CS 88
Fall 2020

Sample Exam.

Midterm

INSTRUCTIONS

This is your exam. Complete it either at exam.cs61a.org or, if that doesn't work, by emailing course staff with your solutions before the exam deadline.

This exam is intended for the student with email address `<EMAILADDRESS>`. If this is not your email address, notify course staff immediately, as each exam is different. Do not distribute this exam PDF even after the exam ends, as some students may be taking the exam in a different time zone.

For questions with **circular bubbles**, you should select exactly *one* choice.

○ You must choose either this option

○ Or this one, but not both!

For questions with **square checkboxes**, you may select *multiple* choices.

☐ You could select this choice.

☐ You could select this one too!

You may start your exam now. Your exam is due at <DEADLINE> Pacific Time. Go to the next page to begin.

Preliminaries

You can complete and submit these questions before the exam starts.

0.0.1 Basic Directions

- You have 2 hours, 120 minutes to complete the exam.
- You **must not** collaborate with anyone inside or outside of CS88.
- You may use the internet, the CS88 site and all it's resources,
- *However*, you **must not** directly search for a question or post questions online.
- You may search for generic Python concepts.
- You may use your Terminal and Python Tutor.
  - However, these are more strict about syntax! The exam is designed to be completed *without* these tools, and using them may take up some time. Be mindful of how long you spend on each question.
- At this point you should have started your Zoom / screen recording. If something happens during the exam, focus on the exam!
- Do not spend more than a few minutes dealing with proctoring.
- Your task is to show us how much you've learned, not to mess with technology.

(a) What is your full name?

(b) What is your student ID number?

(c) What is your Berkeley email address?

1. **What Made Python Print That?**

   With the given lines of code and results, fill in the blanks so that the results properly appear.

   (a)
```
>>> vals = [10, 50, 60, 20, 40, 30]
>>> y = [ _______(1)________ for ______(2)______ in range(len(vals)) ]
>>> y
[10, 51, 62, 23, 44, 35]
```

   i. (2.0 pt) Blank (1)

   ii. (2.0 pt) Blank (2)

(b) What are three (out of 4 total) possible values for the variable `secret_num` so that the return value of the above function call is 36? Briefly explain how you approached this problem (answers without a proper explanation will not receive credit).

```
>>> def mystery1(n):
...     if n > 20:
...             return n - 15
...     else:
...             return n * 3
>>> mystery1(mystery1(secret_num))
36
```

i. (2.0 pt) List out 3 (out of the 4 possible) values for the variable `secret_num`.

ii. (1.0 pt) Provide an explanation for how you got your answers. Answers with no explanation will not be given credit.

(c) Assume `lst` is a list of 3 integers.

```
>>> lst = [_____, _____, ______]
>>> (lst[2] - lst[0]) or (lst[1] < 15) or (lst[0] % 10 != 2) or "hi"
```

i. **(1.0 pt)** If `lst[1]` was 33, the expression would _________ evaluate to `True`.

◯ Always

◯ Sometimes

◯ Never

ii. **(1.0 pt)** If `lst[0]` was 21, the above expression would _________ evaluate to `False`.

◯ Always

◯ Sometimes

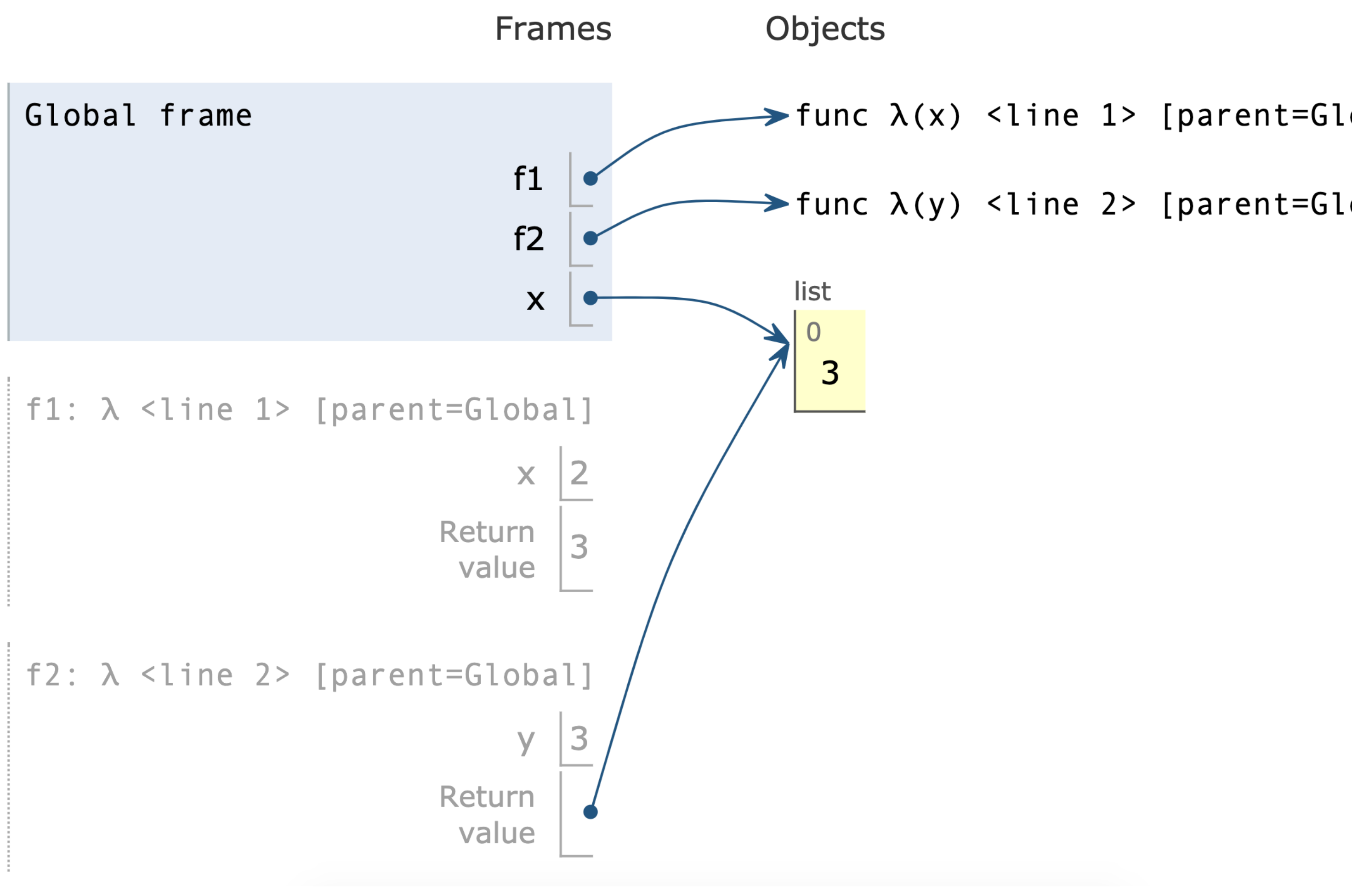
◯ Never

iii. **(2.0 pt)** Fill in the list lst with 3 integers such that the expression on the second line in the original question evaluates to "hi". Express your answers as `[ ... , ... , ...]`. Briefly explain how you approached this problem (answers without a proper explanation will not receive credit).

2. Chi-pot-el

You are given the environment diagram for each question. Please fill in the blank lines so that the last step of the environment diagram will look exactly the same as the diagram given. All lines must be filled in and cannot be left blank. There is potentially more than one way to do a given problem, but all blanks must be used.

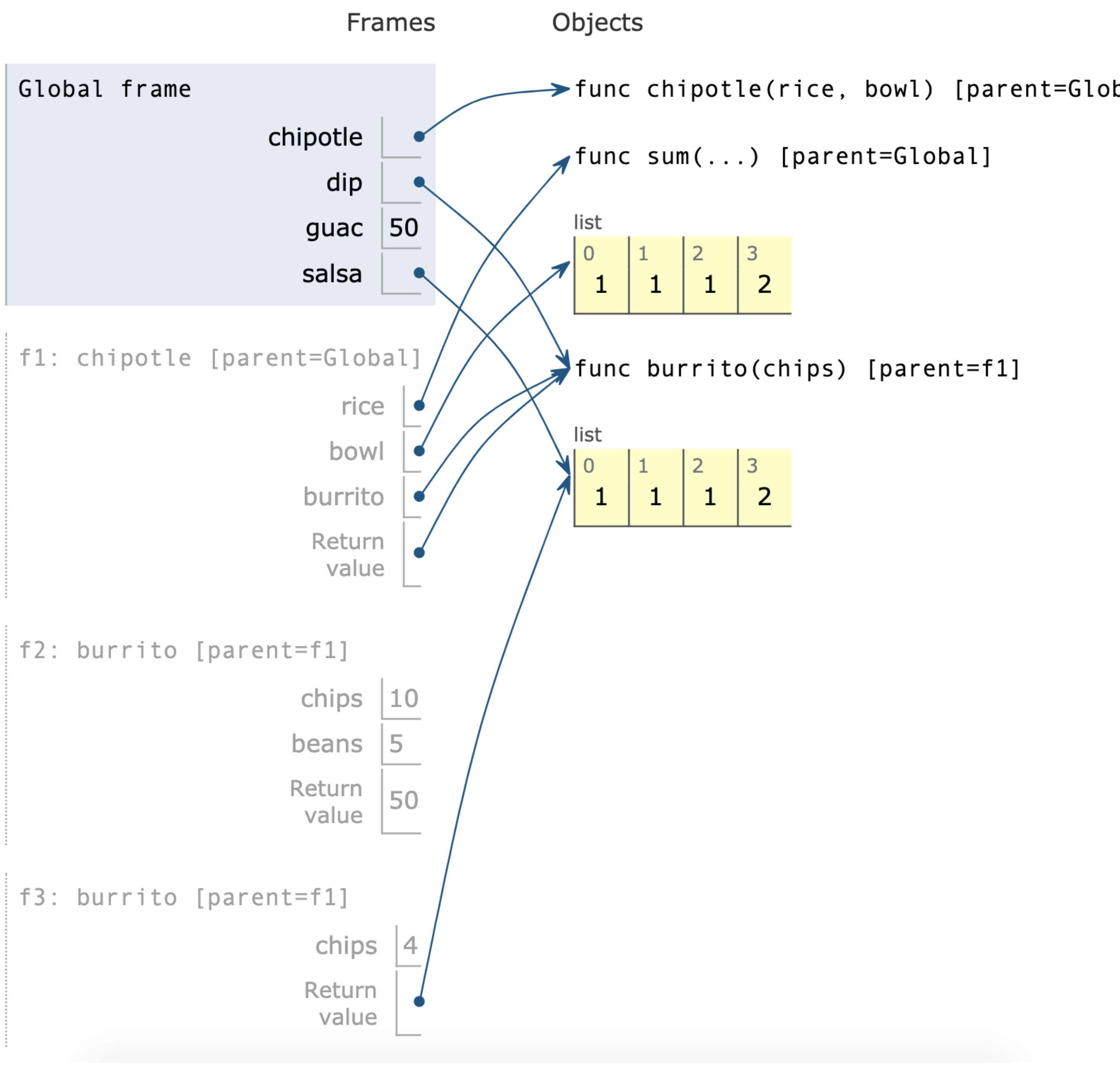
(a) (2.0 pt)


Frames
Objects
Global frame
f1
f2
x
func λ(x) <line 1> [parent=Gl
func λ(y) <line 2> [parent=Gl
list
0
3
f1: λ <line 1> [parent=Global]
x 2
Return value 3
f2: λ <line 2> [parent=Global]
y 3
Return value

environment diagram for 2.1

```
f1 = lambda x: __________
f2 = lambda y: __________
x = f2(f1(2))
```

Write the lines of code so that when run will result in the given environment diagram.



environment diagram for 2.2

(b) (3.0 pt)

```
def chipotle(______, ______):
    ___________________
    def burrito(chips):
        if chips > 6:
            beans = ________ ( ________ )
            return beans * ________
        else:
            return _________
    return burrito

dip = chipotle(sum, [1, 1, 1])
guac = dip(10)
salsa = dip(4)
```

Write the lines of code so that when run will result in the given environment diagram.

3. (6.0 points) I've Had enHOF of HOFs

In this question you will be building a function that checks if calling 3 different functions on the same input will yield the same result.

(a) **i.** **(4.0 pt)** The `comparer` function takes in a number `x` and returns a HOF. This HOF returns another function that takes in the first input function and returns a second function. This second function takes in a second input function and returns a third function. The third function takes in a third input function and returns True if all 3 input functions return the same value when called on `x`.

Fill out the skeleton code for the `comparer` function.

```
def comparer(x):
    """
    >>> f1 = lambda x: 2*x
    >>> f2 = lambda x: x**2
    >>> f3 = lambda x: x + 2
    >>> comp = comparer(2)
    >>> h1 = comp(f1)
    >>> h2 = h1(f2)
    >>> h2(f3)
    True
    >>> comp = comparer(10)
    >>> h1 = comp(f1)
    >>> h2 = h1(f2)
    >>> h2(f3)
    False
    """
    def func1(f1):
        def func2(f2):
           def func3(f3):
                return f2(x) == f3(x) ____________________________
            return ______________
        return ______________
    return ______________
```

Write the *completed* `comparer` function below. You may not add new lines.

ii. **(4.0 pt)** The `comparer` function takes in a number `z` and returns a HOF. This HOF returns another function that takes in the first input function and returns a second function. This second function takes in a second input function and returns a third function. The third function takes in a third input function and returns True if all 3 input functions return the same value when called on `z`.

Fill out the skeleton code for the `comparer` function.

```
def comparer(z):
    """
    >>> f1 = lambda a: 2*a
    >>> f2 = lambda b: b**2
    >>> f3 = lambda c: c + 2
    >>> comp = comparer(2)
    >>> h1 = comp(f1)
    >>> h2 = h1(f2)
    >>> h2(f3)
    True
    >>> comp = comparer(10)
    >>> h1 = comp(f1)
    >>> h2 = h1(f2)
    >>> h2(f3)
    False
    """
    def hof1(h1):
        def hof2(h2):
           def hof3(h3):
                return h2(z) == h3(z) ____________________________
            return ______________
        return ______________
    return ______________
```

Write the *completed* `comparer` function below. You may not add new lines.

(b) (2.0 pt) Complete the return statement in one line using lambda expressions.

```
def comparer(x):
    return __________________________________________________
```

4. **(6.0 points) Subset Summing**

   Given a list and an index, find the sum of all the summations of each subset of the list starting from the given index. For example, `subsetSum([1, 2, 3], 1)` would return 7. This is because the two subsets of the list starting at index 1 are [2] and [2, 3]. Then summing the sums of each subset we get $2 + 5 = 7$. You can assume that a valid index will always be passed in.

   Use the provided lines of code to fill in the body of the function. They appear out of order and without any indentation. You may not use an additional lines. You should fill in the blanks such that the provided code makes sense.

(a) (6.0 pt) Function Definiton:

```
def subsetSum(lst, i):
    """
    >>> subsetSum([1, 2, 3], 1)
    7
    >>> subsetSum([1], 0)
    1
    >>> subsetSum([1, 2, 3, 4, 5], 2) # the subsets are [3], [3, 4], and [3, 4, 5]
    22
    """
    """Your Solution Here"""
```

Lines to Use:

```
--------------------------
total = 0
currentSum = sum( ____________ )
return _________
for j in range( ________ + 1, ________ + 1):
```

Fill in the entire `subsetSum` function below. (You must use the lines above).

(b) (6.0 pt) Function Definiton:

```
def subsetSum(nums, x):
    """
    >>> subsetSum([1, 2, 3], 1)
    7
    >>> subsetSum([1], 0)
    1
    >>> subsetSum([1, 2, 3, 4, 5], 2) # the subsets are [3], [3, 4], and [3, 4, 5]
    22
    """
    """Your Solution Here"""
```

Lines to Use:

```
__________________________
result = 0
total = sum( ____________ )
return _________
for i in range( ________ + 1, ________ + 1):
```

Fill in the entire `subsetSum` function below. (You must use the lines above).

5. (6.0 points) Eat, Sleep, Debug, Repeat

You are trying to make a function tester which takes in an integer paramn, a list of functions fnlst, and a conditional function (a one argument function that returns a boolean) condition. The function tester runs each of the functions from fnlst on the integer paramn, and returns a list of all of the values for which condition returns `True`. However, as you are programming you find that your program is buggy. Find the 3 bugs in the following program. The line numbers are included for easy reference. They are *not* one of the bugs. (You may identify the bugs in any order.)

```
def tester(paramn, fnlst, condparam):
    """
    >>> add_one = lambda x: x + 1
    >>> negate = lambda x: -x
    >>> double = lambda x: x * 2
    >>> is_positive = lambda x: x > 0
    >>> tester(88, [add_one, negate, double], is_positive)
    [89, 176] #-88 is not included because it is negative
    """
1.    new_list = []
2.    for fn in fn_list:
3.        i = fn(n)
4.        if cond_param(n):
5.            new_list + i
6.        else:
7.            return new_list
```

(a) (2.0 pt) Describe a bug and explain why it is a bug.

(b) (2.0 pt) Describe a bug and explain why it is a bug.

(c) (2.0 pt) Describe a bug and explain why it is a bug.

6. (6.0 points) DDDuplicate EEElements

In this question you will be building a recursive function that, given a list and a non-negative integer count, will return a new list with count duplicates of each item in list.

Hint: `['a'] * 3` will return `['a', 'a', 'a']`

```
def duplicateElements(lst, count):
    """
    >>> duplicateElements([1, 2, 3, 4], 3)
    [1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4]
    >>> duplicateElements(['a', 'b', 'c'], 2)
    ['a', 'a', 'b', 'b', 'c', 'c']
    """
    if ________:
        return ________________________

    first = ______________________________
    rest = duplicateElements(______, ______)

    return ______________________________
```

(a) (6.0 pt) Write the entire completed `duplicateElements` function below. You may not add new lines.

7. The FinaLIST

You are in the finale of an exciting game called Treasure. You're given a board (represented as a list `lst`) that contains the amount of gold you can collect at each index, and two starting indices that you are considering (`start_idx_1` and `start_idx_2`).

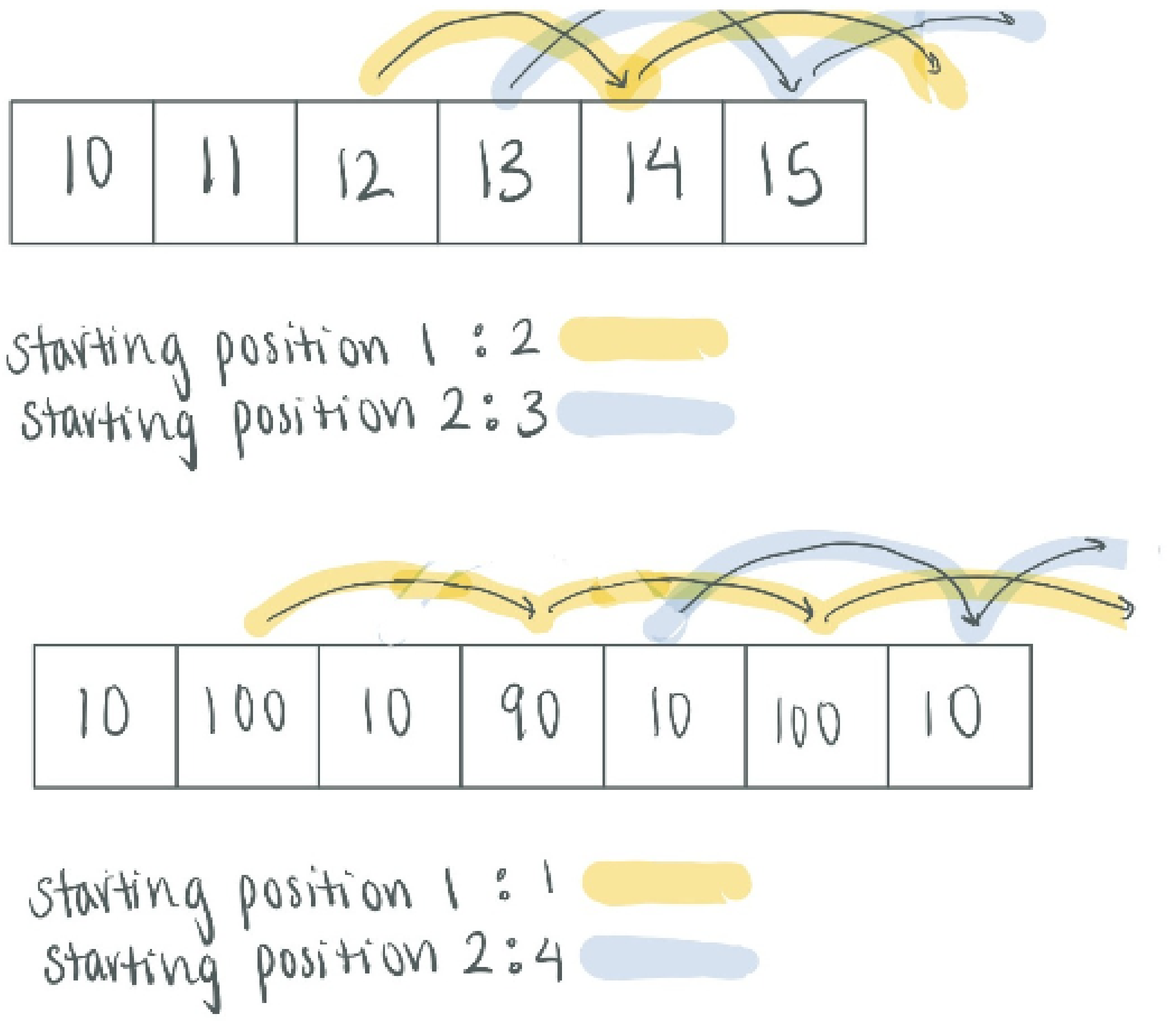
After choosing a starting index, you must move towards the right of the board to collect gold, skipping one spot every time, (see the graphic below). Find which of the two starting indices will earn you more treasure.

Your final return value should be a 2 element list containing the better starting index between the two options and the amount of treasure you would accumulate starting from there. You can assume the input list only contains positive numbers.

Use this graphic to better understand the doctests.

```
def better_starting_position(lst, start_idx_1, start_idx_2):
    """
    >>> board1 = [10, 11, 12, 13, 14, 15]
    >>> better_starting_position(board1, 2, 3)
    [3, 28]
    >>> board2 = [10, 100, 10, 90, 10, 100, 10]
    >>> better_starting_position(board2, 1, 4)
    [1, 290]
    """
```

(a) (8.0 pt)



question 4 exmplation hopping over elements

8. Cycloaddition

Create a function `cycle_add` that takes in a 2-D list of integers in which the int lists can be of different lengths. It returns a list of the result of summing up the values of each of the lists elementwise, cycling if necessary (if the list is too short). For example `cycle_add([[1,2], [3,5,7]])` will return `[4, 7, 8]`, the 4 is the result of 1 + 3, the 7 is the result of 2 + 5, 8 is the result of 1 + 7 (notice how we cycled back to the first value of the first list to get 1).

```
def cycle_add(lsts):
    """
    >>> cycle_add([[1,2,3], [1]])
    [2, 3, 4] # [1+1, 2+1, 3+1]
    >>> cycle_add([[1,2], [3,5,7]])
    [4, 7, 8] # [1+3, 2+5, 1+7]
    >>> cycle_add([[4,2,0], [1,2], [2]])
    [7, 6, 3] # [4+1+2, 2+2+2, 0+1+2]
    """
    new_list = []
    longest = __________________
    for ____________________:
        value = 0
        for __________________:
            index = _______________ % len(lst)
            value += ________________
        _______________
    return new_list
```

(a) **(10.0 pt)** Write the `cycle_add` function below. Do not add any new lines.

No more questions.