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

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

```python
>>> 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:

```
**************

********

******************************

**************
*    *
**************

*****
*  *
*  *
*  *
*****
```
stars.py

```python
# Draws a box of stars with the given width and height.
def box(width, height):
    print width * "*
    for i in range(height - 2):
        print "*" + (width - 2) * " " + "*
    print width * "*

# main
print 13 * "*
print 7 * "*
print 35 * "*
box(10, 3)
box(5, 4)
```
Default Parameter Values

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

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

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

- **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

```python
>>> 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!
```
from math import *

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

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>2.7182818...</td>
</tr>
<tr>
<td>pi</td>
<td>3.1415926...</td>
</tr>
</tbody>
</table>
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
• Instructor-provided `drawingpanel.py` file must be in the same folder as your Python program

• At the top of your program, write:
  
  ```python
  from drawingpanel import *
  ```

• Panel's `canvas` field behaves like `Graphics g` in Java

• need to say `panel.mainloop()` at bottom of program!
from drawingpanel import *

panel = DrawingPanel(400, 300)
panel.set_background("yellow")
panel.canvas.create_rectangle(100, 50, 200, 300)
panel.mainloop()
## Drawing Methods

<table>
<thead>
<tr>
<th>Java</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>drawLine</td>
<td><code>panel.canvas.create_line(x1, y1, x2, y2)</code></td>
</tr>
<tr>
<td>drawRect, fillRect</td>
<td><code>panel.canvas.create_rectangle(x1, y1, x2, y2)</code></td>
</tr>
<tr>
<td>drawOval, fillOval</td>
<td><code>panel.canvas.create_oval(x1, y1, x2, y2)</code></td>
</tr>
<tr>
<td>drawString</td>
<td><code>panel.canvas.create_text(x, y, text=&quot;text&quot;)</code></td>
</tr>
<tr>
<td>setColor</td>
<td><em>(see next slide)</em></td>
</tr>
<tr>
<td>setBackground</td>
<td><code>panel.set_background(color)</code></td>
</tr>
</tbody>
</table>

- 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](http://wiki.tcl.tk/16166)
  - **Visual display of all colors**

```python
drawcolors.py

from drawingpanel import *

panel = DrawingPanel(400, 300)
panel.canvas.create_rectangle(100, 50, 200, 200, outline="red", fill="yellow")
panel.canvas.create_oval(20, 10, 180, 70, fill="blue")
panel.mainloop()
```
Polygons

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

**drawpoly.py**

```python
from drawingpanel import *

panel = DrawingPanel(200, 200)

panel.canvas.create_polygon(100, 50, 150, 0, 150, 100, fill="green")

panel.canvas.create_line(10, 120, 20, 160, 30, 120, 40, 175)

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.
    • $v_x = v \cdot \cos(\Theta)$
    • $v_y = v \cdot \sin(\Theta)$
    • totalTime = $-2 \cdot v_y / \text{GRAVITY}$
    • $\text{dt} = \text{totalTime} / \text{steps}$
    • Create a colored circle, with default color black, at each step in time where the projectile should be.
Animation

• Pause the panel with `sleep`

```python
from drawingpanel import *

panel = DrawingPanel(350, 300)
for i in range(20):
    # clear any previous image
    panel.canvas.create_rectangle(0, 0, 400, 400,
                                 outline="white", fill="white")

    panel.canvas.create_polygon(20 * i, 50, 20 * i,
                                100, 20 * i + 50, 75)

    # sleep for 100ms
    panel.sleep(100)
```