



Week 3

parameters, return, math, graphics

Special thanks to Scott Shawcroft, Ryan Tucker, and Paul Beck for their work on these slides.

Except where otherwise noted, this work is licensed under:

<http://creativecommons.org/licenses/by-nc-sa/3.0>

Parameters

```
def name(parameter, parameter, ..., parameter):  
    statements
```

- Parameters are declared by writing their names (no types)

```
>>> def print_many(word, n):  
...     for i in range(n):  
...         print word  
  
>>> print_many("hello", 4)  
hello  
hello  
hello  
hello
```

Exercise

- Recreate the lines/boxes of stars example from lecture:

```
*****
```

```
*****
```

```
*****
```

```
*****
```

```
*           *
```

```
*****
```

```
*****
```

```
*       *
```

```
*       *
```

```
*****
```

Exercise Solution

stars.py

```
1  # Draws a box of stars with the given width and height.
2  def box(width, height):
3      print width * "*"
4      for i in range(height - 2):
5          print "*" + (width - 2) * " " + "*"
6      print width * "*"
7
8  # main
9  print 13 * "*"
10 print 7 * "*"
11 print 35 * "*"
12 box(10, 3)
13 box(5, 4)
```

Default Parameter Values

```
def name(parameter=value, ..., parameter=value):  
    statements
```

- Can make parameter(s) optional by specifying a default value

```
>>> def print_many(word, n=1):  
...     for i in range(n):  
...         print word  
  
>>> print_many("shrubby")  
shrubby  
>>> print_many("shrubby", 4)  
shrubby  
shrubby  
shrubby  
shrubby
```

- **Exercise:** Modify `stars.py` to add an optional parameter for the character to use for the outline of the box (default `"*"`).

Parameter Keywords

name (parameter=value , ... , parameter=value)

- Can specify the names of parameters as you call a function
- This allows you to pass the parameters in any order

```
>>> def print_many(word, n):
...     for i in range(n):
...         print word

>>> print_many(str="shrubbery", n=4)
shrubbery
shrubbery
shrubbery
shrubbery

>>> print_many(n=3, str="Ni!")
Ni!
Ni!
Ni!
```

Math commands

```
from math import *
```

Function name	Description
<code>ceil(value)</code>	rounds up
<code>cos(value)</code>	cosine, in radians
<code>degrees(value)</code>	convert radians to degrees
<code>floor(value)</code>	rounds down
<code>log(value, base)</code>	logarithm in any base
<code>log10(value)</code>	logarithm, base 10
<code>max(value1, value2, ...)</code>	largest of two (or more) values
<code>min(value1, value2, ...)</code>	smallest of two (or more) values
<code>radians(value)</code>	convert degrees to radians
<code>round(value)</code>	nearest whole number
<code>sin(value)</code>	sine, in radians
<code>sqrt(value)</code>	square root
<code>tan(value)</code>	tangent

Constant	Description
<code>e</code>	2.7182818...
<code>pi</code>	3.1415926...

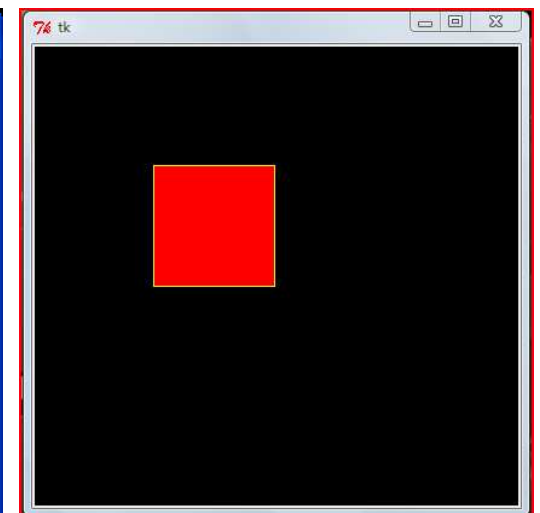
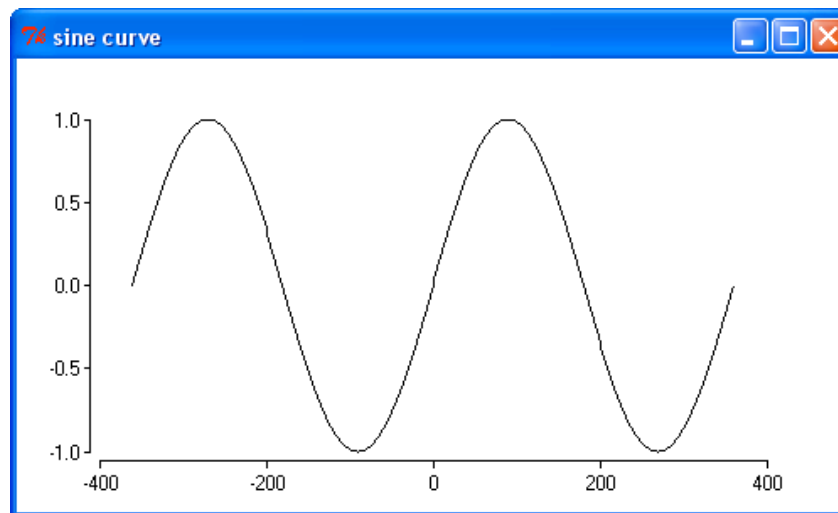
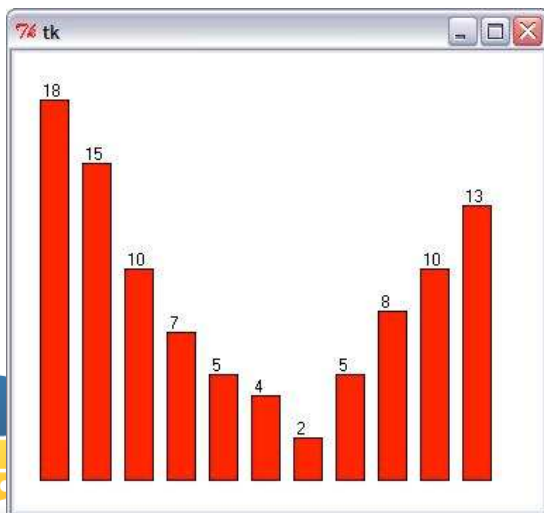
Returning Values

```
def name( parameters ) :  
    statements  
    ...  
    return value
```

```
>>> def ftoc(temp):  
...     tempc = 5.0 / 9.0 * (temp - 32)  
...     return tempc  
  
>>> ftoc(98.6)  
37.0
```


DrawingPanel

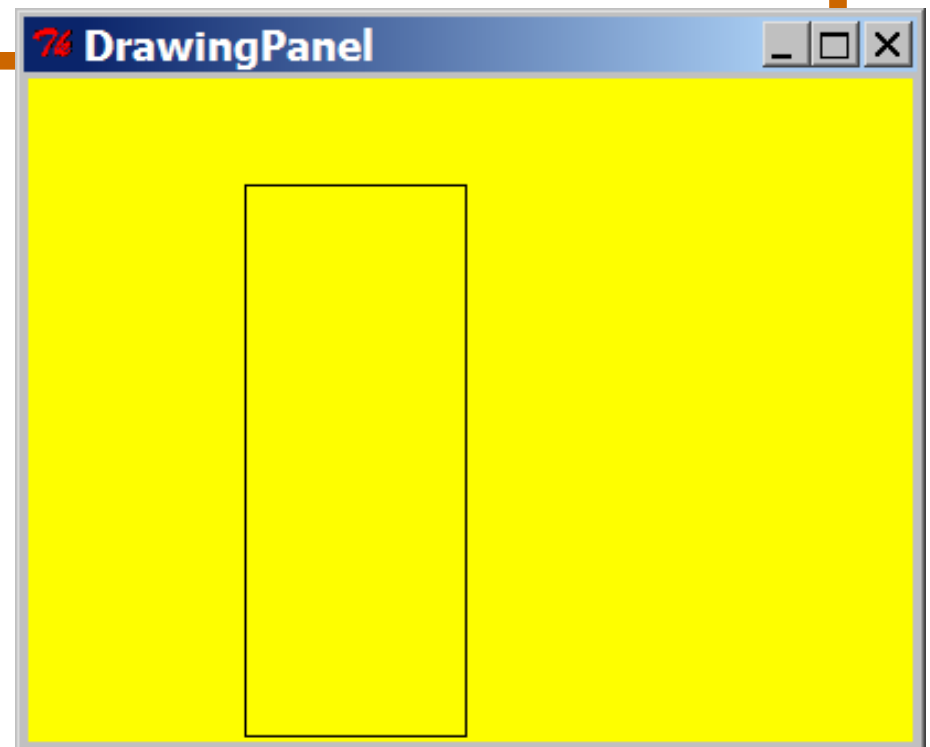
- Instructor-provided `drawingpanel.py` file must be in the same folder as your Python program
- At the top of your program, write:
 - `from drawingpanel import *`
- Panel's canvas field behaves like `Graphics g` in Java
- need to say `panel.mainloop()` at bottom of program!



DrawingPanel Example

draw1.py

```
1 from drawingpanel import *
2
3 panel = DrawingPanel(400, 300)
4 panel.set_background("yellow")
5 panel.canvas.create_rectangle(100, 50, 200, 300)
6 panel.mainloop()
```



Drawing Methods

Java	Python
drawLine	panel .canvas.create_line(x1 , y1 , x2 , y2)
drawRect, fillRect	panel .canvas.create_rectangle(x1 , y1 , x2 , y2)
drawOval, fillOval	panel .canvas.create_oval(x1 , y1 , x2 , y2)
drawString	panel .canvas.create_text(x , y , text=" text ")
setColor	<i>(see next slide)</i>
setBackground	panel .set_background(color)

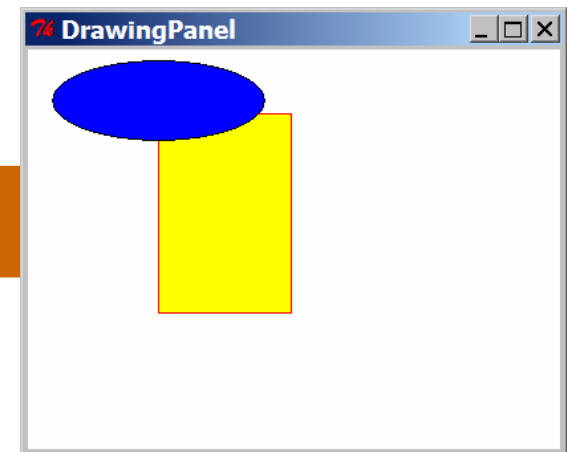
- Notice, methods take x2/y2 parameters, not width/height

Colors and Fill

- Python doesn't have `fillRect`, `fillOval`, or `setColor`.
 - Instead, pass outline and fill colors when drawing a shape.
 - List of all color names: <http://wiki.tcl.tk/16166>
 - [Visual display of all colors](#)

drawcolors.py

```
1 from drawingpanel import *
2
3 panel = DrawingPanel(400, 300)
4 panel.canvas.create_rectangle(100, 50, 200, 200,
5                               outline="red", fill="yellow")
6 panel.canvas.create_oval(20, 10, 180, 70, fill="blue")
7 panel.mainloop()
```

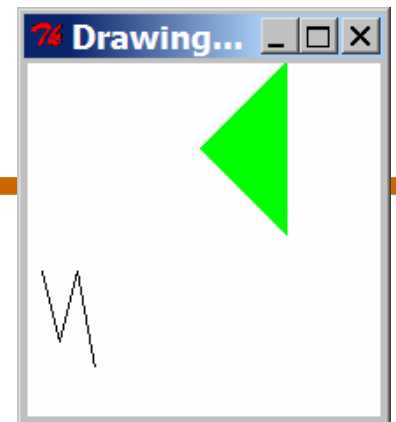


Polygons

- Draw arbitrary polygons with `create_polygon`
- Draw line groups by passing more params to `create_line`

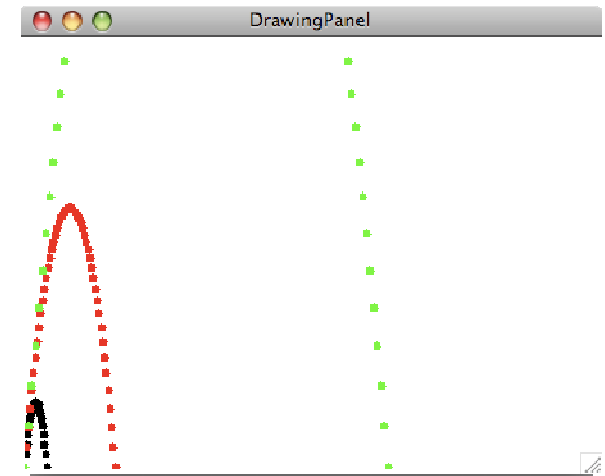
drawpoly.py

```
1 from drawingpanel import *
2
3 panel = DrawingPanel(200, 200)
4 panel.canvas.create_polygon(100, 50, 150, 0,
                             150, 100, fill="green")
5 panel.canvas.create_line(10, 120, 20, 160,
                           30, 120, 40, 175)
6 panel.mainloop()
```



Exercise

- Write a modified [Projectile](#) program:
 - Draw projectiles traveling at:
 - 85 degrees at 30 m/s
 - 85 degrees at 60 m/s
 - 85 degrees at 120 m/s
 - First write a **projectile** function to draw a single projectile.
 - $v_x = v \cdot \cos(\Theta)$
 - $v_y = v \cdot \sin(\Theta)$
 - $\text{totalTime} = -2 * v_y / \text{GRAVITY}$
 - $\text{dt} = \text{totalTime} / \text{steps}$
 - Create a colored circle, with default color black, at each step in time where the projectile should be.



Animation

- Pause the panel with `sleep`

animation.py

```
1 from drawingpanel import *
2
3 panel = DrawingPanel(350, 300)
4 for i in range(20):
5     # clear any previous image
6     panel.canvas.create_rectangle(0, 0, 400, 400,
7                                   outline="white", fill="white")
8     panel.canvas.create_polygon(20 * i, 50, 20 * i,
9                                   100, 20 * i + 50, 75)
10
11     # sleep for 100ms
12     panel.sleep(100)
```

