Arts: New band "The Recursions" performs songs from first album on first album.

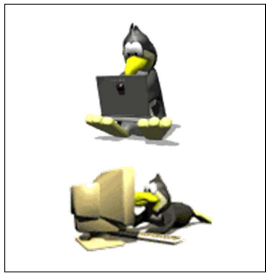
Sports: HMC Web surfing team advances to Nationals

The CS 5 Times

"Zach and Geoff to be Replaced by Two Penguins," says HMC Dean

Read Chapter 3 this week!

Claremont(AP): Beginning next week, CS 5 will be taught by a pair of penguins, announced an HMC dean. "Penguins are very smart," said the dean, "and they are also quite adorable, which is more than we can say about the CS 5 faculty. Plus, we won't need to heat their offices in the winter."



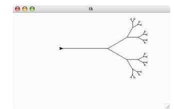
Photos of the two new CS 5 penguin professors

This Week

Homework 2:

- Reading on computer interfaces
- Lab: Fractal art

You're ready for this today!



You're ready for this today!

- Problem 2: Higher-order functions
- Problem 3: 42andme
- Problem 4: RNA folding

Thursday!

Speaking of Illness…

Please e-mail the prof as soon as you feel bad!

Comparing DNA via Longest Common Subsequence (LCS)

AGGACAT
ATTACGAT

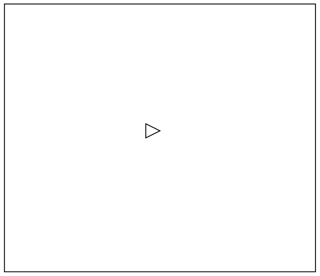
```
>>> LCS("AGGACAT", "ATTACGAT")
5
>>> LCS("can", "man!")
2
```

Recursive Approach…

```
def LCS(s1, s2):
    if BASE CASE:
        ???
    else:
```

LCS("spam", "sam!")

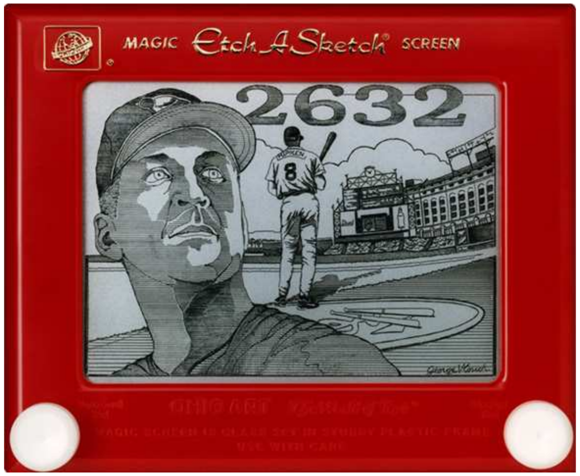
Turtle Graphics



forward(100)

right(90)

Etch-a-Sketch craziness…



No way this is real…

except that it is !

Turtle Graphics

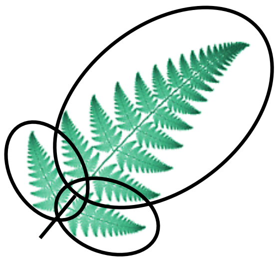
Turtle graphics are built into Python!

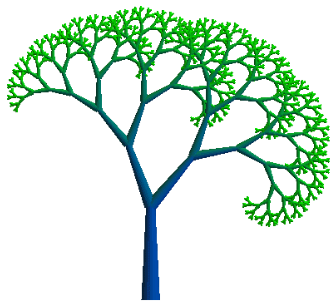
```
>>> import turtle
>>> turtle.forward(50)
>>> turtle.right(90)
>>> turte.backward(50)
```

degrees()	radians()	reset()
clear()	tracer(flag)	forward(distance)
backward(distance)	left(angle)	right(angle)
up()	down()	width(width)
color(*args)	fill(flag)	heading()
setheading(angle)	window_width()	window_height()
position()	setx(xpos)	sety(ypos)
goto(x,y)		

Problem 2 has a link to the turtle documentation

Fractals

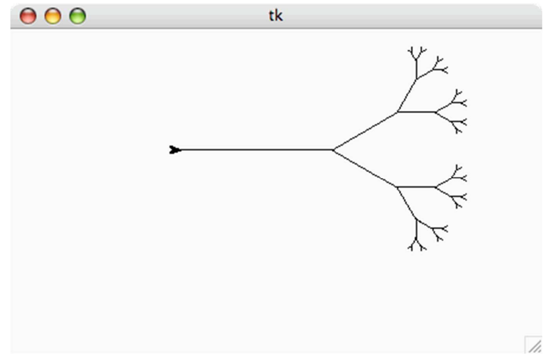




“I Wonder About Trees” – Robert Frost

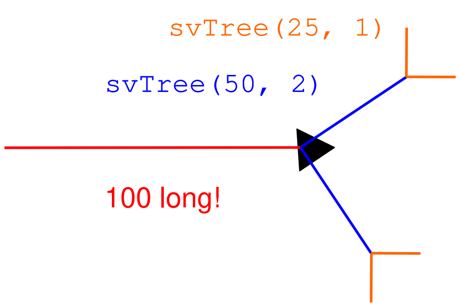
“We wonder about Robert Frost” - Trees

```
>>> svTree(128, 6)
```



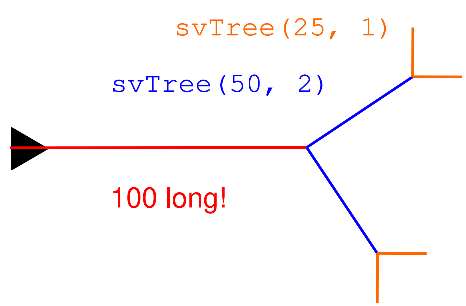
trunk length   recursion level

```
>>> svTree(100, 3)
```



trunk length   recursion level

```
>>> svTree(100, 3)
```



Tuples ("Immutable Lists")

```
>>> foo = (42, 'hello', (5, 'spam'), 'penguin')
>>> foo
(42, 'hello', (5, 'spam'), 'penguin')
>>> foo[0]
42
>>> foo[-1]
'penguin'
>>> foo[0:2]
(42, 'hello')
>>> foo[0:1]
(42,)
```

Tuples ("Immutable Lists")

```
>>> foo = (42, 'hello', (5, 'spam'), 'penguin')
>>> foo
(42, 'hello', (5, 'spam'), 'penguin')
>>> foo[0]
42
>>> foo[-1]
'penguin'
>>> foo[0:2]
(42, 'hello')
>>> foo[0:1]
(42,)
>>> foo[0] = 100
BARF!!!
```

Dictionaries

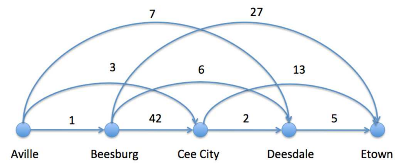
```
>>> D = {}
>>> D["Geoff"]= "spam"
>>> D["Zach"]= "donuts"
>>> D["Alien"]= 42
>>> D["Geoff"]
'spam'
>>> D["Alien"]
42
>>> D["Suicide Squad"]
BARF!
```

"Geoff", "Zach", and "Alien" are called the "keys" in the dictionary. Any ***immutable*** object can be a key.

Dictionaries

```
>>> D = {}
>>> D["Geoff"]= "spam"
>>> D["Zach"]= "donuts"
>>> D["Alien"]= 42
>>> D["Geoff"]
'spam'
>>> D["Alien"]
42
>>> D["Suicide Squad"]
BARF!
>>> D
{'Geoff': 'spam', 'Zach': 'donuts', 'Alien': 42}
```

Giigle maps

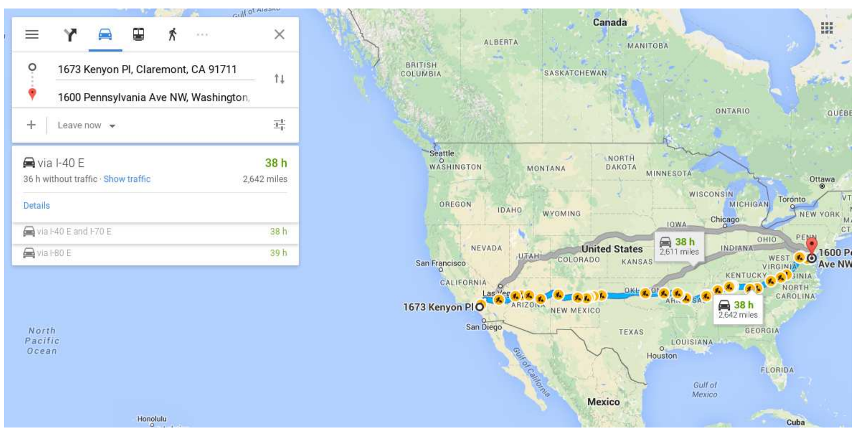



```
Inf = float("inf")

FiveCities = ["A", "B", "C", "D", "E"]

FiveDists  = {("A","A"):0, ("A","B"):1, ("A","C"):3, ("A","D"):7 , ("A","E"):Inf,
              ("B","A"):Inf, ("B","B"):0, ("B","C"):42, ("B","D"):6, ("B","E"):27,
              ("C","A"):Inf, ("C","B"):Inf, ("C","C"):0, ("C","D"):2, ("C","E"):13,
              ("D","A"):Inf, ("D","B"):Inf, ("D","C"):Inf, ("D","D"):0, ("D","E"):5,
              ("E","A"):Inf, ("E","B"):Inf, ("E","C"):Inf, ("E","D"):Inf, ("E","E"):0
             }
```

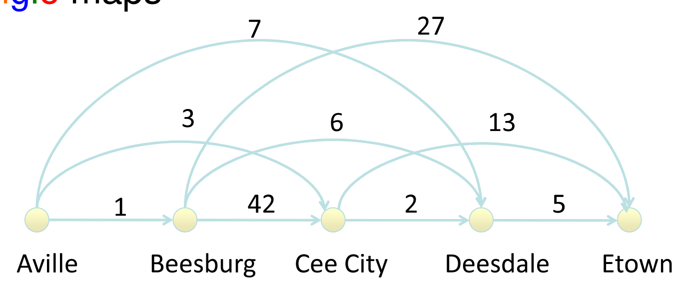
```
>>> FiveDists[("B", "C")]
42
```

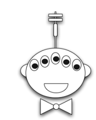
Sometimes We Need to Make More Than 2 Recursive Calls!

Finding Shortest Paths



Giigle maps



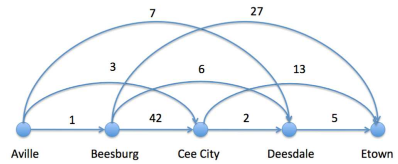


All roads point east!

Shortest path?

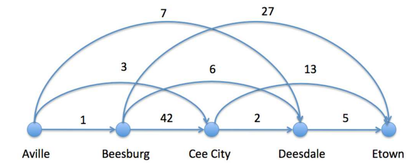
Is greed good?

How does the use-it-or-lose-it idea get used here?



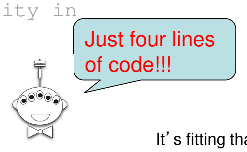

```python
Inf = float("inf")

FiveCities = ["A", "B", "C", "D", "E"]

FiveDists  = {("A","A"):0, ("A","B"):1, ("A","C"):3, ("A","D"):7 , ("A","E"):Inf,
              ("B","A"):Inf, ("B","B"):0, ("B","C"):42, ("B","D"):6, ("B","E"):27,
              ("C","A"):Inf, ("C","B"):Inf, ("C","C"):0, ("C","D"):2, ("C","E"):13,
              ("D","A"):Inf, ("D","B"):Inf, ("D","C"):Inf, ("D","D"):0, ("D","E"):5,
              ("E","A"):Inf, ("E","B"):Inf, ("E","C"):Inf, ("E","D"):Inf, ("E","E"):0
             }
```

```
>>> shortestPath (FiveCities, FiveDists)
10

>>> shortestPath (["C", "D", "E"], FiveDists)
7

>>> shortestPath (["E"], FiveDists)
0
```




```python
Inf = float("inf")

FiveCities = ["A", "B", "C", "D", "E"]

FiveDists  = {("A","A"):0, ("A","B"):1, ("A","C"):3, ("A","D"):7 , ("A","E"):Inf,
              ("B","A"):Inf, ("B","B"):0, ("B","C"):42, ("B","D"):6, ("B","E"):27,
              ("C","A"):Inf, ("C","B"):Inf, ("C","C"):0, ("C","D"):2, ("C","E"):13,
              ("D","A"):Inf, ("D","B"):Inf, ("D","C"):Inf, ("D","D"):0, ("D","E"):5,
              ("E","A"):Inf, ("E","B"):Inf, ("E","C"):Inf, ("E","D"):Inf, ("E","E"):0
             }
```

```python
def shortestPath (cities, dists):
   '''Returns the length of the shortest path
      from the leftmost to the rightmost city in
      in the cities list.'''
   if BLAH:
      return BLAH BLAH
   else:
      return BLAH BLAH BLAH
```



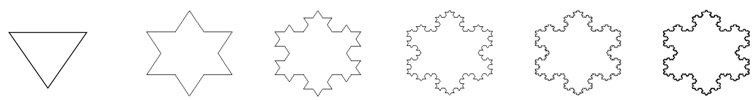
It's fitting that `map` gets used here!

We Admit It's Tricky

```python
def shortestPath (cities, dists):
  '''Returns the length of the shortest path
     from the leftmost to the rightmost city in
     in the cities list.''

  if len(Cities) <= 1:
    return 0
  else:
    return min(map(
      lambda hop: dists[(cities[0], cities[hop])]
        + shortestPath(cities[hop:], dists),
      range(1, len(cities))))
```

SnowFlake Fractals

The Koch Snowflake Fractal:



level 0 level 1 level 2 level 3 level 4 level 5

Snowflake Fractals

The Koch Snowflake Fractal: