Data Structures (CS 1520) Lecture 2 Name: Mark F.

1. An alternative to functional-decomposition design is to use object-oriented design(OOD). For the following program, what objects would be useful and what methods (operations on the objects) should each support?
"Write a program to roll two 6-sided dice 1,000 times to determine the percentage of each outcome (i.e., sum boths dice). Report the outcome(s) with the highest percentage." (You only need consider the program's OOD)

Die objects with methods; roll, getRoll, etc.
TallySheet object with methods: addLabel, increment label, str,

2. Consider the Die and AdvancedDie classes from the Python Summary handed out last class.

a) What data attributes of AdvancedDie are inherited from the parent Die class? _currentRoll

b) What new data attributes are added as part of the subclass AdvancedDie? _numSides

c) Which Die class methods are used directly for an AdvancedDie object? getRoll

d) Which Die class methods are redefined/overridden by the AdvancedDie object? __init__, roll, __str__

e) Which methods are new to the AdvancedDie class and not in the Die class? getSides, __eq__, etc.

f) If `die1` and `die2` are AdvancedDie objects, then the statement "`if die1 == die2:`" invokes the `__eq__` method of AdvancedDie with `die1` "passed" as `self` and `die2` passed as `rhs_Die`.

```
def __eq__(self, rhs_Die):
    """Overrides default '__eq__' operator to allow for deep comparison of dice"""
    return self._currentRoll == rhs_Die._currentRoll
```

What would the code be for AdvancedDie `__le__` method to allow for the "`if die1 <= die2:`" statement?

```
def __le__(self, rhs_Die):
    return self._currentRoll <= rhs_Die._currentRoll
```

g) Good software engineering practice is to include *precondition* and *postcondition* comments on each method/function where the:

- *precondition* - indicates what must be true for the method to work correctly. Typically, the precondition describes the valid values of the parameters. If the precondition is not satisfied, the method does not need to work correctly!
- *postcondition* - describes the expected state after the method has executed

Consider the AdvancedDie constructor:

```
class AdvancedDie(Die):
    """Advanced die class that allows for any number of sides"""

    def __init__(self, sides = 6):
        """Constructor for any sided Die that takes an the number of sides
        as a parameter; if no parameter given then default is 6-sided."""

        Die.__init__(self)  # call Die parent class constructor
        self._numSides = sides
        self._currentRoll = randint(1, self._numSides)
```

"4eq"

code to check for int

if not isinstance(sides, int):
raise TypeError, "sides must be positive int"

What precondition and postcondition comments should we add?

precond: sides is an int with a positive value.
post: currentRoll initialize to random value between 1 and sides

h) If a method/function has a precondition that is not met when invoked (e.g., `die1 = AdvancedDie("six")`), why should the method raise an error? To aid debugging, by having the program crash immediately when the error occurred instead of later when it might be hard to track down.

3. **Terminology:**

problem - question we seek an answer for, e.g., "What is the sum of all the items in a list/array?"

parameters - variables with unspecified values

problem instance - assignment of values to parameters, i.e., the specific input to the problem

	0	1	2	3	4	5	6
myList:	5	10	2	15	20	1	11

sum: ? 64

(number of elements) n: 7

algorithm - step-by-step procedure for producing a solution

basic operation - fundamental operation in the algorithm (i.e., operation done the most) Generally, we want to derive a function for the number of times that the basic operation is performed related to the *problem size.*

problem size - input size. For algorithms involving lists/arrays, the problem size is the number of elements ("n").

Big-oh notation (*O*( ) - As the size of a problem grows (i.e., more data), how will our program's run-time grow.

Consider the following `sumList` function.

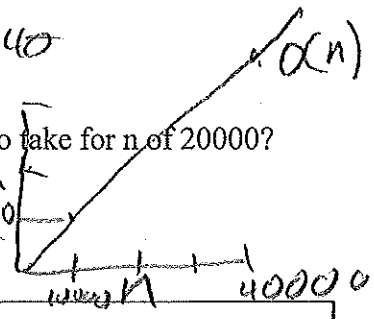
```
def sumList(myList):
    """Returns the sum of all items in myList"""
    total = 0
    for item in myList:
        total = total + item
    return total
```

a) What is the basic operation of `sumList` (i.e., operation done the most) ? addition +=

b) What is the problem size of `sumList`? len(myList) = "n"

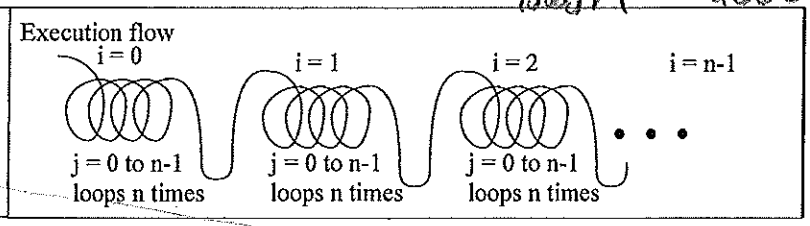
c) If n is 10000 and `sumList` takes 10 seconds, how long would you expect `sumList` to take for n of 20000? 20 sec

d) What is the big-oh notation for `sumList`? O(n) "linear"



4. Consider the following `someLoops` function.

```
def someLoops(n):
    total = 0
    for i in range(n):
        for j in range(n):
            total = total + i + j
    return total
```



a) What is the basic operation of `someLoops` (i.e., operation done the most)? body of inner-most loop

b) How many times will the basic operation execute as a function of n? n^2

c) What is the big-oh notation for `someLoops`? $O(n^2)$ "quadratic"

d) If we input n of 10000 and `someLoops` takes 10 seconds, how long would you expect `someLoops` to take for n of 20000? $O(n^2)$ $T(n) = cn^2 + c_2n + c_3$ $T(10000) = c\,10000^2 = 10\text{sec}$

$c = \frac{10\text{sec}}{10000^2}$

$T(20000) = c\,20000^2 = \frac{10\text{sec}}{10000^2}\,20000^2 = 40\text{sec}.$