

# expression evaluation and graphing on canvas

- 1 Evaluation of Expressions
  - a graphical calculator
  - object oriented version of the GUI
- 2 Graphing Expressions
  - the Canvas widget
  - sampling and plotting expressions
- 3 Binding Keys to Canvas
  - binding arrow keys
- 4 Summary + Assignments

MCS 260 Lecture 31  
Introduction to Computer Science  
Jan Verschelde, 30 March 2016

# expression evaluation graphing on Canvas

- 1 Evaluation of Expressions
  - a graphical calculator
  - object oriented version of the GUI
- 2 Graphing Expressions
  - the Canvas widget
  - sampling and plotting expressions
- 3 Binding Keys to Canvas
  - binding arrow keys
- 4 Summary + Assignments

# Evaluation of Expressions

Thanks to Python's dynamic typing we can build a much more flexible scientific calculator.

```
$ python dyneval.py
f(x) = 3*cos(x) - sin(x)
x = 1.32
3*cos(x) - sin(x) at x = 1.32
equals -0.224188745161
```

Four steps:

- 1 read the expression for  $f(x)$ ,
- 2 read the value for  $x$ ,
- 3 compute the value for the expression  $f$  at  $x$ ,
- 4 print the result.

## the code in the script `dyneval.py`

```
"""
```

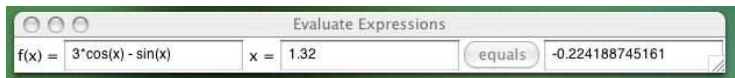
```
Thanks to Python's dynamic typing,  
expressions may be evaluated dynamically  
as well, via the "eval" command.
```

```
Any valid Python expression the user  
enters will be evaluated at the given x.
```

```
"""
```

```
from math import cos, sin  
E = input('f(x) = ')  
x = float(input('x = '))  
Y = eval(E)  
print(E + ' at x = ' + str(x))  
print('equals ' + str(Y))
```

# a GUI for the evaluator



Widgets used in this GUI:

- 1 Two Label widgets:  $f(x) =$  and  $x =$  .
- 2 Three Entry widgets:
  - 1 for the expression, after  $f(x) =$  ;
  - 2 for the value of  $x$ , after  $x =$  ;
  - 3 and the result, after the `equals`.
- 3 One Button, with the text `equals`.

Clicking on the `equals` will trigger a callback function to evaluate the function  $f(x)$  at  $x$ .

# the code for the GUI

```
from tkinter import Tk, Label, Entry, Button
from tkinter import END, INSERT
from math import cos, sin

TOP = Tk()
TOP.title("Evaluate Expressions")
Label(TOP, text="f(x) =").grid(row=0)
F = Entry(TOP)
F.grid(row=0, column=1)
Label(TOP, text="x =").grid(row=0, column=2)
E = Entry(TOP)
E.grid(row=0, column=3)
R = Entry(TOP)
R.grid(row=0, column=5)
```

## the callback function – the code continued

Clicking on the button `equals` triggers the evaluation of the function, done by a call to `calc`:

```
def calc():
    "evaluates the function"
    R.delete(0, END)
    x = float(E.get())
    result = eval(F.get())
    R.insert(INSERT, result)

B = Button(TOP, text="equals", command=calc)
B.grid(row=0, column=4)
TOP.mainloop()
```

**Advantage:** still relatively short code...

**Disadvantage:** layout intermingled with calculations.

# expression evaluation

## graphing on Canvas

- 1 Evaluation of Expressions
  - a graphical calculator
  - **object oriented version of the GUI**
- 2 Graphing Expressions
  - the Canvas widget
  - sampling and plotting expressions
- 3 Binding Keys to Canvas
  - binding arrow keys
- 4 Summary + Assignments

## object oriented version

```
class EvalFun(object):
    """
    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():
    """
    Instantiates the EvalFun object
    and starts the event loop.
    """
```

## code for `__init__`

```
def __init__(self, wdw):  
    """  
    Determines the layout of the GUI.  
    """  
    wdw.title("Evaluate Expressions")  
    self.lb1 = Label(wdw, text="f(x) =")  
    self.lb1.grid(row=0)  
    self.lb2 = Label(wdw, text="x =")  
    self.lb2.grid(row=0, column=2)  
    self.fun = Entry(wdw)  
    self.fun.grid(row=0, column=1)  
    self.ent = Entry(wdw)  
    self.ent.grid(row=0, column=3)  
    self.res = Entry(wdw)  
    self.res.grid(row=0, column=5)  
    self.btt = Button(wdw, text="equals", \  
        command=self.calc)  
    self.btt.grid(row=0, column=4)
```

## code for calc and main

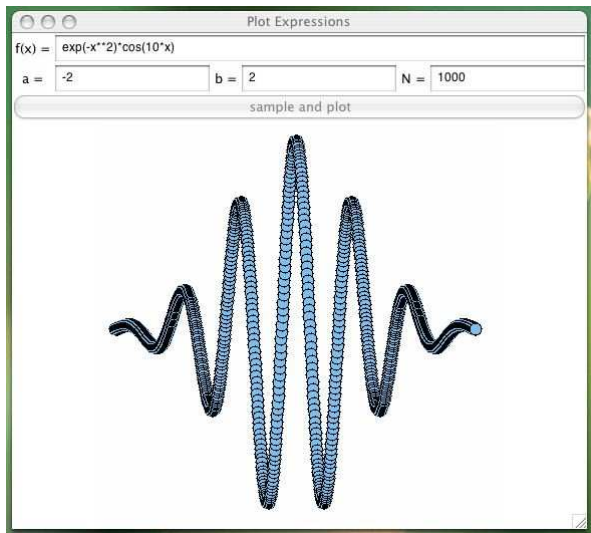
```
def calc(self):  
    """  
    Evaluates the function f at x."  
    """  
    self.res.delete(0, END)  
    x = float(self.ent.get())  
    result = eval(self.fun.get())  
    self.res.insert(INSERT, result)
```

```
def main():  
    """  
    Instantiates the EvalFun object  
    and starts the event loop.  
    """  
    top = Tk()  
    EvalFun(top)  
    top.mainloop()
```

# expression evaluation graphing on Canvas

- 1 Evaluation of Expressions
  - a graphical calculator
  - object oriented version of the GUI
- 2 Graphing Expressions
  - **the Canvas widget**
  - sampling and plotting expressions
- 3 Binding Keys to Canvas
  - binding arrow keys
- 4 Summary + Assignments

# graphing expressions



Uses the Canvas widget.

# expression evaluation

## graphing on Canvas

- 1 Evaluation of Expressions
  - a graphical calculator
  - object oriented version of the GUI
- 2 Graphing Expressions
  - the Canvas widget
  - sampling and plotting expressions
- 3 Binding Keys to Canvas
  - binding arrow keys
- 4 Summary + Assignments

## sampling an expression: `samplefun.py`

```
from math import cos, sin, exp
print('sampling a function in [a, b]')
E = input('f(x) = ')
A = float(input('a = '))
B = float(input('b = '))
N = int(input('#samples = '))
DX = (B-A)/(N+1)
x = A
S = []
for k in range(N):
    Y = eval(E)
    S.append(Y)
    print('x = %.4e' % x + \
          '    f(x) = %.15e' % Y)
    x = x + DX
print('min = %.15e' % min(S))
print('max = %.15e' % max(S))
```

# Scaling the Samples

to fit in a window

Sampled values give coordinates  $(x, y = f(x))$ ,  
where  $x \in [a, b]$  and  $m \leq y \leq M$ .

Problem: a canvas is represented by pixels  $(p, q)$ .

We must use  $(p, q) \in [L, U] \times [L, U]$

to represent points  $(x, y) \in [a, b] \times [m, M]$ .

Scale the coordinates, using the map

$$\phi : [a, b] \rightarrow [L, U] : x \mapsto \frac{U - L}{b - a}(x - a) + L$$

Observe:

- 1  $\phi$  is a linear map;
- 2  $\phi(a) = L$  and  $\phi(b) = U$ .

In  $\phi$  replace  $[a, b]$  by  $[m, M]$  to map  $[m, M]$  into  $[L, U]$ .

# outline of the class

```
class SampleFun():
    """
    GUI to sample  $f(x)$  over  $[a, b]$ 
    """
    def __init__(self, wdw):
        "defines layout and data of GUI"

    def sample(self):
        "samples the function"

    def scale_point(self, xpt, lft, rgh, low, upp):
        "maps xpt from [lft, rgh] into [low, upp]"

    def scale(self):
        "scales the points"

    def show(self):
        "samples, scales, and shows"
```

# the constructor

```
def __init__(self, wdw, size):
    """
    defines layout and data of GUI
    """
    wdw.title("Plot Expressions")
    self.lbl1 = Label(wdw, text="f(x) =")
    self.lbl1.grid(row=0)
    self.fun = Entry(wdw) # for f(x)
    self.fun.grid(row=0, column=1, \
        columnspan=5, sticky=W+E+N+S)
    self.lbl2 = Label(wdw, text="a =")
    self.lbl2.grid(row=1, column=0)
    self.enta = Entry(wdw) # for a
    self.enta.grid(row=1, column=1)
    self.lbl3 = Label(wdw, text="b =")
    self.lbl3.grid(row=1, column=2)
    self.entb = Entry(wdw) # for b
    self.entb.grid(row=1, column=3)
```

## code for `__init__` continued

```
self.lbl4 = Label(wdw, text="N =")
self.lbl4.grid(row=1, column=4)
self.entn = Entry(wdw) # for N
self.entn.grid(row=1, column=5)
self.dim = size # dimension of canvas
self.cnv = Canvas(wdw, width=self.dim, \
                  height=self.dim, bg='white')
self.cnv.grid(row=3, columnspan=6)
self.btt = Button(wdw, text="sample and plot", \
                  command=self.show)
self.btt.grid(row=2, column=0, \
              columnspan=6, sticky=W+E+N+S)
self.org = [] # where sampled
self.val = [] # sampled values
self.pts = [] # scaled points
```

# sampling points

```
def sample(self):
    """
    samples the function
    """
    agt = float(self.enta.get())
    bgt = float(self.entb.get())
    nbr = int(self.entn.get())
    step = (bgt-agt)/(nbr+1)
    x = agt
    self.org = []
    self.val = []
    for k in range(0, nbr):
        eva = eval(self.fun.get())
        self.org.append(x)
        self.val.append(eva)
        x = x + step
```

## scaling one point

To map the  $(x, y)$ -coordinates into canvas coordinates:

$$\phi : [a, b] \rightarrow [L, U] : x \mapsto \frac{U - L}{b - a}(x - a) + L$$

This map  $\phi$  is coded up in the following method:

```
def scale_point(self, xpt, lft, rgh, low, upp):  
    """  
    maps xpt from [lft, rgh] into [low, upp]  
    """  
    return (upp-low)/(rgh-lft)*(xpt-lft) + low
```

## scaling a list of points

```
def scale(self):
    """
    scales the points
    """
    agt = float(self.enta.get())
    bgt = float(self.entb.get())
    miv = min(self.val)
    mav = max(self.val)
    upp = 0.95*self.dim
    low = 0.05*self.dim
    self.pts = []
    for k in range(0, len(self.org)):
        xsc = self.scale_point(self.org[k], agt, bgt, low, upp)
        ysc = self.scale_point(self.val[k], miv, mav, low, upp)
        ysc = upp - ysc + low # origin is topleft
        self.pts.append((xsc, ysc))
```

## showing the samples

```
def show(self):  
    """  
    samples, scales, and shows  
    """  
    self.sample()  
    self.scale()  
    self.cnv.delete(ALL) # refresh canvas  
    for k in range(0, len(self.pts)):  
        self.cnv.create_oval( \  
            self.pts[k][0]-6, self.pts[k][1]-6, \  
            self.pts[k][0]+6, self.pts[k][1]+6, \  
            width=1, outline='black', fill='SkyBlue2')
```

# expression evaluation graphing on Canvas

- 1 Evaluation of Expressions
  - a graphical calculator
  - object oriented version of the GUI
- 2 Graphing Expressions
  - the Canvas widget
  - sampling and plotting expressions
- 3 Binding Keys to Canvas
  - binding arrow keys
- 4 Summary + Assignments

# Binding Arrow Keys

Similar to binding mouse buttons to functions, we can bind the pressing of keys to Canvas.

GUI to explore this feature:

- 1 set up canvas, binds `Any-KeyPress`,
- 2 red dot initially at center of canvas,
- 3 move function reacts to arrow keys,
- 4 main program to launch GUI.

To activate listening to keyboard:

- click on the window, and
- press tab,

then the GUI will listen to the arrow keys.

## class KeyShow

```
from tkinter import Tk, Canvas
```

```
class KeyShow(object):
```

```
    """
```

```
    The GUI consists of a canvas and the arrow keys  
    which allows the user to move a red dot.
```

```
    """
```

```
    def __init__(self, wdw, r, c):
```

```
        """
```

```
        Sets up the layout of the GUI.
```

```
        """
```

```
        wdw.title("binding arrow keys")
```

```
        self.cnv = Canvas(wdw, width=c, height=r)
```

```
        self.cnv.grid(row=0, column=0)
```

```
        self.cnv.bind("<Any-KeyPress>", self.move)
```

```
        self.rows = r
```

```
        self.cols = c
```

```
        self.pos = [self.cols/2, self.rows/2]
```

```
        self.draw_dot()
```

## drawing a dot

```
def draw_dot(self):  
    """  
    Draws a dot at the location defined by self.pos.  
    """  
    self.cnv.delete('dot')  
    [xpd, ypd] = self.pos  
    self.cnv.create_oval(xpd-6, ypd-6, xpd+6, ypd+6, \  
        fill='red', tags='dot')
```

In moving the dot, we adapt a doughnut topology:

- A dot that moves past the left (or right) boundary of the canvas appears again from the right (or left) boundary.
- A dot that exits the upper (or lower) boundary of the canvas enters again from the lower (or upper) boundary.

## the callback function to move a dot

```
def move(self, event):
    """
    Moves the dot in the direction of the arrow key.
    The canvas is organized as a torus.
    """
    if event.keysym == 'Left':
        self.pos[0] = self.pos[0] - 6
        if self.pos[0] < 0:
            self.pos[0] = self.cols - self.pos[0]
    if event.keysym == 'Right':
        self.pos[0] = self.pos[0] + 6
        if self.pos[0] > self.cols:
            self.pos[0] = self.pos[0] - self.cols
    if event.keysym == 'Up':
        self.pos[1] = self.pos[1] - 6
        if self.pos[1] < 0:
            self.pos[1] = self.rows - self.pos[1]
    if event.keysym == 'Down':
        self.pos[1] = self.pos[1] + 6
        if self.pos[1] > self.rows:
            self.pos[1] = self.pos[1] - self.rows
    self.draw_dot()
```

# the main function

```
def main():  
    """  
    Puts up the canvas and then listens for keys.  
    """  
    top = Tk()  
    print 'press tab after clicking on window,'  
    print 'then use arrow keys to move red dot'  
    KeyShow(top, 200, 300)  
    top.mainloop()  
  
if __name__ == "__main__":  
    main()
```

# Summary + Assignments

Chapter 11 of *Python and Tkinter Programming*

by John Grayson, is free to download from the web.

Assignments:

- 1 Add a new Entry widget to the graphical calculator to contain the precision (number of digits after the point) to use for the result of the evaluation.
- 2 Modify the `Show()` function so that line segments are drawn to connect consecutive samples.
- 3 Using canvas, make a GUI for turtle graphics. Provide buttons `PenUp`, `PenDown`, `Move`, and `Turn`.
- 4 Add a `CheckButton` to the graphing calculator. When checked, bars instead of ovals are used to plot.
- 5 Make a GUI for a pie chart. A list of numbers is given via an Entry widget. To each number corresponds a slice in the pie. The angle of each slice is related to the number divided by the sum of the list.