# GUI – What next?

Basic Programming in Python

Sebastian Höffner  Aline Vilks

Wed, 28 June 2017

# User interfaces

How does the user interact with your code?

- Command line interface (CLI)
- Graphical user interface (GUI)

There are many more fine-grained definitions and notions!

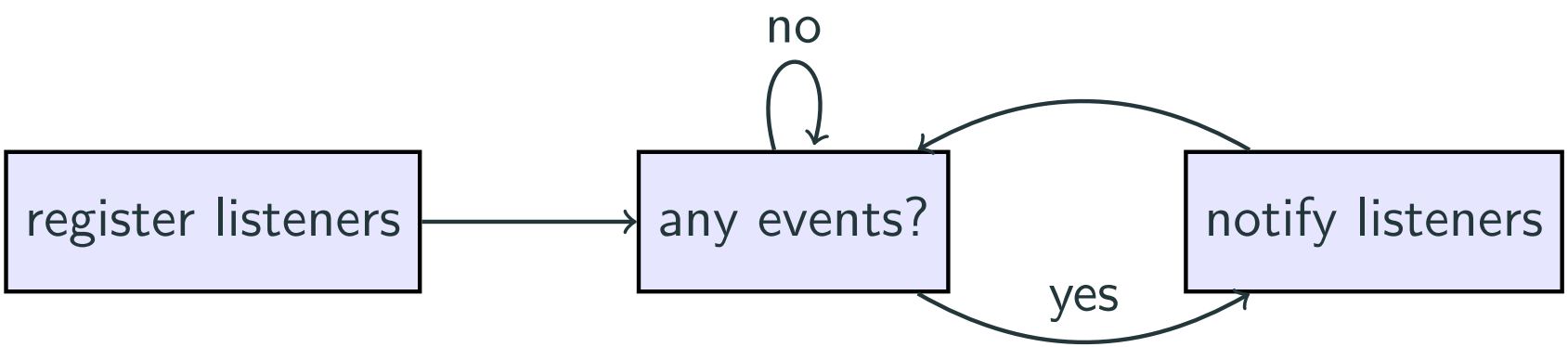
# Command line interfaces

- Issue command after command
- Mostly used inside the terminal
- Examples: Shell, Python, Text adventures, ...

# Graphical user interfaces

- Event driven
- Render windows, buttons, etc.
- Examples: spyder, webbrowsers, office programs. . .

# Event driven



# Event types

What events do you think can happen?

# Event types

- Keyboard inputs/Mouse Inputs
- Opening, Closing, Minimizing, Maximizing
- Screen updates
- Calculation results
- ...

$\Rightarrow$ High complexity and flexibility needed!

# Don’t reinvent the wheel

- Tkinter
- Qt
- native solutions

# GUI – What next?

## Don't reinvent the wheel

Don't reinvent the wheel

- Tkinter
- Qt
- native solutions

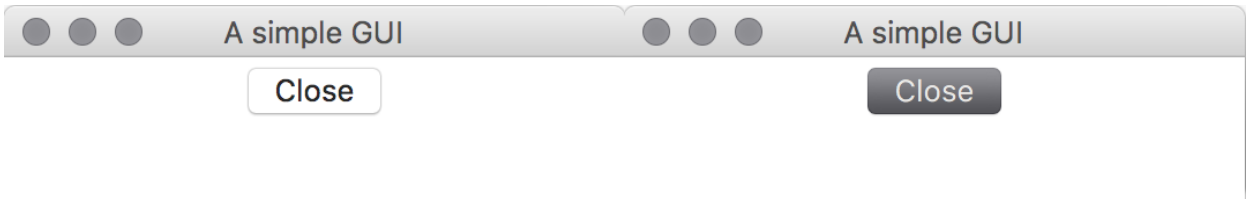
There are many GUI frameworks. The most common one in Python is Tkinter.

Tkinter is just the Python "translation" of Tcl/Tk, which can be found here: https://tcl.tk/man/tcl8.5/TkCmd/contents.htm

- Partial documentation: http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/index.html.
- Introduction to Tkinter: http://effbot.org/tkinterbook/
- Some example codes: https://python-textbok.readthedocs.io/en/1.0/Introduction_to_GUI_Programming.html
- Official documentation: https://docs.python.org/3/library/tk.html

# Redraw

GUIs need to redraw changes, e.g. a button press:



# Redraw demo

*File:* button_only.py

```
from tkinter import Tk, Button

class SimpleWindow:

    def __init__(self, root):
        self.root = root
        root.title("A simple GUI")

        self.close_button = Button(root, text="Close",
                                   command=root.quit)
        self.close_button.pack()

if __name__ == '__main__':
    root = Tk()
    SimpleWindow(root)
    root.mainloop()
    root.destroy()
```

# GUI – What next?

## Redraw demo

**Redraw demo**

*File:* button_only.py

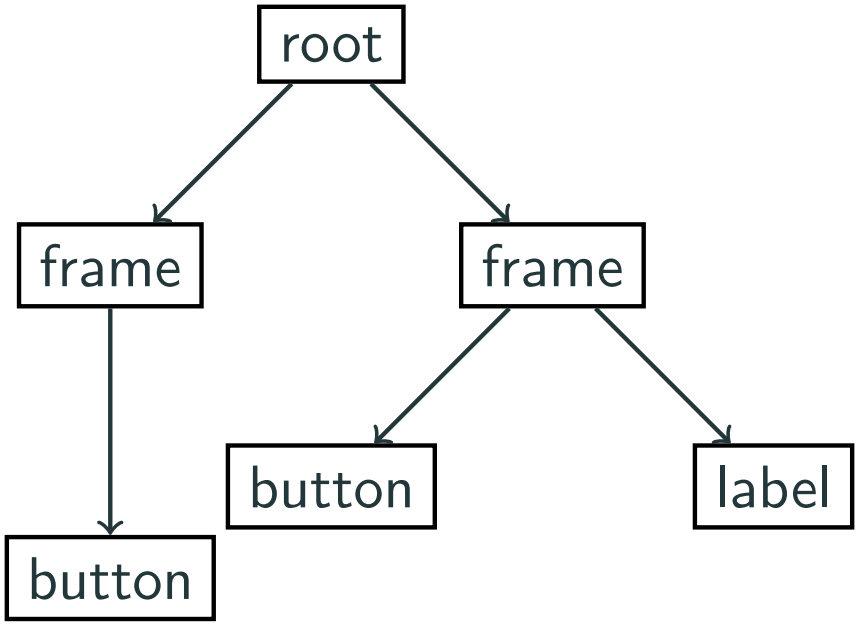
```
from tkinter import Tk, Button

class SimpleWindow:

    def __init__(self, root):
        self.root = root
        root.title("A simple GUI")

        self.close_button = Button(root, text="Close",
                                   command=root.quit)
        self.close_button.pack()

if __name__ == '__main__':
    root = Tk()
    SimpleWindow(root)
    root.mainloop()
    root.destroy()
```

`Tk()` creates the window (the "root" element), `mainloop` runs the event loop and handles events.

Since it runs indefinitely, it also keeps the program from closing!

The Button can close the program (`quit` on the root element).

# Redraw: A tree approach



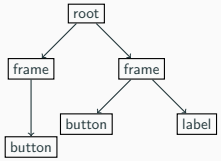

- Each element has a parent (except root).
- Each element knows its children.

Why is this useful?

# GUI – What next?

## Redraw: A tree approach

Redraw: A tree approach




- Each element has a parent (except root).
- Each element knows its children.

Why is this useful?

Using a tree is useful because on updates of an element only that element and its children need to be redrawn.

# The tree GUI

*File:* `tree_example.py`

```
import tkinter as tk

class TreeWindow:

    def __init__(self, root):
        self.root = root
        root.title("The tree example GUI")

        self.frame1 = tk.Frame(root, border=4, relief=tk.SUNKEN)
        self.frame1.pack(fill=tk.X, padx=5, pady=5)
        self.frame2 = tk.Frame(root, border=4, relief=tk.SUNKEN)
        self.frame2.pack(fill=tk.X, padx=5, pady=5)

        self.close_button = tk.Button(self.frame1, text="Close", command=root.quit)
        self.close_button.pack()
        self.do_nothing = tk.Button(self.frame2, text="Do nothing")
        self.do_nothing.pack()

        self.label = tk.Label(self.frame2, text="This is a label.")
        self.label.pack()

if __name__ == '__main__':
    root = tk.Tk()
    TreeWindow(root)
    root.mainloop()
    root.destroy()
```

# GUI – What next?

## └The tree GUI

### The tree GUI

*File:* tree_example.py

```
import tkinter as tk

class TreeWindow:

    def __init__(self, root):
        self.root = root
        root.title("The tree example GUI")

        self.frame1 = tk.Frame(root, border=4, relief=tk.SUNKEN)
        self.frame1.pack(fill=tk.X, padx=5, pady=5)
        self.frame2 = tk.Frame(root, border=4, relief=tk.SUNKEN)
        self.frame2.pack(fill=tk.X, padx=5, pady=5)

        self.close_button = tk.Button(self.frame1, text="Close", command=root.quit)
        self.close_button.pack()
        self.do_nothing = tk.Button(self.frame2, text="Do nothing")
        self.do_nothing.pack()

        self.label = tk.Label(self.frame2, text="This is a label.")
        self.label.pack()

if __name__ == '__main__':
    root = tk.Tk()
    TreeWindow(root)
    root.mainloop()
    root.destroy()
```

The elements (or “Widgets”) used here are:

- Button: A button to click
- Label: Contains descriptions text
- Frame: Groups together different elements

# Buttons and event callbacks

```
def print_action():
    print('Hello World')

tk.Button(root, text="Print!", command=print_action)
```

# GUI – What next?

## Buttons and event callbacks

Buttons and event callbacks

```
def print_action():
    print('Hello World')

tk.Button(root, text="Print!", command=print_action)
```

Buttons take a callback function.

Whenever you click a button, the function is executed.

# Labels

```
tk.Label(root, text='This is static label text')
```

# Changing label texts

*File:* `update_label.py`

```
import tkinter as tk

class ToggleWindow:

    def __init__(self, root):
        root.title("Updating a label")
        self.toggle_button = tk.Button(root, text="Toggle", command=self.toggle)
        self.toggle_button.pack()

        self.label_text = tk.StringVar()
        self.label_text.set('Hello!')
        self.label = tk.Label(root, textvariable=self.label_text)
        self.label.pack()

    def toggle(self):
        if self.label_text.get() == 'Hello!':
            self.label_text.set('Bye!')
        else:
            self.label_text.set('Hello!')

if __name__ == '__main__':
    root = tk.Tk()
    ToggleWindow(root)
    root.mainloop()
    root.destroy()
```

# GUI – What next?

## Changing label texts

`tk.StringVar` (and others: `IntVar`, `DoubleVar`) wrap Python data into a format Tcl/Tk can understand. They allow to update components if you change their values.

Note that you need to change the argument name from `text` for the static solution to `textvariable`.

**Changing label texts**

*File:* update_label.py

```
import tkinter as tk

class ToggleWindow:

    def __init__(self, root):
        root.title("Updating a label")
        self.toggle_button = tk.Button(root, text="Toggle", command=self.toggle)
        self.toggle_button.pack()

        self.label_text = tk.StringVar()
        self.label_text.set('Hello!')
        self.label = tk.Label(root, textvariable=self.label_text)
        self.label.pack()

    def toggle(self):
        if self.label_text.get() == 'Hello!':
            self.label_text.set('Bye!')
        else:
            self.label_text.set('Hello!')

if __name__ == '__main__':
    root = tk.Tk()
    ToggleWindow(root)
    root.mainloop()
    root.destroy()
```

# Organizing interface components: Layout management

Noticed the `pack` call everywhere?

```
label = tk.Label(root, text='Some label')
label.pack()
```

It registers the widget with the *geometry manager*

# GUI – What next?

## Organizing interface components: Layout management

Organizing interface components: Layout management

Noticed the pack call everywhere?

```
label = tk.Label(root, text='Some label')
label.pack()
```

It registers the widget with the *geometry manager*

http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/layout-mgt.html

The geometry manager determines where layout components need to be placed. You just need to tell it what to include (and in which relations) and it will do all the pixel level math for you.

# Layout management

More common than the `pack()` method is `grid()`:

```
label = tk.Label(root, text='Some label in the second row')
label.grid(row=1, column=0, columspan=3)
```

# Layout management example

*File:* `grid_example.py`

```
import tkinter as tk

class GridWindow:

    def __init__(self, root):
        root.title("The tree example GUI")

        self.label = tk.Label(root, text='Some label in the second row')
        self.label.grid(row=1, column=0, columnspan=3)

        self.close_button = tk.Button(root, text="Close", command=root.quit)
        self.close_button.grid(row=0, column=1, columnspan=2)
        self.do_nothing = tk.Button(root, text="Do nothing")
        self.do_nothing.grid(row=0, column=0)

if __name__ == '__main__':
    root = tk.Tk()
    GridWindow(root)
    root.mainloop()
    root.destroy()
```

# Combing everything we learned

In the accompanying zip there is an example project, `iris_viz`.

It contains a lot of things we discussed during the lecture:

- File I/O: It downloads the iris data set if it's not available
- Plotting: It allows to plot iris data
- GUIs
- Error handling and documentation

Some examples are on the next slides, but we will skip those as I'll show it in the code.

# File I/O

```
try:
    with open('iris.data', 'r') as iris_file:
        data = iris_file.read()
except FileNotFoundError:
    url = 'https://archive.ics.uci.edu/ml/machine-learning-
    data = requests.get(url).text
    with open('iris.data', 'w') as iris_file:
        iris_file.write(data)
```

# Plotting

```
def update_plot(self):
    axes = self.figure.gca()
    axes.clear()
    axes.set_title('Iris data')

    x = self._x_selection.get()
    y = self._y_selection.get()
    axes.set_xlabel(self.labels[x] + ' in cm')
    axes.set_ylabel(self.labels[y] + ' in cm')

    for cl, col in zip(list(set(d[4] for d in self.data)),
                       ['orange', 'green', 'blue']):
        axes.scatter(*zip(*((d[x], d[y]) for d in self.data if d[4] == cl)),
                     color=col, label=cl)

    axes.legend()
```

# GUI

```
for row, label in enumerate(self.labels[:-1]):
    radio = tk.Radiobutton(self.frame_x, variable=self._x_selection,
                           value=row, text=label,
                           command=self.update_plot)
    radio.grid(row=row, column=1, sticky=tk.W)
```

# Error handling

```
def maybe_float(v):
    try:
        return float(v)
    except ValueError:
        return v
```

# The end is near

You have seen what you can build with all you know now. It is time to do your own projects!

# What we covered

- Variables, functions, classes, modules, packages
- Collections
- Reading and writing files
- Downloading data, parsing data, regex
- Math, statistics, plotting
- Times, dates
- GUI programming
- and more

# What we did not cover – and where to find it

- Inheritance (Computer Science B)
- Multithreading (Computer Science B)
- Numpy (Neuroinformatics, Machine Learning, Computer Vision)
- Software architectures, project design (Software engineering)
- System architecture, hardware (Computer Science C)
- Algorithms (Computer Science D, many other classes)
- Databases (Database systems)
- ...

# Stay sharp and keep going

> *Program. The best kind of learning is learning by doing.*
>
> *. . .*
>
> *Talk with other programmers; read other programs. This is more important than any book or training course.*
>
> *. . .*
>
> *Learn at least a half dozen programming languages.*

**Peter Norvig**: 21 days[1]

[1] http://norvig.com/21-days.html

# Challenges to improve your skills

**https://github.com/karan/Projects** Small functions up to medium sized projects: Fibonacci sequence to RSS readers

**https://projecteuler.net/** Lots of math problems, e.g. find the sum of the first 1000 prime numbers

**http://natureofcode.com/book/** An introduction to more scientific programming (math, simulations, . . . ) than this class

# My advice

- If you just need Python for some data analyses, install numpy and get started!
- If you enjoyed the class, try to join "Scientific Programming in Python" the next time it's offered.
- If you want to dive deeper into programming, learn other programming languages. Java, C++, Prolog, Haskell, Lisp, there are many many many more.
- Join next week's lecture in 93/E42 to see all awesome projects!

# GUI – What next?

## My advice

My advice

- If you just need Python for some data analyses, install numpy and get started!
- If you enjoyed the class, try to join "Scientific Programming in Python" the next time it's offered.
- If you want to dive deeper into programming, learn other programming languages. Java, C++, Prolog, Haskell, Lisp, there are many many many more.
- Join next week's lecture in 93/E42 to see all awesome projects!

For numpy, definitely checkout these awesome articles:

- https://docs.scipy.org/doc/numpy-1.12.0/reference/arrays.indexing.html
- https://scipy.github.io/old-wiki/pages/EricsBroadcastingDoc

You don't need to master other programming languages, but it helps to get away from "Syntax" and start thinking about "Semantics" of code, you will become better at generalizing ideas and concepts instead of focusing on a particular parenthesis or bracket.

# References