Sequences Worksheet II
CS51 - Fall 2026
ANSWER KEY

Practice Problem 1 - Sorting strings

```
def compare_class(class_number):
    if class_number < 'cs51':
        print(class_number, 'comes before cs51.')
    elif class_number > 'cs51':
        print(class_number, 'comes after s51.')
    else:
        print('Hello, cs51')
```

What will happen if you call `compare_class('cs50')` ?

```
cs50 comes before cs51.
```

What will happen if you call `compare_class('CS62')`?

```
CS62 comes before cs51.
```

Practice Problem 2 - String functions

Define a function `str_odd_indices` that takes one parameter `s` (a string) and returns a string comprised of only the odd indexed characters of `s`. For example, `str_odd_indices('hello!')` would return `'el!'`.

Five options in increasing verbosity:

```
def str_odd_indices(s):
    return s[1::2]
```

```
def str_odd_indices(s):
    result = ''
    for i in range(1, len(s), 2):
        result += s[i]
    return result
```

```
def str_odd_indices(s):
    result = ''
    for i, char in enumerate(s):
        if i % 2 == 1:
            result += char
    return result
```

```
def str_odd_indices(s):
    result = ''
    i = 0
    for ch in s:
        if i % 2 == 1:
            result += ch
        i += 1
    return result
```

```
def str_odd_indices(s):
    result = ‘’
    for i in range(len(s)):
        if i % 2 == 1:
            result += s[i]
    return result
```

Practice Problem 3 - More string functions

Define a function `find_char` that takes two parameters, a string `s` and a character `c` and returns the index of the first instance of that character. If that character does not appear in the string, it returns -1.

For example:

```
find_char(‘hello’, ‘h’) == 0
find_char(‘hello’, ‘l’) == 2
find_char(‘hello’, ‘a’) == -1
```

Four options in increasing verbosity:

```
def find_char(s, c):
    return s.find(c)
```

```
def find_char(s, c):
    for i, ch in enumerate(s):
        if ch == c:
            return i
    return -1
```

```
def find_char(s, c):
    i = 0
    for ch in s:
        if ch == c:
            return i
        i += 1
    return -1
```

```
def find_char(s, c):
    for i in range(len(s)):
       if s[i] == c:
          return i
    return -1
```

Practice Problem 4 - Recursion and strings

Write a recursive function called `reverse` that takes a string as a parameter and reverses the string.

```
def reverse(st):
    if st == ‘’:
        return ‘’
    else:
        return reverse(st[1:]) + st[0]
```

Practice Problem 5 - Recursion and lists

Write a recursive Python function called `rec_linear_search` that takes as input a list, an element, and an index that has a default value of 0, and returns the index of the first encounter of the element in the list, if it exists, or -1 if it does not.

```
def rec_linear_search(lst, element, index=0):
    if index == len(lst):
       return -1
    elif lst[index] == element:
        return index
    else:
        return rec_linear_search(lst, element, index+1)
```

Practice Problem 6 - Mystery function

What does this mystery function do?

```
def rec_mystery(l):
 if len(l) == 1:
    return l[0]
 else:
     m = rec_mystery(l[1:])
     if m > l[0]:
         return m
     else:
         return l[0]
```

Recursive function that returns the max element in the provided sequence.

Practice Problem 7 - Loop invariants

Given a list `my_list` (of size `n`) of numbers, write an iterative function `sum_of_list_numbers` that calculates the sum of the numbers in `my_list`.

```
def sum_of_list_numbers(my_list):
    answer = 0
    for i in range(len(my_list)):
     answer += my_list[i]
    return answer
```

1. What are the pre-and post-conditions? A list of n numbers and a number that is equal to the sum of the n numbers in `my_list`
2. What is a good loop invariant? At the start of iteration `i` of the loop, the variable answer should contain the sum of the numbers from the sublist `my_list[0:i]`.
3. Use loop invariants to prove that your function works correctly.
    - *Initialization:* At the start of the first loop, the sum of the numbers in an empty slice is 0, and this is what `answer` has been set to.
    - *Maintenance:* Assume that the loop invariant holds at the start of iteration `i` . Then it must be that answer contains the sum of numbers in slice `my_list[0:i]`. In the body of the loop, we add `my_list[i]` to answer. Therefore, at the end of iteration and i and before `i+1` iteration begins, answer will contain the sum of numbers in `my_list[0:i+1]`, which is what we needed to prove.
    - *Post-condition:* When the loop terminates, `i` should be equal to `n` and the loop invariant gives that answer contains the sum of all numbers in slice `my_list[0:n]` which is equal to list. Thus, we will indeed get the sum of all numbers in `my_list`.

- *Termination:* `i` increases by 1 in every iteration and ranges from 0 to maximum `n==len(my_list)`. The for loop will terminate in a finite number of steps.

Practice Problem 8 - Aliasing

You are given the function

```
def augment_twice(a_list, val):
    a_list.append(val)
    a_list.append(val)
```

What happens to `numbers` after executing the following code?

```
numbers = [1,2,3]
augment_twice(numbers, 47)
```

```
[1, 2, 3, 47, 47]
```

What if we were to change the function to this and execute the same code?

```
def augment_twice(a_list, val):
    a_list = a_list + [val, val]
```

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