CHALLENGE EXAM REVIEW

- The CSCI 128 challenge exam will be a mixture of true/false, fill-in-the-blank, Short Answer, Code/Error tracing and code writing questions in Python.
- The length and distribution of questions/points/topics may change in the real exam.
- This review exam exists to help familiarize you with the challenge exam format, and should **NOT** be treated as comprehensive review.
- The passing grade for the exam is at least 70 points out of 100. The exam will be 90 minutes.
- You are allowed to bring one 8.5”x11” sheet of handwritten notes to use during the exam. Writing can be on both sides of the note sheet.

1. **True or False**. Write T or F to the left of each statement.

   1. _F_ Binary search will always find a value faster than linear search, for any list.
   2. _T_ Recursive functions can have more than one base case.
   3. _F_ A recursive function call should use the exact same arguments as what was given to the function originally.
   4. _T_ With nested loops, for each iteration of the outer loop, the inner loop runs fully.
   5. _F_ A break statement in a while loop terminates the current iteration of the loop and moves onto the next iteration.
   6. _F_ `2nd_score` is a valid variable name in Python.
   7. _T_ When used with integers, the // operator in Python will always evaluate to an integer.
   8. _T_ The same conditional chain can support multiple `elif` branches.
   9. _T_ The Python `pass` keyword does not actually do anything while a program is running.

2. **Fill in the blanks**.

   1. An object in Python is something that contains a combination of _____data/variables/attributes_____ and _____behavior/functions/methods_____.
   2. `((F or (T and F)) or ((T and T) and F)` evaluates to _____False_____.
   3. The keyword to end not only any current loops but also the current function is _____return_____.
   4. The variables declared when a function is first defined, and assigned when the function is called are _____parameters_____.
   5. The values provided to a function on the line of code where it is called are _____arguments_____.
   6. The last item in a list named `lst` can be retrieved with `lst[`_____len(lst) - 1_____`]` or `lst[`_____-1_____`]` (provide two different ways).

3. **File Paths**. Consider the file system laid out below. Folders don't have file extensions. Answer the following questions about file paths within the system.

```
/
└─ users
   └─ me
      ├─ cs128
      │  ├─ assignment1
      │  │  ├─ main.py
      │  │  └─ data.txt
      │  ├─ assignment2
      │  │  ├─ a2.py
      │  │  └─ data.csv
      │  └─ modules
      │     └─ matplotlib.py
      └─ other_file.txt
```

   (a) What is the *absolute* file path to `a2.py`?
   /users/me/cs128/assignment2/a2.py

   (b) What is the *relative* file path to `data.txt` from `main.py`?
   data.txt

   (c) What is the *relative* file path to `data.csv` from `main.py`?
   ../assignment2/data.csv

4. **Code Tracing**. For the program below, trace the code and write the values of the i, j, and total variables *each time line 5 executes* when the program is run. Then also answer with the final value printed by this program. You can assume this code will not throw any errors. A correct answer will fill all rows of the table.

```
1  def scan_grid(g, num):
2      total = 0
3      for i in range(len(g)):
4          for j in range(len(g[i])):
5              if g[i][j] == num:
6                  total += g[j][i]
7      return total
8
9  if __name__ == "__main__":
10     g = [[1, 2, 3],
11          [3, 4, 2],
12          [1, 4, 3]]
13     g[1] = g[1][::-1]
14     print(scan_grid(g, 3))
```

i	j	total
0	0	0
0	1	0
0	2	0
1	0	1
1	1	1
1	2	1
2	0	5
2	1	5
2	2	5

Final printed value:

8

Type Labeling. Consider the following code. For each line requested, write the Python **type** of the data that will be assigned to the variable defined on that line. You can assume this program will not throw any errors.

```
1  def my_func(a):
2      a = a
3      for i in range(a):
4          l = input()
5          l = float(l)
6          l = l // 3
7          l2 = input().split()
8          part = l2[1:2]
9
10 if __name__ == "__main__":
11     v = my_func(6)
```

Report the type assigned on...

- Line 2: int or integer
- Line 4: string
- Line 5: float
- Line 6: float (only because l was a float)
- Line 7: list
- Line 8: list
- Line 11: None

5. **Error Tracing**. For each broken code snippet and associated error message when the code is run, specify the line number where the error is first introduced, and a re-write of the line to fix the error.

   (a) **TypeError: Point.__init__() takes 2 positional arguments but 3 were given**

```
 1  class Point:
 2      def __init__(x, y):
 3          self.x = x
 4          self.y = y
 5
 6      def __str__(self):
 7          return f'({self.x}, {self.y})'
 8
 9  p = Point(5, 8)
10  print(p)
```

   Line number: 2
   Fixed line of code: def __init__(self, x, y):

   (b) **IndexError: list index out of range**

```
1  nums = [2, 6, 8, 4, 1, 9]
2  sum = 0
3  for i in nums:
4      sum += nums[i]
5  print(sum)
```

   Line number: 3 OR 4
   Fixed line of code: for i in range(len(nums)): OR sum += i

6. **Recursive Code**. The following function is meant to take a string as input and recursively reverse it. The return value should be the original string argument but in reversed order. Fill in the blank line with Python code that allows this function to work as intended. Your line must include a recursive call to `recursive_reverse`.

```
def recursive_reverse(str):
    if str == '':
        return ''
    else:
        return _________
```

recursive_reverse(str[1:]) + str[0]

7. **Recursive Code Tracing**. Analyze the following recursive function, then rewrite it to not use recursion (likely with loop(s) instead). This new function should produce the exact same output as the original when given the same argument.

```
def rec(s):
    if len(s) == 0:
        return ""

    n = s[0] + rec(s[2:])
    return n

rec("abcdefghi")
```

Answer:

```
def rec(s):
    return s[::2]
```

OR

```
def rec(s):
    out = ''
    for i in range(len(s)):
        if i % 2 == 0:
            out += s[i]
    return out
```

8. **Classes and Objects**. Consider the code below for a class to parse a data file with csv. You can assume this class works without error. **Using this class**, write Python code that prints:

   1. The full contents of a file called `samples.csv`
   2. The 2nd and 5th columns from a file called `data.csv`
   3. A Boolean indicating whether there are any 5s in the 3rd data column of `samples.csv`

```python
import csv

class FileParser:
    '''Class to parse a data file into columns, and return data from that file'''
    def __init__(self, filename):
        with open(filename, 'r') as f:
            reader = csv.reader(f, delimiter=',')
            lines = []
            for line in reader:
                lines.append(line)
        self.lines = lines

    def getColumn(self, col_num):
        col = []
        for i in range(len(self.lines)):
            col.append(self.lines[i][col_num - 1])
        return col

    def __str__(self):
        out = ""
        for line in self.lines:
            out += f'{line}\n'
        return out

```

Answer:

```python
fp = FileParser('samples.csv')
print(fp)
fp2 = FileParser('data.csv')
print(fp2.getColumn(2))
print(fp2.getColumn(5))
print('5' in fp.getColumn(3))
```

9. **Parson's Problem**. Given the lines of code below, re-order them into error-free Python code that creates and prints a nested list representing a chess board of alternating `'b'`lack and `'w'`hite cells.

   a) `l.append('w')`

   b) `b = []`

   c) `for i in range(8):`

   d) `l = []`

   e) `b = b + [l]`

   f) `print(b)`

   g) `l.append('b')`

   h) `for j in range(8):`

   i) `else:`

   j) `if (i + j) % 2 == 0:`

   Write the letters for each line above in the order that produces a working program. It is possible there is more than one correct order. You do not need to indicate indentation. Use the space to the right as a scratch work area if you need it.

   1. b
   2. c OR h
   3. d
   4. h OR c
   5. j
   6. g OR a
   7. i
   8. a OR g
   9. e
   10. f

10. **Code Writing 1**. Write a function called `count_writer` that takes a file name and an integer `n` as parameters. The function should write numbers 1 through n to that file, counting up by one and only allowing at most ten numbers per line. Each number should be separated by `*`, asterisks.

    Examples:

    Assuming `count_writer("count.txt", 5)` is called, the first and only line of text in `count.txt` would be:

    ```
    1*2*3*4*5
    ```

    If `count_writer("count.txt", 12)` is called, the contents of `count.txt` would be:

    ```
    1*2*3*4*5*6*7*8*9*10
    11*12
    ```

    Answer:

    ```
    def count_writer(filename, n):
        with open(filename, 'w') as f:
            for i in range(1, n+1):
                if i % 10 == 1:
                    f.write(f'{i}')
                else:
                    f.write(f'*{i}')
                    if i % 10 == 0:
                        f.write('\n')
    ```

11. **Code Writing 2**. Write a program that takes a line of integers, separated by spaces, as input. Your program should re-sort this list to put the smallest values in the middle of the list, increasing as you move out to the edges. In cases where there are two places equidistant from the middle, choose the left option first. Print this list in its new order as the program's output.

    Note: To simplify this problem, you can assume the user input will always contain an odd amount of numbers.

    Hint: Even though it won't give you the final result, Python's sort function will still be useful to you here.

    Examples:

```
NUMS> 1 2 3 4 5
OUTPUT [4, 2, 1, 3, 5]

NUMS> 7 4 2 6 9 1 3
OUTPUT [7, 4, 2, 1, 3, 6, 9]
```

Answer:

```
lst = [int(x) for x in input().split()]
out = lst[:]

lst.sort()
middle = len(out) // 2
out[middle] = lst[0]
for i in range(len(lst) // 2):
    out[middle - (i+1)] = lst[(2*i) + 1]
    out[middle + (i+1)] = lst[(2*i) + 2]
print(out)
```