More on Functions
Announcements

• Lab 5 posted; due Saturday

• Quiz 2 will be handed back Wednesday
Today’s plan

• Review Friday
• Go over shades.py program
• Re-examine functions, mutation, call stack
• functionWorksheet.py
Review

• Our graphics windows are a grid of pixels, where a pixel is a square of continuous color.

• We specify the size of the pixel grid in the \texttt{GraphWin(title, width, height)} constructor

• We can use one of the pre-defined colors, or define our own colors with the \texttt{color\_rgb(red, green, blue)} function, where \texttt{red}, \texttt{green}, and \texttt{blue} are between 0 and 255.
Review

• A computer animation is a series of rapidly changing images.

• We can use the `sleep(seconds)` function from the `time` library to pause our program for a fraction of a second, before changing what is displayed in the graphics window—often with `move(dx, dy)`
shades.py

- Nested for loops
- Randomly generating colors
- Avoiding hard coding
- Calling a function that calls a function
## Recap

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>float</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>bool</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>string</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>list</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Object</td>
<td>usually no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>
Use methods to mutate

• Lists and objects can be mutated through their methods.
  - lst.append(4), lst.pop(), lst.reverse(), lst.sort()
  - p1.move(5, 5), circ.setFill("blue"),
    textObject.setText("Hello")

• Lists can also be mutated at specific indices:
  - L[2] = "new item"

• Not all methods mutate:
  - lst.count(), lst.index()
  - p1.getX(), circ.getRadius()
Mutating in a function

- When passed in as arguments to a function, lists and objects can be mutated within that function.

```python
def foo(lst):
    lst.append(4)

def main():
    L = [1, 2, 3]
    foo(L)
    print("The list is: \%s\" % L)

main()
```
Reassigning vs. Mutating

def foo2(lst, x):
    lst[1] = "two"
    x += 5

def main():
    L = [1, 2, 3]
    x = 10
    foo2(L, x)
    print("The list is: %s" % L)
    print("The number is: %d" % x)

main()
Reassigning is not mutating

• If a variable, \( x \), currently points to value \( A \) and we reassign it so that it points to value \( B \), \( A \) is unchanged.

• If \( x \) points to an object/list and we mutate that object/list with a method or reassignment at an index, we haven’t reassigned \( x \), but we have changed the contents of the compound value it points to.
Reassigning vs. Mutating

```python
def foo3(L):
    L = L[1:]

def main():
    L = [1, 2, 3]
    foo3(L)
    print("The list is: \%s" \% L)

main()
```
Reassigning vs. Mutating

def foo4(L):
    L = L[1:]
    return L

def main():
    L = [1, 2, 3]
    L = foo4(L)
    print("The list is: %s" % L)

main()
Reassigning vs. Mutating

```python
def foo5(L):
    M = L
    M[0] = "one"

def main():
    L = [1, 2, 3]
    foo5(L)
    print("The list is: \%s\" % L)

main()
```
Objects are also mutable

```python
from graphics import *

def movePoint(p):
    p.move(5, 0)

def main():
    q = Point(0, 0)
    movePoint(q)
    print("q's x coordinate is: \%d\" % q.getX())

main()
```
Creating a copy

```python
from graphics import *

def movePoint2(p):
    q = p.clone()
    q.move(5, 0)

def main():
    q = Point(0, 0)
    movePoint2(q)
    print("q's x coordinate is: \%d" % q.getX())

main()
```
functionWorksheet.py