

## 6.189 IAP 2011: Recursion Notes

**Recursion:** (definition) noun. See recursion.

**Recursion: Formal Definition:** An algorithmic technique where a function, in order to accomplish a task, calls itself with some part of the task.

- Recursive solutions involve two major parts:
  1. Base case(s), in which the problem is simple enough to be solved directly.
  2. Recursive case(s). A recursive case has three components:
    - (a) Divide the problem into one or more simpler or smaller parts of the problems,
    - (b) Invoke the function (recursively) on each part, and
    - (c) Combine the solutions of the parts into a solution for the problem.
- Depending on the problem, any of these may be trivial or complex.

### Example: Sum

A non-recursive example:

```
def it_sum(a_list):
    result = 0
    for x in a_list:
        result += x
    return result
```

We say that the above function *iterates* over the values in the variable `a_list`, and returns their sum.

Recursion is similar to iteration, such that the operation being performed is defined (partly) in terms of itself. Such an operation is said to be *recursive*.

Here is a recursive definition of the `sum()` function:

```
def rec_sum(a_list):
    if a_list == []:
        return 0
    else:
        return a_list[0] + rec_sum(a_list[1:])
```

`rec_sum` computes the same exact thing as `it_sum`, but in a different way. The first thing to note is that it does not use a for-loop. The second thing to note is that the `rec_sum` function *calls itself*. That is to say, `rec_sum()` is defined in terms of itself; it is recursive.

How does it work? Let's go through the parts of recursion mentioned at the introduction to this handout.

1. Base Case: What is the *base case* of `rec_sum`?

2. Recursive case:

- (a) How do we *divide the problem*?
- (b) Where do we *invoke the function recursively*?
- (c) Finally, where do we *combine the solutions*?

Now, let's pretend to be a Python interpreter and execute the recursive calls ourselves.

```
rec_sum([1, 2, 3])
= 1 + rec_sum([2, 3])
= 1 + (2 + rec_sum([3]))
= 1 + (2 + (3 + rec_sum([])))
= 1 + (2 + (3 + 0))
= 1 + (2 + 3)
= 1 + 5
= 6
```

Note that our *base case* is when the list is empty. That is the recursive call to `rec_sum([])`, which evaluates to 0. A base case is very important - it is the stopping point for recursion.

The *recursive case* is demonstrated by calls to `rec_sum` where the argument is a non-empty list. During a recursive case, we make incremental progress towards solving the problem, and also make a recursive call to the function with a smaller input space.

MIT OpenCourseWare  
<http://ocw.mit.edu>

6.189 A Gentle Introduction to Programming  
January IAP 2011

For information about citing these materials or our Terms of Use, visit: <http://ocw.mit.edu/terms>.