Outline

1. The Widget Scale
   entering parameter values
   using the variables in Scale

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

3. Summary + Assignments
entering data with scale developing animations

1 The Widget Scale
   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.
To enter parameter variables:

1. programmer can specify a meaningful range;
2. the scale is initialized with a default value.
To enter parameter variables:

1. programmer can specify a meaningful range;
2. the scale is initialized with a default value.
The Widget
Scale
entering parameter values
using the variables in Scale
Building an Animation
sliding canvas coordinates
animating a random walk
Summary + Assignments

# L-32 MCS 260 Fri 7 Nov 2008 : showscale.py
# Illustration of the widget Scale.

from Tkinter import *

# L-32 MCS 260 Fri 7 Nov 2008 : showscale.py
# Illustration of the widget Scale.

from Tkinter import *
top = Tk()
top.title('the widget Scale')
low = 0.0 # lowest value
high = 1.0 # highest value
f = DoubleVar() # variable to scale

s = Scale(top,orient='horizontal',
          from_=low,to=high,
          tickinterval=(high-low)/5.0,
          resolution=(high-low)/100.0,
          length=300,variable=f)
s.set(0.5) # initialize f to 0.5
s.pack()
top.mainloop()
The Widget
Scale
entering parameter values
using the variables in Scale

Building an Animation
sliding canvas coordinates
animating a random walk

Summary + Assignments

---

# L-32 MCS 260 Fri 7 Nov 2008 : showscale.py

# Illustration of the widget Scale.

from Tkinter import *
top = Tk()
top.title('the widget Scale')
low = 0.0       # lowest value
high = 1.0      # highest value
f = DoubleVar() # variable to scale

s = Scale(top,orient='horizontal',
from_=low,to=high,
 tickinterval=(high-low)/5.0,
 resolution=(high-low)/100.0,
 length=300,variable=f)
s.set(0.5)    # initialize f to 0.5
s.pack()
top.mainloop()
Code for a Scale
in the file `showscale.py`

```python
# L-32 MCS 260 Fri 7 Nov 2008 : showscale.py
# Illustration of the widget Scale.
from Tkinter import *
top = Tk()
top.title('the widget Scale')
low = 0.0     # lowest value
high = 1.0    # highest value
f = DoubleVar() # variable to scale

s = Scale(top,orient='horizontal',
          from_=low,to=high,
          tickinterval=(high-low)/5.0,
          resolution=(high-low)/100.0,
          length=300,variable=f)
s.set(0.5)    # initialize f to 0.5
s.pack()
top.mainloop()
```
# L-32 MCS 260 Fri 7 Nov 2008 : showscale.py

# Illustration of the widget Scale.

from Tkinter import *
top = Tk()
top.title('the widget Scale')
low = 0.0 # lowest value
high = 1.0 # highest value
f = DoubleVar() # variable to scale

s = Scale(top,orient='horizontal',
    from_=low,to=high,
    tickinterval=(high-low)/5.0,
    resolution=(high-low)/100.0,
    length=300,variable=f)
s.set(0.5) # initialize f to 0.5
s.pack()
top.mainloop()
entering data with scale
developing animations

1 The Widget Scale
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.
Observe:

1. the Entry widget will display the Scale variable;
2. only when the Button show value is pressed.
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
in the file exscale0.py

The start is very much the same:

```python
from Tkinter import *
top = Tk()
top.title(‘example of a scale’) 
low = 0.0
high = 1.0
f = DoubleVar()
s = Scale(top,orient=’horizontal’,
     from_=low,to=high,
     tickinterval=(high-low)/5.0,
     resolution=(high-low)/1000.0,
     length=300,variable=f)
s.grid(row=0,columnspan=2)
```

Notice the change in resolution.
Adding Entry and Button
in the file exscales0.py

The start is very much the same:

```python
from Tkinter import *
top = Tk()
top.title('example of a scale')
low = 0.0
high = 1.0
f = DoubleVar()
s = Scale(top, orient='horizontal',
          from_=low, to=high, 
          tickinterval=(high-low)/5.0, 
          resolution=(high-low)/1000.0, 
          length=300, variable=f)
s.grid(row=0, columnspan=2)
```

Notice the change in resolution.
Adding Entry and Button
in the file exscale0.py

The start is very much the same:

```python
from Tkinter import *
top = Tk()
top.title('example of a scale')
low = 0.0
high = 1.0
f = DoubleVar()
s = Scale(top,orient='horizontal',
    from_=low,to=high,
    tickinterval=(high-low)/5.0,
    resolution=(high-low)/1000.0,
    length=300,variable=f)
s.grid(row=0,columnspan=2)
```

Notice the change in resolution.
Adding Entry and Button

code in file exscale0.py continued

The added code with Entry and Button

```python
e = Entry(top)
e.grid(row=1,column=0)
def ShowValue():
    "shows value of scale variable f"
    e.delete(0,END)
    e.insert(INSERT,f.get())
b = Button(top,text="show value",
    command=ShowValue)
b.grid(row=1,column=1)
top.mainloop()
```

The function ShowValue must be defined before the definition of the Button.

Already for small examples, intermingling layout and actions will lead to cluttered code.
The added code with `Entry` and `Button`

```python
e = Entry(top)
e.grid(row=1,column=0)
def ShowValue():
    "shows value of scale variable f"
    e.delete(0,END)
    e.insert(INSERT,f.get())
b = Button(top,text="show value",
           command=ShowValue)
b.grid(row=1,column=1)
top.mainloop()
```

The function `ShowValue` must be defined *before* the definition of the `Button`.

Already for small examples, intermingling layout and actions will lead to cluttered code.
Adding Entry and Button

code in file exscale0.py continued

The added code with Entry and Button

```python
# The added code with Entry and Button

e = Entry(top)
e.grid(row=1,column=0)
def ShowValue():
    "shows value of scale variable f"
    e.delete(0,END)
    e.insert(INSERT,f.get())
b = Button(top,text="show value",
    command=ShowValue)
b.grid(row=1,column=1)
top.mainloop()
```

The function ShowValue must be defined before the definition of the Button.

Already for small examples, intermingling layout and actions will lead to cluttered code.
The added code with Entry and Button

```python
import tkinter

f = tkinter.Scale(top, from_=0, to=100)

e = Entry(top)
e.grid(row=1,column=0)
def ShowValue():
    "shows value of scale variable f"
    e.delete(0,END)
    e.insert(INSERT,f.get())
b = Button(top,text="show value",
    command=ShowValue)
b.grid(row=1,column=1)
top.mainloop()
```

The function ShowValue must be defined before the definition of the Button.

Already for small examples, intermingling layout and actions will lead to cluttered code.
With a class we separate the layout from actions, defining the layout in the object data attributes, and the actions in the functional attributes of the class.

```python
from Tkinter import *

class ShowScale():
    
    """
    GUI to demonstrate use of a scale.
    """

    def __init__(self, wdw):
        """determines the layout of the GUI"

    def ShowValue(self):
        """shows the value of the scale variable"
```

The `ShowValue` is the callback function for `Button`. 
Object Oriented version

in the file exscale1.py

With a class we separate the layout from actions, defining the layout in the object data attributes, and the actions in the functional attributes of the class.

```python
from Tkinter import *

class ShowScale():
    """
    GUI to demonstrate use of a scale.
    """

def __init__(self, wdw):
    "determines the layout of the GUI"

def ShowValue(self):
    "shows the value of the scale variable"

The ShowValue is the callback function for Button.
```
Object Oriented version
in the file exscale1.py

With a class we separate the layout from actions, defining the layout in the object data attributes, and the actions in the functional attributes of the class.

from Tkinter import *

class ShowScale():
    """
    GUI to demonstrate use of a scale.
    """
    def __init__(self, wdw):
        "determines the layout of the GUI"

    def ShowValue(self):
        "shows the value of the scale variable"

The ShowValue is the callback function for Button.
Object Oriented version
in the file exscale1.py

With a class we separate the layout from actions, defining the layout in the object data attributes, and the actions in the functional attributes of the class.

from Tkinter import *

class ShowScale():
    """
    GUI to demonstrate use of a scale.
    """
    def __init__(self, wdw):
        "determines the layout of the GUI"

    def ShowValue(self):
        "shows the value of the scale variable"

The ShowValue is the callback function for Button.
Object Oriented version
in the file exscale1.py

With a class we separate the layout from actions, defining the layout in the object data attributes, and the actions in the functional attributes of the class.

```python
from Tkinter import *

class ShowScale():
    """
    GUI to demonstrate use of a scale.
    """
    def __init__(self, wdw):
        """determines the layout of the GUI""

    def ShowValue(self):
        """shows the value of the scale variable""

The ShowValue is the callback function for Button.
```
The Constructor

the code for __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.f = DoubleVar()
    self.s = Scale(wdw,
                   ... same code as before ...
    self.s.grid(row=0, columnspan=2)
    self.e = Entry(wdw)
    self.e.grid(row=1, column=0)
    self.b = Button(wdw, text="show value",
                   command=self.ShowValue)
    self.b.grid(row=1, column=1)
The Constructor

the code for __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.f = DoubleVar()
    self.s = Scale(wdw,
                   ... same code as before ...
    self.s.grid(row=0, columnspan=2)
    self.e = Entry(wdw)
    self.e.grid(row=1, column=0)
    self.b = Button(wdw, text="show value",
                    command=self.ShowValue)
    self.b.grid(row=1, column=1)
The Constructor

the code for \_\_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.f = DoubleVar()
    self.s = Scale(wdw,
        "... same code as before ...
    self.s.grid(row=0, columnspan=2)
    self.e = Entry(wdw)
    self.e.grid(row=1, column=0)
    self.b = Button(wdw, text="show value",
        command=self.ShowValue)
    self.b.grid(row=1, column=1)
ShowValue and main in the file exscale1.py

ShowValue is called when the button is pressed:

```python
def ShowValue(self):
    "shows the value of the scale variable"
    self.e.delete(0, END)
    self.e.insert(INSERT, self.f.get())

def main():
    top = Tk()
    show = ShowScale(top)
    top.mainloop()

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

The last construction allows to run the GUI as a script.
ShowValue and main in the file exscale1.py

ShowValue is called when the button is pressed:

```python
def ShowValue(self):
    "shows the value of the scale variable"
    self.e.delete(0, END)
    self.e.insert(INSERT, self.f.get())
```

```python
def main():
    top = Tk()
    show = 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.
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.
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.
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.
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.f = DoubleVar()
    self.s = 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.f, 
                   command=self.ShowValue)
    self.s.grid(row=0, column=0)
    self.e = Entry(wdw)
    self.e.grid(row=1, column=0)
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.f = DoubleVar()
    self.s = 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.f,
                   command=self.ShowValue)
    self.s.grid(row=0, column=0)
    self.e = Entry(wdw)
    self.e.grid(row=1, column=0)
update code for ShowValue in the file exscale2.py

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

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

```python
def ShowValue(self,v):
    "shows the value of the scale variable"
    self.e.delete(0,END)
    sv = 'value = ' + str(v)
    self.e.insert(INSERT,sv)
```

Observe the string conversions in ShowValue.
update code for ShowValue in the file exscale2.py

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

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

```python
def ShowValue(self,v):
    "shows the value of the scale variable"
    self.e.delete(0,END)
    sv = 'value = ' + str(v)
    self.e.insert(INSERT,sv)
```

Observe the string conversions in ShowValue.
update code for ShowValue
in the file exscale2.py

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

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

```python
def ShowValue(self,v):
    "shows the value of the scale variable"
    self.e.delete(0,END)
    sv = 'value = ' + str(v)
    self.e.insert(INSERT,sv)
```

Observe the string conversions in ShowValue.
entering data with scale
developing animations

1. The Widget Scale
   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
entering parameter values
using the variables in Scale

Building an Animation
sliding canvas coordinates
animating a random walk

Summary + Assignments

Sliding a Dot
to feel the coordinates

Run the code `slidedot.py`!
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.
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.
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.
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.
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.
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.
We separate the layout and the action:

```python
from Tkinter import *

class SlideDot():
    ""
    GUI to demonstrate canvas coordinates.
    ""
    def __init__(self, wdw):
        "determines the layout of the GUI"
    def DrawCircle(self, v):
        "draws the dot and its coordinates"

def main():
    top = Tk()
    show = SlideDot(top)
    top.mainloop()

if __name__ == "__main__": main()
```
We separate the layout and the action:

```
from Tkinter import *

class SlideDot():
    """
    GUI to demonstrate canvas coordinates.
    """
    def __init__(self, wdw):
        "determines the layout of the GUI"

    def DrawCircle(self, v):
        "draws the dot and its coordinates"

def main():
    top = Tk()
    show = SlideDot(top)
    top.mainloop()

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

definition of __init__

The first widget is an instance of Canvas:

```python
def __init__(self, wdw):
    "determines the layout of the GUI"
    wdw.title('sliding a dot')
    self.d = 400
    self.c = Canvas(wdw, width=self.d,\
                    height=self.d, bg = 'white')
    self.c.grid(row=1, column=0)
    self.x = IntVar()
    self.y = IntVar()
```

The data attributes x and y are instances of IntVar, set by the scales.
The Constructor

definition of __init__

The first widget is an instance of Canvas:

```python
def __init__(self, wdw):
    "determines the layout of the GUI"
    wdw.title('sliding a dot')
    self.d = 400
    self.c = Canvas(wdw, width=self.d, \n        height=self.d, bg = 'white')
    self.c.grid(row=1, column=0)
    self.x = IntVar()
    self.y = IntVar()
```

The data attributes x and y are instances of IntVar, set by the scales.
The Constructor

definition of __init__

The first widget is an instance of Canvas:

def __init__(self, wdw):
    "determines the layout of the GUI"
    wdw.title('sliding a dot')
    self.d = 400
    self.c = Canvas(wdw, width=self.d,
                    height=self.d, bg = 'white')
    self.c.grid(row=1, column=0)
    self.x = IntVar()
    self.y = IntVar()

The data attributes \( x \) and \( y \) are instances of \( \text{IntVar} \), set by the scales.
The first widget is an instance of `Canvas`:

```python
def __init__(self,wdw):
    "determines the layout of the GUI"
    wdw.title('sliding a dot')
    self.d = 400
    self.c = Canvas(wdw,width=self.d,\
                    height=self.d,bg = 'white')
    self.c.grid(row=1,column=0)
    self.x = IntVar()
    self.y = IntVar()
```

The data attributes \(x\) and \(y\) are instances of `IntVar`, set by the scales.
The Scales

definition of \_\_init\_\_ continued

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

```
scale.L = 0; scale.H = scale.d
self.sx = Scale(wdw,\
    orient='horizontal',\
    from_=self.L,to=self.H,\
    tickinterval=(self.H-self.L)/10,\
    resolution=(self.H-self.L)/100,\
    length=scale.d,variable=scale.x,\
    command=scale.DrawCircle)
self.sx.grid(row=0,column=0)
scale.sx.set(scale.d/2)
```

The code for self.sy, for the vertical scale is the same, except for orient='vertical', and its position self.sy.grid(row=1,column=1).
The Scales
definition of __init__ continued

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

```python
self.L = 0; self.H = self.d
self.sx = Scale(wdw, 
    orient='horizontal',
    from_=self.L, to=self.H, 
    tickinterval=(self.H-self.L)/10, 
    resolution=(self.H-self.L)/100, 
    length=self.d, variable=self.x, 
    command=self.DrawCircle)
self.sx.grid(row=0, column=0)
self.sx.set(self.d/2)
```

The code for `self.sy`, for the vertical scale is the same, except for `orient=’vertical’`, and its position `self.sy.grid(row=1, column=1)`.
The Scales

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

```python
self.L = 0; self.H = self.d
self.sx = Scale(wdw,
    orient='horizontal',
    from_=self.L, to=self.H,
    tickinterval=(self.H-self.L)/10,
    resolution=(self.H-self.L)/100,
    length=self.d, variable=self.x,
    command=self.DrawCircle)
sel sx.grid(row=0, column=0)
sel sx.set(self.d/2)
```

The code for `self.sy`, for the vertical scale is the same, except for `orient='vertical'`, and its position `self.sy.grid(row=1, column=1)`.
The Scales

definition of __init__ continued

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

self.L = 0; self.H = self.d
self.sx = Scale(wdw,\
    orient='horizontal',\
    from_=self.L,to=self.H,\
    tickinterval=(self.H-self.L)/10,\
    resolution=(self.H-self.L)/100,\
    length=self.d,variable=self.x,\
    command=self.DrawCircle)
self.sx.grid(row=0,column=0)
self.sx.set(self.d/2)

The code for self.sy, for the vertical scale is the same, except for orient='vertical',
and its position self.sy.grid(row=1,column=1).
The Scales

definition of \_\_init\_\_ continued

Ranges of the scale are parameters depending on the dimension of the canvas, stored in \texttt{self.d}.

\begin{verbatim}
self.L = 0; self.H = self.d
self.sx = Scale(wdw, \\
    orient='horizontal', \\
    from_=self.L, to=self.H, \\
    tickinterval=(self.H-self.L)/10, \\
    resolution=(self.H-self.L)/100, \\
    length=self.d, variable=self.x, \\
    command=self.DrawCircle)
self.sx.grid(row=0, column=0)
self.sx.set(self.d/2)
\end{verbatim}

The code for \texttt{self.sy}, for the vertical scale is the same, except for \texttt{orient='vertical'}, and its position \texttt{self.sy.grid(row=1, column=1)}.
The Widget Scale

entering parameter values
using the variables in Scale

Building an Animation

sliding canvas coordinates
animating a random walk

Summary + Assignments

Drawing the Dot

the function DrawCircle

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

def DrawCircle(self,v):
    "draws the dot and its scale variable"
    vx = self.x.get()
    vy = self.y.get()
    self.c.delete("dot","text")
    t = '(' + str(int(vx)) + ',' + str(int(vy)) + ')
    self.c.create_text(vx,vy-10,text=t,tags="text")
    self.c.create_oval(vx-6,vy-6,vx+6,vy+6,width=1,outline='black',fill='SkyBlue2',tags="dot")

Observe that we do not use v.
Both scales activate the same function `DrawCircle`, each with their own scale value.

```python
def DrawCircle(self, v):
    "draws the dot and its scale variable"
    vx = self.x.get()
    vy = self.y.get()
    self.c.delete("dot","text")
    t = '(' + str(int(vx)) + ',' + str(int(vy)) + ')
    self.c.create_text(vx, vy-10, text=t, tags="text")
    self.c.create_oval(vx-6, vy-6, vx+6, vy+6, width=1,
                        outline='black', fill='SkyBlue2', tags="dot")
```

Observe that we do not use v.
Drawing the Dot

the function DrawCircle

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

```python
def DrawCircle(self,v):
    "draws the dot and its scale variable"
    vx = self.x.get()
    vy = self.y.get()
    self.c.delete("dot","text")
    t = '(' + str(int(vx)) + ',' + str(int(vy)) + ')
    self.c.create_text(vx,vy-10,text=t,tags="text")
    self.c.create_oval(vx-6,vy-6,vx+6,vy+6,width=1,
        outline='black',fill='SkyBlue2',tags="dot")
```

Observe that we do not use v.
Both scales activate the same function `DrawCircle`, each with their own scale value.

```python
def DrawCircle(self,v):
    "draws the dot and its scale variable"
    vx = self.x.get()
    vy = self.y.get()
    self.c.delete("dot","text")
    t = '(' + str(int(vx)) + ',' + str(int(vy)) + ')
    self.c.create_text(vx,vy-10,text=t,tags="text")
    self.c.create_oval(vx-6,vy-6,vx+6,vy+6,width=1,outline='black',fill='SkyBlue2',tags="dot")
```

Observe that we do not use \( v \).
Both scales activate the same function DrawCircle, each with their own scale value.

```python
def DrawCircle(self,v):
    "draws the dot and its scale variable"
    vx = self.x.get()
    vy = self.y.get()
    self.c.delete("dot","text")
    t = '(' + str(int(vx)) + ',' + str(int(vy)) + ')
    self.c.create_text(vx,vy-10,text=t,tags="text")
    self.c.create_oval(vx-6,vy-6,vx+6,vy+6,width=1,
                      outline='black',fill='SkyBlue2',tags="dot")
```

Observe that we do not use v.
Both scales activate the same function `DrawCircle`, each with their own scale value.

```python
def DrawCircle(self, v):
    "draws the dot and its scale variable"
    vx = self.x.get()
    vy = self.y.get()
    self.c.delete("dot","text")
    t = '(' + str(int(vx)) + ',' + str(int(vy)) + ')
    self.c.create_text(vx, vy-10, text=t, tags="text")
    self.c.create_oval(vx-6, vy-6, vx+6, vy+6, width=1, outline='black', fill='SkyBlue2', tags="dot")
```

Observe that we do not use `v`. 
entering data with scale
developing animations

1. The Widget Scale
   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

Run the code `movedot.py`!
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.
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.
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.
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.
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.
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.
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.
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.
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 \texttt{start} is pressed;
- and stops when \texttt{stop} is pressed;
- any changes in the values set by the scale take immediate effect when the animation is running.
from Tkinter import *  
import random

class MovingDot():
    ""
    GUI to illustrate a random walk
    ""
    def __init__(self, wdw):
        "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):
    "determines the layout of the GUI"
    wdw.title(’a moving dot’)
    self.d = 400
    self.c = Canvas(wdw, width=self.d, \n        height=self.d, bg =’white’)
    self.c.grid(row=0, column=0, colspan=2)
    self.T = Label(wdw, text="sleep time")
    self.T.grid(row=1, column=0, colspan=2)
    self.x = self.d/2
    self.y = self.d/2
    self.go = False
    self.sleep = IntVar()
    self.step = IntVar()

Data attributes are coordinates x, y of the dot, the state of animation go, the parameters sleep and step.
The horizontal scale determines the speed, i.e.: how much time between each drawing of a new dot.

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

self.h = Scale(wdw,
    orient='vertical', from_=0, to=100,
    tickinterval=10, resolution=1,
    length=400, variable=self.step)
self.h.grid(row=0, column=2)
self.h.set(10)
```

Default values for horizontal and vertical scale variables `sleep` and `step` are respectively 100 and 10.
The Scales
the code for __init__ continued

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

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

self.h = Scale(wdw, 
    orient='vertical', from_=0, to=100, 
    tickinterval=10, resolution=1, 
    length=400, variable=self.step)
self.h.grid(row=0, column=2)
self.h.set(10)
```

Default values for horizontal and vertical scale variables sleep and step are respectively 100 and 10.
The Scales
the code for \_\_init\_\_ continued

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

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

Default values for horizontal and vertical scale variables sleep and step are respectively 100 and 10.
The buttons with text **start** and **stop** trigger the beginning and the ending of the animation.

```python
self.b0 = Button(wdw,text="start", 
    command = self.start)
self.b0.grid(row=3,column=0,sticky=W+E)
self.b1 = Button(wdw,text="stop", 
    command = self.stop)
self.b1.grid(row=3,column=1,sticky=W+E)
```

**start** and **stop** are the respective callback functions for the buttons **b0** and **b1**.
The buttons with text **start** and **stop** trigger the beginning and the ending of the animation.

```python
self.b0 = Button(wdw, text="start", command = self.start)
self.b0.grid(row=3, column=0, sticky=W+E)
self.b1 = Button(wdw, text="stop", command = self.stop)
self.b1.grid(row=3, column=1, sticky=W+E)
```

**start** and **stop** are the respective callback functions for the buttons **b0** and **b1**.
The buttons with text **start** and **stop** trigger the beginning and the ending of the animation.

```python
self.b0 = Button(wdw,text="start",\n        command = self.start)
self.b0.grid(row=3,column=0,sticky=W+E)
self.b1 = Button(wdw,text="stop",\n        command = self.stop)
self.b1.grid(row=3,column=1,sticky=W+E)
```

**start** and **stop** are the respective callback functions for the buttons b0 and b1.
The Widget Scale
entering parameter values
using the variables in Scale

Building an Animation
sliding canvas coordinates
animating a random walk

Summary + Assignments

---

def animate(self):
    "performs the animation"
    vx = self.x; vy = self.y
    while self.go:
        self.c.delete("dot")
        self.c.create_oval(vx-6,vy-6,vx+6,vy+6, width=1, outline='black',
                      fill='SkyBlue2', tags="dot")
        n = self.step.get()
        vx = vx + random.randint(-n,n)
        vy = vy + random.randint(-n,n)
        if vx >= self.d: vx = vx - self.d
        if vy >= self.d: vy = vy - self.d
        if vx < 0: vx = vx + self.d
        if vy < 0: vy = vy + self.d
        self.c.after(self.sleep.get())
        self.c.update()
def animate(self):
    "performs the animation"
    vx = self.x; vy = self.y
    while self.go: 
        self.c.delete("dot")
        self.c.create_oval(vx-6, vy-6, vx+6, vy+6, 
                          width=1, outline='black', 
                          fill='SkyBlue2', tags="dot")
        n = self.step.get()
        vx = vx + random.randint(-n, n)
        vy = vy + random.randint(-n, n)
        if vx >= self.d: vx = vx - self.d
        if vy >= self.d: vy = vy - self.d
        if vx < 0: vx = vx + self.d
        if vy < 0: vy = vy + self.d
        self.c.after(self.sleep.get())
        self.c.update()
def animate(self):
    "performs the animation"
    vx = self.x; vy = self.y
    while self.go:
        self.c.delete("dot")
        self.c.create_oval(vx-6,vy-6,vx+6,vy+6,
            width=1,outline='black',
            fill='SkyBlue2',tags="dot")
        n = self.step.get()
        vx = vx + random.randint(-n,n)
        vy = vy + random.randint(-n,n)
        if vx >= self.d: vx = vx - self.d
        if vy >= self.d: vy = vy - self.d
        if vx < 0: vx = vx + self.d
        if vy < 0: vy = vy + self.d
        self.c.after(self.sleep.get())
        self.c.update()
def animate(self):
    "performs the animation"
    vx = self.x; vy = self.y
    while self.go:
        self.c.delete("dot")
        self.c.create_oval(vx-6,vy-6,vx+6,vy+6,
                          width=1,outline='black',
                          fill='SkyBlue2',tags="dot")
        n = self.step.get()
        vx = vx + random.randint(-n,n)
        vy = vy + random.randint(-n,n)
        if vx >= self.d: vx = vx - self.d
        if vy >= self.d: vy = vy - self.d
        if vx < 0: vx = vx + self.d
        if vy < 0: vy = vy + self.d
        self.c.after(self.sleep.get())
        self.c.update()
def animate(self):
    "performs the animation"
    vx = self.x; vy = self.y
    while self.go:
        self.c.delete("dot")
        self.c.create_oval(vx-6,vy-6,vx+6,vy+6,\
            width=1,outline='black',\
            fill='SkyBlue2',tags="dot")
        n = self.step.get()
        vx = vx + random.randint(-n,n)
        vy = vy + random.randint(-n,n)
        if vx >= self.d: vx = vx - self.d
        if vy >= self.d: vy = vy - self.d
        if vx < 0: vx = vx + self.d
        if vy < 0: vy = vy + self.d
        self.c.after(self.sleep.get())
        self.c.update()
Code for start, stop, and main

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

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

def main():
    top = Tk()
    show = MovingDot(top)
    top.mainloop()

if __name__ == "__main__": main()
Code for start, stop, and main

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

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

def main():
    top = Tk()
    show = MovingDot(top)
    top.mainloop()

if __name__ == "__main__": main()
```
The Widget Scale
entering parameter values
using the variables in Scale

Building an Animation
sliding canvas coordinates
animating a random walk

Summary + Assignments

Code for start, stop, and main

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

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

def main():
    top = Tk()
    show = MovingDot(top)
    top.mainloop()

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

Homework collection on Monday 17 November, at 1PM: #1,2 of Lecture 24; #4 of Lecture 25, #4,5 of Lecture 26.