STSCI 4060

Lecture File 5

Xiaolong Yang
(xy44@cornell.edu)

More Complicated Python Topics

- Python lambda

  function/operator

- Python regular expressions

- Python graphics

Python lambda Operator (Function)

Python supports the creation of **anonymous functions** (i.e., functions that are not bound to a function name) at **runtime**, using a **c**onstruct called "**lambda**".

```
>>> def f (x): return x**2
...
>>> print f(6)
36
>>>
>>> g = lambda x: x**2
>>>
>>> print g(6)
36
```

```
>>> f = lambda x, y : x + y
>>> f(1,1)
2
>>f(2.5, 10.25)
12.75
>>>import math
>>>v = lambda r: (4*math.pi*r**3)/3
>>>v(10)
4188.790204786391
```

Use the lambda Operator with the map() Function

A map() function takes two arguments: **map(function, sequence)**. The first argument is the name of a function and the second is a sequence (e.g., a list). A map() applies the function to all the elements of the sequence. It returns a new sequence with the elements changed by the function.

```
>>> def fahrenheit(T):
    return round((9.0/5)*T + 32, 1)
>>> def celsius(T):
    return round(5.0/9)*(T-32), 1)
>>> F = map(fahrenheit, temp)
>>> F
[97.7, 98.6, 99.5, 102.2]
>>> C = map(celsius, F)
>>> C
[36.5, 37.0, 37.5, 39.0]
```

```
>>> Celsius = [39.2, 36.5, 37.3, 37.8]
>>> Fahrenheit = map(lambda x: (float(9)/5)*x \
+ 32, Celsius)
>>> Fahrenheit
[102.56, 97.7, 99.14, 100.03999999999]
>>> C = map(lambda x: (float(5)/9)*(x-32), \
Fahrenheit)
>>> print C
[39.2, 36.5, 37.300000000000004, 37.8]
```

Use the lambda Operator with the filter() Function

A filter() function also takes two arguments: **filter(function, sequence)**. The first argument is the name of a function and the second is a sequence (e.g., a list). A filter() applies the argument function to all the elements of the sequence. It returns a new sequence from those elements of the argument sequence for which the function returns true.

```
>>> mult2 = filter(lambda x: x % 2 == 0, [0,1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> mult2
[0, 2, 4, 6, 8]
>>>
>>> greater = filter(lambda x: x > 5, [0,1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> greater
[6, 7, 8, 9]
>>> >>> hasStr = filter(lambda x: "or" in x, ['worker', 'country', 'operator', 'organ'])
>>> hasStr
['worker', 'operator', 'organ']
```

Use the lambda Operator to Construct a Function as a Return Value From Another Function

```
>>> def transform(n):
    return lambda x: x*n + n**2

>>> f=transform(10)  # give the value of n
>>> f(2)  # give the value of x
120        # 2*10 + 10**2
>>> f(3)
130
>>> f(10)
200
>>> f1=transform(50)
>>> f1(2)
2600               # 2*50 + 50**2
```

Python Regular Expressions

Regular expressions (REs) are a tiny highly specialized programming language embedded inside Python and made available through the **re** module, with which you can manipulate strings, such as specifying the rules for the set of possible strings that you want to match, or splitting or modifying a string.

Metacharaters in Regular Expressions

Metacharacters are some special characters that signal some special patterns to match, or affect other portions of the RE by repeating them or changing their meaning:

. ^ $ * + ? {} [] \ | ()

Metacharaters in Regular Expressions (cont'd)

[] indicates a set of characters, e.g., '[abc]' or '[a-c]' matches 'a', 'b', or 'c'.

^ complements the character set, e.g., '[^a]' matches any character except 'a'.

. matches any character except a newline.

**$** Matches the end of the string or just before the newline at the end of the string, e.g., 'theEnd$' matches "theEnd".

***** specifies that the previous character can be matched zero or more times, e.g., 'ab*' will match 'a', 'ab', or 'a' followed by any number of 'b's.

+ specifies that the previous character can be matched one or more times, e.g., 'ab+' will match 'a' followed by any non-zero number of 'b's.

? matches either once or zero times, e.g., 'home-?brew' matches either 'homebrew' or 'home-brew'.

{m,n} there must be at least m repetitions, and at most n times of the previous character. For example, 'a/{1,3}b' will match' a/b', 'a//b', and 'a///b'.

{m} specifies that exactly m copies of the previous character should be matched, e.g., 'a{6}' will match exactly six 'a' characters

**** can be followed by various characters to signal some special sequences, or to escape all the metacharacters, e.g., use' \[' or '\\' to match '[' or '\'.

| 'A'|'B' will match either 'A' or 'B'.

(…) matches whatever regular expression inside the parentheses, marking the start and end of a group.

Some of the Special Sequences Beginning With '\'

\d Matches any decimal digit; this is equivalent to the class [0-9].

\D Matches any non-digit character; this is equivalent to the class [^0-9].

\s Matches any whitespace character; this is equivalent to the class [\t\n\r\f\v].

\S Matches any non-whitespace character; this is equivalent to the class [^ \t\n\r\f\v].

\w Matches any alphanumeric character; this is equivalent to the class [a-zA-Z0-9_].

\W Matches any non-alphanumeric character; this is equivalent to the class [^a-zA-Z0-9_].

Methods of a Regular Expression Object

Once you have an object representing a compiled regular expression, you can use the object's methods and attributes. The most significant ones are:

Methods	Purpose
match()	Determines if the RE matches the beginning character(s) of the string. It returns None if no match can be found; otherwise, a match object instance is returned.
search()	Scans through a string, searching for first occurrence of RE pattern within string. It returns None if no match can be found; otherwise, a match object instance is returned.
findall()	Finds all substrings where the RE matches, and returns them as a list.
finditer()	Finds all substrings where the RE matches, and returns them as an iterator.

Methods of a RE Object Instance

These methods can be used to get the match positions and the matched string(s)

Methods	Purpose
group()	Returns the string matched by the RE
start()	Returns the starting position of the match
end()	Returns the ending position of the match
span()	Returns a tuple containing the (start, end) positions of the match

Some Simple Match Examples Using REs

```
>>> import re
>>> re.match('[abcd]', 'books')
<_sre.SRE_Match object at 0x1018852a0>
>>> obj = re.compile('[abcd]')
>>> m = obj.match('books')
>>> m.group()
'b'
>>> m1 = obj.match('my books')
>>> m1.group()

Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    m1.group()
AttributeError: 'NoneType' object has no attribute 'group'
>>> print m1
None
>>> m2 = obj.match('cool books')
>>> m2
<_sre.SRE_Match object at 0x106741f38>
>>> m2.group()
'c'
```

Some Simple Match Examples Using REs (cont'd)

```
>>> if obj.match('cool books'):
        print 'Yes, there is a match!'


Yes, there is a match!
```

```
>>> obj1 = re.compile('[a-z]+')
>>> m = obj1.match('two books')
>>> m
<_sre.SRE_Match object at 0x105b41780>
```

```
>>> m.group()
'two'
>>> m1 = obj1.findall('two books')
```

```
>>> m1
['two', 'books']
```

```
>>> m7=obj.findall('a book')
>>> m7
['a', 'b']
```

```
>>> m8=obj.findall('truccck')
>>> m8
['c', 'c', 'c']
```

Some Simple Match Examples Using REs (cont'd)

```
>>> obj2 = re.compile('\d+')
>>> obj2.findall('100 apples were eaten by 20 people in 4 groups')
['100', '20', '4']
>>>
>>> obj3 = re.compile('\D+')
>>> obj3.findall('100 apples were eaten by 20 people in 4 groups')
[' apples were eaten by ', ' people in ', ' groups']
>>> myit = obj2.finditer('100 apples were eaten by 20 people in 4 groups')
>>> myit
<callable-iterator object at 0x105b5bbd0>
>>> for matched in myit:
        print matched.span()


(0, 3)
(25, 27)
(38, 39)
```

```
>>> myit.next().span()
(0, 3)
>>> myit.next().span()
(25, 27)
>>> myit.next().span()
(38, 39)
>>> myit.next().span()

Traceback (most recent call last):
  File "<pyshell#35>", line 1, in <module>
    myit.next().span()
StopIteration
```

Python Graphics

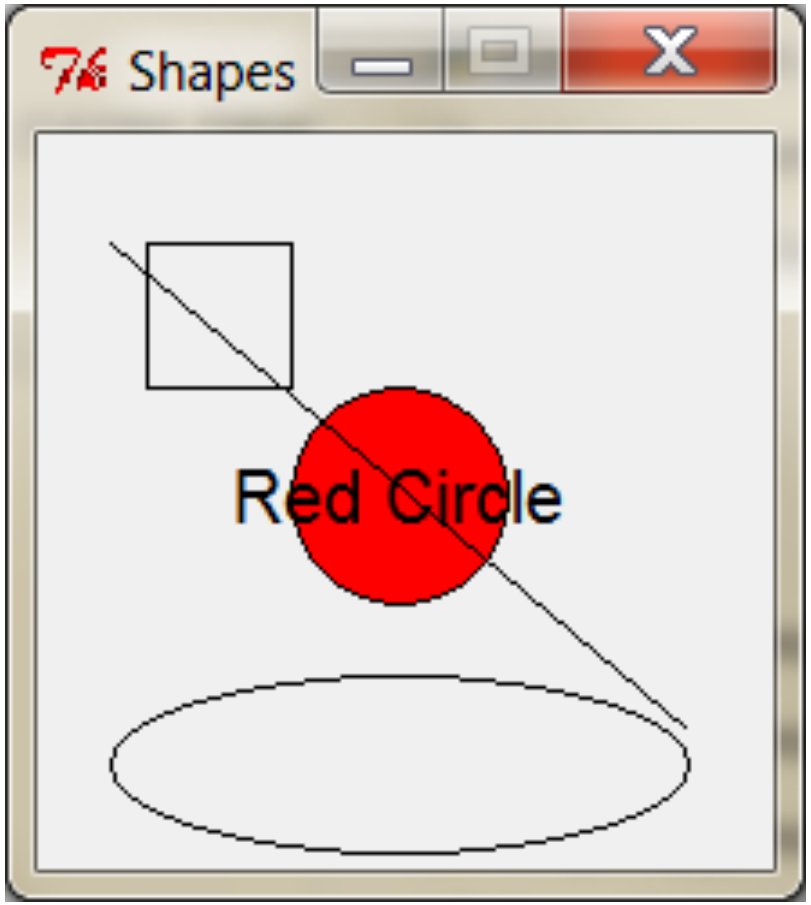
- You can use a module called "graphics" designed by John Zelle to deal with some graphics-related tasks, but this module is not a part of the standard Python distribution. So you must import it.
- You may use the "from graphics import *" statement so that all types of the objects in the module become accessible to you.
- You should put a copy of this module in the root of your Python program or in the folder when you store your graphics program you are writing. It is available on the course web site (within the “Software” folder).
- There are many graphics functions and objects in this module. These are summarized in files called "Graphic Library Quick Reference" and "Graphics Module Reference," which can be downloaded from the course website (within the "Books and References" folder). The following are some classes and constructors defined in the module :
  - GraphWin(title, width, height) of the GraphWin class, for making a new GraphWin object. The default window size is 200x200.
  - Point(x,y) of the Point class, dealing with points.
  - Line(p1, p2) of the Line class.
  - Rectangle(p1, p2) of the Rectangle class; p1 and p2 are upper left and lower right points of a rectangle.
  - Circle(p, radius) of the Circle class; p is the center.

Example: Some Simple Python Graphics

```
from graphics import *
#### Create a graphics window of the default size
win = GraphWin('Shapes')
#### Draw a red circle centered at point (100,100) with radius 30
center = Point(100,100)
circ = Circle(center, 30)
circ.setFill('red')
circ.draw(win)
#### Put a textual label in the center of the circle
label = Text(center, "Red Circle")
label.draw(win)
#### Draw a square using a Rectangle object
rect = Rectangle(Point(30,30), Point(70,70))
rect.draw(win)
#### Draw a line segment using a Line object
line = Line(Point(20,30), Point(180, 165))
line.draw(win)
#### Draw an oval using the Oval object
oval = Oval(Point(20,150), Point(180,199))
oval.draw(win)
```

Example: Some Simple Python Graphics

The output



Example: Get the Coordinates of Your Mouse Click

mouse_click.py - K:\STSCI4060\STSCI 4060_spring_2013\Lectures\File 4 files\mouse_click.py

File Edit Format Run Options Windows Help

```
from graphics import *

win = GraphWin("Click Me!")
for i in range(10):
    p = win.getMouse()
    print "You clicked (%d, %d)" % (p.getX(), p.getY())
```

Ln: 7 Col: 0

Python Shell

File Edit Shell Debug Options Windows Help

```
Python 2.7.3 |EPD 7.3-2 (64-bit)| (default, Apr 12 2012, 15:20:16) [MSC v.1500 64 bit (AMD64)] on win3
2
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
You clicked (109, 124)
You clicked (15, 62)
You clicked (96, 77)
You clicked (99, 81)
You clicked (123, 102)
You clicked (91, 117)
You clicked (70, 65)
You clicked (126, 46)
You clicked (72, 118)
You clicked (44, 45)
>>>
```

Ln: 15 Col: 4

Example: Circles of Random Sizes and Colors

randomCircles.py - K:\STSCI4060\STSCI 4060_spring_2013\examp...

File Edit Format Run Options Windows Help

```
"""Draw random circles.
"""
from graphics import *
import random, time

def main():
    win = GraphWin("Random Circles", 300, 300)
    for i in range(75):
        r = random.randrange(256)
        b = random.randrange(256)
        g = random.randrange(256)
        color = color_rgb(r, g, b)

        radius = random.randrange(3, 40)
        x = random.randrange(5, 295)
        y = random.randrange(5, 295)

        circle = Circle(Point(x,y), radius)
        circle.setFill(color)
        circle.draw(win)
        time.sleep(.05)

    Text(Point(150, 20), "Click to close.").draw(win)
    win.getMouse()
    win.close()

main()
```

Ln: 28 Col: 0

Animation: A Bouncing Ball

- Specify a graphic window size.
- Make the ball, using makeDisk().
  - Ball size (radius).
  - Color(s): outline and body.
- Find a random location in the graphic window for the ball to start with, using getRandomPoint().
- The ball moves from the randomly determined location in jumps (dx, dy), using bounceBall().
  - The ball moves, using bounceInBox().
  - It bounces when its edge reaches a side of the box (assuming the ball is rigid.
  - Move the number of times specified.
  - Close the window.

Animation: A Bouncing Ball

bounce1.py - K:\STSCI4060\STSCI 4060_spring_2013\examples\bounce1.py

File Edit Format Run Options Windows Help

```
'''
Show a ball bouncing off the sides of the window.
'''

from graphics import *
import time, random

def bounceInBox(shape, dx, dy, xLow, xHigh, yLow, yHigh):
    ''' Animate a shape moving in jumps (dx, dy), bouncing when
    its center reaches the low and high x and y coordinates.
    '''

    delay = .005
    for i in range(600):
        shape.move(dx, dy)
        center = shape.getCenter()
        x = center.getX()
        y = center.getY()
        if x < xLow:
            dx = -dx
        elif x > xHigh:
            dx = -dx
        if y < yLow:
            dy = -dy
        elif y > yHigh:
            dy = -dy
        time.sleep(delay)

def getRandomPoint(xLow, xHigh, yLow, yHigh):
    '''Return a random Point with coordinates in the range specified.'''
    x = random.randrange(xLow, xHigh+1)
    y = random.randrange(yLow, yHigh+1)
    return Point(x, y)
```

Ln: 21 Col: 0

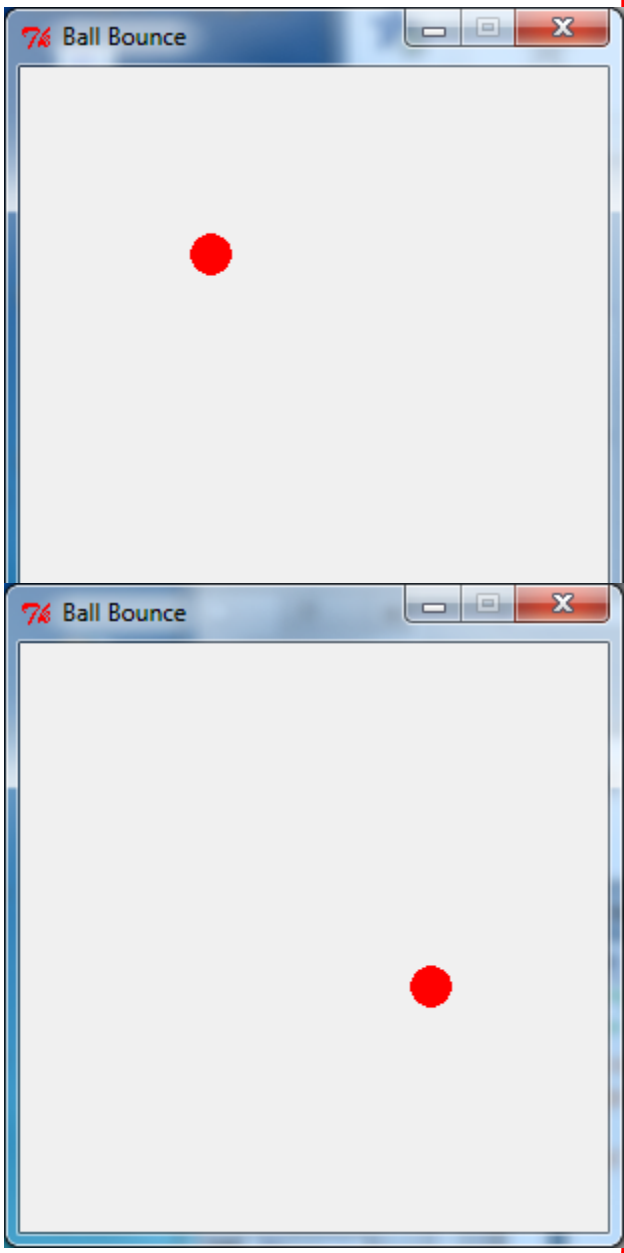
Animation: A Bouncing Ball (cont'd)

bounce1.py - K:\STSCI4060\STSCI 4060_spring_2013\examples\bounce1.py

File Edit Format Run Options Windows Help

```
    return Point(x, y)

def makeDisk(center, radius, win):
    '''return a red disk that is drawn in win with given center and radius.
    disk = Circle(center, radius)
    disk.setOutline("red")
    disk.setFill("red")
    disk.draw(win)
    return disk

def bounceBall(dx, dy):
    '''Make a ball bounce around the screen, initially moving by (dx, dy)
    at each jump.'''

    winWidth = 290
    winHeight = 290
    win = GraphWin('Ball Bounce', winWidth, winHeight)
    win.setCoords(0,0,winWidth, winHeight)

    radius = 10
    xLow = radius # center is separated from the wall by the radius at a bo
    xHigh = winWidth - radius
    yLow = radius
    yHigh = winHeight - radius

    center = getRandomPoint(xLow, xHigh, yLow, yHigh)
    ball = makeDisk(center, radius, win)

    bounceInBox(ball, dx, dy, xLow, xHigh, yLow, yHigh)

    win.close()

bounceBall(3, 5)
```

Ln: 25 Col: 0



Animation: A Bouncing Ball (cont'd)

Make some changes

- The radius of the ball
- The jump of each move
- The delay time

Improvement: A Bouncing Ball—Control Speed and Initial Direction

- Display some text (as instruction) in the window when before the animation starts. The text disappears when as soon as the animation starts.
- Start the ball in the middle of the window.
- The initial ball speed and moving direction is controlled by the mouse.
- The can ball moves continuously till you click in the graphic window.

Improvement: A Bouncing Ball—Control Speed and Initial Direction

```
'''
Show a ball bouncing off the sides of the window.
User graphically shows the initial direction and speed of the ball.
Control the animation with a while loop.'''

from graphics import *
import time, random

def bounceInBox(shape, dx, dy, xLow, xHigh, yLow, yHigh, win):
    ''' Animate a shape moving in jumps (dx, dy), bouncing when
    its center reaches the low and high x and y coordinates.
    The animation stops when the mouse is clicked.'''

    delay = .001
    #win.clearLastMouse()
    while win.checkMouse() == None: #Keep on moving till a mouse click
        shape.move(dx, dy)
        center = shape.getCenter()
        x = center.getX()
        y = center.getY()
        if x < xLow or x > xHigh:
            dx = -dx
        if y < yLow or y > yHigh:
            dy = -dy
        time.sleep(delay)

def makeDisk(center, radius, win):
    '''Return a red disk that is drawn in win with given center and radius.'''
    disk = Circle(center, radius)
    disk.setOutline("red")
    disk.setFill("red")
    disk.draw(win)
    return disk

def getShift(point1, point2): # NEW utility function
    '''Returns a tuple (dx, dy) which is the shift from point1 to point2.'''
    dx = point2.getX() - point1.getX()
    dy = point2.getY() - point1.getY()
    return (dx, dy)
```

Improvement: A Bouncing Ball—Control Speed and Initial Direction (Cont'd)

```
def getUserShift(point, prompt, win): #NEW direction selection
    '''Return the change in position from the point to a mouse click in win.
    First display the prompt string under point.'''

    text = Text(Point(point.getX(), 60), prompt)
    text.draw(win)
    userPt = win.getMouse()
    text.undraw()
    #text1=Text(Point(150,150), 'Click to exit')
    #text1.draw(win)
    return getShift(point, userPt)

def bounceBall():
    '''Make a ball bounce around the screen.  The user sets the initial speed.'''

    winWidth = 290
    winHeight = 290
    win = GraphWin('Ball Bounce', winWidth, winHeight)
    win.setCoords(0,0,winWidth, winHeight)

    radius = 10
    xLow = radius # center is separated from the wall by the radius of a bounce
    xHigh = winWidth - radius
    yLow = radius
    yHigh = winHeight - radius

    center = Point(winWidth/2, winHeight/2) #NEW central starting point
    ball = makeDisk(center, radius, win)

    #NEW interactive direction and speed setting
    prompt = '''
Click to indicate the direction and
speed of the ball:  The farther you
click from the ball, the faster it starts.'''
    (dx, dy) = getUserShift(center, prompt, win)
    scale = 0.01 # to reduce the size of animation steps
    bounceInBox(ball, dx*scale, dy*scale, xLow, xHigh, yLow, yHigh, win)
    win.close()

bounceBall()
```

Improvement: A Bouncing Ball—Control Speed and Initial Direction (Cont'd)