GUIs

David Croft

Coventry University

david.croft@coventry.ac.uk

January 20, 2017

Overview

Event driven

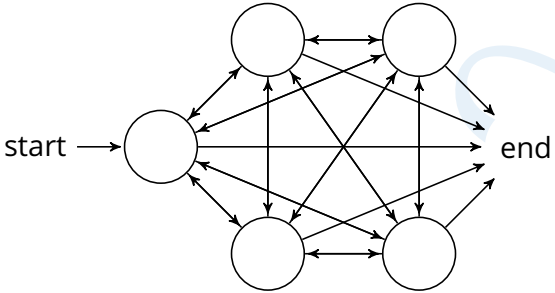
You're programs so far have followed a procedural pattern.

- Program is a series of steps.
- Moves through those steps in a predetermined pattern.
- Expects user input in a very specific order.

Event driven II

Going to look at event driven programming.

- Program reacts to events.
- Events have actions associated with them.
- Order and frequency of events is unpredictable.
- Does not have a predefined sequence of actions to perform.
- Does not have a predefined end.



Benefits

C

What sort of applications would benefit from an event driven paradigm?

Benefits

What sort of applications would benefit from an event driven paradigm?

- GUIs
- Control systems
- Embedded systems

Events examples

GUI events would include...

- Button presses
- Text entry
- Keyboard events
  - Pressing a key
  - Releasing a key
- Mouse events
  - Pressing a button
  - Releasing a button
  - Moving
  - Scrolling

Hello World!

How to create a GUI.

- Wide range of different libraries available.
  - Depends on language and platform.
- Tkinter is the built-in Python default.

Terminology

- Window
- Component/widget/element

Hello World!

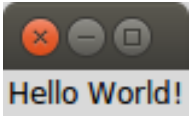

```
import sys
from tkinter import *

def main():
    root = Tk()

    label = Label(root, text='Hello World!')
    label.pack()

    root.mainloop()

if __name__ == '__main__':
    sys.exit(main())
```

lec_getting_started.py

Hello World!


```python
import sys
from tkinter import *

def main():
    root = Tk()

    label = Label(root, text='Hello World!')
    label.pack()

    root.mainloop()

if __name__ == '__main__':
    sys.exit(main())
```

lec_getting_started.py



Classes

GUI code should be structured as a class.

- Become clear later.

```
class Gui:
    def __init__(self, root):
        self.root = root

        self.label = Label(self.root, \
                           text='Hello World!')
        self.label.pack()

def main():
    root = Tk()
    gui = Gui(root)
    root.mainloop()
```

lec_classes.py

Layout

So far we have seen how elements are added to window.

```
class Gui:
    def __init__(self, root):
        self.root = root

        for i in range(1,10):
            button = Button(self.root, text=i)
            button.pack()
```

lec_layout.py

Layout

So far we have seen how elements are added to window.

```
class Gui:
    def __init__(self, root):
        self.root = root

        for i in range(1,10):
            button = Button(self.root, text=i)
            button.pack()
```

lec_layout.py

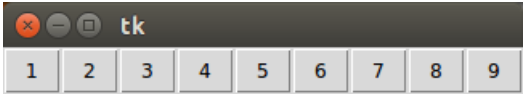
Layout II

Can use the `side` parameter for `.pack()`.

- `TOP` (default).
- Also `LEFT`, `RIGHT` and `BOTTOM`.

```
class Gui:
    def __init__(self, root):
        self.root = root

        for i in range(1,10):
            button = Button(self.root, text=i)
            button.pack(side=LEFT)
```

lec_layout2.py



Layout III

Use `side` to control layout?

```
class Gui:
  def __init__(self, root):
    self.root = root

    Button(self.root, text=1).pack(side=TOP)
    Button(self.root, text=2).pack(side=LEFT)
    Button(self.root, text=3).pack(side=LEFT)
    Button(self.root, text=4).pack(side=TOP)
    Button(self.root, text=5).pack(side=LEFT)
    Button(self.root, text=6).pack(side=LEFT)
    Button(self.root, text=7).pack(side=TOP)
    Button(self.root, text=8).pack(side=LEFT)
    Button(self.root, text=9).pack(side=LEFT)
```

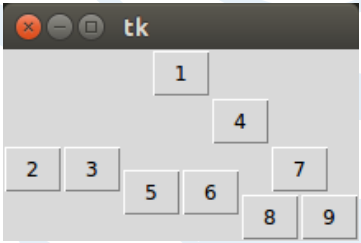
lec_layout3.py

Layout III

Use `side` to control layout?

```
class Gui:
  def __init__(self, root):
    self.root = root

    Button(self.root, text=1).pack(side=TOP)
    Button(self.root, text=2).pack(side=LEFT)
    Button(self.root, text=3).pack(side=LEFT)
    Button(self.root, text=4).pack(side=TOP)
    Button(self.root, text=5).pack(side=LEFT)
    Button(self.root, text=6).pack(side=LEFT)
    Button(self.root, text=7).pack(side=TOP)
    Button(self.root, text=8).pack(side=LEFT)
    Button(self.root, text=9).pack(side=LEFT)
```

lec_layout3.py



Containers

Need to learn about containers.

- Windows are containers.
  - Elements are 'contained' inside.
- Tkinter also has frames.
  - Special type of element.
  - Contains other elements.
- Group elements together using frames.
  - Can be visible/invisible.

Frames


```
class Gui:
  def __init__(self, root):
    self.root = root

    self.frame1 = Frame(self.root)
    self.frame1.pack()

    self.frame2 = Frame(self.root)
    self.frame2.pack()
```

```
    Button(self.frame1, text=1).pack(side=LEFT)
    Button(self.frame1, text=2).pack(side=LEFT)
    Button(self.frame1, text=3).pack(side=LEFT)
```

```
    Button(self.frame3, text=7).pack(side=LEFT)
    Button(self.frame3, text=8).pack(side=LEFT)
    Button(self.frame3, text=9).pack(side=LEFT)
```

lec_frames.py

Frames

```
class Gui:
  def __init__(self, root):
    self.root = root

    self.frame1 = Frame(self.root)
    self.frame1.pack()

    self.frame2 = Frame(self.root)
    self.frame2.pack()
```


```
    Button(self.frame1, text=1).pack(side=LEFT)
    Button(self.frame1, text=2).pack(side=LEFT)
    Button(self.frame1, text=3).pack(side=LEFT)
```

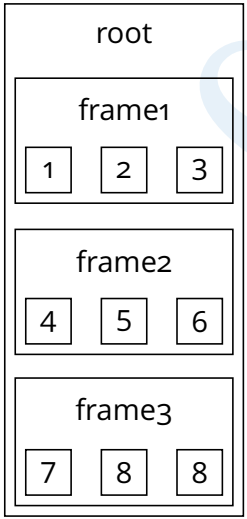
```
    Button(self.frame3, text=7).pack(side=LEFT)
    Button(self.frame3, text=8).pack(side=LEFT)
    Button(self.frame3, text=9).pack(side=LEFT)
```

lec_frames.py

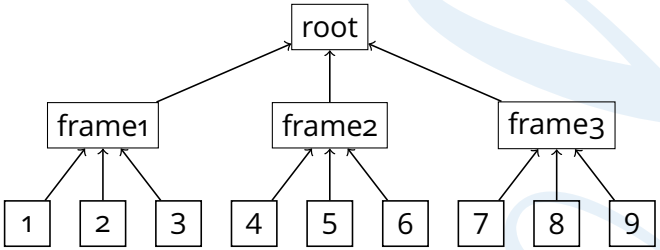
Nesting

So what's happening?

- Elements are nested in containers.
- Containers are nested in other containers.



Hierarchical structure



Break

Events

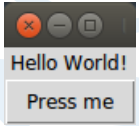
How do we get our code to actually DO stuff?

- Using Python/Tkinter.
- Other languages/frameworks == different syntax.
  - Same concepts.
- Event handling.
  - Bind events to callback functions.

Events II


```
class Gui:
  def __init__(self, root):
    self.root = root

    self.label = Label(self.root, text='Hello World!')
    self.label.pack()

    self.button = Button(self.root, text='Press me')
    self.button.bind('<Button-1>', self.say_bye)
    self.button.pack()

  def say_bye(self, event):
    self.label.config(text='Bye!')
```

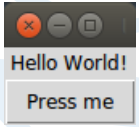
lec_events.py

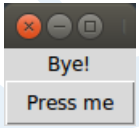


Events II


```
class Gui:
  def __init__(self, root):
    self.root = root

    self.label = Label(self.root, text='Hello World!')
    self.label.pack()

    self.button = Button(self.root, text='Press me')
    self.button.bind('<Button-1>', self.say_bye)
    self.button.pack()

  def say_bye(self, event):
    self.label.config(text='Bye!')
```

lec_events.py





Callbacks

Callbacks are how we respond to events.

- Functions that are passed to another function as an argument.

```
class Gui:
    def __init__(self, root):
        self.root = root

        self.label = Label(self.root, text='Hello World!')
        self.label.pack()

        self.button = Button(self.root, text='Press me')
        self.button.bind('<Button-1>', self.say_bye)
        self.button.pack()

    def say_bye(self, event):
        self.label.config(text='Bye!')
```

lec_events.py

User → Event → Listener → Callback

Standard behaviour

User actions can consist of multiple events.

- I.e. clicking on button.
  1. Press LMB whilst pointer over button.
  2. Release LMB whilst pointer over button.
- Standard behaviour already programmed into Tkinter.
  - Use `command` parameter.

```
class Gui:
    def __init__(self, root):
        self.root = root
```

```
        self.button = Button(self.root, text='Press me' , \
                             command=self.say_bye)
        self.button.pack()

    def say_bye(self):
        self.label.config(text='Bye!')
```

lec_events2.py

Event arguments


```
class Gui:
    def __init__(self, root):
```

```
        Button(self.root, text='1', \
            command=self.pressed_1).pack(side=LEFT)
        Button(self.root, text='2', \
            command=self.pressed_2).pack(side=LEFT)

    def pressed_1(self): # seperate functions to each button
        self.label.config(text='Pressed 1')

    def pressed_2(self): # very similar code
        self.label.config(text='Pressed 2')
```

lec_event_args.py

Event arguments II

Much better to have one function.

- Function takes argument.
- Reuse of each button.

```
class Gui:
    def __init__(self, root):
```

```
        Button(self.root, text='1', \
            command=self.pressed_button(1)).pack(side=LEFT)
        Button(self.root, text='2', \
            command=self.pressed_button(2)).pack(side=LEFT)

    def pressed_button(self, number):
        self.label.config(text='Pressed %d' % number)
```

lec_event_args2.py

Event arguments II

Much better to have one function.

- Function takes argument.
- Reuse of each button.
- Doesn't work.
  - Calls function immediately.
- DEMO


```
class Gui:
    def __init__(self, root):
```

```
        Button(self.root, text='1', \
            command=self.pressed_button(1)).pack(side=LEFT)
        Button(self.root, text='2', \
            command=self.pressed_button(2)).pack(side=LEFT)

    def pressed_button(self, number):
        self.label.config(text='Pressed %d' % number)
```

lec_event_args2.py

Event arguments III

`lambda` functions.

- Only calls function when button is pressed.


```
class Gui:
    def __init__(self, root):
```

```
        Button(self.root, text='1', \
            command=lambda: self.pressed_button(1)).pack(side=LEFT)
        Button(self.root, text='2', \
            command=lambda: self.pressed_button(2)).pack(side=LEFT)

    def pressed_button(self, number):
        self.label.config(text='Pressed %d' % number)
```

lec_event_args3.py

Why do I care?

- Everyone
  - Ability to create simple Graphical User Interfaces (GUIs).
  - Experience in using 3rd party libraries/modules in software.
  - Introduction to event driven programming.
  - Introduction to lambdas.
- Games Tech & MC - Tkinter like APIs are not suited for games but can be used for game menus.
  - Particular attention to callbacks for game input.
- Computing - Similar APIs used in mobile applications.
  - Event driven programming used in ubiquitous computing.
- Ethical Hackers - Security flaws in event driven applications.
- ITB - GUIs programs have lower entry barrier, important for being user friendly.

Quiz

Q1

What is it called when a program is written to respond to button clicks, menu selections and other actions the user performs?

1. Event driven programming
2. Action driven programming
3. User driven programming
4. Mouse driven programming

Q1

What is it called when a program is written to respond to button clicks, menu selections and other actions the user performs?

1. Event driven programming
2. Action driven programming
3. User driven programming
4. Mouse driven programming

Q2

What is wrong with this code?

```
class Gui:
  def __init__(self, root):
    for i in range(1,10):
      b = Button(self.root, text=i, command=self.pressed_button(i))
      b.pack(side=LEFT)

  def pressed_button(self, number):
    print( 'Pressed button {}'.format(number) )
```

1. All the buttons will say they are button 10
2. Each button will print a message twice for each mouse click
3. Each button will only print a message once, as it is created.
4. There will be no buttons

Q2

What is wrong with this code?

```
class Gui:
  def __init__(self, root):
    for i in range(1,10):
      b = Button(self.root, text=i, command=self.pressed_button(i))
      b.pack(side=LEFT)

  def pressed_button(self, number):
    print( 'Pressed button {}'.format(number) )
```

1. All the buttons will say they are button 10
2. Each button will print a message twice for each mouse click
3. Each button will only print a message once, as it is created.
4. There will be no buttons

Q3

What is a callback?

1. The code that deals with GUI events.
2. Unlikely if your first date went badly.
3. A named piece of code that can be repeated multiple times.
4. A function that is passed to another function as an argument.

Q3

What is a callback?

1. The code that deals with GUI events.
2. Unlikely if your first date went badly.
3. A named piece of code that can be repeated multiple times.
4. A function that is passed to another function as an argument.

Q4

What is a container?

1. The class containing your GUI code.
2. A GUI object that can hold other objects within it.
3. A function containing the code to run when a button is pressed.
4. Tupperware.

Q4

What is a container?

1. The class containing your GUI code.
2. A GUI object that can hold other objects within it.
3. A function containing the code to run when a button is pressed.
4. Tupperware.

Recap

- GUIs are an example of event driven programming.
- GUI elements are arranged in containers.
- Containers can hold other containers.
- User actions generate events.
- Callbacks are functions that are run in response to events.

The End