Supplemental Instruction
CS-141 Final Review

Week 1: Introduction: Python Functions
In Python ____**Indentation**____ matters. It is used to define the body of functions, loops, and conditionals.

What are some turtle functions and how do you use them in your code.

Left, right, forward, backward, circle

Week 2: Parameters and Argument Values, Conditional Statements, General Recursion
What are pythons 3 condition keywords:

If, else, elif

Below write the declaration of a function *foobar* that takes parameters *foo* and *bar*.

```
def foobar (foo, bar):
    pass
```

Write recursive function to do factorial and Fibonacci. Also do Substitution trace.

```
def fact (n ):

    if n == 1 or n == 0:
        return 1
    else:
        return n * fact(n – 1)
```

```
def fib (n ):

    if n == 0:
        return 0
    elif n == 1 or n == 2:
        return 1
    else:
        return fib(n - 2) + fib(n - 1)
```

```
fact(3) = 3 ∗ fact(2)
        = 3 ∗ (2 ∗ fact(1))
        = 3 ∗ (2 ∗ (1 ∗ fact(0)))
        = 3 ∗ (2 ∗ (1 ∗ 1))
        = 3 ∗ (2 ∗ 1)
        = 3 ∗ 2
        = 6
```

```
fib(3) = fib(2) + fib(1)
       = (fib(1) + fib(0)) + fib(1)
       = (1 + fib(0)) + fib(1)
       = (1 + 0) + fib(1)
       = 1 + fib(1)
       = 1 + 1
       = 2
```

Week 3: Tail Recursion, 'Fruitful' Functions, & Types
What is different between a recursive function and a tail recursive function?

The final call in a tail recursive call is strictly 'return function(params)' with nothing else

Are the functions you wrote above tail recursive?

Answers will vary. Is the last line of your function just 'return function()'? These given solutions are *not* tail recursive.

Write tail recursive functions to factorial and Fibonacci number. Also do substitution trace.

def factAccum (n , a):
 	
if n == 0:
return a
else :
return factAccum (n - 1, n * a)
	def fibAccum (n , a , b):
	
	if n == 0:
	return a
	elif n == 1:
	return b
	else :
	return fibAccum (n - 1, b , a + b)

fact(3) = factAccum(3, 1) = factAccum(2, 3) = factAccum(1, 6) = factAccum(0, 6) = 6	fib(3) = fibAccum(3, 0, 1) = fibAccum(2, 1, 1) = fibAccum(1, 1, 2) = 2

Week 4: From Recursion to Iteration: while, break. Assignment, Complexity

Compare the key words; break, continue, return and print.

Break: Completely breaks out of loop

Continue: Jumps to next iteration without breaking loop

Return: The actual output of the function

Print: You get a printed output in the console

Write iterative functions to compute factorial and Fibonacci number. Also do a timeline.
(in hindsight this was harder than anticipated)

def factIter (n):
res = 1
while n > 1:
res = res * n
n -= 1
return res
	def fibIter (n):
	last = 1
	nextToLast = 0
	if n == 0:
	return nextToLast
	elif n == 1:
	return last
	else:
	for i in range(2, n + 1):
	res = nextToLast + last
	nextToLast = last
	last = res

<table>
<tr><td></td><td>return res</td></tr>
<tr><td>fact(3) =

<table><tr><td>Time</td><td>0</td><td>1</td><td>2</td><td>3</td></tr><tr><td>res</td><td>1</td><td>3</td><td>6</td><td>6</td></tr><tr><td>n</td><td>3</td><td>2</td><td>1</td><td>0</td></tr></table></td><td>fib(3) =

<table><tr><td>Time</td><td>0</td><td>1</td></tr><tr><td>n</td><td>3</td><td>2</td></tr><tr><td>last</td><td>1</td><td>1</td></tr><tr><td>nextToLast</td><td>0</td><td>1</td></tr><tr><td>res</td><td>1</td><td>2</td></tr><tr><td>i</td><td>3</td><td>4</td></tr></table></td></tr>
</table>

Week 5: Strings, for loops, Files

A string is defined as…

A list of characters

What is the result of each string operation.
s = “CS1 final is gonna be easy”

s[3] = **‘ ’**

s[6:1] = **‘ ’**

s[6:11] = **‘nal i’**

s[13:4:-1] = **‘g si lani’**

s[16:] + s[11:16] + s[:6] + s[6:12] = ‘**na be easys gonCS1 final is’**

Write a function to iterate through a file and print the reverse of each line:

```
def readFile( fname):
    file = open(fname)
    for line in file:
        print( line.strip()[::-1] ) #strip to remove extra newline
    file.close()
```

Week 6: Testing Debugging

What are some good test cases for a function that reverses a string?

Empty string
String with one character
String with two characters
String with even length
String with odd length

Week 7: Python Lists and Tuples, Searching, Sorting

Compare Lists to Tuples.

Lists are mutable and are denoted with [], tuples are immutable and are denoted with ()

What are all the sorting algorithms we learned in class?

Quick sort, merge sort, insertion sort

Show the steps a Binary Search Algorithm will take to find 74 in the sorted list
[12, 56, 74, 96, 112, 114, 123, 567].

[**12, 56, 74, 96, 112, 114, 123, 567**]
[**12, 56, 74, 96, 112**, 114, 123, 567]
[**12, 56, 74,** 96, 112, 114, 123, 567]
[12, **56, 74,** 96, 112, 114, 123, 567]
[12, 56, **74,** 96, 112, 114, 123, 567]

Week 8: Optimal Sorting Algorithms

	Best Case	Average Case	Worst Case
Linear Search	**O(1)**	**O(N)**	**O(N)**
Binary Search	**O(1)**	**O(log N)**	**O(log N)**
Insertion Sort	**O(N)**	**O(N ^ 2)**	**O(N ^ 2)**
Merge Sort	**O(N log N)**	**O(N log N)**	**O(N log N)**
Quick Sort	**O(N)**	**O(N log N)**	**O(N ^ 2)**

Apply merge sort on the following list: [38, 27, 43, 3, 9, 82, 10]

[38, 27, 43, 3, 9, 82, 10]

[38, 27, 43, 3] [9, 82, 10]

[38, 27] [43, 3] [9, 82] [10]

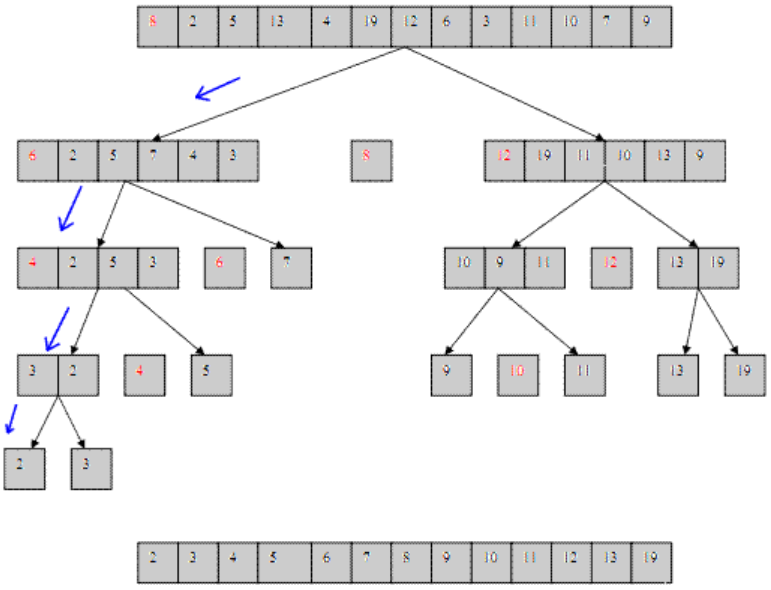
[38] [27] [43] [3] [9] [82] [10]

[27, 38] [3, 43] [9, 82] [10]

[3, 27, 38, 43] [9, 10, 82]

[3, 9, 10, 27, 38, 43, 82]

Do quick sort on the following list: [8, 2, 5, 13, 4, 19, 12, 6, 3, 11, 10, 7, 9]
Just using first element as pivot, kinda low res, sorry about that



Week 9: Python Dictionaries, Sets, User-defined structures

Write a struct for a student, that contains, name, major, gpa, and current courses

```
@dataclass
Class Student:
    name: str
    major: str
    gpa: float
    current_courses: List
```

Make yourself using the struct you made.

stu = Student(‘Chad, ‘Business’, 4.20, [‘Wine Tasting, ‘Beer Tasting, ‘Yard Games’, ‘Juggling’])

Write a function that takes in a filename, the file contains several lines of random words, and parse this file. The function should return a dictionary that contains all the words in the file and the number of times they occur and a dictionary that has the total number of occurrences of each character in the file.

```
def parsefile(filename):
    # creating dictionaries
    words = dict()
    chars = dict()
    # opening file
    file = open(filename)
    # iterating over every line
    for line in file:
        parsedLine = line.strip().split()
        # iterating over every word in the line
        for word in parsedLine:
            if word in words:
                words [word] += 1
            else:
                words [word] == 1
            # iterative over every character in each word
            for char in word:
                if char in chars:
                    chars [char] += 1
                else:
                    chars [char] == 1

    # close the file because we used open()
    file.close()

    return words, chars
```

Complete the table:

	Sets	Both	Dictionaries
Keys? Values?	**Keys**		**Keys and values**
Create an empty instance of the structure	**mySet = set()**	**S = {}**	**Book = dict()**
Ordered?	**no**	**no**	**no**
Look up time (keys only)	**Constant**	**Constant**	**Book['key'], Constant**
What type(s) can the keys / values be?	**String, tuple, anything immutable**		**Any**

Time it takes to iterate over keys? Values?	**O(N)**		**O(N)**

Week 10: Structural Recursion: Linked Structures

Compare the time complexities of linked lists and regular python lists.

	Linked List	Python List
Indexing	**O(N)**	**O(1)**
Insert/delete Beginning	**O(1)**	**O(N)**
Insert/delete End	**O(N)**	**O(1)**
Insert/delete Middle	**Search time + O(1)**	**O(N)**

Write the append function for Linked Lists:

```
def append( lst, value ):
    if lst.head == None :
        lst.head = Node( value, None )
    else:
        curr = lst.head
        while curr.next != None :
            curr = curr.next
        curr.next = Node( value, None )
    lst.size += 1
```

Week 11: Stacks & Queues

What are all the stack and queue operations we have.

Stack	Queue
Pop() **Push()** **Top()** **mkEmptyStack()** **size()** **isEmpty()**	**Enqueue()** **Dequeue()** **Front()** **Back()** **mkEmptyQueue()** **size()**

Show the resulting stack after each operation.
stk = mkEmptyStack()
push(stk, ‘a’)

push(stk, ‘b’)
push(stk, ‘c’)
pop(stk)
pop(stk)
push(stk, ‘z’)
pop(stk)
push(stk, ‘y’)
push(stk, ‘z’)

ZYA

Week 12: Trees

A binary Search tree is…

Root value is greater than all the left side’s values, but less than the right side’s values

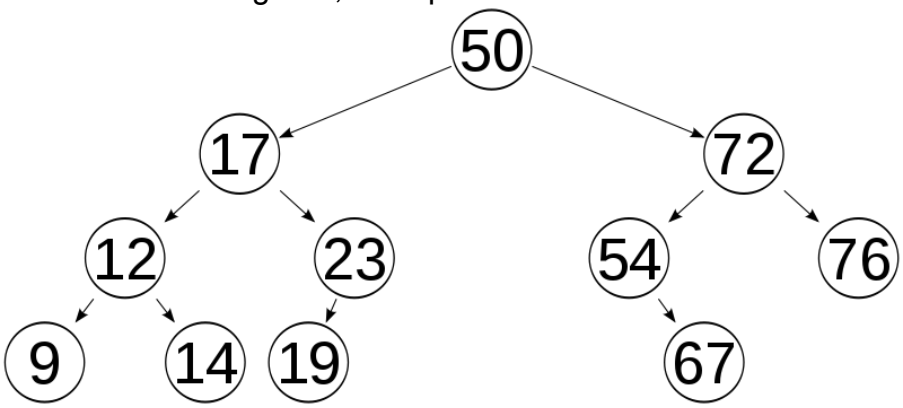
What makes a Binary Search Tree different from a normal tree?

BST can only have up to 2 children per node

How would we initialize a simple tree?

coolTree = Tree(Tree(None, ‘B’, None), ‘A’, Tree(None, ‘C’, None))

Given the following tree, what path is taken to find the value 19?



50 -> 17 -> 23 -> 19

Use the above tree for the following traversals.

I recommend to check this out for practicing tree traversal:
http://faculty.cs.niu.edu/~mcmahon/CS241/Notes/Data_Structures/binary_tree_traversals.html

Inorder: **9, 12, 14, 17, 19, 23, 50, 54, 67, 72, 76**

Preorder: **50, 17, 12, 9, 14, 23, 19, 72, 54, 67, 76**

Postorder: **9, 14, 12, 19, 23, 17, 67, 54, 76, 72, 50**

Week 10: Hashing

What are two methods of dealing with collision?

Open addressing – If there's a collision, send it to the nearest open address

Chaining – If there's a collision, just add it in the same row, but in a linked list/list

Show the resulting hash table after the following puts.

hash function: a = 0, b = 1, …, z = 25 then mod by capacity(5 in this case)

put('cat', 1) – 2 % 5 = 2

0	('ant', 1), ('frog', 1)
1	('bee', 1), ('lion', 1)
2	('cat', 1), ('cheetah', 1)
3	('dog', 1), ('dog', 2)
4	('slug', 1)

put('dog', 1) – 3 % 5 = 3
put('ant', 1) – 0 % 5 = 0
put('frog', 1) – 5 % 5 = 0
put('slug', 1) – 19 % 5 = 4
put('dog', 2) – 3 % 5 = 3
put('bee', 1) – 1 % 5 = 1
put('lion', 1) – 11 % 5 = 1
put('cheetah', 1) – 2 % 5 = 2