Tutorial 6:
Midterm Review
Design Recipe (Purpose/Effects)

Purpose:

- Make sure you mention all of the parameter names in your purpose and how they relate to what is being returned.
- Keep it short and simple; do not copy directly from the question!
- Make it clear if you are “returning”, “printing”, “mutating”, or “reading input”
- Use return, not produce.

Effects:

- Be clear and concise on the different effects: printing, input, and mutation
  - Print to the screen
  - Read input
  - Mutate list
Design Recipe
(Contracts/Requirements)

Contracts:
• Use the form
  • fun_name: types consumed \(\rightarrow\) type returned
• Use the single arrow in contracts!
• Make sure you use the correct type names (i.e. Str not String; it’s Float not Num in Python, etc.)
  • Nat vs Int
  • Lists are written as (listof ...), where ... is/are the types in the list
  • (anyof ... ...) to show multiple possible types for a parameter.

Requirements:
• Include requirements to any of the types consumed, if it has any.
• It’s mentioned in the question description.
Design Recipe (Examples/Tests)

For examples, make sure to have a base case and a non-base case at minimum.

Example Format

- Examples: `fn_call(x1, x2, ..., xn) => expected`
- Use double arrows in examples!

Tests: `check.expect` and `check.within`

- `check.set_input` (when `input(...)` is used)
- `check.set_screen` or `check.set_print_exact` (when information is printed)
- An extra `check.expect` to check mutation if the function mutates a list (parameter)
  - `check.expect("label_1", fcn_call(p1, ..., pn), expected returned value)`
  - `check.expect("label_1(mutation)", L, expected list after mutation)`
Out of the following topics, which one would be most helpful to cover now in preparation of the midterm?

A. **Conditions**
B. **Strings and/or Input/Output**
C. **Lists**
D. **Recursion (Accumulative)**
E. **Recursion (Generative)**
Let’s say we have large bricks that are 5 inches in length and small bricks that are 1 inch in length.

Write a function called `enough_bricks` which has three parameters: `small`, the number of small bricks, `large`, the number of large bricks and `goal`, the length of a row we want to build. `enough_bricks` returns `True` if you can create a row with same length as `goal` with the number of small and large bricks available, `False` otherwise.

Examples:

```
enough_bricks(3,1,8) => True
enough_bricks(3,1,9) => False
enough_bricks(1,2,9) => False
```

Source: Coding Bat, http://codingbat.com/prob/p118406
Write a function called `ends_with_other` that consumes two strings, `s` and `t`, and returns `True` if `s` ends with `t` or if `t` ends with `s`, `False` otherwise. This function should be case insensitive.

Examples:

- `ends_with_other("abc", "Hi abc")` => `True`
- `ends_with_other("HELLO", "hello")` => `True`
- `ends_with_other("HELLO WORLD", "hello")` => `False`
- `ends_with_other("abc", "def")` => `False`
Question 2B
(Module 3 – Input/Output)

Write a function called rmv_every_n_char that consumes a natural number, n and removes every nth character from string, inputted by the user. Afterwards it prints out the transformation of the string (see example) and returns the number of characters that are removed.

You can write your own prompt for the user input.

Examples:
If the user enters “cs116” after prompted,

• rmv_every_n_char(2) => 2, and prints the following to the screen: 'cs116' becomes 'c16' after removing every 2 characters.
• rmv_every_n_char(6) => 0, and prints the following to the screen: 'cs116' becomes 'cs116' after removing every 6 characters.
Write a function `multiples_of` that consumes a list of natural numbers (called `numbers`) and a positive natural number (called `n`), and returns a (new) list containing all entries in `numbers` which are multiples of `n`. The new list must be in the same relative order as `numbers`, and the original list should be unchanged. Use recursion or abstract list functions.

For example:

Constructing a new list:

- `multiples_of([], 4) => []`
- `multiples_of([18, 5, 19, 21, 300, 0, 4], 3) => [18, 21, 300, 0]`

❖ Note: The list that is consumed should remain the same.
Write a function modify_multiples that consumes a list of natural numbers (called numbers) and a positive natural number (called n), and mutates numbers so that all multiples of n are set to 0. The function returns None.

For example:

Mutating numbers:

```
# if nums = [], after calling modify_multiples(nums, 4),
#   nums is []

# if nums = [18, 5, 19, 21, 300, 0, 4], after calling
#   modify_multiples(nums, 3), nums is [0, 5, 19, 0, 0, 0, 4]
```
Write an accumulatively recursive function `find_all` that consumes a list of strings `lst` and a string `s`, and returns the list of indices of positions in `lst` with string `s`. Recall that the first position in a list has index 0.

For example,

- `find_all(["a", "v", "d", "v"], "v")` => `[1, 3]`
- `find_all(["a", "v", "d", "v"], "q")` => `[]`

**Bonus:** Write `find_all` as a generative recursive function and abstract list function.
In mathematics, there's a method of solving a large integer power of a number. The basic rules are:

1. \( x^{(2y+1)} = x \times (x^{2y}) \)
2. \( x^{(2y)} = (x^y) \times (x^y) \)

Write a generative recursive function `power_fast` that consumes two integers, `base` and `exp` and returns the value of `base^{exp}`.

For example,

- `power_fast(-11, 1)` => `-11`
- `power_fast(4, 3)` => `64`

Source: [https://course.ccs.neu.edu/cs2500f14/lab9.html](https://course.ccs.neu.edu/cs2500f14/lab9.html)
General Study tips

Review materials:
- Course notes
- Assignments
- Tutorial Problems
- Module Practices
- Style Guide

Review strategies:
- Spaced practicing
- Make own review notes
- Good Sleep and Rest
- Ask questions
- Teach your friends
- Come to office hours

\( (We \ were \ lonely \ last \ week.!) \) 😞

during reading week