Condensed for review purposes Spring 2014

CS101-Ames Final Exam Name _____
12/18/2013 Section (circle one): NOON 1PM

Part I. Function writing. [5 points each]

1. Write a recursive function that returns the number 1 followed by N zero's (return a number, not a string). For example, `tens(3)` should be 1000. `tens(4)` is 10000, `tens(1)` is 10, `tens(0)` is 1.

2. Write a function with one argument called reverseDigits. Assume that the argument is an integer between 0 and 999 inclusive. The function should return an integer whose digits are the reverse of the three digits of the argument.
For example:
reverseDigits(134) should return 431
reverseDigits(627) should return 726
reverseDigits(500) should return 5
reverseDigits(23) should return 320
reverseDigits(7) should return 700

3. Write a recursive function similar to the one above that will reverse the digits of any positive integer, with any number of digits. For example, reverse(23789) should return 98732. Do the calculation using only integers, not strings. (Don't spend forever on this one, it's difficult but not a lot of points.)

4. Write a Python fragment (need not be a function) that calculates and prints $\sum_{n=1}^{100} \frac{1}{2^n}$

Part II. Reading. For each fragment below, show the expected output in the box to its right. None contain errors. [5 points each]

`k = 22`
`while k>0:`
`  k = k / 2`
`  print k,`	

`def isP2(n):`
`  if n==0:`
`  return False`
`  if n==1:`
`  return True`
`  return n/2==n/2.0 and isP2(n/2)`
`for k in range(64):`
`  if isP2(k):`
`  print k,`
	`def f(s):`
	`  if len(s)==0:`
	`  return ""`
	`  return f(s[1:]) + s[0]`
	
	`print f("Hello world")`

Part III. Writing. [10 points each]

1. In HW5, you wrote an encryption program that encrypted or decrypted messages based on a key; this is called a “Viginere” cipher.

For this problem, write a different encryption function based on a simpler technique, known as a Caesar cipher.

Your function will be passed a string, and an integer N. To encrypt each letter, simply add N to the numeric value of that letter, and convert the sum back to a letter. BUT: if the sum exceeds 126, the number needs to be wrapped back to 32 or more. For example, a sum of 130 should wrap to 35, the character #.

The table on the right is for reference purposes only, it should not appear in your function. Also: recall that the ord function converts a letter to a number, and chr converts a number to a letter.

Your function should return the encrypted message.

You may assume that N is between 1 and 50.

```
def encrypt(message, n):
```

32	(space)
33	!
34	"
35	#
36	$
37	%
38	&
39	'
40	(
41	)
42	*
43	+
44	,
45	-
46	.
47	/
48	0
49	1
50	2
51	3
52	4
53	5
54	6
55	7
56	8
57	9
58	:
59	;
60	<
61	=
62	>
63	?
64	@
65	A
...	...
90	Z
91	[
92	\
93	]
94	^
95	_
96	`
97	a
...	...
122	z
123	{
124	|
125	}
126	~

2.As a reminder, here's the quiet() function from the audio processing homework:

```
def quiet():
    left  = BCAudio.getLeft()
    right = BCAudio.getRight()
    for i in range(len(left)):
        left[i]  = left[i]/2.
        right[i] = right[i]/2.
```

Some music contains one or more silent sections. If a section is silent, the left and right lists will contain zero's for the duration of that section. For example, here is the beginning of a song:

i	0	1	2	3	4	5	6	7	8	9	10	11
Left[i]	0.69	-0.07	-0.55	0	0	0.56	0.25	0.57	-0.44	0	0	0.49
Right[i]	-1	0.05	0.84	0	0	0	-0.84	0.19	-0.5	0	0	0.73

In this case, elements 3 and 4 should be removed (but not element 5), and elements 9 and 10 should be removed. (When finished, Left and Right will be 4 elements shorter.)

Write a new function called removeSilence(). This function should remove all sections where both Left[i] and Right[i] are both zero, leaving everything else unchanged. In the example above, after removing silence the lists should look like this:

i	0	1	2	3	4	5	6	7
Left[i]	0.69	-0.07	-0.55	0.56	0.25	0.57	-0.44	0.49
Right[i]	-1	0.05	0.84	0	-0.84	0.19	-0.5	0.73

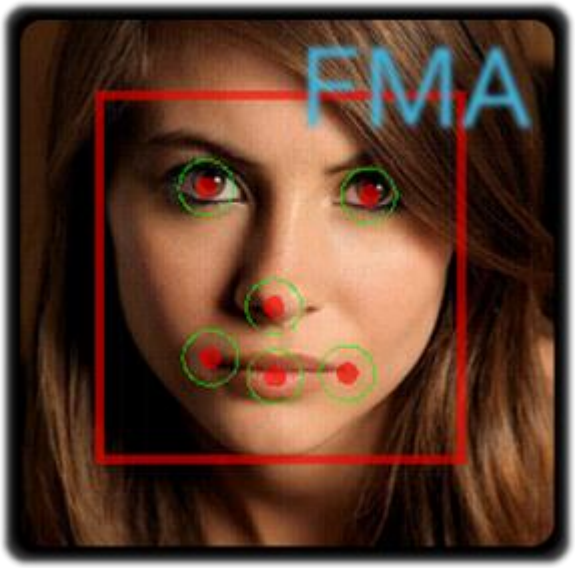
Part IV. Writing a class. [15 points]

Write a class that will compare two photographs of faces, and try to determine whether or not the photographs are of the person.

To do this, we'll compare facial measurements. We'll use the Point class (with x and y attributes) to represent the location of the eyes, corners of the mouth, and center of the mouth in a photograph.

Since the pictures might be different sizes, we can't simply compare something like the distance between the eyes directly. Instead, we'll compare ratios. The ratio of the distance between the eyes to the mouth width will be one metric. The second will be the "mouth asymmetry": the ratio of the width of the left side of the mouth to the width of the right side of the mouth.

These ratios don't change with the overall size of the picture, or even if the head is turned slightly to one side.



Although this picture shows a point on the nose, we won't use that point.

So the metrics we're interested in for each face are:

1. The "eyeToMouthRatio": the distance between the eyes divided by the width of the (entire) mouth.
2. The "mouthAsymetry", defined as the distance from the center of the mouth to the left side, divided by the distance from the center of the mouth to the right side.

Write a class called FaceMetrics. The main program could create an instance of the class like this:

```
face1 = FaceMetrics(leftEye, rightEye, mouthLeft, mouthRight, mouthCenter)
```

Each of the arguments shown above will be a Point, and so each will have an **x** and a **y** attribute.

The main the program could create many faces, each an instance of the FaceMetrics class. Then, two of the faces could be compared like this:

```
correlation = face1.compareTo(face2)
```

The `compareTo` function should find the absolute value of the difference between the mouthAsymmetry's, and the absolute value of the difference between the eyeToMouthRatio's. Return the average of these differences. So, 0 would indicate an exact match, larger numbers would represent lesser match confidence.

Note: you can find the distance between two points using the Pythagorean Theorem:

$$dist = \sqrt{(p1.x - p2.x)^2 + (p1.y - p2.y)^2}$$

[Of course, a real face recognition program would use many more metrics than just these two. We're using a simplified version for this exam.]

Write your answer on the back of this sheet.