Outline

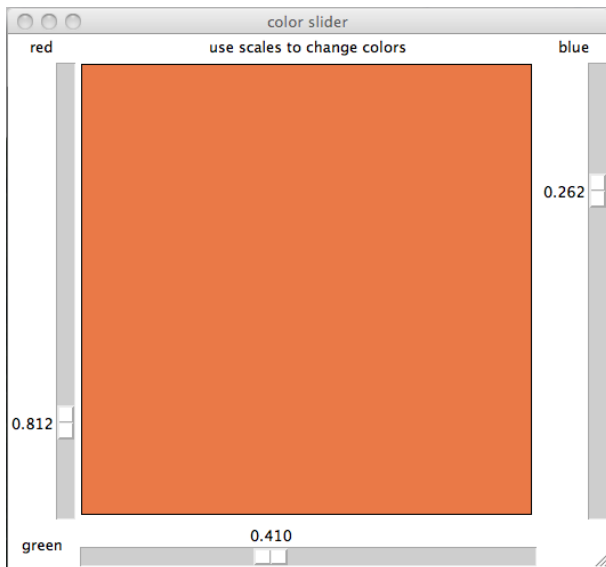
1. The Widget Scale
   - making colors with scale widgets
   - entering parameter values
   - using the variables in Scale
2. Building an Animation
   - sliding canvas coordinates
   - animating a random walk
3. Summary + Assignments

MCS 260 Lecture 32
Introduction to Computer Science
Jan Verschelde, 1 April 2016

entering data with scale
developing animations

1. The Widget Scale
   - making colors with scale widgets
   - entering parameter values
   - using the variables in Scale
2. Building an Animation
   - sliding canvas coordinates
   - animating a random walk
3. Summary + Assignments

an application: 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 `show_colors` is executed.
- When the GUI starts up, the scale has value `0.5`.

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

showing colors

```
def show_colors(self, val):
    """
    Displays a rectangle on canvas, filled with rgb colors.
    """
    xmd = self.dim/2+1
    ymd = self.dim/2+1
    mid = self.dim/2-3
    red = self.scared.get()
    green = self.scagrn.get()
    blue = self.scablu.get()
    print 'r = %f, g = %f, r = %f' % (red, green, blue)
    hxr = '%.2x' % int(255*red)
    hxg = '%.2x' % int(255*green)
    hxb = '%.2x' % int(255*blue)
    color = '#' + hxr + hxg + hxb
    self.cnv.delete('box')
    self.cnv.create_rectangle(xmd-mid, ymd-mid, xmd+mid, \
         ymd+mid, width=1, outline='black', fill=color, \
         tags='box')
```

an explanation for `show_colors`

Key aspects of the method `show_colors`:

- The argument `val` of `show_colors` is the value of the scale, but we need the values of all three intensities.
- With `self.scared.get()` we get the red intensity. Green and blue intensities are set by the scales with names `scagrn` and `scablu` respectively.
- The `print` writes to the terminal.
- `hxr = '%.2x' % int(255*red)` 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.

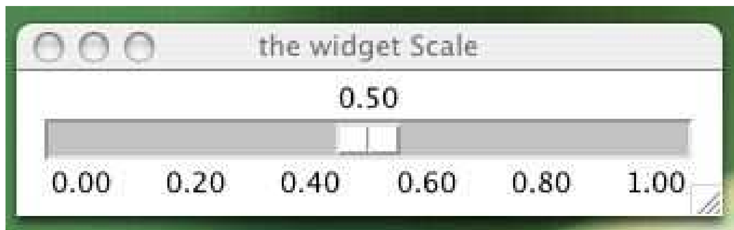
entering data with scale
developing animations

1 The Widget Scale
- making colors with scale widgets
- entering parameter values
- using the variables in Scale

2 Building an Animation
- sliding canvas coordinates
- animating a random walk

3 Summary + Assignments

the widget Scale



To enter parameter variables:

1. programmer can specify a meaningful range;
2. the scale is initialized with a default value.

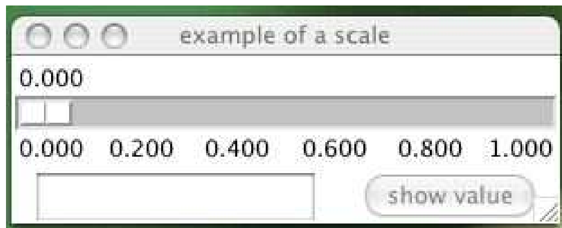
code for a scale in the file `showscale.py`

```
from tkinter import Tk, Scale, DoubleVar
TOP = Tk()
TOP.title('the widget Scale')
LOW = 0.0           # lowest value
HIGH = 1.0          # highest value
VAR = DoubleVar()   # variable to scale
SCL = Scale(TOP, orient='horizontal', \
    from_=LOW, to=HIGH,\
    tickinterval=(HIGH-LOW)/5.0, \
    resolution=(HIGH-LOW)/100.0, \
    length=300, variable=VAR)
SCL.set(0.5)        # initialize VAR to 0.5
SCL.pack()
TOP.mainloop()
```

entering data with scale developing animations

1. The Widget Scale
   - making colors with scale widgets
   - entering parameter values
   - using the variables in Scale
2. Building an Animation
   - sliding canvas coordinates
   - animating a random walk
3. Summary + Assignments

using the variables in Scale



Observe:

1. the Entry widget will display the Scale variable;
2. only when the Button `show value` is pressed.

adding Entry and Button to `exscale0.py`

```
from tkinter import Tk, DoubleVar, Scale, Button
from tkinter import Entry, END, INSERT

TOP = Tk()
TOP.title('example of a scale')
LOW = 0.0
HIGH = 1.0
VAR = DoubleVar()
SCL = Scale(TOP, orient='horizontal', \
    from_=LOW, to=HIGH, \
    tickinterval=(HIGH-LOW)/5.0, \
    resolution=(HIGH-LOW)/1000.0, \
    length=300, variable=VAR)
SCL.grid(row=0, columnspan=2)
```

Notice the change in resolution.

adding Entry and Button to `exscale0.py` continued

```
ENT = Entry(TOP)
ENT.grid(row=1, column=0)

def show_value():
    """
    Displays the value of scale variable Var
    in the entry widget ENT.
    """
    ENT.delete(0, END)
    ENT.insert(INSERT, VAR.get())

BTT = Button(TOP, text="show value", \
    command=show_value)
BTT.grid(row=1, column=1)
TOP.mainloop()
```

The `show_value` must be defined *before* the `Button`.
Intermingling layout and actions leads to cluttered code.

object oriented version in `exscale1.py`

- data attributes define the layout of the GUI,
- functional attribute define the actions.

```
class ShowScale(object):
    """
    GUI to demonstrate use of a scale.
    """
    def __init__(self, wdw):
        """
        determines the layout of the GUI
        """
    def show_value(self):
        """
        Shows the value of the scale variable
        in the entry widget.
        """
```

The `show_value` is the callback function for `Button`.

the constructor: `__init__` in `exscale1.py`

```
def __init__(self, wdw):
    """
    determines the layout of the GUI
    """
    wdw.title('example of a scale')
    self.low = 0.0
    self.high = 1.0
    self.var = DoubleVar()
    self.scl = Scale(wdw, \
        orient='horizontal', \
        from_=self.low, to=self.high, \
        tickinterval=(self.high-self.low)/5.0, \
        resolution=(self.high-self.low)/1000.0, \
        length=300, variable=self.var)
    self.scl.grid(row=0, columnspan=2)
    self.ent = Entry(wdw)
    self.ent.grid(row=1, column=0)
    self.btt = Button(wdw, text="show value", \
        command=self.show_value)
    self.btt.grid(row=1, column=1)
```

show_value and main in the file exscale1.py

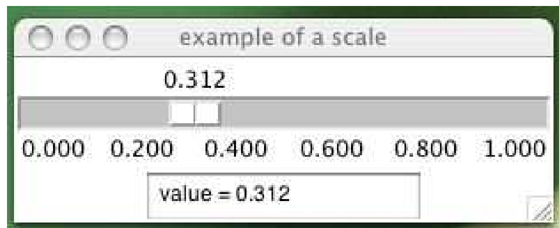
show_value is called when the button is pressed:

```
    def show_value(self):
        """
        Shows the value of the scale variable
        in the entry widget.
        """
        self.ent.delete(0, END)
        self.ent.insert(INSERT, self.var.get())

def main():
    """
    Instantiates the GUI and launches
    the main event loop.
    """
    top = Tk()
    ShowScale(top)
    top.mainloop()

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

The last construction allows to run the GUI as a script.

adding `command` to Scale

avoiding `Button`



The `Scale` widget has the `command` option.

- Use as `command = ShowValue`.
- `ShowValue` is called whenever the user activates the `Scale`.
- `ShowValue` displays the value of the scale variable in the `Entry` widget.

updated code in the file `exscale2.py`

```
def __init__(self, wdw):
    """
    determines the layout of the GUI
    """
    wdw.title('example of a scale')
    self.low = 0.0
    self.high = 1.0
    self.var = DoubleVar()
    self.scl = Scale(wdw, \
        orient='horizontal', \
        from_=self.low, to=self.high, \
        tickinterval=(self.high-self.low)/5.0, \
        resolution=(self.high-self.low)/1000.0, \
        length=300, variable=self.var, \
        command=self.show_value)
    self.scl.grid(row=0, column=0)
    self.ent = Entry(wdw)
    self.ent.grid(row=1, column=0)
```

update code for show_value in exscale2.py

show_value will show the value of the scale variable, preceded by the string 'value = '.

show_value must now have one extra input parameter: the value of the scale variable.

```
def show_value(self, val):
    """
    Displays the value of the scale variable
    in the entry widget.
    """
    self.ent.delete(0, END)
    strval = 'value = ' + str(val)
    self.ent.insert(INSERT, strval)
```

Observe the string conversions in show_value.

entering data with scale
developing animations

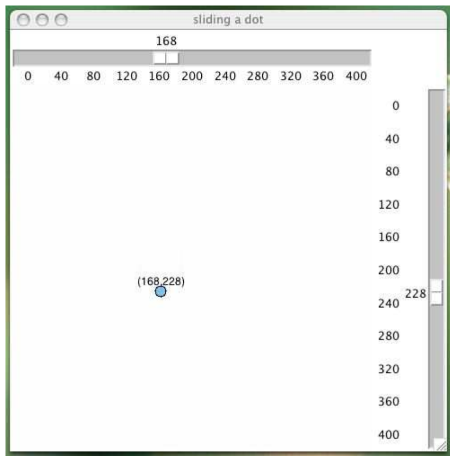
1. The Widget Scale
   - making colors with scale widgets
   - entering parameter values
   - using the variables in Scale

2. Building an Animation
   - sliding canvas coordinates
   - animating a random walk

3. Summary + Assignments

Sliding a Dot

to feel the coordinates



design of the GUI `slidedot.py`

definition of layout and the actions

The layout consists of

- a horizontal scale for the x coordinate;
- a vertical scale for the y coordinate;
- a canvas to draw the dot and its coordinates.

Other data attributes are the values for `x` and `y`.

The action performed by the GUI is to draw the dot on the canvas using the coordinates.

The action is triggered when the user touches the scales, adjusting the coordinates for the dot.

definition of the GUI applying OOP in `slidedot.py`

```
class SlideDot(object):
    """
    GUI to demonstrate canvas coordinates.
    """
    def __init__(self, wdw, size):
        """
        determines the layout of the GUI
        """
    def draw_dot(self, val):
        """
        draws the dot and its scale variable
        """

def main():
    """
    Instantiates the GUI and
    launches the main event loop.
    """
```

the constructor: definition of __init__

The first widget is an instance of `Canvas`:

```
def __init__(self, wdw, size):
    """
    determines the layout of the GUI
    """
    wdw.title('sliding a dot')
    self.dim = size
    self.cnv = Canvas(wdw, width=self.dim, \
        height=self.dim, bg='white')
    self.cnv.grid(row=1, column=0)
    self.xpos = IntVar()
    self.ypos = IntVar()
```

The data attributes `xpos` and `ypos` are instances of `IntVar`, set by the scales.

the scales: `__init__` continued

Ranges of the scale are parameters depending on the dimension of the canvas, stored in `self.dim`.

```
self.low = 0
self.high = self.dim
self.hsx = Scale(wdw, \
    orient='horizontal', \
    from_=self.low, to=self.high, \
    tickinterval=(self.high-self.low)/10, \
    resolution=(self.high-self.low)/100, \
    length=self.dim, variable=self.xpos, \
command=self.draw_dot)
self.hsx.grid(row=0, column=0)
self.hsx.set(self.dim/2)
```

The code for `self.vsy`, the scale to adjust the *y*-coordinate in `ypos` is similar.

drawing the dot: the function draw_dot

Both scales activate the same function draw_dot,
each with their own scale value.

```
def draw_dot(self, val):
    """
    draws the dot and its scale variable
    """
    valx = self.xpos.get()
    valy = self.ypos.get()
    self.cnv.delete("dot", "text")
    txt = '(' + str(int(valx)) + ',' + str(int(valy)) + ')'
    self.cnv.create_text(valx, valy-10, text=txt, \
        tags="text")
    self.cnv.create_oval(valx-6, valy-6, valx+6, valy+6, \
        width=1, outline='black', fill='SkyBlue2', \
        tags="dot")
```

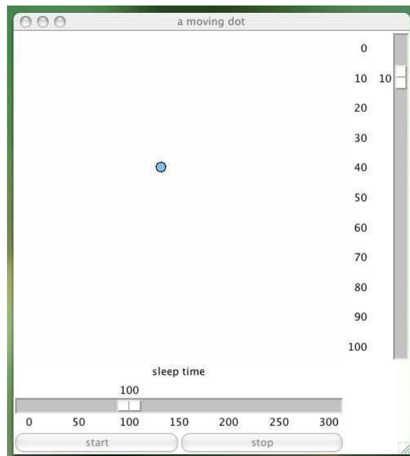
Observe the use of the tags "dot" and "text".

entering data with scale developing animations

1. The Widget Scale
   - making colors with scale widgets
   - entering parameter values
   - using the variables in Scale
2. Building an Animation
   - sliding canvas coordinates
   - animating a random walk
3. Summary + Assignments

A Moving Dot

animating a random walk



design of the GUI `movedot.py`

definition of layout and the actions

The layout of the animated random walk consists of

- the canvas on which the dot will move;
- a horizontal scale to regulate the speed;
- a vertical scale for the step size;
- start and stop buttons.

The actions of the GUI can be summarized as

- the animation starts when `start` is pressed;
- and stops when `stop` is pressed;
- any changes in the values set by the scale take immediate effect when the animation is running.

definition of the GUI: applying OOP in `movedot.py`

```
class MovingDot(object):
    """
    GUI to illustrate an animation.
    """
    def __init__(self, wdw, size):
        """
        determines the layout of the GUI
        """
    def animate(self):
        """
        performs the animation
        """
    def start(self):
        """
        starts the animation
        """
    def stop(self):
        """
        stops the animation
        """
```

the constructor: the code for `__init__`

```
def __init__(self, wdw, size):
    """
    determines the layout of the GUI
    """
    wdw.title('a moving dot')
    self.dim = size
    self.cnv = Canvas(wdw, width=self.dim, \
        height=self.dim, bg ='white')
    self.cnv.grid(row=0, column=0, columnspan=2)
    self.slp = Label(wdw, text="sleep time")
    self.slp.grid(row=1, column=0, columnspan=2)
    self.xpos = self.dim/2
    self.ypos = self.dim/2
    self.togo = False
    self.sleep = IntVar()
    self.step = IntVar()
```

Data attributes are coordinates `xpos`, `ypos` of the dot, the state of animation `togo`, the parameters `sleep` and `step`.

the scales: `__init__` continued

The horizontal scale determines the speed,
i.e.: how much time between each drawing of a new dot.

```
        self.hsc = Scale(wdw, \
            orient='horizontal', from_=0, to=300, \
            tickinterval=50, resolution=1, \
            length=300, variable=self.sleep)
        self.hsc.grid(row=2, column=0, columnspan=2, \
            sticky=W+E+N+S)
        self.hsc.set(100)
        self.vsc = Scale(wdw, \
            orient='vertical', from_=0,to=100, \
            tickinterval=10, resolution=1, \
            length=400, variable=self.step)
        self.vsc.grid(row=0, column=2)
        self.vsc.set(10)
```

Default values for horizontal and vertical scale variables `sleep` and `step` are respectively 100 and 10.

the Buttons – the end of code for _ _init_ _

The buttons with text `start` and `stop` trigger
the beginning and the ending of the animation.

```
self.bt0 = Button(wdw, text="start", \
    command = self.start)
self.bt0.grid(row=3, column=0, sticky=W+E)
self.bt1 = Button(wdw, text="stop", \
    command = self.stop)
self.bt1.grid(row=3, column=1, sticky=W+E)
```

`start` and `stop` are the respective callback functions
for the buttons `b0` and `b1`.

code for the animation

```
def animate(self):
    """
    performs the animation
    """
    vxp = self.xpos
    vyp = self.ypos
    while self.togo:
        self.cnv.delete("dot")
        self.cnv.create_oval(vxp-6, vyp-6, vxp+6, vyp+6, width=1, \
            outline='black', fill='SkyBlue2', tags="dot")
        stp = self.step.get()
        vxp = vxp + randint(-stp, stp)
        vyp = vyp + randint(-stp, stp)
        if vxp >= self.dim:
            vxp = vxp - self.dim
        if vyp >= self.dim:
            vyp = vyp - self.dim
        if vxp < 0:
            vxp = vxp + self.dim
        if vyp < 0:
            vyp = vyp + self.dim
        self.cnv.after(self.sleep.get())
        self.cnv.update()
```

code for start, stop, and main

```
    def start(self):
        """
        starts the animation
        """
        self.togo = True
        self.animate()

    def stop(self):
        """
        stops the animation
        """
        self.togo = False

def main():
    """
    Instantiates the GUI and
    lauches the event loop.
    """
    top = Tk()
    MovingDot(top, 500)
    top.mainloop()
```

Summary + Assignments

Assignments:

1. Define a scale to determine the formatting of floats.
   If `k` is the scale variable, it is used as `'%.k'`.
   Use two Entry fields in the GUI: one for the number
   entered by the user and the other for the formatted number.
2. Use the scale in the GUI to evaluate expressions,
   adjusting `guievaloo.py` of Lecture 31.
3. Make an animation of a dot moving along a circle,
   centered at the center of the canvas.
   Use scales to adjust the speed of the animation
   and the radius of the circle.
4. Add a scale to `sliding_puzzle.py` of Lecture 30
   to adjust the speed of the scrambling & unscrambling.