# Graphical User Interfaces

1. Object-Oriented Design of GUIs
   - a GUI to evaluate expressions
   - making colors with scale widgets
2. Visualizing `polyfit`
   - adding data points with mouse clicks
   - applying inheritance to visualize `polyfit`

MCS 507 Lecture 12
Mathematical, Statistical and Scientific Software
Jan Verschelde, 24 September 2012

# Graphical User Interfaces

1. Object-Oriented Design of GUIs
   - a GUI to evaluate expressions
   - making colors with scale widgets
2. Visualizing `polyfit`
   - adding data points with mouse clicks
   - applying inheritance to visualize `polyfit`

# evaluating expressions


The GUI shown above uses the following widgets:

- 2 label widgets to document entry widgets,
- 2 entry widgets to enter inputs & 1 to display results,
- 1 button to perform the evaluation.

Object-oriented design of the GUI:

- `__init__` defines the layout of the GUI,
- actions of the GUI are implemented by the methods.

# structure of guieval.py

```
from Tkinter import *
from math import *

class EvalFun():
   """
   GUI to evaluate user given expressions.
   """
   def __init__(self,wdw):
      "Determines the layout of the GUI."

   def calc(self):
      "Evaluates the function f at x."

def main():
   top = Tk()
   eva = EvalFun(top)
   top.mainloop()

if __name__ == "__main__": main()
```

# label, entry & grid manager

```
def __init__(self,wdw):
   "Determines the layout of the GUI."
   wdw.title("Evaluate Expressions")
   self.L1 = Label(wdw,text="f(x) =")
   self.L1.grid(row=0)
   self.L2 = Label(wdw,text="x =")
   self.L2.grid(row=0,column=2)
   self.f = Entry(wdw)
   self.f.grid(row=0,column=1)
   self.e = Entry(wdw)
   self.e.grid(row=0,column=3)
   self.r = Entry(wdw)
   self.r.grid(row=0,column=5)
```

The `grid` defines the placement of a widget in the window.
Rows and columns start at zero.

# the button

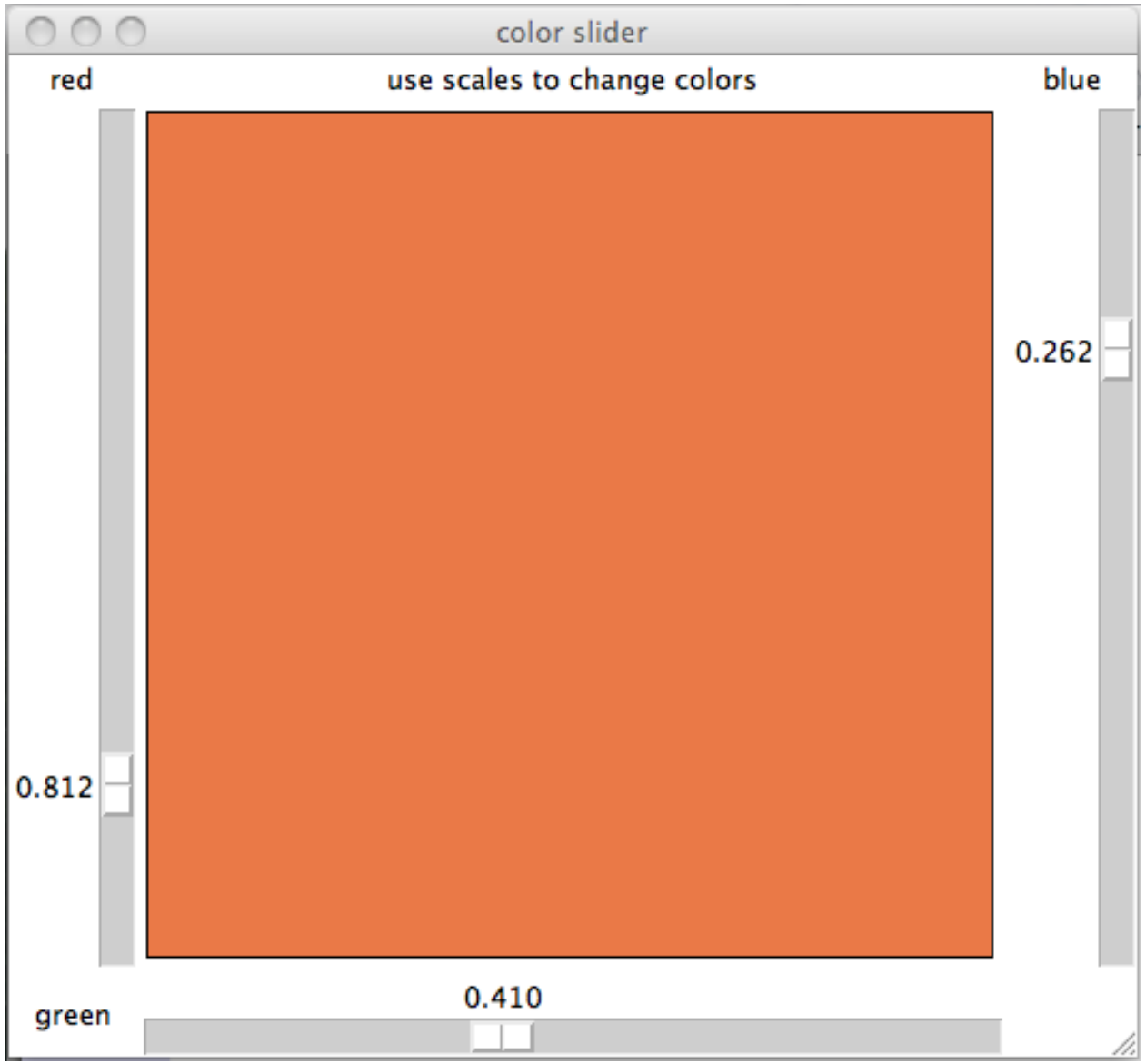
```
def __init__(self,wdw):
    ...
    self.b = Button(wdw,text="equals",
       command=self.calc)
    self.b.grid(row=0,column=4)

def calc(self):
    "Evaluates the function f at x."
    self.r.delete(0,END)
    x = float(self.e.get())
    y = eval(self.f.get())
    self.r.insert(INSERT,y)
```

Note: `calc` is defined *after* the button layout.
What comes from an entry widget has type `string`.

# Graphical User Interfaces

1. Object-Oriented Design of GUIs
   - a GUI to evaluate expressions
   - making colors with scale widgets
2. Visualizing `polyfit`
   - adding data points with mouse clicks
   - applying inheritance to visualize `polyfit`

# mixing red, green, blue



# code for one scale

- The variable set by the scale is a double, ranging from 0 to 1 with resolution `1.0/256`.
- After every change in the variable, the method `ShowColors` is executed.
- When the GUI starts up, the scale has value `0.5`.

```
def __init__(self,wdw):
   ...
   self.r = DoubleVar() # red intensity
   self.sr = Scale(wdw,orient='vertical',
      length=self.d,
      from_=0.0,to=1.0, resolution = 1.0/256,
      variable=self.r,command=self.ShowColors)
   self.sr.set(0.5) # initial value of scale
```

# showing colors

```
def ShowColors(self,v):
    """
    Displays a rectangle, filled with rgb color.
    """
    r = self.sr.get()
    g = self.sg.get()
    b = self.sb.get()
    print 'r = %f, g = %f, r = %f' % (r,g,b)
    hr = '%.2x' % int(255*r)
    hg = '%.2x' % int(255*g)
    hb = '%.2x' % int(255*b)
    color = '#' + hr + hg + hb
    x = self.d/2+1; y = self.d/2+1; d = self.d/2-3
    self.c.delete("box")
    self.c.create_rectangle(x-d,y-d,x+d,y+d,width=1,
       outline='black',fill=color,tags='box')
```

# explaining `ShowColors`

Key aspects of the method `ShowColors`:

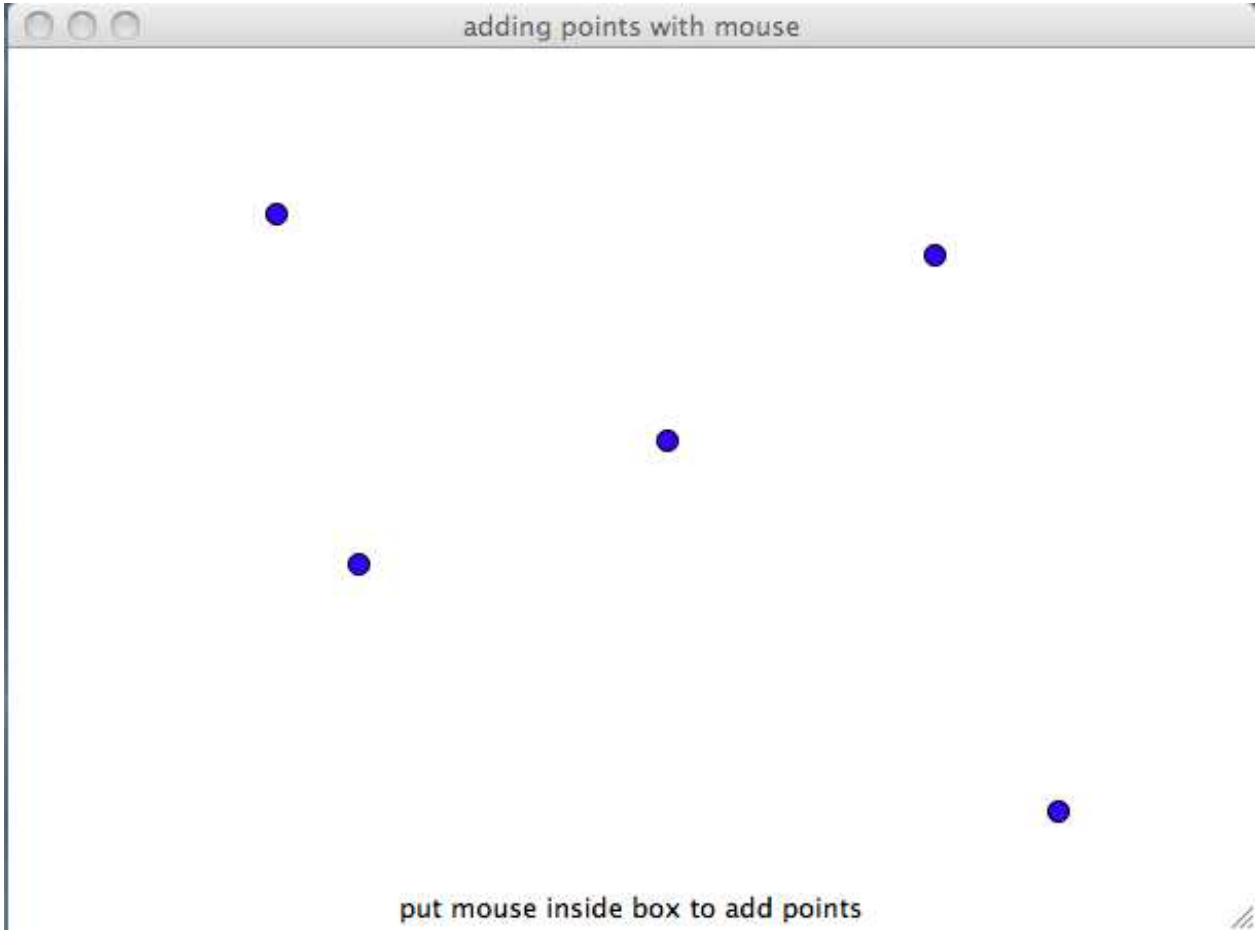
- The argument `v` of `ShowColors` is the value of the scale, but we need the values of all three intensities.
- With `self.sr.get()` we get the red intensity.
  Green and blue intensities are set by the scales with names `sg` and `sb` respectively.
- The `print` writes to the terminal.
- `hr = '%.2x' % int(255*r)` converts the intensity as a float in $[0, 1]$ to a two-digit hexadecimal integer.
- The large rectangle written to canvas has tag `box` and with this name we can wipe out the previous color.

# Graphical User Interfaces

1. Object-Oriented Design of GUIs
   - a GUI to evaluate expressions
   - making colors with scale widgets

2. Visualizing `polyfit`
   - adding data points with mouse clicks
   - applying inheritance to visualize `polyfit`

# adding points



# functionality of the GUI

Two main functions of the GUI:

- add points by clicking mouse on canvas, and
- delete points by clicking on displayed point.

Points on canvas are pixels, but picking out a pixel for deletion can be very hard on user.

Points are disks of radius 10 on canvas.

On a canvas with dimensions $400 \times 600$, we imagine our canvas as composed of 40 rows and 60 columns.

Pixels are mapped as $(248, 141) \rightarrow (25, 14)$.

# optional output

Launching: `$ python mouseptsadd.py output.`

```
def main():
    top = Tk()
    r = 40; c = 60; o = False
    import sys
    if(len(sys.argv) > 1):
       if sys.argv[1] == 'output': o = True
    show = AddPoints(top,r,c,o)
    top.mainloop()

if __name__ == "__main__": main()
```

# binding mouse events

```
from Tkinter import *

class AddPoints():
    """
    Adding points on canvas with mouse clicks.
    """
    def BindMouseEvents(self):
        """
        Binds mouse events to the canvas,
        called at the initialization of the GUI.
        """
        self.c.bind("<Button-1>",self.ButtonPressed)
        self.c.bind("<ButtonRelease-1>",
            self.ButtonReleased)
        self.c.bind("<Enter>",self.EnteredWindow)
        self.c.bind("<Leave>",self.ExitedWindow)
        self.c.bind("<B1-Motion>",self.MouseDragged)
```

# the constructor

```
def __init__(self,wdw,r,c,output=False):
    """
    The window has one column, two rows:
    + row 1: a canvas to draw points
    + row 2: a label to display coordinates
             and messages to the user.
    """
    wdw.title("adding points with mouse")
    self.mag = 10   # magnification factor
    self.rows = r   # number of rows on canvas
    self.cols = c   # number of columns on canvas
    self.c = Canvas(wdw,width=self.mag*self.cols,
       height=self.mag*self.rows,bg='white')
    self.c.grid(row=0,column=0)
```

# __init__ continued

```
# to display mouse position :
self.MousePosition = StringVar()
self.MousePosition.set\
   ("put mouse inside box to add points")
self.PositionLabel = Label(wdw,
   textvariable = self.MousePosition)
self.PositionLabel.grid(row=1,column=0)
# bind mouse events
self.BindMouseEvents()
self.points = []
self.output = output
```

# mapping pixels

```
def MapPixel(self,p):
    """
    Maps pixel p working mod self.mag.
    If self.mag equals 10, then
    MapPixel(248) returns 25,
    MapPixel(141) returns 14.
    """
    m = self.mag
    (x,r) = divmod(p,m)
    return (x+1 if r > m/2 else x)
```

# showing points

```
def DrawCircle(self,i,j):
   """
   Draws a blue circle on canvas with coordinates
   given at (i,j) by mouse and adds or removes
   coordinates to the list of points.
   """
   if self.output: print 'getting i =', i, 'j =', j
   (x,y) = (self.MapPixel(i),self.MapPixel(j))
   i0 = x*self.mag-self.mag/2; i1 = i0 + self.mag
   j0 = y*self.mag-self.mag/2; j1 = j0 + self.mag
   name = '('+str(x)+','+str(y)+')'
   if not (x,y) in self.points:
      self.c.create_oval(i0,j0,i1,j1,
         fill="blue",tags=name)
      self.points.append((x,y))
   else:
      self.c.delete(name)
      self.points.remove((x,y))
   if self.output: print 'list of points :', self.points
```

# dragging the mouse

The method `MouseDragged` is invoked when the mouse is dragged. The pixel coordinates are passed and displayed in the label widget associated to the mouse.

```
def MouseDragged(self,event):
   """
   Displays coordinates of moving mouse.
   """
   self.MousePosition.set("dragging at [ " + \
      str(event.x) + ", " + str(event.y) + " ]" + \
      " release to draw")
```

# button pressed & released

```
def ButtonPressed(self,event):
   """
   Displays coordinates of button pressed.
   """
   self.MousePosition.set("currently at [ " + \
      str(event.x) + ", " + str(event.y) + " ]" + \
      " release to fill, or drag")

def ButtonReleased(self,event):
   """
   Displays coordinates of button released.
   """
   self.MousePosition.set("drawn at [ " + \
      str(event.x) + ", " + str(event.y) + " ]" + \
      " redo to clear")
   self.DrawCircle(event.x,event.y)
```

# enter & exit window

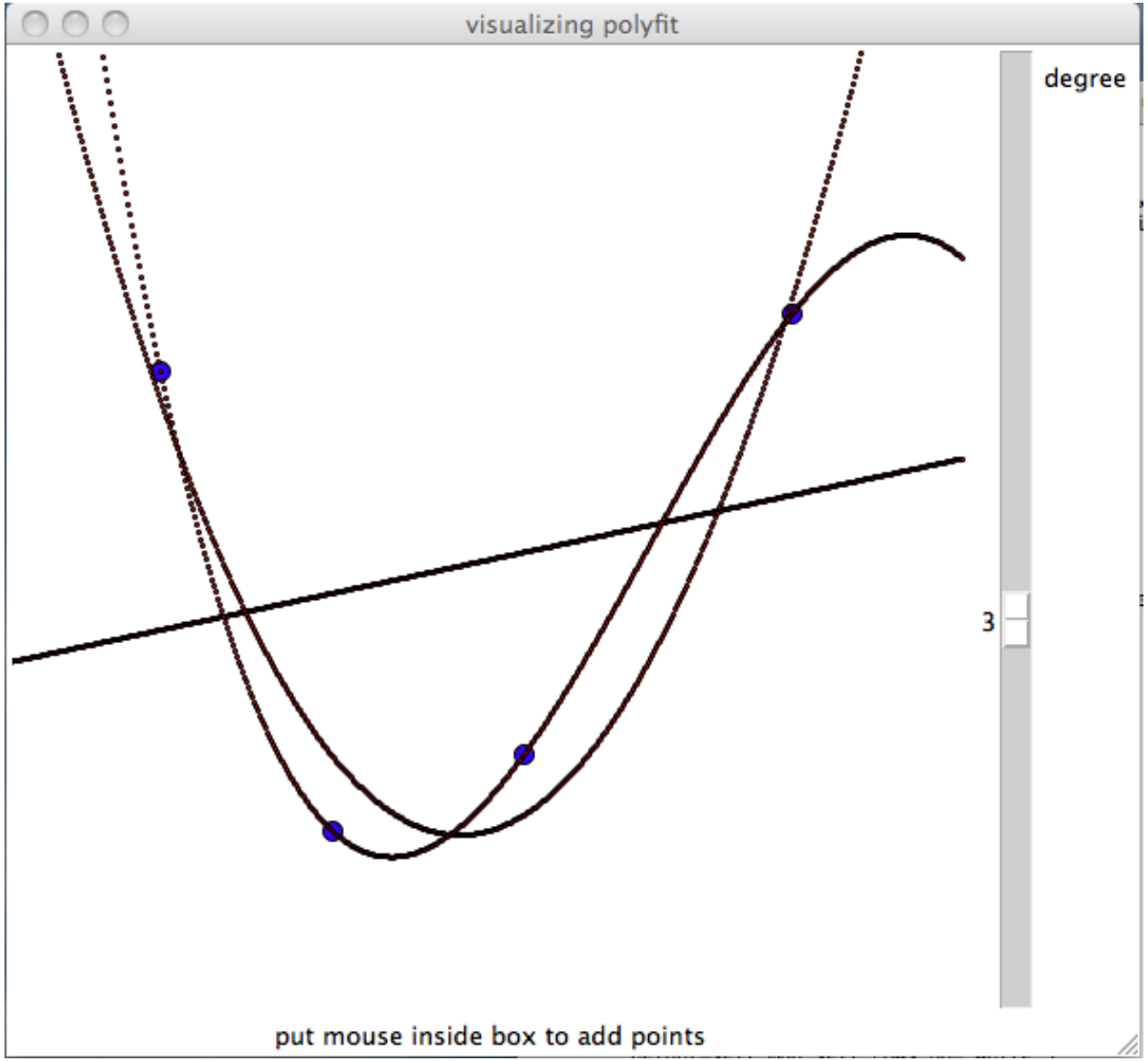
```
def EnteredWindow(self,event):
   """
   Displays message that mouse entered window.
   """
   self.MousePosition.set\
      ("press mouse to give coordinates")

def ExitedWindow(self,event):
   """
   Displays message that mouse exited window.
   """
   self.MousePosition.set\
      ("put mouse inside box to add points")
```

# Graphical User Interfaces

1. Object-Oriented Design of GUIs
   - a GUI to evaluate expressions
   - making colors with scale widgets

2. Visualizing `polyfit`
   - adding data points with mouse clicks
   - applying inheritance to visualize `polyfit`

# visualizing `polyfit`



# applying inheritance

We do not want to copy all code from `mouseptsadd.py`.

```
from mouseptsadd import AddPoints

class FitPoints(AddPoints):
    """
    Visualizing polyfit.
    """
    def __init__(self,wdw,r,c,output=False):
```

Instantiating `AddPoints` copies the layout and functionality, `FitPoints` inherits from `AddPoints`.

# constructor in `FitPoints`

```
def __init__(self,wdw,r,c,output=False):
    """
    We instantiate the AddPoints GUI
    and add a scale for the degree of
    the polynomial that fits the points.
    """
    self.addpts = AddPoints(wdw,r,c,output)
    wdw.title("visualizing polyfit")
    # define the scale next to the canvas
    self.degree = IntVar()
    self.fitdeg = Scale(wdw,orient='vertical',
       length=r*self.addpts.mag,label='degree',
       from_=0,to=5,resolution=1,
       variable=self.degree,command=self.Fit)
    self.fitdeg.grid(row=0,column=1)
```

# functionality of `FitPoints`

Three actions:

1. Computing the polynomial fitting the data points,
   with the degree entered by the scale.
2. Sampling the polynomial that fits the data
   and displaying the graph of the polynomial.
3. Deleting the graph is needed
   as the user changes the location of the data points.

Every pixed plotted of the polynomial is named `fitd-xxx` where `d` is the degree and `xxx` the number of the sample.

# plotting the polynomial

```
def ShowFit(self,p,d):
    """
    Displays the fitting polynomial p
    of degree d.
    """
    ap = self.addpts
    name = 'fit' + str(d) + "-"
    for i in xrange(0,ap.rows*ap.mag):
        x = float(i)/ap.mag
        y = np.polyval(p,x)
        j = y*ap.mag
        name = name + str(i)
        ap.c.delete(name)
        ap.c.create_oval(i-1,j-1,i+1,j+1,
            fill="red",tags=name)
```

# erasing a plot

```
def DeleteFit(self,d):
    """
    Deletes the fitting polynomial p
    of degree d.
    """
    ap = self.addpts
    name = 'fit' + str(d) + "-"
    for i in xrange(0,ap.rows*ap.mag):
        name = name + str(i)
        ap.c.delete(name)
```

# computing the fit

```
def Fit(self,v):
    """
    Calls polyfit and displays
    the fitting polynomial.
    """
    ap = self.addpts
    d = self.degree.get()
    L = ap.points
    if ap.output:
       print 'the points :', L
       print 'the degree :', d
    for i in xrange(len(L),5):
       self.DeleteFit(i)
```

# Fit continued

```
if(len(L) > d):
   A = np.array([x for (x,y) in L])
   B = np.array([y for (x,y) in L])
   p = np.polyfit(A,B,d)
   if ap.output:
      print 'fitting polynomial = '
      print p
   self.ShowFit(p,d)
```

# Summary + Exercises

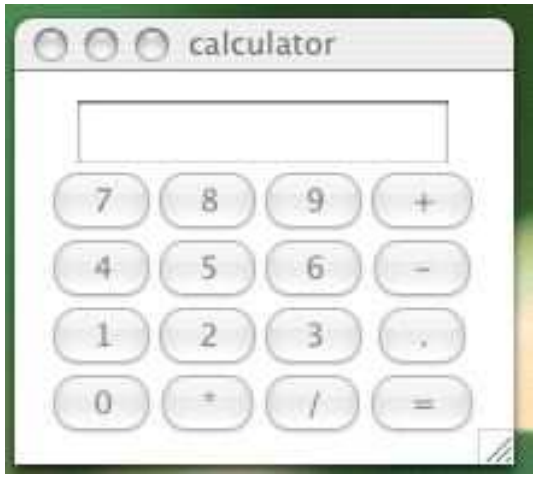
A manual of Tkinter is at
`http://infohost.nmt.edu/tcc/help/pubs/tkinter.pdf`

Exercises:

1. Add buttons `random` and `clear` to the GUI to add points with mouse clicks. When pressed, the button `random` adds a random point to the list and shows it, while the `clear` button clears the canvas and clears the list of stored points.
2. To `rgbgui.py`, add an entry widget to display the code for the color set by the scales.

# more exercises

3. Write Python code to display:




   You should not provide any functionality.

4. Add functionality to the calculator displayed in the previous exercise.

# one last exercise

5. Add a button and an entry widget to the `guifit.py`. Pressing the button generates as many random points as the value of the entry widget. Consider the fitting polynomial for increasing degrees, i.e.: explore what happens if the degree of the scale is set higher.

The third homework is due on Friday 5 October:
solve exercises 2 and 3 of Lecture 8; exercises 3 and 5 of Lecture 9; exercises 3 and 4 of Lecture 10; exercises 2 and 3 of Lecture 11; and exercises 1 and 5 of Lecture 12.