

# SENG 462 Tutorial #6

Chris Pearson  
pearson@csc.uvic.ca



University  
of Victoria

# Logbooks

- ▶ When student001's log book entry is validated:
  - Student001 only sees how many group members have validated that entry
  - Student001 does NOT see if the group members agree or disagree with that entry
  - Student001 does NOT see any comments left by the group members
- ▶ Only the class instructors see the comments and the agree/disagree counts.



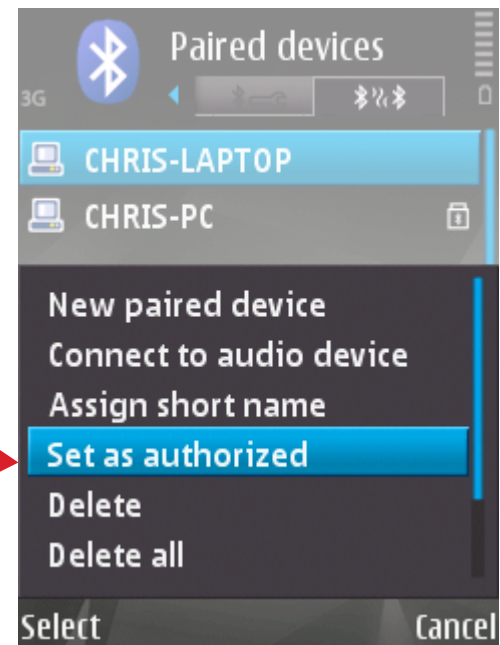
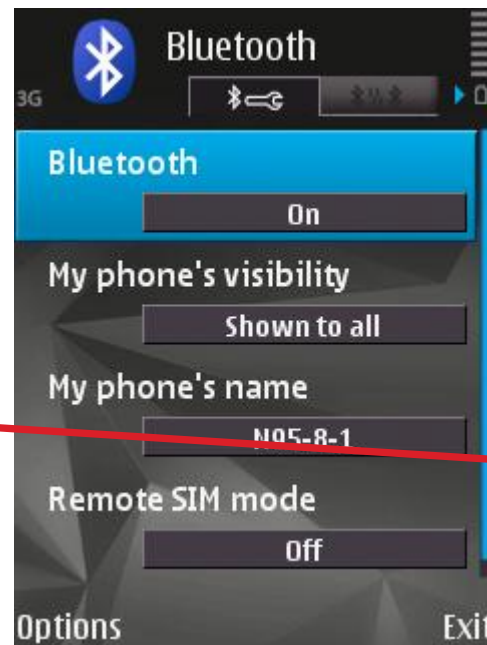
# Testing

- ▶ Once your 2 user workload file is complete, try running all the other workload files through.
  - If you've done it right, they should ALL work but very very slowly



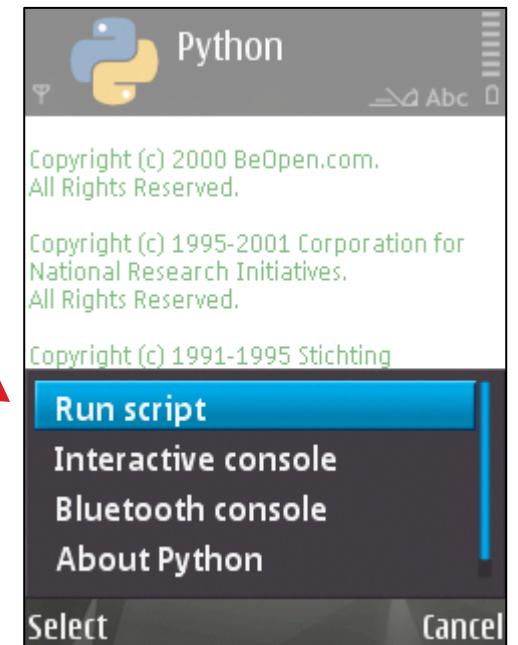
# Connecting to the N97

- ▶ Bluetooth menu on N97:
  - Phone must be set as visible
- ▶ Bluetooth menu on computer:
  - Add Bluetooth device
  - Search for N97
  - Pair the devices
- ▶ Bluetooth menu on N97:
  - Set computer as “authorized”

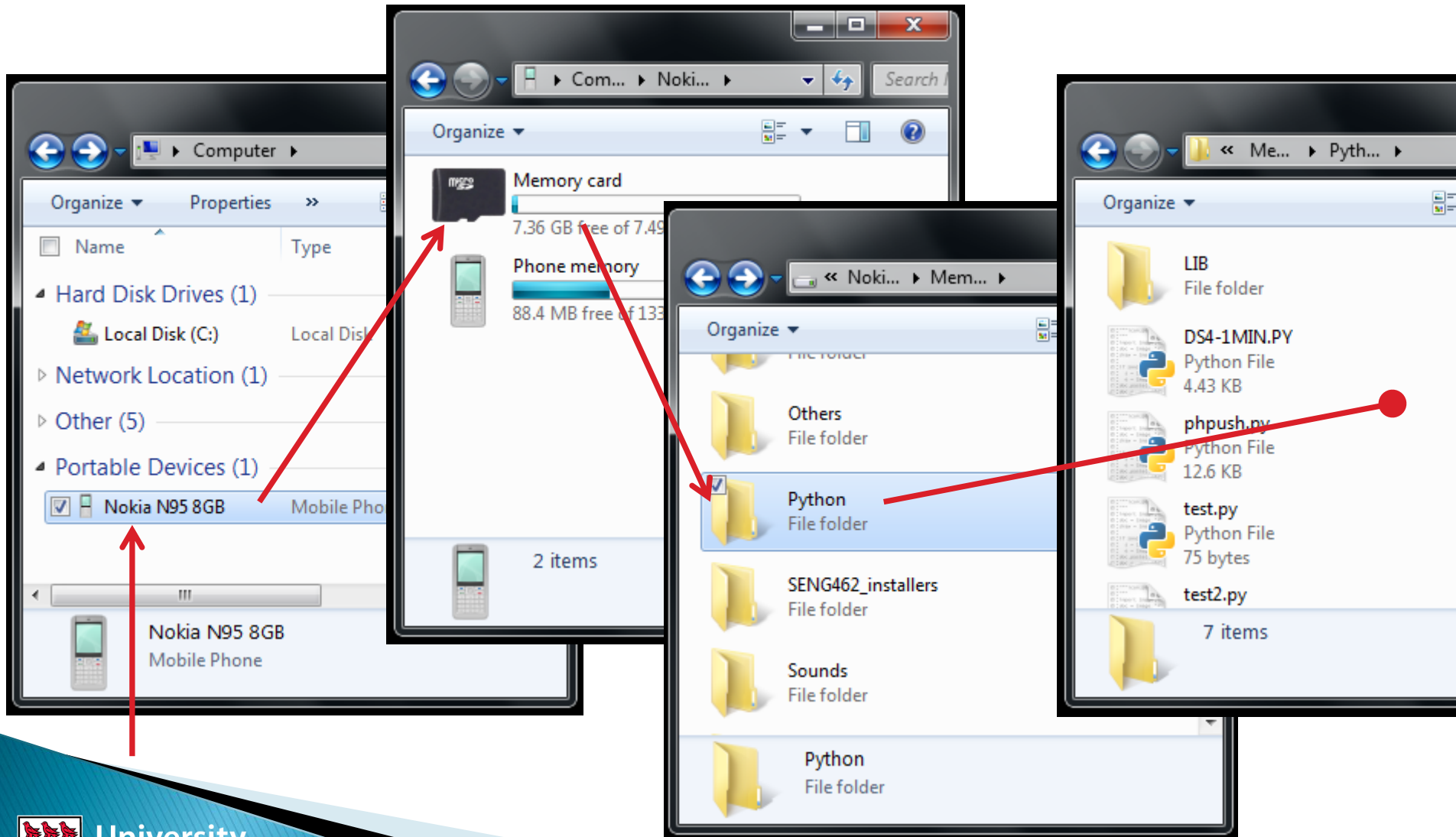


# Drag and Drop Development

- ▶ Quickest for development:
  - Bluetooth to PC/Mac/Linux
  - Edit Python scripts on your computer
  - Drag and drop files into the N97 E:\Python directory
  - Run script from the PyS60 Script Shell



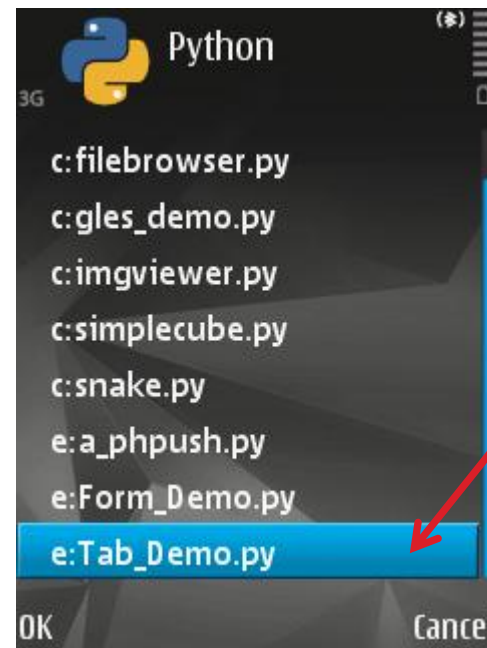
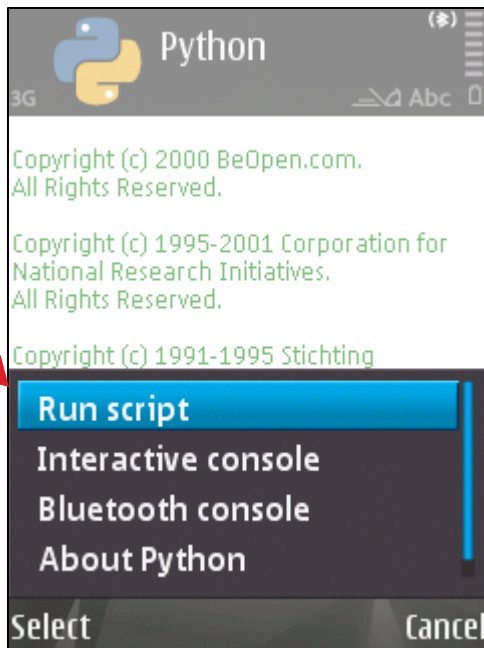
# Drag and Drop Development (2)





# Drag and Drop Development (2)

- ▶ Run your script!



# Bluetooth Console

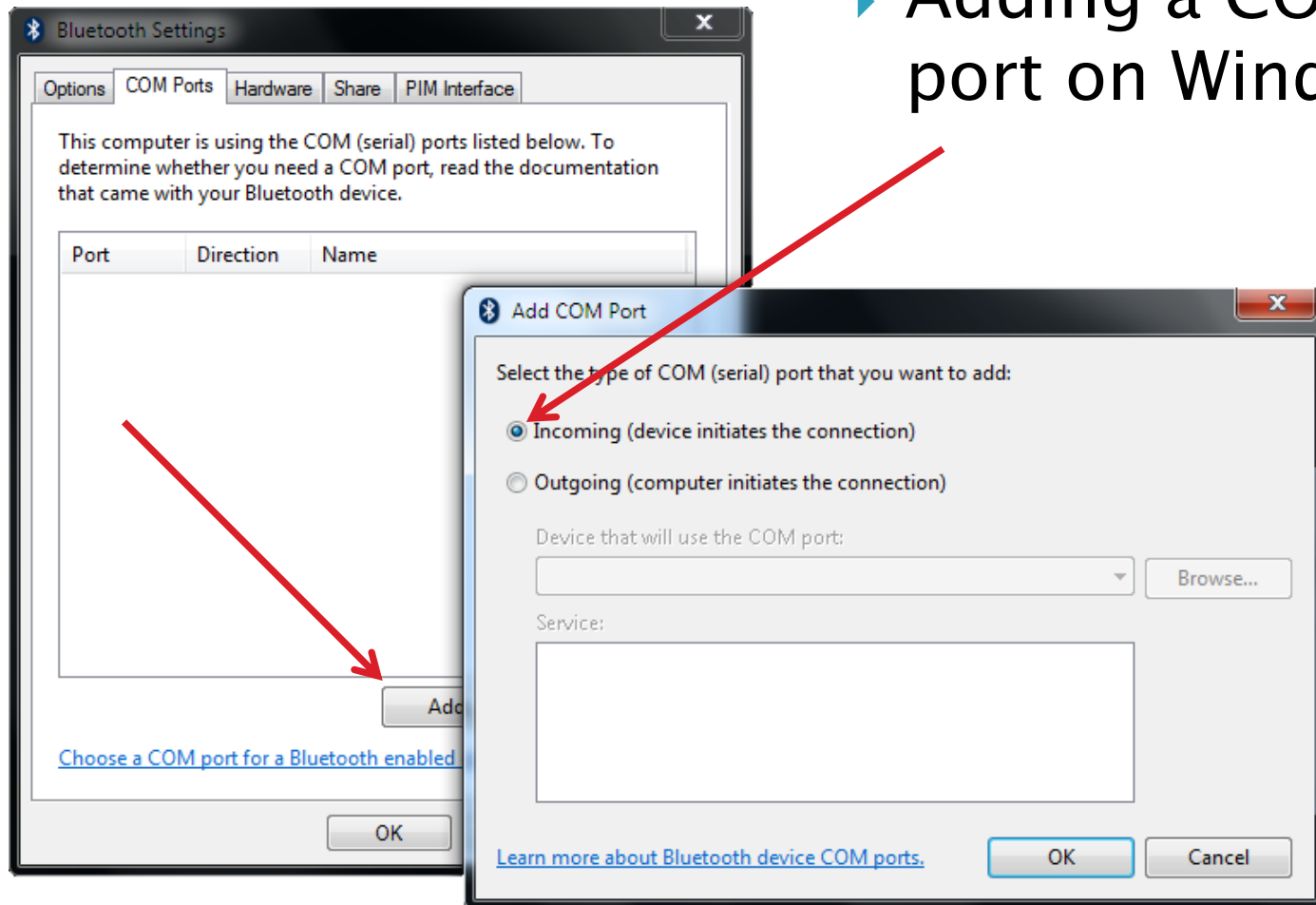
- ▶ Simple for testing: Bluetooth console
  - A console for typing Python commands
  - Connect via terminal program (ie: PuTTY)
  - Requires an outgoing serial connection on the computer (one time setup)





# Bluetooth Console

- ▶ Adding a COM port on Windows



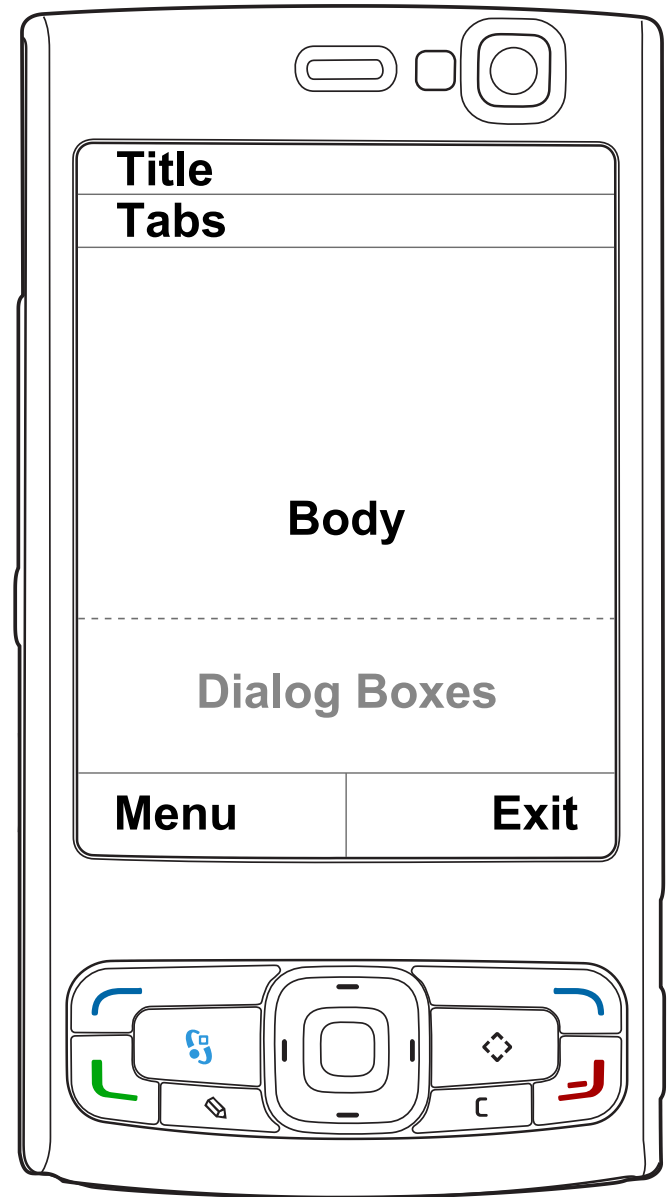
# Bluetooth Console

- ▶ On computer:
  - Start PuTTY
  - Connect to the port number you assigned
- ▶ On N97:
  - Start Python
  - Choose Bluetooth Console
  - Choose computer
- ▶ Test on computer:
  - Type: `import audio`
  - Type: `audio.say("Hello World")`



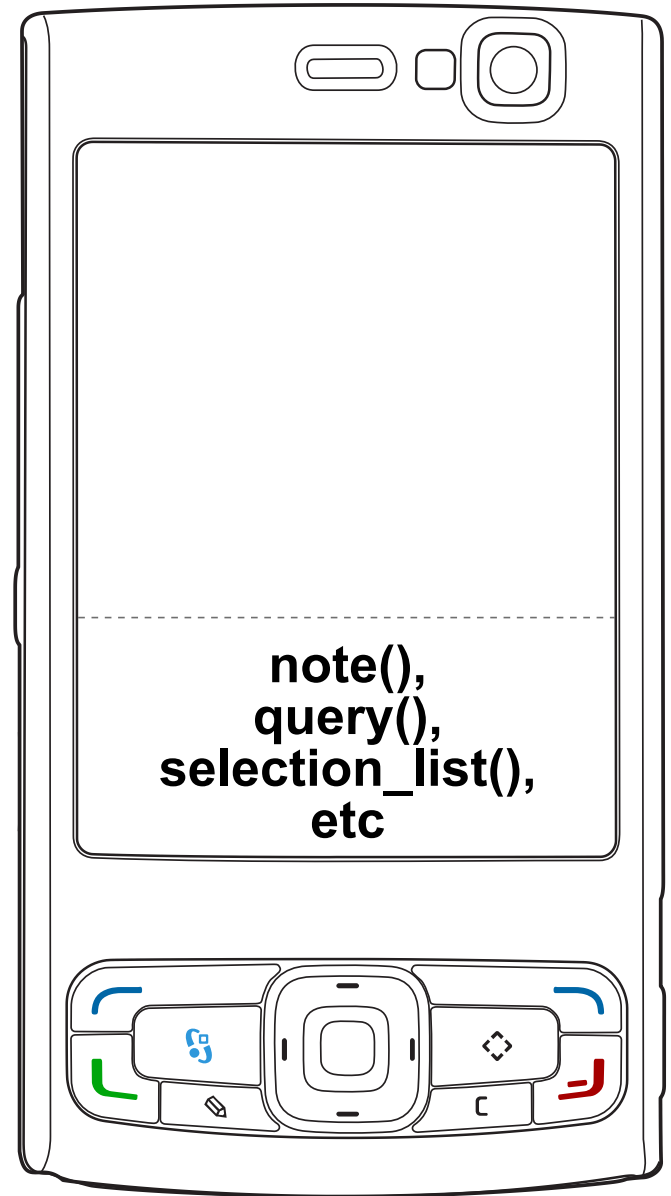
# The Interface

- ▶ The typical Symbian OS interface:
  - Title bar
  - Body
  - Soft keys
- ▶ Other:
  - Tabs overlay the title bar
  - Dialog boxes cover the bottom of the body



# Dialog Boxes

- ▶ **note()**
  - Displays a simple information box
- ▶ **query()**
  - Displays different input requests
- ▶ **Popup Menus**
- ▶ **Selection Lists**



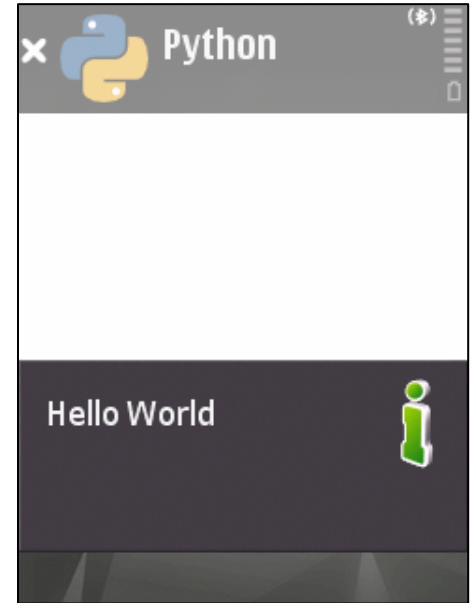
# Dialog Boxes – Notes

```
from appuifw import *
```

```
# Display a note dialog
```

```
note(u"Hello World")
```

- ▶ Only two lines of code are needed
- ▶ Notice the use of Unicode strings...



A note() dialog box

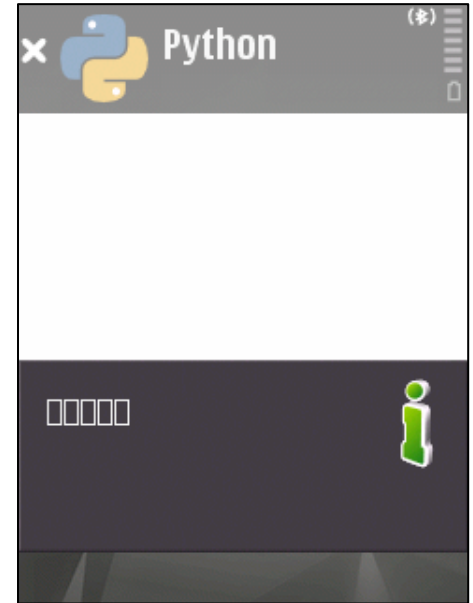


# Dialog Boxes – Notes

```
from appuifw import *
```

```
# Display a note dialog  
note("Hello World")
```

- ▶ Without Unicode strings, things just don't look right...



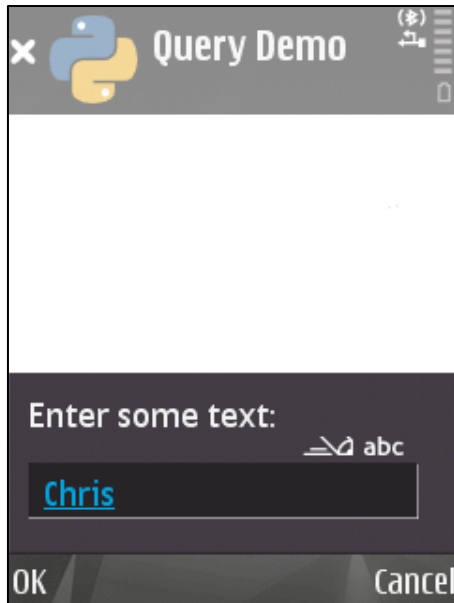
Incorrect Text Encoding





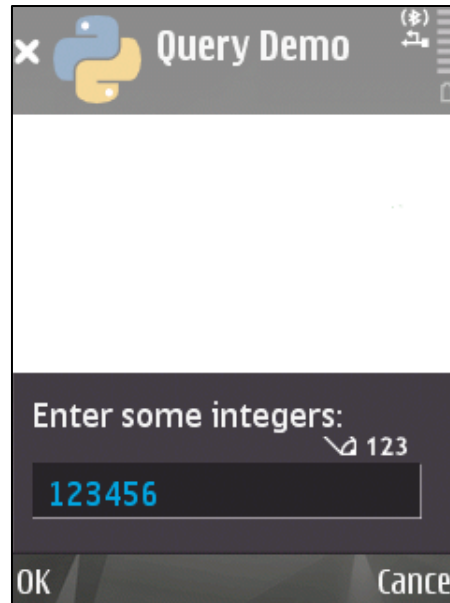
# Dialog Boxes – Queries

- ▶ There are seven types of pop-up queries



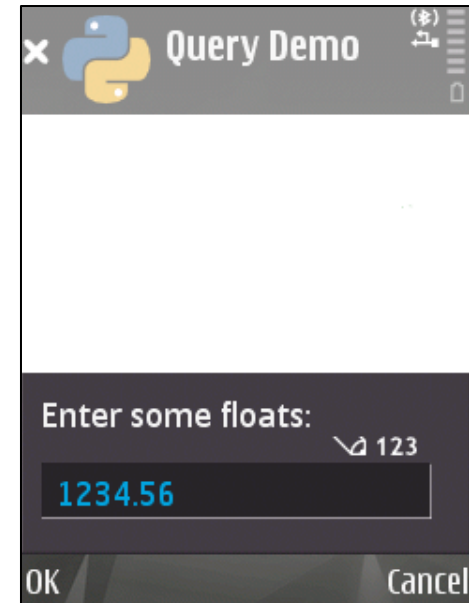
A screenshot of a 'Query Demo' dialog box. The title bar shows a close button, a Python logo, and the text 'Query Demo'. The main area is a large white rectangle. Below it, the text 'Enter some text:' is displayed. To the right of this text is a small icon of a document with a pencil and the text 'abc'. Below the text is a text input field containing the word 'Chris'. At the bottom are 'OK' and 'Cancel' buttons.

text



A screenshot of a 'Query Demo' dialog box. The title bar shows a close button, a Python logo, and the text 'Query Demo'. The main area is a large white rectangle. Below it, the text 'Enter some integers:' is displayed. To the right of this text is a small icon of a document with a pencil and the text '123'. Below the text is a text input field containing the number '123456'. At the bottom are 'OK' and 'Cancel' buttons.

number



A screenshot of a 'Query Demo' dialog box. The title bar shows a close button, a Python logo, and the text 'Query Demo'. The main area is a large white rectangle. Below it, the text 'Enter some floats:' is displayed. To the right of this text is a small icon of a document with a pencil and the text '123'. Below the text is a text input field containing the number '1234.56'. At the bottom are 'OK' and 'Cancel' buttons.

float

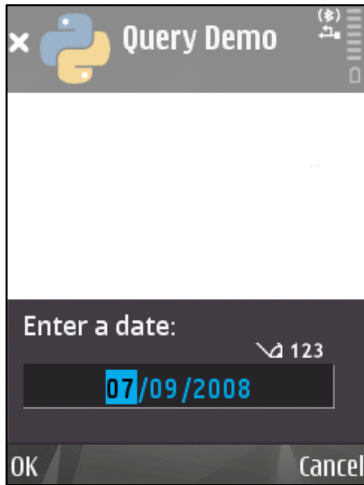
```
query(u"Enter some text:", "text")
```

- ▶ The first field is the message to display
- ▶ The second field is the type of query

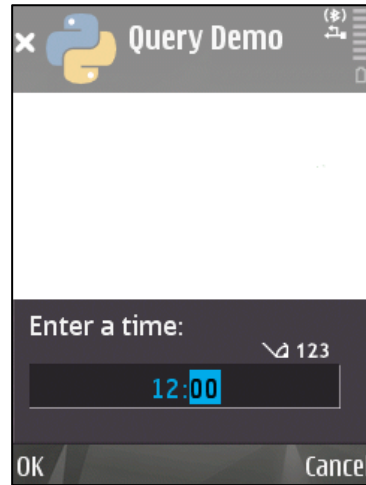


# Dialog Boxes – Queries (2)

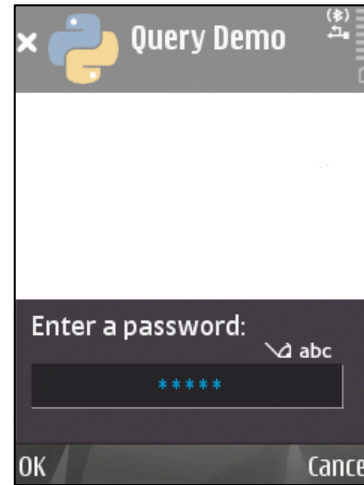
- ▶ There are seven types of pop-up queries

A screenshot of a 'Query Demo' dialog box. The title bar has a close button (X), a Python logo, and the text 'Query Demo'. The main area is empty. At the bottom, it says 'Enter a date:' followed by a small icon and '123'. Below this is a text input field containing '07/09/2008'. At the very bottom are 'OK' and 'Cancel' buttons.

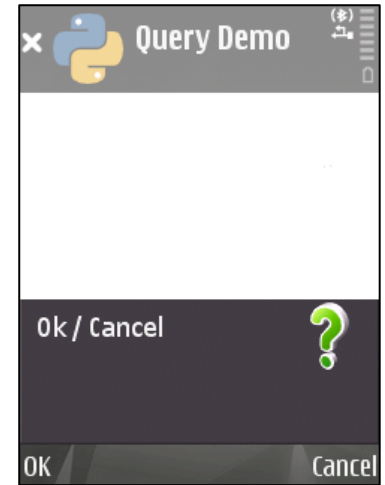
date

A screenshot of a 'Query Demo' dialog box. The title bar has a close button (X), a Python logo, and the text 'Query Demo'. The main area is empty. At the bottom, it says 'Enter a time:' followed by a small icon and '123'. Below this is a text input field containing '12:00'. At the very bottom are 'OK' and 'Cancel' buttons.

time

A screenshot of a 'Query Demo' dialog box. The title bar has a close button (X), a Python logo, and the text 'Query Demo'. The main area is empty. At the bottom, it says 'Enter a password:' followed by a small icon and 'abc'. Below this is a text input field containing '\*\*\*\*\*'. At the very bottom are 'OK' and 'Cancel' buttons.

password

A screenshot of a 'Query Demo' dialog box. The title bar has a close button (X), a Python logo, and the text 'Query Demo'. The main area is empty. At the bottom, it says 'Ok / Cancel' followed by a green question mark icon. At the very bottom are 'OK' and 'Cancel' buttons.

query

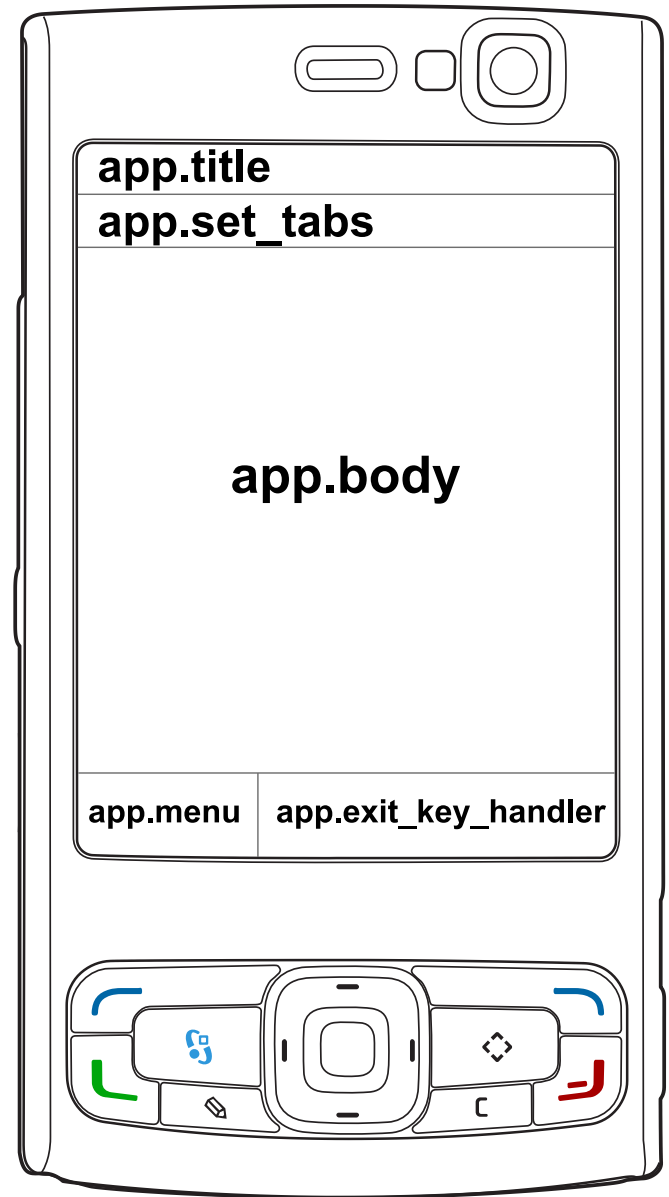
```
query(u"Enter some text:", "text")
```

- ▶ The first field is the message to display
- ▶ The second field is the type of query



# The Interface

- ▶ The interface elements are accessed through the app object



# Primary UI Controls

- ▶ Text
- ▶ Listbox
- ▶ Canvas
  
- ▶ Form
  - Forms have more in common with dialog boxes



# A Simple Application

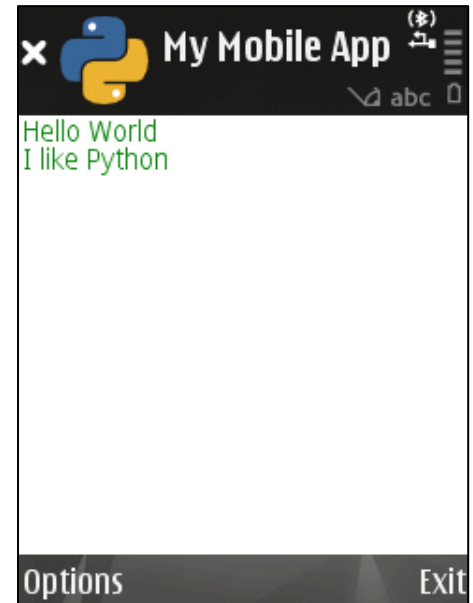
- ▶ An app.body displaying text

```
from appuifw import *

# Define the type of window to use
textArea = Text()
app.body = textArea

# Add some text to the area
textArea.add(u"Hello World\nI like Python")

# Display the app's name
app.title = u"My Mobile App"
```



# A Simple Application

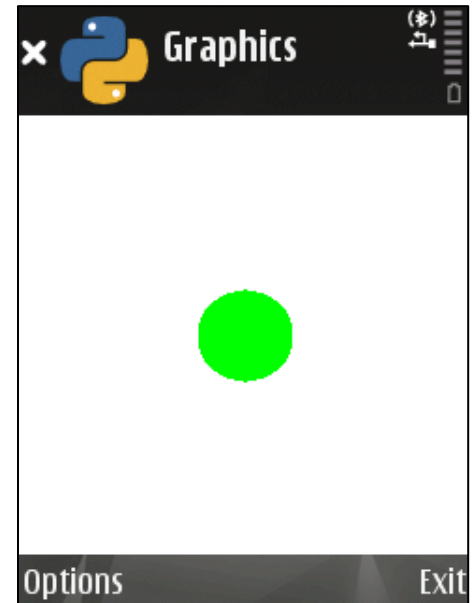
- ▶ This time displaying graphics

```
import graphics
from appuifw import *

# Define the type of window to use
canvas=Canvas()
app.body = canvas
w,h = canvas.size

# Add some text to the area
canvas.clear((255,255,255))
canvas.point((w/2,h/2),(0,255,0), width=50)

# Display the app's name
app.title = u"Graphics"
```





# Menus

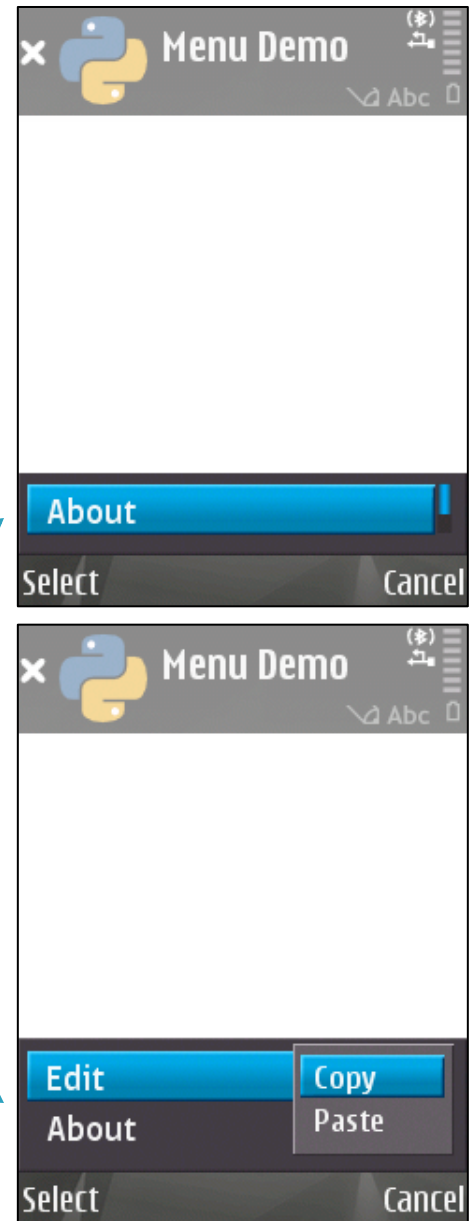
- ▶ Menus are created from tuples stored in lists
- ▶ Lists can be nested for multi-level menus

- ▶ A simple menu:

```
app.menu = [(u"About", about)]
```

- ▶ A multi-level menu:

```
app.menu = [(u"Edit", ((u"Copy", copy),  
                        (u"Paste", paste))), (u"About", about)]
```



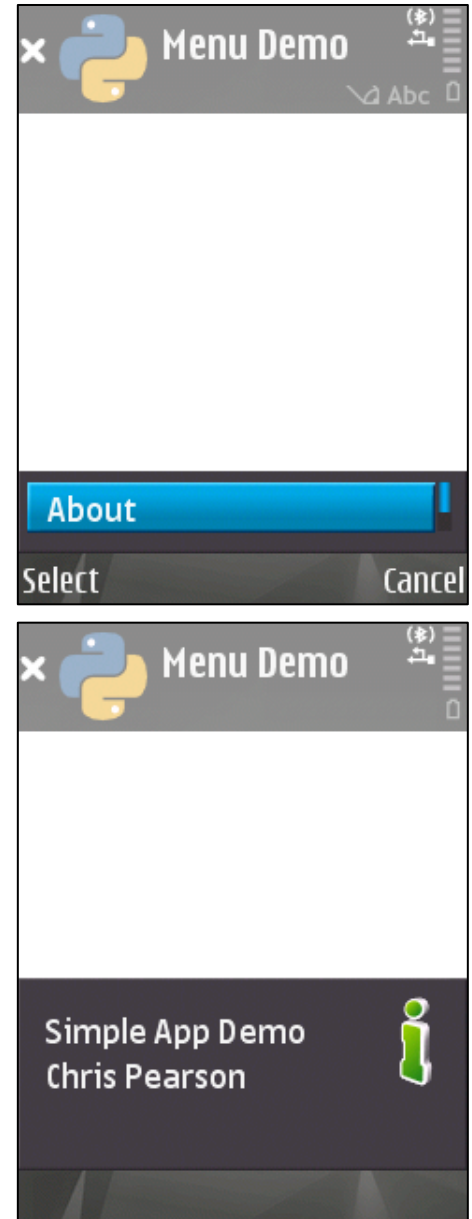
# A Less Simple Application

- ▶ Let's add a menu
  - This requires a callback – a function that will be called when an event occurs

```
import e32
from appuifw import *

# The "about" menu callback function
def about():
    note(u"Simple App Demo\nChris Pearson");

# A menu with "about"
app.menu = [(u"About", about)]
```



# Threads, Locks, and Signals

- ▶ Threads in PyS60 use the phone's:
  - Memory management
  - Power management
- ▶ Locks allow a thread to pause and wait on a signal
- ▶ `app_lock = e32.Ao_lock()`
  - creates a lock object
- ▶ `app_lock.wait()`
  - suspends the thread until signalled
- ▶ `app_lock.signal()`
  - signal all threads waiting on app\_lock



# A Less Simple Application

- ▶ Let's add the GUI thread
  - This halts the main flow of the application
  - Interactive events now control our program

```
# Create the lock variable
app_lock = e32.Ao_lock()
# Tell this thread to wait for a signal
appLock.wait()

# Signal the GUI thread to continue
app_lock.signal()
```



# A Less Simple Application

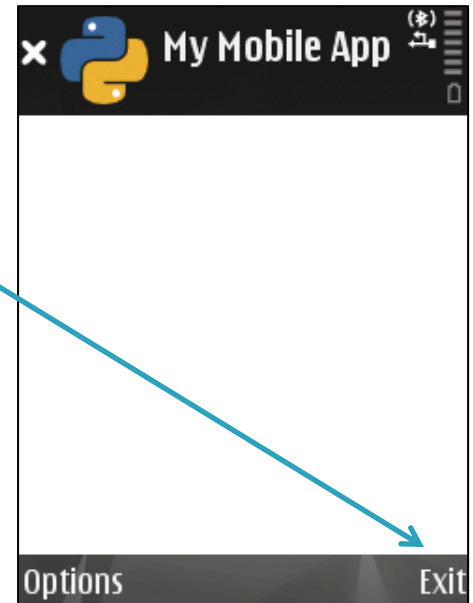
- ▶ A callback for when the “Exit” soft key is pressed:

```
import e32
from appuifw import *
```

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

```
#The function for when exit is pressed
app.exit_key_handler = quit
```

```
#Wait until we get a signal from quit()
app_lock = e32.Ao_lock()
app_lock.wait()
```



# A Less Simple Application

## ► Putting it all together:

```
import e32
from appuifw import *

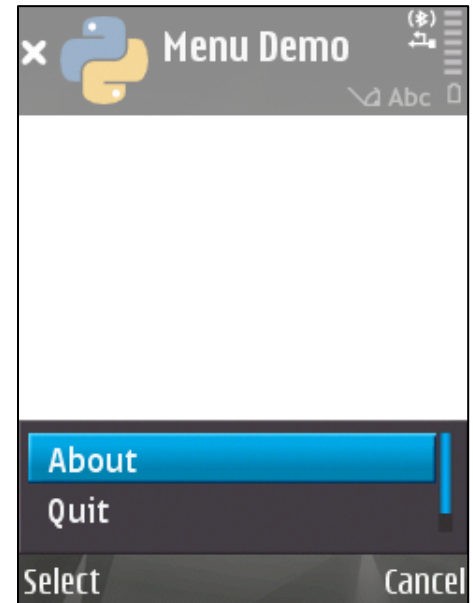
def about():
    note(u"Simple App Demo\nChris Pearson");

#Called by menu and by soft key
def quit():
    app_lock.signal()

#Include the menu
app.menu = [(u"About", about), (u"Quit", quit)]

#The function to call if the quit key is pressed
app.exit_key_handler = quit

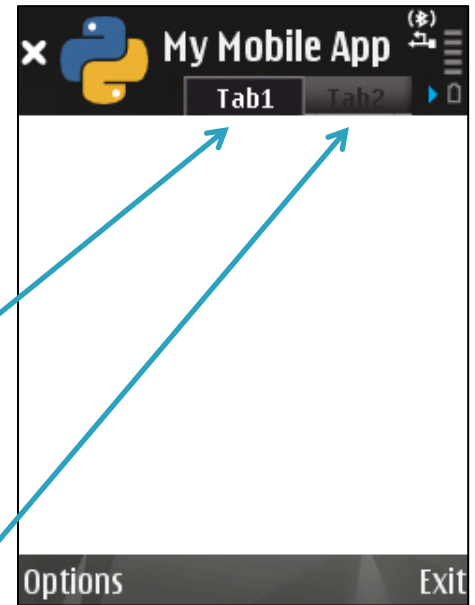
#Wait until we get a signal from quit()
app_lock = e32.Ao_lock()
app_lock.wait()
```





# Tabs

- ▶ Similar to tab gadgets on computers
- ▶ Tab objects are created using:
  - A list of names
  - A callback function name
- ▶ The callback function is passed the index of the selected tab



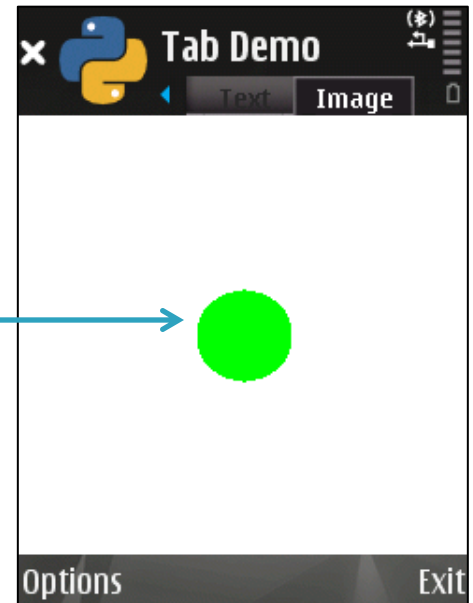
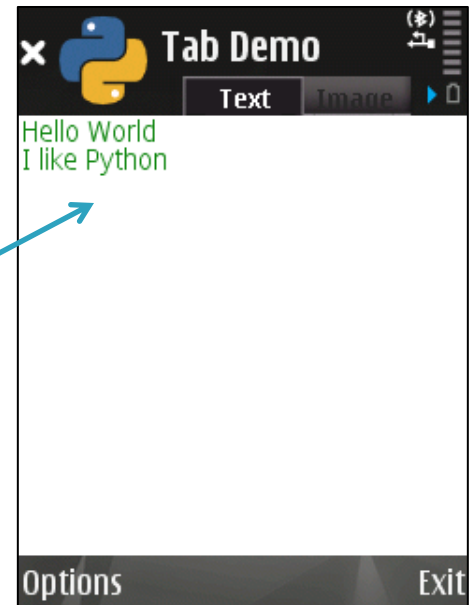
```
app.set_tabs([u"Tab1", u"Tab2"], tabChange)
```



# A Less Simple Application

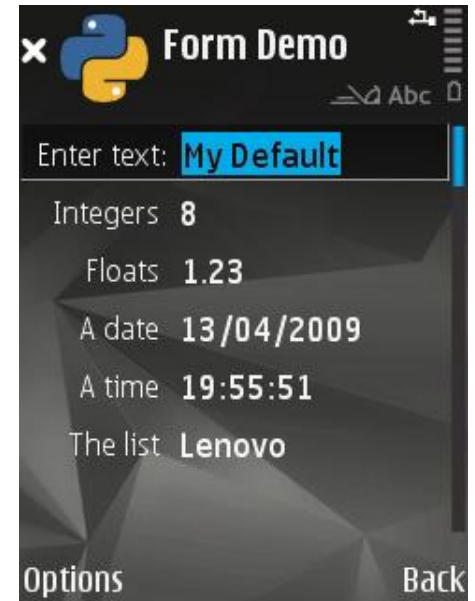
## ▶ Adding tabs:

```
def tabChanged(tabIndex):  
    global canvasArea, isImage  
    # First tab (tab 0) is text  
    if (tabIndex == 0):  
        isImage = False  
        app.body = textArea #display Text object  
  
    # Second tab (tab 1) is canvas  
    elif (tabIndex == 1):  
        isImage = True  
        app.body = canvasArea # display Canvas obj  
        draw_image(None) #Draw the circle  
  
#Call "tabChanged" if tabs are used  
app.set_tabs([u"Text", u"Image"], tabChanged)
```



# Forms

- ▶ Forms are similar to dialog boxes



The screenshot shows a mobile application window titled "Form Demo". It features a dark background with a geometric pattern. The interface includes a text input field with the value "My Default", a numeric input field for "Integers" with the value "8", a numeric input field for "Floats" with the value "1.23", a date input field for "A date" with the value "13/04/2009", a time input field for "A time" with the value "19:55:51", and a list input field for "The list" with the value "Lenovo". At the bottom, there are two buttons: "Options" and "Back".

Field Label	Value
Enter text:	My Default
Integers	8
Floats	1.23
A date	13/04/2009
A time	19:55:51
The list	Lenovo



# Forms

## ► Define a list for a combo box

# Define a combo list for the form

```
a_list = [u"Dell", u"Lenovo",  
u"Acer", u"Microsoft", u"Logitech"]
```



# Forms

## ► Create a form object

# Create the form object

```
formArea = Form([\n    (u"Enter text", "text", u"My Default"),\n    (u"Integers", "number", 8),\n    (u"Floats", "float", 1.23),\n    (u"A date", "date", time.time()),\n    (u"A time", "time", time.time()),\n    (u"The list", "combo", (a_list, 1))\n], FFormEditModeOnly)
```

The screenshot shows a window titled 'Form Demo' with a Python logo. It contains a text input field labeled 'Enter text:' with the value 'My Default'. Below this are several other fields: 'Integers' with value '8', 'Floats' with value '1.23', 'A date' with value '13/04/2009', 'A time' with value '19:55:51', and 'The list' with value 'Lenovo'. At the bottom, there are 'Options' and 'Back' buttons.

## ► Set up a callback for when the form data is saved

# the callback for when the form data is saved

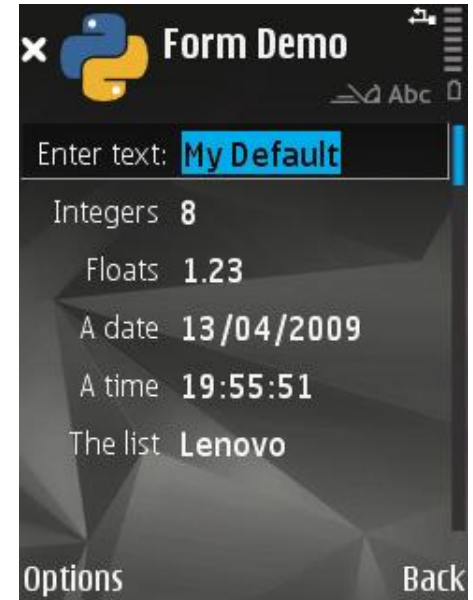
```
formArea.save_hook = save_form_data
```



# Forms

- ▶ Set up a callback for when the form data is saved

```
# the callback for when the form data is saved  
formArea.save_hook = save_form_data
```



The screenshot shows a mobile application interface titled "Form Demo". At the top, there is a Python logo and a close button. Below the title bar, there is a text input field labeled "Enter text:" with the value "My Default". Below this, there are several rows of form fields: "Integers" with the value "8", "Floats" with the value "1.23", "A date" with the value "13/04/2009", "A time" with the value "19:55:51", and "The list" with the value "Lenovo". At the bottom of the screen, there are two buttons: "Options" on the left and "Back" on the right.





# Forms

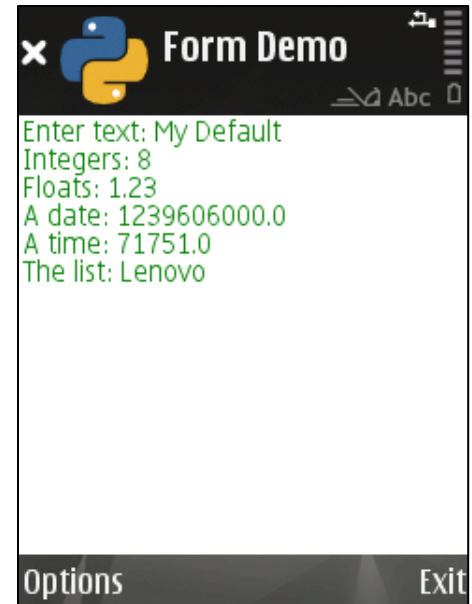
## ► Display the saved data

```
# The callback to handle form data when "save" is chosen
def save_form_data(form_data):
    # put the data into the text area
    text_area.clear()

    # use the title from the form, data the entry field
    text_area.add(form_data[0][0] + u": " + form_data[0][2] + u"\n")

    # the entry field contains an integer, so convert it to a Unicode string
    text_area.add(form_data[1][0] + u": " + unicode(form_data[1][2]) + u"\n")
    text_area.add(form_data[2][0] + u": " + unicode(form_data[2][2]) + u"\n")
    text_area.add(form_data[3][0] + u": " + unicode(form_data[3][2]) + u"\n")
    text_area.add(form_data[4][0] + u": " + unicode(form_data[4][2]) + u"\n")

    # the entry field is a tuple, with the list of items and the index
    new_list, index = form_data[5][2]
    # display the appropriate item from the list returned
    text_area.add(form_data[5][0] + u": " + new_list[index] + u"\n")
```



# Forms

## ► Display the form

```
# display the form and wait for it to exit  
formArea.execute()
```



The screenshot shows a mobile application interface titled "Form Demo". At the top, there is a Python logo and a close button (X). Below the title bar, there is a text input field labeled "Enter text:" with the value "My Default". Below this, there are several rows of form fields: "Integers" with the value "8", "Floats" with the value "1.23", "A date" with the value "13/04/2009", "A time" with the value "19:55:51", and "The list" with the value "Lenovo". At the bottom of the screen, there are two buttons: "Options" on the left and "Back" on the right.



# Form Data

- ▶ These data objects have the same format:
  - The original form data structure
  - The saved form data structure

```
formArea = Form([\n    (u"Enter text", "text", u"My Default"),\n    (u"Integers", "number", 8),\n    (u"Floats", "float", 1.23),\n    (u"A date", "date", time.time()),\n    (u"A time", "time", time.time()),\n    (u"The list", "combo", (a_list, 1))\n], FFormEditModeOnly)
```

```
text_area.add(form_data[0][0] + u": " + form_data[0][2] + u"\n")\n# the entry field contains an integer, so convert it to a Unicode string\ntext_area.add(form_data[1][0] + u": " + unicode(form_data[1][2]) + u"\n")\ntext_area.add(form_data[2][0] + u": " + unicode(form_data[2][2]) + u"\n")\ntext_area.add(form_data[3][0] + u": " + unicode(form_data[3][2]) + u"\n")\ntext_area.add(form_data[4][0] + u": " + unicode(form_data[4][2]) + u"\n")
```



# SENG 462 Tutorial #6

Chris Pearson  
pearson@csc.uvic.ca



University  
of Victoria