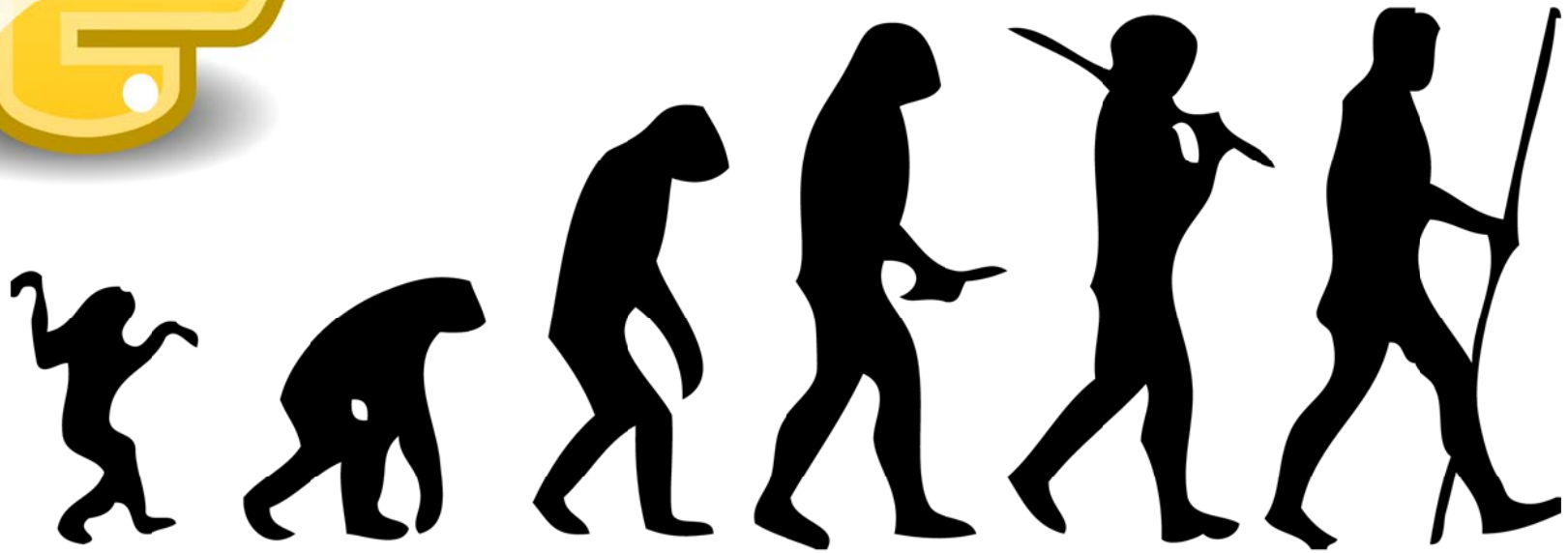
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISCA

Programando en Python

Raspberry Pi (RPi)



2014/06/06

Juan V. Capella

DISCA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Contenido

- Objetivo
- Introducción
- Operadores
- Tipos de datos
- Funciones
- Excepciones
- Módulos
- E/S y ficheros
- GPIO
- Networking
- GUI

Objetivo

- Introducirse en la programación en python con la Raspberry Pi

Introducción

- Creado a finales de los 80 por Guido van Rossum
- El nombre viene de los humoristas británicos Monty Python
- Lenguaje interpretado (script)
- Prototipado y desarrollo rápido de aplicaciones
- Paradigma multiprogramación (orientada a objetos, programación imperativa y programación funcional)
- Tipado dinámico: la misma variable puede contener valores de diferentes tipos.
  - No es necesario declarar el tipo de dato que va a contener una variable
    - éste se determina en tiempo de ejecución según el valor asignado
- Fuertemente tipado
  - No se permite tratar a una variable como si fuera de un tipo distinto al que tiene
    - Hay que convertir de forma explícita el tipo de la variable

Introducción

- Libre
- Muy alto nivel
- *Python standard library* (Librería muy completa)
  - https://docs.python.org/2/library/
  - Referencia sobre todas las funciones de librería podemos encontrarlas en la documentación oficial, disponible en la web..
    - Cadenas, listas, tablas hash, pilas, colas
    - Cálculo científico
    - Serialización, y Persistencia de Objetos
    - Programación Concurrente
    - Acceso a BD, Ficheros Comprimidos...
    - Networking
      - CGIs, URLs, HTTP, FTP,
      - pop3, IMAP, telnet
      - Cookies, Mime, XML, XDR
    - Diversos formatos multimedia
    - Criptografía

Introducción

- Lenguaje de propósito general
- Sencillo, compacto
- Sintaxis clara
- Identación obligatoria:

```
def factorial(x):
    if x == 0:
        return 1
    else:
        return x * factorial (x - 1)
```

  - Un bloque es un conjunto de instrucciones que se ejecutan secuencialmente
  - Python utiliza el indentado para reconocer las líneas que forman un bloque de instrucciones

- http://www.python.org/about/success/

Introducción

- Hola mundo en Python

```
print ”¡Hola Mundo!”  #esto es un comentario
```

- Hola mundo en C

```
#include <stdio.h>
int main (void) {
    printf(”¡Hola Mundo!”);
    return 0;
 }
```

- Hola mundo en Java

```
public class HolaMundo {
   public static void main(String [] args) {
       System.out.println(”¡Hola Mundo!”);
      }
 }
```

Ejecutando nuestros programas

- python fichero.py

- #!/usr/bin/env python
- chmod +x fichero.py
- ./fichero.py

- IDLE

Funciones built-in

		Built-in Functions		
abs()	divmod()	input()	open()	staticmethod()
all()	enumerate()	int()	ord()	str()
any()	eval()	isinstance()	pow()	sum()
basestring()	execfile()	issubclass()	print()	super()
bin()	file()	iter()	property()	tuple()
bool()	filter()	len()	range()	type()
bytearray()	float()	list()	raw_input()	unichr()
callable()	format()	locals()	reduce()	unicode()
chr()	frozenset()	long()	reload()	vars()
classmethod()	getattr()	map()	repr()	xrange()
cmp()	globals()	max()	reversed()	zip()
compile()	hasattr()	memoryview()	round()	__import__()
complex()	hash()	min()	set()	apply()
delattr()	help()	next()	setattr()	buffer()
dict()	hex()	object()	slice()	coerce()
dir()	id()	oct()	sorted()	intern()

Operadores

- ==
- !=
- >
- <
- <=
- >=
- not
- and
- or

Tipos de datos

- Numéricos: integers, floats, longs, complex …
  - a = 3 b = 4 a/b int(0.75) pi = 3.14 round(2.8) b = 3L c = 2+1j
  - a,b = 6,4 a = b = 7
  - a = 3
  - a + z → Error
- Cadenas: string stuffing → \
  - cad = “Hola” ca = 'Hola' ”””
  - print cad
  - print “Saludo” + cad
  - print “Saludo”, cad
  - print “Saludo %s” %cad
  - Print “Saludo %s y %s” % (cad, ca)
  - cad[0]
  - upper() lower() strip()
  - Subcadenas: “o” in cad “Hola amigos”.find(“la”) len(“Adios”)
  -

Tipos de datos

- Vectores: listas, tuplas
  - Puede haber listas de cualquier tipo de elementos (incluso de diferentes tipos), un elemento de una lista es mutable (se puede modificar)

    lista_tiendas=["depor","infor","vest","rest"]

    for tienda in lista_tiendas: print tienda
  - Acceso a listas:

    lista=[3,5+1j,"hola"] lista[0]

    len(lista) lista.append(8) lista.insert(7,1) lista.remove(3) del lista[3]

    3 in lista 5 not in lista lista.index(3)
  - Elevamos al cuadrado todos los números de una lista

    a = [1, 5, 3, 7, 6, 3, 2, 4]

    [x ** 2 for x in a]

    → [1, 25, 9, 49, 36, 9, 4, 16]
  - Y ahora sólo aquellos que son pares

    a = [1, 5, 3, 7, 6, 3, 2, 4]

    [x ** 2 for x in a if x % 2 == 0]

    → [36, 4, 16] *Tachán!!*

Tipos de datos

- Vectores: listas, tuplas
  - Puede haber listas de cualquier tipo de elementos (incluso de diferentes tipos), un elemento de una lista es mutable (se puede modificar)

```
lista_tiendas=["depor","infor","vest","rest"]
for tienda in lista_tiendas: print tienda
```

  - Acceso a listas:

```
lista=[3,5+1j,"hola"]      lista[0]
len(lista)     lista.append(8)   lista.insert(7,1)   lista.remove(3)   del lista[3]
3 in lista      5 not in lista      lista.index(3)
```

- Otros: diccionarios (hashes), objetos ..
  - Hash, conjunto de pares ordenados {clave:valor}

```
dic_ej = {'Nombre': 'Jose', 'Eslora': 20, 'Manga': 6};
dict_ej['Eslora'] = 8
print "Eslora: ", dict_ej['Eslora']
```

Estructuras de control

- Cuando necesitamos modificar el flujo de ejecución del programa:
  - if
  - for
  - while

if

```
if 4 < 0:
    print “negativo!”
elif 4 == 0:
    print “nada!”
else:
     print “positivo!”
```

for

- Itera sobre los elementos de una secuencia:

```
for i in [1, 2, 3]:
        print i
```

- Generando listas:
  - range(7) → [0,1,2,3,4,5,6]
  - range(4,7) → [4,5,6]
  - range(ini, n, step) range(1,11,2) → [1,3,5,7,9]

```
for i in range(3)
```

  - No se usan condiciones de inicio, parada e incremento, sino que se especifica claramente qué elementos se utilizan
    - Más expresivo

```
numeros = [2, 4, 5]
for i in range(len(numeros)):
        print i
```

while

```
while x < 10:
    x = x + 1
    print x
```

- continue / break

Funciones

- Se definen con: 'def'

```
def saludo():
    print ”Hola!”
```

- Parámetros:

```
def saludo1 (nombre):
    print ”Muy buenas D. %s!” % (nombre)
def saludo2 (nombre=”Pepito”):
    print ”Muy buenas D. %s!” % (nombre)
```

- Para devolver valores utilizaremos 'return'

```
def suma(a,b):
    return a+b
```

Excepciones

- Una excepción se lanza cuando ha ocurrido algún error que Phyton no ”sabe” como manejar, y termina el programa
- Podemos capturarlas:
  - try:

```
        division(10,0)
    except ZeroDivisionError:
        print ”No se puede
    except:
        print ”Error inesperado”
    finally:
        print division(10,1)
```

Módulos y paquetes

- Módulo: Conjunto de funciones, clases y variables guardadas en un fi chero .py
- Paquete: Colección de módulos situados en un mismo directorio.

- Formas de usarlos:
  - import pygame

    pygame.draw.rect(playSurface,whiteColour,Rect(position[0], position[1], 20, 20))
  - from socket import *

    clientSocket = socket(AF_INET, SOCK_STREAM)
  - import Tkinter as tk

    tk.mainloop

Listas

- Listas
  - elevamos al cuadrado todos los números

```
>>> a = [1, 5, 3, 7, 6, 3, 2, 4]
>>> [x ** 2 for x in a]
[1, 25, 9, 49, 36, 9, 4, 16]
```

  - Y ahora sólo aquellos que son pares

```
>>> a = [1, 5, 3, 7, 6, 3, 2, 4]
>>> [x ** 2 for x in a if x % 2 == 0]
[36, 4, 16]
```

Tachán!!

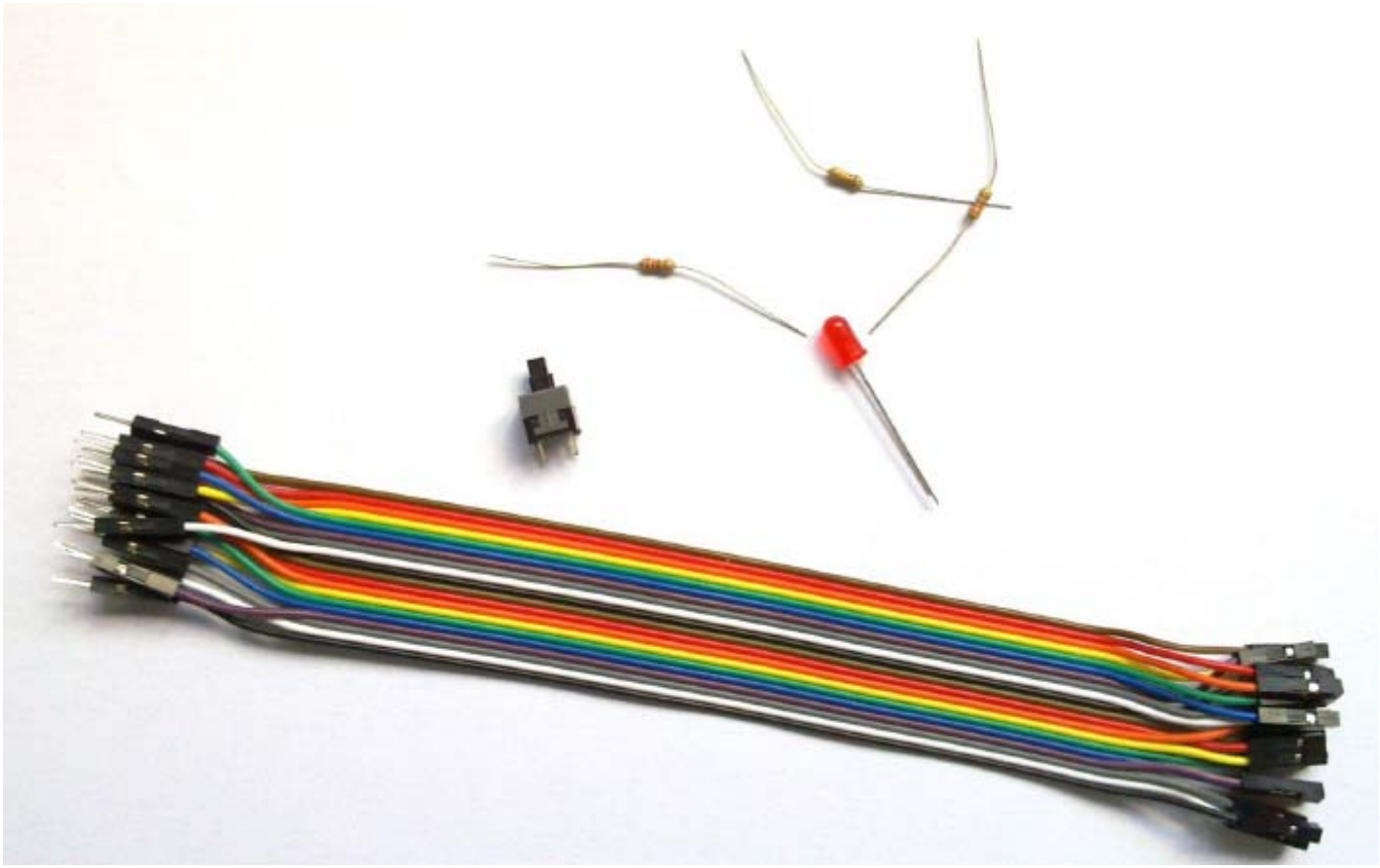
Acceso a bajo nivel: GPIO

- Biblioteca con básicamente funciones para la GPIO
  - Viene ya instalada con python en raspbian
  - Ejemplo:

```
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BOARD)
GPIO.setup(11, GPIO.OUT)
GPIO.setup(12, GPIO.IN)
GPIO.output(11, False)
input_value = GPIO.input(12)
if input_value == False:
            print "Se ha pulsado el boton"
```

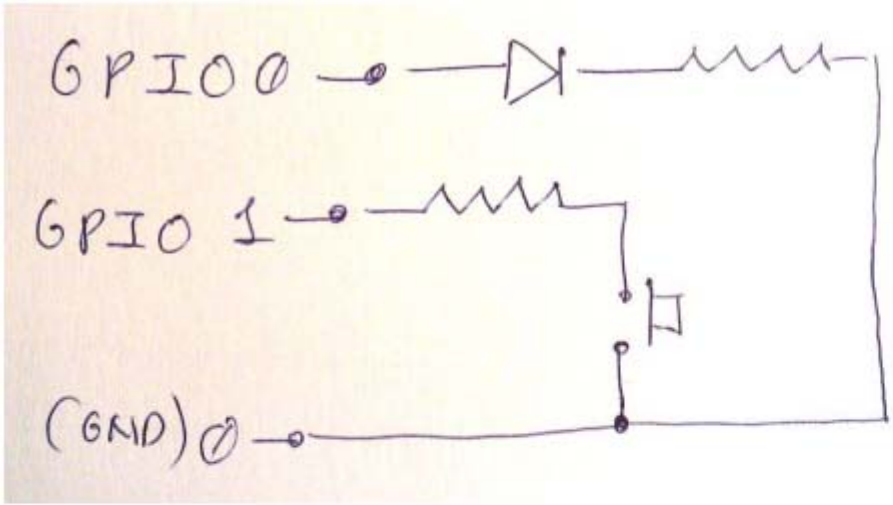
Un caso práctico: montaje

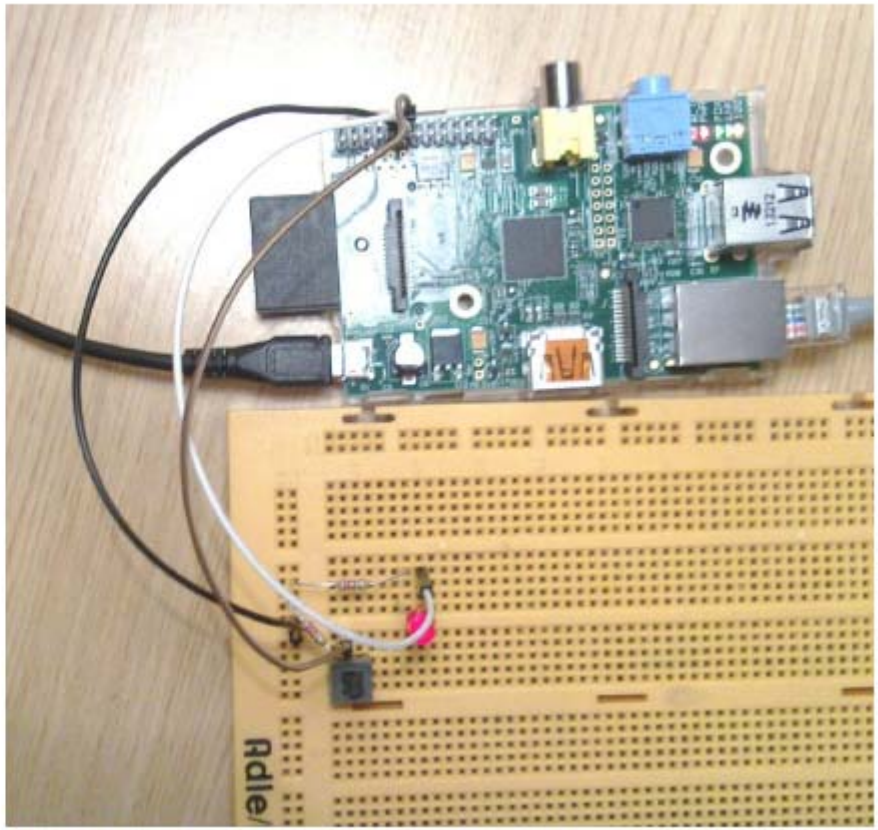
- Material: LED, pulsador, 2 resistencias de 220 Ohms y cables.



Un caso práctico: montaje

- Apagar la RPi, quitar la alimentación, montar con cuidado.





Acceso a bajo nivel: wiringPi

- Biblioteca con funciones para la GPIO, I2C, SPI, que soporta las placas de extensión más populares: piface, gertboard, …

www.wiringpi.com

```
$ sudo apt-get update
$ sudo apt-get install python-dev python-pip
$ sudo pip install wiringpi2
```

E/S y ficheros

- print
  - Convierte a strnig lo que le pases y lo saca por la salida estándar:
  - print "Python esta chulo", "verdad?"
- Hay dos funciones para leer de la entrada estándar (normalmente el teclado):
  - raw_input (siempre devuelve una cadena de texto → podemos necesitar hacer una conversión al tipo de dato que necesitemos)

```
str = raw_input("Enter your input: ");
print "Received input is : ", str
```

  - input

```
str = input("Enter your input: ");
print "Received input is : ", str
```

E/S y ficheros

- Antes de leer o escribir ficheros hay que abrirlos: open()
  - file object = open(file_name [, access_mode])
  - Crea un objeto “file”
- Parámetros:
  - file_name: nombre del fichero
  - access_mode: modo debe ser abierto (lectura, escritura, etc.)
- close() vuelva (flush) todas la información no escrita y cierra el fichero .
  - fileObject.close();

E/S y ficheros

- Leyendo ficheros:

```
fo = open("foo.txt", "r+")
str = fo.read(10);
print "Cadena ledia : ", str
# posicion actual
position = fo.tell();
print "Posicion actual : ", position
# Vamos al principio de nuevo
position = fo.seek(0, 0);
str = fo.read(10);
print "Cadena leida: ", str
fo.close()
```

- Escribiendo ficheros:

```
fo = open("foo.txt", "wb")
fo.write( "Practicando la escritura en ficheros.\n");
fo.close()
```

Gestionando ficheros

- Renombrando ficheros:

```
import os
# Renombrando test1.txt a test2.txt
os.rename( "test1.txt", "test2.txt" )
```

- Borrando ficheros:

```
import os
# Borrando test2.txt
os.remove("text2.txt")
```

- Llamar a otro programa:

```
import subprocess
subprocess.call(['miprogram', 'hola'])
```

Directorios en Python

- Crear directorios:

```
import os
# Creamos el directorio “test”
os.mkdir("test")
```

- Borrar directorios:

```
os.rmdir('test')
```

- Etc.

E/S y ficheros

- Modos de apertura:

Modes	Description
r	Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
rb	Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
r+	Opens a file for both reading and writing. The file pointer will be at the beginning of the file.
rb+	Opens a file for both reading and writing in binary format. The file pointer will be at the beginning of the file.
w	Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
wb	Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
w+	Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
wb+	Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
a	Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
ab	Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
a+	Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
ab+	Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

Networking: Un sencillo servidor

```
import socket               # Importamos el módulo socket
s = socket.socket()         # Creamos un objeto socket
host = socket.gethostname() # Obt nombre de la maquina local
port = 12345                # Puerto para nuestro servicio
s.bind((host, port))        # Asociamos
s.listen(5)                 # Esperamos que venga un cliente
while True:
   c, addr = s.accept()     # Establecemos la conexion.
   print 'Conectado con', addr
   c.send('Gracias por conectarse')
   c.close()                # Cerramos la conexión con el cliente
```

Networking: Un sencillo cliente

```
import socket               # Importamos el modulo socket

s = socket.socket()         # Creates un objeto socket
host = socket.gethostname() # Obt el nombre de nuestra maquina
port = 12345                # Puerto para nuestro servicio

s.connect((host, port))
print s.recv(1024)
s.close                     # Cerramos el socket al finalizar
```

Enviando un mail..

```
#!/usr/bin/python
import smtplib
sender = 'from@fromdomain.com'
receivers = ['to@todomain.com']
message = """From: From Persona <from@fromdomain.com>
To: To Person <to@todomain.com>
Subject: E-Mail desde la RPi
Esto es una prueba de e-mail desde la Raspberry Pi.
"""
try:
   smtpObj = smtplib.SMTP('smtp.upv.es',25)
   smtpObj.sendmail(sender, receivers, message)
   print "Successfully sent email"
except SMTPException:
   print "Error: unable to send email"
```

Programando GUIs con Python

- Hay bastantes opciones para desarrollar GUIs. Algunas son:
  - Tkinter: viene ya (Tk GUI toolkit)
  - wxPython: open-source Python interface para wxWindows http://wxpython.org.
  - JPython: http://www.jython.org.
  - ..

Programando GUIs con Python: Tkinter

- Tkinter es la librería GUI estándar para Python.
- Permite desarrollar interfaces gráficas de usuario de manera sencilla

- Pasos para crear una aplicación con GUI usando Tkinter:

  - Importar el módulo Tkinter.
  - Crear la ventana principal de la aplicación
  - Añadir uno o más elementos gráficos.
  - Entrar en el bucle “mainloop”

Programando GUIs con Python: Tkinter

- Ejemplo:

```
import Tkinter
top = Tkinter.Tk()
top.mainloop()
```

Programando GUIs con Python: Tkinter

- Los widgets más comunes disponibles en Tkinter
  - Button
  - Canvas
  - Checkbutton
  - Entry
  - Label
  - Listbox
  - Menu
  - Radiobutton
  - Text
  - tkMessageBox

Programando GUIs con Python: Tkinter

- Button:

```
import Tkinter
import tkMessageBox

top = Tkinter.Tk()

def holaCallBack():
   tkMessageBox.showinfo( "Hola a todos", "Hola mundo")

B = Tkinter.Button(top, text ="Hola", command = holaCallBack)

B.pack()
top.mainloop()
```

Programando GUIs con Python: Tkinter

- Entry:

```
from Tkinter import *


top = Tk()
L1 = Label(top, text="Nombre")
L1.pack( side = LEFT)
E1 = Entry(top, bd =5)

E1.pack(side = RIGHT)
top.mainloop()
```

Programando GUIs con Python: Tkinter

- Ejemplo Checkbutton:

```
from Tkinter import *
import tkMessageBox
import Tkinter
top = Tkinter.Tk()
CheckVar1 = IntVar()
CheckVar2 = IntVar()
C1 = Checkbutton(top, text = "Music", variable = CheckVar1, \
        onvalue = 1, offvalue = 0, height=5, \
        width = 20)
C2 = Checkbutton(top, text = "Video", variable = CheckVar2, \
        onvalue = 1, offvalue = 0, height=5, \
        width = 20)
C1.pack()
C2.pack()
top.mainloop()
```

Programando GUIs con Python: Tkinter

- Canvas:
  - arc

```
coord = 10, 50, 240, 210
arc = canvas.create_arc(coord, start=0, extent=150,
fill="blue")
```

  - image

```
filename = PhotoImage(file = "sunshine.gif")
image = canvas.create_image(50, 50, anchor=NE, image=filename)
```

  - line

```
line = canvas.create_line(x0, y0, x1, y1, ..., xn, yn, options)
```

  - oval

```
oval = canvas.create_oval(x0, y0, x1, y1, options)
```

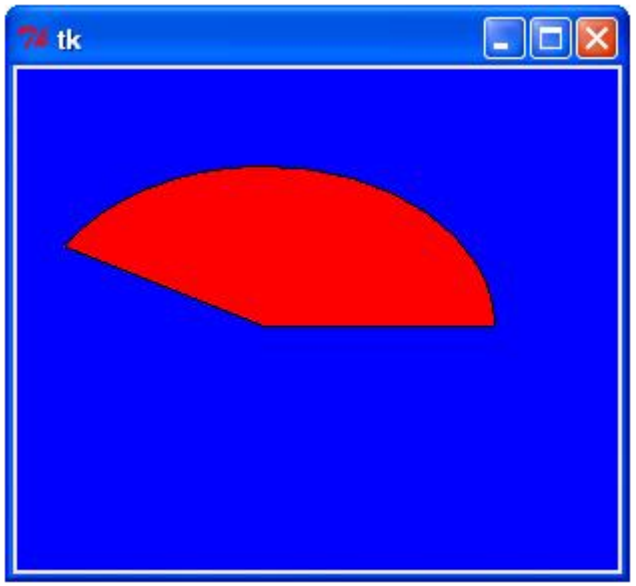
  - polygon

```
pol = canvas.create_polygon(x0, y0, x1, y1,...xn, yn, options)
```

Programando GUIs con Python: Tkinter

- Ejemplo Canvas:

```
import Tkinter
import tkMessageBox
top = Tkinter.Tk()
C = Tkinter.Canvas(top, bg="blue", height=250, width=300)
coord = 10, 50, 240, 210
arc = C.create_arc(coord, start=0, extent=150, fill="red")
C.pack()
top.mainloop()
```



Repasando con un juego...

```
#!/usr/bin/env python
# Raspberry Snake. Written by Gareth Halfacree for the Raspberry Pi User Guide
# http://media.wiley.com/product_ancillary/82/11187954/DOWNLOAD/raspberrysnake.py
import pygame, sys, time, random
from pygame.locals import *
pygame.init()

fpsClock = pygame.time.Clock()
playSurface = pygame.display.set_mode((640, 480))
pygame.display.set_caption('Raspberry Snake')

redColour = pygame.Color(255, 0, 0)
blackColour = pygame.Color(0, 0, 0)
whiteColour = pygame.Color(255, 255, 255)
greyColour = pygame.Color(150, 150, 150)
snakePosition = [100,100]
snakeSegments = [[100,100],[80,100],[60,100]]
raspberryPosition = [300,300]
raspberrySpawned = 1
direction = 'right'
changeDirection = direction
```

Repasando con un juego...

```
def gameOver():
 gameOverFont = pygame.font.Font('freesansbold.ttf', 72)
 gameOverSurf = gameOverFont.render('Game Over', True, greyColour)
 gameOverRect = gameOverSurf.get_rect()
 gameOverRect.midtop = (320, 10)
 playSurface.blit(gameOverSurf, gameOverRect)
 pygame.display.flip()
 time.sleep(5)
 pygame.quit()
 sys.exit()
```

Repasando con un juego...

```
while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
        elif event.type == KEYDOWN:
            if event.key == K_RIGHT or event.key == ord('d'):
                changeDirection = 'right'
            if event.key == K_LEFT or event.key == ord('a'):
                changeDirection = 'left'
            if event.key == K_UP or event.key == ord('w'):
                changeDirection = 'up'
            if event.key == K_DOWN or event.key == ord('s'):
                changeDirection = 'down'
            if event.key == K_ESCAPE:
                pygame.event.post(pygame.event.Event(QUIT))
    if changeDirection == 'right' and not direction == 'left':
        direction = changeDirection
    if changeDirection == 'left' and not direction == 'right':
        direction = changeDirection
    if changeDirection == 'up' and not direction == 'down':
        direction = changeDirection
    if changeDirection == 'down' and not direction == 'up':
        direction = changeDirection
    if direction == 'right':
        snakePosition[0] += 20
    if direction == 'left':
        snakePosition[0] -= 20
    if direction == 'up':
        snakePosition[1] -= 20
    if direction == 'down':
        snakePosition[1] += 20
```

Repasando con un juego...

```
snakeSegments.insert(0,list(snakePosition))
 if snakePosition[0] == raspberryPosition[0] and snakePosition[1] == raspberryPosition[1]:
    raspberrySpawned = 0
 else:
    snakeSegments.pop()
 if raspberrySpawned == 0:
    x = random.randrange(1,32)
    y = random.randrange(1,24)
    raspberryPosition = [int(x*20),int(y*20)]
 raspberrySpawned = 1
 playSurface.fill(blackColour)
 for position in snakeSegments:
    pygame.draw.rect(playSurface,whiteColour,Rect(position[0], position[1], 20, 20))
 pygame.draw.rect(playSurface,redColour,Rect(raspberryPosition[0], raspberryPosition[1], 20, 20))
 pygame.display.flip()
 if snakePosition[0] > 620 or snakePosition[0] < 0:
    gameOver()
 if snakePosition[1] > 460 or snakePosition[1] < 0:
 for snakeBody in snakeSegments[1:]:
    if snakePosition[0] == snakeBody[0] and snakePosition[1] == snakeBody[1]:
       gameOver()
 fpsClock.tick(30)
```

- sudo shutdown -h now