CS 97 Discussion Section

Lun Liu, 1D

Common Mistakes: Week 9 Edition

- Not doing something to exit a while loop
  - In recursion, you have to do a bit of work to move closer to the base case every time and if not you'll recurse infinitely. Likewise, if you're not doing something to change the while loop condition, you'll loop infinitely!
- Putting initialization code inside a loop:
  - ```
    for x in l:
        sum = 0
        sum += x
    ```
  - OK, this is a dumb example but it happens way more than you'd think
    ```

Useful Python Tidbit: Range

- range(n) gives all numbers from 0 up to but **not including n [0,n)**

```
for x in range(2):
    print(x)
>> 0
>> 1
```

- range(n1, n2) gives you all integers **including n1** but **not including n2 [n1, n2)**

```
for x in range(1, 2):
    print(x)
>> 1
```

- range(n1, n2, n3) gives you all the integers from above with a step size of n3 which can be negative!

```
for x in range(9, 6, -2):
    print(x)
>> 9
>> -7
```

Think, "Increment n1 by n3 while it is less than or equal to n2."

Loop Advice

There are three ways of iterating over a container using loops:

1. By element (least general):

```
for x in my_list:
    doSomething(x)
```

2. By index (semi-general):

```
for i in range(len(my_list)):
    doSomething(my_list[i])
```

3. While (most general):

```
i = 0
while(i < len(my_list)):
    doSomething(my_list[i])
    i++
```

Loop Practice

Given a list of integers, return indices of the two numbers such that they add up to a specific target. You may assume that each input would have exactly one solution, and you may not use the same element twice.

```
def twoSum(nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: List[int]
        """
```

Loop Practice

Given a list of integers, return indices of the two numbers such that they add up to a specific target. You may assume that each input would have exactly one solution, and you may not use the same element twice.

```
def twoSum(nums, target):
        for i in range(len(nums)):
            for j in range(i + 1, len(nums)):
                if nums[i] + nums[j] == target:
                    return [i, j]

        return []
```

Loop Practice Tip

Anything that can be written recursively can also be written with a loop. Try rewriting old practice problems using loops! Remember these?

1. Write a function named count that takes an item e and a list l and returns the number of occurrences of e in l.
2. Write a function named minimum that returns the smallest element in a list of integers. You may assume the list is non-empty.
3. **Write a function named q that checks that every number in a list that is between 10 and 100 (inclusive) is even. For example, q([1,12,153,84,64,9]) returns True.**
4. Write a function named tails that takes a list and returns a list of lists containing the original list along with all tails of the list, from longest to shortest. For example, tails([1,2,3]) is [[1,2,3], [2,3], [3], []].
5. Write a function named flatten that takes a list of lists and flattens it into a single list. For example, flatten([[1,2], [3], [], [4,5,6]]) returns [1,2,3,4,5,6].
6. Write a function pairify that takes a list and pairs up consecutive elements of the list. If the list has an odd length, then the last element should be dropped from the result. For example, pairify([1,2,3,4,5]) returns [[1,2], [3,4]].
7. Write a function named rmDups that removes consecutive duplicates from a list. For example, rmDups([1,2,2,3,3,3,4,2,2,3]) returns [1, 2, 3, 4, 2, 3].

Todd's Two Rules

Two key points to understand everything there is to know:

key point #1: variables always contain references to data, never the data itself

key point #2: "copying" a variable copies the reference, not the data

I totally dare one of you to get a fake tattoo of this somewhere.

Reference Practice

```
l1 = [1,2,3]
l2 = [0,2,4]
print(id(l1) == id(l2))    #>> ?? is this always the answer?

print(id(l1[1]) == id(l2[1]))     #>>?? is this always the
answer?

l3 = [[0, 0], [1,1]]
l4 = [[0, 0], [1,1]]
print(l3[0] == l4[0]) # >> ??
print(id(l3[0]) == id(l4[0]))# >> ??
```

Reference Practice

```
def foo(bar):
    bar = 'new value'
    print (bar)


answer_list = 'old value'
foo(answer_list)          # >> ??
print(answer_list)        # >> ??
```

Reference Practice -- What is printed

```
def foo(bar):
    bar[1] = 5 * bar[1]
    bar = [2, 3, 4, 5]
    print(bar)

answer_list = [0, 1, 2, 3]
foo(answer_list)          # >> ??
print(answer_list)        # >> ??
```