CS303E Week 12 Worksheet: Recursion

Name: ______________________________ EID: ______________

Read the questions carefully, and answer each question in the space provided. Use scratch paper to do your work and then copy your answers neatly and legibly onto the test paper. Only answers recorded on the test paper will be graded.

1. (12 points: 1 point each) The following are true/false questions. **Write either T or F in the boxes at the bottom of page 1.** If there's any counterexample, it's false.

   (a) Recursion is a programming technique where a function calls itself.

   (b) A recursive function must have a base case to stop the recursion.

   (c) Any recursive function using a list can be adjusted to use a set.

   (d) Recursive functions are always more efficient than iterative solutions.

   (e) Recursive functions can be called with different parameters in each recursive call.

   (f) A recursive function can have multiple base cases.

   (g) Recursive functions can only call themselves once within their body.

   (h) Recursion is the only way to implement functions that solve certain mathematical problems efficiently, such as factorials.

   (i) If function A calls function B, and function B calls function A, then neither function is recursive because they do not directly call themselves.

   (j) Recursion always leads to infinite recursive calls if not implemented correctly.

   (k) Recursion is generally recommended for problems that can be easily solved using loops.

   (l) Recursive functions in Python cannot be optimized for better performance.

a	b	c	d	e	f	g	h	i	j	k	l

Page total: ________/12

Questions 2-9 are multiple choice. Each counts 2 points. **Write the letter of the BEST answer in the box on the next page. Please write your answer in UPPERCASE. Each problem has a single answer.**

2. What is the purpose of a base case in a recursive function?
   - A. To make the code more readable and maintainable.
   - B. To handle errors and exceptions that may occur during recursion.
   - C. To define the initial condition that stops the recursive calls.
   - D. To ensure that the function returns a value at every step of the recursion.
   - E. Base cases are not required in recursive functions.
3. Which of the following is true about the relationship between recursive and iterative solutions?
   - A. Recursive solutions are always clearer than iterative solutions.
   - B. Iterative solutions are always more efficient than iterative solutions because the function does not call itself repeatedly.
   - C. Iterative solutions are prone to errors such as infinite loops, while recursive solutions are not.
   - D. Recursion and iteration are different techniques with their own strengths and weaknesses, although they can be used interchangeably.
4. What is the primary advantage of using recursion in programming?
   - A. Recursion allows for more efficient memory utilization.
   - B. Recursive implementations are always simpler because they do not use loops.
   - C. Recursion can solve complex problems concisely by breaking them down into smaller instances.
   - D. Recursion improves the speed of program execution.
   - E. Recursive solutions are easier to debug.
5. What is a potential drawback of using recursion in programming?
   - A. Recursion can lead to infinite function calls and stack overflow errors.
   - B. Recursion is only applicable to mathematical calculations.
   - C. Recursion can be challenging with some data structures, like dictionaries.
   - D. Using recursion makes the code more difficult to read and understand.
   - E. Both A and C

6. Which of the following is a correct recursive implementation of the power function?

   A.
   ```
   def power(x, n):
       if n == 0:
           return 1
       return x * power(x, n - 1)
   ```

   B.
   ```
   def power(x, n):
       if n == 1:
           return x
       else:
           return x * power(x, n - 1)
   ```

   C.
   ```
   def power(x, n):
       return x ** n
   ```

   D.
   ```
   def power(x, n):
       if n == 0:
           return 1
       else:
           return power(x, n) * x
   ```

7. What does the term “recursion depth” refer to?

   A. The number of recursive calls made by a function.
   B. The depth of nested loops in a recursive algorithm.
   C. The total number of lines in a recursive function.
   D. The level of indentation in a recursive function.

8. Under what circumstances is it opportune to use recursion in programming?

   A. When the problem can be naturally divided into smaller instances of the same problem, and requires a parameter that can be minimized or truncated
   B. Only when the problem is impossible or too complex to solve iteratively.
   C. When aiming for the most memory-efficient solution due to the inherent nature of recursive algorithms.
   D. Whenever recursion is available as an option, as it often simplifies code and enhances maintainability.

9. What is the correct recursive implementation for a function that takes two ordered strings as input and returns their (ascending) ordered concatenation?

   A.
```
def ordered_concat(str1, str2):
    if not str1:
        return str2
    elif not str2:
        return str1
    else:
        return ordered_concat(str1[1:], str2[1:]) + max(str1[0], str2[0])
```

   B.
```
def ordered_concat(str1, str2):
    if len(str1) == 0:
        return str2
    elif len(str2) == 0:
        return str1
    else:
        return ordered_concat(str1[1:], str2[1:]) + min(str1[0], str2[0])
```

   C.
```
def ordered_concat(str1, str2):
    if not str1:
        return str2
    elif not str2:
        return str1
    elif str1[0] < str2[0]:
        return str1[0] + ordered_concat(str1[1:], str2)
    else:
        return str2[0] + ordered_concat(str1, str2[1:])
```

   D.
```
def ordered_concat(str1, str2):
    if len(str1) == 0:
        return str2
    elif len(str2) == 0:
        return str1
    elif str1[0] > str2[0]:
        return max(str1[0], str2[0]) + ordered_concat(str1[1:], str2)
    else:
        return str2[0] + ordered_concat(str1, str2[1:])
```

2	3	4	5	6	7	8	9

Page total: ________/16

The following 8 questions require you to trace the behavior of some Python code and identify the output of that code. For each question, write the output for the code segment on the provided line.

10. (3 points)

```
def coraline(wybie):
    if wybie <= 0:
        return
    print(wybie, end = " ")
    coraline(wybie - 2)
    print(wybie, end = " ")

print(coraline(5))
```

11. (3 points)

```
def lulu(jerry):
    if not jerry:
        return 0
    elif jerry[0] in "aeiouAEIOU":
        return 1 + lulu(jerry[1:])
    else:
        return lulu(jerry[1:])

print(lulu("Buttercup"))
```

12. (3 points)

```
def fanfiction(trope):
    if len(trope) == 0:
        return ""
    else:
        return fanfiction(trope[1:]) + trope[0]

print(fanfiction("revol2ymene"))
```

Page total: ________/9

13. (3 points)

```
def peanuts(snoopy, woodstock):
    if woodstock == 0:
        return snoopy
    else:
        return peanuts(woodstock, snoopy % woodstock)

print(peanuts(15, 10), peanuts(12, 18), peanuts(49, 77))
```

14. (3 points)

```
def mystery(gang):
    clues = []
    for who in gang:
        if type(who) == list:
            clues.extend(mystery(who))
        else:
            clues.append(who)
    return clues

inc = ["scooby", ["shaggy", ["daphne", "velma"]], ["fred"]]
print(mystery(inc))
```

15. (3 points)

```
def girlPower( arr, l, r, x):
   if r < l:
      return -1
   if arr[l] == x:
      return l
   if arr[r] == x:
      return r
   return girlPower(arr, l+1, r-1, x)

princesses = ["cinderella", "snowWhite", "moana", "jasmine", "aurora", \
            "merida", "anna", "elsa"]
print(girlPower(princesses, 0, len(princesses) - 1, princesses[3]))
```

Page total: ________/9

16. (3 points)

```
def spookyMath(scaryNums):
    if scaryNums == []:
        return 0
    elif scaryNums[0] % 2 == 0:
        return -scaryNums[0] + spookyMath(scaryNums[1:])
    else:
        return scaryNums[0] + spookyMath(scaryNums[1:])

scaryNums = [7, -42, 17, -99, 2, -8]
print(spookyMath(scaryNums))
```

17. (3 points)

```
def toyStory(woody, buzz):
    if buzz == 0:
        return 1
    elif buzz % 2 == 0:
        jessie = toyStory(woody, buzz // 2)
        return jessie * jessie
    else:
        return woody * toyStory(woody, buzz - 1)

print(toyStory(3, 4), toyStory(4, 3))
```

Page total: ________/6