Python for Scientists

Part 8-2 – User Interface (UI)

Cyril Desjouy

June, 2016
Updated : January 20, 2019

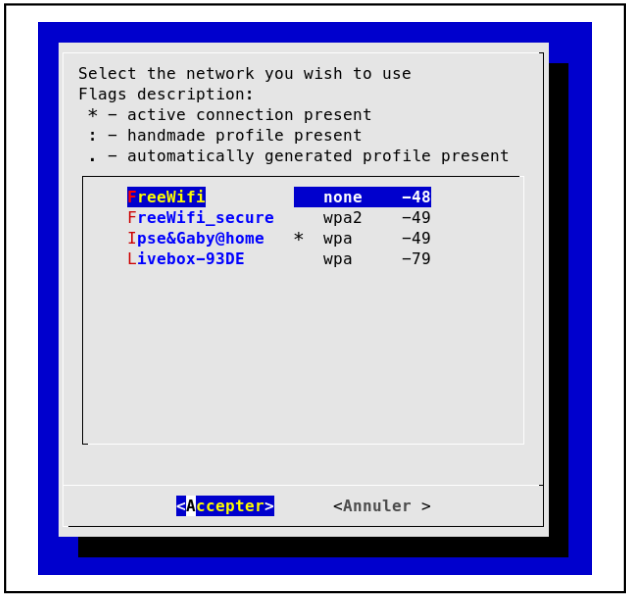
Python has a huge number of UI toolkits : See here a list

- Some are graphicals, others are text-based
- Some are platform-specific, other are cross-platform
- Some are simples, others are complicated

Here are some of the main GUI toolkits :

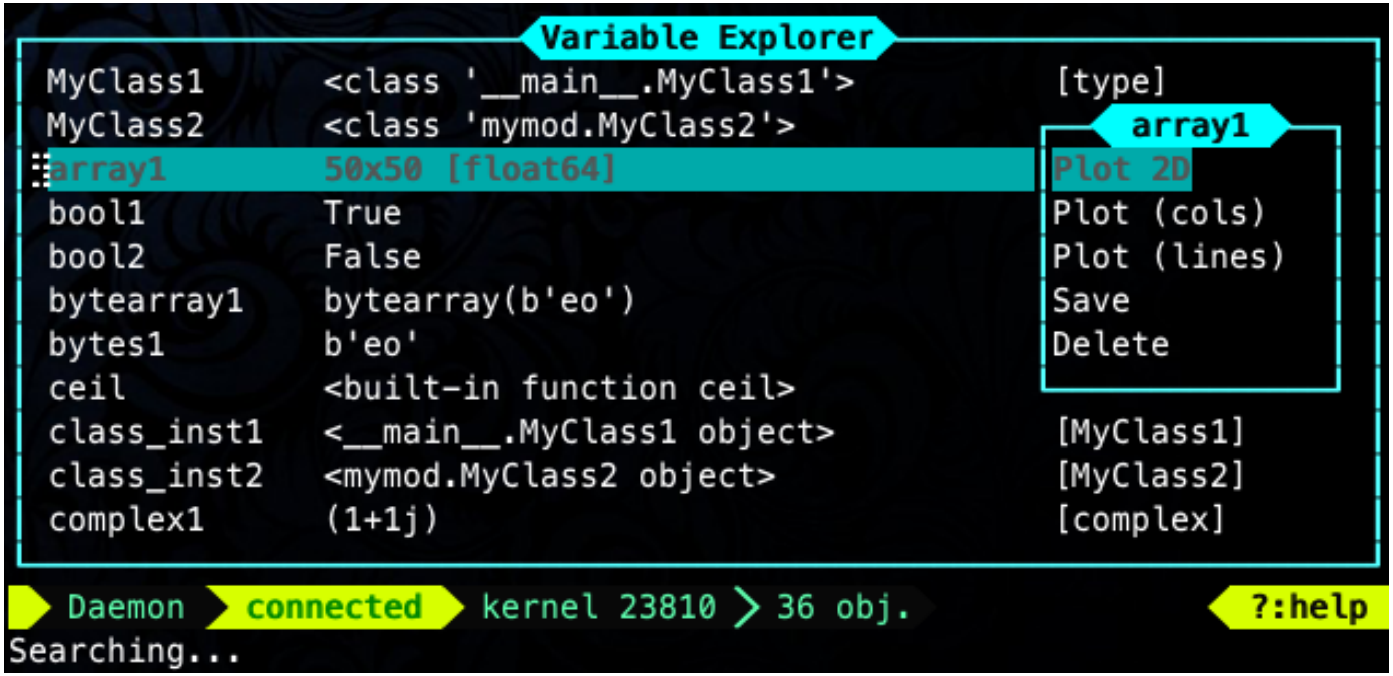
- `Tkinter` : *Preferred for small scale GUI applications*
  - Python interface to Tcl/Tk included in the **standard library**
  - Simplicity, abundance of resources, old and active community
  - Native look on all platforms
- `wxPython` : *Versatile toolkit*
  - Python binding for wxWidgets
  - wxWidgets are wrappers around the native widget of the host OS
  - Complete, often simple, good documentation and large community
- `PyQt` : *Probably preferred for open source applications*
  - The most popular Python binding for Qt toolkit
  - Does not use native objects of the OS, but create objects that mimic them
  - Licence restrictions : buy a licence to write proprietary applications
- `Kivy` : *Preferred for modern looking applications (See also Pygame/Pyglet)*
  - Python library for development of multi-touch applications
  - Compatible with window/Linux/OSX/Android/iOS
  - Support OpenGL, Actively developed by its community

Here are some of the main TUI toolkits :



- `curses`: *Low-level API to C ncurses library*
  - In **standard library**
  - For Unix systems only
- `console`: *Replacement for `curses` module for Windows*
  - In **standard library**
  - For Window only
- `npyscreen`: *Probably the best toolkit*
  - Large widget library
  - Multi platform
- `asciimatics`: *To make modern TUI*
  - Large widgets library & Animation effect capabilities
  - Multi platform
- `pygcurse`: *The way to go to make text games*
  - Curses wrapper based on `pygame`
  - Multi platform
- **Other Popular toolkits :**
  - `Urwid`
  - `prompt_toolkit`
  - `Unicurses`
  - `Blessings`
  - `Blessed`
  - `pyCDK`

A brief tour of curses module



Screenshot of cPyvke written in Python curses

Among others, the curses module provides:

- `initscr`: Initialize `curses`. Return a `window` object representing the whole screen.
- `newwin`: Return a new window with a given size
- `newpad`: Return a new pad (i.e. a window that can also be larger than the screen)
- `resizeterm`: Resize standard and current windows
- `wrapper`: Initialize curses and call a callable object (the *application*)

The wrapper method:

```
def wrapper(application, *args, **kwargs):
    try:
        stdscr = curses.initscr()  # Create the window
        curses.noecho()            # Turn off echoing of keys
        curses.cbreak()            # React to keys without pressing enter
        stdscr.keypad(True)        # Let curses handle special keys
        curses.start_color()       # Initialize 8 basic colors
        return application(*args, **kwargs)
    except:
        pass
    finally:                       # Whatever happens
        curses.nocbreak()          # Leave cbreak mode
        stdscr.keypad(False)       # Let system handles special keys
        curses.echo()              # Enter echo mode
        curses.endwin()            # Restore terminal to a sane state
```

Three sub-modules:

- `curses.ascii`: Utilities for working with ASCII characters
- `curses.panel`: A panel stack extension that adds depth to curses windows
- `curses.pad`: Editable text widget for curses supporting Emacs-like bindings

Main methods of `window` objects:

- `addstr`: Add a string area
- `getstr`: Read a bytes object from the user, with primitive line editing capacity.
- `border`: Draw borders around the object
- `getch/getkey`: Fetch user input. Return the ascci code/character
- `getmaxyx`: Return the size of the object as (y, x)
- `getyx`: Return current position of the cursor as (y, x)
- `hline/vline`: Display a horizontal/vertical line
- `move`: Move cursor
- `resize`: Resize curses window
- `subpab/subwin`: Return a sub-window
- `scroll`: Scroll the screen or a region
- `refresh`: Update the display - ***In curses, it is not refreshed automatically !***
- `erase`: Clear the window.

The Curses module defines several constants

- `KEY_*` Keys if `curses.keypad(True)` (`KEY_HOME`, `KEY_BACKSPACE`, `KEY_NPAGE`, ...)
- `A_*` Text attributes. The mains attributes are :

`curses.A_BLINK`	Blinking text
`curses.A_BOLD`	Extra bright or bold text
`curses.A_DIM`	Half bright text
`curses.A_REVERSE`	Reverse-video text
`curses.A_STANDOUT`	The best highlighting mode available
`curses.A_UNDERLINE`	Underlined text

- `ACS_*` Alternate Character Set (`ACS_PI`, `ACS_BULLET`, `ACS_DEGREE`, ...)
- `COLOR_*` Basic colors if `curses.start_color()` has been called:

❶ `COLOR_BLACK` ❷ `COLOR_RED` ❸ `COLOR_GREEN` ❹ `COLOR_YELLOW` ❺ `COLOR_BLUE` ❻ `COLOR_MAGENTA` ❼ `COLOR_CYAN` ❽ `COLOR_WHITE`

How to use colors ?

- `curses.init_pair()`: Define a pair of colors (foreground/background())
- `curses.color_pair()`: Use a pair of color defined with `init_pair()`

```
curses.init_pair(1, curses.COLOR_RED, curses.COLOR_WHITE)
stdscr.addstr(0,0, "RED ALERT!", curses.color_pair(1)|curses.A_BOLD)
stdscr.refresh()
```

Curses basic example

```
import curses

class App:
    def __init__(self, stdscr):
        self.stdscr = stdscr                    # The screen
        stdscr.clear()                          # Clear screen

    def run(self):
        k = ord(' ')                            # Init user input
        while k not in [ord('q'), ord('e')]:    # Main loop
            self.stdscr.border()                # Make borders
            self.stdscr.addstr(10, 10, 'Yeah')  # A string at (10, 10)
            k = self.stdscr.getch()             # Wait for user input
            self.stdscr.refresh()               # Refresh screen

def main(stdscr):                               # The main function
    app = App(stdscr)                           # App instance
    app.run()                                   # Run app

if __name__ == '__main__':
    curses.wrapper(main)                        # The wrapper
```

Tkinter basics

Philosophy: GUI in three steps

1. Creation of the main window, often called *root* window, as an instance of `Tk` class
2. Creation and display of all widgets { Creation with one of the widget method / Display with a layout manager
3. Enter the main event loop with `mainloop()` method

Rules:

- Any widget is contained in another, except for the root window
- Any widget takes the widget in which it is contained as first argument

Minimal example

```
import tkinter as tk

root = tk.Tk()      # Create the main window

label = tk.Label(root, text="What's up Doc !?")        # Create text area
quit = tk.Button(root, text="Quit", command=root.destroy) # Create button

label.pack()        # Display text on root
quit.pack()         # Display button on

root.mainloop()      # Tkinter loop
```

Tkinter provides 18 basic widgets and more:

	➤ `Menu`: Application menu	➤ `Canvas`: Shape drawing
Text	➤ `Label`: Single-line caption ➤ `Entry`: Single-line text input	➤ `Message`: Multiline text area ➤ `Text`: Multiline text input
Buttons	➤ `Button`: Simple button ➤ `Radiobutton`: One selection at a time ➤ `Checkbutton`: Checkboxes ➤ `Spinbox`: Select value	➤ `Menubutton`: Drop down button ➤ `Scale`: Slider widget ➤ `Listbox`: List of options ➤ `Scrollbar`: Add scrolling to a widget
Layout	➤ `Toplevel`: Separate window container ➤ `Frame`: Container widget to organize other widgets	➤ `PanedWindow`: Container widget for horizontally|vertically arranged panels ➤ `LabelFrame`: Same as `Frame` with title
Misc.	➤ `tkinter.messagebox`: msg or yes/no box ➤ `tkinter.simpledialog`: data entry	➤ `tkinter.colorchooser`: color dialog ➤ `tkinter.filedialog`: open file
ttk	➤ `ttk.ComboBox`: Drop down menu ➤ `ttk.Notebook`: Tabs ➤ `ttk.Progressbar`: Progress bar	➤ `ttk.Separator`: Horizontal|vertical bars ➤ `ttk.Sizegrip`: Resizing capabilities ➤ `ttk.Treeview`: Hierarchical collection
tix	➤ `tix.Balloon`: Ballon popup	➤ *40 more widgets...*

Standard option	Description
background	Background color
foreground	Text color
activebackground	Background color when widget has focus (only for buttons)
activeforeground	Text color when widget has focus (only for buttons)
selectbackground	Background color for the selected items
selectforeground	Foreground color for the selected items
highlightbackground	Background color of the highlight region when widget has focus
highlightcolor	Foreground color of the highlight region when widget has focus
disabledforeground	Foreground color when the widget is disabled
padx	Horizontal margin.
pady	Vertical margin
width	Widget width in font size
height	Widget heigh in font size
borderwidth	Width of the border
highlightthickness	Width of the highlight rectangle when the widget has focus
selectborderwidth	Width of the three-dimentional border around selected items
wraplength	Maximum line length for widgets that perform word wrapping
underline	Index of the character to underline in the widget's text
font	Font description as a tuple `("Times", "16", "bold italic")`
relief	Relief style can be `FLAT`, `RAISED`, `SUNKEN`, `GROOVE`, `RIDGE`
anchor	Position : `NW`, `N`, `NE`, `E`, `SE`, `S`, `SW`, `W`, `CENTER`

snow	deep sky blue	gold	seashell3	SlateBlue2	LightBlue3	SpringGreen2	DarkGoldenrod1	brown4	pink3	purple1	gray26	gray64
ghost white	sky blue	light goldenrod	seashell4	SlateBlue3	LightBlue4	SpringGreen3	DarkGoldenrod2	salmon1	pink4	purple2	gray27	gray65
white smoke	light sky blue	goldenrod	AntiqueWhite1	SlateBlue4	LightCyan2	SpringGreen4	DarkGoldenrod3	salmon2	LightPink1	purple3	gray28	gray66
gainsboro	steel blue	dark goldenrod	AntiqueWhite2	RoyalBlue1	LightCyan3	green2	DarkGoldenrod4	salmon3	LightPink2	purple4	gray29	gray67
floral white	light steel blue	rosy brown	AntiqueWhite3	RoyalBlue2	LightCyan4	green3	RosyBrown1	salmon4	LightPink3	MediumPurple1	gray30	gray68
old lace	light blue	indian red	AntiqueWhite4	RoyalBlue3	PaleTurquoise1	green4	RosyBrown2	LightSalmon2	LightPink4	MediumPurple2	gray31	gray69
linen	powder blue	saddle brown	bisque2	RoyalBlue4	PaleTurquoise2	chartreuse2	RosyBrown3	LightSalmon3	PaleVioletRed1	MediumPurple3	gray32	gray70
antique white	pale turquoise	sandy brown	bisque3	blue2	PaleTurquoise3	chartreuse3	RosyBrown4	LightSalmon4	PaleVioletRed2	MediumPurple4	gray33	gray71
papaya whip	dark turquoise	dark salmon	bisque4	blue4	PaleTurquoise4	chartreuse4	IndianRed1	orange2	PaleVioletRed3	thistle1	gray34	gray72
blanched almond	medium turquoise	salmon	PeachPuff2	DodgerBlue2	CadetBlue1	OliveDrab1	IndianRed2	orange3	PaleVioletRed4	thistle2	gray35	gray73
bisque	turquoise	light salmon	PeachPuff3	DodgerBlue3	CadetBlue2	OliveDrab2	IndianRed3	orange4	maroon1	thistle3	gray36	gray74
peach puff	cyan	orange	PeachPuff4	DodgerBlue4	CadetBlue3	OliveDrab4	IndianRed4	DarkOrange1	maroon2	thistle4	gray37	gray75
navajo white	light cyan	dark orange	NavajoWhite2	SteelBlue1	CadetBlue4	DarkOliveGreen1	sienna1	DarkOrange2	maroon3		gray38	gray76
lemon chiffon	cadet blue	coral	NavajoWhite3	SteelBlue2	turquoise1	DarkOliveGreen2	sienna2	DarkOrange3	maroon4		gray39	gray77
mint cream	medium aquamarine	light coral	NavajoWhite4	SteelBlue3	turquoise2	DarkOliveGreen3	sienna3	DarkOrange4	VioletRed1		gray40	gray78
azure	aquamarine	tomato	LemonChiffon2	SteelBlue4	turquoise3	DarkOliveGreen4	sienna4	coral1	VioletRed2		gray42	gray79
alice blue	dark green	orange red	LemonChiffon3	DeepSkyBlue2	turquoise4	khaki1	burlywood1	coral2	VioletRed3		gray43	gray80
lavender	dark olive green	red	LemonChiffon4	DeepSkyBlue3	cyan2	khaki2	burlywood2	coral3	VioletRed4		gray44	gray81
lavender blush	dark sea green	hot pink	cornsilk2	DeepSkyBlue4	cyan3	khaki3	burlywood3	coral4	magenta2		gray45	gray82
misty rose	sea green	deep pink	cornsilk3	SkyBlue1	cyan4	khaki4	burlywood4	tomato2	magenta3		gray46	gray83
dark slate gray	medium sea green	pink	cornsilk4	SkyBlue2	DarkSlateGray1	LightGoldenrod1	wheat1	tomato3	magenta4		gray47	gray84
dim gray	light sea green	light pink	ivory2	SkyBlue3	DarkSlateGray2	LightGoldenrod2	wheat2	tomato4	orchid1		gray48	gray85
slate gray	pale green	pale violet red	ivory3	SkyBlue4	DarkSlateGray3	LightGoldenrod3	wheat3	OrangeRed2	orchid2		gray49	gray86
light slate gray	spring green	maroon	ivory4	LightSkyBlue1	DarkSlateGray4	LightGoldenrod4	wheat4	OrangeRed3	orchid3		gray50	gray87
gray	lawn green	medium violet red	honeydew2	LightSkyBlue2	aquamarine2	LightYellow2	tan1	OrangeRed4	orchid4		gray51	gray88
light grey	medium spring green	violet red	honeydew3	LightSkyBlue3	aquamarine4	LightYellow3	tan2	red2	plum1		gray52	gray89
midnight blue	green yellow	medium orchid	honeydew4	LightSkyBlue4	DarkSeaGreen1	LightYellow4	tan4	red3	plum2		gray53	gray90
navy	lime green	dark orchid	LavenderBlush2	SlateGray1	DarkSeaGreen2	yellow2	chocolate1	red4	plum3		gray54	gray91
cornflower blue	yellow green	dark violet	LavenderBlush3	SlateGray2	DarkSeaGreen3	yellow3	chocolate2	DeepPink2	plum4		gray55	gray92
dark slate blue	forest green	blue violet	LavenderBlush4	SlateGray3	DarkSeaGreen4	yellow4	chocolate3	DeepPink3	MediumOrchid1		gray56	gray93
slate blue	olive drab	purple	MistyRose2	SlateGray4	SeaGreen1	gold2	firebrick1	DeepPink4	MediumOrchid2		gray57	gray94
medium slate blue	dark khaki	medium purple	MistyRose3	LightSteelBlue1	SeaGreen2	gold3	firebrick2	HotPink1	MediumOrchid3		gray58	gray95
light slate blue	khaki	thistle	MistyRose4	LightSteelBlue2	SeaGreen3	gold4	firebrick3	HotPink2	MediumOrchid4		gray59	gray97
medium blue	pale goldenrod	snow2	azure2	LightSteelBlue3	PaleGreen1	goldenrod1	firebrick4	HotPink3	DarkOrchid1		gray60	gray98
royal blue	light goldenrod yellow	snow3	azure3	LightSteelBlue4	PaleGreen2	goldenrod2	brown1	HotPink4	DarkOrchid2		gray61	gray99
blue	light yellow	snow4	azure4	LightBlue1	PaleGreen3	goldenrod3	brown2	pink1	DarkOrchid3		gray62	
dodger blue	yellow	seashell2	SlateBlue1	LightBlue2	PaleGreen4	goldenrod4	brown3	pink2	DarkOrchid4		gray63	

From science.smith.edu

Function of the layout managers:

- Arange widgets on the window
- Register widgets with the underlying windowing system
- Manage the display of widgets on the screen

Arranging widgets on the screen includes determining the size and position of components. Widgets can provide size and alignment information to geometry managers, but the geometry managers has always the final say on the positioning and sizing.

There are 3 built-in layout managers[1]:

- `pack()`: Relative positioning - *Easiest to use but limited*
- `grid()`: Positioning of the widgets in a 2D grid - *Good control of positioning*
- `place()`: Absolute (or relative) positioning - *Hard to use in practice but **absolute** !*

```
import tkinter as tk
root = tk.Tk()

orange = tk.Label(root, text="Orange", bg="Orange")
khaki2 = tk.Label(root, text="Khaki2", bg="Khaki2")
coral2 = tk.Label(root, text="Coral2", bg="Coral2")
```

[1] The three layout managers should never be mixed in the same container!

```
pack(anchor, side, fill, expand, padx, pady, ipdax, ipady,...)
```

```
orange.pack()
khaki2.pack()
coral2.pack()
```


```
orange.pack()                          # Default is CENTER
khaki2.pack(anchor='w')                # NW, N, NE, W,
coral2.pack(anchor='e')                # E, SW, S, SE
```


```
orange.pack(fill=tk.X)                 # fill takes X, Y
khaki2.pack(fill=tk.X)                 # or BOTH
coral2.pack(fill=tk.X)
```


```
orange.pack(side=tk.LEFT, fill=tk.Y)   # side takes TOP,
khaki2.pack(side=tk.LEFT, fill=tk.Y)   # BOTTOM,
coral2.pack(side=tk.LEFT, fill=tk.Y)   # LEFT, RIGHT
```


```
root.geometry('50x80+200+200')         # WxH+X+Y
orange.pack(expand=1, fill=tk.Y)
coral2.pack(fill=tk.BOTH)
```


```
root.geometry('50x80+200+200')
orange.pack(expand=1, fill=tk.Y)
coral2.pack(expand=1, fill=tk.BOTH)
```

```
grid(row, column, rowspan, columnspan, padx, pady, ipdax, ipady, sticky)
```

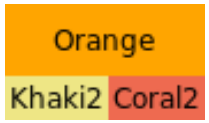
```
orange.grid()
khaki2.grid()
coral2.grid()
```


```
orange.grid()
khaki2.grid(sticky='w')
coral2.grid(sticky='e')
```


```
orange.grid(sticky='we')
khaki2.grid(sticky='we')
coral2.grid(sticky='we')
```


```
orange.grid(row=0, column=0, sticky='ns')
khaki2.grid(row=0, column=1, sticky='ns')
coral2.grid(row=0, column=2, sticky='ns')
```


```
orange.grid(row=0, column=0, columnspan=2, sticky='news')
khaki2.grid(row=1, column=0)
coral2.grid(row=1, column=1)
```

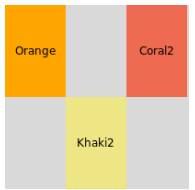



```
# Grid has been customized in these examples :
root.columnconfigure(0, weight=1)
root.rowconfigure(0, weight=1)
```

```
place(anchor, height, width, relheight, relwidth, relx, rely, x, y, ...)
```

```
Orange.place(relheight=1, relwidth=0.33)
Khaki2.place(relheight=1, relwidth=0.33, relx=0.33)
Coral2.place(relheight=1, relwidth=0.33, relx=0.66)
```


```
Orange.place(relheight=0.5, relwidth=0.33)
Khaki2.place(relheight=0.5, relwidth=0.33, relx=0.33, rely
    =0.5)
Coral2.place(relheight=0.5, relwidth=0.33, relx=0.66)
```



Bind command to a widget

- Some widgets have a `command` *kwarg* that has to be a function (called without `()`)
- Use of widget methods or custom functions. To list methods of a widget : `dir(root)`

```
def action():
    print('Button pressed')

action_button = Button(root, text="Action", command=action)
action_quit = Button(root, text="Quit", command=root.quit)
```

Bind events to a widget

- Syntax: `widget.bind('<event_seq>', function)`
- `bind` calls `function` with an argument of type `Event` when `event_seq` is triggered
- Syntax of `event_seq`: `help(tkinter.Label.bind)`
- List of `Event` instance attributes: `help(tkinter.Event)`

```
def disp(event):
    print(event.keysym)     # keysym is the keycode of the pressed key

root = tk.Tk()
root.bind('<Key>', disp)    # <Key> is for all keys from keyboard
tk.mainloop()               # <Key-a> for 'a', <button-1> for right-click
```

Put the code into a class to create a custom widget

```
import tkinter

class Application(tk.Frame):              # Inherits from Frame

    def __init__(self, root=None):       # Application is a widget itself
        super().__init__(root)           # call init of the parent
        self.root = root                 #
        self.pack()                      # Display the frame
        self.create_widgets()            # Setup and display widgets


    def create_widgets(self):
        self.label = tk.Label(self, text="What's up Doc !?")
        self.quit = tk.Button(self, text="Quit",
                              command=self.root.destroy)
        self.label.pack(side="top")
        self.quit.pack(side="bottom")


if __name__ == "__main__":
    root = tk.Tk()
    app = Application(root=root)
    app.mainloop()
```


Outlines

UI programming is a vast topic !

Impossible to cover all functionalities of all UI dedicated modules !

This presentation only gives some keys to start UI programming with Python

Bibliography

- `Tkinter`
  - python.org
  - openclassrooms.com
  - effbot.org
  - nmt.edu (New Mexico Tech)
  - developpez.com
  - zetcode.com
  - xavierdupre.fr
- `wxpython`
  - wxpython.org
  - zetcode.com
  - tutorialspoint.com
- `kivy`
  - kivy.org
  - newthinktank.com
- `curses`
  - python.org