How the Session Works

Outline
• Practical on arrival
• Talk 1
  • Reflect on practical
  • Clarify concepts
• Practical exercises at your own pace
• Talk 2:
  • Further concepts
  • Overall reflection
• Continue practical exercises at home

Getting Started
• Log-on
• Find portable Python on L: drive
• Start IDLE
• Find resources on teachinglondoncomputing.org
  • Exercise sheet (and notes) – START NOW
  • Example programs
  • Slides
First Program – Click the Button

- Code provided but not yet explained
- Use ‘pattern matching’ (i.e. intelligent guessing) to modify it
Teaching London Computing

A Level Computer Science

Programming GUI in Python

Queen Mary University of London
King's College London
Computing at School
Department for Education
Sponsored by
Mayor of London

William Marsh
School of Electronic Engineering and Computer Science
Queen Mary University of London
Outline

• A first program
• Concepts in **Graphical User Interface**
  • Components / widgets and attributes
  • Events / actions
  • Layout
• *Practical examples*
• Challenges of GUI programming
  • Choosing a GUI library
  • Using Object-Oriented programming
Key Concepts

Explained Using the Button Example
Key Concepts

• A widget / component
  • E.g. a button, a frame
  • Attributes e.g. the button text

• Actions
  • E.g. what happens when you press the button

• Layout
  • Positioning widgets
AppInventor

- Code for events
- Hierarchy of components
- Attributes called properties
- Widgets, called components
- Basic:
  - Button
  - Canvas
  - CheckBox
  - Image
  - Label
  - ListPicker
  - PasswordTextBox
  - TextBox
  - TinyDB
- Media:
  - Animation
  - Social
  - Sensors
  - Screen Arrangement
Widgets

• A GUI is made up from widgets
• A widget is created
• Widget has attributes
• One widget may contain another:
  • Frame contains the button
Create a Widget

• Constructor
  • Name same as widget
  • Hierarchy of widget
  • Optional arguments

# Create a main frame with
# - a title
# - size 200 by 200 pixels
app = Tk()
app.title("GUI Example 1")
app.geometry('200x200')

# Create the button
# - with suitable text
# - a command to call when the button is pressed
button1 = Button(app, text="Click Here", command=clicked)
Widgets have Attributes

- E.g. a name, size
- Any property of the widget that makes it specific

```python
# Create a main frame with
# - a title
# - size 200 by 200 pixels
app = Tk()
app.title("GUI Example 1")
app.geometry('200x200')
```

```python
# Create the button
# - with suitable text
# - a command to call when the button is pressed
button1 = Button(app, text="Click Here", command=clicked)
```
How to Set / Get an Attribute

• Method 1 (setting):
  • Set value with the constructor

• Method 2 (setting and getting):
  • Widget is a dictionary

```
# Change button text
mText = button1['text']
button1['text'] = mText.upper()
```

• Method 3 (sometimes)
  • Call a suitable method
Aside: Dictionaries

- Dictionary: a map from a key to a value
  - Unique key
  - Built in (Python) versus library (many other languages)

<table>
<thead>
<tr>
<th>Standard Array</th>
<th>Python Dictionary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index by number</td>
<td>Key can be a string, pair, …</td>
</tr>
<tr>
<td>Indices continuous e.g.</td>
<td>Gaps ok</td>
</tr>
<tr>
<td>0 → 10</td>
<td>Any value – even a dictionary</td>
</tr>
<tr>
<td>Holds only number, character</td>
<td></td>
</tr>
</tbody>
</table>

# Change button text
mText = button1['text']
button1['text'] = mText.upper()
Handle an Event

# This method is called when the button is pressed
def clicked():
    print("Clicked")

# Create the button with
# - a command to call when the button is pressed
button1 = Button(app, text="Click Here", command=clicked)

- Events
  - Button, mouse click, key press
- Action
  - Event ‘bound’ to function
Layout the Widget

# Make the button visible at the bottom of the frame
button1.pack(side='bottom')

• Where does the widget go?
  • Hierarchy
  • Top-level window

• Layout manager
  • Several available
  • Problem of resizing

• The ‘pack’ layout manager is simplest

• **Widget is not visible until packed**
A Minimal Application

```python
# Import the Tkinter package
# Note in Python 3 it is all lowercase
from tkinter import *

# Create a main frame
app = Tk()

# Start the application running
app.mainloop()
```

import with prefix

```python
# Import the Tkinter package
# Note in Python 3 it is all lowercase
import tkinter as tk

# Create a main frame
app = tk.Tk()

# Start the application running
app.mainloop()
```

Loop to handle events
# (Some) tkinter Widgets

<table>
<thead>
<tr>
<th>Widget</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Button</strong></td>
<td>A button</td>
</tr>
<tr>
<td><strong>Canvas</strong></td>
<td>For drawing graphics</td>
</tr>
<tr>
<td><strong>Entry</strong></td>
<td>Entry a line of text</td>
</tr>
<tr>
<td><strong>Frame</strong></td>
<td>A rectangular area containing other widgets</td>
</tr>
<tr>
<td><strong>Label</strong></td>
<td>Display a single line of text</td>
</tr>
<tr>
<td><strong>Menu</strong></td>
<td>A set of options shown when on a menu bar</td>
</tr>
<tr>
<td><strong>Radiobutton</strong></td>
<td>Select one of a number of choices</td>
</tr>
<tr>
<td><strong>Scrollbar</strong></td>
<td>Horizontal or vertical scrolling of a window</td>
</tr>
<tr>
<td><strong>Text</strong></td>
<td>A multi-line text entry</td>
</tr>
<tr>
<td><strong>Toplevel</strong></td>
<td>A top-level frame</td>
</tr>
</tbody>
</table>
Further Practical Exercises

• See exercise sheet
• A sequence of exercises introduce other widgets and apply the core concepts
• … probably too many to finish now

You may also need to refer to the notes at the end
Further Concepts

- Dialog
- Top-level window
- Control variables
Dialogs

• You must respond to a dialog
• Messages
• File choosing
Top-Level Windows

• At least one top-level window
  • Conveniently created using Tk()
  • Like a frame but …

• Menu bar

• Standard buttons

• Borders
Control Variables

• Variables linking
  • Entry widget to its text
  • Choices in a RadioButton

• These are objects in the framework
Challenges in GUI

• Which framework?
• How to design a GUI
• How much OOP?
GUI Framework

- A GUI framework defines a set of widgets
  - Windows has its own GUI framework
- Python uses a portable GUI framework
  - tkinter, depends on Tk and TCL
  - PyQT, depends on QT
- Tkinter
  - Pro: simple, easy to install
  - Cons: a bit limited; documentation weak
- PyQT: more complex
Designing a GUI

• What am I trying to do?
• What widgets do I need?
  • Where will they go?
  • How do they behave?
The OOP Problem

• Why OO and GUI
  • Widgets are classes
  • Default behaviour
• GUI programs are often organised using classes

• Practical Problem: most examples use OOP

```python
#!/usr/bin/env python
import Tkinter as tk

class Application(tk.Frame):
    def __init__(self, master=None):
        tk.Frame.__init__(self, master)
        self.grid()
        self.createWidgets()

    def createWidgets(self):
        self.quitButton = tk.Button(self,
                                     text='Quit',
                                     command=self.quit)
        self.quitButton.grid()

app = Application()
app.master.title('Sample application')
app.mainloop()
```
Summary

• Core concepts common to all framework
• **Understand** principles
• **Learn about** available widgets
• **Look up** attributes and methods

• After programming … interface design