Data Structures (CS 1520) Spring 2017

Time and Place: 8 - 9:15 AM Tuesday and Thursday in Wright 10 **and** either:

- Section 01: 8 – 9:50 AM Wednesday in **Wright 112**, or
- Section 02: 10-11:50 AM Wednesday in **Wright 112.**

Web-site: **http://www.cs.uni.edu/~fienup/cs1520s17/** (Lecture videos posted after class)

Class Email List: Send messages to Google group for the course at CS-1520-01-spring@uni.edu or CS-1520-02-spring@uni.edu

Instructor: Mark Fienup (fienup@cs.uni.edu)
Office: ITTC 313
Phone: 319-273-5918 (Home 319-266-5379)
Office Hours: M: 8-11:45, 1:10-3, T: 9:30-11:45, 1:10-2, W: 1:10-3, Th: 9:30-11:45, 1:10-2, F: 8-10

Prerequisite: Intro. to Computing (CS 1510), and pre- or corequisite Discrete Structures (CS 1800)

Goals: After this course, you should be able to (1) write "medium" sized programs using algorithmic problem solving and functional decomposition in analysis, design, and implementation, (2) implement and understand the algorithms for manipulating data structures: stacks, queues, lists, strings, trees, and graphs, and (3) be able to select appropriate data structures when writing medium size programs.

Text: *Problem Solving with Algorithms and Data Structures Using Python*, 2nd edition, by Bradley N. Miller and David L. Ranum. Franklin, Beedle & Associates. ISBN: 978-1-59028-257-1 (http://www.pythonworks.org/pythonds)
Free on-line version of the textbook: http://interactivepython.org/courselib/static/pythonds/index.html

Assignments: Assignments will consist of weekly laboratory exercises along with concurrent weekly or bi-weekly programming assignments.

Pedagogic Approach: In class, I'll tend to break up the lecture with active (and group) learning exercises to aid learning. While this is not formally graded, part (5%) of your grade will be based on your participation in (and attendence for) these in-class activities. Students benefit by (1) increased depth of understanding, (2) increased comfort and confidence, (3) increased motivation, and (4) being better prepared to work in groups on the job. This might sound great, but it will require you (and me) to work differently to prepare for class. Before the class, you must read the assigned reading, thought about what I asked you to think about, etc.; otherwise you won't be able to effectively participate during class.

Grading policy: There will be three tests (including the final). Tentative test dates and weighting of course components are:

In-class Work:	5 %	
Labs:	15 %	
Programming Assignments:	20 %	
In-class Test 1:	20 %	(Thursday, February 16)
In-class Test 2:	20 %	(Thursday, March 30)
Final:	20 %	(Tuesday, May 2 from 8 - 9:50 AM in Wright 10)

Grades will be assigned based on straight percentages off the top student score. If the top student's score is 92%, then the grading scale will be, i.e., 100-82 A, 81.9-72 B, 71.9-62 C, 61.9-52 D, and below 52 F. Plus and minus grades will be assigned for students near cutoff points.

Scholastic Conduct: You are responsible for being familiar with the University' Academic Ethics Policies (http://www.uni.edu/pres/policies/301.shtml). Copying from other students is expressly forbidden. Doing so on exams or assignments will be penalized every time it is discovered. The penalty can vary from zero credit for the copied items (first offense) up to a failing grade for the course. If an assignment makes you realize you don't understand the material, ask questions designed to improve your understanding, *not* ones designed to discover how another student

solved the assignment. The solutions to assignments should be **individual, original** work unless otherwise specified. Remember: discussing assignments is good. Copying code or test-question answers is cheating.

Any substantive contribution to your assignment solution by another person or taken from a publication (**or the web**) should be properly acknowledged in writing. Failure to do so is plagiarism and will necessitate disciplinary action. In addition to the activities we can all agree are cheating (plagiarism, bringing notes to a closed book exam, texting during an exam, etc.), assisting or collaborating on cheating is cheating (e.g., supplying code for another student). Cheating can result in failing the course and/or more severe disciplinary actions.

Special Notices:

- In compliance with the University of Northern Iowa policy and equal access laws, I am available to discuss appropriate academic accommodations that may be required for students with disabilities. Requests for academic accommodations are to be made during the first three weeks of the semester, except for unusual circumstances, so arrangements can be made. Students are encouraged to register with Student Disability Services, 103 Student Health Center, to verify their eligibility for appropriate accommodations.
- I encourage you to utilize the Academic Learning Center's assistance with writing, math, science, reading, and learning strategies. There is no charge for currently-enrolled UNI students. UNI's Academic Learning Center is located in 007/008 ITTC. Visit the website at http://www.uni.edu/unialc/ or phone 319-273-2361 for more information.

Data Structures Schedule Fall 2016

Lect #	Tuesday		Thursday	
1	1/10	Ch. 1: Python Review; Classes	1/12	Ch. 2: Algorithm Analysis; Big-oh; timing
3	1/17	Ch. 2: Performance of Python Built-in data structures	1/19	Ch. 3: Linear data structures; stack implementations
5	1/24	Stack applications; Queue implementations	1/26	Queue applications; Deque
7	1/31	Ch. 6.6: Priority Queue: binary heap implementation	2/2	Unordered Lists implementations
9	2/7	Ordered List implementation	2/9	Ch. 4: Recursion; Run-time stack
11	2/14	Review for Test 1	2/16	**Test 1**
13	2/21	Recursion examples: backtracking and dynamic programming coin-change problem	2/23	Ch. 5: Searching: linear and binary search
15	2/28	Searching via hashing: chaining/closed-address implementations	3/2	Open-address implementations; Simple sorts: bubble, selecton, and insertion sorts
17	3/7	Advanced sorts: heap, merge and quick sorts	3/9	Ch. 6: tree terminology, binary tree implementation, traversals, parse tree application
19	3/21	Binary Search Tree implementation	3/23	Binary Search Tree delete method
21	3/28	Review for Test 2	3/30	**Test 2**
23	4/4	AVL trees	4/6	File structures vs. In-memory Data Structures
25	4/11	Ch. 7: Graph terminology, traversals (BFS and DFS)	4/13	Graph implementations Select Graph algorithms: topological sort
27	4/18	Select Graph algorithms: Dijkstra's algorithm, Prim's algorithm	4/20	Select Graph algorithms: TSP backtracking and Approximation algorithm
29	4/25	TSP: Backtracking and Best-first search Branch-and-bound	4/27	Review for Final Exam
Final: 8:00 - 9:50 AM Tuesday, May 2 in Wright 10				

Data Structures (CS 1520) Lecture 1 Name: _____

1. The `print` function has optional *keyword arguments* which can be listed last that modify it behavior. The `print` function syntax: `print(value,...,sep=' ',end='\n', file=sys.stdout)`

a) Predict the expected output of each of the following.

Program	Expected Output

`print('cat',5,'dog')`
`print()`
`print('cat',5,end='')`
`print('_horse')`
`print('cow')`
	cat␣5␣dog\n
	\n
	cat␣5␣horse\n
	cow\n

Program	Expected Output
`print ('cat',5,'dog',end='#',sep='23')`	cat23523dog#
`print ('cat',5,'dog',sep='23','horse')`	error
`print ('cat',5,'dog',sep='>'*3)`	cat>>>5>>>dog\n

2. Review of assignment statements. Predict the output of the following programs

```
a = 123
b = a
a += 1
print ('a is', a)
print ('b is', b)
```

a is 124
b is 123

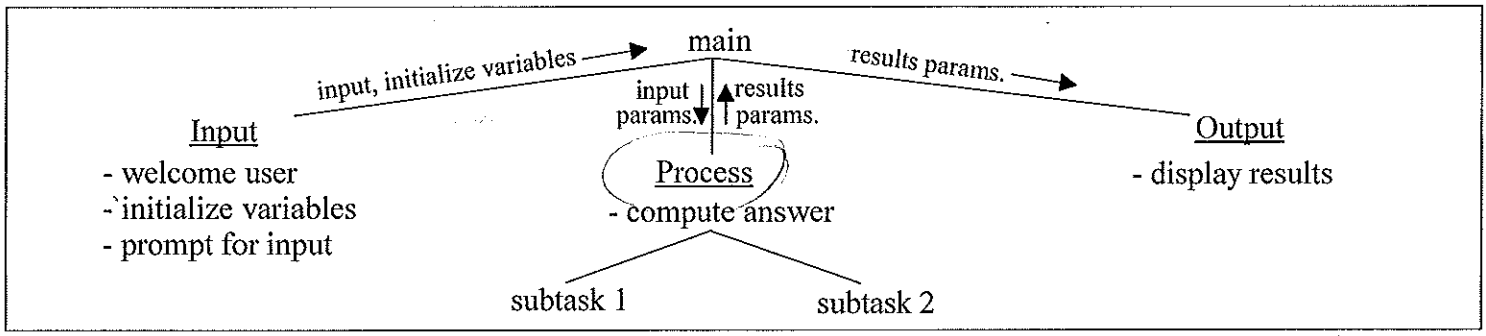
```
c = ['cat', 'dog']
d = c
c.append('cow')
print('c is', c)
print('d is', d)
```

c is ['cat','dog','cow']
d is ['cat','dog','cow']

```
c = 'cat'
d = c
c += 'fish'
print('c is', c)
print('d is', d)
```

c is catfish
d is cat

Most simple programs have a similar functional-decomposition design pattern (IPO - Input, Process, Output):

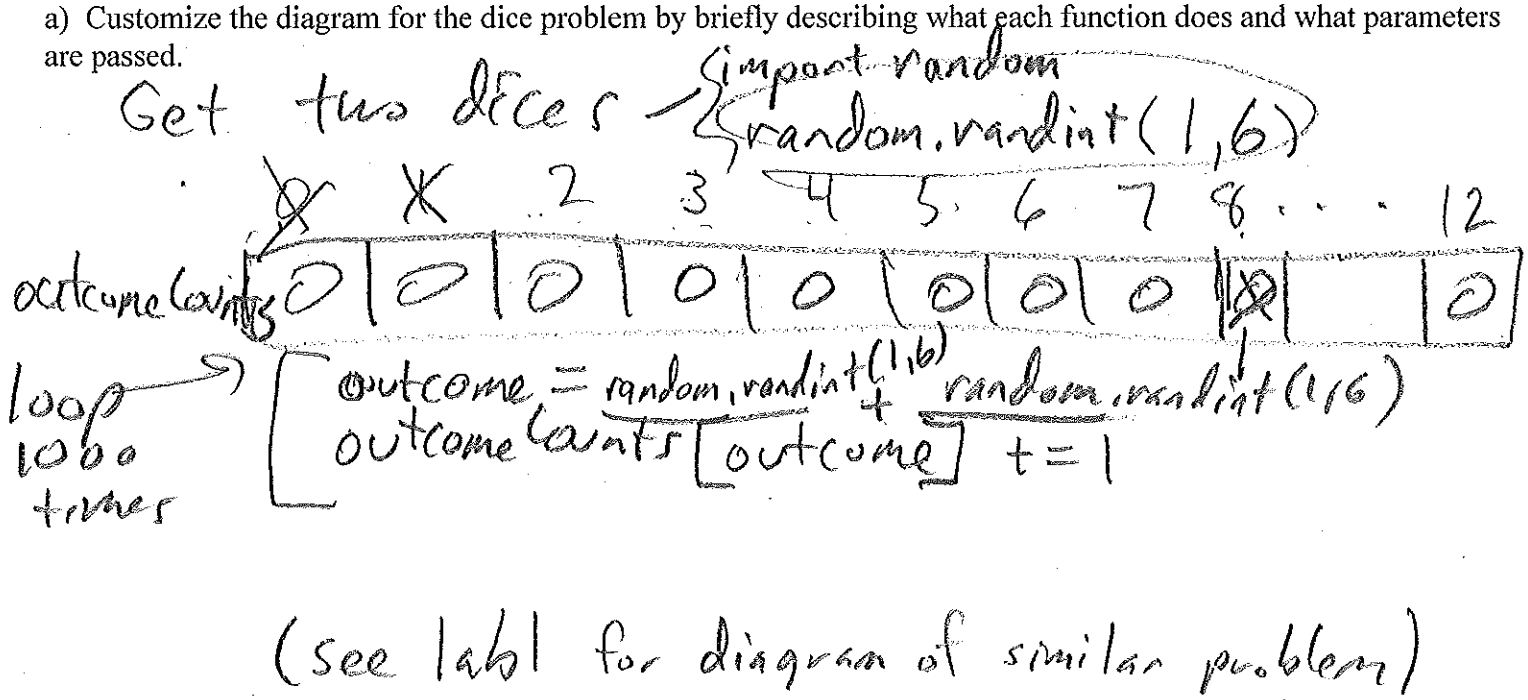



```
""" Simple IPO program to sum a list of numbers. """
def main():
    label, values = getInput()
    total = sum(values)
    displayResults(label, total)

def getInput():
    """ Get label and list of values to sum."""
    label = input("What are we summing? ")
    numberOfValues = int(input("How many values are there? "))
    values = []
    for i in range(numberOfValues):
        values.append(eval(input("Enter the next number: ")))
    return label, values

def displayResults(label, total):
    """ Display sum of values. """
    print("The sum of", label, "values is", total)

main()   # starts the main function running
```

```
What are we summing? money
How many values are there? 4
Enter the next number: 10
Enter the next number: 20
Enter the next number: 30
Enter the next number: 50
The sum of money values is 110
```

3. Design a program to roll two 6-sided dice 1,000 times to determine the percentage of each outcome (i.e., sum of both dice). Report the outcome(s) with the highest percentage.

a) Customize the diagram for the dice problem by briefly describing what each function does and what parameters are passed.



(see lab1 for diagram of similar problem)

b) An alternative design methodology is to use object-oriented design. For the above dice problem, what objects would be useful and what methods (operations on the objects) should each perform?

Data Structures (CS 1520) Lab 1 Name:_____

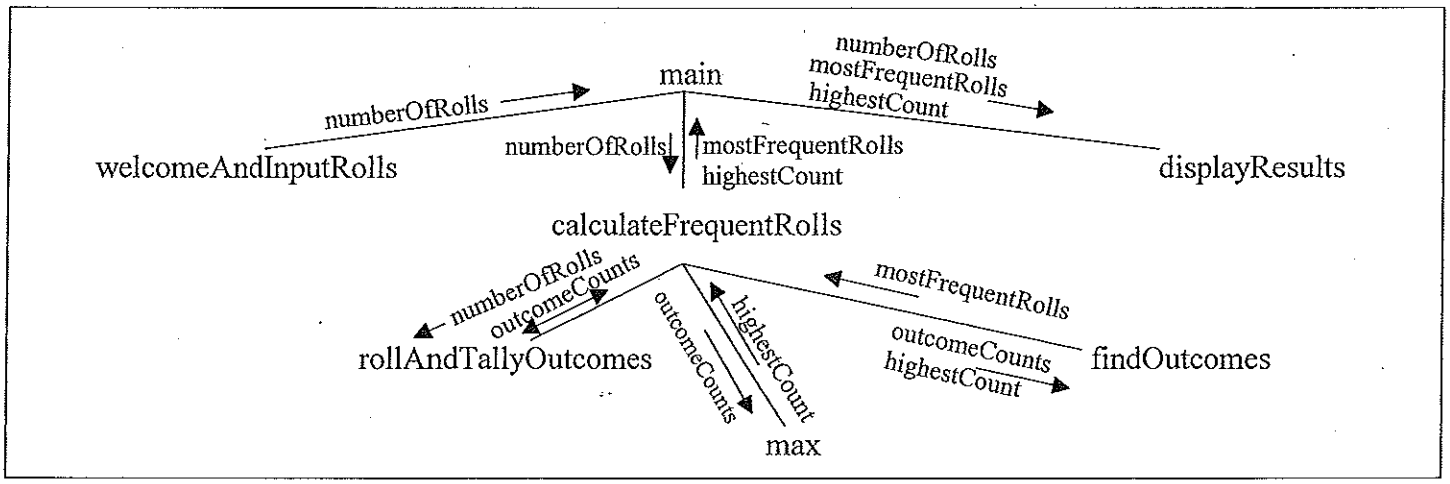
Objective: To practice writing Python code.

To start the lab: Download and unzip the file lab1.zip from
http://www.cs.uni.edu/~fienup/cs1520s17/labs/lab1.zip

Part A: In the folder lab1, open the diceOutcomes.py program in IDLE. (Right-click on diceOutcomes.py | Edit with IDLE) It contains a partial program we started to discuss in class to solve the problem:

> "Write a program to roll two 6-sided dice 1,000 times to determine the percentage of each outcome (i.e., sum of both dice). Report the outcome(s) with the highest percentage."

I decided to functional-decomposition this problem as:



`main` - provides an outline of program by calling top-level functions
`welcomeAndInputRolls` - Displays welcome message for the user. Gets and returns the number of dice rolls from the user.
`calculateFrequentRolls` - Rolls the dice the correct number of times, tallies the outcomes, and returns a list of outcomes with the highest count and highest count.
`rollAndTallyOutcomes` - Rolls the dice the correct number of times and tallies the outcomes. Returns a list of tallies with the index being the outcome.
`max` - built-in function to return the largest item in an iterable data structure like a list.
`findOutcomes` - Returns a list of outcomes with the highest count.
`displayResults` - Displays the outcome(s) with the highest percentage.

Consider running the program with only 10 dice rolls instead of 1,000. The program output with some extra debugging prints showing the two Python lists used: outcomeCounts and mostFrequentRolls.

```
This programs rolls two 6-sided dice many times to
determine the outcome(s) with the highest percentage.
How many times would you like to roll the pair of dice? 10

outcomeCounts: [0, 0, 1, 0, 2, 1, 0, 3, 0, 0, 3, 0, 0]
mostFrequentRolls: [7, 10] and highestCount: 3
The highest percentage is 30.0 for outcome(s): 7 10
```

Your task for lab 1 is to complete the code for the `rollAndTallyOutcomes` and `findOutcomes` functions.

After you have working code, raise your hand and demonstrate your code.

If you complete all parts of the lab, nothing needs to be turned in for this lab. If you do not get done today, then show me the completed lab in next week's lab period. When done, remember save your program to a USB drive (or email to yourself).