CS 120: Intro to Computer Programming II

In-Class Activity - 05 Recursion - Day 1

Activity 1 - Turn in this one

Turn the "how much money is in the cup" algorithm into a Python function. That is, write a function, `how_much_money(cup)`, where the parameter is an array; return the sum of the values in the array.

This algorithm **must** be recursive. That means:

- It must call itself to solve the problem (except in the base case).
- When recursing into itself, it must pass itself a **smaller** bit of data than it was given. (For an array, we often slice one element off of the array, and recurse into the rest.)
- It must have a **base case** - that is, a special case, only triggered when the data is very small or very simple, which solves the problem without recursing.
- Every path through the function must return something - including all base cases **and** all recursive cases.

WARNING:
Do not change the array you are given! (Thus, `cup.pop()` is forbidden.) This is a general rule with **any** function - not just recursion: if you are passed an array as a parameter, you **must not** modify the array unless your function spec specifically makes it clear that you should.

After all, your caller might have lots of interesting data in that array, which they don't want to lose!

Activity 2 - Turn in this one

Now, write the same function again - but this time, assume that the parameter is a linked list. Remember to handle empty lists! (And yes, recursion is still required!)

(activity continues on the next page)

Activity 3 - Optional

OPTIONAL. Complete this if you have time, and turn it in. If you don't have time, you may report to your TA that you ran out of time.

Consider the following function, which builds a linked list recursively:

```
def build_countdown(n):
    assert n >= 0      # sanity check the argument, negatives not allowed!
    if n == 0:
        return None

    head  = ListNode(n)
    later = build_countdown(n-1)
    head.next = later

    return head
```

Suppose that we call `build_countdown(4)`. Draw a reference diagram for **every call** of the function, from (4) all the way down to (0). Include all of the variables; it will be especially helpful for you to see what `head` and `later` point to!

Challenge Activity - Do not turn in this one

Write a **recursive** function to build an **array**, but containing the same "countdown" pattern from the previous Activity.

Challenge Activity - Do not turn in this one

Write a recursive function which, when given an array of values, will return a different array, which contains every other value from the input (starting with element `[1]`). That is, if you are given

```
["abc", "def", "ghi", "jkl", "mno"]
```

then you should return

```
["abc", "ghi", "mno"]
```

TEST OUT YOUR SOLUTION! In my experience, nearly every student solution, of a problem of this sort, is broken at first.