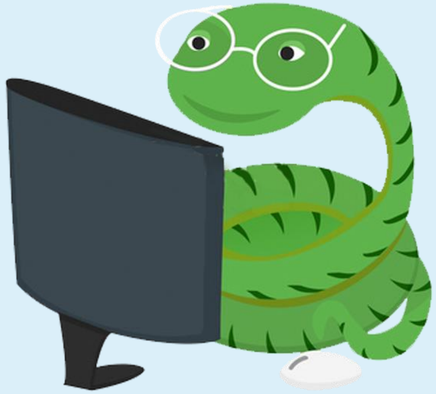
CS 116 TUTORIAL 5

ABSTRACT LIST FUNCTION AND ACCUMULATIVE RECURSION

REMINDER

- Assignment 05 due Wednesday, Feb. 26th at 10:00AM
- Midterm is on March 2nd at 7 PM
  - Midterm reference sheet will be posted on Piazza & course webpage.
  - Q&A session is on Feb. 29th at 2pm. Share the questions on Piazza!
- Final is on April 15th at 4pm



RECURSION

Types:

- Structural recursion
- Accumulative recursion
- Generative recursion

We've been using this so far.

New!!!

New!!!

STRUCTURAL V.S. ACCUMULATIVE

- **Structural Recursion:**
  - Break problems into smaller problems using the recursive definition of our data.
  - recursive subproblem is always one step closer to a base case
  - Uses a recursive template

- **Accumulative Recursion:**
  - Recursion with an accumulator(s)

But these two are not necessarily independent!

ACCUMULATIVE RECURSION

```
def acc_fn(remaining, acc):
    if (base_case of remaining):
        return … acc …
    else:
        …
        return acc_fn(updated_remaining, updated_acc)


def fn(lst):
    return acc_fn(initial_remaining, initial_acc)
```

Use acc to help return the correct result

- Accumulators "keep track" of something so that you can quickly return the expected result
- Sometimes, you may need more than one accumulator.

Generative Recursion: A Summary!

- It's recursion…
- Solving larger problem by solving subproblem(s) <u>inspired by the problem itself.</u>
- There's no "structural" format:
  - Recursing at 'different' places
  - Recursing 'multiple' times
  - Not always counting up/down by "one"
- Classic examples (shown in class):
  - Palindromes
  - Solving GCD's with Euclid's Algorithm

Example: Euclidean Algorithm on GCD

Let m and n be integers such that $\mathrm{m} \geq n$ and $m = qn + r$, where $0 \leq r < n$. Then the following is true:

$$\gcd(m, n) = \gcd(n, r)$$

Note:

- $r = m \bmod n$
- Check out the proof for Euclidean Algorithm on [Wikipedia]

Code	Analysis
`def gcd(m,n):`	
`if m == 0:` `return n`	Base Case #1
`elif n == 0:` `return m`	Base Case #2
`else:` `return gcd(n, m%n)`	Recursive call: • $n \leq m$ • 0 ≤ `m%n` < `n`

```
def gcd(m,n):
    if m == 0:
        return n
    elif n == 0:
        return m
    else:
        return gcd(n, m%n)
```

CQ 1: CONFIDENCE LEVEL

How confident are you with generative recursion?

A. Not at all. (i.e. What the fork is going on?)
B. I'm sort of confident but I would like more examples.
C. I'm sort of confident but I don't want to see more examples.
D. I'm very confident with generative recursion.

CQ 2:

Does this solution use accumulative recursion?

A. Yes
B. No
C. I don't know.

```
def recursion3(lst):
        if lst == []:
                return True
        elif lst[0] == lst[-1]:
                return recursion3(lst[1:-1])
        else:
                return False
```

CQ 3:

Does this solution use accumulative recursion?

A. Yes

B. No

C. I don't know

```
def recursion4(lst,boo):
     if lst == []:
          return boo
      else: L[0] == L[-1]:
          boo = (boo and (L[0] == L[-1]))
          return recursion4(lst[1:-1],boo)
```

QUESTION 1

- Write an accumulatively recursive function `record_digit(n)` that returns a list of integers of length `10`, with each index from 0 to 9 represents a corresponding digit's total appearances in the integer `n`. You cannot use L.count().
- For example:

```
record_digit(19990514)=>[1,2,0,0,1,1,0,0,0,3]
                          0 1 2 3 4 5 6 7 8 9
```

QUESTION 2

Write an accumulatively recursive function `count_max` that consumes a nonempty list of integers `alon` and returns the number of times the largest integer in `alon` appears.

Note: - `max` and `L.count()` cannot be used in this question.

- Your function can only pass through the list once

For example,

```
count_max([1, 3, 5, 4, 2, 3, 3, 3, 5]) => 2
```

since the largest element of the list, `5`, appears twice. Your function should pass through the list only once.

QUESTION 3: REDUCING NUMBERS

Write a function `smaller` that consumes a nonempty string `s`, containing only numeric characters, and generates a new string by repeatedly removing the larger of the first and last characters in `s`. If the first and the last number are the same, remove the last one.

For example, starting from "5284", compare "5" and "4", and recurse on "284", which will compare "2" and "4", and recurse on "28". Comparing "2" and "8", leads to recursing on "2", which is the answer (since it is a string of length `1`).

NOTE: Do not use `min`.

For example,

```
smaller("4325") => "2"
smaller("1") => "1"
smaller("2325") => "2"
smaller("8668") => "6"
```

QUESTION 4: SKIPPING VALUES

Given a list `L` of positive integers, the skip-value of a list is the number of steps to reach the end of the list, using the values in the list

- If `L` is empty, the skip value is `0`
- If `L` is nonempty:
  - Add 1 to the remaining skip value
  - Move ahead L[0] places in the list, and repeat the process with the remainder of the list from that place

Write a function `skip_value` to calculate the skip value of the list `L`.

For example,

```
skip_value([]) => 0
skip_value([1,1,1]) => 3
skip_value([2,100,1]) => 2
```

QUESTION 4: TRACING EXAMPLES

```
skip_value([1,1,1])
⇒ 1+skip_value([1,1])
⇒ 1+(1+skip_value([1]))
⇒ 1+(1+(1+skip_value([])))
⇒ 1+(1+(1+0))
⇒ 3

skip_value([2,100,3,1,1,1])
⇒ 1+skip_value([3,1,1,1])
⇒ 1+(1+skip_value([1]))
⇒ 1+(1+(1+skip_value([])))
⇒ 1+(1+(1+0))
⇒ 3
```

QUESTION 5

Develop an accumulatively recursive function `list_to_num` that consumes a nonempty list, `digits`, of integers between 0 and 9, and returns the number corresponding to `digits`.

For example,

- `list_to_num([9, 0, 8]) => 908`
- `list_to_num([8, 6]) => 86`
- `list_to_num([0, 6, 0]) => 60`