Net ID:___________________________ **Name:**____

x

CSCI-UA.0380-004 – Midterm Exam

March 9th, 2016

Instructor: Joseph Versoza

Ask the person to your left for their first name (leave blank if next to empty seat or wall):

____

Ask the person to your right for their first name (leave blank if next to empty seat or wall):

____

Keep this test booklet closed until the class is prompted to begin the exam

- Computers, calculators, phones, textbooks or notebooks are **not allowed** during the exam
- Please turn off your phone to avoid disrupting others during the exam

Instructions

- A reference can be found in the last page.
- The last page can also be used as scrap paper.

Choose 2 of the 1st 3 questions (#'s 1 through 3) to complete. **Each question is worth 16 points**.

Use the remaining question as extra credit (worth 1/3 of the points earned). **Specify which question is extra credit by writing EXTRA CREDIT** in caps above the question. If you do not specify extra credit, the last question (#3) will be counted as extra credit. **Only do the extra credit if you're finished with the remainder of the exam!**

1. Read the code in the right column. Answer questions about it in the left column.

```
def f(nums):
    d = {}
    for n in nums:
        try:
            d[n] += 1
        except KeyError:
            d[n] = 0
    return max(d, key=d.get)

print(f([4, 3, 5, 3, 3, 4, 1]))
```

What's the output of the program on the left? If it's an error, describe the error that occurs. Show your work for partial credit.

3

```
def call_it(func, n):
    def new_func(x):
        res = x
        for i in range(n):
            res = func(res)
        return(res)
    return new_func

def double(num):
    return num * 2

my_func = call_it(double, 2)
print(my_func(17))
```

What is the output of the program on the left? If it's an error, describe the error that occurs. Show your work for partial credit.

68

```
animals = ['ant', 'bat', 'cat', 'dog']
new_animals = []
for animal in animals:
    if animal[-2] in 'on':
        s = animal.upper()
        new_animals.append('{}?'.format(s))
print(new_animals)
```

Create a **list comprehension** that works on the list, `animals`, so that it produces the same list as the code on the right. Use the space immediately below both of these columns to write your list comprehension:

```
new_animals = ['{}?'.format(a.upper()) for a in animals if a[-2] in 'on']
```

```
class Person:

    # missing method!

    def __str__(self):
        return self.first + ' ' + self.last

p = Person('Guido', 'Van Rossum')
print(p) # prints out Guido Van Rossum
```

The program on the left outputs '`Guido Van Rossum`'. It's missing a special method. Define the method below.

```
def __init__(self, first, last):
    self.first = first
    self.last = last
```

2. **Part 1** – Write short, concise, one or two sentence answers to the following questions:

a) In the context of Python, describe what a module is.

A python module is file containing Python code, such variable, function and class definitions. Modules can be *brought* in to another file using the import statement.

b) To work with csvs , it's actually better to use Python's built-in csv module rather than simply splitting by comma. Why is splitting by comma not always completely adequate for extracting values from a csv?

The delimiter, comma, may appear as a valid part of a value. For example, in 5, (4,2), 3 … the 2nd element may be (4, 2)!

c) Besides csvs (and other character delimited formats), name and briefly describe two other text-based data formats.

JSON – like javascript object literals, XML – tags that describe document, like html, fixed width – each column is some number of characters.

d) What module is used for asking a web site for data? Besides turtle, what module can be used for creating graphs and plotting points?

requests, matplotlib

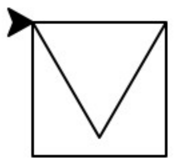
Part 2 – The program on the left creates a 7 pixel by 7 pixel black-and-white image. Draw the resulting image in the grid on the right. Each box represents a pixel. **Write X in the box for a black** pixel or **leave the box blank for a white** pixel.

```
from PIL import Image
img1 = Image.new('RGB', (7, 7), (255,) * 3)
pixels1 = img1.load()
w, h = img1.size
for x in range(w):
    for y in range(h):
        if y >= h // 2 and x == y:
            pixels1[x, y] = (0, 0, 0)
        if y < h // 2 and x == h // 2:
            pixels1[x, y] = (0, 0, 0)
        if x > w // 2 and y == h // 2:
            pixels1[x, y] = (0, 0, 0)
```

			X			
			X			
			X			
			X	X	X	X
				X		
					X	
						X

Part 3 – Read the program on the left. Draw its output in the space on the right. The exact coordinates and exact pixel size doesn't matter – **just a general idea of what the output drawing will be is adequate**. Assume that the turtle starts of facing right.

```
import turtle
t = turtle.Turtle()
wn = turtle.Screen()

for i in range(4):
    t.forward(50)
    t.right(90)

for i in range(3):
    t.forward(50)
    t.right(120)

wn.mainloop()
```



3. Read the program in the left column. Write the output of the program in the right column.

```
def chomp(my_list, num):
    for i in range(num):
        my_list.pop()

nonsense = ['beep', 'bop', 'bzz', 'boo']
print(nonsense)
res = chomp(nonsense, 2)
print(nonsense)
print(res)
```

What's the output of the program on the left? If it's an error, describe the error that occurs. Show your work for partial credit.

```
['beep', 'bop', 'bzz', 'boo']
['beep', 'bop']
None
```

```
my_points = [(0, 0), (40, 32), (9, 12)]
def move_points(points):
    for p in points:
        p[0] = p[0] + 1
        p[1] = p[1] + 1

move_points(my_points)
print(my_points)
```

What's the output of the program on the left? If it's an error, describe the error that occurs. Show your work for partial credit.

Error – tuples are immutable

```
from collections import namedtuple
Cat = namedtuple('Cat', ['name', 'lives'])
c = Cat(name='Paw Newman', lives=9)
print(c[1])
```

What's the output of the program on the left? If it's an error, describe the error that occurs. Show your work for partial credit.

```
9
```

```
inventory = {'grapes': 3, 'oranges': 8}
item = input('Enter the name of an item')

try:
    qty = inventory[item]
    msg = "There are " + qty + " " + item
    print(msg)
except KeyError:
    print('That item does not exist')
except TypeError:
    print('Uh oh, something went wrong')
```

What is the output if the user types in `'apples'`?

```
That item does not exist
```

What is the output if the user types in `'grapes'`?

```
Uh oh, something went wrong
```

```
address = {'city':'Buffalo','state':'NY'}
food = ['butter', 'mayo', 'cheese']

for enigma, riddle in enumerate(food):
    print(enigma, riddle)

for mystery in address:
    print(mystery)
```

What's the output of the program on the left? If it's an error, describe the error that occurs. Show your work for partial credit.

```
0 butter
1 mayo
2 cheese
city
state
```

Choose 4 of the last 5 questions (#'s 4 through 8) to complete. **Specify which question you are skipping by writing DO NOT GRADE** in caps above the question. If you do not, I will count the last question as the question you are skipping. **Each question is worth 17 points**.

4. The **data** below **represents players** and a **list of words** they played in a **scrabble-like word** game. Each word is worth a number of points: the first 3 letters are worth 1 point, and every letter after the first 3 is worth an additional point. The word 'wig' is worth 1 point, while 'wiggled' is worth 5 (1 for the first 3 letters... plus 4 for the next 4). Words that are less than 3 letters are worth 1 point.

   Using the data below, define a function, **`get_winner`** that **returns a 2-tuple representing** the **player** with the **highest number of points**; it should contain their **first** and **last name** as the first element and **their score** as the second. Here's the data:

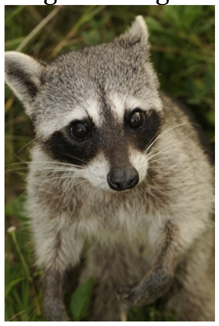
```
game_data = [
    {'first':'Alice', 'last':'Aslan', 'words':['badger', 'hack', 'crew', 'orchard']},
    {'first':'Bob', 'last':'Berman', 'words':['wig', 'hazy', 'duck', 'rude']},
    {'first':'Carol', 'last':'Cruz', 'words':['gargantuan', 'ordeal', 'hop', 'complete']},
    {'first':'Dan', 'last':'Dewan', 'words':['anxiety', 'freckled', 'dolphins', 'event']},
]
```

   (a) you can create as many helper functions as you need
   (b) **avoid using regular for loops to do this** (however, a **`for` in a list comprehension is ok**)
   (c) there is a minor point deduction (-1) for every *regular* for loop that is used
   (d) example usage: `get_winner(game_data) # ---> returns ('Dan Dewan', 20)`

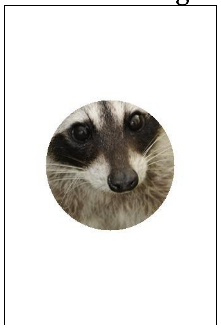
```
def make_name(d):
    return "{} {}".format(d['first'], d['last'])

def score_word(w):
    return 1 if len(w) <= 3 else 1 + len(w[3:])

def get_score(d):
    return sum([score_word(w) for w in d['words']])

def transform(d):
    return {'name': make_name(d), 'score':get_score(d)}

def get_winner(data):
    max_d = max([transform(d) for d in data], key=lambda d: d['score'])
    return max_d['name'], max_d['score']

print(get_winner(game_data))
```

5. Create two functions, **`pixel_mapper`** and **`circle_it`**, to change an image so that a **circular portion of the image is retained** in the center of the image, while the rest of the image is white.

Original Image



Processed Image



The first function to implement is **`pixel_mapper`**. It takes **2 arguments**, an **image** and a **function**... and applies the function to all pixels in the image passed in.

The function passed in **differs from the implementation of `pixel_mapper` in the slides and the book** in that the function passed to this version should have 3 parameters – a 3-tuple representing the color of a pixel, a 2-tuple representing the coordinates of the pixel, and a 2-tuple representing the width and height of the entire image.

The second **function, `circle_it`, will be the function that you pass to `pixel_mapper`** so that it will be used to transforms every pixel. When passed in as the 2nd argument to pixel_mapper, it will create the image shown above. It has **3 parameters**: a 3-tuple representing color, a 2-tuple representing coordinates of the pixel, and 2-tuple representing the width and height of the entire image.

It should return a 3-tuple representing the color of the pixel that the current pixel should be transformed to. The circle that results from the transformation should have a radius of 100px. Use the distance formula to help with implementation: $d=\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$

Some setup code has been provided for you. Fill in the missing blank lines for `pixel_mapper`. Implement `circle_it`. You can write additional helper function(s) if necessary.

```
def pixel_mapper(img, f):
    pixels = img.load()

     ______________________________________ # FILL THIS IN

         _______________________________________ # FILL THIS IN

            __________________________________________________# FILL THIS IN

# IMPLEMENT circle_it...
```

```
def pixel_mapper(img, f):
    pixels = img.load()
    for x in range(img.size[0]):
        for y in range(img.size[1]):
            pixels[x, y] = f(pixels[x, y], (x, y), img.size)

def dist(x1, y1, x2, y2):
    return math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)

def circle_it(color, point, size):
    center = size[0] // 2, size[1] // 2
    if dist(point[0], point[1], center[0], center[1]) > 100:
        return (255, 255, 255)
    else:
        return color
```

6. Create a **function** that implements a **Caesar cipher**. A Caesar cipher is a type of **substitution cipher** in which each letter in the plain text is replaced by a letter that's a constant number of positions away from it (*shifted*) in the alphabet. For example, a Caesar cipher with a shift of 2 would map A's to C's, B's to D's, and so forth until we get to Z's, which loop around and would map to B's. Here's what the substitution would look like with a shift of 2 and with a shift of -2:

```
original letters:   abcdefghijklmnopqrstuvwxyz   ...the plain text 'baz'
shifted 2 letters:  cdefghijklmnopqrstuvwxyzab   ...shifted by 2 is 'dcb'
shifted -2 letters: yzabcdefghijklmnopqrstuvwx   ...shifted by -2 is 'zyx'
```

Your function, caesar_encrypt, should have **2 arguments**: `msg`, the string to encrypt, and `shift`, the number of positions to shift every letter. Assume that `msg` will always be a string and that `shift` will always be an int (positive or negative). Here are some example function calls:

```
print(caesar_encrypt('baz', 0))  # prints out baz
print(caesar_encrypt('baz', 2))  # prints out dcb
print(caesar_encrypt('baz', -2)) # prints out zyx
```

```
def caesar(s, shift):
    alpha = gen_alpha()
    msg = ''
    for letter in s:
        if letter.isalpha():
            i = (alpha.index(letter) + shift)
            if i >= 0:
                i = i % 26
            else:
                i = -(abs(i) % 26)
            msg += alpha[i]
            # or no check, just: i = (alpha.index(letter) + shift) % len(alpha)
        else:
            msg += letter
    return msg
```

7. Create a **turtle animation** of a **circle moving back and forth** horizontally across a window . The setup code is given for you. Your animation should meet the following requirements:

   (a) the radius of the animated circle should be 10
   (b) the circle should start at the center of the window (assume that the center is at 0,0 with x increasing to right and y increasing to the top)
   (c) you can set the *animation speed* to `50` ms
   (d) when your circle hits the left or right edge of the window, it should move in the opposite direction
   (e) you don't have to worry about calling `wn.update()` (assume that it will be done for you)
   (f) setup code is given; your code will go where the comment says `### YOUR CODE HERE ###`

```
import turtle
t = turtle.Turtle()
wn = turtle.Screen()
WIDTH, HEIGHT = 500, 500
wn.setup(WIDTH, HEIGHT)
wn.tracer(0)
t.hideturtle()

### YOUR CODE HERE ###

wn.mainloop()
```

```
x, y, v = 0, 0, 5

def draw_circle():
    global x, y, v
    t.clear()
    t.up()
    t.goto(x, y)
    t.down()
    t.circle(10)
    x += v
    if x > WIDTH/2 or x < -WIDTH/2:
        v *= -1
    wn.update()
    wn.ontimer(draw_circle, 20)

draw_circle()

wn.mainloop()
```

8. Create a function, **`get_word_freq`**, that **opens a file** and **returns a dictionary** containing the number of times every word in the file occurs. For example, if the file, `anthem.txt` contains:

```
I do not surrender my treasures, nor do I share them. The fortune of my
spirit is not to be blown into coins of brass and flung to the winds as
alms for the poor of the spirit. I guard my treasures: my thought, my
will, my freedom. And the greatest of these is freedom.
```

Calling `get_word_freq('anthem.txt')` would return `{'my':6, 'winds':1, … 'to':2}`. This is because `'my'`, appears 6 times in the text, `'to'` appears twice (the ellipses represent additional words as keys and frequencies as values), etc. ...Note that:

(a) `get_word_freq` has **a single argument**, a string that's the **name** of the **file** to open and read
(b) As it reads the file, it treats the **space character** as a **word separator**
(c) All words are dealt with as lowercase – `'And'` is counted as `'and'`
(d) Only letter characters are considered part of a word – `'treasures:'` is counted as `'treasures'`
(e) You can create helper function(s) if required for your implementation

```
def clean_word(word):
    clean = ''
    for ch in word.lower():
        if ch.isalpha():
            clean += ch
    return clean

def get_word_freq(fn):
    with open(fn, 'r') as f:
        d = {}
        for line in f:
            parts = line.split(' ')
            print(parts)
            for word in parts:
                w = clean_word(word)
                d[w] = d.get(w, 0) + 1
    return d
```

Net ID:____ **Name:**____

Reference Material / Scrap Paper

Built-in	**String Methods**	**Turtle and Screen**	**Math Module**	**PIL**	**List Methods**	**File Object Methods**
abs	capitalize	Turtle Object	acos	Image Module	append	read
bool	count		acosh		count	readline
chr	endswith	back(distance)	asin	Image.new(...)	extend	readlines
dict	find	begin_fill()	asinh	Image.open(...)	index	write
enumerat	format	circle(radius)	atan		insert	
e filter	index	clear()	atan2	Image Object	pop	
float	isalnum	color(color)	atanh		remove	
format	isalpha	color(colorstring)	ceil	img.size	reverse	
input	isdecimal	down()	cos	img.load()	sort	
int	isdigit	end_fill()	cosh			
len	islower	forward(distance)	degrees	PixelAccess		
list	isnumeric	goto(x, y)	floor	Object		
map	isprintable	hideturtle()	log			
max	isspace	left(angle)	log10	(like a dict)	**Dictionary Methods**	**Random Module Functions**
min	istitle	pensize(size)	log2	get using []'s		
open	isupper	right(angle)	pi	set using []'s		
ord	join	setheading(angle)	pow		Get	randint
pow	lower	up()	radians		items	choice
print	replace		sin		keys	shuffle
range	split	Screen Object	sinh		pop	
round	startswith		sqrt		popitem	
set	strip	bgcolor(colorstring)	tan		update	
sorted	title	listen()	tanh		values	
str	upper	onkeypress(func, key				
sum		ontimer(func, time_ms)				
		setup(width, height)				
		tracer(0)				
		update()				

ASCII Chart

```
Char Dec  | Char Dec | Char Dec | Char Dec
 -----------------------------------------
 (nul)  0  | (sp) 32  | @    64  | `     96
 (soh)  1  | !    33  | A    65  | a     97
 (stx)  2  | "    34  | B    66  | b     98
 (etx)  3  | #    35  | C    67  | c     99
 (eot)  4  | $    36  | D    68  | d    100
 (enq)  5  | %    37  | E    69  | e    101
 (ack)  6  | &    38  | F    70  | f    102
 (bel)  7  | '    39  | G    71  | g    103
 (bs)   8  | (    40  | H    72  | h    104
 (ht)   9  | )    41  | I    73  | i    105
 (nl)  10  | *    42  | J    74  | j    106
 (vt)  11  | +    43  | K    75  | k    107
 (np)  12  | ,    44  | L    76  | l    108
 (cr)  13  | -    45  | M    77  | m    109
 (so)  14  | .    46  | N    78  | n    110
 (si)  15  | /    47  | O    79  | o    111
 (dle) 16  | 0    48  | P    80  | p    112
 (dc1) 17  | 1    49  | Q    81  | q    113
 (dc2) 18  | 2    50  | R    82  | r    114
 (dc3) 19  | 3    51  | S    83  | s    115
 (dc4) 20  | 4    52  | T    84  | t    116
 (nak) 21  | 5    53  | U    85  | u    117
 (syn) 22  | 6    54  | V    86  | v    118
 (etb) 23  | 7    55  | W    87  | w    119
 (can) 24  | 8    56  | X    88  | x    120
 (em)  25  | 9    57  | Y    89  | y    121
 (sub) 26  | :    58  | Z    90  | z    122
 (esc) 27  | ;    59  | [    91  | {    123
 (fs)  28  | <    60  | \    92  | |    124
 (gs)  29  | =    61  | ]    93  | }    125
 (rs)  30  | >    62  | ^    94  | ~    126
 (us)  31  | ?    63  | _    95  | (del)127
```