Python programming + GUI

Sourangshu Bhattacharya

For More Information?

http://python.org/

- documentation, tutorials, beginners guide, core distribution, ...

Books include:

- *Learning Python* by Mark Lutz
- *Python Essential Reference* by David Beazley
- *Python Cookbook*, ed. by Martelli, Ravenscroft and Ascher
- (online at http://code.activestate.com/recipes/langs/python/)
- http://wiki.python.org/moin/PythonBooks

Python Interactive Shell

```
% python
Python 2.6.1 (r261:67515, Feb 11 2010, 00:51:29)
[GCC 4.2.1 (Apple Inc. build 5646)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

You can type things directly into a running Python session

```
>>> 2+3*4
14
>>> name = "Andrew"
>>> name
'Andrew'
>>> print "Hello", name
Hello Andrew
>>>
```

Basics

simple data types

Values of different type allow different operations

`12 + 3`	=> 15	`12 - 3`	=> 9
`"12" + "3"`	=> "123"	`"12" - "3"`	=> ERROR!
`"12"*3`	=> "121212"		

- bool: Boolean, e.g. True, False
- int: Integer, e.g. 12, 23345
- float: Floating point number, e.g. 3.1415, 1.1e-5
- string: Character string, e.g. "This is a string"
- ...

structured types

structured data types are composed of other simple or structured types

- string: 'abc'
- list of integers: [1,2,3]
- list of strings: ['abc', 'def']
- list of lists of integers: [[1,2,3],[4,5,6]]
- ...

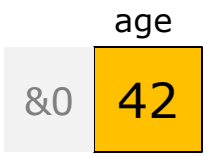
there are much more advanced data structures...

data structures are models of the objects you want to work with

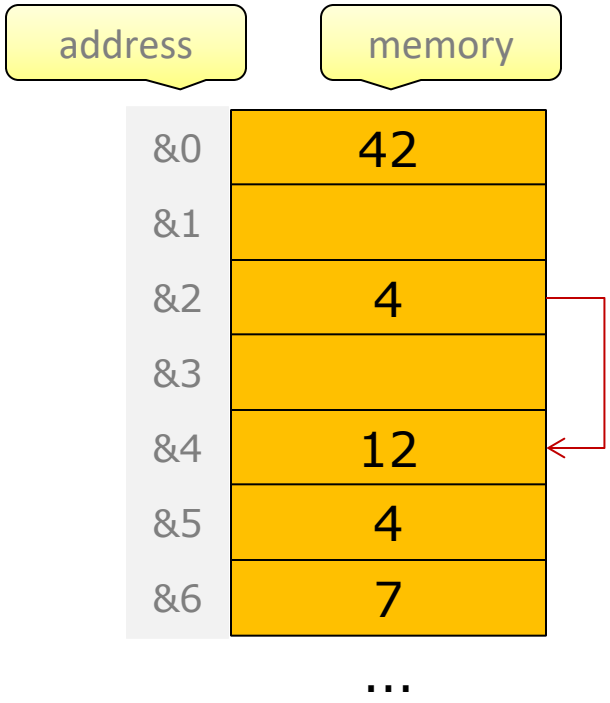
variables and references

- container for a value, name of a memory cell
- primitive values: numbers, booleans, ...
- reference values: address of a data structure, eg. a list, string, ...

```
age = 42
```




```
ages = [12,4,7]
```

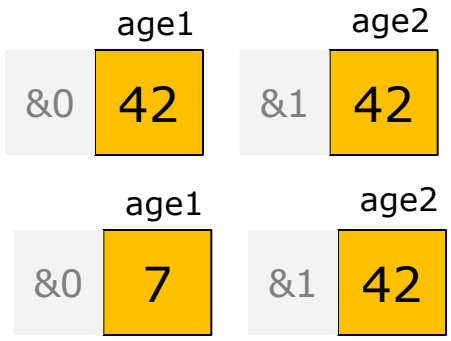


variables and references 2

```
age1 = 42
age2 = age1
```

```
age1 = 7
```

=> age2 is 42

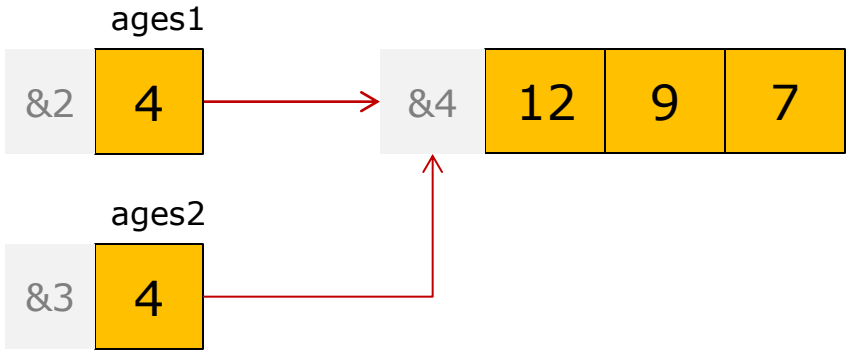


try:
id(age1)
id(age2)

```
ages1 = [12,4,7]
ages2 = ages1
```

```
ages1[1] = 9
```

=> ages2[1] is 9 !



data structures 1

- organizing data depending on task and usage pattern

- List (Array)
  - collection of (many) things
  - constant access time
  - linear search time
  - changeable (=mutable)

```
colors = ['red', 'red', 'blue']
colors[1]                          => 'red'
'blue' in colors                   => True
colors[1] = “pink”
```

- Tuple
  - collection of (a few) things
  - constant access time
  - linear search time
  - not changeable (=immutable)
    => can be used as hash key

```
address = (3, 'Queen Street')
address[0]                         => 3
3 in address                       => True
address[0] = 4
```

data structures 2

- Set
  - collection of unique things
  - no random access
  - constant search time
  - no guaranteed order of values

```
even = set([2,4,6,8])
even[1]
6 in even                    => True
for e in even: print e
```

- Dictionary (Hash)
  - maps keys to values
  - constant access time via key
  - constant search time for key
  - linear search time for value
  - no **guaranteed** order

```
tel = {'Stef':62606, 'Mel':62663}
tel['Mel']                   => 62663
'Stef' in tel                => True
62663 in tel.values()        => True
for name in tel: print name
```

data structures - tips

when to use what?

- List
  - many similar items to store
    e.g, numbers, protein ids, sequences, ...
  - no need to find a specific item fast
  - fast access to items at a specific position in the list
- Tuple
  - a few (<10), different items to store
    e.g. addresses, protein id and its sequence, ...
  - want to use it as dictionary key
- Set
  - many, unique items to store
    e.g. unique protein ids
  - need to know quickly if a specific item is in the set
- Dictionary
  - map from keys to values, is a look-up table
    e.g. telephone dictionary, amino acid letters to hydrophobicity values
  - need to get quickly the value for a key

Control and execution

how to do something

- statement
  - executes some function or operation, e.g.

```
print 1+2
```

- condition
  - describes when something is done, e.g.

```
if number > 3:
 print "greater than 3"
```

- iteration
  - describes how to repeat something, e.g.

```
for number in [1,2,3]:
 print number
```

condition

```
if condition :
    do_something
```

```
if condition :
    do_something
else:
    do_something_else
```

```
if condition :
    do_something
elif condition2 :
    do_something_else1
else:
    do_something_else2
```

```
if x < 10:
    print "in range"
```

```
if x < 5:
    print "lower range"
else:
    print "out of range"
```

```
if x < 5:
    print "lower range"
elif x < 10:
    print "upper range"
else:
    print "out of range"
```

Boolean logic

Python expressions can have “and”s and “or”s:

```
if (ben <= 5 and chen >= 10 or
chen == 500 and ben != 5):
   print “Ben and Chen“
```

iteration

```
for variable in sequence :
    do_something
```

```
while condition :
    statement1
    statement2
    …
```

```
for color in ["red","green","blue"]:
    print color
```

```
for i in xrange(10):
    print i
```

```
for char in "some text":
    print char
```

```
i = 0
while i < 10 :
    print i
    i += 1
```

functions

- break complex problems in manageable pieces
- encapsulate/generalize common functionalities

```
def function(p1,p2,...):
    do_something
    return ...
```

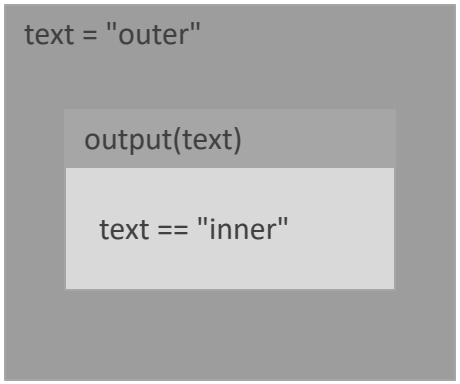
```
def add(a,b):
    return a+b
```

```
def divider():
    print "--------"
```

```
def divider(ch,n):
    print ch*n
```

```
def output(text):
    print text

text = "outer"
print text
output("inner")
print text
```

Scope of a variable



Object Oriented Programming

Object-oriented programming (OOP) is a programming paradigm that uses "objects" – data structures consisting of data fields and associated methods.

- brings data and functions together
- helps to manage complex code
- limited support in Python

Dot notation

object.attribute
object.method()

Class
- attributes
+ methods

Car
- color
- brand
+ consumption(speed)

Examples

```
text = "some text"
text.upper()
len(text) #not a method call
text.__len__()

f = open(filename)
if not f.closed :
  lines = f.readlines()
f.close()
```

try:
dir(file)
help(file)

GUI programming

What is TKinter

- Tkinter is a Python interface to the Tk graphics library.
- Tk is a graphics library widely used and available everywhere
- Tkinter is included with Python as a library. To use it:

import * from Tkinter

or

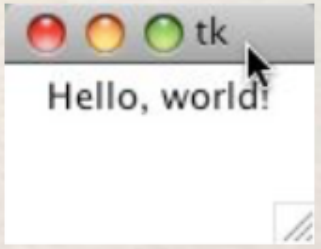
from Tkinter import *

What can it do ?

- Tkinter gives you the ability to create Windows with widgets in them
- Definition: widget is a graphical component on the screen (button, text label, drop-down menu, scroll bar, picture, etc...)
- GUIs are built by arranging and combining different widgets on the screen.

Hello world

```
# File: hello1.py
from Tkinter import *
root = Tk()  # Create the root (base) window where all widgets go
w = Label(root, text="Hello, world!") # Create a label with words
w.pack() # Put the label into the window
root.mainloop() # Start the event loop
```



Widgets are objects

- Label is a class, w is an object
- w = Label(root, text="Hello, world!")
- Call the “pack” operation:
- w.pack()
- Hint: An operation is just a function... nothing more, nothing less.. it is just defined inside the class to act upon the object’s current data.
- Objects usually hide their data from anyone else and let other programmers access the data only through operations. (This is an OO concept called encapsulation)

Buttons

```
#Button1.py
from Tkinter import *

root = Tk()  # Create the root (base) window where all widgets go

w = Label(root, text="Hello, world!") # Create a label with words
w.pack() # Put the label into the window

myButton = Button(root, text="Exit")
myButton.pack()

root.mainloop() # Start the event loop
```

But nothing happens when we push the button! Lets fix that with an event!

Callback functions

```
#Button2.py
from Tkinter import *

def buttonPushed():
    print "Button pushed!"

root = Tk()  # Create the root (base) window where all widgets go

w = Label(root, text="Hello, world!") # Create a label with words
w.pack() # Put the label into the window

myButton = Button(root, text="Exit",command=buttonPushed)
myButton.pack()

root.mainloop() # Start the event loop
```

This says, whenever someone pushes the button, call the buttonPushed function. (Generically any function called by an action like this is a "callback")

Using a text entry box

```
#Textentrybox1.py
from Tkinter import *

# Hold onto a global reference for the root window
root = None

# Hold onto the Text Entry Box also
entryBox = None

def buttonPushed():
    global entryBox
    txt = entryBox.get()
    print "The text is:",txt

def createTextBox(parent):
    global entryBox
    entryBox = Entry(parent)
    entryBox.pack()

def main():
    global root
    root = Tk()  # Create the root (base) window where all widgets go

    myButton = Button(root, text="Show Text",command=buttonPushed)
    myButton.pack()
    createTextBox(root)
    root.mainloop() # Start the event loop

main()
```

Call the get() operation on the entry box to get the text when button is pushed

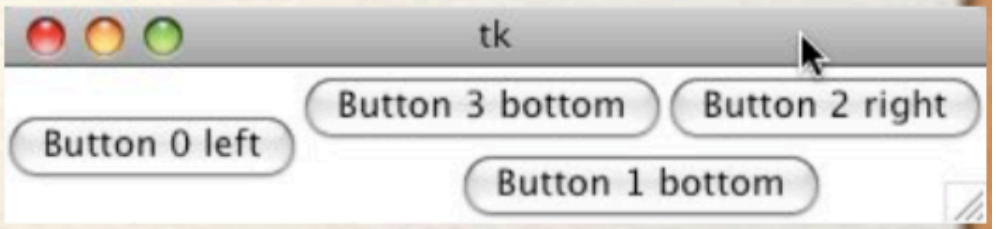
Create the global entry box!

Layout Management

- You may have noticed as we pack widgets into the window they always go under the previous widget
  - What if we want to get them to go side- by-side or some other place?
- Most windowing toolkits have layout management systems to help you arrange widgets!
- You've been using one – the packer is called when you pack()
  - pack can have a side to pack on:
  - myWidget.pack(side=LEFT)
  - this tells pack to put this widget to the left of the next widget
  - Let's see other options for pack at:
  - http://epydoc.sourceforge.net/stdlib/Tkinter.Pack-class.html#pack

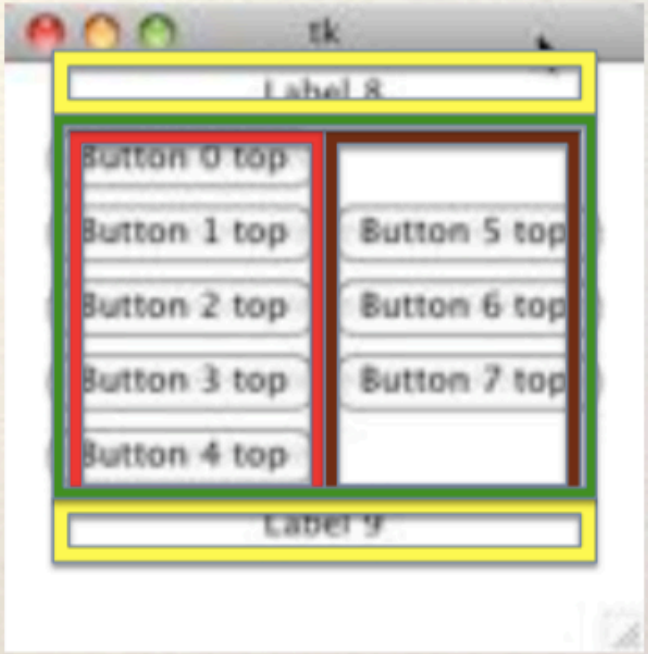
Pack Examples

```
#pack_sample.py
from Tkinter import *

# Hold onto a global reference for the root window
root = None
count = 0 # Click counter

def addButton(root, sideToPack):
    global count
    name = "Button "+ str(count) +" "+sideToPack
    button = Button(root, text=name)
    button.pack(side=sideToPack)
    count +=1

def main():
    global root
    root = Tk()  # Create the root (base) window where all widgets go
    addButton(root, LEFT) # Put the left side of the next widget close to me
    addButton(root, BOTTOM) # Put bottom of next widget close to me
    addButton(root, RIGHT) # Put right of next widget close to me
    addButton(root, BOTTOM) # Put bottom of next widget close to me
    root.mainloop() # Start the event loop

main()
```



Packing Frames

- Lets say you want this GUI



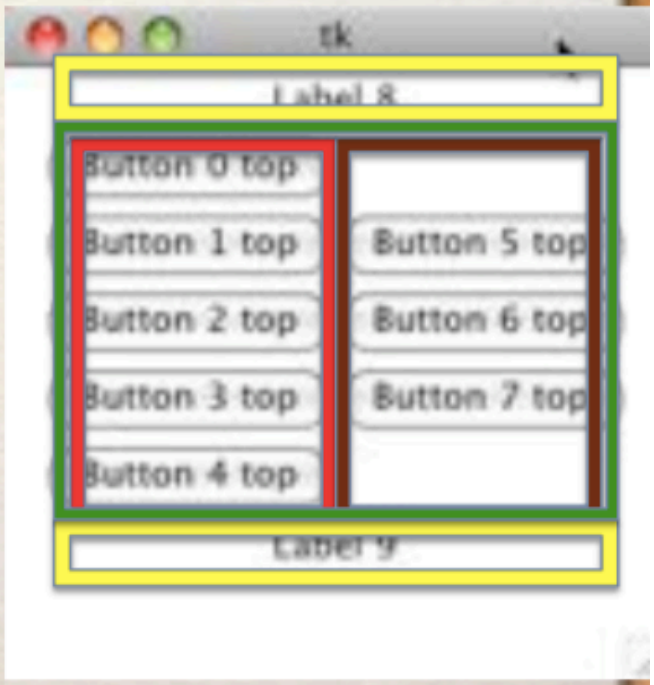
- Lets look at the frames

Packing Frames

Create Left Red frame

Create the frame like any other widget!

- frame1 = Frame(root)
- addButton(frame1 , TOP)
- addButton(frame1 , TOP)
- addButton(frame1 , TOP)
- addButton(frame1 , TOP)
- addButton(frame1 , TOP)



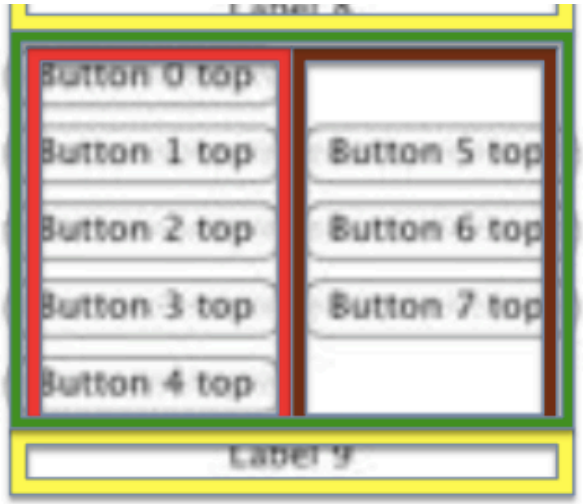
- Now you can treat the frame as one big widget!

Packing Frames

- redFrame.pack(side=LEFT)
- brownFrame.pack(side=LEFT)
- topYellow.pack(side=TOP)
- green.pack(side=TOP)
- bottomYellow.pack(side=TOP)

Who is the parent of the red and brown frames?

Ans: The green frame!



Other Geometry Managers

- Python has other geometry managers (instead of pack) to create any GUI layout you want
- grid – lets you specify a row,column grid location and how many rows and columns each widget should span
- place – specify an exact pixel location of each widget

Conclusion

- Not discussed
  - Canvas – You will find out in the assignment
  - Menus
  - Showing images
  - Capturing Mouse clicks
- Resources:
  - https://docs.python.org/3/library/tkinter.html
  - Python and Tkinter Programming Book by John Grayson
  - https://pythonbasics.org/tkinter/
  - https://www.tutorialspoint.com/python/python_gui_programming.htm