

Teaching **L**ondon **C**omputing

A Level Computer Science

Programming GUI in Python



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

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



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

The screenshot shows the AppInventor interface with the following components:

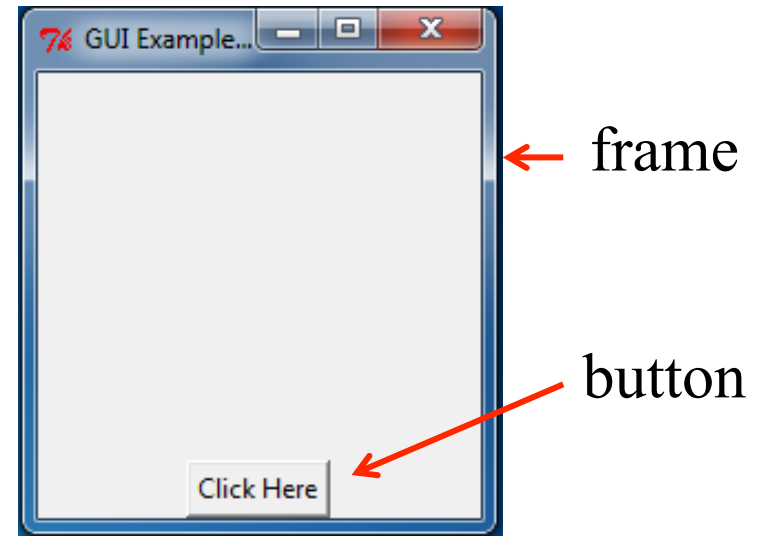
- Palette:** A list of widgets on the left, including Button, Canvas, CheckBox, Image, Label, ListPicker, PasswordTextBox, TextBox, and TinyDB.
- Viewer:** A central workspace showing a mobile screen layout with a title bar (5:09 PM), a header (Screen1), and two text labels (Text for CheckBox1 and Text for Button1).
- Components:** A tree view on the right showing the hierarchy: Screen1 (containing CheckBox1 and Button1).
- Properties:** A panel on the far right showing the attributes for the selected component (Screen1), such as BackgroundColor (White) and Title (Screen1).

Four red callout boxes with white text are overlaid on the interface:

- Code for events:** Points to the top of the Viewer area.
- Widgets, called components:** Points to the Palette area.
- Hierarchy of components:** Points to the Components tree view.
- Attributes called properties:** Points to the Properties panel.

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')
```

Constructor

Parent
widget

Optional
argument

```
# 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

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

Attributes set by
constructor (note use of
keyword arguments)

Methods to
set attributes

```
# 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
-

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

Name of a
Method



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

```
# 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()
```

Loop to
handle events

import with
prefix

```
# 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()
```

(Some) tkinter Widgets

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



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

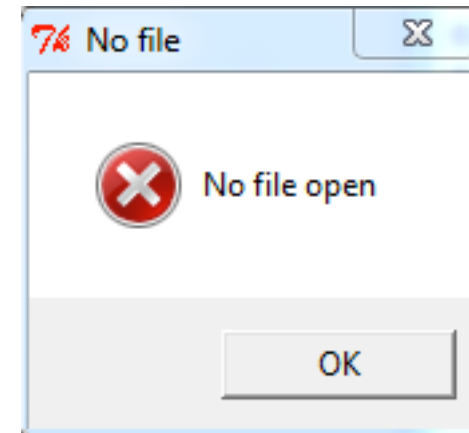
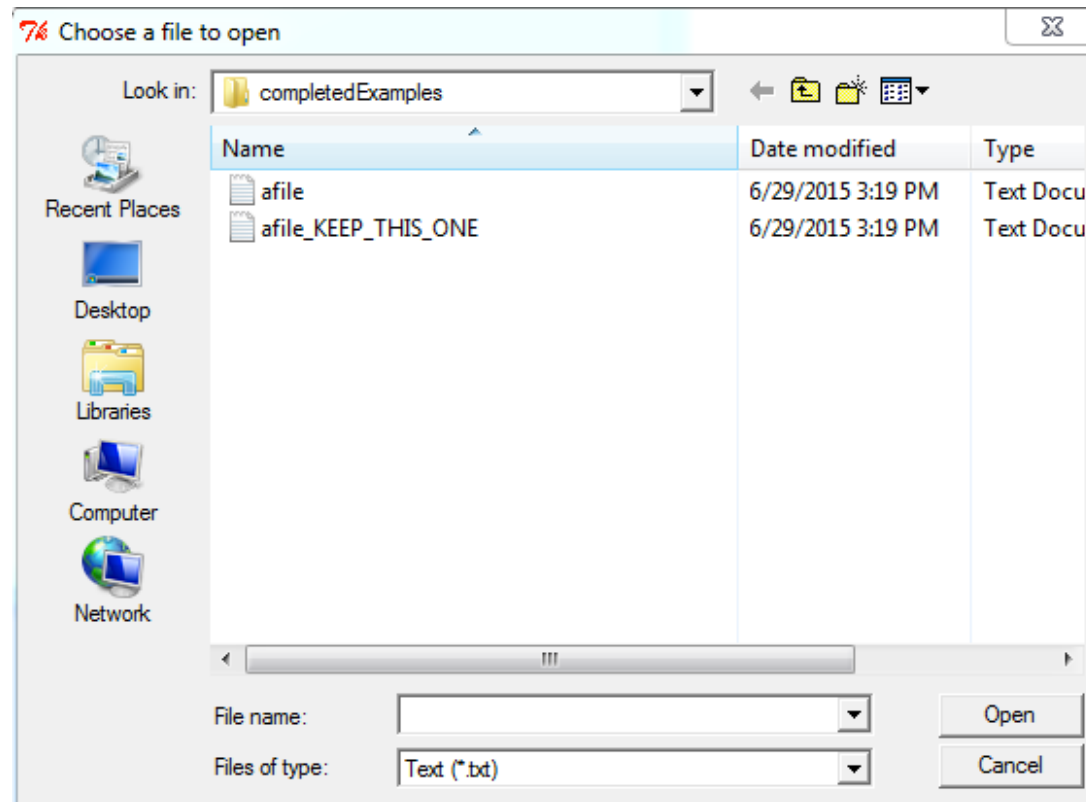


Further Concepts

- 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

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

```
#!/usr/bin/env python      1
import Tkinter as tk      2

class Application(tk.Frame):  3
    def __init__(self, master=None):  4
        tk.Frame.__init__(self, master)  5
        self.grid()

    def createWidgets(self):
        self.quitButton = tk.Button(self, text='Quit',  6
                                     command=self.quit)  7
        self.quitButton.grid()

app = Application()  8
app.master.title('Sample application')  9
app.mainloop()  10
```

- Practical Problem: most examples use OOP
-

Summary

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

 - After programming ... interface design
-