Basic template for recursion

- Recursion is about
  - Solving a **complicated** problem
  - by solving a **simpler version** of the problem

- But you have to stop sometime
  - **Stop** when you get to "**easy**" problem

You need to write code for

- **Recognizing** easy cases
- **Solving** easy (base) cases
- **Simplifying** the problem (get one step closer to the base/easy cases)
- **Fixing** the simplified solution into a solution to the full problem

```
(define function
  (lambda (args)
    (if easy-case?
        solve-easy-case
        (fix-solution
          (function simplified)))))
```

Iterative Recursion

```
(define (help args … accum)
   (if easy-case?
       accum
       (help args …
             (updater accum …))))
```

```
(define (func args …)
   (help args … start-accum))
```

- Instead of waiting until the end to "combine" all the simpler answers
- We could instead keep track of the "answer so far" in an *accumulator*
- This is called **iterative recursion**
- Uses a helper function to maintain the correct number of args.

Tree Recursion

```
(define (func args …)
   (if easy-case?
       solve-easy-case
       (combine (func one-part)
                (func other-part))))
```

- You can often simplify the problem by **splitting** it
- Then the fixup step consists of **combining** the answers to the two problems
- This is called **tree recursion**