Recursion is fantastic…

And is often “handy”…

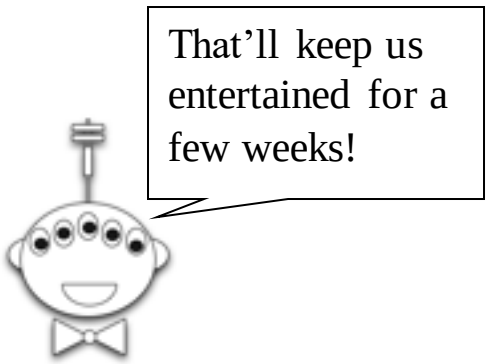
Imperative Programming

Imperative programming is the oldest programming paradigm. A program based on this paradigm is made up of a **clearly-defined sequence of instructions** to a computer.

Therefore, the source code for imperative languages is a series of commands, which specify what the computer has to do – and when – in order to achieve a desired result. Values used in variables are changed at program runtime. To control the commands, **control structures such as loops or branches are integrated into the code**.

What's Up Next…

- Loop structures: **`for`** and **`while`**
- Writing some "bigger" programs
  - Secret Sharing (cryptography)
  - Games (Nim, Mastermind)
  - Data compression



Loops!

for

```
for <variable> in <iterable>:
    Do stuff!
```

```
for symbol in “blahblahblah”:
    print(symbol)
```

```
for element in [1, 2, 3, 4]: …
```

```
for index in range(42): …
```

Three uses of for!

I’d like to see four uses of three!

while

```
while <condition>:
    Do stuff!
```

```
i = 0
while i < 100:
    print(i)
    i += 1
```

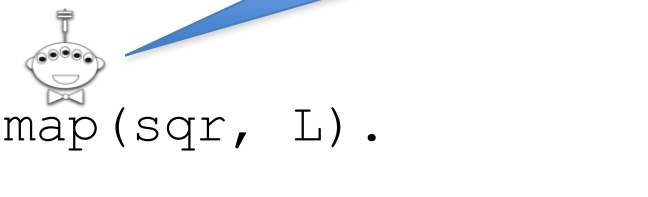
```
sum = 0
i = 0
while i < 10:
    sum = sum + i
    i += 1
print(sum)
```

Write equivalent for-loops.

Draw flow charts.

Using for

```
def mapSqr(L):
    '''
    Assume L is a list.  Return map(sqr, L).
    '''
```



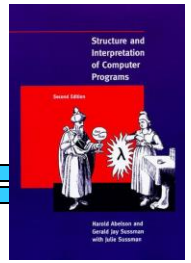
Exercises

Implement factorial, using a for-loop.

Use a loop to implement fib, where

fib(0) = 0, fib(1) = 1, fib(n) = fib(n-1)+fib(n-2)

Good Design



Programs must be written for people to read, and only incidentally for machines to execute. - Abelson and Sussman

1. Design your program "on paper" first. Identify the separate logical parts and the input/output for each parts.
2. Once your design is established, write the function "signatures" (function name, inputs) and docstrings.
3. Fill in the code for a function, **test that function carefully, and proceed only when you are convinced that the function works correctly.**
4. Use descriptive function and variable names (how about **`x, stuff, florg, jimbob`** ?).
5. Don't replicate functionality.
6. Keep your code readable and use comments to help! `# Here's one now!`
7. Avoid global variables unless absolutely necessary! Instead, pass each function just what it needs.
8. Use recursion and functional constructs (e.g. map, reduce, filter, lambda) where appropriate.

2-D “Arrays”

```
>>> A = [ [0, 0, 0, 1], [1, 1, 0, 0], [0, 0, 0, 1] ]
>>> A = [ [0, 0, 0, 1],
          [1, 1, 0, 0],
          [0, 0, 0, 1] ]
>>> A[0][3]
???
```

Shallow Copy

```
>>> A = [1, 2, 3, 4]
>>> B = A
>>> B[0] = 42
>>> A[0]
???

def f():
    L = [1, 2, 3, 4]
    g(L)
    return L

def g(List):
    List[0] = 42
```

Deep Copy

```
def f():
    L = [1, 2, 3, 4]
    M = g(L)
    print(L)
    print(M)

def g(List):
    return map(lambda X: X+1, List)
```

Exercise

```
def f(L):
    '''Assume L is a list of at least 3 floats.
    Return a copy of L, changed as follows.
    Each element is the average of itself and the
    two adjacent elements.  But the first and last
    are unchanged.'''
```