CSCSCI-UA.0002 – Final Exam Practice Questions Set 1

(this practice set has more questions than the final exam, it is not an indication of the actual length of the final, and some of the topics may be extra credit)

1. **Lists** and **tuples** are pretty similar, but there are two major differences. (2 points)

   a) Obviously, they're syntactically different; write a syntactically correct tuple, called **`t`**, that contains the values 1, 2, and 3 and a syntactically correct list , called **`lst`**, that contains the values 1, 2 and 3.

```
# a tuple
t = (1, 2, 3)

# another way to write a tuple
t = 1, 2, 3

# a list
lst = [1, 2, 3]
```

   b) What can you do with a list that you can't with a tuple ?

   Lists are mutable, tuples are not. Consequently, you can change a list (append, extend, etc.), but you cannot change a tuple.

2. Your computer became sentient, and it decided that it wanted to become an artist. Its first work of *art* is the code below. In the space to the right, draw the ASCII art image that is printed out by the program (with a character per box; leave the box blank if the character is a space). (2 points)

```
s = ""
numbers = (5, 10, 15, 20, 25)
for i in range(5):
    row = ""
    for j in numbers:
        if j % 2 == 1:
            row += "x"
        else:
            row += "\\"
    s += row + "\n"
print(s)
```

x	\	x	\	x
x	\	x	\	x
x	\	x	\	x
x	\	x	\	x
x	\	x	\	x

3. You've been hired to write an English to Spanish translator by someone that loves cats and dogs. Write a **function** called **`en_espanol`** that translates English words into Spanish words. You'll **write this two ways**, but both will do the following:

   - Your function should take one argument – the word to be translated. It should return the translated word.
   - If the word is 'cat' in any casing ('Cat', 'CAT', 'cAt', etc.) return 'gato', and if the word is 'dog' in any casing return 'perro'.
   - For all other words, return 'no se'.

Example usage:

```
print(en_espanol('Cat'))
print(en_espanol('Final Exam'))
```

Resulting Output:

```
gato
no se
```

Part 1: Implement **using conditionals**. (2 points)

```
def en_espanol(word):
    new_word = None
    word = word.lower()
    if word == 'cat':
        new_word = 'gato'
    elif word == 'dog':
        new_word = 'perro'
    else:
        new_word = 'no se'
    return new_word
```

Part 2: Implement **using a dictionary to "look up" translations** instead of conditionals. (2 points)

```
def en_espanol(word):
    d = {'cat':'gato', 'dog':'perro'}
    try:
        return d[word.lower()]
    except KeyError:
        return 'no se

def en_espanol(word):
    d = {'cat':'gato', 'dog':'perro'}
    return d.get(word.lower(), 'no se')
```

4. Fill in the chart below. Write in **Yes**, **No** or **N/A** based on the attribute in the column header and the type in the left-most column. Write a syntactically correct literal value for each type in the right-most column. An example row has been filled in. (4 points)

	mutable	ordered	compound	syntax
float (float)	No	N/A	No	5.2
int (integer)	No	N/A	No	5
str (string)	**No**	**Yes**	**Yes**	**"a string"**
list (list)	**Yes**	**Yes**	**Yes**	**[1, 2, 3]**
dict (dictionary)	**Yes**	**No**	**Yes**	**{"a":1, "b":2}**
tuple (tuple)	**No**	**Yes**	**Yes**	**(1, 2, 3)**

5. What sequence of numbers would be generated from calling **range** with the following arguments? (2 points)

(a) range(3) ___**0,1,2**_____ (b) range(0, 16, 4) ___**0,4,8,12**_____

(c) range(1,3) ___**1,2**_____ (d) range(9, -1, -3) ___**9,6,3,0**_____

6. Bored with games like Scrabble, Words With Friends, and Boggle, you decide to design your own word game. In your game, you create words from a random set of letters. Your word's length and your inclusion of hard-to-use letters, such as q and z determine how many points your word is worth. Implement a function to score your words using the rules below. (5 points)

   a) the **first thee letters** count as **one point** and each **additional letter above three** counts as another **point**

   b) if the word is **less than three letters**, your word counts as **zero points**

   c) if the word has one or more **q's** in it, **multiply points by three**

   d) **one or more z's, multiply points by two**

   e) if the word has **both q'z and z's** in it, **multiply points by ten** (instead of just three or two)

   f) write a function **called score_word** that takes a **string as input** and **returns an integer representing the score**

   g) assume that the input you will get will consist of only lowercase letters (no punctuation, numbers, spaces or uppercase)

   h) example words and their corresponding scores (make sure you try them all as inputs to your actual implementation):

quiz = (1 + 1) * 10 = **20** points
quits = ((1 + 2) * 3) = **9** points
no = 0 + 0 = **0** points
zip = (1 + 0) * 2 = **2** points
toast = (1 + 2) = **3** points

```
def score_word(word):

    # start with default score of zero
    score = 0

    if len(word) >= 3:
        score = 1 + len(word) - 3

        # make sure most 'narrow' condition is first!
        if 'q' in word and 'z' in word:
            score *= 10
        elif 'q' in word:
            score *= 3
        elif 'z' in word:
            score *= 2

    # else is not necessary, since everything that's not >= 3...
    # is already taken care of by the default score
    return score
```

7. Read the code in the 1st column. Write the output of the code in the 2nd column, and answer the question in the 3rd column. **No output** and **error** are both possible. (3 points)

Code	Output	Question

`def cheer(a, b, c, d):`
`  s = '%s %s %s %s' % (a, b, c, d)`
`  return s`
 	
`a, b = 'hip', 'hip'`
`c, d = 'hoorah', '!'`
`e = cheer(d, c, b, a)`
	What is the output of this program?
	No output
	(no print statement)
		What value is in **e**?
		! hoorah hip hip
		(see the order of the arguments passed in)

`a = {'sound': 'beep', 1:200}`
`def make_some_noise():`
`  noise = (a['sound'] + '!')`
`  print(noise)`
 	
`b = make_some_noise()`
	What is the output of this program?
	beep!
	(dictionary, a, is global)
		What value is in **b**?
		None
		(nothing returned)

`def make_some_noise(noise):`
`  return noise + '!' * 4`
 	
`result = make_some_noise('pop')`
`print(noise)`
	What is the output of this program?
	There's an error:
	NameError: name 'noise' is not defined
		What value is in **result**?
		pop!!!!

8. Answer the following questions about exceptions. (2 points)

   a) Name two kinds of exceptions and describe why/when they occur:

   - **IndexError – index doesn't exist in sequence**
   - **TypeError – type is not supported for operation or function**
   - **ZeroDivisionError – attempted to divide by 0**
   - **KeyError – key doesn't exist in dictionary**
   - **etc. ...**

   b) Use **exception handling** to prevent an error from happening when the user types in `'foo'`. Instead of an error, the program should write out, "I don't have that data!". Use arrows or cross out and rewrite code if indentation is required.

```
person = {'first':'Alice', 'last':'Abel', 'middle':'A'}
response = input('What data would you like?')


try:
    print(person[response])
except:
    print('I don\'t have that data!')
```

9. Imagine that the code in the first column is executed line-by-line. After each line is executed, examine the state of the variables **a**, **b**, **c** and **d**, and write the values that are bound to them in the columns to the right of the code. The variables may contain a value, may not yet exist (**N/A**), or may have a value that hasn't changed from the previous value (**unchanged**).

   All of the **N/A** and **unchanged** items have been filled in (this can be used as a hint!). Fill in the remainder of the missing values. (4 points)

code	a	b	c	d
`a = [5, -1, 4, {'x':7}]`	**[5, -1, 4, {'x':7}]**	N/A	N/A	N/A
`b = a.pop()`	**[5, -1, 4]**	**{'x': 7}**	N/A	N/A
`b['y'] = 24`	unchanged	**{'x': 7, 'y': 24}**	N/A	N/A
`c = b.get('z', 10)`	unchanged	unchanged	**10**	N/A
`d = a.extend([3, 8])`	**[5, -1, 4, 3, 8]**	unchanged	unchanged	**None**
`del a[-1]`	**[5, -1, 4, 3]**	unchanged	unchanged	unchanged

10. You have two lists, each composed of integers:

```
numbers = [1, 2, 3, 4, 5]
more_numbers = [6, 7]
```

a) Write out two different ways to remove the number 5 from the above list, **numbers**. Do this ***in place***; this means that you cannot overwrite the **numbers** variable using assignment.(1 point)

- `numbers.pop()`
- `del numbers[-1]`

b) Write out two different ways to add the elements from the list, **more_numbers**, to the other list, **numbers**, as individual, and discrete elements (either creating a new list or modifying the existing **numbers** list). (1 point)

- `numbers.extend(more_numbers)`
- `numbers + more_numbers`

11. Using the two variable declarations below, what is the output of the following lines of code? If the code results in an error, write "error" in the space provided. (3 points)

```
animal = "giraffe"
idx = len("animal")
```

```
a) print("animal"[idx])
b) print(animal[2:])
c) print(animal[-1] + str(idx))
d) print(animal[4:(idx * 2)])
e) print("animal"[-2])
f) print(animal[idx])
```

(a)**error**

(b)**raffe**

(c)**e6**

(d)**ffe**

(e)**a**

(f)**e**

12. Use the following dictionary to answer the questions below. (2 points)

```
d = {'x':100, 'y':200}
```

a) Write two syntactically correct ways of **retrieving** the value at key, **'x'** from the dictionary, **d**.

```
d.get('x')
d['x']
```

b) Write a syntactically correct way of **adding** a new key value pair, **'z':300**, to the dictionary, **d**.

```
d['z'] = 300
```

c) Write a syntactically correct way of **removing** the element, **'x'**, from the dictionary **d**.

```
del d['x']
```

13. Complete the code below to draw a **triangle** with a side of **75** pixels (the triangle can be pointed up or down). (3 points total)

```
import turtle

don = turtle.Turtle()
wn = turtle.Screen()

for i in range(3):
    don.forward(75)
    don.right(120)

wn.mainloop()
```

14. You have an electronic journal that you'd like to protect with a password, so you wrote a short program. (2 points)

    a) The program continually asks the user (you!) for a password (it's *pickles*!)...

    b) If the user enters the right password (again, it's *pickles*), the loop stops, and your journal is printed out to the screen

    c) Otherwise... it just loops forever, asking for the password

    However, just as you were about to run your program, your cat jumped onto your keyboard and deleted two very important lines. Now, whatever you enter, you'll still be asked for the password again, even if you got the password right! Fix the program so that when you enter the right password, the loop stops, and the line that prints out secret journal stuff is executed.

```
while True:
        cmd = input('What's the secret password?\n>')


        if cmd = 'pickles':
                break


print('Some super secret journal stuff here')
```

15. Name two built-in Python modules. For each **module**, name **one function** that you can call from that module and what it does. (2 points)

    Module Name: `math` Function: `sqrt(number)`

    Module Name: `random` Function: `randint(1, 5)`

    Other modules include: `turtle, sys`

16. Write a function called `count_letters`. It should take a string as an argument. It should return a dictionary that contains individual letters as keys and the corresponding counts as values. (4 points)

    a) Only count actual letters; do not count punctuation, whitespace or numbers

    b) Consider uppercase and lowercase letters as the same letter, and put them in the key that's the lowercase version of the letter

    c) Write a test (use an assertion) for the result of your function

    d) For example:

```
s = 'Eeeeaasy!!!'
d = count_letters(s)
print(d)

# output should be similar to
# {'a': 2, 'y': 1, 's': 1, 'e': 4}


def count_letters(s):
    d = {}
    for c in s:

        # normalize casing for counting purposes (all lower is fine)
        letter = c.lower()

        # ignore punctuation and numbers
        if letter.isalpha():
            d[letter] = d.get(letter, 0) + 1

    return d

# can also use try / except version
```

17. Because you find yourself computing factorial *so often*, you decide to write a function that does it for you. Unfortunately, before you could finish your implementation, you fell asleep, dreaming of exclamation points and unicode snowmen … and when you woke up, you found code that wasn't quite right. (3 points)

Wait – what's factorial again? Oh yes: n factorial or n! is the product of all non-negative integers less than n. For example, 4 factorial is 4 * 3 * 2 * 1 .

```
def fact(n):

    result = 1

    while n > 0:

        result = result * n

        # n need to change in order for the loop to eventually reach an end condition.
        # (the end condition is when n is no longer > 0... so keep subtracting 1 from n)
        n = n – 1

    return result

print(fact(4))
```

a) When you run it, what happens? Circle one of the answers below:

Loops forever / eventually crashes; prints "1" repeatedly	Loops forever / eventually crashes; prints "4" repeatedly	**Loops forever / eventually crashes; nothing printed (no output)**
Program ends normally; prints "24" exactly once	Program ends normally; prints "4" exactly once	Program ends normally; prints "1" exactly once
Error	Program ends normally; no output	Program ends; prints out "result"

b) How would you fix the implementation so that factorial is computed appropriately? You can make the fix directly in the code above.

See code above

18. Create a function called **`flip_sign`**. It will flip the sign (negative ^ positive, positive ^ negative) of every other number in a list of numbers, starting with the first. It will do this ***in place***. (4 points)

a) the function takes one argument, a list of numbers

b) the list is modified by…

c) changing the sign of every other list element, starting with the first

d) it **does not return** anything, rather it changes the list passed in ***in place***

the output below shows the expected behavior (note that my_numbers, which is passed in, is changed *in place*)

```
my_numbers = [1, 2, 3, 4, 5, 6]
print(my_numbers)
flip_sign(my_numbers)
print(my_numbers)
```

Results in the following output:

```
[1, 2, 3, 4, 5, 6]
[-1, 2, -3, 4, -5, 6]
```

```
# 3 different ways to do this... just need current index of element

def flip_sign(numbers):
    for i, number in enumerate(numbers):
        if i % 2 == 0:
            numbers[i] *= -1

def flip_sign(numbers):
    for i in range(len(numbers)):
        if i % 2 == 0:
            numbers[i] *= -1

def flip_sign(numbers):
    i = 0
    for number in numbers:
        if  i % 2 == 0:
           numbers[i] *= -1
        i += 1
```

19. What are the values bound to the variables **n** and **points** after running this program? (2 points)

```
n = 0
points = [(1, 2, 3), (4, 5, 1), (1, 2, 1), (2, 2, 2)]
for i in range(2):              # loops twice (i is 0 and 1)
    x, y, z = points.pop()      # pops two elements... and unpacks
    n += x * y * z              # 1* 2 * 1 and 2 * 2 * 2    2 + 8
```

(1) **n**_____**10**_____ (2)**points**_____**[(1, 2, 3), (4, 5, 1)]**_____

20. **Read the code sample** in the first column. **Answer the questions** in the second column. (5 points)

Code	Question

`a = [[1, 2, 3], [4, 5, 6], [7, 8, 8]]`
`b = []`
`for i in range(len(a) - 1, -1, -1):`
`  b.append(a[i].pop())`
	What are the values of **a** and b?
	a [[1, 2], [4, 5], [7, 8]]
	b [8, 6, 3]
	(pops the last element off of every inner list)

`a = 'umxexcusemex!'`
`b = a.split('x')`
`c = '8'.join(b)`
	What are the values of **b** and **c**?
	b ['um', 'e', 'cuseme', '!']
	c 'um8e8cuseme8!'

`a = [[[1,2,3],[4,5,6]], [1,2,3,4,5]]`
`val = a[1].index(4)`
`print(val // 2)`
	What code would you use to retrieve the value **6** from **a**. What is the *output* of the program?
	retrieve 6 a[0][1][2]
	output 1
	(val is 3, 3 // 2 1)

`car = {'x':0, 'y':200}`
 	
`def move_car():`
`  for i in range(4):`
`  car['x'] += 20`
 	
`move_car()`
`print(car)`
	What is the *output* of this program?
	{'x': 80, 'y': 200}
	(car is global; it's a dictionary and is mutable... each iteration, 20 is added to the value that's at x)

`s = ''`
`greeting = 'Hello there!'`
`for c in greeting.upper():`
`  if c not in 'AEIOU' and c not in s:`
`  s += c`
`print(s)`
	What is the *output* of this program?
	HL TR!

21. Create a function called **slice_and_splice**. It takes a string as an argument. It returns a new string starting with the last half of the original word and ending with the first half of the word. If the word cannot be split evenly, the last half of the original word should get the extra letter. See the `'hello'` example below. (3 points)

```
>>> slice_and_splice('hi')
'ih'
>>> slice_and_splice('i')
'i'
>>> slice_and_splice('hello')
'llohe'
>>> slice_and_splice('violin')
'linvio'
```

```
def slice_and_splice(s):

    # integer division rounds down
    # i represents split so that end of word gets extra character
    i = len(s) // 2

    # end of original word + beginning of original word using i
    return s[i:] + s[:i]
```

22. Cross out all of the statements that are false or evaluate to False. (3 points)

(a) ~~`7 not in ['hi', None, 7, 3]`~~

(b) `['a', 6, 4, 5] > ['a', -2, 4, 8]`

(c) ~~strings are an unordered sequence of characters~~

(e) strings are immutable

(f) ~~join and split are both methods called on list objects~~

(g) ~~`'twelve'.isdigit()`~~

23. What is the output of this code? Show some indication of how you arrived at your answer. (4 points)

```
def add_to_list(lst, step, total_length):
    if len(lst) == total_length:
        return lst
    else:
        try:
            next_element = lst[-1] + step
        except IndexError:
            next_element = 0
        new_lst = lst[:]
        new_lst.append(next_element)
        return add_to_list(new_lst, step, total_length)
print(add_to_list([], 1, 3))
print(add_to_list([99, 100, 101], 1, 5))
```

```
#1    []
#     index error so new_lst is [0]
#     add_to_list([0], 1, 3)
#     add_to_list([0, 1], 1, 3)
#     add_to_list([0, 1, 2], 1, 3)
#     stops because length of list is 3
```

Output line 1: __[0, 1, 2]_____

Output line 2: __[99, 100, 101, 102, 103]_____

24. You have a file called `numbers.txt`. It contains the following text:

```
7,hello,2,8,1,0
2,44
5
23,not a number,11,12
```

Each item is separated by a comma, each line is separated by a new line. Write a program to read the file and **print** out the **sum all of the numbers** contained in the file. It should ignore items that are not numbers. (5 points)

Hints:
(a) use the built-in function, open, with the appropriate mode, 'r', to read the file
(b) you can iterate over a file object to get every line
... or use the read() method to read the entire contents of the file as a string
... or use the readlines() method to read the entire contents of the file as a list of strings
... or use readline() to read one line at a time (remember to loop forever and break if string is length 0)
(c) use a string method to break up each line into a list
(d) there may be nested loops involved (iterate over every line, itereate over every item in each line)
(e) depending on which method you used for reading the file, you may need to strip off a newline character...

```
f = open('numbers.txt', 'r')
total = 0
for line in f:                          # iterate over the file object
    line_clean = line.strip()           # remove new line
    items = line_clean.split(',')       # split up each line by comma
    for item in items:
        if item.isnumeric():            # determine if element is numeric
            total += int(item)          # don't forget to change to an int!
print(total)
f.close()
```

```
f = open('numbers.txt', 'r')
entire_file = f.read()                  # read the entire contents of the file as a string!
lines = entire_file.split('\n')         # split that string into separate lines by newline
total = 0
for line in lines:
    items = line.split(',')
    for item in items:
        if item.isnumeric():
            total += int(item)
print(total)
f.close()
```

```
f = open('numbers.txt', 'r')
total = 0
for line in f.readlines():              # read entire contents of file as a list and iterate
    items = line.strip().split(',')     # strip and split in same line
    for item in items:
        if item.isnumeric():
            total += int(item)
print(total)
f.close()
```