

### INSTRUCTIONS

This is your exam. Complete it either at [exam.cs61a.org](http://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. Reference Links**

**Clarifications Document:** <https://docs.google.com/document/d/14FS5w-2k1VkmKhMbRw9jjjArQWaFLssa21JzOm6ALM>

**Reference Sheet:** <https://drive.google.com/file/d/1bTRKCwtGSo4MMd42bv7YGYgyEyDUOnBq/view?usp=sharing>

(There is nothing to submit for question 1.)

**2. (2.0 points) Conceptual Questions**

**(a) (1.0 pt)** What are the two different ways a function can be a higher order function?



(b) (1.0 pt) What is a recursive function and how do you avoid infinite recursion?

A large empty rectangular box with a thin black border, intended for the student to write their answer to the question. The box is currently blank.

**3. (7.0 points) What Made Python Print That?**

Given the following lines of code and respective output, fill in the blanks to produce the desired result.

```
(a) >>> lst = [_____1_____]  
>>> len(lst) == len(lst[1]) and (not lst[2]) and lst[0] * 2  
10
```

Create a list `lst` such that the following expression evaluates to 10.

**i. (3.0 pt) Blank (1)**

```
(b) >>> def f1(m):
...     def f2(lst):
...         c = 0
...         for i in range(len(lst) - 1):
...             if lst[i+1] - lst[i] == m:
...                 c += 1
...         return c
...     return f2

>>> f1(____)(_____)
```

4

Write an expression that would evaluate to 4 by using the above function and identifying what goes in the blanks.

**i. (2.0 pt)**

(c) Assuming `apple`, `orange`, and `pear` are variables with integer values and `salad` is a dictionary, answer the two following questions.

i. (1.0 pt) The statement below will Always/Sometimes/Never evaluate to a Truthy value if `orange - apple == 5`.

```
>>> (pear > orange) or (apple or orange)
```

- Always
- Sometimes
- Never

ii. (1.0 pt) The following expression will Always/Sometimes/Never evaluate to a Falsy value if the variable `pear` exists **at least once as a value** in the dictionary `salad`.

```
>>> len(list(salad.values())) == len(list(salad.keys())) and pear in salad
```

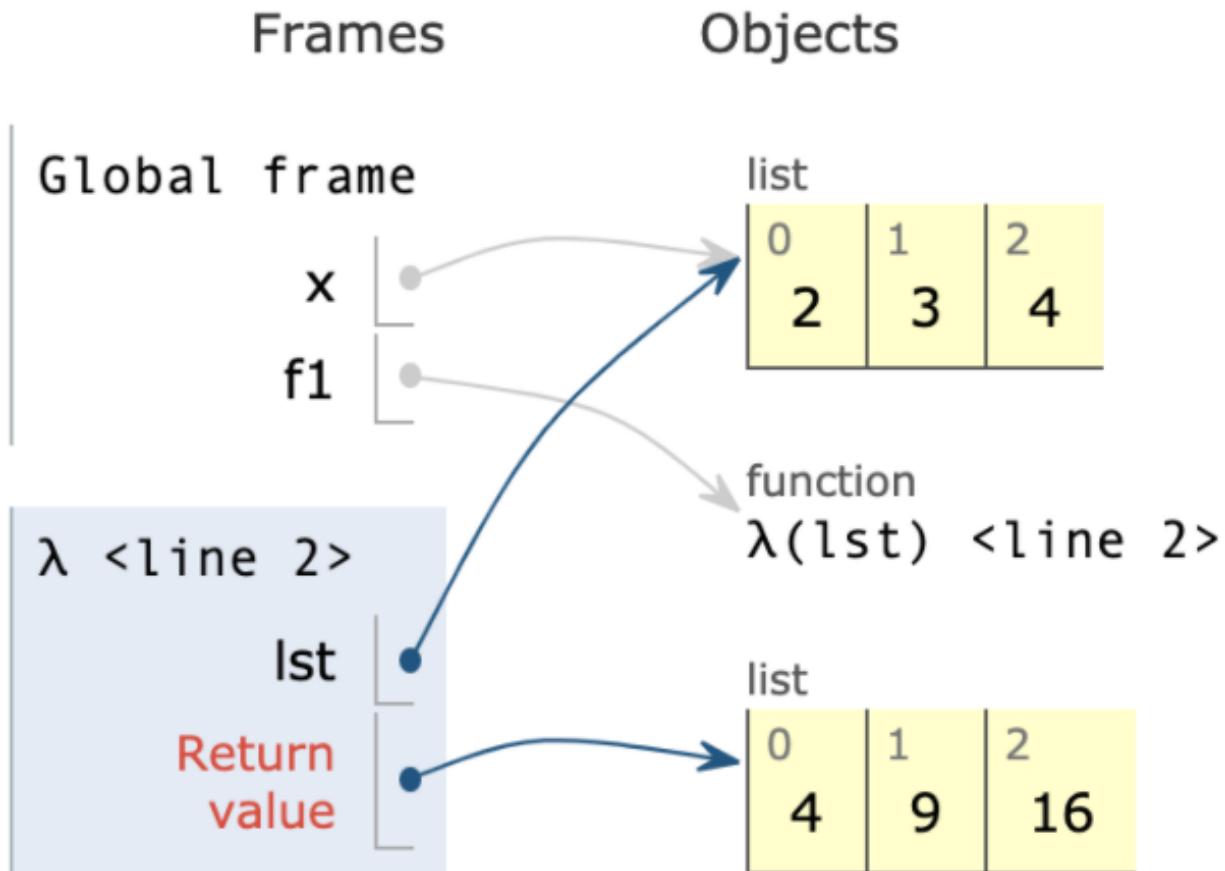
- Always
- Sometimes
- Never



**4. (9.0 points) Reverse Environment Diagram**

You are given the environment diagram for each question. Please fill in the blank lines so that the final 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) (3.0 pt)



environment diagram for 2.1

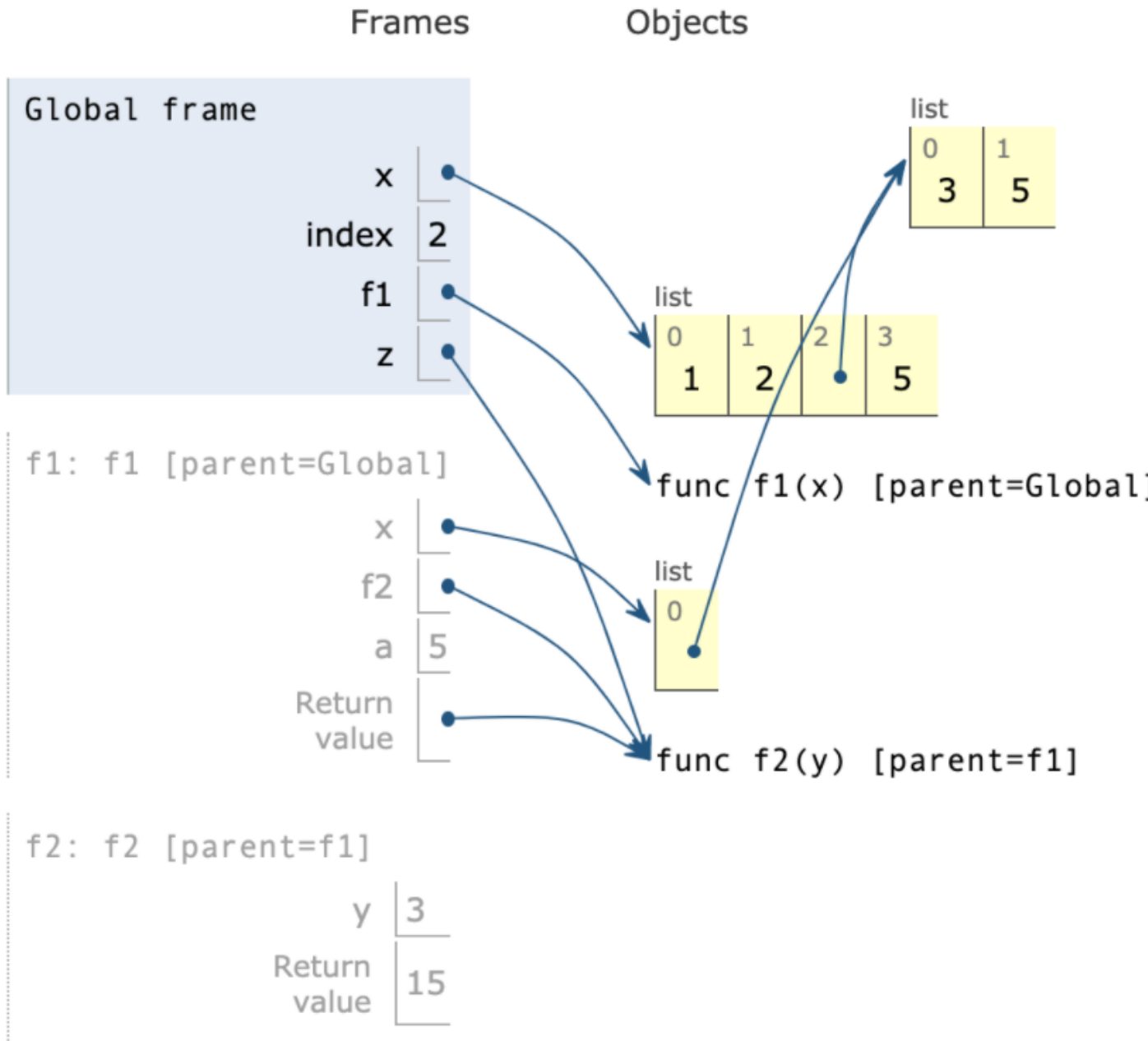
```

x = _____
f1 = lambda ___ : [____for ___ in____]
f1(x)

```

Copy and paste the skeleton code from above into the box below and fill in the blanks in the code so that when it is executed, it will result in the given environment diagram.





environment diagram for 2.2

(b) (6.0 pt)

```
x, index = [1, 2, [3, 4], 5], 2
```

```
def f1(x):  
    def f2(____):  
        return ____ * a  
    a = x.pop()  
    ___[___][1] = a  
    return _____
```

```
z = f1(x[index:])  
z(____)
```

Copy and paste the skeleton code from above into the box below and fill in the blanks in the code so that when it is executed, it will result in the given environment diagram.

**5. (7.0 points) The Ehrman Elevator**

- (a) i. (3.0 pt) One of the Unit 2 elevators has broken down yet again and all the code was lost! Fill in the function `elevator` so that it returns a list `order` that represents the order of the floors from the list `requests` that the elevator should visit. Each floor in the `requests` list is represented as an integer.

The `next_floor` function takes in the current floor `cur_floor` and a non-empty list `requests` that contains the floors that the elevator still must go to. `next_floor` returns an integer representing the next floor that the elevator should go to next and removes this integer from `requests`.

```
def elevator(cur_floor, requests, next_floor):
    """
    >>> def smallest_to_largest(cur_floor, requests):
    ...     smallest = min(requests)
    ...     requests.remove(smallest)
    ...     return smallest
    >>> elevator(4, [55, 2, 1, 3], smallest_to_largest)
    [1, 2, 3, 55]
    >>> def descending_only(cur_floor, requests):
    ...     largest = max(requests)
    ...     while largest > cur_floor:
    ...         requests.remove(largest)
    ...         largest = max(requests)
    ...     requests.remove(largest)
    ...     return largest
    >>> elevator(4, [55, 2, 1, 3], descending_only)
    [3, 2, 1]
    """
    order = []
    while _____(a)_____:
        cur_floor = _____(b)_____
        _____(c)_____
    return order
```

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





- (b) i. (4.0 pt) Now implement the `closest_floor` function that can be passed into the `elevator` function for the `next_floor` parameter. `closest_floor` returns the floor from the non-empty list `requests` that is closest to the current floor of the elevator `cur_floor`. The floor that is returned should be removed from `requests` within this function. Assume there are no ties for the closest floor.

For more information about the `min` function, you may refer to the documentation from Prolem 0.2 of the maps project.

```
def closest_floor(cur_floor, requests):
    """
    >>> requests = [1, 7]
    >>> closest_floor(6, requests)
    7
    >>> requests
    [1]
    >>> closest_floor(7, requests)
    1
    >>> elevator(4, [6, 10, 5, 1, 30], closest_floor)
    [5, 6, 10, 1, 30]
    """
    floor = min(____(a)____, key=lambda x:_____(b)_____)
    _____(d)_____
    return floor
```

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

**6. (7.0 points) Exaggerateeee**

Implement the `exaggerate` function that returns a string that is identical to the input string `phrase` except that every vowel in `phrase` is replaced by itself `repeat` times.

HINT: Recall that the operator `*` can be applied to strings (e.g. `'ab' * 2 = 'abab'` and `'c' * 3 = 'ccc'`).

**(a) (7.0 pt)**

```
def exaggerate(phrase, vowels, repeat):
    """
    >>> vowels = ['a', 'e', 'i', 'o', 'u']
    >>> exaggerate('nice', vowels, 3)
    'niiiiceee'
    >>> exaggerate("Woah so cool", vowels, 2)
    'Wooaah soo coool'
    """
    if _____:
        return _____
    else:
        cur = phrase[0]
        if _____:
            cur = _____
        rest = exaggerate(_____, _____, _____)
        return _____
```


**7. (10.0 points) Debugging**

Your friend wrote a function `product_slice` that takes in a `start` and `stop` value. It returns another function `slicer` that takes in a list and returns the product of all the numbers from `start`(inclusive) to `stop`(exclusive). However, it has exactly 3 distinct bugs.

```
(a) def product_slice(start, stop):  
    """  
    >>> lst = [1,2,3,4,5]  
    >>> product_slice(1,4)(lst) # 2 * 3 * 4  
    24  
    >>> product_slice(0,4)(lst) # 1 * 2 * 3 * 4  
    24  
    >>> product_slice(0,5)(lst) # 1 * 2 * 3 * 4 * 5  
    120  
    """  
    def slicer(lst):  
        output = 1  
        while start <= stop:  
            output = output * lst[start]  
            start += 1  
        return output  
    return slicer(lst)
```

- i. (6.0 pt) Identify the 3 unique bugs and explain how to fix each bug. After fixing all the bugs, the code should work as intended.

- (b) i. (4.0 pt) Write a working function for `product_slice`.



**8. (10.0 points) Shhh, I'm Booked**

Your boss has asked you to implement an interface that allows users to donate and borrow books from local libraries. You've started some work, and have already built out methods for two abstract data types representing **libraries** and **users**.

A **user** can donate books to a **library** and borrow books from the **library**, but there are some constraints that need to be enforced.

For a **user** to borrow a book from a **library**, they must not already have the book (if not, print "The user already owns this book.") and the **library** they are borrowing from must have that book available (if not, print "The library does not have this book.").

For a **user** to donate a book to a **library**, they must currently have that book (if not, print "The user does not have this book.") and the **library** should not already be at max capacity (if it is at maximum capacity, print "The library is already at max capacity."). The books are removed from the **user** and added to the **library** for a donation and added to the **user** and removed from the **library** for a borrow.

Implement the `borrow` and `donate` functions below to implement the desired behavior. You may find reading the doctests useful for understanding the behavior of these functions. Assume that the following functions of the ADTs are already defined for you (their implementation is hidden).

Note: not all the lines need to be used.

Library ADT Functions

```
def make_library(capacity):
    # creates library ADT
def get_capacity(lib):
    # returns this library's max capacity for books
def get_library_books(lib):
    # returns books currently in this library as a list
def add_book_for_lib(lib, book):
    # adds book to this library
def remove_book_for_lib(lib, book):
    # removes book from this library
```

User ADT Functions

```
def make_user(name, books):
    # creates user ADT
def get_user_books(user):
    # returns books this user currently has as a list
def add_book_for_user(user, book):
    # adds this book to user's current books
def remove_book_for_user(user, book):
    # removes this book from this user's current books
```

```
>>> doe = make_library(2)
>>> shelby = make_user("Shelby", ["The Firm", "Gone Girl", "The Client"])
>>> rachel = make_user("Rachel", ["The Kite Runner", "When Breath Becomes Air"])
>>> humbart = make_user("Humbart", ["The Firm", "The Book Thief"])
```

```
>>> donate(shelby, doe, "The Firm")
>>> donate(shelby, doe, "The Firm")
This user does not have this book.
>>> borrow(humbart, doe, "The Firm")
This user already owns this book.
```

```
>>> donate(rachel, doe, "The Kite Runner")
>>> donate(shelby, doe, "Gone Girl")
This library is already at max capacity.
```

```
>>> borrow(shelby, doe, "The Alchemist")  
This library does not have this book.
```

```
>>> borrow(humbart, doe, "The Kite Runner")  
>>> borrow(humbart, doe, "The Kite Runner")  
The library does not have this book.  
>>> donate(rachel, doe, "The Kite Runner")  
This user does not have this book.
```

(a) i. (5.0 pt)

```
def borrow(user, lib, book):  
    user_books = get_user_books(user)  
    library_books = get_library_books(lib)  
    -----  
    -----  
    -----  
    -----  
    -----  
    -----  
    -----  
    -----  
    -----
```

Implement the borrow function.

(b) i. (5.0 pt)

```
def donate(user, lib, book):  
    user_books = get_user_books(user)  
    library_books = get_library_books(lib)
```

```
-----  
-----  
-----  
-----  
-----  
-----  
-----  
-----
```

Implement the donate function.

**9. (10.0 points) Compress**

Write a function `compress` that takes in a list of integers. It returns a compressed version of the integers in the form of a list that contains two element lists where the first element is a number and the second element is the amount of times that number appears consecutively.

```
def compress(lst):  
    """  
    >>> lst = [1,1,1,0,0,1]  
    >>> compress(lst)  
    [[1, 3], [0, 2], [1, 1]]  
    # [1, 3] represents that there are 3 consecutive 1s  
    # [0, 2] following that, there are 2 consecutive 0s  
    # [1, 1] finally there is 1 consecutive 1s  
    >>> lst2 = [1,0,1,0]  
    >>> compress(lst2)  
    [[1, 1], [0, 1], [1, 1], [0, 1]]  
    >>> lst3 = [1,1,1,1,1]  
    >>> compress(lst3)  
    [[1, 5]]  
    """
```

```
-----  
-----  
-----  
-----  
-----  
-----  
-----  
-----  
-----
```



(a) (10.0 pt) Write the `compress` function below.

A large empty rectangular box with a thin black border, intended for the student to write the implementation of the `compress` function.

**No more questions.**