Name:______________________________________ **NetID:**_______________🌲🌲🌲

CSCI-UA.0002-0010 – Midterm Exam #2

November 18th, 2015

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, and notebooks are **not allowed** during the exam
- Please turn off your phone to avoid disrupting others during the exam
- The back of this cover sheet can be used as scratch paper

1. Circle True or False (3 points)

a) `(True / `[✓ `False`]`)` `["yy", "az", "xx"] < ["yy", "az", "xx"]`

b) `(`[✓ `True`]` / False)` `chr(ord('Z') – 2) == 'X'`

c) `(True / `[✓ `False`]`)` **`pop`** is the **only list method returns a value**

d) `(True / `[✓ `False`]`)` A **string** is an **ordered sequence of elements of any type**

e) `(`[✓ `True`]` / False)` `'Passaic' not in ['Queens', 'Kings', 'Richmond', 'New York', 'Bronx']`

f) `(`[✓ `True`]` / False)` `[1, 2, 3] + [4, 5] == [1, 2, 3, 4, 5]`

2. You have a list of your favorite cat names: `cat_names = ['Yo Yo Meow', 'Paw Newman', 'Katy Purry']`

Using the variable, `cat_names`, write out **three** ways to **get rid of the last element in the list.** You must use **three different list methods or operators** to do this. **Reassignment and slicing are not allowed)**. (3 points)

`cat_names.pop()`   **`cat_names.remove('Katy Purry')`**   **`del cat_names[-1]`**

3. **Read the code sample** in the first column. **Answer the question** in the second column. (14 points)

Code | **Question**

```
noise = 'bop'
def make_electronic_music(sound):
    noise = 'bzzzz'
    print('{}... {}'.format(noise, sound))

result = make_electronic_music('beep')
print(result)
print(noise)
```

What is the output (shown on screen) of this code? (3 points)

```
bzzzz... beep

None

bop
```

```
def find(needle, haystack):
    for item in haystack:
        if item == needle:
            return True
        else:
            return False

print(find(4, [4, 3, 2, 1]))
```

(2 points)

a) **How many times will the body of the for loop in the function run?**

Once

b) There 's a logical error in this program; it's supposed to return True if the needle exists in the list, haystack... and False otherwise. **Find a pair of arguments that reveal this error.**

4, and [3, 4, 2, 1]

```
def gimme(x):
    return x
    print('here')
print(gimme(5))
```

(2 points)

a) What is the output (shown on screen) of this code?

`5`

b) Explain why.

`the second line in the function is never reached because return stops the function`

```
def sum_two_numbers(num_one, num_two):
    x = num_one + num_two
    return x


num_one = 24
print(sum_two_numbers(12, num_one))
```

(2 points)

a) What is the output of this program? **`36`**

b) Draw an arrow pointing to the argument(s)

c) Draw an arrow pointing to the parameter(s)

```
words = ["bravo", "golf"]
result = words.append("echo")
print(result)
print(words)
words.extend("lima")
words.pop()
popped = 2 * words.pop()
print(popped)
print(words)
print(words.index("golf"))
```

What is the output (shown on screen) of this code? (5 points)

```
None
['bravo', 'golf', 'echo']
mm
['bravo', 'golf', 'echo', 'l', 'i']
1
```

4. What is the output of the following program? Use the grid to the right of the program as a guide; **each individual character of output can be placed in a single box**. You do not have to use all of the boxes. (4 points)

```
def make_pattern(start, end):
    pattern = ''
    for i in range(start, end):
        row = ''
        for j in range(start, end):
            if j >= i:
                row += str(j)
            else:
                row += "0"
        pattern += '%s\n' % (row)
    return pattern.strip()

print(make_pattern(1, 5))
```

1	**2**	**3**	**4**				
0	**2**	**3**	**4**				
0	**0**	**3**	**4**				
0	**0**	**0**	**4**				

5. Using the following variable declaration, write out the output (error is possible) of the print statements below. (5 points)

`animals = [['dog', 'cat'], ['bat'], ['cow', 'bee', 'ant']]`

a) `print(animals[:2])`	(a)**`[['dog', 'cat'], ['bat']]`**
b) `print(animals[len(animals)])`	(b)**`error`**
c) `print(animals[-1][1:99])`	(c)**`['bee', 'ant']`**
d) `print(animals[0])`	(d)**`['dog', 'cat']`**
e) `print("animals[-1]")`	(e)**`animals[-1]`**

6. Create a function called **`uppercase_first`**. It should take two arguments: (6 points)

   a) A **string** that specifies what character will separate words (for example, a hyphen: '-')
   b) Another **string** composed of words **separated by the character specified** (for example: '`word1-word2-word3`')
   c) It will **give back the same string**, but with the **first word uppercased** (for example: '`WORD1-word2-word3`')
   d) If there is only one word, then the entire word should be uppercase
   e) An empty string gives back an empty string

   Example output:

```
>>> print(uppercase_first('-', 'cabeza-mano-pie'))
CABEZA-mano-pie
>>> print(uppercase_first(',', 'cabeza,mano,pie'))
CABEZA,mano,pie
>>> print(uppercase_first('-', 'tigre'))
TIGRE
```

```
def uppercase_first(sep, s):
    idx = s.find(sep)
    if idx > -1:
        up = s[:idx].upper()
        return up + s[idx:]
    else:
        return s.upper()
```

```
def uppercase_first(sep, s):
    words = s.split(sep)
    words[0] = words[0].upper()
    return sep.join(words)
```

```
def uppercase_first(sep, s):
    new_s, sep_found = '', False
    for char in s:
        if char == sep:
            sep_found = True

        if sep_found:
            new_s += char
        else:
            new_s += char.upper()
    return new_s
```

7. You and your best friend are planning on going on a road trip. You're keeping it super close, since it's only for the weekend. Because your friend is a total nerd, they decided to send you a list of lists containing potential destinations. They want you to write a program that does **two things**: c**ollapses the lists of lists into a single list**... and **picks a random destination from the resulting list**. You'll **write a function** to *flatten* the original list, and you'll **use the result of that function** to find a place to go! (8 points)

   a) Write a function called **`combine_city_state`**.
   b) Parameters: a list of lists representing cities and their states
      - the format of this is `[[state1, city1, city2, … cityn], [state2, city1, city2, … cityn], …]`
      - each sub list contains a sate name as the first element, and cities for every element after the first
   c) Processing: go through every city/state combination and create an entirely new list, where each element is a string that has the city name and the state name put together, but separated by a string
      - for example `[['NY', 'Brooklyn', 'Beacon'], ['NJ', 'Paterson']]...`
      - would result in ... `['Brooklyn NY', 'Beacon NY', 'Paterson NJ']`
   d) Return... the newly created list, which does not have nested lists in it; it should appear as above
   e) Use this function to write a program that *flattens* the following list (you can just use the variable name to represent it)...
   f) `cities_and_states = [['NY', 'Brooklyn', 'Beacon'], ['NJ', 'Paterson', 'Ringwood'], ['PA', 'Philadelphia']]`
   g) Using the result of calling your function on the list above, select a random destination and print it out
   h) (Again, **you don't have to rewrite the `cities_and_states` variable, assume that exists**)
   i) example usage:

```
cities_and_states = [['NY', 'Brooklyn', 'Beacon'], ['NJ', 'Paterson', 'Ringwood'], ['PA',
'Philadelphia']]
print(combine_city_state(cities_and_states))
# --> ['Brooklyn NY', 'Beacon NY', 'Paterson NJ', 'Ringwood NJ', 'Philadelphia PA']
```

```
# assume that cities_and_states = [['NY', 'Brooklyn', 'Beacon'], ['NJ', 'Paterson', 'Ringwood'],
['PA', 'Philadelphia']]

def combine_city_state(main_list):
    new_list = []
    for sub_list in main_list:
        for i in range(1, len(sub_list)):
            new_list.append("{0} {1}".format(sub_list[i], sub_list[0]))
    return new_list

import random
cities = combine_city_state(cities_and_states)
print(random.choice(cities))
```

8. You're tired of inadvertently writing Python variable names that aren't valid, so you decide to write a program that checks the variable name that you're about to use. To write this program, you'll create a function called **`is_valid_name`**. Your program will then continually ask the user for a variable name... and you'll use your function to determine whether or not it's valid. If the user enters an invalid variable name 3 times in a row or if they enter a valid name, stop asking for a variable name! (8 points)

   a) Create a function called **`is_valid_name`**
      - parameters: a string representing a variable name
      - processing: use the rules below to determine whether or not the variable is valid
      - return: either true or false depending on whether or not the variable name is valid

   b) A valid variable name:
      - starts with only an underscore or a letter
      - is only composed of underscores, letters or numbers

   c) Continually ask the user for a variable names
   d) Use your function to check if it's valid
   e) If the user enters a valid name... or if they enter 3 invalid names, stop asking

Example usage:

```
print(is_valid_name('1asdf'))
print(is_valid_name('#foo'))
print(is_valid_name('asdf1'))
print(is_valid_name('_foo'))
print(is_valid_name('f_oo'))
```

Example Interaction:

```
Variable name plz
> $hello
Variable name plz
> _hello
```

```
def is_valid_name(s):
    for c in s:
        if not c.isalnum() and c != '_':
            return False
    if s[0].isnumeric():
        return False
    return True



count = 0
while count < 3:
    if is_valid_name(input('Variable name plz\n> ')):
        break
    else:
        count += 1
```

9. It's 1984, and you've been hired by the Ministry of Truth to redact information from lists of words. You have a list of illegal words, and you have an incoming list of uncensored words. If any word in the uncensored list is in the list of illegal words, the first three characters of the illegal word must be replaced with dashes (`-`'s). If the word is three letters or less, then the entire word must be replaced with a series of dashes equal to the length of the word. For example: '`hello`' is converted to '`---lo`', '`hey`' is converted to '`---`', and '`hi`' is converted to '`--`'. You decide to make a **function called `redact`** to do this work for you. (6 points)

Your function takes takes **two arguments, `words` and `illegal_words`**. It returns:

a) a new list of strings composed of all of the strings in the original list, words
b) but if a string in word is in the list of illegal words, it must be partially replaced by `-`'s
c) if the word is three letters or less, the whole word is replaced by a series of `-`'s equal to the number of characters in the string; otherwise, the first three letters are replaced by `-`'s
d) example output below:

```
>>> print(redact(['hi', 'hello', 'there'], ['hi', 'hello']))
['--', '---lo', 'there']
>>> print(redact(['cute', 'puppies', 'eating', 'a', 'cake'], ['puppies', 'cake', 'a'])
['cute', '---pies', 'eating', '-', '---e']
```

```
def redact(words, illegal_words):
    redacted = []
    for word in words:
        if word in illegal_words:
            if len(word) <= 3:
                redacted.append('-' * len(word))
            else:
                redacted.append('---' + word[3:])
        else:
            redacted.append(word)
    return redacted
```

10. Write a function called **`too_much_filler`**. (5 points)

a) It should have **3 parameters**: a list of strings called **`words`**, a string called **`filler`**, and a number called **`limit`**.
b) It should **return a boolean value**. If the number of times the string, `filler`, occurs in the list, `words`, is greater than the `limit`, give back `True`. Otherwise, give back `False`.
c) **Write two assertions to test your program**
d) Example usage:

```
>>> print(too_much_filler(['you', 'know', 'like', 'words', 'and','stuff'], 'like', 2)
False
>>> print(too_much_filler(['um', 'try', 'um', 'not', 'saying', 'um'], 'um', 2))
True

# Don't forget to write two assertions as specified in part d

def too_much_filler(words, filler, limit):
    return words.count(filler) > limit

assert too_much_filler(['you', 'know', 'like', 'words'], 'like', 2) == False, 'not too much'
assert too_much_filler(['you', 'know', 'like', 'like', 'like'], 'like', 2) == True, 'too much!'
```

11. Extra Credit (3 total points)

Use the description and instructions in the first column to fill in the blanks in the second column.

The program on the right draws the image below:

The image consists of **5 squares**. The distance between the first two squares is **10 pixels**. The distance between the 2nd and 3rd is twice that. **The distance continues to double between each adjacent square**.

Several parts of the program are missing:

fill in the blank parts of the program

the function header as well as the entire function body for `draw_square` is partially blank

to get an interior angle of 90, turn 90 degrees

notice that the turtle object is passed in to the function along with the size of the square

remember that the turtle starts out facing right

```
import turtle

def draw_square(__size_______(½ pt)__):

    # implement your draw_square function here (½ pt)

   for i in range(4):

        my_turtle.forward(size)

        my_turtle.right(90)

   #  done with function!

leo = turtle.Turtle()

wn = turtle.Screen()

length = 20

spacing = 10

__for i in range 5:_______(½ pt)_

    draw_square(length)

    leo.____up()_________________(½ pt)___

    leo.forward(length + spacing)

    leo.____down()________________(½ pt)___

    spacing = _spacing * 2_____(½ pt)_

wn.mainloop()
```

Name:____ **NetID:**__________________

Scratch Paper and Reference Material

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

String Methods

capitalize
count
endswith
find
format
index
isalnum
isalpha
isdecimal
isdigit
islower
isnumeric
isprintable
isspace
istitle
isupper
join
lower
replace
split
startswith
strip
title
upper

List Methods

append
count
extend
index
insert
pop
remove
reverse
sort