CSC 221: Computer Programming I

Fall 2011

Fun with turtle graphics

- `turtle` module
- relative motion (setup, reset, left, right, forward, backward)
- pen control (penup, pendown, clear, reset)
- example: random walk
- absolute motion (goto, setheading)
- example: alley walk
- class project: drawing letters

1

`turtle` module

the `turtle` module contains many functions for simple graphics

- load them all via: `from turtle import *`

to initialize:

`setup()`	opens the graphics screen with a turtle/arrow in the center (warning: KLUDGY)
`reset()`	clears the screen and resets the turtle position

relative motion:

`forward(numPixels)`	moves the turtle in the direction it is facing
`backward(numPixels)`	moves the turtle in the opposite direction
`right(numDegrees)`	turns the turtle to its right
`left(numDegrees)`	turns the turtle to its left

EXERCISE: experiment/play with these

2

Drawing with the turtle

if you want the turtle to leave a trail as it moves,

`pendown()`	subsequent movement leaves a line
`penup()`	subsequent movement leaves no line
`home()`	returns the turtle to the center of the screen
`clear()`	clears any lines without resetting the turtle

```
def mystery():
    pendown()
    forward(100)
    right(90)
    forward(50)
    backward(100)
    penup()
```

3

Shape functions

define functions for drawing shapes:

```
def triangle(sideLength):
    """Draws a triangle whose sides are sideLength pixels"""
```

```
def square(sideLength):
    """Draws a square whose sides are sideLength pixels"""
```

```
def polygon(numSides, sideLength):
    """Draws a polygon with numSides sides, each of which is
       sideLength pixels"""
```

4

Example: random walks

a *random walk* is a path in which each step is in a randomly chosen direction

- can use the `randint` function from `random.py` for selecting the turn angle
- can also speed up the walk by calling the speed function (from the `turtle.py`)

```
def randomWalk(numSteps):
    home()
    speed(10)
    for i in range(numSteps):
        right(randint(1, 360))
        forward(10)
```

interestingly, the average distance covered by a random walk of `numSteps` can be predicted

final distance from start location = step size * $\sqrt{\text{numSteps}}$

→ assuming 64 steps & step size = 10, final distance from start is 80 (on avg)

modify `randomWalk` to return the distance from (0,0) at the end

`distance(X, Y)` returns the distance from turtle to (X, Y)

5

Example: alley walks

a variation of a random walk is a 1-dimensional walk, or *alley walk*

- the turtle moves along a line, 50/50 chance each step is forward or backward
- the walk ends when the turtle reaches a specific distance

```
def alleyWalk(distance):
    home()
    speed(10)
    while abs(xcor()) < distance:
        if flipCoin() == "heads":
            forward(10)
        else:
            backward(10)
```

hard to follow the progress of the turtle without reference points

- would like to draw finish lines at each end of the alley
  i.e., draw vertical lines at –distance and distance

- this is simpler using absolute motion:

`goto(X, Y)`	moves turtle to the specified coordinate
`setheading(numDegrees)`	sets the heading (→ is 0, ↑ is 90, …)

6

Example: alley walks (cont.)

again, the average length of alley walks is predictable

numSteps = (goal distance from start location / step size)^2

→ assuming goal distance 100 & step size = 10, average number of steps is 100

modify `alleyWalk` so that it keeps track of the number of steps and returns that count at the end

7

Class project

suppose we want to define functions for drawing letters

- we could draw each letter using `forward`, `backward`, `right`, `left`, …
- probably easier to use `goto`, `setheading`

- we want each letter to fit in a 50x100 pixel rectangle, relative to current position
- when done drawing the letter, the turtle should end up 5 pixels to the right

```
def A():
    x = xcor()
    y = ycor()
    pendown()
    goto(x+25, y+100)
    goto(x+50, y)
    penup()
    goto(x+5, y+50)
    pendown()
    goto(x+45, y+50)
    penup()
    goto(x+55, y)
```

note: many ways to draw the letter

simplifies things to save the current coordinates in variables

can then draw lines using `goto` (or a combination of `goto` & `forward` & `backward`)

8

Class project (cont.)

everyone do a letter (or two)

- be sure to name the function using the capital letter, e.g., `A()`, `B()`, …
- be sure to stay within the 50x100 dimensions and end 55 pixels to right of the starting position

- we can even define a function for underscores
- to top it off, the `spell` function will extract the characters in a string and call the appropriate functions to display those letters (more details later)

```
def _():
    x = xcor()
    y = ycor()
    pendown()
    goto(x+50, y)
    penup()
    goto(x+55, y)

def spell(message):
    for letter in message:
        eval(letter+"()")
```

9