Lehman College, CUNY **CMP 230 Final Exam—SAMPLE** **Spring 2011**

1. What will the following code print? Show your work tracing the code.

```
one = True
two = False
list = [6,2,8,4,9]
for n in list:
    one = one and n > 3
    two = two or n < 8
print(one,two)
```

2. Write a *function* that takes a list of numbers as input and returns the standard deviation. The standard deviation of a list of numbers is defined to be:

$$\sqrt{\frac{(x_1 - a)^2 + \cdots + (x_n - a)^2}{n}}$$

where n is the number of items in the list and a is the mean (average) of the list. In words, the procedure for computing standard deviation of a list of numbers is this.

   (a) compute the average a of the list

   (b) sum the following: for each x in the list, $(x - a)^2$

   (c) divide by the length of the list

   (d) take the square root

3. Write a graphics *program* that draws three circles. The first circle is specified by the user using mouse clicks. The first click is the center of the circle, and the distance between the first click and the second is the radius. Then two more circles are drawn having the same radius as the first, *but with centers determined randomly.*

4. Write a *program* that prompts the user for the names of an input file and an output file. Assume the input file consists of words separated by spaces. The program creates an output file containing the same words, but leaving out those words that start with vowels.

5. Write a *function* that takes inputs a, b, and c, and returns the two solutions to the quadratic equation $ax^2 + bx + c = 0$. (Assume the equation has real solutions.)

6. Assume the summer solstice occurs on day 171 of the year. Write a *function* that takes a number, representing a day of the year, as input and then prints "'before", "after", or "is", depending on whether the day is before the solstice, after it, or is the solstice.

7. Write a *function* that takes as input a list of numbers and a single number, call it n. It then prints the members of the list out until it reaches the number n, and then stops. If n does not occur in the list, the entire list should be printed.

8. This problem involves creating a program when parts of it have already been written by someone else. Assume a file containing a list of numbers has already been created. You can also assume that a function called `readNums` has been written that takes a filename as its parameter and returns a list composed of the numbers in the file. `readNums` takes care of opening, reading, and closing the file, and parses the input into the appropriate list structure; you don't need to write the definition for this function. Additionally, `min` and `max` functions in the Python built-in library provide methods for determining the minimum and maximum values in a list.

Given all this, write a program that prompts the user for a file name, where the file contains a list of numbers. The program determines and prints the difference between the largest and the smallest number in the list. Your program's design must exhibit a reasonable modular decomposition with at least three levels of abstraction.

Useful String Methods: (from p 140 of the textbook)

Function	Meaning
`s.capitalize()`	Copy of `s` with only the first character capitalized.
`s.center(width)`	Copy of `s` is centered in a field of given width.
`s.count(sub)`	Count the number of occurrences of `sub` in `s`.
`s.find(sub)`	Find the first position where `sub` occurs in `s`.
`s.join(list)`	Concatenate `list` into a string using `s` as a separator.
`s.ljust(width)`	Like `center`, but `s` is left-justified.
`s.lower()`	Copy of `s` with all characters converted to lowercase.
`s.lstrip()`	Copy of `s` with leading whitespace removed.
`s.replace(oldsub,newsub)`	Replace all occurrences of `oldsub` in `s` with `newsub`.
`s.rfind(sub)`	Like `find`, but returns rightmost position.
`s.rjust(sub)`	Like `center`, but `s` is right-justified.
`s.rstrip()`	Copy of `s` with trailing whitespace removed.
`s.split()`	Split `s` into a list of substrings.
`s.title()`	Copy of `s` with first character of each word capitalized.
`s.upper()`	Copy of `s` with all characters converted to uppercase.

Graphics Reference: (from p 108-111 of the textbook)

GraphWin Objects
`GraphWin(title, width, height)`
`plot(x,y,color)`
`plotPixel(x,y,color)`
`setBackground(color)`
`close()`
`getMouse()`
`checkMouse()`
`setCoords(xll,yll,xur,yur)`

Graphics Objects
`setFill(color)`
`setOutline(color)`
`setWidth(pixels)`
`draw(aGraphWin)`
`undraw()`
`move(dx,dy)`
`clone()`

Text Methods
`Text(anchorPoint, string)`
`setText(string)`
`getText()`
`getAnchor()`
`setFace(family)`
`setSize(point)`
`setStyle(style)`
`setTextColor(color)`

Point Methods
`Point(x,y)`
`getX()`
`getY()`

Line Methods
`Line(point1, point2)`
`setArrow(string)`
`getCenter()`
`getP1(), getP2()`

Circle Methods
`Circle(centerPoint, radius)`
`getCenter()`
`getRadius()`
`getP1(), getP2()`

Rectangle Methods
`Rectangle(point1,point2)`
`getCenter()`
`getP1(), getP2()`

Oval Methods
`Oval(point1, point2)`
`getCenter()`
`getP1(), getP2()`

Polygon Methods
`Polygon(P1, P2, P3,...)`
`getPoints()`

1. What will the following code print? Show your work tracing the code.

```
one = True
two = False
list = [True, False, True, False]
for n in list:
    one = one and n
    two = two or n
print(one,two)
```

2. The *root mean square* of a list of numbers is computed as follows:

   (a) sum the squares of the members of the list
   (b) divide by the length of the list
   (c) take the square root of the result

   Write a *function* that takes a list of numbers as input and returns the root mean square of the list.

3. Write a graphics *program* that draws three circles. Each circle has a center that is chosen randomly. The radius for all three circles should be 1/4 the width of your graphics window.

4. Write a *program* that prompts the user for the names of an input file and an output file. Assume the input file consists of words separated by spaces. The program creates an output file containing the same words, but with all occurrences of your name replaced by your initials.

5. The quadratic equation $ax^2 + bx + c = 0$ has two distinct real solutions exactly when the *discriminant*, $b^2 - 4ac$, is positive. Write a *function* that takes a, b, and c as inputs, and returns true if there are two distinct real solutions, and false otherwise.

6. Write a function that takes an integer as input and then prints "'positive", "negative", or "zero", depending on whether the integer is above 0, below 0, or is zero.

7. Write a function that takes as input a list of numbers and prints out the numbers on the list until it encounters a negative number.

8. This problem involves creating a program when parts of it have already been written by someone else. Assume a file containing a list of numbers has already been created. You can also assume that a function called `readNums()` has been written that takes a filename as its parameter and returns a list composed of the numbers in the file. `readNums()` takes care of opening, reading, and closing the file, and parses the input into the appropriate list structure; you do not need to write the definition for this function. Additionally, the sorted function in the Python built-in library provides a method for sorting a list of numbers into ascending order.

   Recall that the median of a list of numbers is the value found at the center of the list when the list is sorted and the list has an odd number of entries.

   Given all this, write a program that prompts the user for a file name, where the file contains an odd number of numbers. The program determines and prints the median of the numbers in the list. Your program's design must exhibit a reasonable modular decomposition with at least three levels of abstraction.

   (a) Design the top level (main). At this first level of abstraction there should be no details pertaining to how the work is done, just function calls and assignments that tell us what will be done in terms of functions that read input, functions that do calculations, and functions that write output. Think of the racquetball program as a guide.

(b) Refine the top level by writing code with two more levels of abstraction. More specifically, write out one of the functions that main calls on, and write out one of the functions that is called by that function.

Useful String Methods: (from p 140 of textbook)

Function	Meaning
`s.capitalize()`	Copy of `s` with only the first character capitalized.
`s.center(width)`	Copy of `s` is centered in a field of given width.
`s.count(sub)`	Count the number of occurrences of `sub` in `s`.
`s.find(sub)`	Find the first position where `sub` occurs in `s`.
`s.join(list)`	Concatenate `list` into a string using `s` as a separator.
`s.ljust(width)`	Like `center`, but `s` is left-justified.
`s.lower()`	Copy of `s` with all characters converted to lowercase.
`s.lstrip()`	Copy of `s` with leading whitespace removed.
`s.replace(oldsub,newsub)`	Replace all occurrences of `oldsub` in `s` with `newsub`.
`s.rfind(sub)`	Like `find`, but returns rightmost position.
`s.rjust(sub)`	Like `center`, but `s` is right-justified.
`s.rstrip()`	Copy of `s` with trailing whitespace removed.
`s.split()`	Split `s` into a list of substrings.
`s.title()`	Copy of `s` with first character of each word capitalized.
`s.upper()`	Copy of `s` with all characters converted to uppercase.

Graphics Reference: (from p 108-111 of the textbook)

GraphWin Objects
`GraphWin(title, width, height)`
`plot(x,y,color)`
`plotPixel(x,y,color)`
`setBackground(color)`
`close()`
`getMouse()`
`checkMouse()`
`setCoords(xll,yll,xur,yur)`

Graphics Objects
`setFill(color)`
`setOutline(color)`
`setWidth(pixels)`
`draw(aGraphWin)`
`undraw()`
`move(dx,dy)`
`clone()`

Text Methods
`Text(anchorPoint, string)`
`setText(string)`
`getText()`
`getAnchor()`
`setFace(family)`
`setSize(point)`
`setStyle(style)`
`setTextColor(color)`

Point Methods
`Point(x,y)`
`getX()`
`getY()`

Line Methods
`Line(point1, point2)`
`setArrow(string)`
`getCenter()`
`getP1(), getP2()`

Circle Methods
`Circle(centerPoint, radius)`
`getCenter()`
`getRadius()`
`getP1(), getP2()`

Rectangle Methods
`Rectangle(point1,point2)`
`getCenter()`
`getP1(), getP2()`

Oval Methods
`Oval(point1, point2)`
`getCenter()`
`getP1(), getP2()`

Polygon Methods
`Polygon(P1, P2, P3,...)`
`getPoints()`

1. What will the following code print? Show your work tracing the code.

```
one = True
two = False
list = [False, True, True, False]
for n in list:
    one = one and not n
    two = two or n
print(one,two)
```

2. Write a *function* that takes a list of numbers as input and calculates the average deviation. The method for computing average deviation of a list of numbers is this.
   (a) compute the average a of the list
   (b) sum the following: for each x in the list, $x - a$
   (c) divide by the length of the list

3. Write a graphics *program* that draws three circles at positions indicated by user by mouse clicks. The radius of each circle should be chosen randomly, but should be less than 1/4 the width of your graphics window.

4. Write a *program* that prompts the user for the names of an input file and also for two words. Assume the input file consists of words separated by spaces, and the two words occur in the file. The program should print which of the two words occurs first in the file.

5. The quadratic equation $ax^2 + bx + c = 0$ has just one solution exactly when the *discriminant*, $b^2 - 4ac$, is zero. Write a *function* that takes a, b, and c as inputs, and returns true if there is just one real solution, and false otherwise.

6. Assume the average of a list of numbers is 35. Write a function that takes an integer as input and then prints "'above", "below", or "is", depending on whether the integer is above that average, below it, or equals it.

7. Write a function that takes as input a list of numbers and prints out the numbers on the list until it encounters -1.

8. You have two coins, but they are not fair—they do not come up heads with probability 0.5. You are to simulate flipping the two coins a large number of times and report in what proportion of the trials both come up heads. Your code should explain to the user what is going on; ask for the probability of heads on coin 1, and on coin 2, and the number of trials to make; simulate the trials; and report the results. You are not to write the entire program, instead, do the following:
   (a) Design the top level (main). At this first level of abstraction there should be no details pertaining to how the work is done, just function calls and assignments that tell us what will be done in terms of functions that read input, functions that do calculations, and functions that write output. Think of the racquetball program as a guide.
   (b) Refine the top level by writing code with two more levels of abstraction. More specifically, write out one of the functions that main calls on, and write out one of the functions that is called by that function.

Useful String Methods: (from p 140 of textbook)

Function	Meaning
`s.capitalize()`	Copy of `s` with only the first character capitalized.
`s.center(width)`	Copy of `s` is centered in a field of given width.
`s.count(sub)`	Count the number of occurrences of `sub` in `s`.
`s.find(sub)`	Find the first position where `sub` occurs in `s`.
`s.join(list)`	Concatenate `list` into a string using `s` as a separator.
`s.ljust(width)`	Like `center`, but `s` is left-justified.
`s.lower()`	Copy of `s` with all characters converted to lowercase.
`s.lstrip()`	Copy of `s` with leading whitespace removed.
`s.replace(oldsub,newsub)`	Replace all occurrences of `oldsub` in `s` with `newsub`.
`s.rfind(sub)`	Like `find`, but returns rightmost position.
`s.rjust(sub)`	Like `center`, but `s` is right-justified.
`s.rstrip()`	Copy of `s` with trailing whitespace removed.
`s.split()`	Split `s` into a list of substrings.
`s.title()`	Copy of `s` with first character of each word capitalized.
`s.upper()`	Copy of `s` with all characters converted to uppercase.

Graphics Reference: (from p 108-111 of the textbook)

GraphWin Objects
`GraphWin(title, width, height)`
`plot(x,y,color)`
`plotPixel(x,y,color)`
`setBackground(color)`
`close()`
`getMouse()`
`checkMouse()`
`setCoords(xll,yll,xur,yur)`

Graphics Objects
`setFill(color)`
`setOutline(color)`
`setWidth(pixels)`
`draw(aGraphWin)`
`undraw()`
`move(dx,dy)`
`clone()`

Text Methods
`Text(anchorPoint, string)`
`setText(string)`
`getText()`
`getAnchor()`
`setFace(family)`
`setSize(point)`
`setStyle(style)`
`setTextColor(color)`

Point Methods
`Point(x,y)`
`getX()`
`getY()`

Line Methods
`Line(point1, point2)`
`setArrow(string)`
`getCenter()`
`getP1(), getP2()`

Circle Methods
`Circle(centerPoint, radius)`
`getCenter()`
`getRadius()`
`getP1(), getP2()`

Rectangle Methods
`Rectangle(point1,point2)`
`getCenter()`
`getP1(), getP2()`

Oval Methods
`Oval(point1, point2)`
`getCenter()`
`getP1(), getP2()`

Polygon Methods
`Polygon(P1, P2, P3,...)`
`getPoints()`

1. What will the following code print? Show your work tracing the code.

```
one = True
two = False
list = [True, True, False, True]
for n in list:
    one = one and n
    two = two or not n
print(one,two)
```

2. Write a *function* that takes a list of numbers as input returns three items: the average of the positive numbers in the list; the average of the negative numbers in the list; and the average of all the numbers in the list.

3. Write a graphics *program* that draws three circles. The first circle is specified by the user with two mouse clicks. The first click determines the center, and the distance between the two clicks is the radius. Then two more circles are drawn, with the same radius, but with centers determined randomly. The centers of these circles should be in the graphics window.

4. Write a *program* that prompts the user for the names of an input file and an output file. Assume the input file consists of words separated by spaces. The program creates an output file containing the same words, but with all occurrences of the word "one" replaced with occurrences of the word "two".

5. The quadratic equation $ax^2+bx+c=0$ has complex solutions exactly when the *discriminant*, b^2-4ac, is negative. Write a *function* that takes a, b, and c as inputs, and returns true if the equation has complex solutions, and false otherwise.

6. Assume time is represented with a 24 hour clock, and only hours are being used, no minutes. Write a function that takes an hour as input and then prints "am", "pm", or "noon", depending on what part of the day the input falls into.

7. Write a function that takes as input a list of strings and prints out the strings on the list until it encounters the empty string.

8. You have two coins: one is fair (it comes up heads with probability 0.5) and one is unfair (it comes up heads with a different probability). You are to simulate flipping the two coins a large number of times and report in what proportion of the trials both come up heads. Your code should explain to the user what is going on; ask for the probability of heads on the unfair coin, and the number of trials to make; simulate the trials; and report the results. You are not to write the entire program, instead, do the following:

   (a) Design the top level (main). At this first level of abstraction there should be no details pertaining to how the work is done, just function calls and assignments that tell us what will be done in terms of functions that read input, functions that do calculations, and functions that write output. Think of the racquetball program as a guide.

   (b) Refine the top level by writing code with two more levels of abstraction. More specifically, write out one of the functions that main calls on, and write out one of the functions that is called by that function.

Useful String Methods: (from p 140 of textbook)

Function	Meaning
`s.capitalize()`	Copy of `s` with only the first character capitalized.
`s.center(width)`	Copy of `s` is centered in a field of given width.
`s.count(sub)`	Count the number of occurrences of `sub` in `s`.
`s.find(sub)`	Find the first position where `sub` occurs in `s`.
`s.join(list)`	Concatenate `list` into a string using `s` as a separator.
`s.ljust(width)`	Like `center`, but `s` is left-justified.
`s.lower()`	Copy of `s` with all characters converted to lowercase.
`s.lstrip()`	Copy of `s` with leading whitespace removed.
`s.replace(oldsub,newsub)`	Replace all occurrences of `oldsub` in `s` with `newsub`.
`s.rfind(sub)`	Like `find`, but returns rightmost position.
`s.rjust(sub)`	Like `center`, but `s` is right-justified.
`s.rstrip()`	Copy of `s` with trailing whitespace removed.
`s.split()`	Split `s` into a list of substrings.
`s.title()`	Copy of `s` with first character of each word capitalized.
`s.upper()`	Copy of `s` with all characters converted to uppercase.

Graphics Reference: (from p 108-111 of the textbook)

GraphWin Objects
`GraphWin(title, width, height)`
`plot(x,y,color)`
`plotPixel(x,y,color)`
`setBackground(color)`
`close()`
`getMouse()`
`checkMouse()`
`setCoords(xll,yll,xur,yur)`

Graphics Objects
`setFill(color)`
`setOutline(color)`
`setWidth(pixels)`
`draw(aGraphWin)`
`undraw()`
`move(dx,dy)`
`clone()`

Text Methods
`Text(anchorPoint, string)`
`setText(string)`
`getText()`
`getAnchor()`
`setFace(family)`
`setSize(point)`
`setStyle(style)`
`setTextColor(color)`

Point Methods
`Point(x,y)`
`getX()`
`getY()`

Line Methods
`Line(point1, point2)`
`setArrow(string)`
`getCenter()`
`getP1(), getP2()`

Circle Methods
`Circle(centerPoint, radius)`
`getCenter()`
`getRadius()`
`getP1(), getP2()`

Rectangle Methods
`Rectangle(point1,point2)`
`getCenter()`
`getP1(), getP2()`

Oval Methods
`Oval(point1, point2)`
`getCenter()`
`getP1(), getP2()`

Polygon Methods
`Polygon(P1, P2, P3,...)`
`getPoints()`