

CS 97 Discussion Section




Lun Liu

Welcome back to CS97 Discussion!

- Some administrative things:
 - Potentially moving office hours for less overlap
 - HW 1 Solutions will be released by Monday morning.
 - Because of late days, we can't review them in discussion section until next week, but we'll make sure to go over them before the first midterm
 - HW 2 will be posted soon and not due until at least a week from Monday.

Syntax Review: Conditions

Remember that for our purposes:

- an `if` is followed by a condition
- a condition evaluates to either `True` or `False`
- we need `thing1 [some operator] thing2` to make a complete *expression*
- we can also use `and`, `or` to combine *expressions*, but remember that we need a complete *expression* on either side
 - Example: `if x > 7 or x == 3` 
 - `if x > 7 or == 3` 
 - `if x == 3 or 2` 

Syntax Review: Indexing Strings and Lists

- The syntax for strings and lists are very similar (basically the same).
 - Strings can often be treated as a list made up only of characters.
 - For both, starting counting from the left-most side beginning with 0
 - Question: how would you describe the index of the right-most item?
 - Use the "slice" notation to get a single item or multiple items

```
some_list[0]      # RETURN the first element of a list
some_list[0:4]    # RETURNS a sublist of items from position 0 to position 3
some_list[1:]     # RETURNS a sublist of items from positions 1 to end of list
some_list[:3]     # RETURNS a sublist of items from position 0 to 2
len(some_list)    # RETURNS length of items
some_list[len(some_list)-1] #RETURNS the last item in a string or list
```

What To Do If You Finish Early

Everyone works at a different pace! If you happen to finish a problem in discussion section early, take the following three steps:

- 1) Write "test cases" for your code. Test cases make sure your program returns the correct answer every time (including the wrong answer when it should be wrong!)
What values would you put into your code to see if it works correctly?
- 2) Find someone else who finished early. Look at their code: did they do it differently and if so why?
- 3) Help someone else! You can always work on your own homework or something else, but if you're feeling kind, try helping someone around you who's stuck.

Warm Up Questions #1

What do the following expressions evaluate to?

```
str = 'this is a string'
```

```
s0 = str[1:len(str) - 1] # what is the value of s0
```

```
s1 = str[3:] # what is the value of s1?
```

```
items = ['this', 'is', 'a', 'string']
```

```
s2 = items[1:][2][3] # what is the value of s2? HINT: break it into parts!
```

```
s3 = (items[1:] + [items[0]])[3][0] # what is the value of s3?
```

#Challenge problem:

```
not(5 > 7 and 3 <= 9) == (5 <= 7 or 3 > 9) # what does this evaluate to?
```

Warm Up Questions #2

How would you translate the following into code? (Try not to use IDLE for these questions! Write them into Text Editor first and then transfer them over.)

- an expression that evaluates to all positive numbers that are evenly divisible by 3 and 4
- $n \geq 0$ and $n \% 3 == 0$ and $n \% 4 == 0$:
- an expression that evaluates to the **last** letter in a String
- $s[\text{len}(s) - 1]$
- an expression that tells you whether a letter is a vowel or not
- $a == \text{"a"}$ or $a == \text{"o"}$

Warm Up Questions #2

How would you translate the following into code? (Try not to use IDLE for these questions! Write them into Text Editor first and then transfer them over.)

- an expression that evaluates to all positive numbers that are evenly divisible by 3 and 4
 - `x > 0 and x % 3 == 0 and x % 4 == 0`
`x > 0 and x % 12 == 0`
- an expression that returns the **last** letter in a String
 - `str[len(str)-1]`
- an expression that tells you whether a letter is a vowel or not
 - `x == 'a' or x == 'e' or x == 'i' or x == 'u' or x == 'o'`
 - or, `x in 'aeiou'` in Python only

Functions Review:

What are they? What do they want?

Functions are **reusable blocks of code** that may have some plug-in parts that you get to name!

They seem weird, but there are plenty of similar things in human languages! For example, **memes** in pop culture follow a similar format but you plug in different things to express different ideas and things like **the happy birthday song** are mostly similar other than plugging in a name and age.

What's the difference between defining and calling a function?

```
def birthday_song(name):  
    return "Happy Birthday to " + name  
birthday_song("Todd")
```

Function Practice

Remember quadratic equations? They take the form $ax^2 + bx + c = y$. Write a function that takes four arguments and returns y .

Now, use the function you just wrote, and write a function that takes values for a , b , c , and x and tells you whether or not " x " is a solution to that equation. Remember that a solution to a quadratic equation is a value for x that makes the whole thing equal to 0.

Function Solution

```
def quad(a, b, c, x):  
    y = a * pow(x, 2) + b * x + c  
    return y
```

```
def isSolution(a, b, c, x):  
    if quad(a, b, c, x) == 0:  
        return True  
    else:  
        return False
```

Common Mistakes-Week 2 Edition

- Forgetting that `and` and `or` are binary operators that need complete expressions on either side
 - `if x == 2 or x == 3`, not `if x == 2 or 3!`
- Using something other than square `[]` brackets for slice notation
 - `some_string[2:5]`, not `some_string(2:5)` or `{2:5}`
- "Indexing" in strings and lists starts at 0. Remember off by one errors?
- Not having faith! Trying to follow recursion in your head can be tricky. If each part works independently, sometimes you just have to let go and trust that the function will eventually return the right answer.

Recursion Review By Example

Remember that recursive functions are just functions that call themselves. Usually, it takes a look at the argument. If it already knows the answer, it returns. Otherwise, it performs a little bit of work and then calls itself with new arguments.

From the lecture: Given a list of integers, e.g. [3, -1, 2, 0, 7, -5], return the list modified so that it contains only positive (≥ 1) integers.

Recursion Review: Strategies

If you're stuck, try answering the following questions for this problem:

How can I break this into smaller pieces? What's the smallest possible input? How do I make it bigger or smaller? (recursive strategy)

When are some situations I have enough information to give a definite answer to this question? (base cases)

What *operations* do I have to do repeatedly to answer this question? (recursive body)

Recursion Solution

```
def positives(l):  
    if l == []:  
        return []  
    else:  
        head = l[0]  
        tail = l[1:] #on a size 1 list, returns empty list  
        posRest = positives(tail)  
        if head > 0:  
            return [head] + posRest  
        else:  
            return posRest
```

TIP: the "head" and "tail" expressions you see here are an example of a kind of Pythonic idiom. Much like idioms in English ("What's up?"), they are common expressions in Python that don't make as much sense translated word-for-word ("What is located above you, vertically?") in other computing languages as much as they communicate a general idea ("What new things have happened to you lately?").

Here it means: "I have a list! Give me the first item and everything that isn't the first item."

Recursion Practice #1

Read the following code. What does it do? How many times is this function called, and with what arguments? Hint: try expanding the last statement with each new function call!

```
# assume n > 0
def foo(n):
    if n == 1:
        return 3
    elif n == 2:
        return 1
    else:
        return foo(n - 1) + foo(n - 2)

>> foo(4)
```


Recursion Practice: Once More With Feeling

What does this function return when $x=36$ and $y=12$? Can you come with a general description of what it does?

```
def mystery(x, y)
    if x < y:
        return mystery(x, y-x)
    elif y < x:
        return mystery(x-y, x)
    else:
        return x
```

Recursion Practice #2

Write code to reverse a list, making sure you use recursion in a useful way.

```
>>> revList([1,2,3,4])
```

```
[4, 3, 2, 1]
```

Stretch question: if you finish that early, try using the function you wrote to write another function that takes in a list of characters and returns True if the list of strings represents a palindrome.

A palindrome is a sequence of letters like "racecar" where it's the same when reversed :) It should work for either strings or lists

Recursion Practice #2: A Solution

```
def revList(l):  
    if l == []:  
        return []  
    else:  
        head = l[0]  
        tail = l[1:]  
        return revList(tail) + [head]
```