CIS 1100

Recursion Cont.

Python
Fall 2024
University of Pennsylvania

Reach Out!

You are invited to reach out to Me (tqmcgaha@seas) or Harry (sharry@seas) for any of the following reasons:

- A request for a no-questions-asked extension on Emoji Blender till Friday @ 11:59pm
  - unfortunately can't offer extra office hours or later submission than that. Same HW schedule for next HW
  - **NO SUBMISSION AFTER FRIDAY**
  - **NO SUBMISSION AFTER FRIDAY**
  - **NO SUBMISSION AFTER FRIDAY**
- Setting up a time to talk privately
- Just to say "I'm overwhelmed"
  - We'll follow up and try to figure out what to do next

Today

- Lecture for about half the time, Office Hours for the rest
- There is still a worksheet, but we will not collect it. Everyone will get the 1/3 of a late token
- No new check-in, if you did not finish the check-in due today, it was extended to be open till beginning of class Friday.

Recursive Thinking

The journey of a thousand miles starts with one mile.

And then a journey of 999 miles.

Recursive Thinking

A function is recursive if it invokes itself to do part of its work.

Recursion is a problem-solving approach that can be used to generate simple solutions to certain kinds of problems that are difficult to solve by other means.

Recursion reduces a problem into one or more simpler versions of itself.

Recursion

An alternate to using loops for solving problems

The core of recursion is taking a big task and breaking it up into a series of related small tasks.

- Example: handing out papers for an exam
  - Iterative: have a TA walk down a row of students, giving each person an exam
  - Recursive: A student takes one exam, pass the rest down the aisle
- Example: Which row are you in?

Anatomy of a Recursive Function

Every recursive function needs at least one **base case** and at least one **recursive part**.

The **base case:**

- handles a simple input that can be solved without resorting to a recursive call. Can also be thought of as the case where we "end“ our recursion.

The **recursive part:**

- contains one or more recursive calls to the function.
- In every recursive call, the parameters must be in some sense "closer" to the base case than those of the original call

Practice (L11)

In mathematics, the Fibonacci sequence is a sequence in which each number is the sum of the two preceding ones. Numbers that are part of the Fibonacci sequence are known as Fibonacci numbers.

The sequence starts with 0 and 1:

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...]
```

`fib(0)` is `0`, `fib(1)` is `1`.

(L11)

We want to write a recursive function to calculate the Nth fibonacci number.

What are the base case(s) and recursive(s)? (e.g. when do we recurse, when do we not).

Practice

In mathematics, the Fibonacci sequence is a sequence in which each number is the sum of the two preceding ones. Numbers that are part of the Fibonacci sequence are known as Fibonacci numbers.

The sequence starts with 0 and 1:

`[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...]`

`fib(0)` is 0, `fib(1)` is 1.

(C12)

Write the function:

```
def fib(N):
    # TODO
```

Range of a recursive function

An important thing to think about when desigining recrusive functions is thinking about:

- Where we start with the problem
- the little bit of work that is done on each step
- the end of the problem

What were each of these things for `def fib(N)`?

What about `def print_stars(N)`?

Practice:

Consider we want to write the function `remove_vowels(word)` that takes in a string and returns the same string without any vowels in it.

You can assume you have access to the set vowels:

```
vowels = {'A', 'a', 'E', 'e', 'I', 'i', 'O', 'o', 'U', 'u', 'Y', 'y'}
```

So `remove_vowels("Hello")` returns `"Hll"`

Before writing any code (L13)

- What is the base case? (Why is it the empty string? `""`)
- What is the recursive case?
- What is the work done on each step?

Practice: (C14)

Finish writing this function:

```
def remove_vowels(word):
    # takes in a string and returns the same string without any vowels in it.
    vowels = {'A', 'a', 'E', 'e', 'I', 'i', 'O', 'o', 'U', 'u', 'Y', 'y'}
    # TODO: What do you put here?
```

Practice: (L15)

Consider we want to write the function `find_factors(N)` that returns a set containing all positive factors of the input integer `N`.

A number `x` is a factor of `N` if and only if `N % x == 0`

NOTE: This is a different problem than one you will see on the homework called `gcd`

Before writing any code (L15)

- What is the base case?
- What is the recursive case?
- What is the work done on each step?

Helper functions

Sometimes to do recrusion we need to remember a bit more information than is provdided to the overall problem.

In this case, what other information do we need for `find_factors(N)` to get a working recursive solution?

Why can't we just recursively call `find_factors(N-1)`?

Practice: (C16)

Finish writing this function:

```
def find_factors_helper(current, N):
    # TODO: What do you put here?

def find_factors(N):
    return find_factors_helper(0, N);
```