CS 88
Fall 2022

Computational Structures in Data Science

MIDTERM

INSTRUCTIONS

- **Do NOT open the exam until you are instructed to do so!**
- You **must not** collaborate with anyone inside or outside of C88C.
- You **must not** use any internet resources to answer the questions.
- If you are taking an online exam, 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.
- When a question specifies that you must rewrite the completed function, you should **not** recopy the doctests.
- The exam is closed book, closed computer, closed calculator, except your hand-written 8.5" x 11" cheat sheets of your own creation and the official C88C Reference Sheet

Full Name	
Student ID Number	
Official Berkeley Email (@berkeley.edu)	`CS88 In Person`
What room are you in?	
Name of the person to your left	
Name of the person to your right	
By my signature, I certify that all the work on this exam is my own, and I will not discuss it with anyone until exam session is over. **(please sign)**	

POLICIES & CLARIFICATIONS

- If you need to use the restroom, bring your phone and exam to the front of the room.
- For fill-in-the-blank coding problems, we will only grade work written in the provided blanks.
- Unless otherwise specified, you are allowed to reference functions defined in previous parts of the same question.
  - You must include all answers within the boxes.
  - **Online Exams: You may start your exam as soon as you are given the password.**
  - **You may have a digital version of the C88C Reference Sheet, or the PDF, but no other files.**

Exam Clarifications: https://tinyurl.com/clarifications-fa22 Reference Sheet: https://tinyurl.com/mt-reference

1. (3.0 points) What the HOF?

For each of the following scenarios, select the function that best provides a solution. Assume that the input list is a sequence of numbers.

(a) (0.5 pt) Return the product of all numbers in a list.

- ○ `map`
- ○ `filter`
- [✓ `reduce`]
- ○ None of these

(b) (0.5 pt) Return a sequence of the squares of all the elements in a list.

- [✓ `map`]
- ○ `filter`
- ○ `reduce`
- ○ None of these

(c) (0.5 pt) Return a sequence containing only the elements in a list that are greater than 10.

- ○ `map`
- [✓ `filter`]
- ○ `reduce`
- ○ None of these

(d) (0.5 pt) As a reminder, each of `map`, `filter`, and `reduce` takes in a function and a sequence as arguments. For each of the following, would it make the *most sense* to use the function as an input to `map`, `filter`, or `reduce`? All three answers are intended to be distinct (i.e. no two input functions will both correspond to the same list function).

```
def f(x):
    return len(x) < 5
```

- ○ `map`
- [✓ `filter`]
- ○ `reduce`
- ○ None of these

(e) (0.5 pt)

```
def g(x, y):
    return x + y
```

- ○ `map`
- ○ `filter`
- [✓ `reduce`]
- ○ None of these

(f) (0.5 pt)

```
def h(x):
    return x * 3
```

- [✓ `map`]
- ○ `filter`
- ○ `reduce`
- ○ None of these

2. (6.0 points) What Would Python Do (WWPD)

For each expression below, write the output displayed by the interactive Python interpreter when the expression is evaluated. The output may have multiple lines. If an error occurs, write “Error” (if any lines are displayed before the error, include those in your output). If a function is returned, write “Function”.

(a) (0.5 pt)

```
>>> 'young' and 'sweet' or 'seventeen'
```

```
'sweet'
```

(b) (0.5 pt)

```
>>> see = ['see', 'that', 'girl', 'watch', 'that', 'scene']
>>> see[1:3]
```

```
['that', 'girl']
```

(c) (1.0 pt)

```
>>> see[6]
```

```
Error
```

(d) (1.0 pt)

```
>>> digging = lambda x: lambda: 17 + y
>>> digging
```

```
Function
```

(e) (1.0 pt)

```
>>> digging()
```

```
Error
```

(f) (1.0 pt)

```
def having_the_time_of_your_life(dance, jive):
    if dance and jive:
        print(True)
    if dance > 0 and jive:
        print('dance')
    elif dance == 17:
        print('jive')
    else:
        print('QUEEN')
    print('queen')
```

```
>>> having_the_time_of_your_life(88, False)
```

```
QUEEN
queen
```

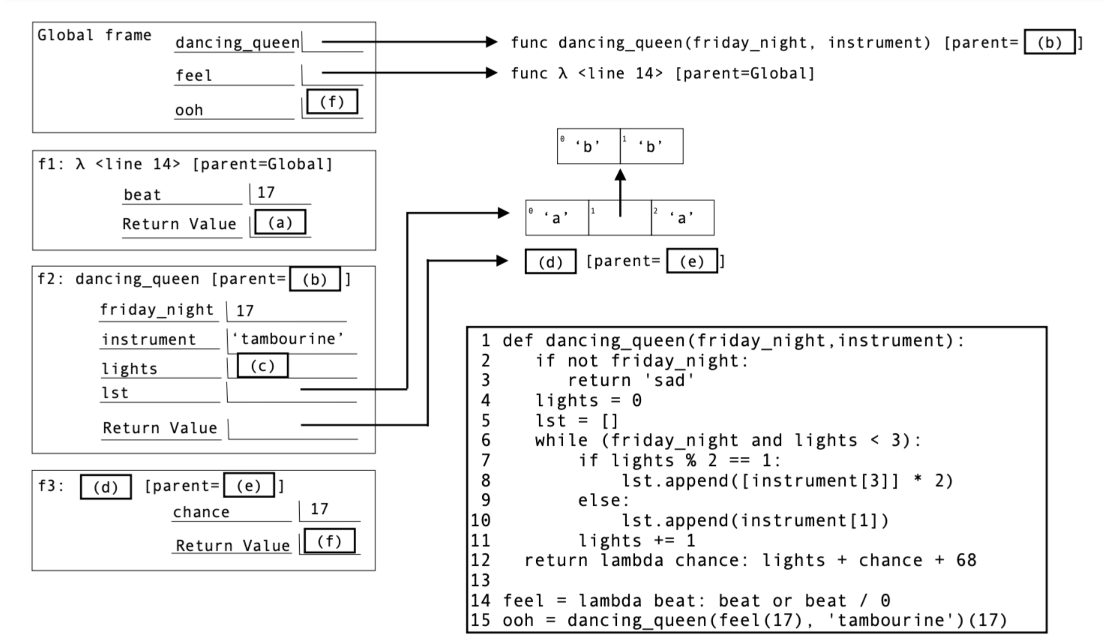
(g) (1.0 pt)

```
>>> having_the_time_of_your_life(17, not False)
```

```
True
dance
queen
```

3. (7.0 points) More Abba!

Fill in the blanks to complete the environment diagram. All the code used is in the box to the right, and the code runs to completion.



```
1  def dancing_queen(friday_night,instrument):
2      if not friday_night:
3          return 'sad'
4      lights = 0
5      lst = []
6      while (friday_night and lights < 3):
7          if lights % 2 == 1:
8              lst.append([instrument[3]] * 2)
9          else:
10             lst.append(instrument[1])
11         lights += 1
12     return lambda chance: lights + chance + 68
13
14 feel = lambda beat: beat or beat / 0
15 ooh = dancing_queen(feel(17), 'tambourine')(17)
```

(a) (1.0 pt) What is the return value of the lambda function in `f1` frame? (box a)

- [✓ 17]
- Error
- 'beat'
- beat

(b) (0.0 pt) !!IGNORE THIS QUESTION!!

- 17
- Error
- 'beat'
- beat

(c) (1.0 pt) What is the parent frame of the `dancing_queen` function? (box b)

- [✓ Global]
- f1
- f2
- f3

(d) (1.5 pt) What is the value of `lights` upon returning from the `f2` frame? (box c)

3

(e) (1.0 pt) What is the return value of the `f2` frame? (box d)

[✓ `func lambda <line 12>`]

○ `func dancing_queen`

○ `func lambda <line 14>`

(f) (1.0 pt) What is the parent frame of the `lambda` function in `f3`? (box e)

```
f2
```

(g) (1.5 pt) What is the value of `ooh` in the global frame when the environment diagram is complete? (box f)

```
88
```

4. (4.0 points) You Get Bug! And You Get Bug! And You Get A Bug!

In a previous homework, we saw a recursive version of `remove_last`, which returns a new list identical to the input list s but with the last element in the sequence that is equal to `x` removed. Here is the expected behavior:

```
>>> remove_last(1,[])
[]
>>> remove_last(1, [1])
[]
>>> remove_last(1, [1, 1])
[1]
>>> remove_last(1, [2, 1])
[2]
>>> remove_last(1, [3, 1, 2])
[3, 2]
>>> remove_last(1, [3, 1, 2, 1])
[3, 1, 2]
>>> remove_last(5, [3, 5, 2, 5, 11])
[3, 5, 2, 11]
```

Jessica tried to write an iterative version of this function. Here is her code:

```
def remove_last(x, s):
    new_list = []
    for elem in s:
        if elem != x:
            new_list.append(x)
    return new_list
```

Use this code to answer the following questions.

(a) (2.0 pt) Will this code execute the expected behavior?

Explanation: First, we create an empty list `new_list`. Then in the loop, we add every element in `s` that is not equal to `x` to `new_list`. At the end, we return `new_list`, which is now identical to `s` except without any elements that are equal to `x`. This is incorrect because we wanted a list that is identical to `s` except without the last element equal to `x`. For example, the expected output of `remove_last(1, [3, 1, 2, 1])` would be `[3, 1, 2]`, but our code would return `[3, 2]`.

- ○ Yes, this code correctly removes the last element of `s` that is equal to `x`.
- ○ No, this code is removing only the first element of `s` that is equal to `x`.
- [✓ No, this code is removing all elements of `s` that are equal to `x`.]
- ○ No, this code is removing elements of `s` that are not equal to `x`.
- ○ No, this code is wrong for a different reason not listed here.

(b) **(2.0 pt)** Amit also tried to write an iterative version of this function. Here is his code:

```
def remove_last(x, s):
    new_list = s.copy()
    i = 0
    while i < len(s):
        if s[i] == x:
            new_list.pop(i)
            return new_list
        i += 1
```

Will this code execute the expected behavior?

Explanation: First, we create the `new_list` by making a copy of `s`. Now, the original goal was to return a list that is identical to `s` except without the last element equal to `x`. However, this code starts from the beginning of the list and removes the first element equal to `x`, and then returns `new_list` with that first element removed. Here is a revised version of this code that correctly starts from the end of the list:

```
def remove_last(x, s):
    new_list = s.copy()
    i = len(s) - 1
    while i >= 0:
        if s[i] == x:
            new_list.pop(i)
            return new_list
        i -= 1
```

Note: "No, this code is wrong for a different reason" received partial credit. There should also be a second `return new_list` statement at the end of the function in the case that there are no elements in `s` that are equal to `x`. In that scenario, we would never reach the return statement inside of the loop because `s[i] == x` would be `False` for all `s[i]`. Thus, we would need to return `new_list` at the end.

- ○ Yes, this code correctly removes the last element of `s` that is equal to `x`.
- [✓ No, this code is removing only the first element of `s` that is equal to `x`.]
- ○ No, this code is removing all elements of `s` that are equal to `x`.
- ○ No, this code is removing elements of `s` that are not equal to `x`.
- ○ No, this code is wrong for a different reason not listed here.

5. (4.0 points) Sum Divisible Digits

Complete the function `sum_divisible_digits` that accepts two integers, `num` and `x`. The function returns the sum of the digits of `num` that are divisible by `x`. (Write you solution in the box provided.)

You can assume `x` will be greater than 0.

```
def sum_divisible_digits(num, x):
    """
    >>> sum_divisible_digits(93731, 3) # 9 + 3 + 3
    15
    >>> sum_divisible_digits(162, 1) # 1 + 6 + 2
    9
    """
    total = ______________
    ____________________:
        digit = _________________
        if _________________:
            _____________________
        _____________________
    return total
```

(a) (3.0 pt)

```
def sum_divisible_digits(num, x):
    total = 0
    while num:
        digit = num % 10
        if digit % x == 0:
            total += digit
        num = num // 10
    return total
```

6. (6.0 points) Do The n-step Fibonacci

Recall the Fibonacci sequence is a famous sequence of numbers: 0, 1, 1, 2, 3, 5,... We can define the sequenece: $F(n) = F(n-1) + F(n-2)$.

The Fibonacci sequence can be generalized to the n-step Fibonacci sequence, F_n. The first $n-1$ elements of the sequence are 0s, and the `n`th element is 1. The i th element is defined as

$$F_n(i) = F_n(i-1) + F_n(i-2) + ... + F_n(i-n)$$

The "original" Fibonacci sequence is the 2-step sequence.

For example, the tribonacci sequence is the 3-step sequence. Its first 3 elements are 0, 0, 1. Its i th element is defined as $F_n(i) = F_n(i-1) + F_n(i-2) + F_n(i-3)$

fibonacci sequence = 0, 1, 1, 2, 3, 5, 8, 13, ...

tribonacci sequence = 0, 0, 1, 1, 2, 4, 7, 13, ...

tetranacci sequence = 0, 0, 0, 1, 1, 2, 4, 8, ...

Complete the function `n_step_fibonacci_maker` which takes in a single argument `n` and returns a function. The returned function takes in a single argument `i` and returns the element of the n-step Fibonacci sequence at position i. You should use recursiobn to complete this function.

```
def n_step_fibonacci_maker(n):
    """
    >>> fib = n_step_fibonacci_maker(2)
    >>> for i in range(4):
    ...     print(fib(i)) # fib(0) = 0, fib(1) = 1
    ...
    0
    1
    1
    2
    >>> tribonacci = n_step_fibonacci_maker(3)
    >>> tribonacci(3) # 0 + 0 + 1
    1
    >>> tribonacci(4) # 0 + 1 + 1
    2
    >>> tribonacci(5) # 1 + 1 + 2
    4
    """
    def n_step_fib(i):
        if i < n-1:
            __________________________
        if i == n-1:
            __________________________
        ith_element = __________________________
        for counter in range(1, n+1):
            _____________________________________
        _____________________________________
    __________________________
```

(a) (6.0 pt)

```
def n_step_fibonacci_maker(n):
    def n_step_fib(i):
        if i < n-1:
            return 0
        if i == n-1:
            return 1
        ith_element = 0
        for counter in range(1, n+1):
            ith_element += n_step_fibonacci(i-counter)
        return ith_element
    return n_step_fib
```

7. (5.0 points) The Big C's

Complete the recursive function `c_helper` so that `c_galore` returns `True` if `phrase` contains at least two uppercase `'C'` letters and `False` otherwise. `c_helper` takes in a string `phrase` and a boolean `has_seen_c` that represents whether a single `'C'` has been seen so far.

Note that indexing and slicing works on not only lists but also strings. If `a = 'hello'` then `a[1]` evaluates to `'e'` and `a[2:4]` evaluates to `'ll'`.

```
def c_galore(phrase):
    """ Returns True if the string `phrase` has at least 2 'C' letters and False otherwise.

    >>> c_galore('CS 88')
    False
    >>> c_galore('CS C88')
    True
    >>> c_galore('C3PC')
    True
    >>> c_galore('CC: CS C8C88C')
    True
    """
    return c_helper(phrase, False)

def c_helper(phrase, has_seen_c):
    if ____________________________________:
        return False
    if phrase[0] == 'C':
        if ____________________________________:
            return ____________________________________
        else:
            return ____________________________________
    else:
        return ____________________________________
```

(a) (5.0 pt)

```
def c_galore(phrase):
    return c_helper(phrase, False)

def c_helper(phrase, has_seen_c):
    if len(phrase) == 0:
        return False
    if phrase[0] == 'C':
        if has_seen_c:
            return True
        else:
            return c_helper(phrase[1:], True)
    else:
        return c_helper(phrase[1:], has_seen_c)
```

8. (13.0 points) Let's Get Ice Cream

You really like ice cream so you decided to open an ice cream store. In order to keep track of inventory and sales, you decide to write an `IceCreamStore` and `IceCream` class.

To start, complete the class `IceCreamStore`. The `IceCreamStore` class has four instance attributes: `flavors`, a list containing the flavors you have in stock, `scoop_price`, the price of a scoop (a float), `cone_price`, the cost of an ice cream cone, and `revenue` (a float)representing how much money the store has made cumulatively. Fill out the constructor so that `self.flavors`, `self.scoop_price`, and `self.cone_price` store `flavors`, `scoop_price`, and `cone_price` respectively, and initialize `self.revenue` with our starting revenue of 0.

```
class IceCreamStore:
    def __init__(self, flavors, scoop_price, cone_price):
        """
        >>> flavors = ['basil', 'sesame', 'sweet corn']
        >>> my_store = IceCreamStore(flavors, 1.5, 0.5)
        >>> self.revenue
        0
        """
        self.flavors = _____________
        self.scoop_price = _____________
        self.cone_price = ____________
        self.revenue = ___________

    def make_order(self, flavors, cone):
        """
        Implemented in part d
        """
```

(a) (2.0 pt)

```
def __init__(self, flavors, scoop_price, cone_price):
    self.flavors = flavors
    self.scoop_price = scoop_price
    self.cone_price = cone_price
    self.revenue = 0
```

(b) **(3.0 pt)** In order to represent an individual ice cream order, we will write an IceCream class with two instance variables, `scoops`, a list containing which flavors are in the order, and `scoop_price`, which is the price of a single `scoop` of ice cream. The constructor is provided for you. Your goal is to write the method, `price`, which should calculate the overall price of the ice cream.

```
class IceCream:
    def __init__(self, scoops, scoop_price):
        """
        >>> ice_cream = IceCream(['salted caramel', 'sesame'], 1.5)
        >>> ice_cream.scoops
        ['salted caramel', 'sesame']
        >>> ice_cream.scoop_price
        1.5
        """
        self.scoops = scoops
        self.scoop_price = scoop_price

    def price(self):
        """
        Returns the price of the ice cream order

        >>> ice_cream = IceCream(['mint chip', 'mint chip'], 1.5)
        >>> ice_cream.price()
        3.0 # 2 scoops multiplied by 1.5
        """
        return ________________
```

Let's implement the `price` function. `price` calculates the price of the ice cream by this formula: the number of scoops multiplied by the price of a single scoop.

```
def price(self):
    return len(self.scoops) * self.scoop_price
```

(c) **(2.0 pt)** We also want to be able to sell ice cream in a cone, for a slight upcharge of course! `IceCreamCone` inherits from `IceCream` but now has a new instance variable `cone_price` which is the cost of the cone. The constructor has been provided for you, your task is to overwrite `price` to accomodate the cost of the cone. Select the implementation that is both correct and avoids redundant code.

```
class IceCreamCone(IceCream):
    def __init__(self, scoops, scoop_price, cone_price):
        super().__init__(scoops, scoop_price)
        self.cone_price = cone_price

    def price(self):
        """
        >>> ice_cream = IceCreamCone(['chocolate', 'kinako'], 1.5, 0.5)
        >>> ice_cream.price()
        3.5 # 2 scoops multiplied by 1.5 plus the 0.5 cost of the cone
        """
        return _____________
```

- ○ `super().price() + cone_price`
- [✓ `super().price() + self.cone_price`]
- ○ `self.scoop_price * len(self.scoops) + cone_price`
- ○ `price() + self.cone_price`

(d) **(6.0 pt)** Now let's put it all together! Implement the `make_order` method of the `IceCreamStore` class to be able to represent a customer's order. `make_order` takes in two arguments, `flavors`, a list of the scoops the customer wants, and `cone`, a boolean which indicates whether the customer wants their ice cream in a cone.

i. First check that the flavors requested are made by our store, and if so `make_order` should make a new `IceCream` object or an `IceCreamCone` object if the customer wants a cone.
ii. Then, add the price of the ice cream to `revenue`.
iii. At the end, return the `IceCream` or `IceCreamCone` object, or `None` if the store doesn't carry a flavor in the order.

```
class IceCreamStore:
    def __init__(self, flavors, scoop_price, cone_price):
        """
        Implementation done in part a
        """

    def make_order(self, flavors, cone):
        """
        >>> flavors = ['salted caramel', 'matcha', 'blue moon']
        >>> my_store = IceCreamStore(flavors, 1.5, 0.5)
        >>> my_store.make_order(['salted caramel', 'salted caramel'], False)
        <IceCream Object> # Order was successfully made
        >>> my_store.revenue
        3.0 # 2 scoops * $1.5
        >>> my_store.make_order(['matcha', 'blue moon'], True)
        <IceCreamCone Object>
        >>> my_store.revenue
        6.5 # made $3.5 (2 scoops * $1.5 + $0.5 for cone) and had $3 already
        >>> my_store.make_order(['pistachio'], False)
        pistachio not in stock # Store doesn't carry pistachio
        """
        for flavor in flavors:
            if ____ not in ______:
                print(flavor + " not in stock")
                return ____________
        if _______:
            order = IceCreamCone(________)
        else:
            order = __________
        _______________
        return order
```

```
def make_order(self, flavors, cone):
    for flavor in flavors:
        if flavor not in self.flavors:
            print(flavor + " not in stock")
            return None
    if cone:
        order = IceCreamCone(flavors, self.scoop_price, self.cone_price)
    else:
        order = IceCream(flavors, self.scoop_price)
    self.revenue += order.price()
    return order
```

9. (8.0 points) Jackson Pollock's Data Table

In this problem, we will be creating basic data table, which will allow you to add, delete and access columns. You may remember the tables that were used in Data 8 or may have used a table-based program like Excel or google Sheets.

The table will be represented as a 2D array with the first element being a list of column names (like a "header" row). Subsequent elements will be lists that represent the values in row.

Complete the functions `create_table`, `add_row`, `delete_row`, `num_rows`, and `get_row`, to match the behavior in the doctests. The functions implement our Data Table abstract data type.

Hint: You might want to use list functions `append` and `pop`.

```
def create_table(columns):
    """
    >>> t = create_table(['product', 'inventory', 'price'])
    >>> t
    [['product', 'inventory', 'price']]
    >>> add_row(t, ['apple', 4, 2.99])
    >>> t
    [['product', 'inventory', 'price'], ['apple', 4, 2.99]]
    >>> num_rows(t)
    1
    >>> get_row(t, 0)
    ['apple', 4, 2.99]
    >>> delete_row(t, 0)
    >>> t
    [['product', 'inventory', 'price']]
    >>> num_rows(t)
    0
    """
    return ______________________________________________________

def add_row(table, row):
    """
    Adds a row to the table.
    Assume that the number of entries in the row matches the number of columns in the table
    """
    ______________________________________________________

def delete_row(table, index):
    """
    Deletes the row at index (assume that the user always provides a valid index).
    The row of column titles does not count in this indexing and the table is zero-indexed.
    See doctests for more details
    """
    ______________________________________________________

def num_rows(table):
    """
    Return the number of rows in the table. Column names do not count as a row
    """
    return ______________________________________________________

def get_row(table, index):
    """
    Return the row at index (assume that the user always provides a valid index).
    The row of column titles does not count in this indexing and the table is zero-indexed.
    See doctests for more details
    """
    return ______________________________________________________
```

(a) (1.0 pt)

33.0px0.75
```
def create_table(columns):
    return [columns]
```

(b) (1.0 pt)

33.0px0.75
```
def add_row(table, row):
    table.append(row)
```

(c) (1.0 pt)

33.0px0.75
```
def delete_row(table, index):
    table.pop(index + 1)
```

(d) (1.0 pt)

33.0px0.75
```
def num_rows(table):
    return len(table) - 1
```

(e) (1.0 pt)

33.0px0.75
```
def get_row(table, index):
    return table[index + 1]
```

(f) (1.0 pt) The following code returns the second to last row from the table `tb`. Does it break the abstraction barrier?

```
x = get_row(tb, len(tb) - 2)
```

[✓ `Breaks Abstraction Barrier`]

○ `Does NOT Break Abstraction Barrier`

(g) (1.0 pt) The following code returns the middle row from table `tb`. Does it break the abstraction barrier?

```
x = num_rows(tb)
middle = x // 2
r = get_row(tb, middle)
```

○ `Breaks Abstraction Barrier`

[✓ `Does NOT Break Abstraction Barrier`]

(h) (1.0 pt) The following code moves the first row to the bottom of the table `tb`. Does it break the abstraction barrier?

```
first = get_row(tb, 0)
delete_row(tb, 0)
add_row(tb, first)
```

○ `Breaks Abstraction Barrier`

[✓ `Does NOT Break Abstraction Barrier`]

No more questions.