CSCI 02
INTRO TO PROGRAMMING
WITH PYTHON

LECTURE 8
WARNING: ENGINEERING CONTENT
MICHAEL GROSSBERG
import pylab
import math

MAX_STEPS = 20
for index in range(0,MAX_STEPS):
    angle = 2.0*math.pi*index/MAX_STEPS
    pylab.plot([angle],[math.sin(angle)],'r.')

# Little space at the top and bottom
pylab.ylim([-1.25,1.25])
pylab.show()
LIST ACCESS STARTS AT ZERO

-1 LAST ELEMENT

PAST LAST => ERROR

In [14]: x = [6,7,8,9] # setup list
In [15]: x[0] # first element
Out[15]: 6
In [16]: x[1] # second element
Out[16]: 7
In [17]: len(x)
Out[17]: 4
In [18]: x[-1] # last element
Out[18]: 9
In [19]: x[3] # also last because len=4
Out[19]: 9
In [20]: x[4] # index start from 0
IndexError: list index out of range
In [27]: x = [6,7,8,9] + range(0,5)
In [28]: print x
-------> print(x)
[6, 7, 8, 9, 0, 1, 2, 3, 4]
In [29]: x = [6,7,8,9];
In [30]: x.append(10)
In [31]: print(x)
[6, 7, 8, 9, 10]

In [32]: x = [6,7,8,9] + 10
TypeError: can only concatenate list (not "int") to list
In [33]: x = [6,7,8,9].append(10)
In [34]: print(x)
None

BUILD UP WITH APPEND
import pylab
import math

MAX_STEPS = 20
angles = []
for index in range(0,MAX_STEPS):
    angles.append(2.0*math.pi*index/MAX_STEPS)

ys = []
for angle in angles:
    ys.append(math.sin(angle))

pylab.plot(angles,ys,'b-')

# Little space at the top and bottom
pylab.ylim([-1.25,1.25])
pylab.show()
LIST COMPREHENSIONS

```
angles = []
for index in range(0,MAX_STEPS):
    angles.append(2.0*math.pi*index/MAX_STEPS)
```

BECOMES

```
angles = [2.0*math.pi*index/MAX_STEPS
       for index in range(0,MAX_STEPS)]
```

USE WISELY

YOUNG JEDIS
THE EPITROCHOID

WANKEL ROTARY ENGINE

\[ x = (a+b)\cos(\theta) - h\cos((a+b/b) \theta) \]
\[ y = (a+b)\sin(\theta) - h\sin((a+b/b) \theta) \]
THE EPITROCHOID

import pylab
import math

LOOPS, MAX_STEPS = 10, 100

thetas = [2.0*math.pi*index/MAX_STEPS
            for index in range(0,LOOPS*MAX_STEPS)]

a, b, h = 2.0, 1.0, 0.5

xs = [(a+b)*math.cos(theta) -
       h*math.cos(((a+b)/b)*theta)
       for theta in thetas]

ys = [(a+b)*math.sin(theta) -
       h*math.sin(((a+b)/b)*theta)
       for theta in thetas]

pylab.plot(xs,ys,'r-')
pylab.show()
EPITROCHOIDS GONE WILD

\[ a=3, b=1, h=.5 \]
\[ a=3, b=1, h=8 \]
\[ a=1, b=1, h=1 \]
3.7.1 Application to a Simple RC Circuit

The solution giving the voltage across the capacitor in Figure 3.2 following the closing of the switch can be written in the following form:

\[ V_c(t) = V_c(0) \exp \left( -\frac{t}{RC} \right) + V_s \left[ 1 - \exp \left( -\frac{t}{RC} \right) \right] \]  

(3.15)

\( V_c(t) \) is called the time response of the RC circuit, or the circuit output resulting from the constant input \( V_s \). The time constant \( RC \) of the circuit has the units of seconds and, as you will observe in the present analysis and other problems.

**FIGURE 3.2**
The circuit used in charging a capacitor.
WHAT HAPPENS TO THE CIRCUIT

```python
import pylab
import math

SAMPLES = 100.
Vc0, Vs, RC = 3., 10., 1.

times = [t/10.0 for t in range(SAMPLES)]

Vcs = [(Vc0*math.exp(-t/RC) +
       Vs*(1.-math.exp(-t/RC)))
       for t in times]

pylab.plot(times, Vcs, 'r')
pylab.show()
```
ALGORITHM

☐ Step by step description

☐ Nothing vague

☐ Good example: "Multiply previous answer by 10"

☐ Bad examples: "Get the answer", "Pick an easy number"
Paper Glider Directions

1. Fold down upper two corners.

2. Fold Paper in half-length wise.

3. Take outer two corners and fold like this:

4. Your glider should look like this.
SWAP X AND Y

- □ X = 5
- □ Y = 2
- □ SWAP X AND Y
  - □ Y = X FAIL!
  - □ LOST VALUE OF Y
- □ SWAP X AND Y (2)
  - □ Z = Y
  - □ Y = X
  - □ X = Z
  - □ DONE
    - □ Y == 2
    - □ X == 5
Babylonian method

Perhaps the first algorithm used for approximating $\sqrt{S}$ is known as the "Babylonian method", named after the Babylonians,[1] or "Heron's method", named after the first-century Greek mathematician Hero of Alexandria who gave the first explicit description of the method.[2] It can be derived from (but predates) Newton's method. This is a quadratically convergent algorithm, which means that the number of correct digits of the approximation roughly doubles with each iteration. It proceeds as follows:

1. Begin with an arbitrary positive starting value $x_0$ (the closer to the root, the better).
2. Let $x_{n+1}$ be the average of $x_n$ and $S / x_n$ (using the arithmetic mean to approximate the geometric mean).
3. Repeat step 2 until the desired accuracy is achieved.

It can also be represented as:

$$x_0 \approx \sqrt{S},$$
$$x_{n+1} = \frac{1}{2} \left( x_n + \frac{S}{x_n} \right),$$
$$\sqrt{S} = \lim_{n \to \infty} x_n.$$

This algorithm works equally well in the $p$-adic numbers, but cannot be used to identify real square roots with $p$-adic square roots; it is easy, for example, to construct a sequence of rational numbers by this method that converges to $+3$ in the reals, but to $-3$ in the 2-adics.
STEPS

- GET FLOAT S FROM USER
- MAKE A GUESS OF SQUARE ROOT
- TO MAKE NEXT GUESS
  - DIVIDE S BY OLD GUESS AND
  - AVERAGE WITH OLD GUESS
- STOP WHEN NEW AND OLD GUESS REALLY CLOSE
  - THIS MEANS NO MORE PROGRESS
import math

S = float(input('Input a number for to take the square root: '))

threshold = 10.0**(-15)

old_guess = S
guess = S/2

while abs(guess-old_guess) > threshold:
    old_guess = guess
    guess = 0.5 *( old_guess + (S/old_guess))

print "Babylonian method square root estimate of", S,"is",
print guess
print "math.sqrt(" , S ,") = ",
print math.sqrt(S)
DECIMAL EXPANSION

$1345 = 1 \times 10^3 + 3 \times 10^2 + 4 \times 10^1 + 5 \times 10^0$

BINARY EXPANSION ... REPLACE WITH 10 WITH 2
BINARY EXPANSION

DEC 60 = 1*2^5 + 1*2^4 + 1*2^3 + 1*2^2 + 0*2^1 + 0*2^0

= BIN 111100

<table>
<thead>
<tr>
<th>128</th>
<th>64</th>
<th>32</th>
<th>16</th>
<th>8</th>
<th>4</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Wednesday, September 29, 2010
BINARY IMPLEMENTATION

```python
user_number = int(input('Enter a number (pos. int): '))

number = user_number
bin_expansion = []

while number > 1:
    r = number % 2
    bin_expansion.append(r)
    number = number / 2
else:
    r = number % 2
    bin_expansion.append(r)

bin_string = ''
for bit in bin_expansion:
    bin_string = str(bit)+bin_string

print "dec: ", user_number, "bin: ", bin_string
```

Enter a number (pos. int): 60
dec: 60 bin: 111100
WHAT ABOUT OCTAL? BASE 8?

- IMPORTANT BASES FOR CS
  - BINARY (BASE 2) BIT
  - OCTAL (BASE 8)
  - HEXADECIMAL (BASE 16) 1/BYTE
    - CAN REPRESENT BYTE 0-255:
    - THATS 2 CHARs IN HEX
BUILT IN FUNCTIONS FOR THIS

In [115]: bin(60)
Out[115]: '0b111100'

In [116]: oct(60)
Out[116]: '074'

In [117]: int(074)
Out[117]: 60

In [118]: hex(60)
Out[118]: '0x3c'
POSSIBLE JOKE

why don't programmers know halloween from christmas?

dec 25 = oct 31