Coverage

- 5.5 - Graphing Future Value
- 5.6 - Coordinate Systems
Choosing Coordinate System

- When placing objects on the window, you have two choices
  - Actual pixels on the screen
  - A logical coordinate system
Using Actual Pixels

• You can use actual pixels - but you are responsible when the screen is resized.

    from graphics import *

    win = GraphWin("A Title", 200, 200)
    circ = Circle(Point(50,50),30)
    circ.setFill('red')
    circ.draw(win)

    where = win.getMouse()
    print where
Using a Logical Coordinate System

- You can place objects into a logical coordinate system that lays on top of the pixels. When using logical coordinates, the lower left is 0,0.

```python
from graphics import *
win = GraphWin("A Title", 200, 200)
win.setCoords(0,0,1,1)
circ = Circle(Point(0.25, 0.75),0.15)
circ.setFill('red')
circ.draw(win)
where = win.getMouse()
print where
```

(0.0,0.0)  (0.25,0.75)  (1.0,1.0)
Which Coordinate System?

- For early applications it is simplest to use a logical coordinate system from (0,0) to (1,1)
- This is easier to visualize as coordinates are a percentage of the overall screen height
- Later when you want more precise control over every pixel, you can switch to actual pixels.
Building a Histogram in Python
Histograms

- A histogram is a very common way to present a lot of data when there is interest in seeing how data is distributed.

- Distributions are an essential element of basic statistics.

- A series of values are grouped into buckets and the number in each bucket is counted.
• Python’s string functions allow you to pull apart input lines and extract data from those lines

• You can easily convert strings to integers as well once the lines are parsed

    words = line.split()
    time = words[5]
    print "Time", time
    tsplit = time.split(':')
    hour = int(tsplit[0])
    print "Hour", hour
Dealing With Bad Data

- Sometimes data is bad or corrupted.
- This would cause your program to blow up with a trace back

Time XX:23:48
Traceback (most recent call last):
  File "assn7.py", line 24, in <module>
    hour = int(tsplit[0])
  ValueError: invalid literal for int() with base 10: 'XX'

From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
From louis@media.berkeley.edu Fri Jan  4 18:10:48 2008
From cwen@iupui.edu Fri Jan  11 XX:23:48 2008
Try / Except

- With the try / except structure you can “catch” the error and add your own recovery code

```python
try:
    hour = int(tsplit[0])
    print "Hour", hour
except:
    print "Error in hour value"
```

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
From louis@media.berkeley.edu Fri Jan 4 18:10:48 2008
From cwen@iupui.edu Fri Jan 11 XX:23:48 2008
Recovery Approaches

• What is a good recovery approach?

• When reading and parsing data - typically print an error message and ignore the line

• Or make a “marker” value for the bad data

```python
for line in infile:
    # parse the line
    try:
        hour = int(tsplit[0])
        print "Hour", hour
    except:
        print "Hour not found"
        continue

    # process the data
```
Computing the Bucket

- Unless you are doing a Histogram bar for each value in your input set, you need to transform the input numbers into some number of “buckets” for display

- Use truncation of float to integer conversion

- In this example numbers go from 0 - 250 and we want 5 buckets

```
print nxt
pos = nxt / 250.0
x = pos * 5
buc = int (x)
if buc > 4:
    buc = 4

nxt  pos   x   buc
0    0.0   0.0  0
90   0.36  1.8  1
110  0.44  2.2  2
190  0.76  3.8  3
240  0.96  4.8  4
250  1.00  5.0  4
```
Natural Buckets

• If you are doing a histogram of integer values and can use a bucket for each distinct input value - no bucket computation is needed

• You might want to make sure that your bucket value is in range (i.e. $0 \leq buc \leq 23$) and deal with values outside of the range
Filling the Histogram Buckets

- A list is a natural Python data structure to represent the number of input values which fall into in each bucket

- You start with a list of zeros and increment the bucket each time an input value “falls into” the bucket

```python
# Set up a 5 element list of zeros
totals = [0] * 5

for line in infile:
    # Read and parse data
    # Compute bucket
    totals[buc] = totals[buc] + 1

print totals
```
Making the Graph
Screen Layout

- A sketch can be really helpful - it is hard to compute all of the layout values in your head
Python to Make a Window

- You can create a window and set the title, width, and height (in pixels)
- Also you can use `setCoords()` to give a nice simple virtual coordinate system from 0.0 to 1.0

```python
win = GraphWin("Distribution of Commits "+fname, 600,400)
win.setCoords(0,0,1,1)
```
Margins

- You need margins to make the graph look good and to give you space for titles and labels.
- Remember that the margins take away from the width of the body of the graph (0.8).
Axis Labels

- Remember that Text is centered around a point vertically and horizontally.
- When placing things horizontally you must know the width of each bar.

# Label the X-Axis - we have 24 hours (0-23)
# so we need to know each slot's width
width = 0.8 * (1.0 / 24.0)
Axis Labels

- To place the label for “4” compute the left edge of bar 4 and then add width/2 to get to the center of the bar

- Make sure to add the margin

\[
\text{left} = (4 \times \text{width}) + 0.1 \\
\text{center} = \text{left} + (\text{width} / 2) \\
\text{txt} = \text{Text(}\text{Point(}\text{center, 0.066)}, \text{“4”})
\]
Axis Labels

- You will want to write a loop to place the axis labels

- Text objects need to be strings (not integers) so you need to use the `str()` function to convert

```python
# Loop through 0..23
for label in range(24):
    left = (label * width) + 0.1
    center = left + (width / 2)
    txt = Text(Point(center, 0.066), str(label))
    txt.draw(win)
```
Scaling the Y-Axis

- It is nice to auto-scale the Y-Axis so the graph looks nice for various ranges of data.
- Simple Approach: Find the largest bucket and round up to the nearest multiple of 10 and then have 10 ticks
- Max = 6 -> 10 Max = 198 -> 200
Scaling the Y-Axis

- We can use the max() function to find the largest bucket in a list
  
  ```
  bmax = max(totals)
  print "Maximum value", bmax
  
  ymax = ( int(bmax / 10) + 1 ) * 10
  print "Y-Axis Maximum", ymax
  ```

- And then we cleverly use the float to integer conversion’s truncation feature
Drawing Rectangles

- Remember that the Rectangle object is defined by its two “corner” points.

- So you need to compute the left and right side of the rectangle as well as the height.

- Don’t forget that there is a margin (0.1) and that the graph body is not the entire heights of the page (0.8).
Drawing Rectangles

- The computation for a particular rectangle needs to compute the left edge, right edge, height, and bottom of the rectangle.

- Remember that the graph body is only 0.8 because of the 0.1 margins.

\[
\text{left} = (\text{width} \times 4) + 0.1 \\
\text{right} = \text{left} + \text{width} \\
\text{height} = (\text{float(value)} / \text{ymax}) \times 0.8
\]
One Last Thing...

- Make sure to wait for a mouse click after your drawing is complete so it does not vanish from the screen.

```python
win.getMouse()
```
Summary
# Prompt for File name and Open
# Create a list of zeros with one entry for each bucket
# Read and parse data, checking for errors
  # For each input value
  # Compute bucket position
  # Add one to the right bucket

# Figure out the maximum bucket
# Figure out the Y-Axis maximum

# Create Window and set coordinates
# Draw and label X and Y Axis
# Draw the histogram rectangles

# Wait for a mouse click
Concepts

- Multi-step parsing of input data and checking for errors
- Using a Python List to track bucket counts
- Using numeric expressions and float/integer conversion appropriately
- Layout of a graph including margins and graph body
- Computations necessary to label X and Y axis
- Computations necessary to place histogram rectangles