16th October 2009 Lecture #15 CSCI 1105

```
def estimate_pi(n)
      def count_darts(n)
            if n=0:
                  Return 0
            else:
                  x = random()
                  y = random()
                  if x**2 + y ** 2 <= 1:
                        return 1 + count_darts(n-1)
                  else
                  return count_darts(n-1)
      return count_darts(n)*4.0/n
```

The problem here is that if n is a really higher number, our program crashes while trying to compute.

What can be done about this?
If we can arrange it so that the program doesn't have anything to do after the reccursive Call,
then it shouldn't need to remember to do anything afterwards, and MAYBE then it won't run out of memory.

This would require us to get rid of the "1+" in count_darts.

How can we do this?

Use an extra parameter to track how many hits have been seen so far.

```
def estimate_pi(n):
      def count_darts(n,a): #Says: tell me how many total darts,with n
left to throw, arein the circle.
                        # "a" darts have hit the circle so far.
            if n==0:
                  return a
            else:
                  x=random()
                  y=random()
                  if x**2 + y**2 <= 1:
                        return count_darts(n-1,a+1)
                  else:
                        return count_darts(n-1,a)
      return count_darts(n,0) * 4.0/n
```

Recursive call is the last thing count_darts does - This is called the Tail Recursion

Now, count_darts has no computation to perform once the recursive call returns --> Hence, no extra memory is needed
and the program should be able to recurse to the arbitrary depth.

Let's try the program:

```
>>> estimate_pi(1)
0.0
>>> estimate_pi(2)
4.0
>>> estimate_pi(10)
3.2000000000000002
>>> estimate_pi(100)
3.2400000000000002
```

```
>>> sys.setrecursionlimit(1000000)
>>> estimate_pi(1000)
3.2400000000000002
>>> estimate_pi(2000)
3.246
>>> estimate_pi(3000)
3.1360000000000001
>>> estimate_pi(4000)
FAIL :( Program Crashes
```

But still the program crashes. Why?
The Python interpreter is not "smart" enough to recognize that the extra memory is not needed
and it continues to waste memory on tail recursive calls.
This is a short coming of the programming environment, not of the technique. Our technique would work
in many other languages.

Higher-Order Functions

Excercise: double every element of a list.

```
def double_list(L):
      if L == Empty:
            return Empty
      else:
            return List(L.hd*2,double_list(L.tl))
```

Excercise: Add 1 to every element of a list.

```
def add1_list(L):
      if L == Empty:
            return Empty
      else:
            return List(L.hd+1,add1_list(L+1))
```

Excercise: Square every element of a list

```
def square_list(L):
      if L == Empty:
            return Empty
      else:
            return List(L.hd**2,square_list(L.tl))
```

These functions are identical in form, and differ only in detail.

They all look like the following:

```
def fn(L):
      if L == Empty:
            return Empty
      else:
            return List(L.hd,fn(L.tl)
```

It would be nice if we could capture the common behaviour, name it and reuse it.
Luckily, Python provides a way, via higher-order functions - functions that take other functions as parameters or
return other functions as results.

Higher Order Functions

So if we can capture the difference among these programs as functions and then pass them
to a more generic recursive functions, we may be able to achieve our objective:

```
def map(f,L):
      if L == Empty:
            return Empty
      else:
            return List(f(L.hd),map(f,L.tl))
```