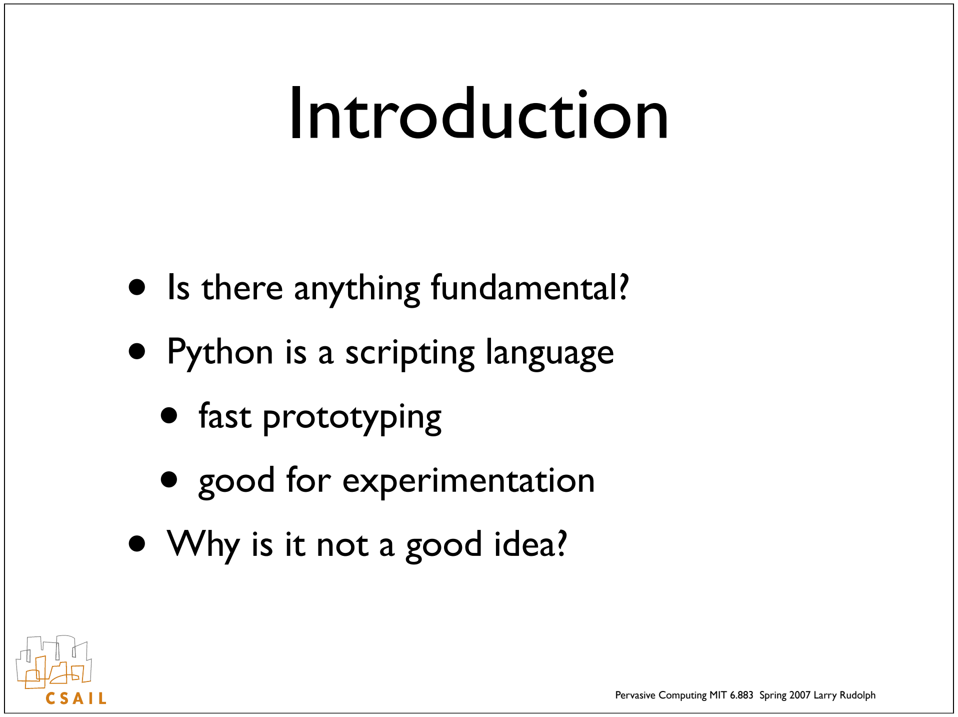
Intro Python Material

To be redone but useful anyway



Introduction

- Is there anything fundamental?
- Python is a scripting language
  - fast prototyping
  - good for experimentation
- Why is it not a good idea?

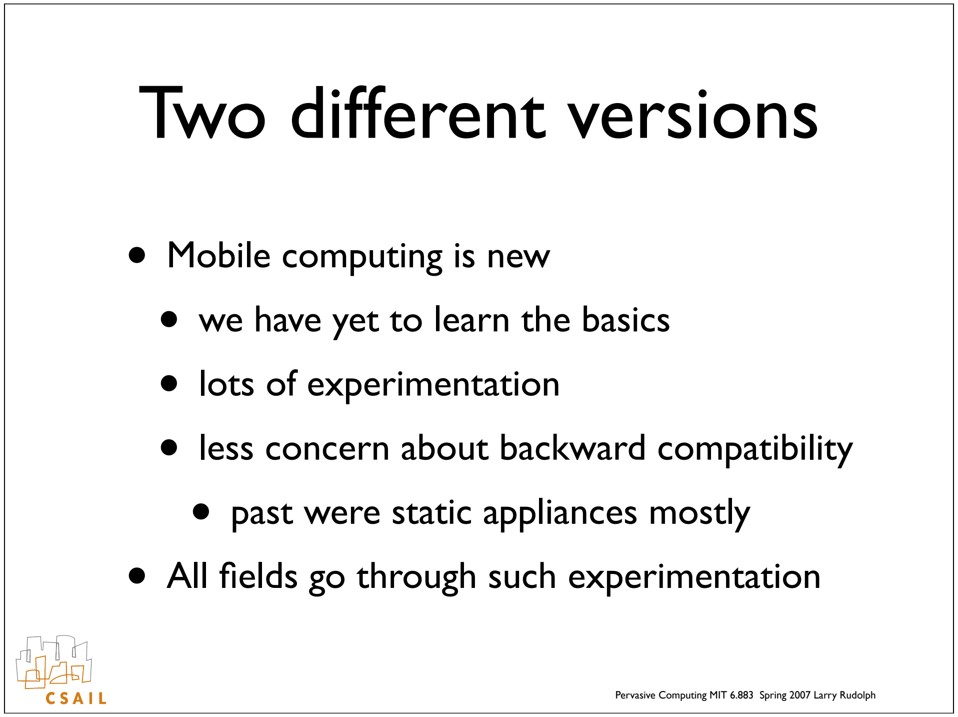
Pervasive Computing MIT 6.883 Spring 2007 Larry Rudolph

There is a bit of tension at this point in the course. On the one hand, the lecture notes should contain information, ideas, concepts, techniques and insights that should be general and eternal. But we need to understand how our phones work.

The big point is that programming languages come in all different shapes and sizes. We have learned that it is best to use the right language for the right task. (If you go on an interview and you are asked which language you prefer, the right answer is that it depends upon the task.)

Python is good for us because we can get started quickly and we are not ready to develop large applications for the phone. It may be surprising how much functionality is possible from having a bunch of phones interacting.

On the otherhand, no language is ideal. What are Python's drawback for cell phone programming? The most obvious is the lack of libraries – there are many things that



Two different versions

- Mobile computing is new
  - we have yet to learn the basics
  - lots of experimentation
  - less concern about backward compatibility
    - past were static appliances mostly
- All fields go through such experimentation

New technologies, especially ones with major economic payoffs, go through a period of great diversity and experimentation. For example, when automobiles were new, there were many different manufactures and lots of innovation. Old movies often show office desks with five or six phones. It takes time to see what works and what doesn't (and often this has little to do with technological superiority -- features win or lose based on a whole range of factors).

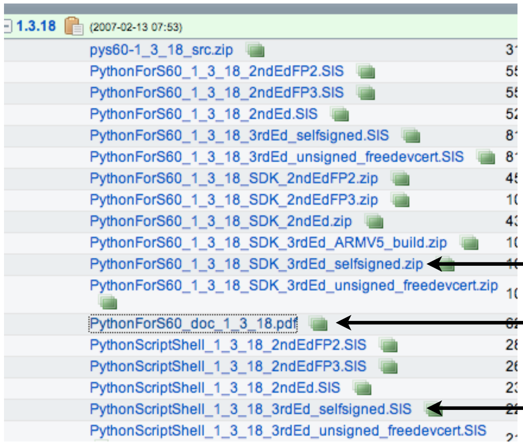
Mobile phones are enjoying a period of great diversity in the search for the right combination of features. But even more so, the appliance aspect imply that there really is little need to be backward compatibility. So, we suffer because of this innovation.

Where to find installation files?

- Sourceforge -- look for pys60
  - Mirrored on our wiki
- Python document is 73 pages, get it
- S60 2nd Edition phones (6680): easy
- S60 3rd Edition phones (N80): trickier

Do not try to understand the numbering. Once again, it is worthwhile to learn to appreciate that technology does not always “rule” -- there are other considerations. Personally, since programs that ran on S60, 2nd edition may not run on 3rd edition phones, it would make sense to give it a new major number, say calling it S65, but I do not make these decisions.

Installation for the certificate phones

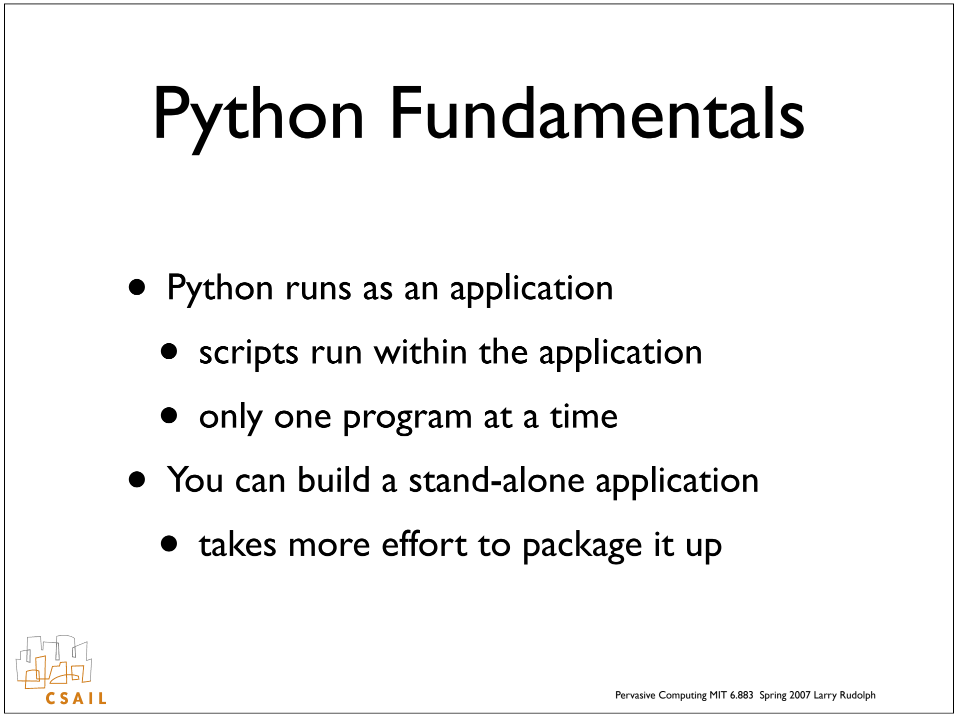


1.3.18 (2007-02-13 07:53)

pys60-1_3_18_src.zip	31
PythonForS60_1_3_18_2ndEdFP2.SIS	55
PythonForS60_1_3_18_2ndEdFP3.SIS	55
PythonForS60_1_3_18_2ndEd.SIS	52
PythonForS60_1_3_18_3rdEd_selfsigned.SIS	81
PythonForS60_1_3_18_3rdEd_unsigned_freedevcert.SIS	81
PythonForS60_1_3_18_SDK_2ndEdFP2.zip	45
PythonForS60_1_3_18_SDK_2ndEdFP3.zip	10
PythonForS60_1_3_18_SDK_2ndEd.zip	43
PythonForS60_1_3_18_SDK_3rdEd_ARMV5_build.zip	10
PythonForS60_1_3_18_SDK_3rdEd_selfsigned.zip	1
PythonForS60_1_3_18_SDK_3rdEd_unsigned_freedevcert.zip	10
PythonForS60_doc_1_3_18.pdf	6
PythonScriptShell_1_3_18_2ndEdFP2.SIS	28
PythonScriptShell_1_3_18_2ndEdFP3.SIS	26
PythonScriptShell_1_3_18_2ndEd.SIS	23
PythonScriptShell_1_3_18_3rdEd_selfsigned.SIS	2
PythonScriptShell_1_3_18_3rdEd_unsigned_freedevcert.SIS	2

- Install the self-signed version on Symbian 9
- The freedevcert have more capabilities but you must sign them yourself

CSAIL



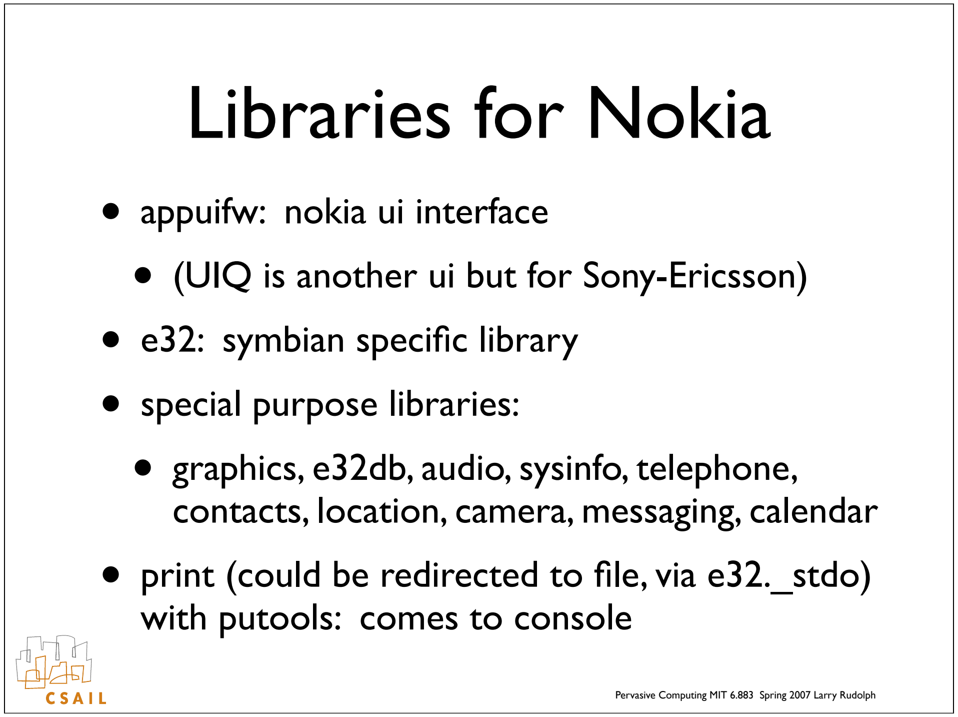
Python Fundamentals

- Python runs as an application
  - scripts run within the application
  - only one program at a time
- You can build a stand-alone application
  - takes more effort to package it up

A program is something that has a start point. It gets loaded by the OS and control is passed to its first instruction. There is meta-information required to tell the loader how and where to place the various parts. An application is a whole lot more than a program. It might have resources, license requirements, an icon, and perhaps capabilities. It is like buying a item in the store. There is all this packaging, SKU's, UPC codes, and a whole bunch of other things that is required. This si true even for a simple connector; the packaging can cost more than the product itself. Modern applications are like this as well.

What is great about using an interpreter, is that Python has already been packaged for us. We need to only write the scripts and we can let the Python application run them for us.

The drawback is that we can only have one python programming running at a time, and the startup actions are multiphased: first startup Python, then select a script, and then execute the script.



Libraries for Nokia

- appuifw: nokia ui interface
  - (UIQ is another ui but for Sony-Ericsson)
- e32: symbian specific library
- special purpose libraries:
  - graphics, e32db, audio, sysinfo, telephone, contacts, location, camera, messaging, calendar
- print (could be redirected to file, via e32._stdo) with putools: comes to console

There are two main libraries. The first is the application user interface frame work (appuifw).
Obviously, it contains all the user interface routines that are part of Nokia’s user interface.
There are other user interface frameworks, UIQ is the other famous one.

- Threads:
  - first thread must be last thread to exit
  - first thread is the UI thread
    - non-first thread may not use appuifw
  - first thread should not use thread.lock()
  - cannot share file handles, sockets, etc.

Processes, Threads, Active Objects

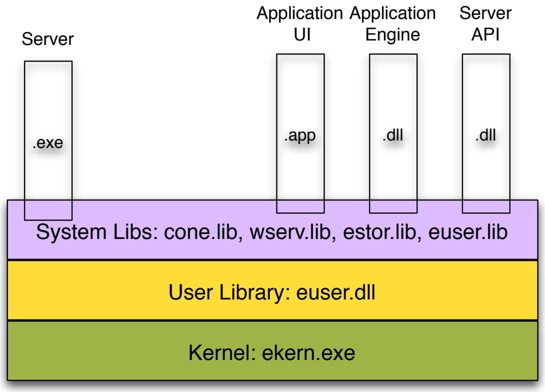
- Process: address space + threads
  - A main thread interacts with user interface. Can directly call UI. If blocks, application blocks
  - Heap (shared by all threads)
  - No statics (in DLL), but yes in new: S60-FP3
- Thread: Program Counter + AO + stack (small)
- AO (Active Object): Pieces of thread code that interacts with User Interface

DLL’s and API’s

- API: the exported published behavior that a system component exposes to other components
- Symbian OS DLL components can:
  - define an API for use by other components (system libs, app. engines)
  - implement an API defined by a framework
    - GUI applications, device drivers
    - these are plug-in’s into a framework

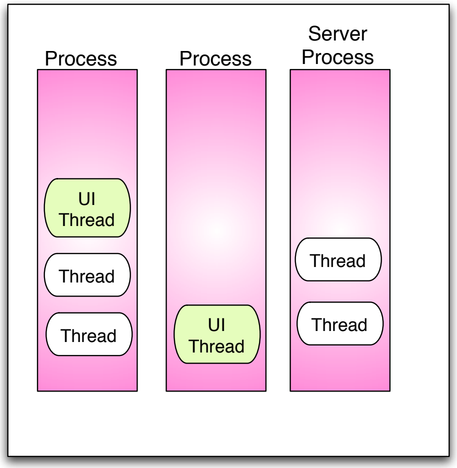
Massachusetts Institute of Technology

Apps, DLL, API


Server
.exe
Application UI
.app
Application Engine
.dll
Server API
.dll
System Libs: cone.lib, wserv.lib, estor.lib, euser.lib
User Library: euser.dll
Kernel: ekern.exe

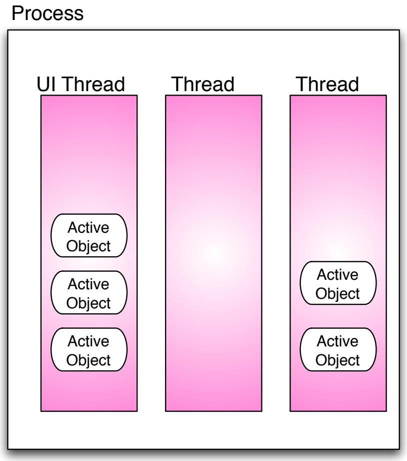
Processes (exe, app)

- an application has only one user interface thread
- sockets & file objects cannot be shared among threads
- why?



Process & Threads

- only one thread in process has access to UI
- sockets & file objects cannot be shared among threads
- why?



What kind of OS?

- Multi-tasking
- Multi-threading
- Real-time

UI Thread

- places objects on screen
- registers callbacks procedures associated with screen & keyboard events
- when event occurs, want to pass control to the callback procedure.
    - what if thread is executing something else?
- Callbacks should execute quickly
- UI thread should spend most of the time idle

Coordination

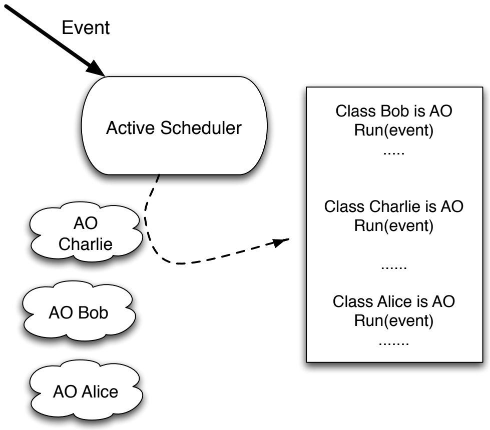
- Don’t use normal thread locks:

```
import thread
lock = thread.allocate_lock()
```

- Whole application gets blocked, since no UI actions would be handled
- Use e32.Ao_lock instead

Active Objects

- If Symbian written today, AO's would be called "listeners"
- Get called by scheduler (have a little bit of state)
- Run to completion then return to scheduler



Event

Active Scheduler

AO Charlie

AO Bob

AO Alice

Class Bob is AO
Run(event)
.....

Class Charlie is AO
Run(event)

......

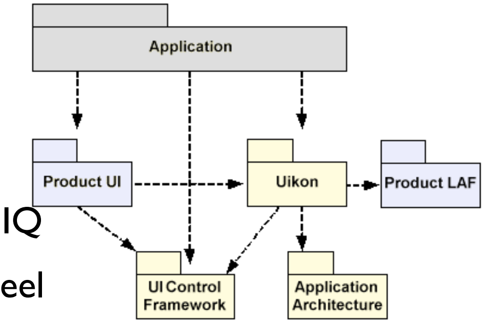
Class Alice is AO
Run(event)
....

Python’s AO

- Previous discussion was for Symbian in general
- Python hides the scheduler
  - but after setting up callbacks, just do a return
- Can control AO by allocating an e32.Ao_lock() and then doing wait() and signal() with this lock

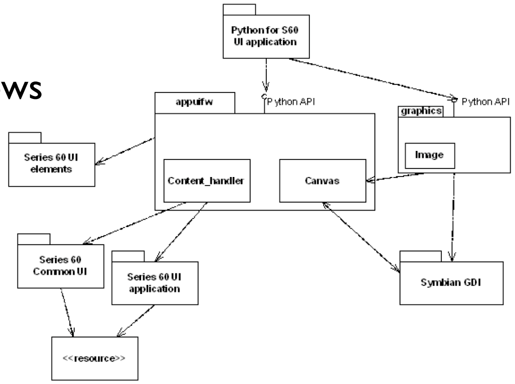
Symbian UI

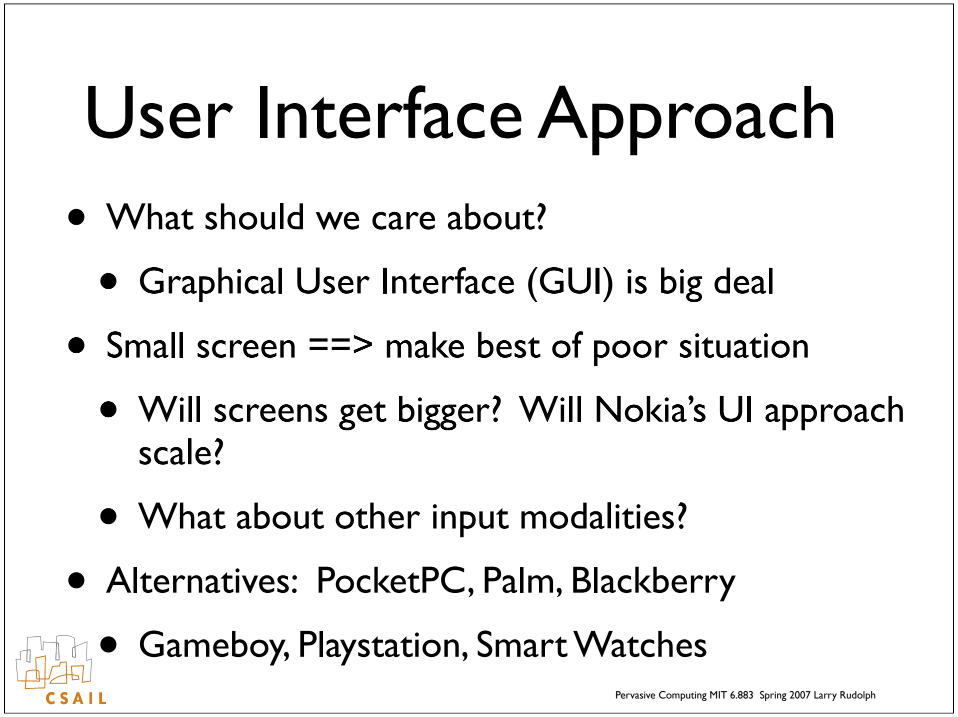
- Uikon is generic user interface components
- Product UI is S60 or UIQ
- Product LAF: Look & Feel
  - None of this is relevant to us now



Python User Interface

- This diagram shows the pieces
- Ignore it





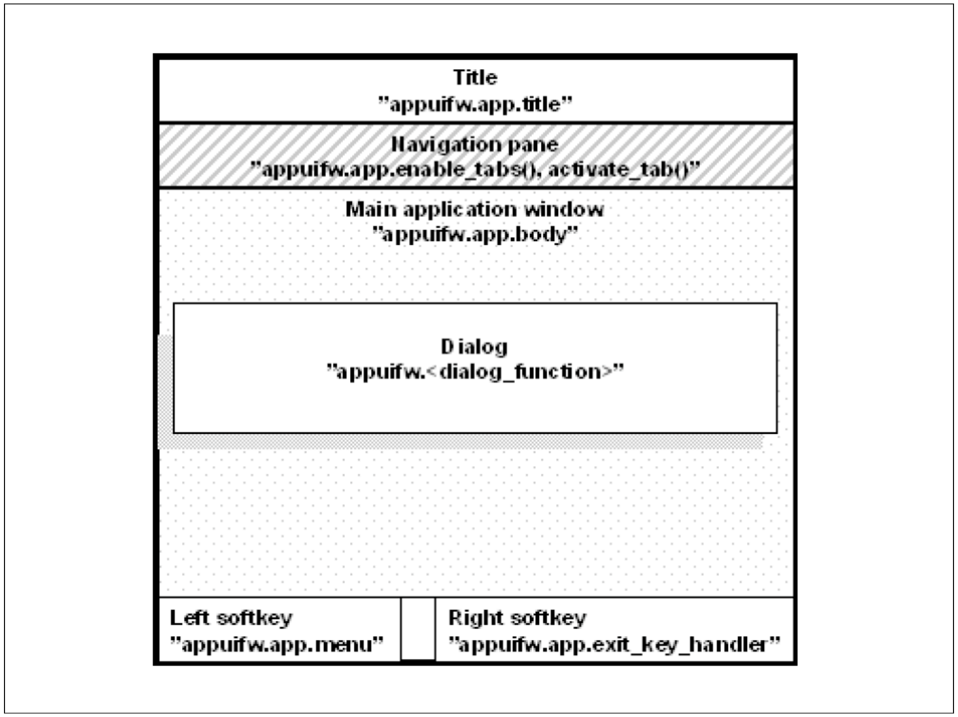
User Interface Approach

- What should we care about?
  - Graphical User Interface (GUI) is big deal
- Small screen ==> make best of poor situation
  - Will screens get bigger? Will Nokia's UI approach scale?
  - What about other input modalities?
- Alternatives: PocketPC, Palm, Blackberry
  - Gameboy, Playstation, Smart Watches

Personally, I do not think any of these UI will survive. They are not suited to mobile, one-handed operation. Perhaps it will be speech-based UI, but that probably does not cover all the usage scenarios. There are lots of alternatives for input: pen, gestures (think about the Nintendo WII), buttons (like on appliances, such as ipod), or telepathy.   In 20 years, come back at tell me if I was right or wrong.

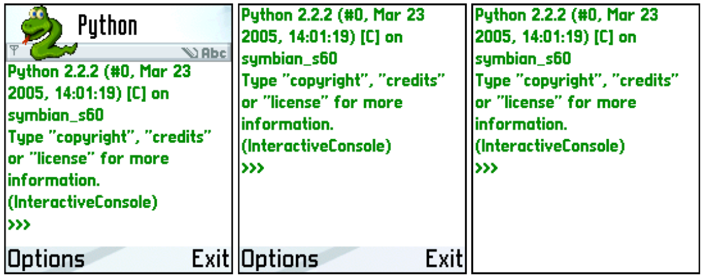
Nokia’s Approach

- Nokia’s UI philosophy (are they unique?)
- Uniform across apps; branded look&feel
  - Screen title at top
  - Optional tabs just below that
  - Body (and for pop-ups)
  - Bottom softkeys: Menu (left), Exit (right)


Title
"appuifw.app.title"
Navigation pane
"appuifw.app.enable_tabs(), activate_tab()"
Main application window
"appuifw.app.body"
Dialog
"appuifw.<dialog_function>"
Left softkey
"appuifw.app.menu"
Right softkey
"appuifw.app.exit_key_handler"

SPy60 Approach

- provide option for more usable screen area
- great for prototyping.
  - Use default font & size; minor graphics



```
Python 2.2.2 (#0, Mar 23
2005, 14:01:19) [C] on
symbian_s60
Type "copyright", "credits"
or "license" for more
information.
(InteractiveConsole)
>>>
```

```
Python 2.2.2 (#0, Mar 23
2005, 14:01:19) [C] on
symbian_s60
Type "copyright", "credits"
or "license" for more
information.
(InteractiveConsole)
>>>
```

```
Python 2.2.2 (#0, Mar 23
2005, 14:01:19) [C] on
symbian_s60
Type "copyright", "credits"
or "license" for more
information.
(InteractiveConsole)
>>>
```

Figure 5.3: UI layouts. left: 'normal', middle: 'large', right: 'full'

Using the screen

- Appuifw contains an instance of the class application, called app

```
appuifw.app.title = u’title of screen’
appuifw.app.screen = ‘normal’ # size
from appuifw import *
app.body = Text | Listbox | Canvas
app.menu = list of (title, callback)
app.set_tabs( list of tab names, callback)
```

SMS messaging

- Can send SMS: sms_send(nmbr, mess)
  - limit of 160 characters
- Can access phone's inbox
  - plop it into a list, access fields of mess
- Register callback for whenever mess arrives
- Need to be connect to phone network and need to be running when msg arrives

```
import e32
import appuifw

e32.ao_yield()

class MyApp:
    def __init__(self):
        self.lock = e32.Ao_lock()

        self.old_title = appuifw.app.title
        appuifw.app.title = u"My Application"
        self.exit_flag = False
        appuifw.app.exit_key_handler = self.abort

        appuifw.app.body = appuifw.Listbox([u"Loading..."], self.handle_modify)
        appuifw.app.menu = [  (u"Add", self.handle_add), (u"Delete", self.handle_delete)]
```

```
import e32
import appuifw
from MyDataAccess import MyDataAccess

e32.ao_yield()

def format(item):
    # Format the item as a short unicode string.
    return u"" # omitted
class MyApp:
    def __init__(self):
        self.lock = e32.Ao_lock()

        self.old_title = appuifw.app.title
        appuifw.app.title = u"My Application"

         self.exit_flag = False
        appuifw.app.exit_key_handler = self.abort

        self.data = []
        appuifw.app.body = appuifw.Listbox([u"Loading..."], self.handle_modify)

        self.menu_add = (u"Add", self.handle_add)
        self.menu_del = (u"Delete", self.handle_delete)
        appuifw.app.menu = []
                # First call to refresh() will fill in the menu.
```

```
def loop(self):
    try:
        self.lock.wait()
        while not self.exit_flag:
            self.refresh()
            self.lock.wait()
    finally:
        self.db.close()

def close(self):
    appuifw.app.menu = []
    appuifw.app.body = None
    appuifw.app.exit_key_handler = None
    appuifw.app.title = self.old_title

def abort(self):
    # Exit-key handler.
    self.exit_flag = True
    self.lock.signal()
```

```
def handle_modify(self):
    item = self.get_current_item()
    if item is not None:
        # Display data in Form for user to edit.
        # Save modified record in database.
        pass                 # omitted

def handle_add(self):
    new_item = self.edit_item(ToDoItem())
    if new_item is not None:
        # User enters new data into Form.
        # Save new record in database.
        pass                 # omitted

def handle_delete(self):
    item = self.get_current_item()
    if item is not None:
        # Remove record from database.
        pass                 # omitted

def get_current_item(self):
    # Return currently selected item, or None if the list is empty.
    if not self.data:
        return None
    else:
        current = appuifw.app.body.current()
        return self.data[current]
```

```
def main():
    app = MyApp()
    try:
        hosts = [u"some.foo.com", u"other.foo.com"]
        i = appuifw.popup_menu(hosts, u"Select server:")
        if i is not None:
            app.connect(hosts[i])
            app.loop()
    finally:
        app.close()

if __name__ == "__main__":
    main()
```

Processes

- Each application and each service (I think) execute as separate processes
  - Each process has its own address space
  - We will not deal with interprocess communication (but could use sockets)
- An application is a process that may have
  - UI and Engine parts
  - Access System and Server APIs

Server Code

```
import sys, socket

if len(sys.argv) < 2:
    print "usage: socketserver <port>"
    sys.exit(2)

# create the server socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

port = int(sys.argv[1])

# allow the socket to be re-used immediately after a close
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

s.bind( ("0.0.0.0", port) )

s.listen(5)              # start the server socket

(client, address) = s.accept()
print "accepted connection from %s:%d" % (address[0], address[1])

while True:
    data = client.recv(1024)
    if len(data) == 0:
        print "connection with %s closed." % address[0]
        break
    sys.stdout.write(data)
client.close()
```

Client Code

```
import sys
import socket

if len(sys.argv) < 3:
    print "usage: socketclient <address> <port>"
    sys.exit(2)

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

s.connect( (sys.argv[1], int(sys.argv[2]) ) )

print "connected.  type stuff."

while True:
    data = sys.stdin.readline()
    if len(data) == 0:
        print "closing connection with server"
        break

    s.send(data)
```

Online Tutorials

- Tutorials
  - http://www.python.org/doc/tut/tut.html
  - http://diveintopython.org/
  - http://www.intelinfo.com/newly_researched_free_training/Python.html
- use google or go to python.org

Discussion about network infrastructure

- Initialization
  - Network
    - Static IP, DNS server -- why IPv6 and why not
    - DHCP: get ip and dns server -- vast improvement
  - Servers
    - Feed, chat, device, anything new
    - too many servers & must always be up
  - What will naive user do?