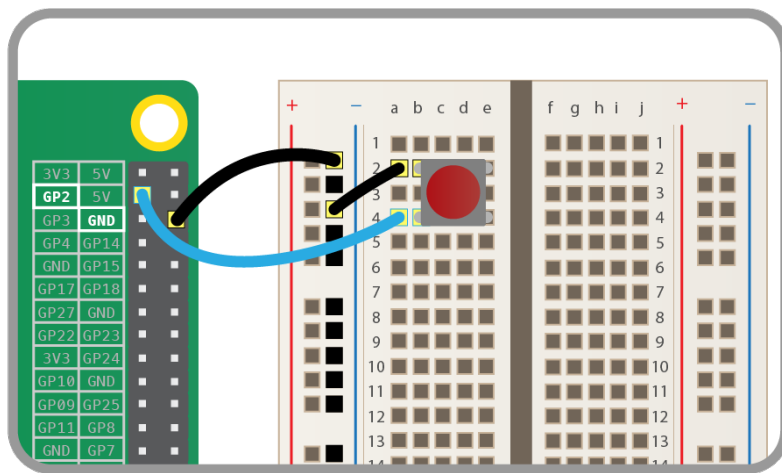
```

pause()
#fin del programa

```

Como podemos observar, con dos programas diferentes, y dos métodos de ejecución logramos obtener el mismo resultado. En el primer programa, utilizamos un clásico bucle infinito, que puede ser detenido mediante Ctrl+C, mientras que en el segundo programa, la función `blink()` aplicada al objeto LED, hace parpadear indefinidamente al led, mientras que no se presione ninguna tecla, acción que la función `pause()` espera.

Podemos usar también los pines del GPIO como entradas digitales. Para realizar una prueba de esta funcionalidad, vamos a conectar un pulsante de acuerdo al siguiente diagrama.



Cabe anotar que internamente, los pines del GPIO tienen una resistencia PullUp Programable, que se activa directamente al configurar el pin deseado como entrada.

Para probar el funcionamiento de las entradas digitales, vamos a probar los siguientes programas:

```

# programa switch1.py
# pooling de GPIO
from gpiozero import Button
boton=Button(2)
while (True):
    if (boton.is_pressed):
        print "Botón presionado"
    else:
        print "Botón no presionado"
# fin del programa

```

```

# programa switch2.py
# Espera de acción para GPIO
from gpiozero import Button
boton=Button(2)
boton.wait_for_press()
print "Boton presionado"
#fin del programa

```

```
# programa switch3.py
# Llamadas a eventos
from gpiozero import Button

def hola():
    print "Hola! presionaste el botón"

boton=Button(2)
boton.when_pressed = hola
pause()
#fin del programa
```

Al ejecutar cada uno de estos programas, podemos observar los distintos métodos de acceso a los pines de entrada; ya sea por pooling del pin, espera de estado, o por eventos, podemos realizar la lectura de las entradas digitales de la Raspberry Pi.

9.2.- PWM

La librería **gpiozero** permite la generación (via software) de señales PWM en los pines configurados como salida en la Raspberry Pi. Si conectamos el led tal como lo realizamos en el ejemplo anterior, al pin 17 de la Raspberry, podemos probar la generación de PWM usando el siguiente programa:

```
# programa pwm1.py
# generacion de PWM
from time import sleep
from gpiozero import PWMLED
led=PWMLED(17)

led.on()
sleep(1)
led.value(0.75)
sleep(1)
led.value(0.5)
sleep(1)
led.value(0.25)
sleep(1)
led.value(0.0)
#fin del programa
```

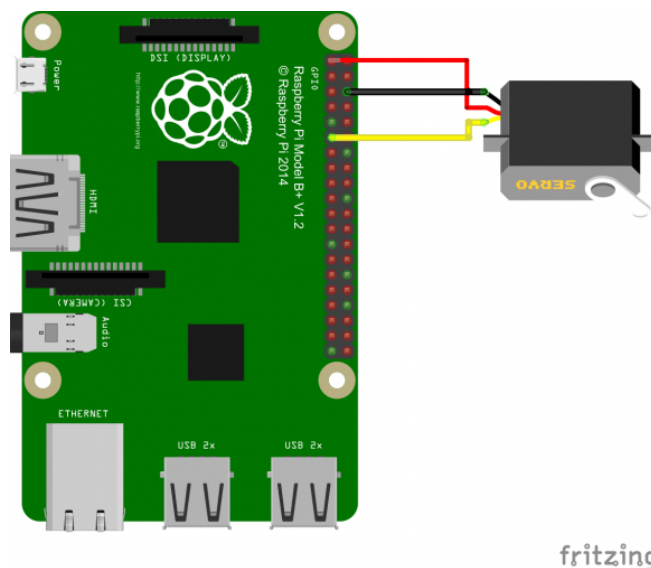
De la ejecución de este programa, podemos observar que podemos variar los valores del ancho de pulso PWM con valores entre 0.0 y 1.0

9.3.- Control de Servos mediante PWM

La capacidad de generación de pulsos PWM, nos brinda la posibilidad de comandar servomotores con la Raspberry Pi. Al igual que en el caso de las entradas y salidas digitales, existen ciertas consideraciones a tomar en cuenta:

- A menos que se use un microservo, no se recomienda alimentarlo directo desde los pines de +5v ó +3.3v de la Raspberry, porque la corriente de salida de estos pines es muy limitada
- Se debe tener especial cuidado de no causar cortocircuitos a los pines del GPIO
- Al ser una señal PWM generada por software, ésta no será tan estable como la generada por un microcontrolador, por lo que el servo podría no estabilizarse en algunos casos

Para realizar la conexión de un servo, utilizaremos la librería Rpi.GPIO, y lo conectaremos según el siguiente diagrama:



El programa para el control del servo es el siguiente:

```
#programa servo.py
#control de servo mediante Rpi.GPIO
import Rpi.GPIO as GPIO
from time import sleep

GPIO.setmode(GPIO.BCM)
GPIO.setup(17, GPIO.OUT)
pwm=GPIO.PWM(17,50)
pwm.start(7)
while True:
    pwm.ChangeDutyCycle(7)
    sleep(1)
    pwm.ChangeDutyCycle(8)
    sleep(1)
    pwm.ChangeDutyCycle(9)
```

```

sleep(1)
pwm.ChangeDutyCycle(10)
sleep(1)
#fin del programa

```

Debemos presentar especial atención a la función `GPIO.PWM(pin, freq)`, que nos permite configurar el PWM en un pin determinado, y a una frecuencia determinada. En este caso usamos 50Hz porque nos dará el ancho de pulso necesario (2ms) para el control del servo.

9.4.- Sensores 1-Wire

La Raspberry Pi permite el uso de sensores que utilizan el protocolo 1-Wire para comunicación y transmisión de datos. Ya que el driver para manejo de estos dispositivos se encuentra cargado dentro del kernel de Linux, es necesario activarlo antes de usarlo. Para esto necesitamos modificar el archivo `/boot/config.txt` mediante el comando

```
sudo nano /boot/config.txt
```

y agregar (o editar) la siguiente línea al final del archivo:

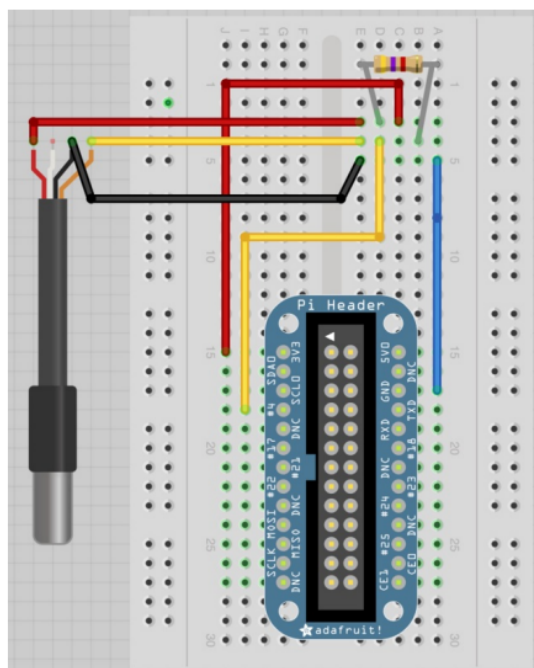
```
dtoverlay=w1-gpio
```

Guardamos el archivo y reiniciamos la raspberry mediante

```
sudo reboot
```

y el driver estará listo para recibir dispositivos 1-Wire.

Para probar la funcionalidad de este protocolo, vamos a utilizar el sensor de temperatura digital Dallas DS18B20. Lo conectaremos de acuerdo al siguiente diagrama:



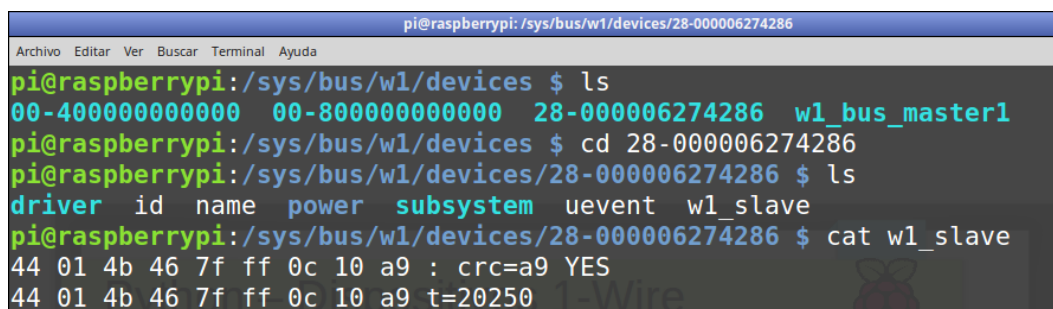
Para comprobar el correcto funcionamiento del sensor, ejecutaremos los siguientes comandos desde la consola de terminal:

```
sudo modprobe w1-gpio
sudo modprobe w1-therm
```

Si no se presenta ningún mensaje de error, la prueba del protocolo y del sensor ha sido exitosa. Ahora, para verificar que el sensor se encuentre activo, debemos cambiarnos a la carpeta devices de los dispositivos w1, mediante el siguiente comando

```
cd /sys/bus/w1/devices
```

y si hacemos un listado del directorio, veremos que existe una carpeta que inicia con "28-". Esta carpeta corresponde al sensor de temperatura, ya que el driver la incluye como una entrada en el sistema de archivos. Si queremos verificar que el sensor esté entregando lecturas correctamente, debemos entrar a la carpeta del sensor, y visualizar el contenido del archivo w1_slave, obteniendo un resultado como el que presentamos a continuación:



```
pi@raspberrypi: /sys/bus/w1/devices/28-000006274286
Archivo Editar Ver Buscar Terminal Ayuda
pi@raspberrypi:/sys/bus/w1/devices $ ls
00-400000000000 00-800000000000 28-000006274286 w1_bus_master1
pi@raspberrypi:/sys/bus/w1/devices $ cd 28-000006274286
pi@raspberrypi:/sys/bus/w1/devices/28-000006274286 $ ls
driver id name power subsystem uevent w1_slave
pi@raspberrypi:/sys/bus/w1/devices/28-000006274286 $ cat w1_slave
44 01 4b 46 7f ff 0c 10 a9 : crc=a9 YES
44 01 4b 46 7f ff 0c 10 a9 t=20250
```

Si nos fijamos en el resultado del comando cat w1_slave, vemos que devuelve 2 líneas de texto. Al final de la primera, el YES nos indica que la última lectura realizada fué correcta, y al final de la segunda línea encontramos t=20250 (en este ejemplo), que nos indica que la temperatura es 20.250 grados centígrados.

Para usar esto, debemos realizar en Python un programa que lea dicho archivo, y lo procese para que podamos obtener el dato de temperatura y utilizarlo según nuestra necesidad. Para esto, vamos a realizar el siguiente ejercicio:

```
# programa sensor1wire.py
# lee de manera continua el sensor e imprime el valor de temperatura
dir="/sys/bus/w1/devices/"
codigo="28-800000007f6cd"
archivo="/w1_slave"

rutaFinal=dir+codigo+archivo

f=open(rutaFinal, "r")
contLinea=0
```

```

while True:
    linea=f.readline()
    if not linea: break
    if contLinea==0:
        datoValido=linea[-4:-1]
        if datoValido=="YES":
            linea=f.readline()
            datoTemp=linea[-6:-1]
            tempInt=int(datoTemp)
            temperatura=tempInt/1000.0
            print "La temperatura actual es: " + str(temperatura)
# fin del programa

```

Como podemos observar en el código, el programa realiza 3 tareas:

- Abre el archivo w1_slave en la ruta determinada por el código del sensor
- Lee la primera línea del archivo y busca "YES" para verificar si la lectura es correcta
- Lee los dígitos finales de la segunda línea y divide para 1000.0, para obtener el valor en grados centígrados

Con esto, de una manera sencilla hemos logrado usar un sensor de protocolo 1-Wire con nuestra Raspberry.

[10] - Conexión con servicios Web

La Raspberry Pi, al tener conexión ethernet, y usar el sistema operativo Linux, nos brinda la posibilidad de conectarnos directamente con la internet, y no solo mediante el uso del navegador web, sino conectarnos a servicios web que nos permiten leer o almacenar datos en la nube.

Antes de explicar mediante algunos ejemplos la conexión a servicios web, es necesario que conozcamos el formato de datos usados en la transmisión y recepción de información de la internet.

10.1.- Datos en formato JSON

JSON es el acrónimo de Javascript Object Notation, y es el formato con el que javascript maneja sus objetos. Si nos fijamos en el ejemplo de datos en formato JSON que presentamos a continuación, notamos cierta similitud con el formato del Diccionario utilizado en Python.

```
{
  hey: "guy",
  anumber: 243,
  - anobject: {
    whoa: "nuts",
    - anarray: [
      1,
      2,
      "thr<h1>ee"
    ],
    more: "stuff"
  },
  awesome: true,
  bogus: false,
  meaning: null,
  japanese: "明日がある。",
  link: http://jsonview.com,
  notLink: "http://jsonview.com is great"
}
```

JSON no es una estructura de datos como tal, sino un mecanismo de serialización de datos, ya que los datos transferidos en JSON son cadenas de caracteres, que necesitan ser interpretadas antes de ser utilizadas. Python incluye la librería json para el manejo de este formato, e incluye las funciones loads() y dumps() para leer y escribir datos JSON respectivamente. Esto lo podemos ilustrar mediante los siguientes ejemplos:

```

>>> # paso de diccionario a json
>>> dato1={'nombre': 'pedro', 'telefono': '2800111'}
>>> import json
>>> json.dumps(dato1)
'{"nombre": "pedro", "telefono": "2800111"}'

>>> # paso de string json a diccionario
>>> datojson='{"nombre": "pedro", "telefono": "2800111", "formato":"json"}'
>>> json.loads(datojson)
{'nombre': 'pedro', 'formato': 'json', 'telefono': '2800111'}
>>> a=json.loads(datojson)
>>> a['formato']
'json'

```

Como podemos observar, en el primer ejemplo hemos tomado una variable tipo diccionario, y la hemos serializado a un string en formato JSON. En el segundo ejemplo, tenemos una cadena de caracteres que representa una estructura de datos en formato JSON, y mediante la función loads() la hemos transformado a un diccionario de datos, por lo que podemos acceder al elemento "formato" del mismo.

Para ilustrar de mejor manera el uso de este formato de datos, vamos a utilizar el siguiente ejercicio:

```

# programa firebase.py
# conexión via request con un servicio web, y envío de datos en formato JSON
# -*- coding: latin-1 -*-

import time
import requests
import json

firebase_url="https://prueba-raspi.firebaseio.com"

intervalo=10
ubicacion="raspberrypi"
valor=1

#leemos los datos
result=requests.get(firebase_url+"/"+ubicacion+"/raspberrypi.json")
datos=result.text
arrayDatos=json.loads(datos)
for dato in arrayDatos:
    print arrayDatos[dato]

#guardamos 10 datos

while (valor<=10):
    try:
        hora=time.strftime('%H:%M:%S')
        fecha=time.strftime('%Y-%m-%d')
        datos={'fecha':fecha, 'hora':hora, 'ubicacion': ubicacion,
'valor':valor}

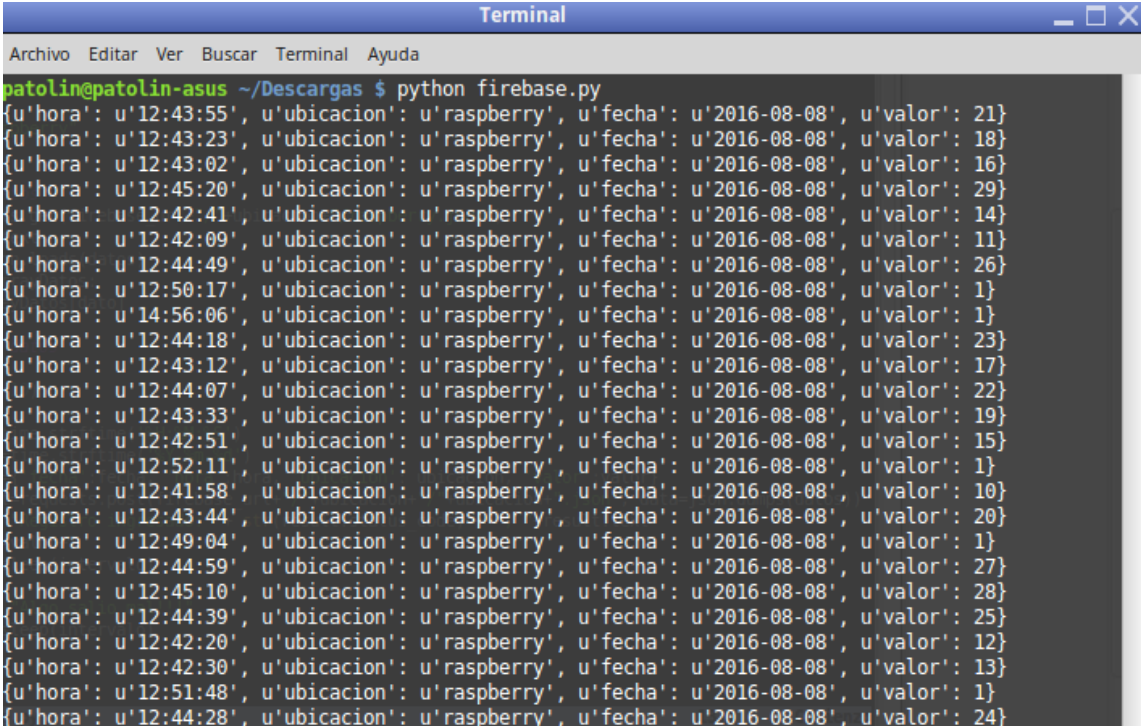
```

```

        result=requests.post(firebase_url+"/"+ubicacion+"/"+ubicacion+".json",
data=json.dumps(datos))
        print "Registro ingresado: "+ str(result.status_code) + ", "+result.text
        valor=valor+1
        time.sleep(intervalo)
    except:
        print "Algo salio mal!!!"
        time.sleep(intervalo)
#fin del programa

```

Este programa, se conecta a un servicio web. Específicamente una base de datos creada previamente en www.firebaseio.com. Firebase es una base de datos no relacional orientada a Big Data, que permite guardar datos mediante envíos POST de datos en formato JSON. Este programa realiza dos tareas: primero lee los datos que se encuentran actualmente en la base de datos para visualizarlos en pantalla (convirtiendo de JSON a diccionario de datos), y luego toma un diccionario de datos de ejemplo, lo convierte a JSON y lo envía mediante un evento POST hacia la url provista por el motor de base de datos. De esta manera sencilla, podemos guardar datos en la nube, valiéndonos de un servicio web.



```

Terminal
Archivo Editar Ver Buscar Terminal Ayuda
patolin@patolin-asus ~/Descargas $ python firebase.py
{'u'hora': u'12:43:55', u'ubicacion': u'raspberry', u'fecha': u'2016-08-08', u'valor': 21}
{'u'hora': u'12:43:23', u'ubicacion': u'raspberry', u'fecha': u'2016-08-08', u'valor': 18}
{'u'hora': u'12:43:02', u'ubicacion': u'raspberry', u'fecha': u'2016-08-08', u'valor': 16}
{'u'hora': u'12:45:20', u'ubicacion': u'raspberry', u'fecha': u'2016-08-08', u'valor': 29}
{'u'hora': u'12:42:41', u'ubicacion': u'raspberry', u'fecha': u'2016-08-08', u'valor': 14}
{'u'hora': u'12:42:09', u'ubicacion': u'raspberry', u'fecha': u'2016-08-08', u'valor': 11}
{'u'hora': u'12:44:49', u'ubicacion': u'raspberry', u'fecha': u'2016-08-08', u'valor': 26}
{'u'hora': u'12:50:17', u'ubicacion': u'raspberry', u'fecha': u'2016-08-08', u'valor': 1}
{'u'hora': u'14:56:06', u'ubicacion': u'raspberry', u'fecha': u'2016-08-08', u'valor': 1}
{'u'hora': u'12:44:18', u'ubicacion': u'raspberry', u'fecha': u'2016-08-08', u'valor': 23}
{'u'hora': u'12:43:12', u'ubicacion': u'raspberry', u'fecha': u'2016-08-08', u'valor': 17}
{'u'hora': u'12:44:07', u'ubicacion': u'raspberry', u'fecha': u'2016-08-08', u'valor': 22}
{'u'hora': u'12:43:33', u'ubicacion': u'raspberry', u'fecha': u'2016-08-08', u'valor': 19}
{'u'hora': u'12:42:51', u'ubicacion': u'raspberry', u'fecha': u'2016-08-08', u'valor': 15}
{'u'hora': u'12:52:11', u'ubicacion': u'raspberry', u'fecha': u'2016-08-08', u'valor': 1}
{'u'hora': u'12:41:58', u'ubicacion': u'raspberry', u'fecha': u'2016-08-08', u'valor': 10}
{'u'hora': u'12:43:44', u'ubicacion': u'raspberry', u'fecha': u'2016-08-08', u'valor': 20}
{'u'hora': u'12:49:04', u'ubicacion': u'raspberry', u'fecha': u'2016-08-08', u'valor': 1}
{'u'hora': u'12:44:59', u'ubicacion': u'raspberry', u'fecha': u'2016-08-08', u'valor': 27}
{'u'hora': u'12:45:10', u'ubicacion': u'raspberry', u'fecha': u'2016-08-08', u'valor': 28}
{'u'hora': u'12:44:39', u'ubicacion': u'raspberry', u'fecha': u'2016-08-08', u'valor': 25}
{'u'hora': u'12:42:20', u'ubicacion': u'raspberry', u'fecha': u'2016-08-08', u'valor': 12}
{'u'hora': u'12:42:30', u'ubicacion': u'raspberry', u'fecha': u'2016-08-08', u'valor': 13}
{'u'hora': u'12:51:48', u'ubicacion': u'raspberry', u'fecha': u'2016-08-08', u'valor': 1}
{'u'hora': u'12:44:28', u'ubicacion': u'raspberry', u'fecha': u'2016-08-08', u'valor': 24}

```

[11] - Visualización de Datos

Como se ha visto a lo largo de este manual, Python es muy versátil al leer y escribir datos, pero en la mayoría de programas que tienen interacción con el usuario, será necesario visualizar estos datos de una manera amigable.

Entre varias librerías disponibles, Python dispone de la librería `matplotlib` que permite mostrar gráficas estáticas o dinámicas de conjuntos de datos generados por un determinado programa. Para utilizar esta librería, será necesario instalarla previamente, por lo que debemos ejecutar el siguiente comando desde una consola de terminal:

```
sudo apt-get install python-matplotlib
```

Una vez instalada la librería, podemos explicar su uso mediante el siguiente ejercicio:

```
# programa plot1.py
# genera una gráfica usando matplotlib
import math
import matplotlib.pyplot as plt
x=[]
y=[]
for i in range(-2*314, 2*314, 10):
    x.append(i/100)
    y.append(math.sin(i/100))

plt.plot(x,y)
plt.show()
#fin del programa
```

[12] – Observaciones finales

Como comenté al inicio, este manual pretende ser una pequeña introducción al lenguaje Python, aplicado sobre la Raspberry Pi. Espero que la lectura y la elaboración de los ejercicios haya despertado un poco el gusto por la programación en este práctico lenguaje, y haya abierto la mente hacia posibles aplicaciones que se le puede dar a este interesante pedazo de hardware.

Mucha de la información presentada en este manual, ha sido tomada desde sitios como

- <http://www.python.org>
- <http://www.raspberrypi.org>
- <http://www.instructables.com>
- <http://www.stackoverflow.com>
- ...y muchos mas

Además, hay que agradecer a la gran comunidad de desarrolladores de Python a nivel mundial. Ellos han hecho posible que este lenguaje sea tan popular y amigable.

Y para finalizar, el infaltable Comic de XKCD

