Quiz 6

Due May 11 at 1:59am **Points** 24 **Questions** 6
Available until May 21 at 1:59am **Time Limit** 15 Minutes
Allowed Attempts 2

Instructions

More recursion

The questions are all multiple choice or true/false. You may take the quiz twice, with 15 minutes for each attempt. Questions may differ between attempts. The highest score of the two attempts will be considered. You can only view the results once, immediately after finishing an attempt.

This is an open book test, however, you may not share the questions/answers of your quiz with others.

Take the Quiz Again

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	9 minutes	24 out of 24

ⓘ Correct answers are hidden.

Score for this attempt: **24** out of 24
Submitted May 6 at 6:22pm
This attempt took 9 minutes.

Question 1 — 4 / 4 pts

Given this definition:

```
def reverse_str(st, pos):
    """
    Returns a new string that is the reverse of st
    When first called, the pos parameter must be zero
    """
```

```
    if pos == len(st):
        return ""

    return reverse_str(st, pos+1) + st[pos]
```

If you were to make a call like so:

```
reverse_str("apples", 0)
```

What would be the value of *pos* when the base case is reached?

- 0
- 7
- 'apples'
- 6

Question 2

4 / 4 pts

Memoization avoids recomputing subproblems by keeping a record of solutions to subproblems that have already been solved.

- True
- False

Question 3

4 / 4 pts

How should one go about writing a recursive function?

Have the function call itself, and use a try...except clause to terminate the function when a memory error occurs.

Write the function using a for loop first, then try to replace each iteration with a recursive call.

Just insert a call to the function inside its definition and see what happens.

Figure out how to break the problem down into smaller problems of the same type, so that the base case is eventually reached.

Question 4

4 / 4 pts

How does the *reverse_str* function improve the following program?

```
def rec_reverse_str(st, pos):
    """
    Returns a new string that is the reverse of st
    When first called, the pos parameter must be zero
    """
    if pos == len(st):
        return ""

    return rec_reverse_str(st, pos+1) + st[pos]

def reverse_str(st):
    """
    Calls recursive reverse_str function with zero for the second para
meter
    """
    return rec_reverse_str(st, 0)
```

It allows the user to call reverse_st without having to add the initial 0 argument.

It adds lines to the program, which looks more professional.

- It checks to make sure the argument passed to rec_reverse_str is a non-empty string.
- It reduces the time complexity of the rec_reverse_str function.

Question 5

4 / 4 pts

Given this definition:

```
def reverse_str(st, pos=0, rev=""):
    """
    Returns a new string that is the reverse of st
    When first called, the pos parameter must be zero and
    the rev parameter must be "" (the empty string)
    """
    if pos == len(st):
        return rev

    return reverse_str(st, pos+1, st[pos] + rev)
```

If you were to call the function like so:

```
reverse_str("bananas")
```

What would be the value of "rev" when the base case is reached?

- s
- 'bananas'
- 'sananab'
- 7

Question 6

4 / 4 pts

When writing recursive functions to solve problems, what can you do to make them efficient?

- Use as few variables as possible.
- Have the base case precede the recursive call in the function definition.
- Use as many helper functions as possible.
- Avoid recomputing subproblems.

Quiz Score: **24** out of 24