Supplemental Instruction

CS-141 Final Review

**Week 1: Introduction: Python Functions**

In Python \_\_\_\_\_\_ 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.

**Week 2: Parameters and Argument Values, Conditional Statements, General Recursion**

What are pythons 3 condition keywords:

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

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

|  |  |
| --- | --- |
|  |  |
|  |  |

**Week 3: Tail Recursion, 'Fruitful' Functions, & Types**

What is different between a recursive function and a tail recursive function?

Are the functions you wrote above tail recursive?

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

|  |  |
| --- | --- |
|  |  |
|  |  |

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

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

Write iterative functions to compute factorial and Fibonacci number. Also do a timeline.

|  |  |
| --- | --- |
|  |  |
|  |  |

**Week 5: Strings, for loops, Files**

A string is defined as…

What is the result of each string operation.

s = “CS1 final is gonna be easy”

s[3] = s[6:11] =

s[6:1] = s[13:4:-1] =

s[16:] + s[11:16] + s[:6] + s[6:12] =

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

**Week 6: Testing Debugging**

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

**Week 7: Python Lists and Tuples, Searching, Sorting**

Compare Lists to Tuples.

What are all the sorting algorithms we learned in class?

Show the steps a Binary Search Algorithm will take to find 74 in the sorted list

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

**Week 8: Optimal Sorting Algorithms**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Best Case | Average Case | Worst Case |
| Linear Search |  |  |  |
| Binary Search |  |  |  |
| Insertion Sort |  |  |  |
| Merge Sort |  |  |  |
| Quick Sort |  |  |  |

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

Do quick sort on the following list: [8, 2, 5, 13, 4, 19, 12, 6, 3, 11, 10, 7, 9]

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

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

Make yourself using the struct you made.

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.

Complete the table:

|  |  |  |
| --- | --- | --- |
|  | Sets | Dictionaries |
| Keys? Values? |  |  |
| Create an empty instance of the structure |  |  |
| Ordered? |  |  |
| Look up time (keys only) |  |  |
| What type(s) can the keys / values be? |  |  |
| Time it takes to iterate over keys? Values? |  |  |

**Week 10: Structural Recursion: Linked Structures**

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

|  |  |  |
| --- | --- | --- |
|  | Linked List | Python List |
| Indexing |  |  |
| Insert/delete Beginning |  |  |
| Insert/delete End |  |  |
| Insert/delete Middle |  |  |

Write the append function for Linked Lists:

**Week 11: Stacks & Queues**

What are all the stack and queue operations we have.

|  |  |
| --- | --- |
| Stack | Queue |
|  |  |

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’)

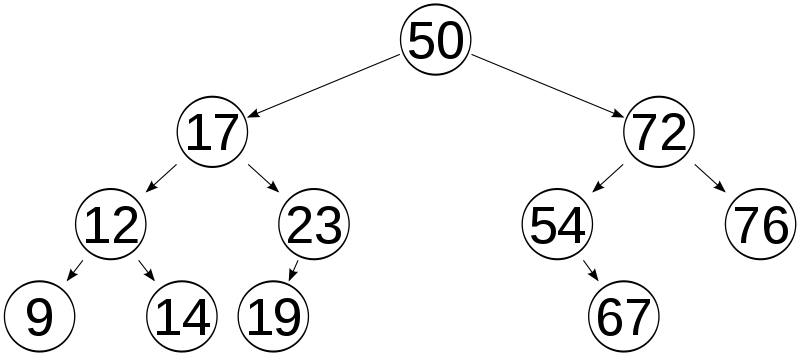
**Week 12: Trees**

A binary Search tree is…

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

How would we initialize a simple tree?

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



Use the above tree for the following traversals.

Inorder:

Preorder:

Postorder:

**Week 10: Hashing**

What are two methods of dealing with collision?

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