# CSCSCI-UA.0002 – Final Exam Practice Questions Set 2

1. At the end of this program, what values do the variables **d1**, **d2**, **a** and **s** contain? Write your answers in the space provided to the right of the code. **The order of the key value pairs will not be taken into account**. (2 points)

```
d1 = {'first_name':'Stanley', 'year_born':1928}
d2 = {'first_name':'Ridley', 'year_born':1937}
a = d1.get('last_name', 'Unknown')
d2['last_name'] = 'Scott'
s = ''
for k, v in d2.items():
    s += str(v)
```

**d1** ___{'first_name': 'Stanley', 'year_born': 1928} (order does not matter)

**d2** ___{'last_name': 'Scott', 'first_name': 'Ridley', 'year_born': 1937} (order does not matter)

**a** ___Unknown_____

**s** ___ScottRidley1937 (order does not matter)_____

2. Find 2 values that a user can input that will cause the following program to crash. Why is an error caused by the input? Fill in the blanks below the code. (2 points)

```
import random
cards = ['empty', 'empty', 'penny']
random.shuffle(cards)
a = input('Please choose a card: 0, 1 or 2\n>')
number = int(a)
if cards[number] == 'penny':
    print('you guessed right!')
else:
    print('nope, nothing there')
```

a) **Error Causing Input 1:**_____5_____ **Reason for Error:**___IndexError (index larger than len of list)

b) **Error Causing Input 2:**___hi_____ **Reason for Error:**___ValueError (can't change 'hi' to int)

3. Instead of letting the program above crash, use exception handling so that a different message is displayed for each of the possible errors from user input . The error messages that are printed out can be of your choosing (for example, "bad input" for one and "sorry, try again" for the other). Rewrite the code from the previous program in the space below. (3 points)

```
# use can use try/except starting here:
try:
    a = input('Please choose a card: 0, 1 or 2\n>')
    number = int(a)

    if cards[number] == 'penny':
        print('you guessed right!')
    else:
        print('nope, nothing there')
except IndexError:
    print('sorry try again – that number does not exist')
except ValueError:
    print('bad input – not a number')
```

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**. **Leave a box blank to represent a space character**. You do not have to use all of the boxes. (3 points)

```
def create_table(size, letters):
    table = ''
    for i in range(1, size + 1):
        row = ''
        for c in letters:
            row += str(i) + c + ' '
        table += row + '\n'
    return table

def main():
    num, s = 4, 'abc'
    print(create_table(num, s))
main()
```

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | a | | 1 | b | | 1 | c | | |
| 2 | a | | 2 | b | | 2 | c | | |
| 3 | a | | 3 | b | | 3 | c | | |
| 4 | a | | 4 | b | | 4 | c | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

5. Complete the two function definitions below for `square_the_odds` and `square_the_odds_in_place`.

   a) It should take a list of integers as a parameter

   b) It should either:
   - return a **new** list with all of the elements in the original list, except with every odd number squared...
   - modify the list passed in ***in place*** (the original list passed in will be modified), so that the odd numbers are squared...

   c) For example: [1, 2, 3, 4, 5, 6, 7] ^ [1, 2, 9, 4, 25, 6, 49] (either returned new or in place)

   d) Don't worry about input that's not a list of integers.

```
def square_the_odds(numbers):

    new_list = []
        for number in numbers:
            if number % 2 != 0:
                new_list.append(number * number)
            else:
                new_list.append(number)
    return new_list
```

```
def square_the_odds_in_place(numbers):

    for i in range(len(numbers)):
        if numbers[i] % 2 == 1:
            numbers[i] *= numbers[i]
```

6. Write the output of the following code in the space to the right.

```
numbers  = [1, 2, 3]
same_numbers = numbers
copy_numbers = numbers[:]

print('part 1')
print(numbers)
print(same_numbers)
print(copy_numbers)

result1 = same_numbers.append(4)
result2 = copy_numbers.pop()

print('part 2')
print(result1)
print(result2)

print('part 3')
print(numbers)
print(same_numbers)
print(copy_numbers)

copy_numbers.append(numbers)
copy_numbers.extend(same_numbers)

print('part 4')
print(copy_numbers)
```

```
Part 1
[1, 2, 3]
[1, 2, 3]
[1, 2, 3]

part 2
None
3

part 3
[1, 2, 3, 4]
[1, 2, 3, 4]
[1, 2]

part 4
[1, 2, [1, 2, 3, 4], 1, 2, 3, 4]
```

7. Create a function called **`fib`** that generates a Fibonacci sequence. An example sequence is `0, 1, 1, 2, 3, 5, 8, 13, 21, 34`. By definition, the **first two numbers in the Fibonacci sequence are 0 and 1**, and **each subsequent number is the sum of the previous two**.

   a) it should take **a single parameter, the number of digits to generate** (remember, the first two are always 0 and 1)

   b) **it will not return anything**; instead, it should **print out each digit on a new line**

   c) for example: `fib(10)  0, 1, 1, 2, 3, 5, 8, 13, 21, 34`

   d) hint: again, the sequence starts with 0 and 1, so the numb that need to be generated programmatically are actually 2 less than the argument passed in

   e) hint: swapping may help

   hint: you'll have to keep track of the previous 2 and the calculated value

```
# with a for loop (doesn't really handle 0 or 2, but easy to continue with elifs!)
def fib(n):
    if n > 2:
        a, b = 0, 1
        print(a)
        print(b)
        for i in range(n – 2):
            c = a + b
            print(c)
            a, b = b, c

# another way with while
def fib(n):
       i = 0
       cur = 1
       prev = 1
       prev_prev = 0
       while i < n:
              print(cur)
              prev_prev = prev
              prev = cur
              cur = prev + prev_prev
              i += 1
```

8. Create a function called **`encrypt`**. It will create a new string from an input string, with certain characters substituted. (3 points)

   a) it should take 2 arguments, a dictionary that has characters as keys and values as translated characters

   b) the encrypted string will be the original string inputted, and the translated string will have the letters a, b, c, d, and e substituted with punctuation marks, and with everything else substituted with dashes ('-')

   c) the 'translation guide' for the letters will be stored in a dictionary that is passed on into the function call

   d) for example:

```
# assume that the dictionary d exists, and you can use it pass in to the function
d = {'a':'!', 'b':'@', 'c':'#', 'd':'$', 'e':'%'}
print(encrypt(d, 'cab'))    #!@
print(encrypt(d, 'modes'))    --$%-
```

   e) the 'everything else' substituted by a dash character is not specified in the dictionary; that particular logic can be handled by your program

```
def encrypt(translation, s):
    translated = ''
    for c in s:
        translated += translation.get(c, '-')
    return translated

def encrypt2(translation, s):
    translated = ''
    for c in s:
        if c in translation:
            translated += translation[c]
        else:
            translated += '-'
    return translated

def encrypt3(translation, s):
    translated = ''
    for c in s:
        translated += translation.get(c, '-')
    return translated

def encrypt4(translation, s):
    for c in s:
        s = s.replace(c, translation.get(c, '-'))
    return s
```

9. Answer the following questions about lists, strings and tuples. (3 points)

   a) Name **two operations and/or functions** that are supported by **sequence** types (lists, strings, and tuples)

   (1)___**slice – [m:n]**_____ (2)_____**len()**_____

   b) Give two examples of how lists and strings different (excluding different methods)

   (1)_____**strings are immutable, lists are not**_____

   (2)_____**strings consist of characters; lists consist of values of any type**_____

   c) What's the difference between a tuple and a list?

   **A tuple is immutable, a list is mutable**

10. You love pizza parties, but organizing them is a *drag*... especially finding out how many pies to purchase! Write a short program that helps you calculate the number of pies to purchase for a pizza party.

    Part 1: Define a function called `how_many_pies`. It should take two arguments: the number of people eating pizza and the number of slices each person will have (this will be the same for every person; none of that 3 for her and 2 for him stuff!).

    1. a) It should assume that every pie comes with 8 slices.
    2. b) The function will always over-order pies... meaning that if the number of pies has to be rounded, round up.
    3. c) For example, if 5 people are coming to the party, each person wants 2 slices, and each pie has 8 slices, we would want 2 pies

       to accommodate everybody!
    4. d) You don't have to worry about non-integer, zero or negative input
    5. e) You can use a function in the math module called ceil to round up. It takes one argument and it returns the smallest integer

       value that's greater than or equal to the original argument. For example: `math.ceil(1.2) 2`.
    6. f) An example run of the function itself: `print(how_many_pies(9, 3)) 4`

    Part 2: Use this function after asking from input for the user. (That is, assume that you've already written your function in the same file; you just have to use it below). It should ask for number of people and number of slices. You don't have to worry about non-integer input. Example interaction below (everything after > is user input):

```
How many people?
>9
How many slices?
>3


You'll need 4 pies for a pizza party
```

```
# part 1
import math
def number_of_pies(people, num_slices):
    return math.ceil((people * num_slices) / 8)
assert 4 == how_may_pies(9, 3), 'test round up'

# part 2

p = int(input("How many people?\n>"))
n = int(input("How many slices?\n>"))
print("You'll need %s pies for a pizza party" % (number_of_pies(p, n)))
```

11. True or False (3 points)

```
a) str(5) == '5'                          (a)_____True_____
b) '10'.isdigit()                         (b)_____True_____
c) 25 < 7 * 4 - 1 and True                (c)_____True_____
d) False and not False or True            (d)_____True_____
e) 'a' in {'b': 2} or 12 != 'string'      (e)_____True_____
f) False and (10 == 10 or 'a' == 'a')     (f)_____False_____
```

12. List three methods that you can call on file objects, along with what they do:

**read() - reads in entire contents of a file as a single string**
**readline() - reads a single line from a file**
**readlines() - reads in entire contents of a file as a list**
**write() - writes a line to a file**
**close() - closes a file object**

13. What data structure would you use to hold a word and all of its synonyms. The data structure should be flexible enough to add and/or remove synonyms. For example, lists, strings, tuples, some combination of data types (a dictionary of tuples), etc.?

```
good – bully, cracking, great
```

**A dictionary with the key being the word and the value being a list of synonyms.**

```
{'good':['bully', 'cracking', 'great'], 'happy':['glad', 'pleased']}
```

**A list of 2-element tuples, with the 1st element being the word, and the 2nd being a list of synonyms.**

```
[('good',['bully', 'cracking', 'great']), ('happy', ['glad', 'pleased'])]
```

14. 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**, and **c**, 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 or may have a value that hasn't changed yet (that is, it's the same as the previous value that it had).

For example, after the code in the first row is executed, **a** contains `[21, 7, 14]`, while **b** and **c** do not exist yet. Some of the table is already filled in; fill in the remainder of the missing values. (3 points)

| Code | a | b | c |
|---|---|---|---|
| `a = [21, 7, 14]` | `[21, 7, 14]` | does not exist | does not exist |
| `b = [35, 35, 35]` | `[21, 7, 14]` | `[35, 35, 35]` | does not exist |
| `a.append([14, 28])` | **`[21, 7, 14, [14, 28]]`** | `[35, 35, 35]` | does not exist |
| `b.extend(a.pop())` | **`[21, 7, 14]`** | **`[35, 35, 35, 14, 28]`** | does not exist |
| `c = a.sort()` | **`[7, 14, 21]`** | **`[35, 35, 35, 14, 28]`** | **`None`** |

15. Write a recursive version of a factorial function: (4 points)

fact(4) ^ 24

```
def fact(n):
    if n == 0:
        return 1
    else:
        return n * fact(n - 1)
```