PyS60: Python para teléfonos Symbian

Aplicaciones Móviles
Curso de Adaptación
Grado en Ingeniería de Sistemas Audiovisuales

Celeste Campo - Carlos García Rubio
celeste, cgr@it.uc3m.es

Índice

- Introducción.
- Instalación.
- “Hola mundo”.
- Breve introducción a Python.
- Módulos PyS60.
- Depuración con PyS60.
- Referencias.

Introducción

- PyS60 = Python para S60.
- Python:
  - Lenguaje de programación creado por Guido van Rossum en el año 1991.
  - Sintaxis sencilla (cercana a lenguaje natural).
  - Interpretado (o de *script*):
    - Se ejecuta utilizando un programa intermedio llamado intérprete.
    - En realidad es semi interpretado código fuente se traduce a un pseudo código máquina intermedio llamado *bytecode* la primera vez que se ejecuta, generando archivos .pyc o .pyo (bytecode optimizado).
  - Tipado dinámico:
    - No es necesario declarar el tipo de dato que va a contener una determinada variable.
    - Se determinará en tiempo de ejecución según el tipo del valor que se asigne.
    - El tipo de esta variable puede cambiar si se le asigna un valor de otro tipo.

Introducción

- Python (sigue):
  - Fuertemente tipado:
    - No se permite tratar a una variable como si fuera de un tipo distinto al que tiene, es necesario convertir de forma explícita dicha variable al nuevo tipo.
  - Multiplataforma:
    - El intérprete de Python está disponible en multitud de plataformas (Linux, Windows, Mac OS, etc.).
    - Si no utilizamos librerías específicas, el programa podrá correr en todos estos sistemas sin grandes cambios.
  - Orientado a objetos:
    - Python permite programación orientado a objetos, pero también imperativa, funcional y orientada a aspectos.
  - Modular:
    - Permite agrupar elementos relacionados en módulos reutilizables desde otros programas python.
  - Gran cantidad de librerías disponibles:
    - Hay una librería estándar, *Python Standard Library,* muy completa.

Introducción

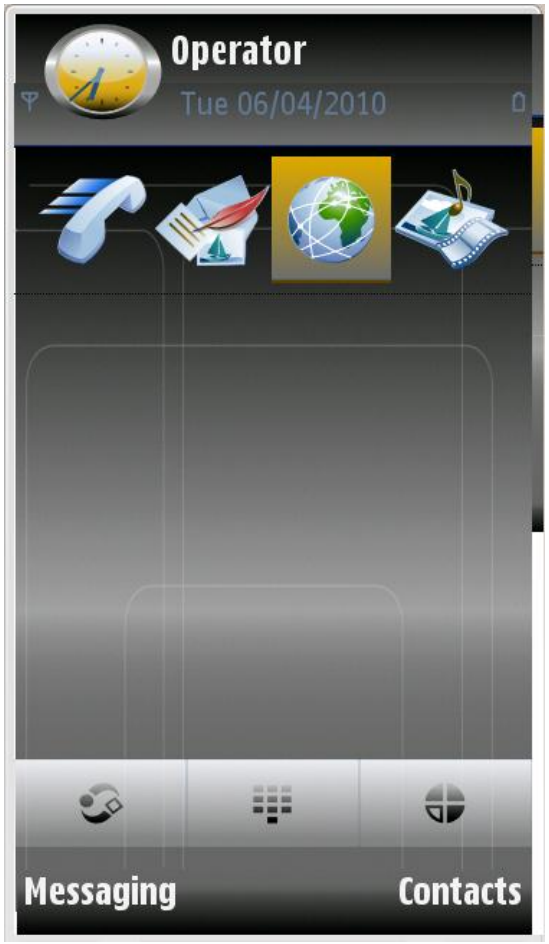
- PyS60:
  - Incluye muchas de las librerías estándar de Python.
  - Más muchos módulos adicionales específicos para móviles.
    - Por ejemplo: acceso a SMS y MMS, teléfono, cámara, red, bluetooth, grabación y reproducción de sonido, *text-to-speach*, sensores, GPS, librerías gráficas 2D y 3D, soporte de pantalla táctil, etc.
    - Proporcionan un interfaz muy simple para funciones complejas.
  - Hay que instalar en el teléfono el entorno de ejecución Python, para que lo interprete.
  - También podemos generar una aplicación Symbian autocontenida, que incluya el código y el intérprete.
  - La seguridad se gestiona a través del marco de seguridad de la plataforma Symbian.
    - No hay el concepto de *sandbox*.

Instalación de PyS60 en el móvil

- Instalar el Nokia PC Suite: http://europe.nokia.com/A4144903
- Descargar e instalar PyS60:
  - 1.4.x: http://sourceforge.net/projects/pys60/
    - Basado en Python 2.3.
  - 1.9.x+: https://garage.maemo.org/projects/pys60/
    - Basado en Python 2.5, soporte de sensores y pantallas táctiles.
  - Última versión: 2.0.0 (11/02/2010).
- En el teléfono, instalar:
  - Software: PythonForS60_1_x_x_3rdEd.SIS
  - Script shell: PythonScriptShell_1_x_x_3rdEd.SIS
- Aparecerá el icono de Python en el menú de aplicaciones.

Instalación en el PC

- Descargar el *S60 Platform SDK for Symbian* de http://forum.nokia.com
- Descargar el *Python plug-in for the Symbian SDK* de http://sourceforge.net/projects/pys60/ o de https://garage.maemo.org/projects/pys60/
  - PythonForS60_1_x_x_SDK_3rdEd.zip
- Extraer el plug-in de python en el directorio donde está instalado el *S60 Platform SDK*.
- Arrancar el *S60 Emulator*.



IDEs

- Podemos utilizar cualquier editor de texto.
  - Muchos entienden la sintaxis de Python y muestran el código coloreado.
- Hay algunos IDEs diseñados para Python. Por ejemplo:
  - IDLE:
    - http://docs.python.org/library/idle.html
  - PythonWin + Win32 Extensions
    - http://sourceforge.net/projects/pywin32/
  - PyDev Eclipse/Carbide.c++ Plug-in
    - http://pydev.sf.net/
  - SPE
    - http://pythonide.stani.be/

"Hola mundo"

- Crear con un editor un fichero "`hola.py`" con el siguiente contenido:

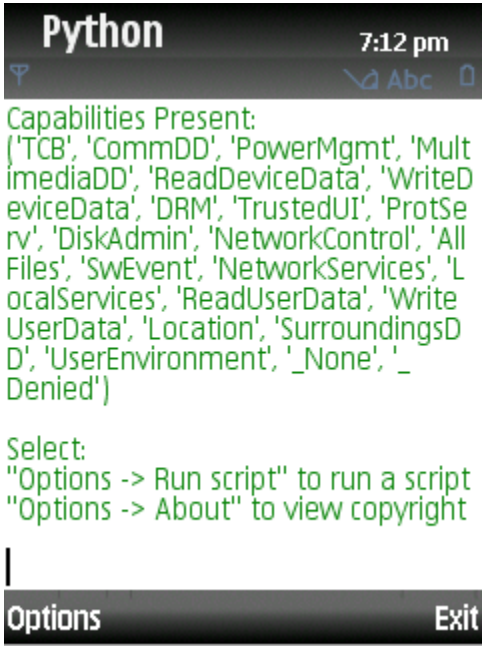
```
print "Hola mundo"
```

- Copiar el fichero al teléfono:
  - En un terminal real, conectarlo al PC y copiarlo usando el *PC Suite* al directorio `E:\Python` (tarjeta de memoria) del teléfono.
  - En el emulador, copiarlo al directorio `<epocroot>\winscw\c\Data\python\`
    - En nuestra máquina virtual, `C:\S60\devices\S60_5th_Edition_SDK_v1.0\epoc32\winscw\c\Data\python`
- Lanzar el script.

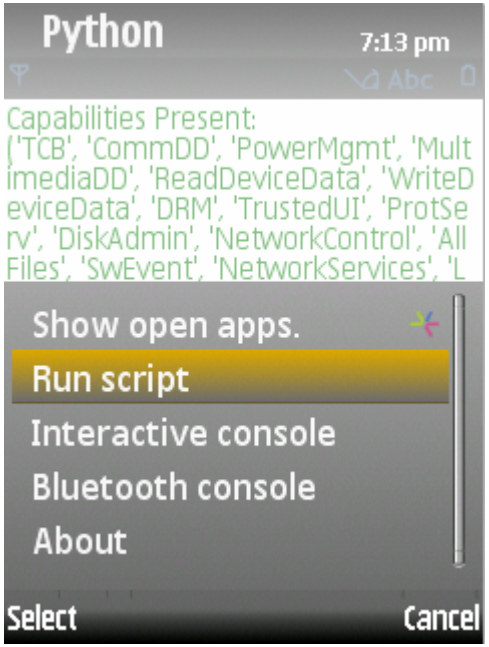
“Hola mundo”


Menu
7:11 pm
Calendar
Contacts
Log
Web
Messaging
Gallery
RealPlayer
Music player
Radio
Settings
Help
Applications
Options
Open
Exit

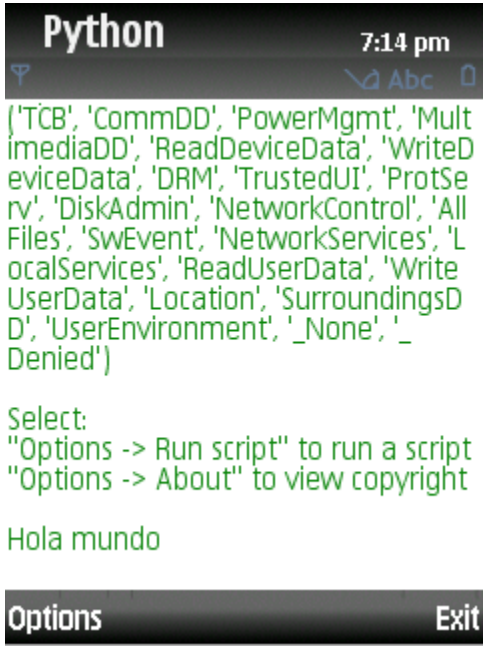

Applications
7:11 pm
Search
Camera
User Dict.
Debug Agent
Bilingual D...
eshell
Location
Notes
Oper. menu
Python
testapp
Options
Open
Back


Python
7:12 pm
Abc
Capabilities Present:
('TCB', 'CommDD', 'PowerMgmt', 'MultimediaDD', 'ReadDeviceData', 'WriteDeviceData', 'DRM', 'TrustedUI', 'ProtServ', 'DiskAdmin', 'NetworkControl', 'AllFiles', 'SwEvent', 'NetworkServices', 'LocalServices', 'ReadUserData', 'WriteUserData', 'Location', 'SurroundingsDD', 'UserEnvironment', '_None', '_Denied')
Select:
"Options -> Run script" to run a script
"Options -> About" to view copyright
Options
Exit

“Hola mundo”


Python
7:13 pm
Capabilities Present:
('TCB', 'CommDD', 'PowerMgmt', 'Mult
imediaDD', 'ReadDeviceData', 'WriteD
eviceData', 'DRM', 'TrustedUI', 'ProtSe
rv', 'DiskAdmin', 'NetworkControl', 'All
Files', 'SwEvent', 'NetworkServices', 'L
Show open apps.
Run script
Interactive console
Bluetooth console
About
Select
Cancel


Python
7:14 pm
c:hola.py
OK
Cancel


Python
7:14 pm
('TCB', 'CommDD', 'PowerMgmt', 'Mult
imediaDD', 'ReadDeviceData', 'WriteD
eviceData', 'DRM', 'TrustedUI', 'ProtSe
rv', 'DiskAdmin', 'NetworkControl', 'All
Files', 'SwEvent', 'NetworkServices', 'L
ocalServices', 'ReadUserData', 'Write
UserData', 'Location', 'SurroundingsD
D', 'UserEnvironment', '_None', '_
Denied')
Select:
"Options -> Run script" to run a script
"Options -> About" to view copyright
Hola mundo
Options
Exit

Un “Hola mundo” un poco mejor

```
import appuifw

data = appuifw.query(u"Escribe una frase:", "text")

appuifw.note(u"Has escrito: " + data, "info")
```

Un “Hola mundo” un poco mejor

```
import appuifw

data = appuifw.query(u"Escribe una frase:", "text")

appuifw.note(u"Has escrito: " + data, "info")
```

- Importamos el módulo *appuifw* que nos permite mostrar un objeto en el interfaz de usuario (*UI widget*).
- Un módulo en Python agrupa varias funciones relacionadas. Para referirnos a una de ellas, usamos `nombre_módulo.nombre_función`, como en `appuifw.query()`
- La función `query()` abre una ventana de diálogo. Toma dos parámetros, `label` y `type`, y devuelve una cadena de caracteres.

Un “Hola mundo” un poco mejor

```
import appuifw

data = appuifw.query(u"Escribe una frase:", "text")

appuifw.note(u"Has escrito: " + data, "info")
```

- El primer parámetro es una cadena de texto.
  - La `u` al comienzo de `u"Escribe una frase:"` declara la cadena de texto como unicode, que es lo que manejan los teléfonos.
- El segundo parámetro indica qué tipo de diálogo queremos.
  - Hay `text`, `code`, `number`, `date`, `time`, `query` y `float`.
- Con `data = appuifw.query(…)` asignamos el resultado de la función `query()` a la variable `data` (no es necesario definirla primero).

Un “Hola mundo” un poco mejor

```
import appuifw

data = appuifw.query(u"Escribe una frase:", "text")

appuifw.note(u"Has escrito: " + data, "info")
```

- La función `appuifw.note(label, type)` saca un mensaje emergente con el texto en `label`, del tipo indicado en `type`.
  - El primer parámetro se forma concatenando dos cadenas de caracteres.
  - El segundo indica que queremos que sea de tipo `info`.
    - Puede ser `info`, `error` o `conf`.
- Antes de seguir, conviene estudiar un poco de Python.

Breve introducción a Python

- Variables:
  - `a=7`
  - `a="Hola"`
  - En Python no es necesario declarar el tipo de datos que una variable va a almacenar. Python lo gestiona de forma dinámica.
  - Los nombres de las variables pueden tener una longitud arbitraria, pudiendo contener letras y números. Siempre deben empezar con una letra.
  - Se considera que una letra minúscula es diferente de la misma letra mayúscula.
  - Gestión de memoria automática.
    - Cuenta del número de referencias, recolección de basura
    - Con la sentencia `del` se puede liberar explícitamente.

Breve introducción a Python

- Cadenas de caracteres (*strings*):
  - ```
    print "Hola %s, Hola %s" ("Mundo", Python)
    print "Host: %s\tPort: %d" % ("Earth", 80)
    print "DD.MM.YYYY = %02d.%02d.%d" % (12, 3, 82)
    ```
  - El resultado mostrado por pantalla por las llamadas anteriores será:

    ```
    Hola Mundo, Hola Python.
    Host: Earth Port: 80
    DD.MM.YYYY = 12.03.82
    ```
  - Comillas dobles y comillas simples:
    - `"Python"`
    - `'Python'`
    - En Python las comillas dobles y las simples sirven para lo mismo. No existe diferencia entre el uso de una u otra.
    ```

Breve introducción a Python

- Manejo de cadenas:

```
txt = "I like Python”
print txt[2:6]
print txt.find("like“)

if txt.find("love") == -1:
  print "What's wrong with you?”

print txt.upper()
print "Length", len(txt)
txt2 = “”
if txt2:
  print "txt2 contains characters”
else:
  print "txt2 doesn't contain characters"
```

Salida:
like
2
What's wrong with you?
I LIKE PYTHON
Length 13
txt2 doesn't contain characters

Breve introducción a Python

- Manejo de cadenas:

```
url= " http://www.mopius.com "
url= url.strip()
if url.startswith("http://"):
  print url, "is a valid URL"

webServiceInput= " 1, 2, 3, 4 "
print webServiceInput.replace(" ", "")

txt = "one;two;three"
print txt.split(";")
```

Salida:
http://www.mopius.com is a valid URL
1,2,3,4
['one', 'two', 'three']

Breve introducción a Python

- Listas:
  - `nombres = [u"Python", u"PyS60", u"Symbian"]`
  - Son conjuntos ordenados con una serie de valores almacenados, encerrados entre corchetes.
- Tuplas:
  - `(1,2,3,4)` tupla de 4 elementos.
  - Una tupla es una lista que no se puede cambiar a lo largo de la ejecución del programa.
  - Los elementos de la tupla se separan con comas y los valores se encierran entre paréntesis.

Breve introducción a Python

- Diccionarios:
  - `{'jamon':2, 'queso':3}`
  - Son estructuras tipo “clave:valor”. Cada clave tiene asociado un valor determinado. Estas parejas se encierran entre llaves.
- Comentarios:
  - `# Esto es un comentario`
  - Los comentarios son líneas no ejecutables del programa. El intérprete de Python ignora los comentarios, pero son de gran utilidad para documentar el programa. Empiezan con almohadilla y terminan en el final de la línea.

Breve introducción a Python

- Condición *if*:

```
if x>0:
    # Realiza la acción 1
elseif x==2:
    # Realiza la acción 2
elseif x==3:
    # Realiza la acción 3
else:
    # Realiza la acción 4
```

- Bucle *for*:

```
nombres = [u"Python", u"PyS60", u"Symbian"]
for elemento in nombres:
    print elemento
```

Breve introducción a Python

- Bucle *for*:

```
# Iterando en una cadena
for eachLetter in "Text":
    print "current letter:", eachLetter

# Iterating by sequence item
nameList= ["Mike", "Sarah", "Charles"]
for eachName in sorted(nameList):
    print eachName

# Iterándo por un índice de secuencia
for nameIndex in range(len(nameList)):
    print nameList[nameIndex]
```

Salida:
current letter: T
current letter: e
current letter: x
current letter: t

Salida:
Charles
Mike
Sarah

Salida:
Mike
Sarah
Charles

Breve introducción a Python

- Bucle *for*:

```
# Iterando en un rango
for eachVal in range(3):
    print "value: ", eachVal
```

Salida:
value: 0
value: 1
value: 2

```
# Sintaxis extendida:
# range(start, end, step = 1)
for eachVal in range(2, 10, 3):
    print "value: ", eachVal
```

Salida:
value: 2
value: 5
value: 8

Breve introducción a Python

- Bucle *while*:
  - Añade respecto a otros lenguajes un `else` que se ejecuta si no se abandona el bucle con un `break`.

```
def showMaxFactor(num):
    count = num / 2
    while count > 1:
        if num % count == 0:
            print "Largest factor of %d is %d" % (num, count)
            break
        count -= 1
    else:
        print num, "is prime"

for eachNum in range(10, 21):
    showMaxFactor(eachNum)
```

Salida:

```
Largest factor of 10 is 5
11 is prime
Largest factor of 12 is 6
13 is prime
Largest factor of 14 is 7
Largest factor of 15 is 5
Largest factor of 16 is 8
17 is prime
Largest factor of 18 is 9
19 is prime
Largest factor of 20 is 1
```

Breve introducción a Python

- Funciones:
  - Sintaxis:

```
def function_name(arguments):
    ["function_documentation_string"]
    function_body_suite
```

  - Ejemplo:

```
def foo():
  "foo() --does'tdo anything special."
  print "in foo"

# Execute the function
foo()
# Print the help text of the function
print foo.__doc__
# foo() --does'tdo anything special.
```

Breve introducción a Python

- Funciones:
  - Argumentos por defecto:

```
def calcTax(amount, rate=0.0275):
  return amount + (amount * rate)
print calcTax(100)# 102.75
print calcTax(100, 0.05)# 105.0
```

  - Pueden tener un número de argumentos variable.
- También se pueden definir clases (no lo vamos a ver aquí).

Breve introducción a Python

- Ámbito de las variables:
  - Global:
    - Las definidas fuera de una función.
    - Duran mientras el script siga ejecutándose.
  - Local:
    - Vida temporal, mientras la función en que están definidas siga activa.
  - Búsqueda de identificadores:
    - Primero local, luego global.
    - Se puede definir una local con el mismo nombre que una global.
      - Crea una nueva variable salvo que se use global.

Breve introducción a Python

- Para más información, ver:
  - "The Python Tutorial", http://docs.python.org/tutorial/
  - "Python para todos", Raúl González Duque, http://mundogeek.net/tutorial-python/
- Para consultar las funciones incluidas en la librería estándar de Python, ver http://docs.python.org/library/
  - Recordar que PyS60 1.4.x se basa en Python 2.3, y PyS60 1.9.x y 2.0.x se basa en Python 2.5.

Módulos PyS60

- PyS60, además de muchas de las funciones definidas en la la librería estándar de Python, viene con un conjunto de módulos específicos de teléfonos móviles.
- Para verlos en detalle, consultar el "*S60 Module Reference*" en https://garage.maemo.org/projects/pys60/

Módulos PyS60

- *appuifw*:
  - Proporciona una interfaz para las aplicaciones S60: Cuadros de diálogo, listas de selección, notas, *pop-up*, etc.
- *audio*:
  - Permite la reproducción y grabación de ficheros de audio y provee de acceso a al motor *text-to-speech.*
- *messaging*:
  - Proporciona APIs para el envío de SMS y MMS.
- *inbox*:
  - Proporciona herramientas para la lectura de SMS entrantes en el terminal, así como su contenido, tiempo de llegada y número del emisor. También permite borrar SMS.

Módulos PyS60

- *camera*:
  - Permite tomar fotografías y activar/apagar el visor de imágenes.
- *graphics*:
  - Proporciona acceso a las primitivas para gráficos 2D, así como la carga de imágenes en el terminal. Permite realizar acciones como el guardado de imágenes, redimensionado y transformación.
- *sysinfo*:
  - Ofrece un API para comprobar la información del sistema del terminal móvil S60: nivel de batería, código IMEI, nivel de señal o espacio disponible en memoria.

Módulos PyS60

- *gcanvas*:
  - Proporciona una interfaz de usuario para la presentación de gráficos OpenGL ES.
- *gles*:
  - Proporciona una capa entre Python y los gráficos OpenGL ES 3D/2D.
- *location*:
  - Ofrece un API de servicios de localización: Lectura de *Cell-ID*, etc.
- *positioning*:
  - Proporciona un acceso básico a la información de localización geográfica del terminal.

Módulos PyS60

- *btsocket (*antes *socket)*:
  - Extensiones S60 para el módulo estándar de Python *socket*, para soporta los protocolos Bluetooth RFCOMM y OBEX, y la configuración del punto de acceso por defecto.
- *urllib*:
  - Proporciona un interfaz de alto nivel para la carga de datos a través de la *World Wide Web*.
- *httplib*:
  - Proporciona una interfaz con el protocolo HTTP.
- *telephone*:
  - Proporciona un API para manejar diversas funcionalidades del teléfono como el marcado o el descolgado.

Módulos PyS60

- *calendar:*
  - Proporciona un API para el manejo de los servicios de calendario: Lectura, creación de entradas, configuración de alarmas.
- *contacts:*
  - Proporciona un API para el manejo de los servicios del libro de direcciones, permitiendo la creación de una base de datos con información de nuestros contactos.
- *sensor*:
  - Permite acceder a los sensores físicos del teléfono :
    - Acelerómetro.
    - Pulsar dos veces en la pantalla (*tapping*).
    - Sensor de rotación
- *keycapture:*
  - Ofrece un API para la captura global de eventos de teclado.

Módulos PyS60

- *topwindow:*
  - Interfaz para la creación de ventanas que vayan a ser mostradas en la parte superior de otras aplicaciones.
- *globalui*:
  - Permite a aplicaciones que no tienen UI mostrar notas o diálogos de petición.
- *e32:*
  - Proporciona utilidades relacionadas con el sistema operativo Symbian que no están referidas al interfaz del usuario y que no son proporcionadas por la librería de módulos estándar de Python.
- *e32db* y *e32dbm:*
  - Proporciona un API para la manipulación de bases de datos relacionales con una sintaxis SQL limitada.
- *logs*:
  - Permite acceder a los *log* del teléfono.

appuifw: elementos de la UI

- La estructura del UI de una aplicación PyS60 es la misma que la de otras aplicaciones S60.
- Se accede a este interfaz a través del objeto `app` dentro de *apfuifw*.
- Los elementos `title`, `body`, `menu` y `exit_key_handler` son variables del objeto `app`.

appuifw: título

- Para cambiar el título: `appuifw.app.title= u"Hola Mundo"`

```
import appuifw

appuifw.app.title= u"Hola Mundo"
data = appuifw.query(u"Escribe una frase:", "text")
appuifw.note(u"Has escrito: " + data, "info")
```

appuifw: menú de aplicación

```
import appuifw, e32

def play():
    print "Play file"

def volume_up():
    print "Volume up"

def volume_down():
    print "Volume down"

def quit():
    print "Exit key pressed"
    app_lock.signal()

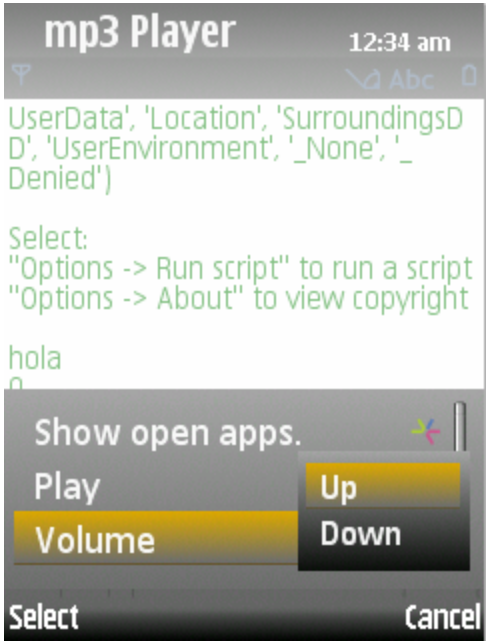
appuifw.app.exit_key_handler= quit
appuifw.app.title= u"mp3 Player"
appuifw.app.menu= [(u"Play", play),
     (u"Volume", ( (u"Up", volume_up), (u"Down", volume_down) ) )]

print"App is now running"
app_lock= e32.Ao_lock()
app_lock.wait()
```

Ya veremos para qué sirve esto.

Ya veremos para qué sirve esto.

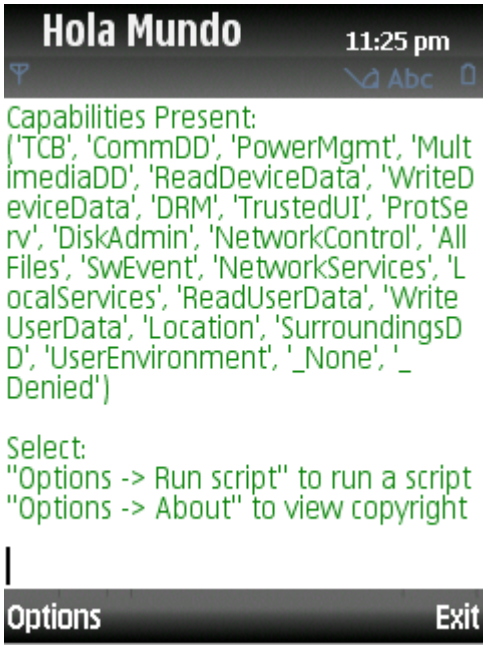
Ya veremos para qué sirve esto.



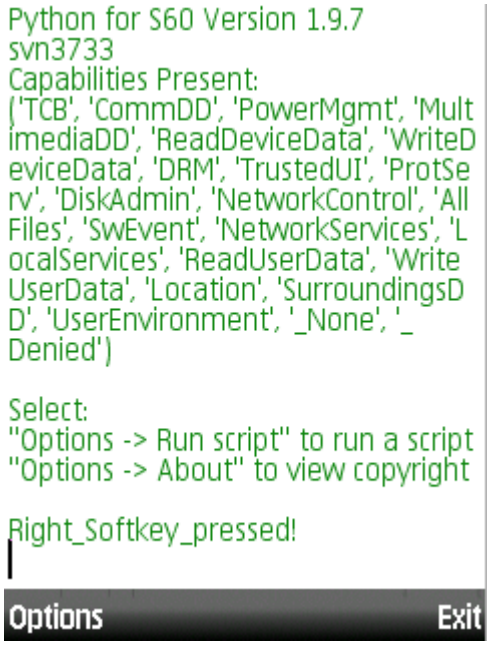
appuifw: tipos de pantalla

- No todas las aplicaciones tienen que tener todos los elementos de la UI.
- Con la variable `appuifw.app.screen` podemos controlar el tipo de pantalla que va a usar nuestra aplicación:

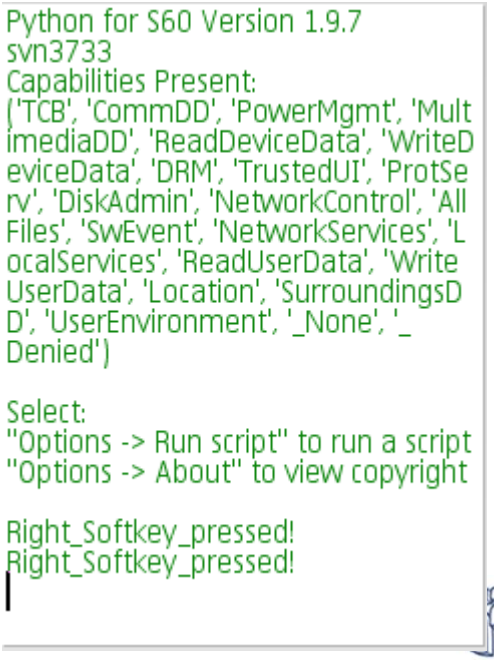
`appuifw.app.screen="normal"`



`appuifw.app.screen="large"`



`appuifw.app.screen="full"`



appuifw: cuerpo

- Se puede asignar distintos tipos de objeto al cuerpo (“appuifw.app.body”) de la aplicación:
  - **Text**: texto
  - **Canvas**: proporciona un área de pantalla en la que se puede dibujar, y soporta manejar los eventos de teclado
  - **Listbox**: muestra una lista de items.
  - **Form**: formularios complejos con varios campos de entrada.

appuifw: cuerpo *Text*

```
import appuifw
import e32

def exit_key_handler():
    app_lock.signal()

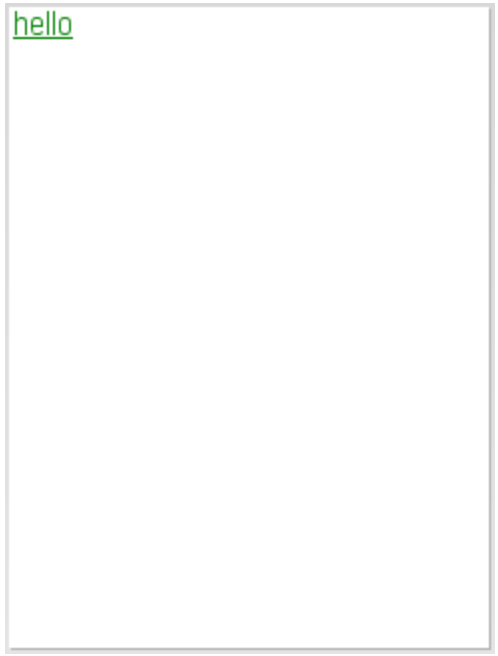
# create an instance of appuifw.Text()
round = appuifw.Text()
# change the style of the text
round.style = appuifw.STYLE_UNDERLINE
# set the text to 'hello'
round.set(u'hello')

# put the screen size to full screen
appuifw.app.screen='full'

# create an Active Object
app_lock = e32.Ao_lock()

# set the application body to Text
# by handing over "round" which is an instance of appuifw.Text() as definded above
appuifw.app.body = round

appuifw.app.exit_key_handler = exit_key_handler
app_lock.wait()
```



appuifw: cuerpo *Listbox*

```
import appuifw
import e32

def exit_key_handler():
    app_lock.signal()

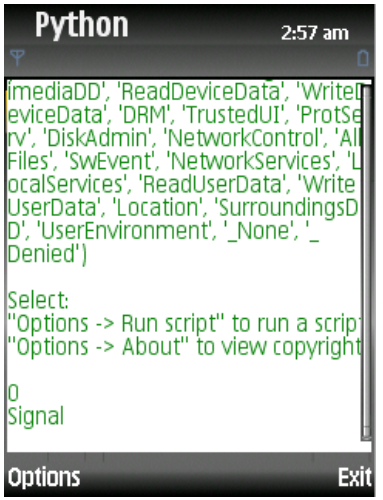
# define a callback function
def shout():
    index = lb.current()
    print index
    print entries[index]

# create your content list of your listbox
entries = [u"Signal",u"Battery"]
lb = appuifw.Listbox(entries,shout)

# create an Active Object
app_lock = e32.Ao_lock()

# create an instance of appuifw.Listbox(), include the content list "entries"
# and the callback function "shout"and set the instance of Listbox now as the
# application body
appuifw.app.body = lb

appuifw.app.exit_key_handler = exit_key_handler
app_lock.wait()
```



appuifw: pestañas *(tabs)*

```
import appuifw
import e32

# define application 1: text app
app1 = appuifw.Text(u'Appliation o-n-e is on')
# define application 2: text app
app2 = appuifw.Text(u'Appliation t-w-o is on')
# define application 3: text app
app3 = appuifw.Text(u'Appliation t-h-r-e-e is on')

def exit_key_handler():
    app_lock.signal()

# create a tab handler that switches the application based on what tab is selected
def handle_tab(index):
    global lb
    if index == 0:
        appuifw.app.body = app1 # switch to application 1
    if index == 1:
        appuifw.app.body = app2 # switch to application 2
    if index == 2:
        appuifw.app.body = app3 # switch to application 3
```

appuifw: pestañas *(tabs)*

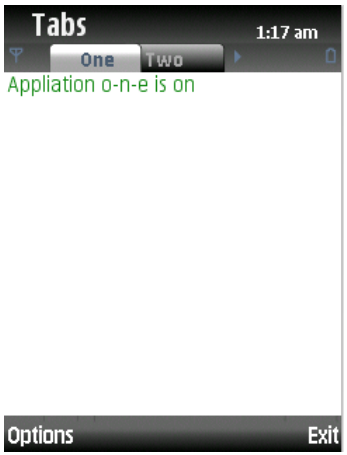
```
# create an Active Object
app_lock = e32.Ao_lock()

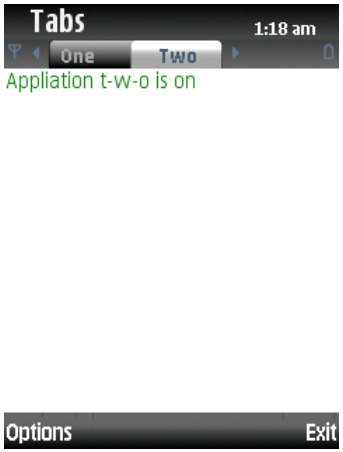
# create the tabs with its names in unicode as a list, include the tab handler
appuifw.app.set_tabs([u"One", u"Two", u"Three"],handle_tab)

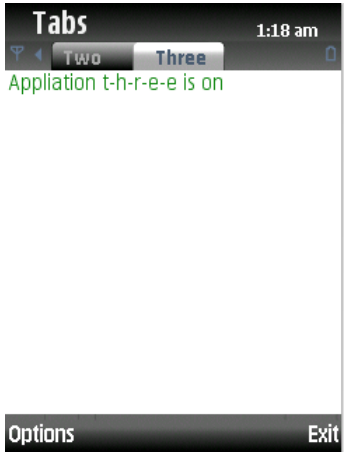
# set the title of the script
appuifw.app.title = u'Tabs'

# set app.body to app1 (for start of script)
appuifw.app.body = app1


appuifw.app.exit_key_handler = exit_key_handler
```







appuifw: diálogos

- Un cuadro de diálogo ofrece al desarrollador la posibilidad de presentar por pantalla un cuadro de texto o campo simple para la introducción de algún tipo de información que requiera la aplicación.
- Hay varios tipos:
  - *query*: pregunta que se presenta al usuario con el objetivo de que este proporcione algún tipo de dato a la aplicación.
  - *note*: nota que muestra por pantalla información al usuario sin necesidad de que este último deba realizar alguna acción o introducir algún dato adicional.
  - *popup_menu*: menú que ofrece al usuario la posibilidad de elegir entre diversas opciones.
  - *selection_list*: permite al usuario elegir un elemento de una lista previamente definida.

appuifw: diálogos *query*

- "text":

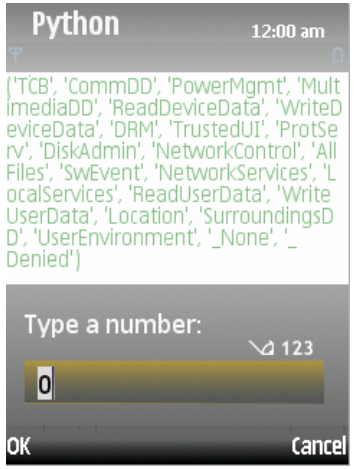
```
import appuifw

data = appuifw.query(u"Type a word:", "text")
print data
```

- "number":

```
import appuifw

data = appuifw.query(u"Type a number:", "number")
print data
```



appuifw: diálogos *query*

- "date":

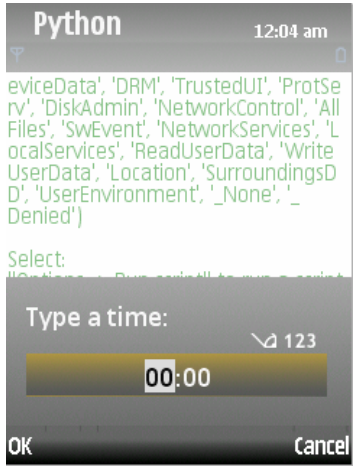
```
import appuifw

data = appuifw.query(u"Type a date:", "date")
print data
```

- "time":

```
import appuifw

data = appuifw.query(u"Type a time:", "time")
print data
```

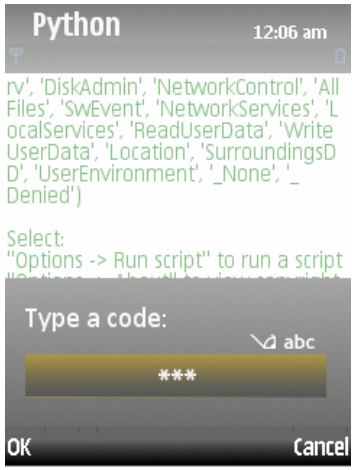


appuifw: diálogos *query*

- "code":

```
import appuifw

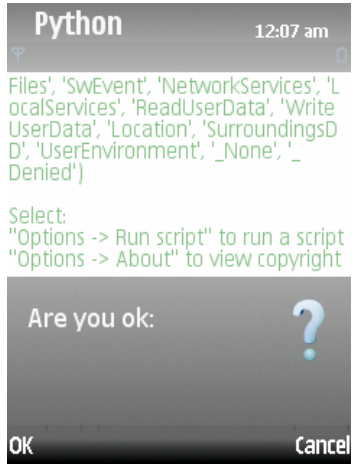
data = appuifw.query(u"Type a code:", "code")
print data
```



- "query":

```
import appuifw

if appuifw.query(u"Are you ok:", "query") == True:
    print "you pressed ok"
else:
    print "you pressed cancel"
```



appuifw: notas

- Sacan una información por pantalla, sin pedir entrada al usuario.

```
import appuifw

# info:
appuifw.note(u"Hello", "info")

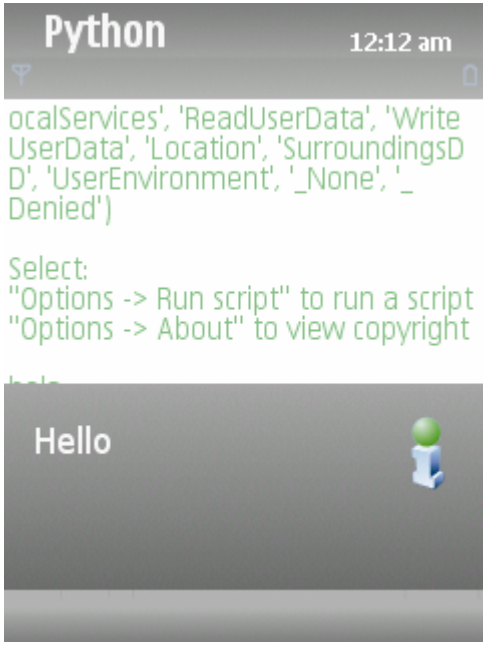
# error:
appuifw.note(u"file not found", "error")

# conf:
appuifw.note(u"upload done", "conf"
```

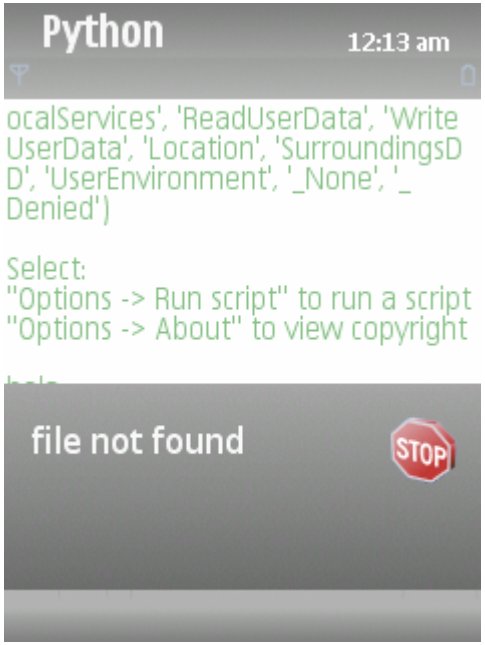
- Las notas aparecen sólo durante unos pocos segundos.

appuifw: notas

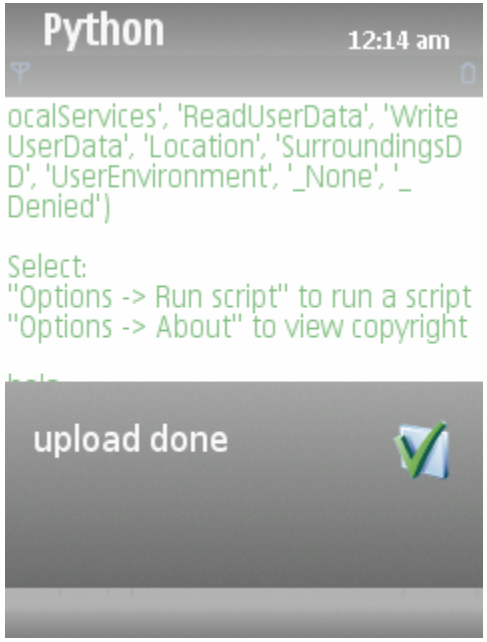
"info"



"error"



"conf"



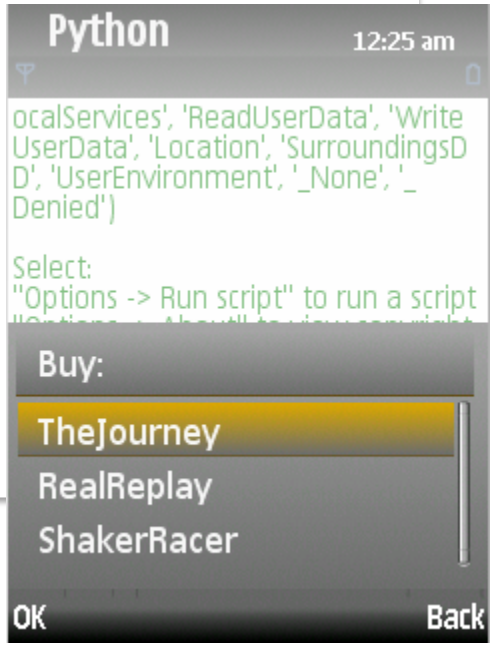
appuifw: *popup_menu*

- Sintaxis: `appuifw.popup_menu(list[, label ])`

```
import appuifw

items = [u"TheJourney", u"RealReplay", u"ShakerRacer"]
index = appuifw.popup_menu(items, u"Buy:")

if index == 0:
    appuifw.note(u"Greatchoice")
elif index == 1:
    appuifw.note(u"Cool")
elif index == 2:
    appuifw.note(u"Ilike that")
elif index == None:
    appuifw.note(u"Purchasecancelled"
```



appuifw: *selection_list*

- Sintaxis: `appuifw.selection_list(choices[, search_field=0])`

```
import appuifw

# define the list of items (items must written in unicode! -> put a u in front)
L = [u'cakewalk', u'com-port', u'computer', u'bluetooth', u'mobile', u'screen',
u'camera', u'keys']

# create the selection list
index = appuifw.selection_list(choices=L , search_field=1)

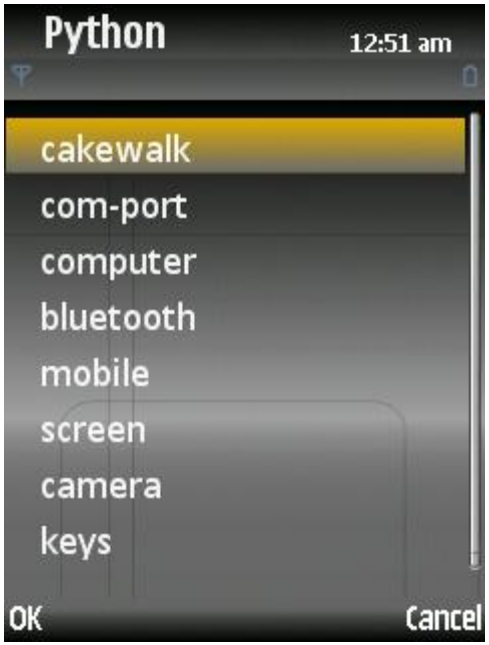
# use the result of the selection to trigger some action
if index == 0:
    print "cakewalk was selected"
else:
    print "thanks for choosing"

# the search_field=1 (set to 1) enables a search functionality for looking
# up items in the list. IMPORTANT: to activate the find pane
# (search functionality) you need to press a keyboard key when the script
# is executed and the list has appeared on the screen.
# if search_field=0 no search functionality is enabled.
```

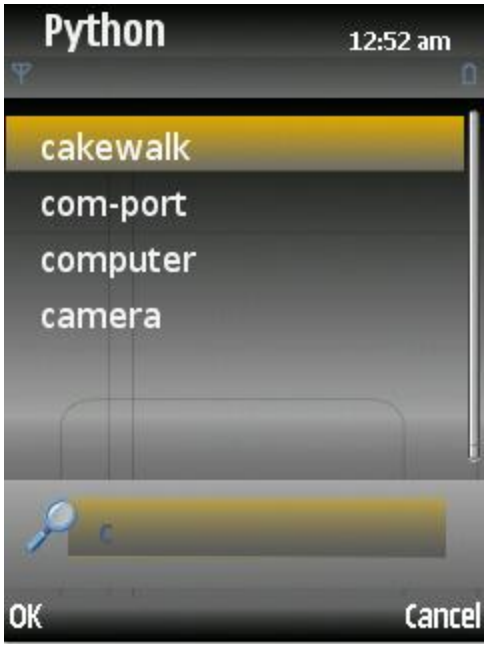
appuifw: *selection_list*

- Si `search_field` se pone a 1, aparece un campo para buscar en la parte de abajo.

`search_field=0`



`search_field=1`



appuifw: *multi_selection_list*

- Sintaxis: `appuifw.multi_selection_list(choices[, style= "checkbox", search_field=0])`

```
import appuifw

names = [u"Michael", u"Devon", u"Bonnie", u"April", u"RC3"]
selections = appuifw.multi_selection_list(names, 'checkbox', 1)
print selection
```

– Con `'checkmark'`:

Con `'checkbox'`:

appuifw: objetos activos

- Los *scripts* que hemos visto hasta ahora se ejecutan línea a línea, y finalizan.
- Muchas veces queremos hacer aplicaciones que se ejecuten hasta que el usuario presione el botón de "*Exit*".
- Esto se hace en Symbian OS con la ayuda de un objeto activo.
  - También se necesitan normalmente cuando se manejan sockets.

appuifw: objetos activos

- Para usar un objeto activo, hay que importar el módulo e32.
  - `import e32`
- Tenemos que crear una instancia del objeto activo.
  - `app_lock = e32.Ao_lock()`
- A continuación se arranca un planificador. Esto significa que el *script* va a porcesar eventos (por ejemplo del UI) hasta que se llame lock.signal() .
  - `app_lock.wait()`
- Se para este planificador llamando a:
  - `app_lock.signal()`

appuifw: *exit_key_handler*

```
import appuifw, e32

def quit():
   print "Right_Softkey_pressed!"
   app_lock.signal()

appuifw.app.title= u"Hola Mundo"
appuifw.app.exit_key_handler= quit
appuifw.note(u"Aplicacion ejecutandose")

app_lock= e32.Ao_lock()
app_lock.wait()
```

- No preocuparse si suena complicado. Podemos copiar y pegar este código en nuestras aplicaciones.

messaging

- Enviar SMS y MMS:

```
import messaging, appuifw

# Enviando SMS
messaging.sms_send('+3412345678', u'Hola!')
appuifw.note(u"SMS enviado", "info“)

# Enviando MMS
messaging.mms_send('+3412345678', u'Hola!',
attachment='e:\\Images\\picture1.jpg')
appuifw.note(u”MMS enviado", "info“)
```

inbox

- Recibir SMS:

```
import inbox, appuifw, e32

def message_received(msg_id):
        box = inbox.Inbox()
        sms_text = box.content(msg_id)
        appuifw.note(u"sms content: " + sms_text , "info")
        app_lock.signal()

box = inbox.Inbox()
box.bind(message_received)

print "Waiting for new SMS messages.."
app_lock = e32.Ao_lock()
app_lock.wait()
print "Message handled!"
```

Asocia una función de *callback* que será llamada cada vez que llegue un mensaje nuevo.

inbox

- Aprovechando el potente manejo de cadenas de Python, podemos hacer un buscador de SMS:

```
Import inbox, appuifw
# Create an instance of the Inbox object
box = inbox.Inbox()
# Query search phrase
query = appuifw.query(u"Searchfor:", "text").lower()
hits = []
ids = []
# sms_messages() returns message IDs for all messages
# in the SMS inbox
For sms_id in box.sms_messages():
  # Retrieve the full message text and convert it to
  # lowercase
  msg_text= box.content(sms_id).lower()
  if msg_text.find(query) != -1:
    # If the text was found, store a preview
    hits.append(msg_text[:25])
    ids.append(sms_id)
# Display all results in a list
index = appuifw.selection_list(hits, 1)
if index >= 0:
  # Show the full text of the selected message
  appuifw.note(box.content(ids[index]))
```

telephone y *contacts*

- Uso de la funcionalidad del teléfono

```
import telephone

telephone.dial('+3412345678')
telephone.hang_up()
```

- Acceso a la agenda

```
import appuifw, telephone, contacts

db = contacts.open()
entry = db.find(appuifw.query(u'Insert a name', 'text'))
L = [ ]
for item in entry :
    L.append(item.title)
if len(L)>0 :
    index = appuifw.selection_list(choices = L, search_field = 0)
    num=entry[index].find('mobile_number')[0].value
    telephone.dial(num)
    appuifw.note(u'... Llamando ...', 'info')
else :
    appuifw.note(u'No le tienes en tu agenda', 'error')

appuifw.app.set_exit()
```

audio: grabar y reproducir sonidos

```
import appuifw, e32, audio

filename = 'e:\\boo.wav'

def recording():
    global S
    S=audio.Sound.open(filename)
    S.record()
    print "Recording on! To end it, select stop from menu!"

def playing():
    global S
    try:
        S=audio.Sound.open(filename)
        S.play()
        print "Playing"
    except:
        print "Record first a sound!"
```

Ya veremos para qué sirve esto.

Ya veremos para qué sirve esto.

audio: grabar y reproducir sonidos

```
def closing():
    global S
    S.stop()
    S.close()
    print "Stopped"

def quit():
    script_lock.signal()
    appuifw.app.set_exit()

appuifw.app.menu = [(u"play", playing),
                    (u"record", recording),
                    (u"stop", closing)]

appuifw.app.title = u"Sound recorder"

appuifw.app.exit_key_handler = quit
script_lock = e32.Ao_lock()
script_lock.wait()
```

audio: *text-to-speach*

```
# import the module called audio
import appuifw
import audio

# trigger a text input-field to let the user type a word

text = appuifw.query(u"Type few words:", "text")

# the phone speaks out the text that you have just typed
# (you can hear it through the loudspeakers of your phone)

audio.say(text)
```

Aplicación que lee SMSs

- A partir de lo que acabamos de ver, es fácil hacer una aplicación que lea SMSs:

```
import inbox
import e32
import audio

#create a function that does the reading of the content of the sms
def read_sms(id):
    # read the content out of the message that has just arrived
    sms_text = i.content(id)
    # the phone speaks out the text that just arrived by sms
    # (you can hear it through the loudspeakers of your phone)
    audio.say(sms_text)

# create an instance of the inbox() class
i=inbox.Inbox()
print "send now sms to this phone"
# put the phone into waiting stage to wait for an incoming message
i.bind(read_sms)
```

graphics y *key_codes*

- Vamos a hacer una aplicación que dibuja un círculo azul sobre fondo amarillo cuando pulsamos la flecha hacia ariba, y lo borra cuando pulsamos hacia abajo.

```
import appuifw, e32, graphics, key_codes

def draw_point():
    img.clear(0xffff00)
    img.point((120,100), 0x0000ff, width = 70)
    c.blit(img)

def not_draw_point():
    img.clear(0xffff00)
    c.blit(img)

def quit():
    app_lock.signal()

appuifw.app.title = u"Dibujando con graphics"
appuifw.app.screen = 'normal'
```

Borra la pantalla y la pinta de color 0xffff00 (amarillo)

Punto azul (centro, color, radio)

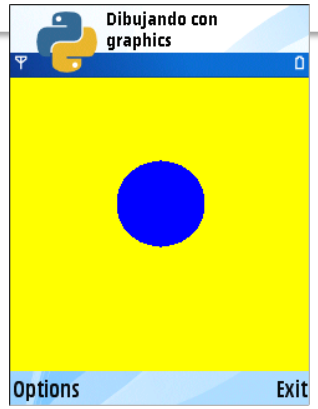
Esto hace que la imagen se dibuje en el *canvas* (la pantalla).

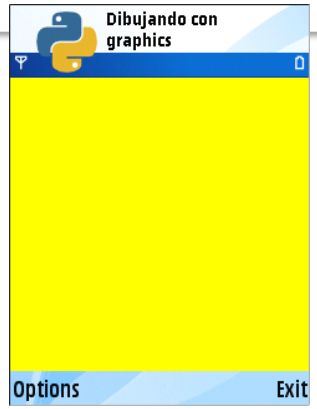
graphics y key_codes

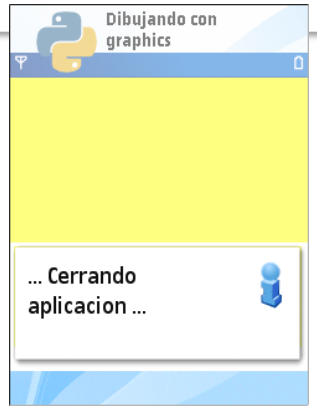
```
c = appuifw.Canvas()
appuifw.app.body = c
s = c.size
img = graphics.Image.new(s)

c.bind(key_codes.EKeyUpArrow, draw_point)
c.bind(key_codes.EKeyDownArrow, not_draw_point)

appuifw.app.exit_key_handler = quit
draw_point()
app_lock = e32.Ao_lock()
app_lock.wait()
appuifw.note(u"... Cerrando aplicacion ...")
```







Manejo de ficheros y directorios

- Listar el contenido de un directorio:

```
import e32, os
# Definimos el directorio
dir ='c:\\images'
# Leemos el directorio
files = map(unicode, os.listdir(dir))
```

- Leer el contenido de una imagen y almacenarlo en una variable (`rb` = modo lectura binario):

```
# Abrimos el fichero que contiene la imagen
f = open('e:\\images\\test.jpg', 'rb')
pic = f.read()
f.close()
```

Manejo de ficheros y directorios

- Escribir texto en un fichero (`w` = modo escritura):

```
f = open('c:\\writetest.txt', 'w')
f.write('hello world')
f.close()
```

- Conversión cadena caracteres / objeto:

```
# Para convertir de string a objeto utilizamos la
  función:
eval()

# Para convertir cualquier objeto en un string
  utilizamos la función:
repr()
```

Manejo de ficheros y directorios

- Ejemplo: cómo guardar unos valores en un fichero de configuración. Usamos un diccionario:

```
import appuifw, e32, os

# Almacenamos los parámetros deseados en dos variables
value1 = 'Python'
value2 = 2.6
config = {}
config['var1'] = value1
config['var2'] = value2

# Abrimos el fichero donde almacenaremos los parámetros de configuración.
f = open(u'e:\\mysettings.txt', 'wt')

# Escribimos en el fichero los parámetros, convirtiéndolos a string
previamente.
f.write(repr(config))

# Cerramos el fichero
f.close()
```

Manejo de ficheros y directorios

```
# Volvemos a abrir el fichero y comprobamos que los
parámetros
# se guardaron correctamente. Para ello realizamos el
# procedimiento inverso.
f = open(u'e:\\mysettings.txt', 'wt')
f.read()
config = eval(content)
f.close()

value1 = config.get('var1', '')
value2 = config.get('var2', '')

# Por último imprimimos por pantalla para comprobar que
# efectivamente los parámetros se guardaron correctamente.
print value1, value2
```

camera

- Acceso a la cámara del móvil para tomar fotografías o grabar vídeos.
- Funciones de las que dispone este módulo:
  - `cameras_available()`
    - Devuelve el número de cámaras disponibles en el dispositivo.
  - `image_modes()`
    - Devuelve una cadena de caracteres con los modos de imagen soportados, como por ejemplo: 'RGB' o 'JPEG_Exif'.
  - `image_sizes()`
    - Devuelve la resolución en una tupla del estilo: (640, 480), (160, 120), etc.
  - `flash_modes()`
    - Devuelve una lista de *strings* con los modos de flash disponibles.
  - `max_zoom()`
    - Devuelve un entero con el máximo zoom digital soportado por el dispositivo.

camera

- `exposure_modes()`
  - Devuelve una lista de *strings* con los ajustes de exposición.
- `white_balance_modes()`
  - Devuelve una lista de *strings* con los ajustes de balanceo de blanco disponibles.
- `take_photo([mode, size, zoom, flash, exposure, white_balance, position])`
  - Toma una fotografía y devuelve la imagen
- `start_record(filename, callable)`
  - Comienza la grabación de video en el dispositivo.
- `stop_record()`
  - Finaliza la reproducción de video.

camera: ejemplo de captura de foto

```
import e32, camera, appuifw, key_codes

def finder_cb(im):
    canvas.blit(im)

def take_picture():
    camera.stop_finder()
    pic = camera.take_photo(size = (640,480))
    w,h = canvas.size
    canvas.blit(pic,target=(0, 0, w, 0.75 * w), scale = 1)
    pic.save('e:\\Images\\picture1.jpg')

def quit():
    app_lock.signal()
canvas = appuifw.Canvas()
appuifw.app.body = canvas

camera.start_finder(finder_cb)
canvas.bind(key_codes.EKeySelect, take_picture)

appuifw.app.title = u"My Camera"
appuifw.app.exit_key_handler = quit
app_lock = e32.Ao_lock()
app_lock.wait()
```

sysinfo

- Permite acceder a información del sistema:

```
import sysinfo
Sysinfo.battery()
```

- Podemos ver:
  - `active_profile()`
    - Devuelve una cadena de caracteres con el último perfil que se activó en el terminal: *'general', 'silent', 'meeting', 'outdoor', 'pager', 'offline', 'drive'* o *'<user profile value>'.*
  - `battery()`
    - Muestra el nivel actual de batería del terminal devolviendo un número comprendido entre el 0 y el 7. Si se está usando un emulador, el valor devuelto es siempre 0.
  - `display_pixels()`
    - Devuelve la anchura y altura de la pantalla en pixels.
  - `display_twips()`
    - Devuelve la anchura y altura de la pantalla en twips, los cuales son independientes de las unidades de la pantalla, con el fin de asegurar que la proporción de pantalla que utilizan los diversos elementos para ser mostrados será la misma en cualquier dispositivo. Por ejemplo, un twip se define como 1/567 para un centímetro o 1/1440 para una pulgada.

sysinfo

- `free_drivespace()`
  - Devuelve la cantidad de bytes de espacio libre disponible en disco.
- `free_ram()`
  - Devuelve la cantidad de memoria RAM disponible en el dispositivo.
- `imei()`
  - Devuelve unacadena de caracteres codificada en Unicode con el código IMEI del dispositivo. Si estamos utilizando un emulador se devolverá la cadena u'000000000000000'.
- `max_ramdrive_size()`
  - Devuelve el tamaño máximo de memoria RAM del dispositivo.
- `os_version()`
  - Devuelve un entero con el número de versión del sistema operativo del dispositivo.
- `ring_type()`
  - Devuelve un *string* o cadena de caracteres con el tipo de tono actual, el cual puede ser uno de los siguientes: *'normal', 'ascending', 'ring_once', 'beep'* o *'silent'.*

sysinfo

- `signal_bars()`
  - Devuelve el nivel actual de señal mediante un número comprendido entre 0 y 7 (0 significa que no hay señal y 7 indica que tenemos una fuerte señal). Si estamos usando un emulador se devolverá siempre el valor 0.
- `signal_dbm()`
  - Devuelve el nivel de señal actual en dBm. Si estamos usando un emulador se devolverá siempre el valor 0.
- `sw_version()`
  - Devuelve una cadena de caracteres codificada en Unicode con la versión de software. Si estamos usando un emulador se devolverá la cadena u'emulator'
- `total_ram()`
  - Devuelve la cantidad total de memoria RAM del dispositivo.
- `total_rom()`
  - Devuelve la cantidad total de memoria ROM (solo lectura) del dispositivo.

urllib y *appuifw.Content_handler()*

- Descarga de Internet (con *urlib*) y reproducción (usando *appuifw.Content_handler())* de un vídeo:

```
import appuifw, urllib

# define the url where the video file is located on the server
url = "http://www.leninsgodson.com/courses/pys60/resources/vid001.3gp"

# define the loction on the phone where the fetched video file shall be
# stored
tempfile = "e:\\video01.3gp"

# fetch down the video and store it to you hard drive
urllib.urlretrieve(url, tempfile)

# a content handler handles the playing of the video
content_handler = appuifw.Content_handler()

# open the video via the content handler. It will start playing
# automatically
content_handler.open(tempfile)
```

urllib y *appuifw.Content_handler()*

- Descarga y reproducción de un vídeo (un poco mejor):
  - Tras terminar la reproducción, vuelve a nuestra aplicación.

```
import appuifw, e32, urllib

print "press options"

def fetching():
    url = "http://www.leninsgodson.com/courses/pys60/resources/vid001.3gp"
    tempfile = "e:\\video01.3gp"
    try:
        print "Retrieving information..."
        urllib.urlretrieve(url, tempfile)
        lock=e32.Ao_lock()
        content_handler = appuifw.Content_handler(lock.signal)
        content_handler.open(tempfile)
        # Wait for the user to exit the image viewer.
        lock.wait()
        print "Video viewing finished."
    except:
        print "Problems."
```

urllib y appuifw.Content_handler()

```
def quit():
    app_lock.signal()

appuifw.app.menu = [(u"get video", fetching)]

appuifw.app.title = u"Get video"

appuifw.app.exit_key_handler = quit
app_lock = e32.Ao_lock()
app_lock.wait()
```

Arrancar cualquier aplicación del teléfono

```
import appuifw
import e32

def camera():
    try:
        #start_exe(filename, command [,wait ])
        # this starts the normal camera app on your phone
        e32.start_exe('z:\\system\\programs\\apprun.exe',
'z:\\system\\apps\\camcorder\\camcorder.app')
    except:
        print "something wrong"

def exit_key_handler():
    script_lock.signal()
    appuifw.app.set_exit()

script_lock = e32.Ao_lock()

appuifw.app.title = u"test"

appuifw.app.menu = [(u"camera", camera)]

appuifw.app.exit_key_handler = exit_key_handler
script_lock.wait()
```

httplib

- httplib da un interfaz de más bajo nivel que urllib.
- Ejemplo: enviar un fichero a un servidor (con HTTP POST):

```
import appuifw,e32,httplib

def upload_image_to_url():

    filename = 'e:\\Images\\picture1.jpg'
    picture = file(filename).read()

    conn = httplib.HTTPConnection("www.mobilenin.com")
    conn.request("POST", "/pys60/php/upload_image_to_url.php",
picture)
    print "upload started ..."
    e32.ao_yield()
    response = conn.getresponse()
    remote_file = response.read()
    conn.close()
    appuifw.note(u" " + remote_file, "info")
    print remote_file

upload_image_to_url()
```

socket

- Comunicaciones a más bajo nivel.
- API standard de Python.
- Cliente:

```
import socket

HOST = '217.30.180.11'    # The remote host
PORT = 12008              # The same port as used by the server
print "define socket"
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print "trying to connect to socket"
s.connect((HOST, PORT))
print "connected"
s.send('Hello, world')
print "data send"
data = s.recv(1024)
s.close()
print 'Received', `data`
```

socket

- Servidor:

```
import socket

HOST = ''                 # Symbolic name meaning the local host
PORT = 12008              # Arbitrary non-privileged port
print "define the socket"
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print "bind the socket"
s.bind((HOST, PORT))
s.listen(1)
print "waiting of the client to connect"
conn, addr = s.accept()
print 'Connected by', addr
while 1:
    data = conn.recv(1024)
    if not data: break
    conn.send(data)
conn.close()
```

sensor: Acelerómetro

```
import sensor, appuifw

accelerometer =
sensor.AccelerometerXYZAxisData(data_filter=sensor.LowPassFilter())
counter = 0

def my_callback():
    global counter
    if counter == 0:
        print "X: ", accelerometer.x, "  Y: ", accelerometer.y, "
Z: ", accelerometer.z
    counter = counter + 1
    if counter == 5:
        counter = 0

def quit():
    accelerometer.stop_listening()
    print "Exiting Accelorometer"

appuifw.app.exit_key_handler = quit
accelerometer.set_callback(data_callback=my_callback)
accelerometer.start_listening()
```

sensor: Rotación

```
import sensor, appuifw, audio, e32

sensor_type = sensor.sensors()['RotSensor']
N95_sensor = sensor.Sensor(sensor_type['id'],sensor_type['category'])
N95_sensor.set_event_filter(sensor.RotEventFilter())

def get_sensor_data(status):
    if status == sensor.orientation.TOP :
        appuifw.note(u"BOTTOM", "info")
    elif status == sensor.orientation.BOTTOM :
        appuifw.note(u"TOP", "info")
    elif status == sensor.orientation.LEFT :
        appuifw.note(u"RIGHT", "info")
    elif status == sensor.orientation.RIGHT :
        appuifw.note(u"LEFT", "info")

def quit():
    N95_sensor.disconnect()
    app_lock.signal()

N95_sensor.connect(get_sensor_data)
print ' Turn your phone right or left or up or down!'

appuifw.app.title = u"N95 Rotation sensor"
appuifw.app.exit_key_handler = quit
app_lock = e32.Ao_lock()
app_lock.wait()
```

positioning: GPS

```
# note: your phone must have the 'Location' (GPS data /position) application on,
# and receive satalite data in order to make this script work. (can be
# problematic indoors).

import positioning

positioning.select_module(positioning.default_module())
positioning.set_requestors([{"type":"service",
                             "format":"application",
                             "data":"test_app"}])

def getmyposition():
    result = positioning.position()
    #print 'all gps data: ', result
    coordinates=result["position"]
    mylatitude = coordinates["latitude"]
    mylongitude = coordinates["longitude"]
    print 'mylatitude: ', mylatitude
    print 'mylongitude:', mylongitude

getmyposition()
```

location: Localización por celda

```
import appuifw
import e32
import location

def main_menu_setup():
    appuifw.app.menu = [(u"get location", gsm_location)]

def gsm_location() :
    (mcc, mnc, lac, cellid) = location.gsm_location()
    print u"MCC: " + unicode(mcc)
    print u"MNC: " + unicode(mnc)
    print u"LAC: " + unicode(lac)
    print u"Cell id: " + unicode(cellid)

def exit_key_handler():
    global script_lock
    script_lock.signal()
    appuifw.app.set_exit()

script_lock = e32.Ao_lock()

appuifw.app.title = u"Get cell-id"

#appuifw.app.body = appuifw.Text(u"Press Options button below ...")

main_menu_setup()
appuifw.app.exit_key_handler = exit_key_handler
script_lock.wait()
```

location: Localización por celda

- Usando el API de google podemos pasar a coordenadas:
  - http://code.google.com/p/birdnest/source/browse/branches/gae/birdnest/glm.py?spec=svn82&r=82
- Y con *el PyS60 Google Maps API* podemos representar el punto en un mapa de Google Maps:
  - http://wiki.forum.nokia.com/index.php/PyS60_Google_Maps_API

Interfaces táctiles

```
import appuifw, e32, key_codes

def one(dummy):
    appuifw.note(u"This works")

def two(dummy):
    appuifw.note(u"I love Pittsburgh")

def quit(dummy):
    global lock
    lock.signal()

canvas=appuifw.Canvas()
appuifw.app.screen = 'full'
appuifw.app.body=canvas

canvas.bind(key_codes.EButton1Down, one, ((50,50),(150,150)))
canvas.bind(key_codes.EButton1Down, two, ((200,200),(300,300)))
canvas.bind(key_codes.EButton1Down, quit, ((350,50),(450,150)))

canvas.rectangle(((50,50),(150,150)), fill = (255,0,0))
canvas.rectangle(((200,200),(300,300)), fill = (0,255,0))
canvas.rectangle(((350,50),(450,150)), fill = (0,0,255))

lock = e32.Ao_lock()
appuifw.app.exit_key_handler=quit
lock.wait()
```

Depuración con PyS60

- Diferentes maneras de depurar un programa Python.
  - La más sencilla es utilizar la función `print` para ir comprobando los valores de las diversas variables de nuestro código.
  - Otra posibilidad es importar el módulo `traceback` y utilizar la función `traceback.print_exc()`, que captura las excepciones lanzadas por nuestro programa y las almacena en un fichero.
  - También podemos declarar bloques de código con `try:` y `except:`
    - Sirven para capturar excepciones.
    - El bloque de código acotado por `try:` intentará ser ejecutado por el intérprete Python, pero si sucede algún error en tiempo de ejecución, se capturará la excepción el bloque de código acotado por `except:` será ejecutado.
    - El bloque de código acotado por `except:` suele ser aprovechado para incluir información útil para depurar el código posteriormente.

Ejemplos *try* y *except*

```
def playing():
    global S
    try:
        S=audio.Sound.open(filename)
        S.play()
        print "Playing"
    except:
        print "Record first a sound!"
```

```
def camera():
    try:
        #start_exe(filename, command [,wait ])
        # this starts the normal camera app on your phone
        e32.start_exe('z:\\system\\programs\\apprun.exe',
'z:\\system\\apps\\camcorder\\camcorder.app')
    except:
        print "something wrong"
```

Crear aplicaciones PyS60 autocontenidas (*stand-alone*)

- Se puede generar a partir de un script PyS60 una aplicación Symbian (`.sis`) que se puede lanzar sin necesitar tener instalado el intérprete.
- Los pasos son los siguientes (cambian según versión Symbian OS):
  1. Descargar e instalar S60 SDK para Symbian. Asegurarse de que las utilidades `makesis` y `uidcrc` estén correctamente configuradas en la variable PATH del sistema.
  2. Instalar el plug-in de Python, que viene en el paquete SDK de Python para S60.
  3. Abrir una ventana de sistema
  4. Utilizar el comando:
     `py2sis <src> [sisfile] [--uid=0x1234567]`
     - `<src>` es la ruta donde se encuentra el script Python que deseamos empaquetar.
     - `[sisfile]` es el nombre de l nuevo fichero *.sis que será generado.
     - `[--uid=0x1234567]`: es un UID que puede elegirse en un rango entre 0x01000000 y 0x0fffffff.
- Por ejemplo, podemos ejecutar:
  - `py2sis mi_script_python.py mi primer_programa.sis --uid=0x0FFFFFF`

Referencias

- Python:
  - "The Python Tutorial", http://docs.python.org/tutorial/
  - "Python para todos", Raúl González Duque, http://mundogeek.net/tutorial-python/
- PyS60:
  - "Python for Symbian Phones", de Jürgen Scheible. Capítulo 2 del libro: "Mobile Phone Programming and its Application to Wireless Networking". Fitzek, Frank H. P. and Reichert, Frank (Editors). Springer 2007. http://books.google.com/books?id=s_OnKP3VAQ4C&lpg=PA23&dq=python%20for%20symbian%20phones&pg=PA23#v=onepage&q=python%20for%20symbian%20phones&f=false
  - "Mobile Python: Rapid Prototyping of Applications on the Mobile Platform". Scheible, Jürgen and Tuulos, Ville. John Wiley & Sons. 2007.
  - "Symbian OS: Python/PyS60", Andreas Jakl, Marzo 2009, http://symbianresources.com/tutorials/general.php
  - Garage Python for S60: https://garage.maemo.org/projects/pys60/
  - Mobilenin: http://www.mobilenin.com/pys60/menu.htm