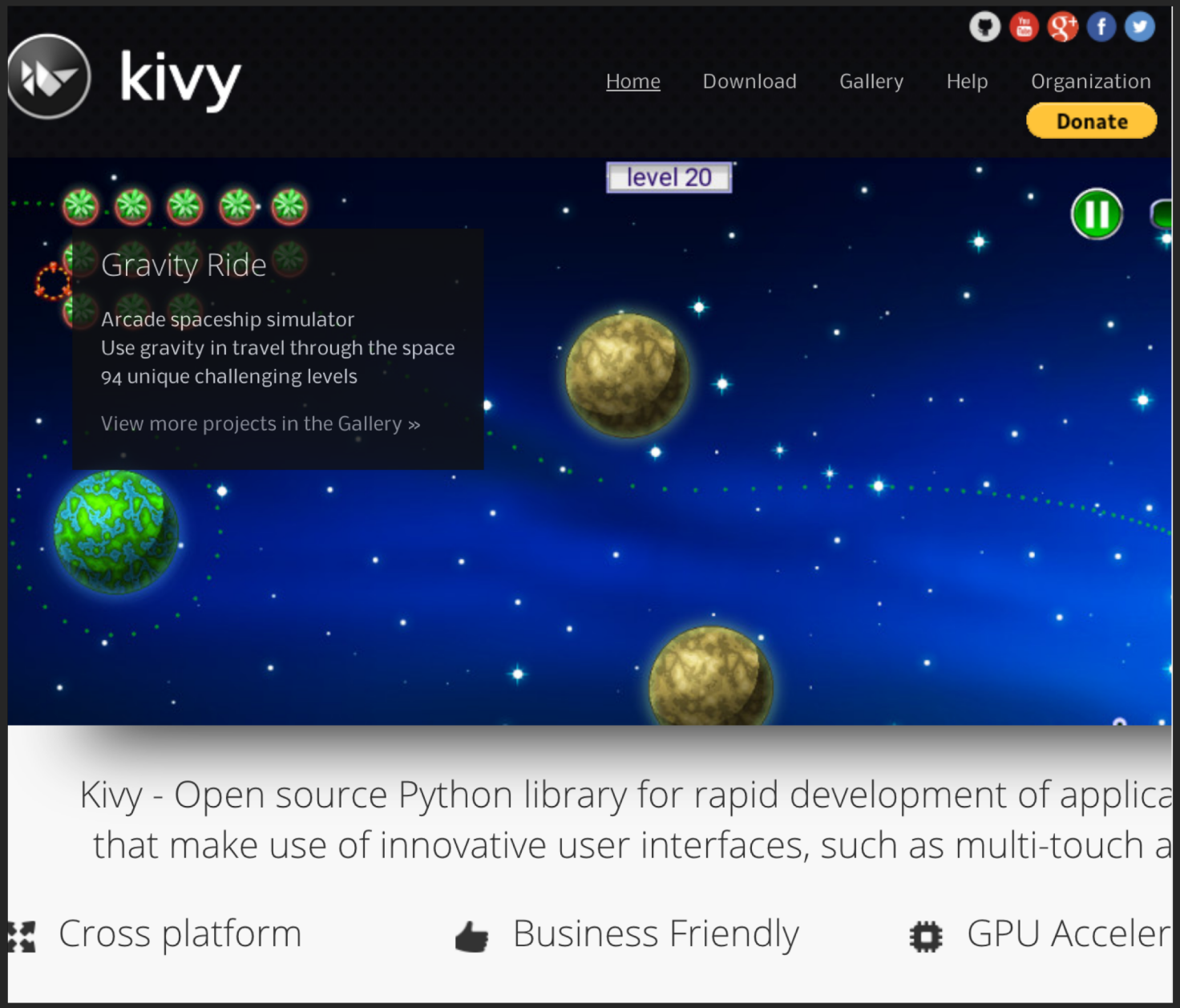
User Interface Design and Evaluation

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