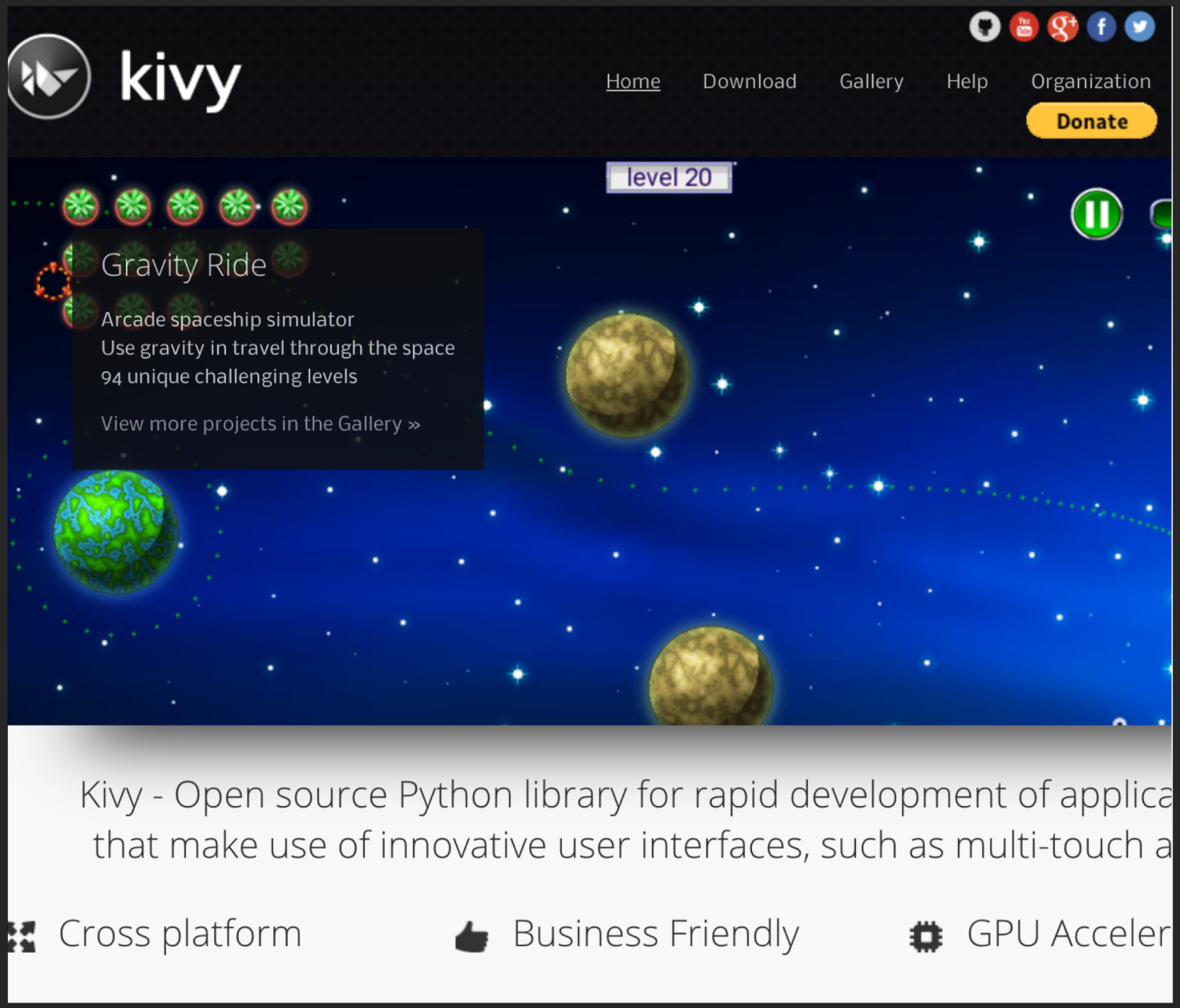
User Interface Design and Evaluation

PYTHON UI DEVELOPMENT



https://kivy.org/#home

FrontPage
RecentChanges
FindPage
HelpContents
PyQt

Page
- Immutable Page
- Info
- Attachments
- More Actions:

User
- Login

About PyQt

PyQt is one of the most popular Python bindings for the Qt cross-platform C++ framework. PyQt developed by Riverbank Computing Limited. Qt itself is developed as part of the Qt Project. PyQt provides bindings for Qt 4 and Qt 5. PyQt is distributed under a choice of licences: GPL version 3 or a commercial license.

PyQt is available in two editions: PyQt4 which will build against Qt 4.x and 5.x and PyQt5 which will only build against 5.x. Both editions can be built for Python 2 and 3. PyQt contains over 620 classes that cover graphical user interfaces, XML handling, network communication, SQL databases, Web browsing and other technologies available in Qt.

The latest iteration of PyQt is v5.11.3. It fully supports Qt 5.11.2.

PyQt4 runs on Windows, Linux, Mac OS X and various UNIX platforms. PyQt5 also runs on Android and iOS.

PyQt Documentation

Current documentation is available for PyQt4 and PyQt5.

A collection of links to books can be found on the Books page.

- Mark Summerfield's book, **Rapid GUI Programming with Python and Qt**, is a guide to GUI application development with Python 2.5, PyQt4 and Qt 4.2/4.3. More information can be found at http://www.qtrac.eu/pyqtbook.html. Mark recommends, incidentally, **GUI Bloopers**, as appropriate supplementary reading for PyQt programmers.
- A list of tutorials can also be found on the Tutorials page.
  - **A recent PyQt5 tutorial** is available at https://build-system.fman.io/pyqt5-tutorial.

On this Wiki, you can find the following tutorials:

- A tutorial presented by Jonathan Gardner at the 2003 Northwest Linux Fest is available at JonathanGardnerPyQtTutorial.
- A tutorial presented by Oleksandr Yakovlyev for embedding PyQt in C++/Qt application EmbedingPyQtTutorial

Developing with PyQt and PyKDE

- Tutorials contains a list of tutorials and walkthroughs
- Books contains a list of books about Qt, PyQt, KDE and PyKDE
- Development With PyQt can be made even easier with some extra tools and information
- Sample Code lists some pieces of code to solve some common programming problems
- Overviews and Guides provides in-depth information and detailed examples
- Docs And Howtos contains links to API documentation and articles about developing with PyQt and PyKDE
- Some Existing Applications written with PyQt and PyKDE
- Third Party Packages and Modules for use with PyQt and PyKDE
- GUI Testing
- Videos about PyQt on various video sites

https://wiki.python.org/moin/PyQt

https://wiki.python.org/moin/TkInter

- Introduction to Python GUI
- Adding a label to the GUI form
- Creating buttons and text box widgets
- Exploring widgets
- Adding extra features
- Adding several widgets in a loop

Python UI Development

- Creating our first GUI
- How to prevent it from being resized

Tkinter

https://docs.python.org/3/library/tkinter.html

IDLE

- Is enough to start
- Created with Tkinter !

```python
import tkinter as tk

# Create instance
win = tk.Tk()

# Add a title
win.title("Python GUI")

#=====================
# Start GUI
#=====================
win.mainloop()
```

Import and alias as tk

Call constructor of the class and () turn it into an instance.

We assign class instance to variable “win”
And give it a title via the title property

Python infers the type from the assignment
Every variable always has a type

```python
import tkinter as tk

# Create instance
win = tk.Tk()

# Add a title
win.title("Python GUI")

# Disable resizing the GUI
win.resizable(0,0)

#======================
# Start GUI
#======================
win.mainloop()
```

Set attributes to 0
Prevents resizing

Method of tk class

Try it !

Notice something?

```python
import tkinter as tk

# Create instance
win = tk.Tk()

# Add a title
win.title("Python GUI")

# Disable resizing the GUI
win.resizable(0,0)

#======================
# Start GUI
#======================
win.mainloop()
```

(x, y) can be hard coded
Try it !

Why is this important?

Because once we resize our UI
And add widgets to the UI it might look ugly!

- Adding a Label widget to our GUI form

```python
import tkinter as tk
from tkinter import ttk

# Create instance
win = tk.Tk()

# Add a title
win.title("Python GUI")

# Disable resizing the GUI
#win.resizable(0,0)

# Adding a Label
ttk.Label(win, text="A Label").grid(column=0, row=0)

#======================
# Start GUI
#======================
win.mainloop()
```

To add labels

ttk module has some advanced widgets

Could be seen as an extension

To add labels:

We pass our win tk Instance into the ttk.label Constructor and call the Method grid to specify Where

Grid is a layout manager

Try it !

Why did it become so small?

Adding a widget overrides the default layout!

Causes optimization! As little space as possible

Try resizing it!

What happens?

- Adding buttons and text boxes

```python
import tkinter as tk
from tkinter import ttk

# Create instance
win = tk.Tk()

# Add a title
win.title("Python GUI")

# Disable resizing the GUI
#win.resizable(0,0)

# Modify adding a Label
aLabel = ttk.Label(win, text="A Label")
aLabel.grid(column=0, row=0)

# Button Click Event Function
def clickMe():
    action.configure(text="** I have been Clicked! **")
    aLabel.configure(foreground='red')
    aLabel.configure(text='A Red Label')

# Adding a Button
action = ttk.Button(win, text="Click Me!", command=clickMe)
action.grid(column=1, row=0)


#======================
# Start GUI
#======================
win.mainloop()
```

Here we add button onclick() action;:
Update label
Update text of button

Assign label to variable
Use grid manager to position label

Label accessible as declared above
Function that calls it

Clickme is eventhandler – gets invoked
Once clicked

Create button & bind command to clickme

Aligns both label and button

Creating Textbox Widgets

```python
import tkinter as tk
from tkinter import ttk

# Create instance
win = tk.Tk()

# Add a title
win.title("Python GUI")

# Disable resizing the GUI
#win.resizable(0,0)

# Modify adding a Label
aLabel = ttk.Label(win, text="A Label")
aLabel.grid(column=0, row=0)

#Modified Button Click Function
def clickMe():
    action.configure(text='Hello ' + name.get())

# Changing our Label
ttk.Label(win, text="Enter a name:").grid(column=0, row=0)

# Adding a Textbox Entry widget
name = tk.StringVar()
nameEntered = ttk.Entry(win, width=12, textvariable=name)
nameEntered.grid(column=0, row=1)

# Adding a Button
action = ttk.Button(win, text="Click Me!", command=clickMe)
action.grid(column=1, row=1)

#======================
# Start GUI
#======================
win.mainloop()
```

In tkinter textboxwiget is called entry

Without oop callback function
Works!

More meaningful name (text)

Tkinter NOT python!

Creating variable name
Is bound to entry

Try resizing it!

What happens?

Hardcoded width 12!

- Setting the focus to a widget
- Disabling widgets
- Adding drop-down combo boxes with initial default values

Would be nicer to make cursor
Appear once widget appears?

```python
import tkinter as tk
from tkinter import ttk

# Create instance
win = tk.Tk()

# Add a title
win.title("Python GUI")

# Disable resizing the GUI
#win.resizable(0,0)

# Modify adding a Label
aLabel = ttk.Label(win, text="A Label")
aLabel.grid(column=0, row=0)

#Modified Button Click Function
def clickMe():
    action.configure(text='Hello ' + name.get())

# Changing our Label
ttk.Label(win, text="Enter a name:").grid(column=0, row=0)

# Adding a Textbox Entry widget
name = tk.StringVar()
nameEntered = ttk.Entry(win, width=12, textvariable=name)
nameEntered.grid(column=0, row=1)

# Adding a Button
action = ttk.Button(win, text="Click Me!", command=clickMe)
action.grid(column=1, row=1)
action.configure(state='disabled')    # Disable the Button Widget

nameEntered.focus()      # Place cursor into name Entry
#======================
# Start GUI
#======================
win.mainloop()
```

All we have to do to set focus to Specific control when the GUI appears is to call the focus() method

Assign ttk entry to variable

That's it …focus!

No OOP so have to declare above main loop!

On mac – maybe set focus on GUI win first

Try it!

What happens?!

ComboBox Widget

```python
import tkinter as tk
from tkinter import ttk

# Create instance
win = tk.Tk()

# Add a title
win.title("Python GUI")

# Disable resizing the GUI
#win.resizable(0,0)

# Modify adding a Label
aLabel = ttk.Label(win, text="A Label")
aLabel.grid(column=0, row=0)

#Modified Button Click Function
def clickMe():
    action.configure(text='Hello ' + name.get())

# Changing our Label
ttk.Label(win, text="Enter a name:").grid(column=0, row=0)

# Adding a Textbox Entry widget
name = tk.StringVar()
nameEntered = ttk.Entry(win, width=12, textvariable=name)
nameEntered.grid(column=0, row=1)

# Adding a Button
action = ttk.Button(win, text="Click Me!", command=clickMe)
action.grid(column=2, row=1)
#action.configure(state='disabled')    # Disable the Button Widget

ttk.Label(win, text="Choose a number:").grid(column=1, row=0)
number = tk.StringVar()
numberChosen = ttk.Combobox(win, width=12, textvariable=number)
numberChosen['values'] = (1, 2, 4, 42, 100)
numberChosen.grid(column=1, row=1)
numberChosen.current(0)


nameEntered.focus()      # Place cursor into name Entry
#======================
# Start GUI
#======================
win.mainloop()
```

While we can restrict user to only certain Choices – we can allow the user to type in whatever they wish

Inserting another column between entry Widget and button using the grid layout manager

Here is the code for a combo box

Add state='readonly'

Try it!

What happens?!

```python
import tkinter as tk
from tkinter import ttk

# Create instance
win = tk.Tk()

# Add a title
win.title("Python GUI")

# Disable resizing the GUI
#win.resizable(0,0)

# Modify adding a Label
aLabel = ttk.Label(win, text="A Label")
aLabel.grid(column=0, row=0)

# Modified Button Click Function
def clickMe():
    action.configure(text='Hello ' + name.get() + ' '+ numberChosen.get())

# Changing our Label
ttk.Label(win, text="Enter a name:").grid(column=0, row=0)

# Adding a Textbox Entry widget
name = tk.StringVar()
nameEntered = ttk.Entry(win, width=12, textvariable=name)
nameEntered.grid(column=0, row=1)

# Adding a Button
action = ttk.Button(win, text="Click Me!", command=clickMe)
action.grid(column=2, row=1)
#action.configure(state='disabled')    # Disable the Button Widget

ttk.Label(win, text="Choose a number:").grid(column=1, row=0)
number = tk.StringVar()
numberChosen = ttk.Combobox(win, width=12, textvariable=number, state='readonly')
numberChosen['values'] = (1, 2, 4, 42, 100)
numberChosen.grid(column=1, row=1)
numberChosen.current(0)


nameEntered.focus()      # Place cursor into name Entry
#======================
# Start GUI
#======================
win.mainloop()
```

- Adding three Checkbutton widgets
- Creating three Radiobutton widgets
- Adding scrolled text widgets

```python
import tkinter as tk
from tkinter import ttk

# Create instance
win = tk.Tk()

# Add a title
win.title("Python GUI")

# Disable resizing the GUI
#win.resizable(0,0)

# Modify adding a Label
aLabel = ttk.Label(win, text="A Label")
aLabel.grid(column=0, row=0)

#Modified Button Click Function
def clickMe():
    action.configure(text='Hello ' + name.get())

# Changing our Label
ttk.Label(win, text="Enter a name:").grid(column=0, row=0)

# Adding a Textbox Entry widget
name = tk.StringVar()
nameEntered = ttk.Entry(win, width=12, textvariable=name)
nameEntered.grid(column=0, row=1)
```

```python
# Adding a Button
action = ttk.Button(win, text="Click Me!", command=clickMe)
action.grid(column=2, row=1)
#action.configure(state='disabled')    # Disable the Button Widget

ttk.Label(win, text="Choose a number:").grid(column=1, row=0)
number = tk.StringVar()
numberChosen = ttk.Combobox(win, width=12, textvariable=number)
numberChosen['values'] = (1, 2, 4, 42, 100)
numberChosen.grid(column=1, row=1)
numberChosen.current(0)

# Creating three checkbuttons
chVarDis = tk.IntVar()
check1 = tk.Checkbutton(win, text="Disabled", variable=chVarDis, state='disabled')
check1.select()
check1.grid(column=0, row=4, sticky=tk.W)

chVarUn = tk.IntVar()
check2 = tk.Checkbutton(win, text="UnChecked", variable=chVarUn)
check2.deselect()
check2.grid(column=1, row=4, sticky=tk.W)

chVarEn = tk.IntVar()
check3 = tk.Checkbutton(win, text="Enabled", variable=chVarEn)
check3.select()
check3.grid(column=2, row=4, sticky=tk.W)

# Place cursor into name Entry
nameEntered.focus()
#======================
# Start GUI
#======================
win.mainloop()
```

First is disabled and has a Check mark in it.
User can not modify it

Type of the variable is Tkinter integer (0/1)

means widget will be aligned to The west of the grid

Test it and resize – where will it go?

Radio Button Widget

```python
import tkinter as tk
from tkinter import ttk
from tkinter import scrolledtext

# Create instance
win = tk.Tk()

# Add a title
win.title("Python GUI")

# Disable resizing the GUI
#win.resizable(0,0)

# Modify adding a Label
aLabel = ttk.Label(win, text="A Label")
aLabel.grid(column=0, row=0)

#Modified Button Click Function
def clickMe():
    action.configure(text='Hello ' + name.get())

# Changing our Label
ttk.Label(win, text="Enter a name:").grid(column=0, row=0)

# Adding a Textbox Entry widget
name = tk.StringVar()
nameEntered = ttk.Entry(win, width=12, textvariable=name)
nameEntered.grid(column=0, row=1)

# Adding a Button
action = ttk.Button(win, text="Click Me!", command=clickMe)
action.grid(column=2, row=1)
#action.configure(state='disabled')    # Disable the Button Widget
```

```python
# Adding a Button
action = ttk.Button(win, text="Click Me!", command=clickMe)
action.grid(column=2, row=1)
#action.configure(state='disabled')    # Disable the Button Widget

ttk.Label(win, text="Choose a number:").grid(column=1, row=0)
number = tk.StringVar()
numberChosen = ttk.Combobox(win, width=12, textvariable=number)
numberChosen['values'] = (1, 2, 4, 42, 100)
numberChosen.grid(column=1, row=1)
numberChosen.current(0)

# Creating three checkbuttons
chVarDis = tk.IntVar()
check1 = tk.Checkbutton(win, text="Disabled", variable=chVarDis, state='disabled')
check1.select()
check1.grid(column=0, row=4, sticky=tk.W, columnspan=3)

chVarUn = tk.IntVar()
check2 = tk.Checkbutton(win, text="UnChecked", variable=chVarUn)
check2.deselect()
check2.grid(column=1, row=4, sticky=tk.W, columnspan=3)

chVarEn = tk.IntVar()
check3 = tk.Checkbutton(win, text="Enabled", variable=chVarEn)
check3.deselect()
check3.grid(column=2, row=4, sticky=tk.W, columnspan=3)

# GUI Callback function
def checkCallback(*ignoredArgs):
    # only enable one checkbutton
    if chVarUn.get(): check3.configure(state='disabled')
    else:       check3.configure(state='normal')
    if chVarEn.get(): check2.configure(state='disabled')
    else:       check2.configure(state='normal')
```

```python
# trace the state of the two checkbuttons
chVarUn.trace('w', lambda unused0, unused1, unused2 : checkCallback())
chVarEn.trace('w', lambda unused0, unused1, unused2 : checkCallback())

# Radiobutton Globals
COLOR1 = "Blue"
COLOR2 = "Gold"
COLOR3 = "Red"

# Radiobutton Callback
def radCall():
    radSel=radVar.get()
    if   radSel == 1: win.configure(background=COLOR1)
    elif radSel == 2: win.configure(background=COLOR2)
    elif radSel == 3: win.configure(background=COLOR3)

# create three Radiobuttons using one variable
radVar = tk.IntVar()

rad1 = tk.Radiobutton(win, text=COLOR1, variable=radVar, value=1, command=radCall)
rad1.grid(column=0, row=5, sticky=tk.W, columnspan=3)

rad2 = tk.Radiobutton(win, text=COLOR2, variable=radVar, value=2, command=radCall)
rad2.grid(column=1, row=5, sticky=tk.W, columnspan=3)

rad3 = tk.Radiobutton(win, text=COLOR3, variable=radVar, value=3, command=radCall)
rad3.grid(column=2, row=5, sticky=tk.W, columnspan=3)

# Place cursor into name Entry
nameEntered.focus()
#======================
# Start GUI
#======================
win.mainloop()
```

Create global variables

DRY principle
Don’t
Repeat
Yourself!
OOP concept

Creating only
One variable
To be used by three
RadioButtons

Try it!
Beautiful or ugly?

TCL manual page for colorschemes

ScrollText Widget

Much larger than simple Entry Widgets
Span multiple times
Like notepad
Automatically enabling scrollbars

```python
# GUI Callback function
def checkCallback(*ignoredArgs):
    # only enable one checkbutton
    if chVarUn.get(): check3.configure(state='disabled')
    else:       check3.configure(state='normal')
    if chVarEn.get(): check2.configure(state='disabled')
    else:       check2.configure(state='normal')

# trace the state of the two checkbuttons
chVarUn.trace('w', lambda unused0, unused1, unused2 : checkCallback())
chVarEn.trace('w', lambda unused0, unused1, unused2 : checkCallback())

# Radiobutton Globals
COLOR1 = "Blue"
COLOR2 = "Gold"
COLOR3 = "Red"

# Radiobutton Callback
def radCall():
    radSel=radVar.get()
    if   radSel == 1: win.configure(background=COLOR1)
    elif radSel == 2: win.configure(background=COLOR2)
    elif radSel == 3: win.configure(background=COLOR3)

# create three Radiobuttons using one variable
radVar = tk.IntVar()

rad1 = tk.Radiobutton(win, text=COLOR1, variable=radVar, value=1, command=radCall)
rad1.grid(column=0, row=5, sticky=tk.W)

rad2 = tk.Radiobutton(win, text=COLOR2, variable=radVar, value=2, command=radCall)
rad2.grid(column=1, row=5, sticky=tk.W)

rad3 = tk.Radiobutton(win, text=COLOR3, variable=radVar, value=3, command=radCall)
rad3.grid(column=2, row=5, sticky=tk.W)

# Using a scrolled Text control
scrolW  = 30
scrolH  =  3
scr = scrolledtext.ScrolledText(win, width=scrolW, height=scrolH, wrap=tk.WORD)
scr.grid(column=0, columnspan=3)

# Place cursor into name Entry
nameEntered.focus()
#======================
# Start GUI
#======================
win.mainloop()
```

Import module that contains Scrolltext widget class

Define height and width
Hardcoded passing into scrolltext Constructor
MagicNumbers!

Setting property on widget by passing wrap (break lines by words) default tk.CHAR
See what happens!

Span all three columns

- Refactoring the code

So far copy paste the same code a lot
For example for RadioButtons etc

```python
# GUI Callback function
def checkCallback(*ignoredArgs):
    # only enable one checkbutton
    if chVarUn.get(): check3.configure(state='disabled')
    else:       check3.configure(state='normal')
    if chVarEn.get(): check2.configure(state='disabled')
    else:       check2.configure(state='normal')

# trace the state of the two checkbuttons
chVarUn.trace('w', lambda unused0, unused1, unused2 : checkCallback())
chVarEn.trace('w', lambda unused0, unused1, unused2 : checkCallback())


# Using a scrolled Text control
scrolW  = 30
scrolH  =  3
scr = scrolledtext.ScrolledText(win, width=scrolW, height=scrolH, wrap=tk.WORD)
scr.grid(column=0, sticky='WE', columnspan=3)

# First, we change our Radiobutton global variables into a list.
colors = ["Blue", "Gold", "Red"]

# We have also changed the callback function to be zero-based, using the list instead of module-level
# Radiobutton callback function
def radCall():
    radSel=radVar.get()
    if   radSel == 0: win.configure(background=colors[0])
    elif radSel == 1: win.configure(background=colors[1])
    elif radSel == 2: win.configure(background=colors[2])

radVar = tk.IntVar()

# Next we are selecting a non-existing index value for radVar.
radVar.set(99)

# Now we are creating all three Radiobutton widgets within one loop.
for col in range(3):
    curRad = 'rad' + str(col)
    curRad = tk.Radiobutton(win, text=colors[col], variable=radVar, value=col, command=radCall)
    curRad.grid(column=col, row=6, sticky=tk.W)


# Place cursor into name Entry
nameEntered.focus()
#======================
# Start GUI
#======================
win.mainloop()
```

Setting default value
So value outside of trigger Values – so nothing is selected

Replacement for single statements
Concise
Imagine 100 radiobuttons!
Just change range(x) !

Try it! Now same outcome but
Code cleaner and easier to maintain!

Layout Management

- Arranging several labels within a label frame
- Using padding to add space around widgets
- Expanding the GUI dynamically using widgets
- Aligning the GUI widgets by embedding frames within frames

How do we arrange widgets within widgets?
Let’s make the first steps for a great UIs

- Creating menu bars
- Creating tabbed widgets
- Using the grid layout manager

- The label frame widget
- Playing around with the row and column variables

Label frame widgets give us much more control about our layout of our UI design

```python
scrolW  = 30
scrolH  =  3
scr = scrolledtext.ScrolledText(win, width=scrolW, height=scrolH, wrap=tk.WORD)
scr.grid(column=0, row=5, sticky='WE', columnspan=3)
# scr.grid(column=0, row=5, columnspan=3)
# scr.grid(column=0, row=5)

# First, we change our Radiobutton global variables into a list.
colors = ["Blue", "Gold", "Red"]

# We have also changed the callback function to be zero-based, using the list instead of module-level global
# Radiobutton callback function
def radCall():
    radSel=radVar.get()
    if   radSel == 0: win.configure(background=colors[0])
    elif radSel == 1: win.configure(background=colors[1])
    elif radSel == 2: win.configure(background=colors[2])

radVar = tk.IntVar()

# Next we are selecting a non-existing index value for radVar.
radVar.set(99)

# Now we are creating all three Radiobutton widgets within one loop.
for col in range(3):
    curRad = 'rad' + str(col)
    curRad = tk.Radiobutton(win, text=colors[col], variable=radVar, value=col, command=radCall)
    curRad.grid(column=col, row=6, sticky=tk.W, columnspan=3)


# Create a container to hold labels
labelsFrame = ttk.LabelFrame(win, text=' Labels in a Frame ')
labelsFrame.grid(column=0, row=7)

# Place labels into the container element
ttk.Label(labelsFrame, text="Label1").grid(column=0, row=0, sticky=tk.W)
ttk.Label(labelsFrame, text="Label2").grid(column=1, row=0, sticky=tk.W)
ttk.Label(labelsFrame, text="Label3").grid(column=2, row=0, sticky=tk.W)

# Place cursor into name Entry
nameEntered.focus()
#======================
# Start GUI
#======================
win.mainloop()
```

Created first ttk label frame
Widget; given it a name

Create label names and
place them in the label frame

Use grid to arrange the labels
In the label frame
Parent is labelFram not win

Test it! What do you see?
What happens If you remove the sticky argument?

- Adding horizontal and vertical spacing

First we use the procedural way
Then change it to a loop

```
# Create a container to hold labels
labelsFrame = ttk.LabelFrame(win, text=' Labels in a Frame ')
labelsFrame.grid(column=0, row=7, padx=20, pady=30)
```

```python
# Create a container to hold labels
labelsFrame = ttk.LabelFrame(win, text=' Labels in a Frame ')
labelsFrame.grid(column=0, row=7, padx=20, pady=40)

# Place labels into the container element - vertically
ttk.Label(labelsFrame, text="Label1").grid(column=0, row=0)
ttk.Label(labelsFrame, text="Label2").grid(column=0, row=1)
ttk.Label(labelsFrame, text="Label3").grid(column=0, row=2)

for tomato in labelsFrame.winfo_children():
    tomato.grid_configure(padx=8, pady=4)

# Place cursor into name Entry
nameEntered.focus()
#======================
# Start GUI
#======================
win.mainloop()
```

Returns a list of all elements Belonging to the labelsFrame Variable

This enables us to loop through them and assign a padding to each label

Enables us to modify ui elements before the main loop displays them

Try it!
Change a label to a verrrrry looooooong label and
See what happens!
Then remove the name of the label Frame
and check again!

Expanding the GUI Dynamically Using Widgets

- Creates both an advantage and a little bit of a challenge
- At the beginning of the previous video we added a label frame widget
- The grid layout manager widget that lays out our widgets in a zero-based grid

Why? Widgets extend themselves to the space they need
JAVA introduced the concept of dynamic layout management
Longest name influences cell

```python
# Create a container to hold labels
labelsFrame = ttk.LabelFrame(win, text=' Labels in a Frame ')
labelsFrame.grid(column=0, row=7, padx=20, pady=40)

# Place labels into the container element - vertically
ttk.Label(labelsFrame, text="Label1").grid(column=0, row=0)
ttk.Label(labelsFrame, text="Label2").grid(column=0, row=1)
ttk.Label(labelsFrame, text="Label3").grid(column=0, row=2)

for tomato in labelsFrame.winfo_children():
    tomato.grid_configure(padx=8, pady=4)

# Place cursor into name Entry
nameEntered.focus()
#======================
# Start GUI
#======================
win.mainloop()
```

→ Longer than others

Checkboxes did not get centered why?
because we used sticky !

```
# Using a scrolled Text control
scrolW  = 30; scrolH  =  3
scr = scrolledtext.ScrolledText(monty, width=scrolW, height=scrolH, wrap=tk.WORD)
scr.grid(column=0, sticky='WE', columnspan=3)
```

remove columnspan & see

remove sticky

Now return to default
&
Change the labelFrame to column 1

Still messy as we use individual widgets

Lets use frames

Aligning the GUI Widgets by Embedding Frames within Frames

- Aligning the GUI widgets
- How to have a better control of our GUI layout

Here we create a top-level frame that will contain other frames and widgets

```python
#======================
import tkinter as tk
from tkinter import ttk
from tkinter import scrolledtext

# Create instance
win = tk.Tk()

# Add a title
win.title("Python GUI")

# Disable resizing the GUI
#win.resizable(0,0)

# Modify adding a Label
aLabel = ttk.Label(win, text="A Label")
aLabel.grid(column=0, row=0)

#Modified Button Click Function
def clickMe():
    action.configure(text='Hello ' + name.get())

# Changing our Label
ttk.Label(win, text="Enter a name:").grid(column=0, row=0)

# Adding a Textbox Entry widget
name = tk.StringVar()
nameEntered = ttk.Entry(win, width=12, textvariable=name)
nameEntered.grid(column=0, row=1)

# Adding a Button
action = ttk.Button(win, text="Click Me!", command=clickMe)
action.grid(column=2, row=1)
#action.configure(state='disabled')    # Disable the Button Widget
```

First embed our controls in a central .ttk label frame

This .ttk label frame is a child of the main parent window & all the controls will bechildren of this ttk label framee

So far we arranged all controls to the main GUI

Frame directly now we only assign the label Frame to the main window

- After that we will make this label frame the Parent container for all widgets

This creates a hierarchy

Such as here:

Win is the variable,

Window frame

Label frame reference
And is a child og the
Main window frame win

The label and other widgets
Are placed into that container

```
# We are creating a container frame to hold all other widgets
monty = ttk.LabelFrame(win, text=' Monty Python ')
monty.grid(column=0, row=0, padx=8, pady=4)
```

Then replace win and turn it into monty – replacing the controls – everywhere in The code

```
#Modified Button Click Function
def clickMe():
    action.configure(text='Hello ' + name.get())

# Changing our Label
ttk.Label(monty, text="Enter a name:").grid(column=0, row=0, sticky='W')

# Adding a Textbox Entry widget
name = tk.StringVar()
nameEntered = ttk.Entry(monty, width=12, textvariable=name)
nameEntered.grid(column=0, row=1, sticky='W')

# Adding a Button
action = ttk.Button(monty, text="Click Me!", command=clickMe)
action.grid(column=2, row=1)

ttk.Label(monty, text="Choose a number:").grid(column=1, row=0)
number = tk.StringVar()
```

Try and test – note how all the widgets are now in teh monty python label frame

Next we can reset the labels in a frame widget to the left without messing up The GUI layout;

Try a few moments

Oops have to use sticky property to align!

```
# Changing our Label
ttk.Label(monty, text="Enter a name:").grid(column=0, row=0, sticky='W')
```

If you only have one element in the framw you can make it
Fill the entire space with options NSWE !

Check all the W in the code file 7 and run it

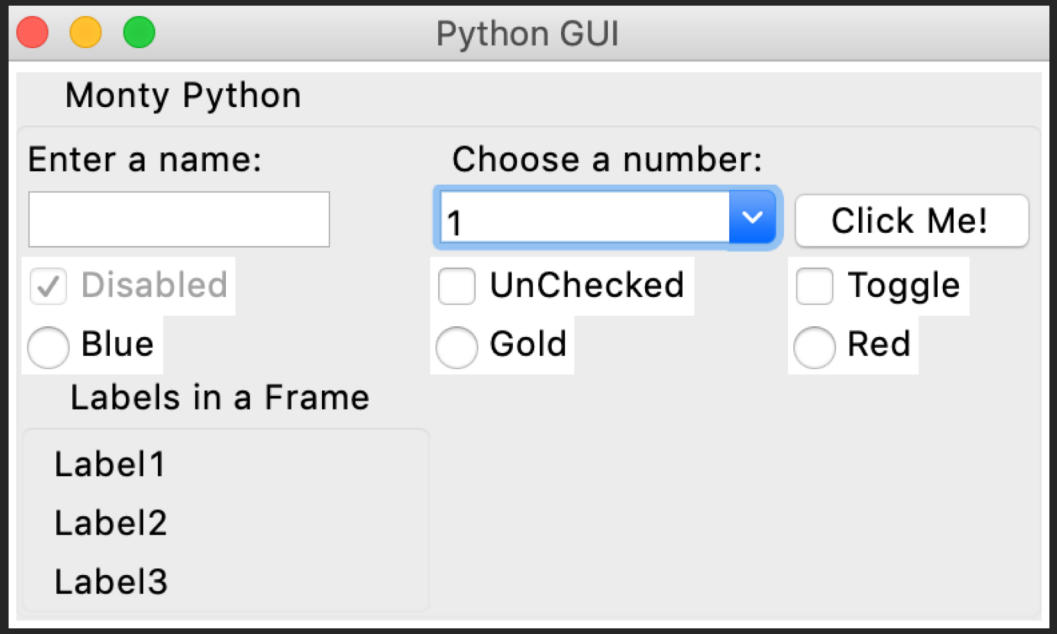
Cereating menu bars

- Adding a menu bar to our main window
- Adding menus to the menu bar
- Adding menu items to the menus

Clicking in a menu item will have no effect
Next we will ad a functionality to a menu item

For example
Closing the main window when clicking the exit menu
Item and displaying a help about dialogue

We extend teh menu file from the last section in to
A new file called 8MenuBar.py


Python GUI
Monty Python
Enter a name:
Choose a number:
1
Click Me!
Disabled
UnChecked
Toggle
Blue
Gold
Red
Labels in a Frame
Label1
Label2
Label3

First import Meu funcitons at the top

```
from tkinter import Menu
```

Next add menu bar

```
# Creating a Menu Bar
menuBar = Menu(win)
win.config(menu=menuBar)
```

Now we add menu item to the bar
And also assign a menu item to teh menu

```
# Add menu items
fileMenu = Menu(menuBar, tearoff=0)
fileMenu.add_command(label="New")
menuBar.add_cascade(label="File", menu=fileMenu)
```

```python
# Create a container to hold labels
labelsFrame = ttk.LabelFrame(monty, text=' Labels in a Frame ')
labelsFrame.grid(column=0, row=7)

# Place labels into the container element - vertically
ttk.Label(labelsFrame, text="Label1").grid(column=0, row=0)
ttk.Label(labelsFrame, text="Label2").grid(column=0, row=1)
ttk.Label(labelsFrame, text="Label3").grid(column=0, row=2)

# Add some space around each label
for child in labelsFrame.winfo_children():
    child.grid_configure(padx=8, pady=1)


# Creating a Menu Bar
menuBar = Menu(win)
win.config(menu=menuBar)

# Add menu items
fileMenu = Menu(menuBar, tearoff=0)
fileMenu.add_command(label="New")
menuBar.add_cascade(label="File", menu=fileMenu)

# Place cursor into name Entry
nameEntered.focus()

win.mainloop()
```

Here we cann constructor
And assign the menu to the
Main gui window

We save it in the instance
Then use it to configure
Our GUI

How we first add a menu item
Then add a menu

The add_cascade methods
Aligns the menu items one
Below the other

Now add second meny item !

File 8 menubar

We can add a separator line to
Group related menu items !

Remember the design lecture !

```
# Add menu items
fileMenu = Menu(menuBar, tearoff=0)
fileMenu.add_command(label="New")
fileMenu.add_separator()
fileMenu.add_command(label="Exit", command=_quit)
```

tearoff=0

In the menu item removes the first dashed
Line in the menu

Try it remove it and run the program
Then double click the dashed line

What happens?
(might not work on the mac)

Then add it again !

Next we add a second menu
And place it horizontally next to the first menu

In order for this to work we need to add it to the menu bar

It will be File and Help and in the menu Item for Help there will be About

Very common GUI layouts that we are familiar with

```
# Add menu items
fileMenu = Menu(menuBar, tearoff=0)
fileMenu.add_command(label="New")
fileMenu.add_separator()
fileMenu.add_command(label="Exit", command=_quit)
menuBar.add_cascade(label="File", menu=fileMenu)

# Add another Menu to the Menu Bar and an item
helpMenu = Menu(menuBar, tearoff=0)
helpMenu.add_command(label="About")
menuBar.add_cascade(label="Help", menu=helpMenu)
```

So we have menu items

Don’t to much

So lets add some stuff

Lets add a function that closes the application Pythonically

```
# Exit GUI cleanly
def _quit():
    win.quit()
    win.destroy()
    exit()
```

```
# Exit GUI cleanly
def _quit():
    win.quit()
    win.destroy()
    exit()
```

So how do we bind the function to a menu item?

```
fileMenu.add_command(label="Exit", command=_quit)
```

Note we use a pythojn convention here to indicate that Quit is a private function not used by clients of our code Using _code

Try it !
8 tabbed…. code

- Creating tabbed widgets to organize our expanding GUI

Use minimum code now and fill in new widgets into this
New tabbed layout

```
import tkinter as tk
from tkinter import ttk
win = tk.Tk()
win.title("Python GUI")

# Tab Control introduced here --------------------------------------
tabControl = ttk.Notebook(win)          # Create Tab Control

tab1 = ttk.Frame(tabControl)            # Create a tab
tabControl.add(tab1, text='Tab 1')      # Add the tab
tabControl.pack(expand=1, fill="both")  # Pack to make visible
# ~ Tab Control introduced here -----------------------------------------

win.mainloop()
```

Lets try and run it

First we used grid layout manager for simple GUIs
Here we can use pack !

```python
import tkinter as tk
from tkinter import ttk
win = tk.Tk()
win.title("Python GUI")

# Tab Control introduced here -----------------------------------------
tabControl = ttk.Notebook(win)          # Create Tab Control

tab1 = ttk.Frame(tabControl)            # Create a tab
tabControl.add(tab1, text='Tab 1')      # Add the tab
tabControl.pack(expand=1, fill="both")  # Pack to make visible
# ~ Tab Control introduced here ---------------------------------------

win.mainloop()
```

```
# Tab Control introduced here -------------------------------------
tabControl = ttk.Notebook(win)          # Create Tab Control

tab1 = ttk.Frame(tabControl)            # Create a tab
tabControl.add(tab1, text='Tab 1')      # Add the tab

tab2 =ttk.Frame(tabControl)             # Add a second tab
tabControl.add(tab2, text='Tab 2')      # Make secondtab visible
tabControl.pack(expand=1, fill="both")  # Pack to make visible
# ~ Tab Control introduced here -----------------------------------

win.mainloop()
```

Lets try and run it with a second tab

Lets try and run it with a second tab

Now lets add the monty part labelFrame back into one tab

```
import tkinter as tk
from tkinter import ttk
win = tk.Tk()
win.title("Python GUI")

# Tab Control introduced here -----------------------------------------
tabControl = ttk.Notebook(win)          # Create Tab Control

tab1 = ttk.Frame(tabControl)            # Create a tab
tabControl.add(tab1, text='Tab 1')      # Add the tab
tabControl.pack(expand=1, fill="both")  # Pack to make visible
# ~ Tab Control introduced here ---------------------------------------

# We are creating a container frame to hold all other widgets
monty = ttk.LabelFrame(tab1, text=' Monty Python ')
monty.grid(column=0, row=0, padx=8, pady=4)
# Changing our Label
ttk.Label(monty, text="Enter a name:").grid(column=0, row=0, sticky='W')

win.mainloop()
```

```python
import tkinter as tk
from tkinter import ttk
from tkinter import scrolledtext
from tkinter import Menu

win = tk.Tk()
win.title("Python GUI")

# Tab Control introduced here -------------------------------------
tabControl = ttk.Notebook(win)          # Create Tab Control

tab1 = ttk.Frame(tabControl)            # Create a tab
tabControl.add(tab1, text='Tab 1')      # Add the tab

tab2 = ttk.Frame(tabControl)            # Add a second tab
tabControl.add(tab2, text='Tab 2')      # Make second tab visible

tabControl.pack(expand=1, fill="both")  # Pack to make visible
# ~ Tab Control introduced here -----------------------------------------

# We are creating a container frame to hold all other widgets
monty = ttk.LabelFrame(tab1, text=' Monty Python ')
monty.grid(column=0, row=0, padx=8, pady=4)

#Modified Button Click Function
def clickMe():
    action.configure(text='Hello ' + name.get())

# Changing our Label
ttk.Label(monty, text="Enter a name:").grid(column=0, row=0, sticky='W')

# Adding a Textbox Entry widget
name = tk.StringVar()
nameEntered = ttk.Entry(monty, width=12, textvariable=name)
nameEntered.grid(column=0, row=1, sticky='W')

# Adding a Button
action = ttk.Button(monty, text="Click Me!", command=clickMe)
action.grid(column=2, row=1)
```

We can import scrolled text and add our previous code

Note changed ”win” in labelFrame to be assigned to a tab !

Try running it !

Codefile 9 again

Try moving some UI features (widgets) to Tab2!

Note first we add a new container frame Where we will put all our widgets in the New tab

```
# We are creating a container frame to hold all other widgets -- Tab2
monty2 = ttk.LabelFrame(tab2, text=' The Snake ')
monty2.grid(column=0, row=0, padx=8, pady=4)

# Creating three checkbuttons
chVarDis = tk.IntVar()
check1 = tk.Checkbutton(monty2, text="Disabled", variable=chVarDis, state='disabled')
check1.select()
check1.grid(column=0, row=4, sticky=tk.W, columnspan=3)
```

monty2

Then try to rename the actions of teh RadioButtons

```
def radCall():
    radSel=radVar.get()
    if   radSel == 0: monty2.configure(text='Blue')
    elif radSel == 1: monty2.configure(text='Gold')
    elif radSel == 2: monty2.configure(text='Red')
```

This results in chaging teh name of teh label frame

Let’s review

- Some of the techniques of the grid layout manager

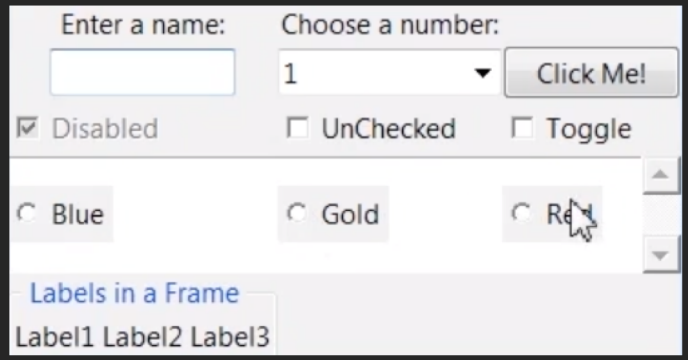
Grid layout manager is one of the most useful layout tools.

```
# Using a scrolled Text control
scrolW  = 30
scrolH  =  3
scr = scrolledtext.ScrolledText(win, width=scrolW, height=scrolH, wrap=tk.WORD)
scr.grid(column=0, sticky='WE', columnspan=3)
```

Tkinter automatically adds teh missing rows

- Where we did not specify

If you forget to specify as we did in the list
Then it will be automatically incremented



Be careful if you forget and assign this later

No let’s customize our Gui

- Creating message boxes
- Creating independent message boxes
- Creating the title and icon of the main root window
- Using a spin box control
- Creating tooltips and using the canvas widget

So let’s first look at creating message boxes

- Displaying an information box in our first example
- Creating a warning and error message boxes

Message box gives feedback to the user
We will add functionality to the help – about menu item

First we follow common design patterns of information display then warnings and then errors !

First let us add the functions from the mBox package

```
import tkinter as tk
from tkinter import ttk
from tkinter import scrolledtext
from tkinter import Menu
from tkinter import messagebox as mBox

# Create instance
win = tk.Tk()

# Add a title
win.title("Python GUI")

# We are creating a container frame to hold all other widgets
monty = ttk.LabelFrame(win, text=' Monty Python ')
monty.grid(column=0, row=0, padx=8, pady=4)

#Modified Button Click Function
def clickMe():
    action.configure(text='Hello ' + name.get())

# Changing our Label
ttk.Label(monty, text="Enter a name:").grid(column=0, row=0, sticky='W')

# Adding a Textbox Entry widget
name = tk.StringVar()
```

Then we create a callback function that creates a message box
Note the private function convention used here again

```
# Display a Message Box
def _msgBox():
    mBox.showinfo('Python Message Info Box', 'A Python GUI created using tkinter:\nThe year is 2018.')
```

Also code still procedural so we need to add this code for the
Callback before we use / call it above the specific segment

So just above where we create the help menu

```
# Display a Message Box
def _msgBox():
    mBox.showinfo('Python Message Info Box', 'A Python GUI created using tkinter:\nThe year is 2018.')

# Add another Menu to the Menu Bar and an item
helpMenu = Menu(menuBar, tearoff=0)
helpMenu.add_command(label="About", command=_msgBox)
menuBar.add_cascade(label="Help", menu=helpMenu)
```

Try adding this and run the code !

Now lets change the infobox to a warning box!

```
# Display a Message Box
def _msgBox():
    mBox.showwarning('Python Message
```

Try again !

Now lets try this to error message !

```
# Display a Message Box
def _msgBox():
    mBox.showerror('Python Me
```

Try again !

This message boxes are modal
Which means the user needs to interact with them
Click ok to disappear

Now we also have the option to give the user
Multiple choices and act differently on what is selected
For example in a dual choice case

```
# Display a Message Box
def _msgBox():
#    mBox.showinfo('Python Message Info Box', 'A Python GUI created using tkinter:\nThe year is 2015.')
#     mBox.showwarning('Python Message Warning Box', 'A Python GUI created using tkinter:\nWarning: There
#     mBox.showerror('Python Message Error Box', 'A Python GUI created using tkinter:\nError: Houston ~ w
    answer = mBox.askyesno("Python Message Dual Choice Box", "Are you sure you really wish to do this?")
    print(answer)
```

- Creating tkinter message boxes as stand-alone top-level GUI windows
- Ways to hide the extra window

Why customize our message boxes?

Well we might want to reuse them in Uis

Instead of copy & past the same code!

Remember Python lessons about moldularisation!

Lets start a new UI

Create a simple message box

Try it !

```
from tkinter import messagebox as mBox
mBox.showinfo('Python GUI created using tkinter:\nThe year is 2018')
```

What did you notice?!

Create a simple message box

Try it !

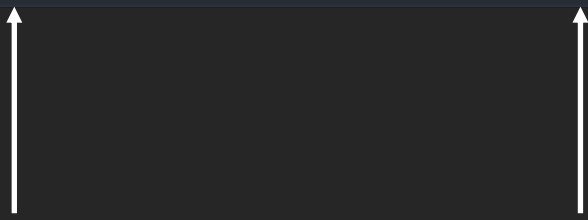
```
from tkinter import messagebox as mBox
mBox.showinfo('Python GUI created using tkinter:\nThe year is 2018')
```

What did you notice?!

Same thing happens in C#:
DOS debug window
Caused by a windows event loop!

How about now?

```
from tkinter import messagebox as mBox
mBox.showinfo('', 'Python GUI created using tkinter:\nThe year is 2018')
```



title

message

Can be suppressed

```
from tkinter import messagebox as mBox
from tkinter import Tk
root = Tk()
root.withdraw()
mBox.showinfo('', 'Python GUI created u
```

- Changing the main window title
- Changing its icon

Principle of changing the title of the main root window is the Same as we did before !

```
# Create instance
win = tk.Tk()

# Add a title
win.title("Python GUI")
```

win is a Tkinter instance and can use all the built in properties; here we give it a title

Now lets give our window a different icon than the default one

```
# Change the main windows icon
#win.iconbitmap(r'c:\Python34\DLLs\pyc.ico')
win.iconbitmap(r'pyc.ico')
```

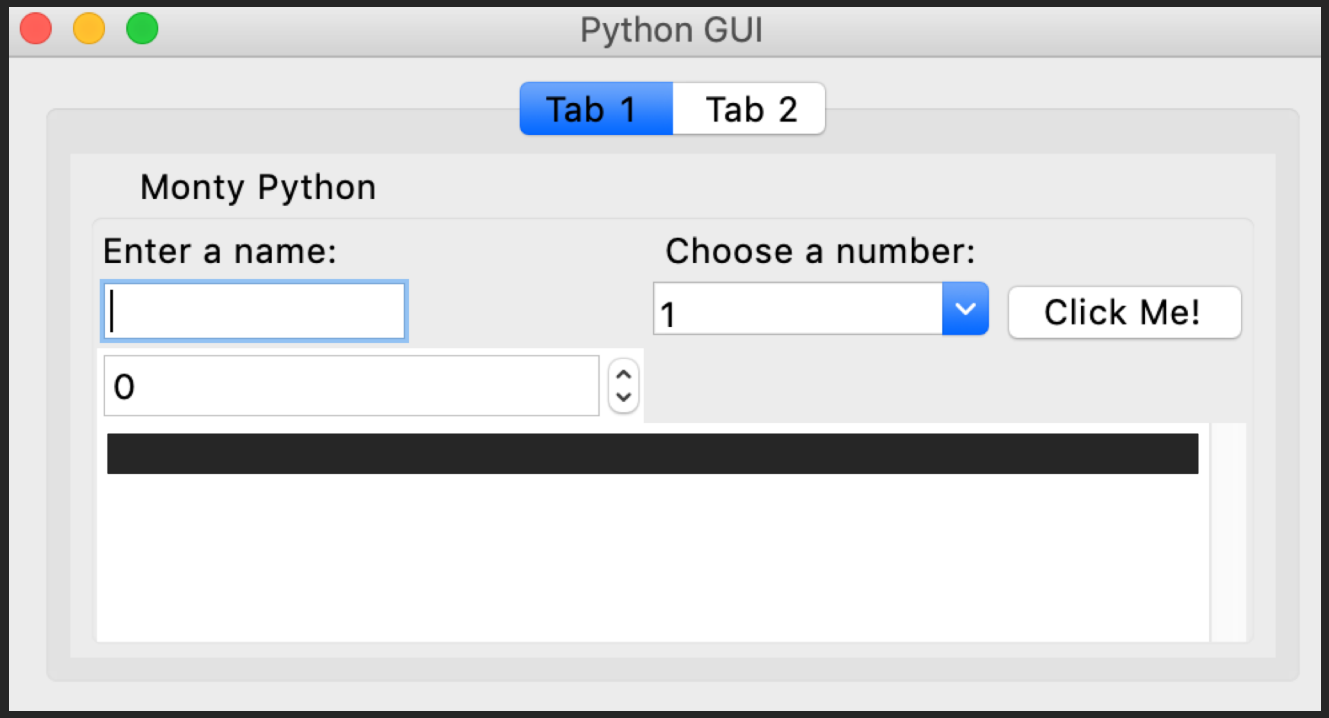
- Using a Spinbox widget and bind the Enter key to one of our widgets
- Controlling the appearance of our Spinbox widgets

Ok we are using our tabbed UI from before

```
# Adding a Spinbox widget
spin = Spinbox(monty, from_=0, to=10)
spin.grid(column=0, row=2)

# Using a scrolled Text control
scrolW  = 30; scrolH  =  3
scr = scrolledtext.ScrolledText(monty, width=scrolW, height=scrolH, wrap=tk.WORD)
scr.grid(column=0, row=3, sticky='WE', columnspan=3)
```

And we add a Spinbox widget above the scrolled Text control

Notice how the Spinbox stretches the UI



Also notice that the from_= & to= values are set

```
# Adding a Spinbox widget
spin = Spinbox(monty, from_=0, to=10)
spin.grid(column=0, row=2)

# Using a scrolled Text control
scrolW  = 30; scrolH  =  3
scr = scrolledtext.ScrolledText(monty, width=scrolW, height=scrolH, wrap=tk.WORD)
scr.grid(column=0, row=3, sticky='WE', columnspan=3)
```

Tkinter sets it to be default 10 (max)

What happens if you reverse this?

Try it!

Ok so lets reduce the size by specifying ‘width’)

```
# Adding a Spinbox widget
spin = Spinbox(monty, from_=0, to=10, width=5
spin.grid(column=0, row=2)
```

Try it!

bd is a short notation for borderwidth

```
# Adding a Spinbox widget
spin = Spinbox(monty, from_=0, to=10, width=5, bd=8
spin.grid(column=0, row=2)
```

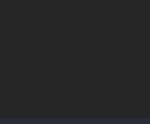
Now lets add functionality via a callback

Reference to scrollText widget

```
# Spinbox callback
def _spin():
    value = spin.get()
    print(value)
    scr.insert(tk.INSERT, value + '\n')

# Adding a Spinbox widget
spin = Spinbox(monty, from_=0, to=10, width=5, bd=8, command=_spin)
spin.grid(column=0, row=2)
```

Instead of range we can specify a set of values

```
from_=0, to=10
```



```
spin = Spinbox(monty, values=(1, 2, 4, 42, 100), width=5, bd=8, command=_spin,
spin.grid(column=1, row=2)
```

This restricts the values to a hard coded set
Can be read in via text or xml file

Relief can be set to raised , sunken, ridge and two more

```
spin = Spinbox(monty, values=(1, 2, 4, 42, 100), width=5, bd=8, command=_spin, relief=tk.RIDGE)
spin.grid(column=1, row=2)
```

Changing the relief results in a visual queue without changing anything else

- Creating Tooltips
- Adding dramatic color effects to our GUI

```
from tkinter import Menu
from tkinter import Spinbox


class ToolTip(object):
    def __init__(self, widget):
        self.widget = widget
        self.tipwindow = None
        self.id = None
        self.x = self.y = 0

    def showtip(self, text):
        "Display text in tooltip window"
        self.text = text
        if self.tipwindow or not self.text:
            return
        x, y, _cx, cy = self.widget.bbox("insert")
        x = x + self.widget.winfo_rootx() + 27
        y = y + cy + self.widget.winfo_rooty() +27
        self.tipwindow = tw = tk.Toplevel(self.widget)
        tw.wm_overrideredirect(1)
        tw.wm_geometry("+%d+%d" % (x, y))

        label = tk.Label(tw, text=self.text, justify=tk.LEFT,
                         background="#ffffe0", relief=tk.SOLID, borderwidth=1,
                         font=("tahoma", "8", "normal"))
        label.pack(ipadx=1)

    def hidetip(self):
        tw = self.tipwindow
        self.tipwindow = None
        if tw:
            tw.destroy()
```

Toolkits are not as simple as we'd like it to be

Lets place it into its own OOP class

```
        tw = self.tipwindow
        self.tipwindow = None
        if tw:
            tw.destroy()

#================================================
def createToolTip( widget, text):
    toolTip = ToolTip(widget)
    def enter(event):
        toolTip.showtip(text)
    def leave(event):
        toolTip.hidetip()
    widget.bind('<Enter>', enter)
    widget.bind('<Leave>', leave)

# Create instance
win = tk.Tk()

# Add a title
win.title("Python GUI")
```

```
# Add a Tooltip
createToolTip(spin, 'This is a Spin control.')
```

After declaring the class
We need to have a function
That makes use of it…

…so here it is!

Right after the class

Note Python allows us to
Place multiple classes into
The same module

The ToolTip class is a Python
class and in order to use it
We have to instantiate it!

Then all we have to do is
Adding the tooltip:

Try it!

How would you add a tooltip for the scrolledText widget?

```
# Add a Tooltip
createToolTip(spin, 'This is a Spin control.')


# Using a scrolled Text control
scrolW  = 30; scrolH  =  3
scr = scrolledtext.ScrolledText(monty, width=scrolW, height=scrolH, wrap=tk.WORD)
scr.grid(column=0, row=3, sticky='WE', columnspan=3)

createToolTip(src, 'This is a src control.')
```

```
class Tooltip(object):
    pass


class ToolTip():
    pass
```

Lets add a a new tab and use canvas

```python
# Tab Control introduced here -------------------------------------
tabControl = ttk.Notebook(win)          # Create Tab Control

tab1 = ttk.Frame(tabControl)            # Create a tab
tabControl.add(tab1, text='Tab 1')      # Add the tab

tab2 = ttk.Frame(tabControl)            # Add a second tab
tabControl.add(tab2, text='Tab 2')      # Make second tab visible

tab3 = ttk.Frame(tabControl)            # Add a third tab
tabControl.add(tab3, text='Tab 3')      # Make second tab visible

tabControl.pack(expand=1, fill="both")  # Pack to make visible
# ~ Tab Control introduced here -----------------------------------------
```

We add the canvas widget to the third tab

```
# Tab Control 3 ------------------------------
tab3 = tk.Frame(tab3, bg='blue')
tab3.pack()
for orangeColor in range(2):
    canvas = tk.Canvas(tab3, width=150, height=80, highlightthickness=0, bg='orange')
    canvas.grid(row=orangeColor, column=orangeColor)
```

Try it!

- Creating beautiful charts using Matplotlib
- Downloading modules for Matplotlib
- Creating our first chart
- Placing labels on charts
- Giving the chart a legend
- Scaling charts
- Adjusting the scale dynamically

Depending on the data source we can plot one or
Several columns / rows of data in the same chart

Matplotlib & Numpy

Note using this in Eclipse (pydev) may cause errors
Just ignore – it’s a bug

- Downloading modules using pip
- Downloading more modules with the whl extension

Manual but better with pip or conda install !

NumPy, a fundamental package needed for scientific computing with Python.
Numpy+MKL is linked to the Intel® Math Kernel Library and includes required DLLs in the numpy.core dire

numpy-1.11.1+mkl-cp27-cp27m-win32.whl
numpy-1.11.1+mkl-cp27-cp27m-win_amd64.whl
numpy-1.11.1+mkl-cp34-cp34m-win32.whl
numpy-1.11.1+mkl-cp34-cp34m-win_amd64.whl
numpy-1.11.1+mkl-cp35-cp35m-win32.whl
numpy-1.11.1+mkl-cp35-cp35m-win_amd64.whl

ODE, the Open Dynamics Engine, a high performance library for simulating rigid body dynamics.

ode-0.13.1-cp27-none-win32.whl
ode-0.13.1-cp27-none-win_amd64.whl
ode-0.13.1-cp34-none-win32.whl
ode-0.13.1-cp34-none-win_amd64.whl
ode-0.13.1-cp35-none-win32.whl
ode-0.13.1-cp35-none-win_amd64.whl

OpenBabel, the open source chemistry toolbox.

openbabel-1.8.4-cp27-none-win32.whl
openbabel-1.8.4-cp27-none-win_amd64.whl
openbabel-1.8.4-cp34-none-win32.whl
openbabel-1.8.4-cp34-none-win_amd64.whl
openbabel-1.8.4-cp35-none-win32.whl

- Creating our first Matpotlib chart

Note only works after calling show()

```
import numpy as np
import matplotlib.pyplot as plt
from pylab import show

x = np.arange(0, 5, 0.1);
y = np.sin(x)
plt.plot(x, y)

show()
```

Lets try this in jupyter !

Lets try another

```
from pylab import show, arange, sin, plot, pi

t = arange(0.0, 2.0, 0.01)
s = sin( 2 * pi * t )
plot(t, s)

show()
```

Careful with function names of Numpy

- arange
- linspace
- pyglet

Placing Labels on Charts

- Position labels onto both the horizontal and vertical axes
- Using sub plots to draw many graphs in the same window

```
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import tkinter as tk
#--------------------------------------------------------------------
fig = Figure(figsize=(12, 8), facecolor='white')
#--------------------------------------------------------------------
axis = fig.add_subplot(211)     # 2 rows, 1 column, Top graph
#--------------------------------------------------------------------
xValues = [1,2,3,4]
yValues0 = [5,7,6,8]
axis.plot(xValues, yValues0)

axis.set_xlabel('Horizontal Label')
axis.set_ylabel('Vertical Label')

axis.grid(linestyle='-')        # solid grid lines
#--------------------------------------------------------------------
axis1 = fig.add_subplot(212)    # 2 rows, 1 column, Bottom graph
#--------------------------------------------------------------------
xValues1 = [1,2,3,4]
yValues1 = [7,5,8,6]
axis1.plot(xValues1, yValues1)
axis1.grid()                    # default line style
#--------------------------------------------------------------------
def _destroyWindow():
    root.quit()
    root.destroy()
#--------------------------------------------------------------------
root = tk.Tk()
root.withdraw()
root.protocol('WM_DELETE_WINDOW', _destroyWindow)
#--------------------------------------------------------------------
canvas = FigureCanvasTkAgg(fig, master=root)
canvas._tkcanvas.pack(side=tk.TOP, fill=tk.BOTH, expand=1)
#--------------------------------------------------------------------
root.update()
root.deiconify()
root.mainloop()
```

Using canvases allows us to take control over our paintings

Try modifying the plots

Legend added

```
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import tkinter as tk
#--------------------------------------------------------------------
fig = Figure(figsize=(12, 5), facecolor='white')
#--------------------------------------------------------------------
axis  = fig.add_subplot(111)                # 1 row, 1 column

xValues  = [1,2,3,4]

yValues0 = [6,7.5,8,7.5]
yValues1 = [5.5,6.5,8,6]
yValues2 = [6.5,7,8,7]

# the commas after t0, t1 and t2 are required
t0, = axis.plot(xValues, yValues0)
t1, = axis.plot(xValues, yValues1)
t2, = axis.plot(xValues, yValues2)

# t0, = axis.plot(xValues, yValues0, color = 'purple')  # change the color of the plotted
# t0, = axis.plot(xValues, yValues0, color = 'r')  # change the color of the plotted line
# t1, = axis.plot(xValues, yValues1, color = 'b')
# t2, = axis.plot(xValues, yValues2, color = 'purple')

axis.set_ylabel('Vertical Label')
axis.set_xlabel('Horizontal Label')

axis.grid()

fig.legend((t0, t1, t2), ('First line', 'Second line', 'Third line'), 'upper right')

#--------------------------------------------------------------------
def _destroyWindow():
    root.quit()
    root.destroy()
#
```

```
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import tkinter as tk
#--------------------------------------------------------------------
fig = Figure(figsize=(12, 5), facecolor='white')
#--------------------------------------------------------------------
axis  = fig.add_subplot(111)        # 1 row, 1 column

xValues  = [1,2,3,4]

yValues0 = [6,7.5,8,7.5]
yValues1 = [5.5,6.5,50,6]           # one very high value (50)
yValues2 = [6.5,7,8,7]

axis.set_ylim(5, 8)                 # limit the vertical display

t0, = axis.plot(xValues, yValues0)
t1, = axis.plot(xValues, yValues1)
t2, = axis.plot(xValues, yValues2)

axis.set_ylabel('Vertical Label')
axis.set_xlabel('Horizontal Label')

axis.grid()

fig.legend((t0, t1, t2), ('First line', 'Second line', 'Third line'), 'upper right')
```


Scaling the data

- Setting a limit to the data
- Analyzing the data before we represent it

Scaling the charts dynamically

```
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import tkinter as tk
#--------------------------------------------------------------------
fig = Figure(figsize=(12, 5), facecolor='white')
#--------------------------------------------------------------------
axis  = fig.add_subplot(111)        # 1 row, 1 column

xValues  = [1,2,3,4]

yValues0 = [6,7.5,8,7.5]
yValues1 = [5.5,6.5,50,6]           # one very high value (50)
yValues2 = [6.5,7,8,7]
yAll = [yValues0, yValues1, yValues2]   # list of lists

# flatten list of lists retrieving minimum value
minY = min([y for yValues in yAll for y in yValues])

yUpperLimit = 20
# flatten list of lists retrieving max value within defined limit
maxY = max([y for yValues in yAll for y in yValues if y < yUpperLimit])

# dynamic limits
axis.set_ylim(minY, maxY)
axis.set_xlim(min(xValues), max(xValues))

t0, = axis.plot(xValues, yValues0)
t1, = axis.plot(xValues, yValues1)
t2, = axis.plot(xValues, yValues2)

axis.set_ylabel('Vertical Label')
axis.set_xlabel('Horizontal Label')
```

- Displaying widget text in different languages
- Changing the entire GUI language all at once
- Localizing the GUI
- Preparing the GUI for internationalization
- Designing a GUI in an agile fashion

- Do we need to test the GUI code
- Setting debug watches
- Configuring different debug output levels
- Creating self-testing code using Python's __main__ section
- Creating robust GUIs using unit tests
- Writing unit tests using the Eclipse PyDev IDE

So lets start with this

- Internationalizing our GUI
- Expanding the window's title string
- Defining the GUI title string in two languages
- Modifying the init() method

```python
class I18N():
    '''Internationalization'''
    def __init__(self, language):
        if   language == 'en': self.resourceLanguageEnglish()
        elif language == 'de': self.resourceLanguageGerman()
        else: raise NotImplementedError('Unsupported language.')

    def resourceLanguageEnglish(self):
        self.title = "Python Graphical User Interface"

        self.file  = "File"
        self.new   = "New"
        self.exit  = "Exit"
        self.help  = "Help"
        self.about = "About"

        self.WIDGET_LABEL = ' Widgets Frame '

        self.disabled  = "Disabled"
        self.unChecked = "UnChecked"
        self.toggle    = "Toggle"

        # Radiobutton list
        self.colors   = ["Blue", "Gold", "Red"]
        self.colorsIn = ["in Blue", "in Gold", "in Red"]

        self.labelsFrame  = ' Labels within a Frame '
        self.chooseNumber = "Choose a number:"
        self.label2       = "Label 2"

        self.timeZones = "All Time Zones"
        self.localZone = "Local Zone"
        self.getTime   = "New York"

        self.mgrFiles = ' Manage Files '

        self.browseTo = "Browse to File..."
        self.copyTo   = "Copy File To :   "
```

Note this code has some Mysql part we didn't Look into in this course

Lets remove the hard Coded string values and Create a new class for them (resources.py)

```
    def resourceLanguageGerman(self):
        self.title = 'Python Grafische Benutzeroberfl' + "\u00E4" + 'che'

        self.file  = "Datei"
        self.new   = "Neu"
        self.exit  = "Schliessen"
        self.help  = "Hilfe"
        self.about = "\u00DC" + "ber"
#         self.about = "Über"

        self.WIDGET_LABEL = ' Widgets Rahmen '

        self.disabled  = "Deaktiviert"
        self.unChecked = "Nicht Markiert"
        self.toggle    = "Markieren"

        # Radiobutton list
        self.colors   = ["Blau", "Gold", "Rot"]
        self.colorsIn = ["in Blau", "in Gold", "in Rot"]

        self.labelsFrame  = ' Etiketten im Rahmen '
        self.chooseNumber = "Waehle eine Nummer:"
        self.label2       = "Etikette 2"

        self.timeZones = "Alle Zeitzonen"
        self.localZone = "Lokale Zone"
        self.getTime   = "Zeit"

        self.mgrFiles = ' Dateien Organisieren '

        self.browseTo = "Waehle eine Datei... "
        self.copyTo   = "Kopiere Datei zu :     "

if __name__ == '__main__':
    language = 'en'
    inst = I18N(language)
    print(inst.title)

    language = 'de'
    inst = I18N(language)
    print(inst.title)
```

We include two functions
In that class:

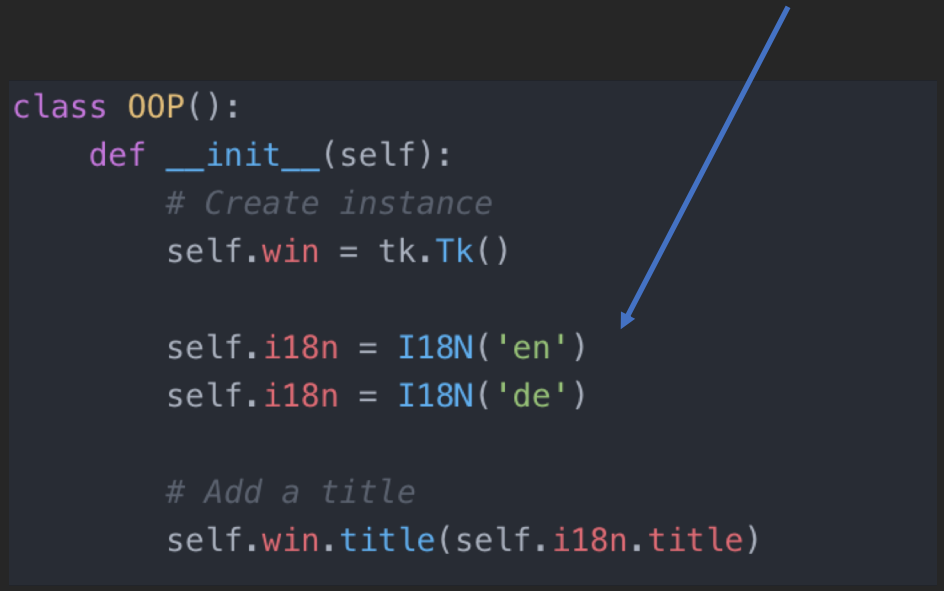
One for English

And one for German

We are separating the UI
From the language display
Functionality! OOP !

```
from Resources import I18N
```

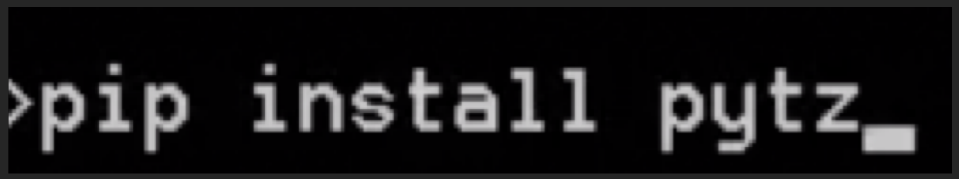
Then use the new way



```
class OOP():
    def __init__(self):
        # Create instance
        self.win = tk.Tk()

        self.i18n = I18N('en')
        self.i18n = I18N('de')

        # Add a title
        self.win.title(self.i18n.title)
```

- Converting the local time of the USA west coast to UTC
- Displaying the USA east coast time in our GUI

Timezone customisation



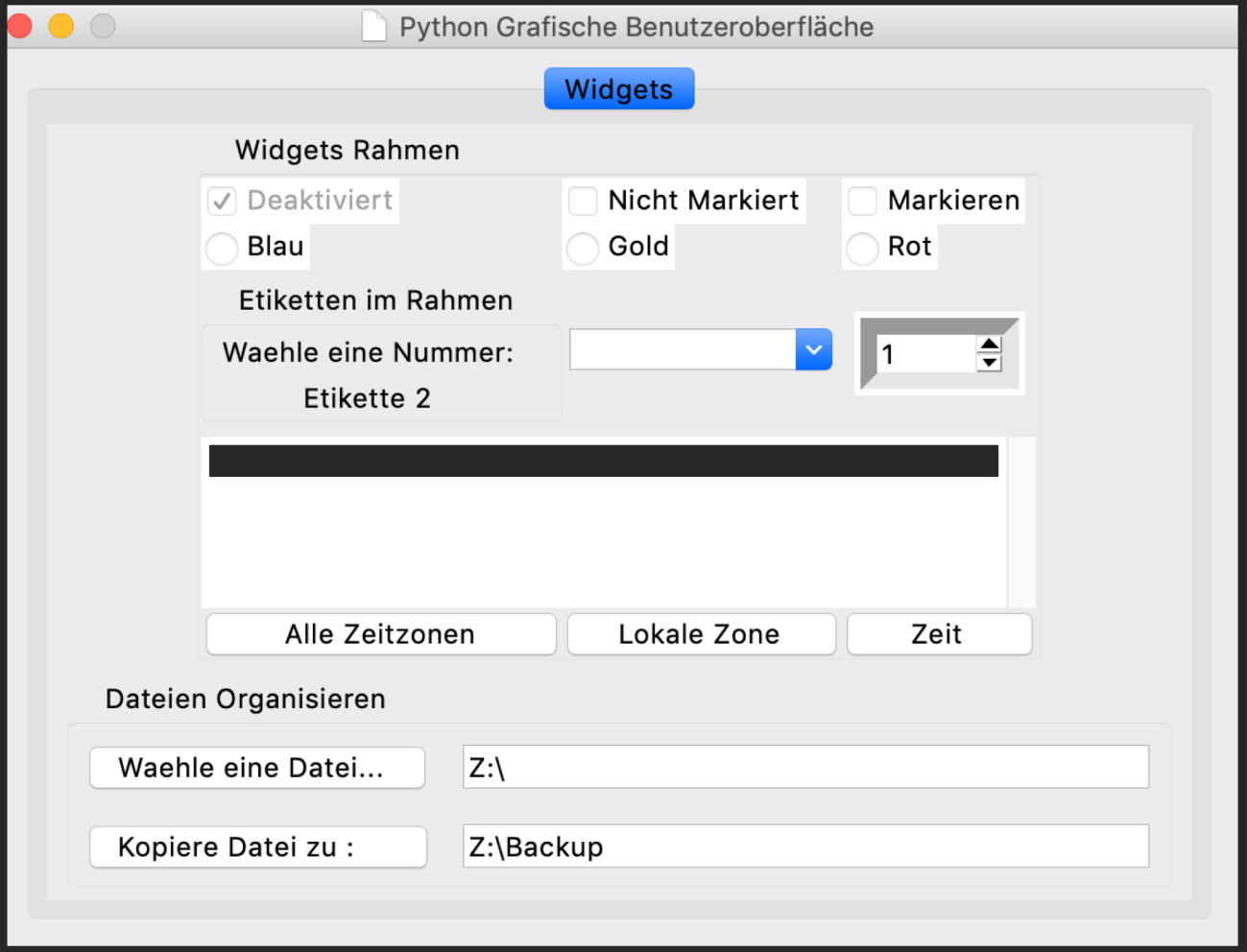

```
from datetime import datetime
from pytz import all_timezones, timezone
```

First lets add the function to create a time zone

```
# TZ Button callback
def allTimeZones(self):
    for tz in all_timezones:
        self.scr.insert(tk.INSERT, tz + '\n')
```

Then a button widget

```
# Adding a TZ Button
self.allTZs = ttk.Button(self.widgetFrame,
                         text=self.i18n.timeZones,
                         command=self.allTimeZones)
self.allTZs.grid(column=0, row=9, sticky='WE')
```


Python Grafische Benutzeroberfläche
Widgets
Widgets Rahmen
Deaktiviert
Blau
Nicht Markiert
Gold
Markieren
Rot
Etiketten im Rahmen
Waehle eine Nummer:
Etikette 2
1
Alle Zeitzonen
Lokale Zone
Zeit
Dateien Organisieren
Waehle eine Datei...
Z:\
Kopiere Datei zu :
Z:\Backup

```
# TZ Local Button callback
def localZone(self):
    from tzlocal import get_localzone
    self.scr.insert(tk.INSERT, get_localzone())


# Format local US time with TimeZone info
def getDateTime(self):
    fmtStrZone = "%Y-%m-%d %H:%M:%S %Z%z"
    # Get Coordinated Universal Time
    utc = datetime.now(timezone('UTC'))
    print(utc.strftime(fmtStrZone))

    # Convert UTC datetime object to Los Angeles TimeZone
    la = utc.astimezone(timezone('America/Los_Angeles'))
    print(la.strftime(fmtStrZone))

    # Convert UTC datetime object to New York TimeZone
    ny = utc.astimezone(timezone('America/New_York'))
    print(ny.strftime(fmtStrZone))

    # update GUI label with NY Time and Zone
    self.lbl2.set(ny.strftime(fmtStrZone))
```

For the other two buttons

- Improving the readability of our code
- Breaking out the callback methods into their own class and module

```
import tkinter as tk
from tkinter import ttk, scrolledtext, Menu, Spinbox, \
                    filedialog as fd, messagebox as mBox
from queue import Queue
from os import path
import ToolTip as tt
from MySQL import MySQL
from Resources import I18N
from Callbacks_Refactored import Callbacks
from Logger import Logger, LogLevel
```

Already broken out the strings

Then added more imports at the Top and functions or code at the bottom:

Waterfall model

Now lets group our imports

```
class OOP():
    def __init__(self, language='en'):
        # Create instance
        self.win = tk.Tk()

        self.i18n = I18N(language)
#         self.i18n = I18N(language)

        # Add a title
        self.win.title(self.i18n.title)

        # Callback methods now in different module
        self.callBacks = Callbacks(self)

        # Disable resizing the window
        self.win.resizable(0,0)
```

Then we pass an instance of our own class

Check 4Callbacks… & run

- Testing the GUI we have recently created
- Examples of what can go wrong and why we need to keep testing our code

- Interpreter self checking our code
- Second level is source code control & versioning system
- Third level: use of API (encapsulate potential future changes)
- Integration testing (half of the road meets the other one)
- Then there is user testing

Eclipse Pydev

IDLE Debugger

```
1
2 def multiply(num):
3     print(num *num)
4 multiply(3)
```

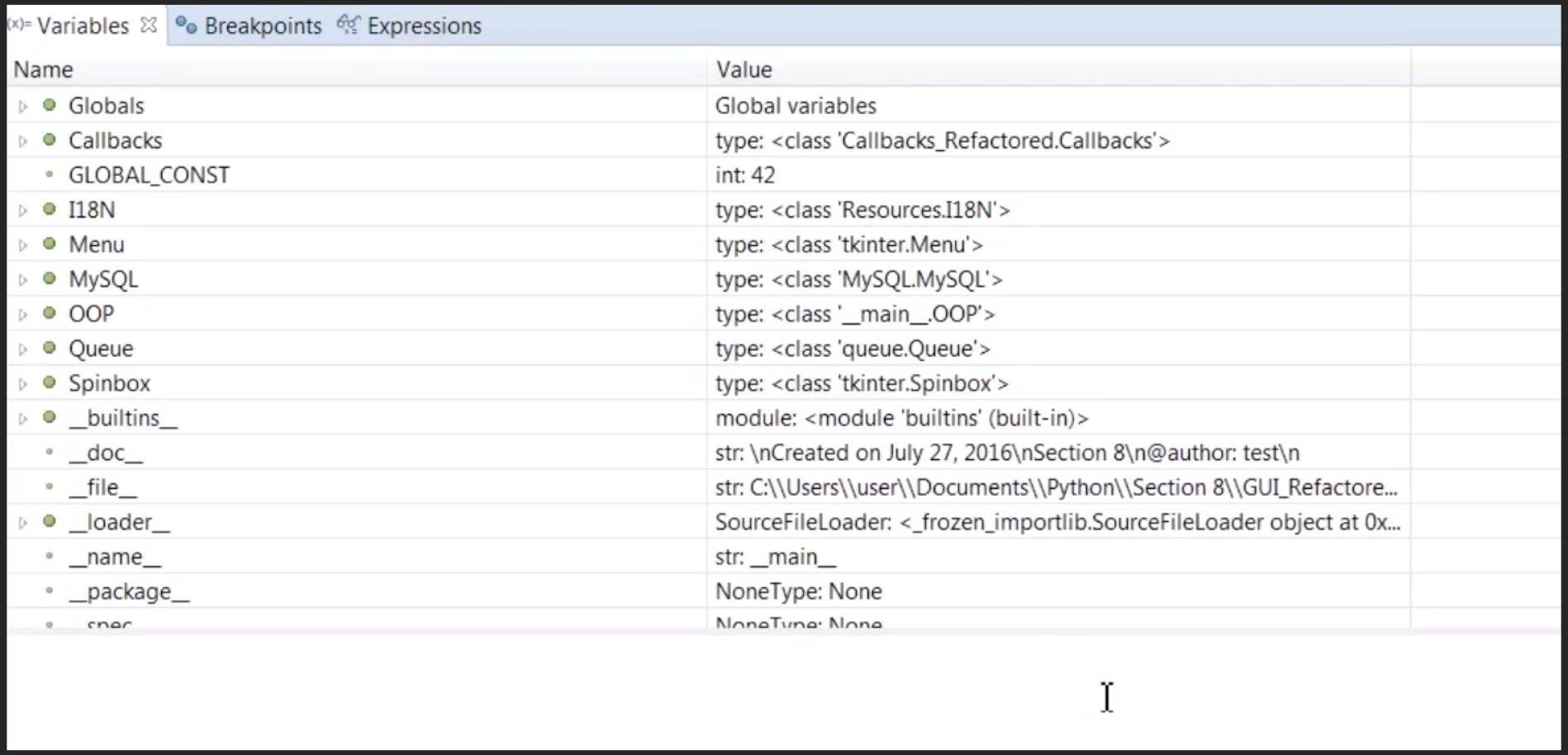
```
def multiply(num):
    print(num ** num)
multiply(3)
```

Regression bug

- The code we previously developed and set debug watches

```
301             self.bookTitle.focus()
302
303             # Add a Tooltip to the Spinbox
304             tt.createToolTip(self.spin, 'This is a Spin control.')
305
306             # Add Tooltips to more widgets
307             tt.createToolTip(self.bookTitle, 'This is an Entry control.')
308             tt.createToolTip(self.action, 'This is a Button control.')
309             tt.createToolTip(self.scr,    'This is a ScrolledText control.')
310
311 if __name__ == '__main__':
312     #======================
313     # Start GUI
314     #======================
315     oop = OOP()
316     oop.win.mainloop()
```

Setting breakpoints at the stage where we make our UI visible


Variables
Breakpoints
Expressions
Name
Value
Globals
Global variables
Callbacks
type: <class 'Callbacks_Refactored.Callbacks'>
GLOBAL_CONST
int: 42
I18N
type: <class 'Resources.I18N'>
Menu
type: <class 'tkinter.Menu'>
MySQL
type: <class 'MySQL.MySQL'>
OOP
type: <class '__main__.OOP'>
Queue
type: <class 'queue.Queue'>
Spinbox
type: <class 'tkinter.Spinbox'>
__builtins__
module: <module 'builtins' (built-in)>
__doc__
str: \nCreated on July 27, 2016\nSection 8\n@author: test\n
__file__
str: C:\\Users\\user\\Documents\\Python\\Section 8\\GUI_Refactore...
__loader__
SourceFileLoader: <_frozen_importlib.SourceFileLoader object at 0x...
__name__
str: __main__
__package__
NoneType: None

- F5 – step into code
- F6 – step over
- F7 – step out of the current method

- Creating our own logging class
- Configuring different debug levels that we can select and change at runtime

```python
import os, time
from datetime import datetime

class LogLevel:
    '''Define logging levels.'''
    OFF     = 0
    MINIMUM = 1
    NORMAL  = 2
    DEBUG   = 3


class Logger:
    ''' Create a test log and write to it. '''
    #-------------------------------------------------------------------------
    def __init__(self, fullTestName, loglevel=LogLevel.DEBUG):
        testName = os.path.splitext(os.path.basename(fullTestName))[0]
        logName  = testName  + '.log'

        logsFolder = 'logs'
        if not os.path.exists(logsFolder):
            os.makedirs(logsFolder, exist_ok = True)

        self.log = os.path.join(logsFolder, logName)
        self.createLog()

        self.loggingLevel = loglevel
        self.startTime    = time.perf_counter()

    #-------------------------------------------------------------------------
    def createLog(self):
        with open(self.log, mode='w', encoding='utf-8') as logFile:
            logFile.write(self.getDateTime() +
                          '\t\t*** Starting Test ***\n')
        logFile.close()

    #-------------------------------------------------------------------------
    def setLoggingLevel(self, level):
        '''change logging level in the middle of a test.'''
```

Enumerator class

Creates a logging file and
Places it into a logs folder

If it doesn’t exist then we
Create the folder automatically

```
#-----------------------------------------------------------------------------
def setLoggingLevel(self, level):
    '''change logging level in the middle of a test.'''
    self.loggingLevel = level

#-----------------------------------------------------------------------------
def writeToLog(self, msg='', loglevel=LogLevel.DEBUG):
    # control how much gets logged
    if loglevel > self.loggingLevel:
        return

    # open log file in append mode
    with open(self.log, mode='a', encoding='utf-8') as logFile:
        msg = str(msg)
        if msg.startswith('\n'):
            msg = msg[1:]
        logFile.write(self.getDateTime() + '\t\t' + msg + '\n')

    logFile.close()

#-----------------------------------------------------------------------------
def getDateTime(self,):
    return datetime.now().strftime("%Y-%m-%d %H:%M:%S")

#-----------------------------------------------------------------------------
def writeTestRunTime(self):
    elapsed = time.perf_counter() - self.startTime
    self.writeToLog('\nElapsed Test Time (seconds): {0:.2f}'.format(elapsed), logle
```

Check if message has
Logging level

If message has a level
We want to log
Then check if message
Starts with newline
If so discard it

Then write one line

In GUI_refactored create the logger instance

```
# create Logger instance
fullPath = path.realpath(__file__)
self.log = Logger(fullPath)

# create Log Level instance
self.level = LogLevel()
```

Self testing code using __main__ section !

Python can make modules self test &
main self testing section can serve as documentation!

- Introducing our own logging file
- Creating different logging levels

```python
if __name__ == '__main__':
    language = 'en'
    inst = I18N(language)
    print(inst.title)

    language = 'de'
    inst = I18N(language)
    print(inst.title)
```

First lets add self testing section
to our resources.py module

Every module who has this section will run when Executed and self test

Note **NOT** if you import it and execute – only self test

```python
if __name__ == '__main__':
    #======================
    # Start GUI
    #======================
    oop = OOP()
    print(oop.log)
#     oop.log.setLoggingLevel(oop.level.DEBUG)
    oop.log.setLoggingLevel(oop.level.MINIMUM)
    oop.log.writeToLog('Test message')
    oop.win.mainloop()
```

Now lets do this for our
main Refactored UI

- Writing unit tests in Python
- Some Python unit test code that can contain mistakes which need to be corrected

Python comes with built in Unit test capability

1) Design testing strategy
2) Separate application code from Unit test code

UnitTests.py

```
import unittest

class GuiUnitTests(unittest.TestCase):
    pass

if __name__ == '__main__':
    unittest.main()
```

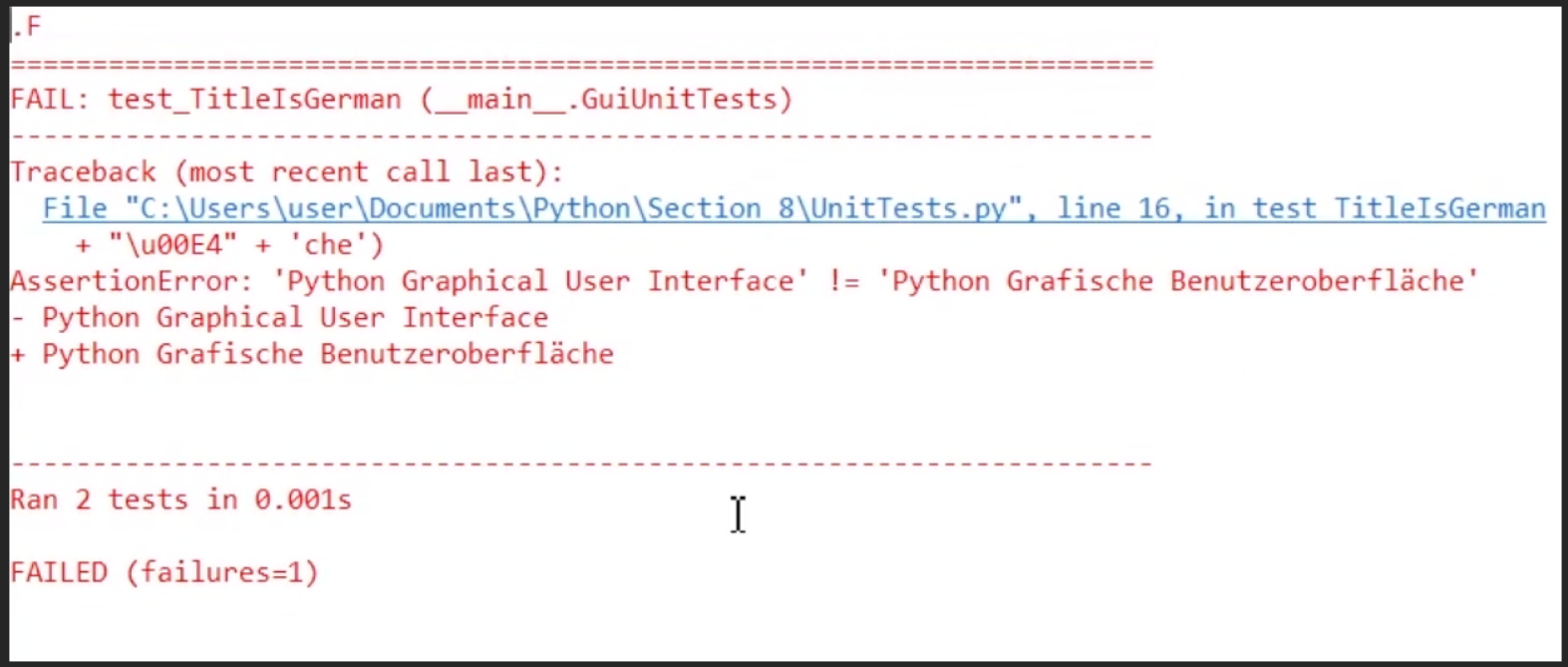
Try and run this !

```
import unittest
from Resources import I18N
from GUI_Refactored import OOP as GUI

class GuiUnitTests(unittest.TestCase):

    def test_TitleIsEnglish(self):
        i18n = I18N('en')
        self.assertEqual(i18n.title,
                         "Python Graphical User Interface")

    def test_TitleIsGerman(self):
        # i18n = I18N('en')           # <= Bug in Unit Test
        i18n = I18N('de')
        self.assertEqual(i18n.title,
                         'Python Grafische Benutzeroberfl'
                         + "\u00E4" + 'che')
```

First test here for the title
assertEqual() will check this

Note we import our classes

Now change the en to de
And see what happens!



Test Driven Development TDD you create the tests first! ☺

- Extending our unit testing code by testing labels and programmatically invoking a radio button
- Verifying in our unit tests that the text property of the LabelFrame widget has changed as expected
- Testing two different languages
- Using the built-in Eclipse/PyDev graphical unit test runner

```
class WidgetsTestsGerman(unittest.TestCase):

    def setUp(self):
        self.gui = GUI('de')

    def test_WidgetLabels(self):
        self.assertEqual(self.gui.i18n.file, "Datei")
        self.assertEqual(self.gui.i18n.mgrFiles,
                                        ' Dateien Organisieren ')
        self.assertEqual(self.gui.i18n.browseTo,
                                        "Waehle eine Datei... ")

    def test_LabelFrameText(self):
        labelFrameText = self.gui.widgetFrame['text']
        self.assertEqual(labelFrameText, " Widgets Rahmen ")
        self.gui.radVar.set(1)
        self.gui.callBacks.radCall()
        labelFrameText = self.gui.widgetFrame['text']
        self.assertEqual(labelFrameText,
                                    " Widgets Rahmen in Gold")
```

Unit test main
Runs any method
That starts with prefix
'test'

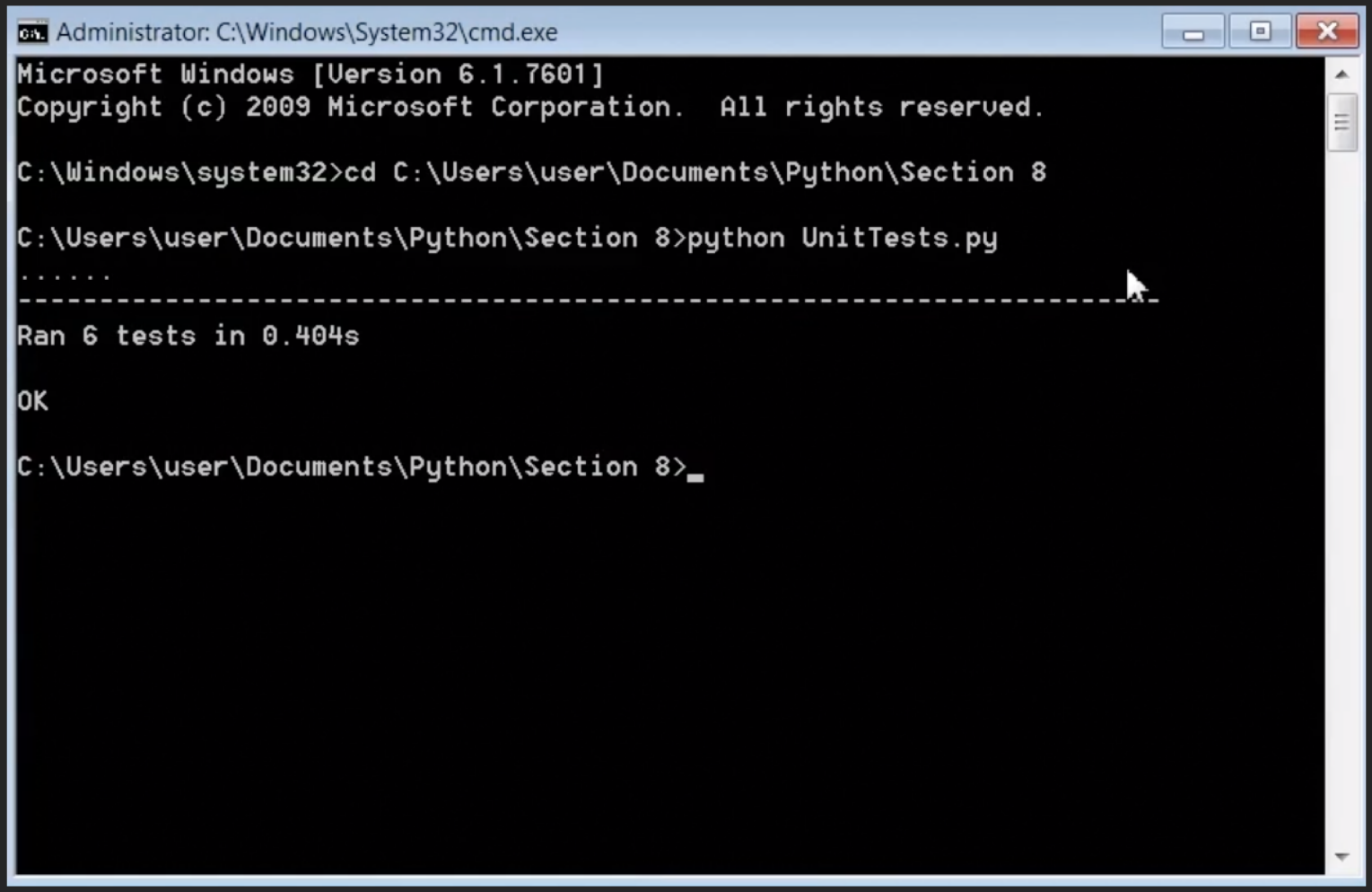
Note setUp method
won't count

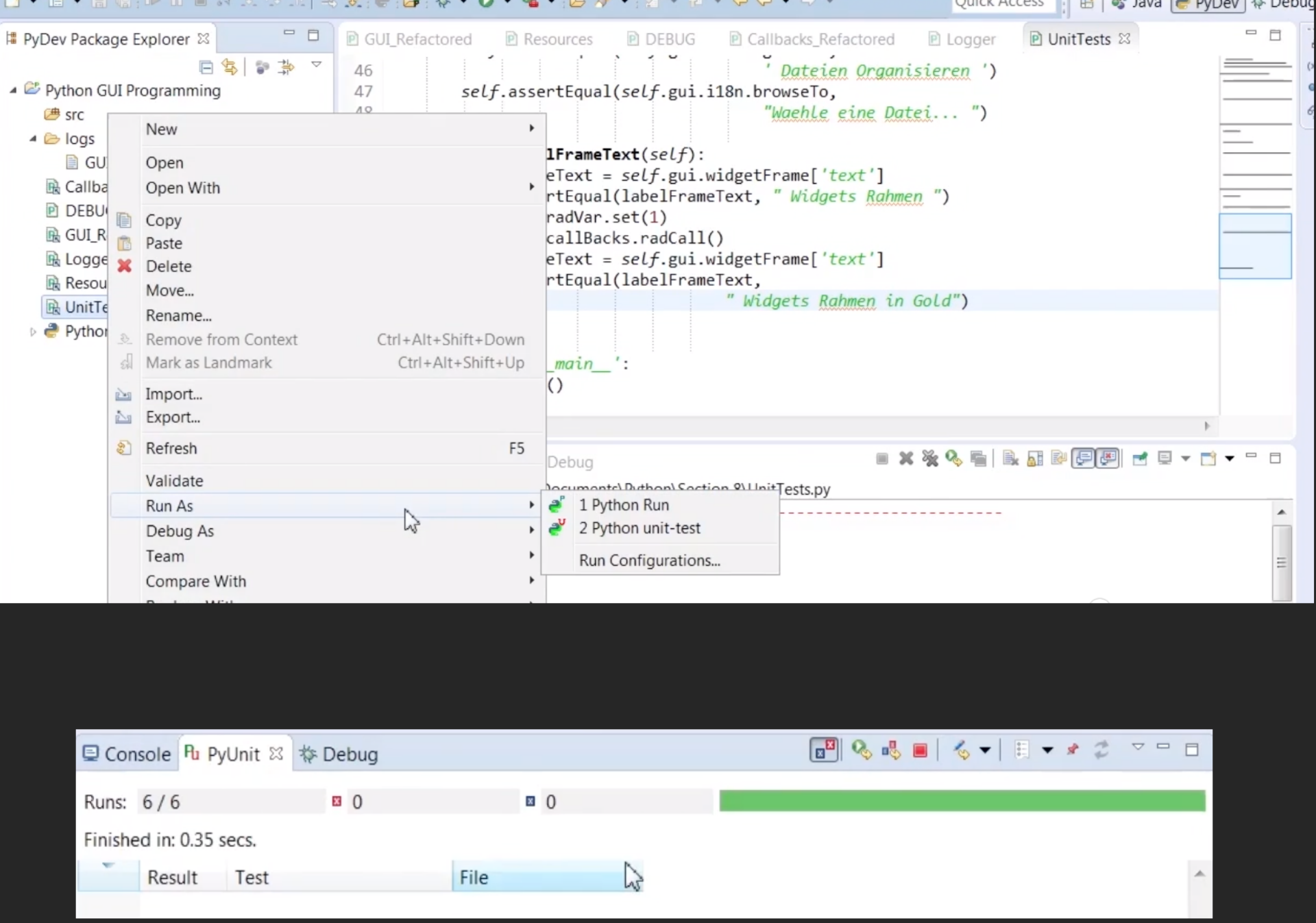
Our tests will catch
String errors for our
Labels here

Set RadioButton to 1
then invoke callback()
Same as user !

```
from GUI_Refactored import OOP as GUI
```

Note: this works as we
Import our GUI class


Administrator: C:\Windows\System32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.
C:\Windows\system32>cd C:\Users\user\Documents\Python\Section 8
C:\Users\user\Documents\Python\Section 8>python UnitTests.py
.....
Ran 6 tests in 0.404s
OK
C:\Users\user\Documents\Python\Section 8>


PyDev Package Explorer
Python GUI Programming
src
logs
New
Open
Open With
Copy
Paste
Delete
Move...
Rename...
Remove from Context Ctrl+Alt+Shift+Down
Mark as Landmark Ctrl+Alt+Shift+Up
Import...
Export...
Refresh F5
Validate
Run As
Debug As
Team
Compare With
1 Python Run
2 Python unit-test
Run Configurations...
GUI_Refactored
Resources
DEBUG
Callbacks_Refactored
Logger
UnitTests
' Dateien Organisieren ')
self.assertEqual(self.gui.i18n.browseTo,
"Waehle eine Datei... ")
lFrameText(self):
eText = self.gui.widgetFrame['text']
rtEqual(labelFrameText, " Widgets Rahmen ")
radVar.set(1)
callBacks.radCall()
eText = self.gui.widgetFrame['text']
rtEqual(labelFrameText,
" Widgets Rahmen in Gold")
_main__':
()
Debug

Console
PyUnit
Debug
Runs: 6 / 6
0
0
Finished in: 0.35 secs.
Result
Test
File

Extending the UI with

wxPython Library

postponed

PyOpenGL & Pyglet

- Displaying widget text in different languages
- Changing the entire GUI language all at once
- How Pyglet transforms our GUI more easily than PyOpenGL
- Creating a slideshow using tkinter

- Installing the PyOpenGL extension module
- Creating an example code
- Solving the challenges that come along the way

```
source activate p37

pip install pyopengl
```

PACKAGE INDEX »
Browse packages
Package submission
List trove classifiers
RSS (latest 40 updates)
RSS (newest 40 packages)
PyPI Tutorial
PyPI Security
PyPI Support
PyPI Bug Reports
PyPI Discussion
PyPI Developer Info
ABOUT »
NEWS »
DOCUMENTATION »
DOWNLOAD »
COMMUNITY »
FOUNDATION »
CORE DEVELOPMENT »

PyOpenGL 3.0.2

Standard OpenGL bindings for Python

Latest Version: 3.1.1a1

Downloads ↓

File	Type	Py Version	Uploaded on	Size
PyOpenGL-3.0.2.tar.gz (md5)	Source		2012-10-02	871KB
PyOpenGL-3.0.2.win-amd64.exe (md5) AMD64 Installer	MS Windows installer	any	2012-10-02	1MB
PyOpenGL-3.0.2.win32.exe (md5)	MS Windows installer	any	2012-10-02	1MB
PyOpenGL-3.0.2.zip (md5)	Source		2012-10-02	1MB

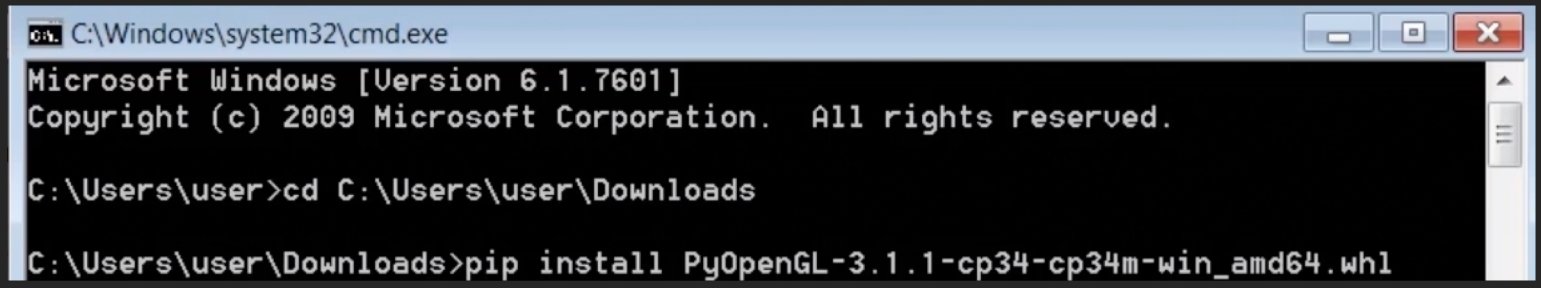
Author: Mike C. Fletcher
Home Page: http://pyopengl.sourceforge.net
Download URL: http://sourceforge.net/projects/pyopengl/files/PyOpenGL/
Keywords: Graphics,3D,OpenGL,GLU,GLUT,GLE,GLX,EXT,ARB,Mesa,ctypes
License: BSD

PyOpenGL provides bindings to OpenGL, GLUT, and GLE.

PyOpenGL-3.1.1-cp27-cp27m-win32.whl
PyOpenGL-3.1.1-cp27-cp27m-win_amd64.whl
PyOpenGL-3.1.1-cp34-cp34m-win32.whl
PyOpenGL-3.1.1-cp34-cp34m-win_amd64.whl
PyOpenGL-3.1.1-cp35-cp35m-win32.whl
PyOpenGL-3.1.1-cp35-cp35m-win_amd64.whl
PyOpenGL_accelerate-3.1.1-cp27-cp27m-win32.whl
PyOpenGL_accelerate-3.1.1-cp27-cp27m-win_amd64.whl
PyOpenGL_accelerate-3.1.1-cp34-cp34m-win32.whl
PyOpenGL_accelerate-3.1.1-cp34-cp34m-win_amd64.whl
PyOpenGL_accelerate-3.1.1-cp35-cp35m-win32.whl
PyOpenGL_accelerate-3.1.1-cp35-cp35m-win_amd64.whl

Pypmc, a toolkit for adaptive importance sampling.

pypmc-1.0-cp27-none-win32.whl
pypmc-1.0-cp27-none-win_amd64.whl
pypmc-1.0-cp34-none-win32.whl



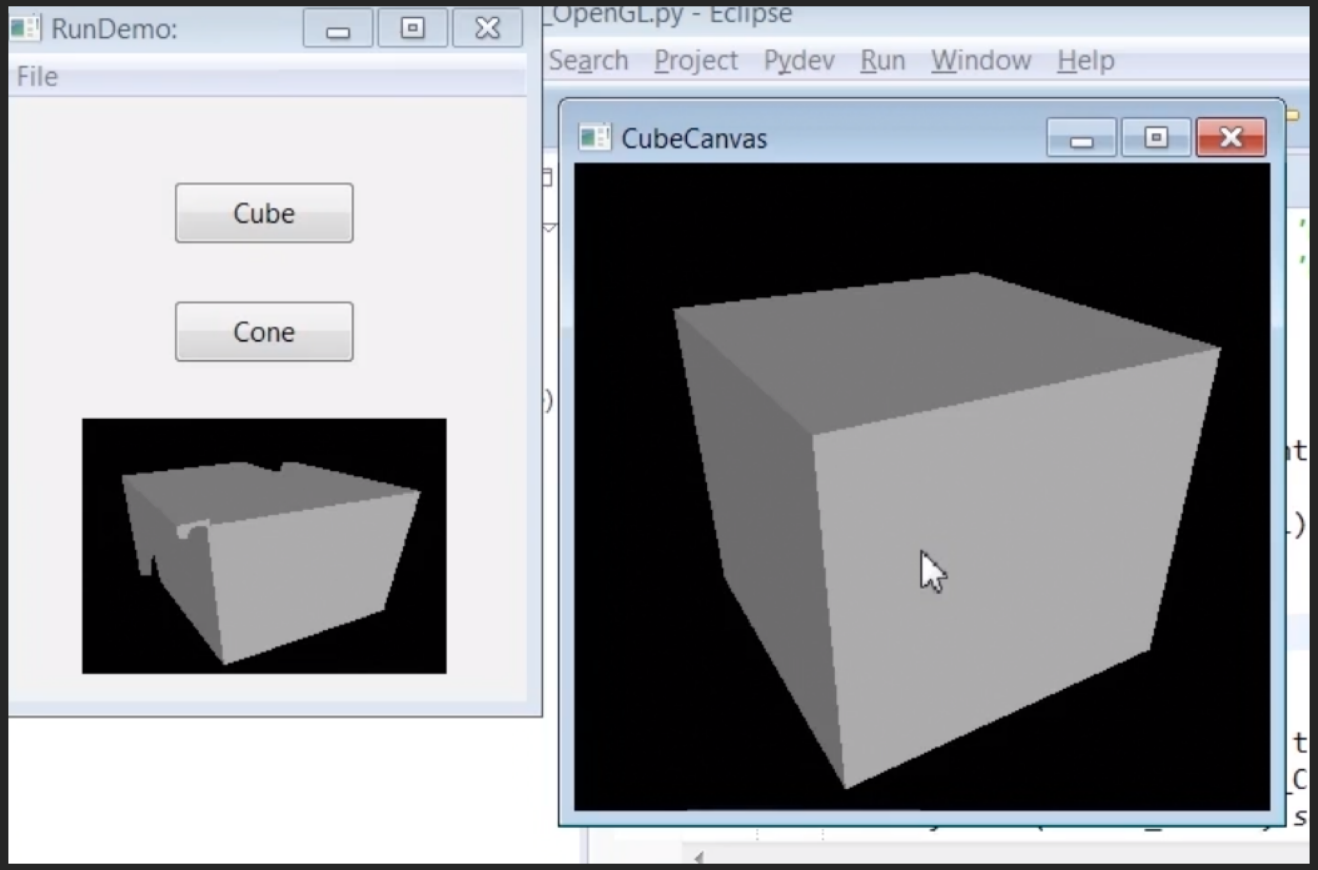
Try import_OpenGL.py

```
import wx
from wx import glcanvas
from OpenGL.GL import *
from OpenGL.GLUT import *
```

https://wiki.wxpython.org/GLCanvas

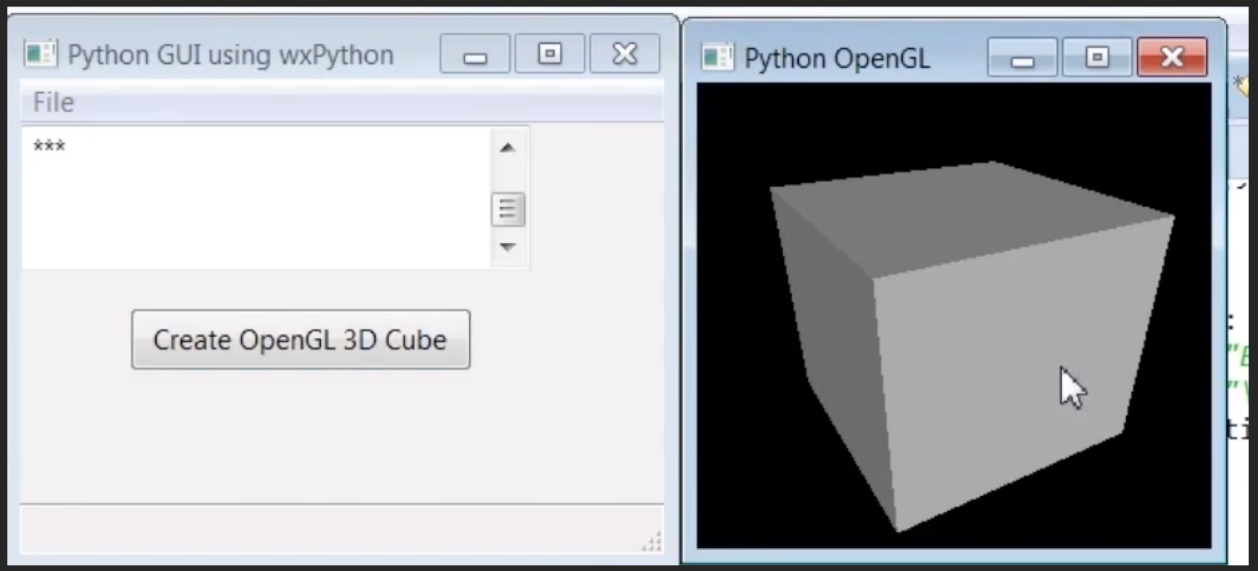
Lets copy example canvas

And run it !

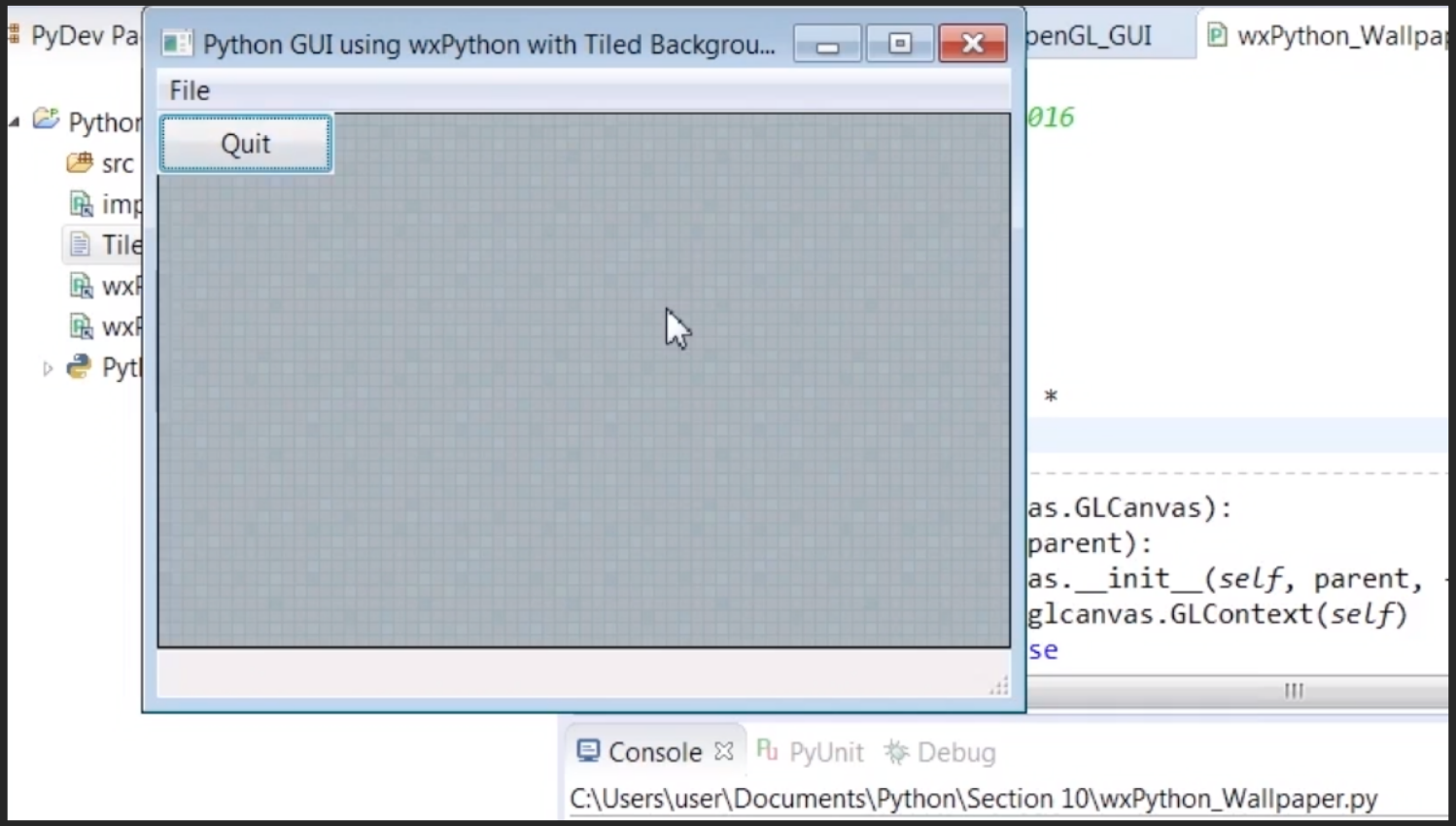

RunDemo:
File
Cube
Cone
_OpenGL.py - Eclipse
Search
Project
Pydev
Run
Window
Help
CubeCanvas

- Creating a GUI using wxPython
- How to make it pretty by using bitmaps

Run the code



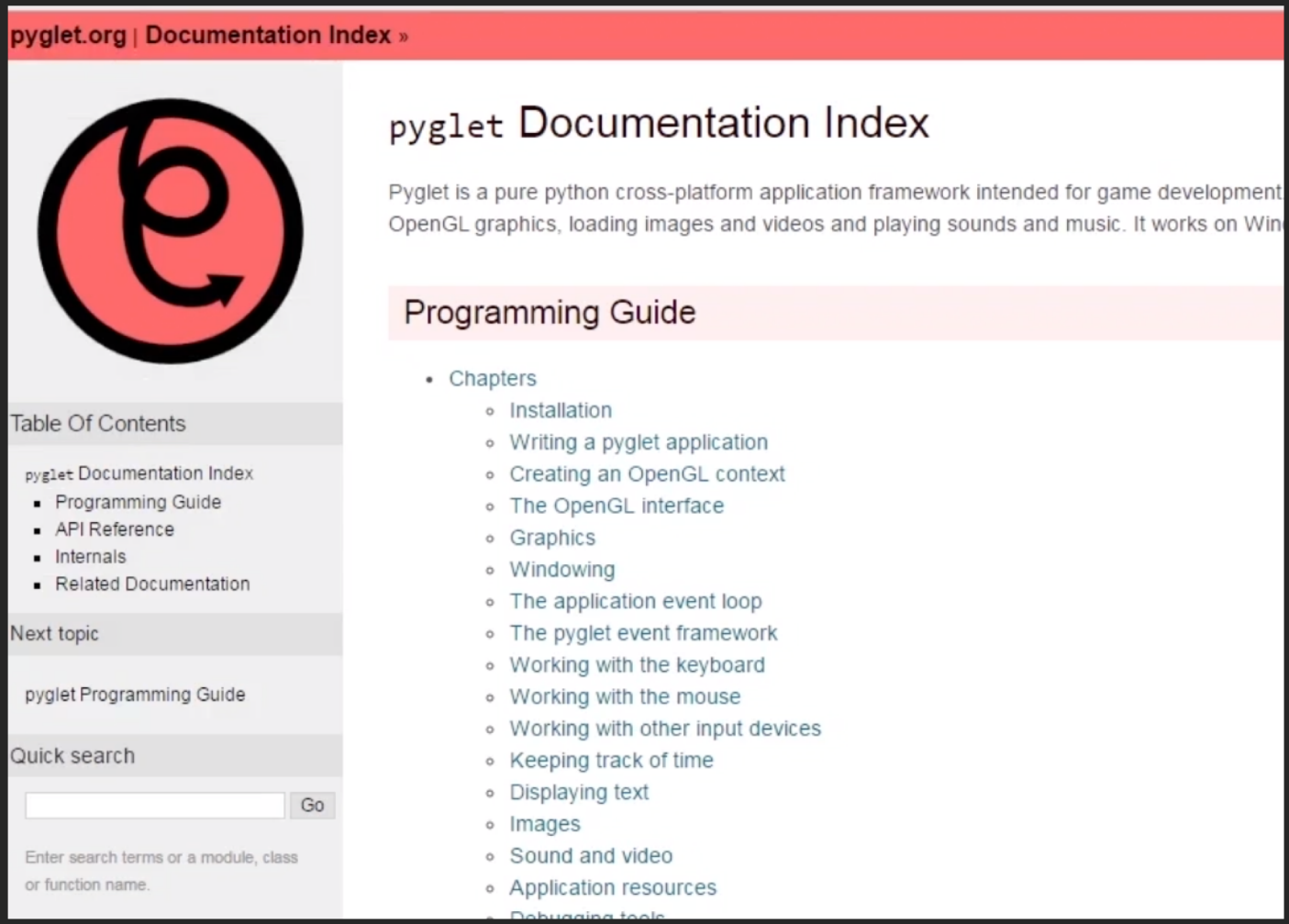
Note issues on the mac !


PyDev Pa
Python GUI using wxPython with Tiled Backgrou...
File
Quit
penGL_GUI
wxPython_Wallpa
016
*
as.GLCanvas):
parent):
as.__init__(self, parent,
glcanvas.GLContext(self)
se
Console
PyUnit
Debug
C:\Users\user\Documents\Python\Section 10\wxPython_Wallpaper.py

- The official website to install and use Pyglet
- Creating a simple GUI
- Enhancing it to 3D
- Adding some fancy colors to it

Pyglet comes with its on UI functionality
We do not need to use Tkinter !

```
pip install pyglet
```

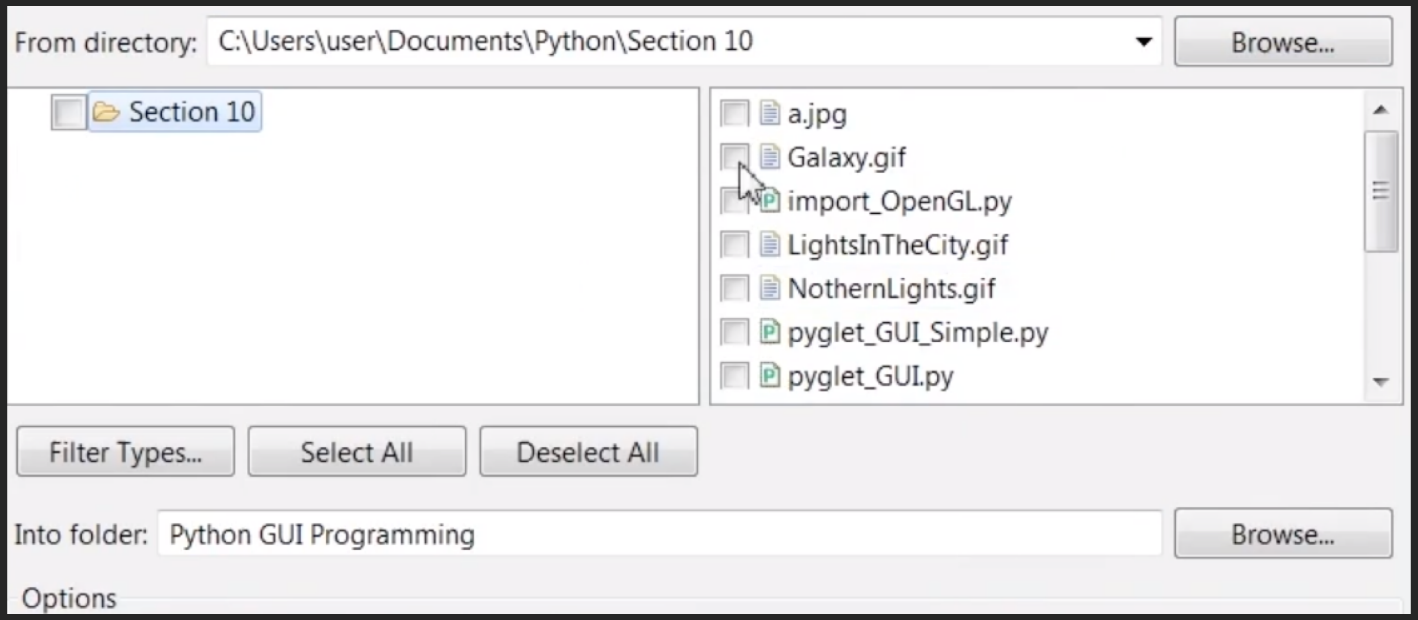


```python
import pyglet

window = pyglet.window.Window()
label = pyglet.text.Label('PyGLet GUI',
                          font_size=42,
                          x=window.width//2, y=window.height//2,
                          anchor_x='center', anchor_y='center')

@window.event
def on_draw():
    window.clear()
    label.draw()

pyglet.app.run()
```

Run it !

- Creating a working slideshow GUI using pure Python
- The limitations the core Python built-ins have
- Exploring Pillow, which extends tkinter's functionality

Import the image files




```
from PIL import Image, ImageTk
```