Computational Structures in Data Science

Midterm Review

Berkeley
UNIVERSITY OF CALIFORNIA

Announcements & Policies

- Midterm:
  - 2 hours, 120 Minutes
  - 1 Hand-written double-sided cheat sheet.
    - You must use our template.
  - 1 CS88 Provided Reference Sheet
- Remember: HW6, Maps are in scope.
- Warning: Misconduct is a -100% score, no clobbering.
  - It's no fun. ☹

You are not your grades!

Do your best!

© CC BY-NC-SA

My Advice

- Don't rush!
  - Slow is fast and fast is slow
  - BREATHE!
- Skim the exam first
  - It's ok to do questions out of order!
  - Get the stuff you're good without out of the way
  - BUT don't spend too much time planning the exam.
- Read through the question once
  - What's it asking you to do at a high level?
  - What do the doctests suggest?
  - What techniques should you be using?
- Use the scratch space!

Midterm Topics

- Everything Through OOP w/ Inheritence
- Functions
- Higher Order Functions
  - Functions as arguments
  - Functions as return values
- Environment Diagrams
- Lists, Dictionaries
- List Comprehensions, Dictionary Comprehensions
- Abstract Data Types
- Recursion
- Object-Oriented Programming

© CC BY-NC-SA

Computational Structures in Data Science

Recursion Review

Berkeley
UNIVERSITY OF CALIFORNIA

Michael Ball | UC Berkeley | https://c88c.org | © CC BY-NC-SA

The Recursive Process

Recursive solutions involve two major parts:

- Base case(s), the problem is simple enough to be solved directly
- Recursive case(s). A recursive case has three components:
  - Divide the problem into one or more simpler or smaller parts
  - Invoke the function (recursively) on each part, and
  - Combine the solutions of the parts into a solution for the problem.

Recursion Key concepts – by example

1. Test for simple “base” case

2. Solution in simple “base” case

```
def sum_of_squares(n):
    if n < 1:
        return 0
    else:
        return sum_of_squares(n-1) + n**2
```

3. Assume recusive solution to simpler problem

4. ”Combine” the simpler part of the solution, with the recursive case

Fall 2025 Q8

- Exam: https://c88c.org/sp26/assets/pdfs/exams/c88c/fa25-mt.pdf
- Solutions: https://c88c.org/sp26/assets/pdfs/exams/c88c/fa25-mt-sol.pdf

Practice Question Fall 2025 Q8

You want to bring your nicest jewelry to vacation, however your suitcase has a weight limit. Luckily, you can use your Python skills to solve this problem! Implement suitcase, which takes in:

- A list of positive integers weights
- A list of positive integers values
- A nonnegative weight capacity p

It returns the **max value** of your jewelry that fits within the capacity (you want to pick some subset of the items so that you maximize the value you're bringing). Assume that the item at index i, weighs weights[i], pounds, and is worth values[i] dollars. You may also assume that the lengths of weights and values are always the same.

© CC BY-NC-SA

Code

```
def suitcase(weights, values, p):
    if ___(a)___:
        return 0
    else:
        first_weight = weights[0]
        rest_weights = weights[1:]
        first_value = values[0]
        rest_values = values[1:]
        with_first = first_value + ___(b)___
        without_first = ___(c)___
        if first_weight <= p:
            return ___(d)___
        else:
            return without_first
```

Recursion Process

- Recognize similar problems:
  - Subsets – kind of like count change
- Do we need to use recursion?
  - The problem doesn't say!
  - No scaffolding w/ for or while
  - Considering multiple possible combinations
- Notice first / rest pattern
  - We're splitting our data in 2 parts.
  - Combining this parts together.
- You don't need to solve the parts in order.
- *Classic* tree recursion. ☺

Practice Question Fall 2025 Q8

You want to bring your nicest jewelry to vacation, however your suitcase has a weight limit. Luckily, you can use your Python skills to solve this problem! Implement suitcase, which takes in:

- A list of positive integers `weights`
- A list of positive integers `values`
- A nonnegative weight capacity `p`

It returns the **max value** of your jewelry that fits within the capacity (you want to pick some subset of the items so that you maximize the value you're bringing). Assume that the item at index i, weighs weights[i], pounds, and is worth values[i] dollars. You may also assume that the lengths of weights and values are always the same.

© CC BY-NC-SA

Q8.1 (2 points) Select all the correct choices for blank (a).

- ☐ `weights == [[]] and p == 0`
- ☐ `weights == [] and p > 0`
- ☐ `weights == [] or p == 0`
- ☐ `len(weights) == 0 or p == 0`

Q8.2 (2 points) Fill in blank (b).

Q8.3 (2 points) Fill in blank (c).

Q8.4 (2 points) Fill in blank (d).

Q8.1 (2 points) Select all the correct choices for blank (a).

- ☐ `weights == [[]] and p == 0`
- ☐ `weights == [] and p > 0`
- ■ `weights == [] or p == 0`
- ■ `len(weights) == 0 or p == 0`

Solution: The function is correct as-is, but we should technically check `p <= 0` instead of just `p == 0` to avoid making unnecessary recursive calls. However, since we only call `suitcase` with `p - first_weight` when `first_weight <= p`, it does not affect the correctness of the function.

Q8.2 (2 points) Fill in blank (b).

```
suitcase(rest_weights, rest_values, p - first_weight)
```

Q8.3 (2 points) Fill in blank (c).

```
suitcase(rest_weights, rest_values, p)
```

Q8.4 (2 points) Fill in blank (d).

```
max(with_first, without_first)
```

Computational Structures in Data Science

Questions from Ed

Berkeley
UNIVERSITY OF CALIFORNIA

ADTs

- Constructors / Selectors / Operators
  - help us think about the actions to represent an object.
- Abstraction Barrier:
  - Means that a function does not rely *directly* on a particular implementation
- Should a function return data?
  - Usually yes! It depends on the goals!

Computational Structures in Data Science

ADTs Practice Questions

Berkeley
UNIVERSITY OF CALIFORNIA

Exam Practice

- Spring 22 Q7
  - Text: https://c88c.org/sp26/assets/pdfs/exams/c88c/sp22-mt.pdf
  - Solutions: https://c88c.org/sp26/assets/pdfs/exams/c88c/sp22-mt-sol.pdf
- Spring 20 Q5:
  - Text: https://c88c.org/sp26/assets/pdfs/exams/c88c/sp20-mt.pdf
  - Solutions: https://c88c.org/sp26/assets/pdfs/exams/c88c/sp20-mt-sol.pdf

SP22

7. (5.0 points) Closet Overhaul

You've designed a closet abstract data type to help you organize your wardrobe.

A closet contains two things:

- `owner`: the name of the closet owner represented as a string
- `clothes`: the collection of clothes in the closet represented as a dictionary, where the key is the clothing item name and the value is the number of times the clothing item has been worn.

The `make_closet` constructor takes in `owner` (a string) and `clothes` (a **list** of strings representing clothing items) and returns a closet ADT.

Given this, you've implemented the abstract data type as follows:

```
def make_closet(owner, clothes):
    """ Create and returns a new closet. """
    clothes_dict = {}
    for item in clothes:
        clothes_dict[item] = 0
    return (owner, clothes_dict)

def get_owner(closet):
    """ Returns the owner of the closet """
    return closet[0]

def get_clothes(closet):
    """ Returns a dictionary of the clothes in the closet """
    return closet[1]
```

Given the closet ADT, implement the functions `wear_clothes` and `favorite_clothing_item`. You may not need all the lines provided, and you may need to change the indentation for some lines.

5. **(10 points) Atey Ate Already**

It's a lot more fun to think about food than take midterms, so let's look at the cheapest places to fulfill an order. Given the function `total_cost` and assuming it works as described, fill out `find_restaurant` to find the cheapest restaurant to fulfill the order.

Remember: Pay close attention to the doctests to guide your solution.

```
def total_cost(restaurant, order):
        """
        Function that returns the total cost of an order at a certain
        restaurant. Returns -1 if fulfilling the request is not possible.
        >>> total_cost('chipotle', ['burrito', 'taco'])
        11.96
        >>> total_cost('sliver', ['boba'])
        -1.0
        """
        # We have omitted how this function works.
def find_restaurant(restaurants, order):
    """
    Function that returns the cheapest restaurant and price as the first
    element of a list followed by the prices for each of the restaurants.
    In the case that no restaurant can fulfill the order, the first
    element should be ['None found!', -1]. In the case that two
    restaurants have the same price, keep the first restaurant.
    Hint: Use total_cost!
    >>> find_restaurant(['chipotle', 'la burrita'], ['burrito', 'taco'])
    [['la burrita', 9.78], [['chipotle', 11.96], ['la burrita',9.78]]

    >>> find_restaurant(['sliver','cheeseboard'], ['boba'])
    [[None found!, -1.0], ['sliver', -1.0]['cheeseboard', -1.0]]
    """
```

SP20 #6

(10 points) Rooms within Rooms within Rooms

You are a Data Scientist hired by UC Berkeley to find the largest room on campus. In order to schedule midterms, your job is return the room and its capacity. The data on all the rooms plus capacity is in a weird format of three element lists, where the first element is the room, the second element is the capacity, and the third element is either the rest of the data or None. Assume that the capacity of each of the rooms is unique.

That is, the data look like `['Room', Number, [...]]`.

Use the following lines of code to fill in the body of the function. You will need to fill in the blanks of the lines provided. Some lines are optional.

```
return [rooms[0], rooms[1]]
largest_left = find_largest(______________________)
if rooms[2] == ______________:
if largest_left[1] > rooms[1]:
return ____________________
return ____________________
else: # this line is optional, depending upon your solution
else: # this line is optional, depending upon your solution
```

```
def find_largest(rooms):
    """
    Return the largest room from a weirdly nested list.
    You can assume rooms is always 3 items long.
    >>> rooms = ['Evans', 150, ['Wheeler', 700, ['Stadium', 50000, None]]]
    >>> find_largest(rooms)
    ['Stadium', 50000]
    >>> find_largest(['Evans', 150, None])
    ['Evans', 150]
    """
```

```
def find_largest(rooms):
    """
    Return the largest room from a weirdly nested list.
    You can assume rooms is always 3 items long.
    >>> rooms = ['Evans', 150, ['Soda', 200, ['Wheeler', 700, ['Stadium', 50000, None]
    >>> find_largest(rooms)
    ['Stadium', 50000]
    >>> find_largest(['Evans', 150, ['Hearst Annex', 50, None]])
    ['Evans', 150]
    """
    if rooms[2] == None:
        return [rooms[0], rooms[1]]
    largest_left = find_largest(rooms[2])
    if largest_left[1] > rooms[1]:
        return largest_left
    return [rooms[0], rooms[1]]
```

i. **(4.0 pt)** Implement the `find_new_interest` method that returns a string representing a new potential interest for this `User`. To determine this new interest, first identify this user's most similar follower that has the largest number of mutual interests with this user. Then return a randomly selected interest from this follower. But be careful, this randomly selected interest must not already exist in this user's interests (otherwise it would not be new!).

For this problem, assume that the user's interests and followers are non-empty. Note that the `separate_interests` function (see `User` class skeleton) may be helpful here. You may use `random.choice(lst` to return a radomly selected item from a list, `lst`.

```
def find_new_interest(self):
    """
    >>> u1 = User('bob', ['cooking', 'archery', 'tv'])
    >>> u2 = User('alice', ['shopping', 'guitar', 'cooking']) # has one in common with bob
    >>> u3 = User('mike', ['poker', 'tv', 'cooking']) # has two in common with bob
    >>> u1.add_follower(u2)
    >>> u1.add_follower(u3)
    >>> u1.find_new_interest()
    'poker'
    """
    most_similar_follower = max(
        ____________________________________,
        key = _______________________________________
    )
    return random.choice(_____________________________________________)
```

Write the fully *completed* `find_new_interest` function below using the skeleton code provided. You may not add, change, or delete lines from the skeleton code.

SP20 #5

(10 points) Atey Ate Already

It's a lot more fun to think about food than take midterms, so let's look at the cheapest places to fulfill an order. Given the function `total_cost` and assuming it works as described, fill out `find_restaurant` to find the cheapest restaurant to fulfill the order.

Remember: Pay close attention to the doctests to guide your solution.

```
def total_cost(restaurant, order):
        """
        Function that returns the total cost of an order at a certain
        restaurant. Returns -1 if fulfilling the request is not possible.
        >>> total_cost('chipotle', ['burrito', 'taco'])
        11.96
        >>> total_cost('sliver', ['boba'])
        -1.0
        """
        # We have omitted how this function works.
def find_restaurant(restaurants, order):
    """
    Function that returns the cheapest restaurant and price as the first
    element of a list followed by the prices for each of the restaurants.
    In the case that no restaurant can fulfill the order, the first
    element should be ['None found!', -1]. In the case that two
    restaurants have the same price, keep the first restaurant.
    Hint: Use total_cost!
    >>> find_restaurant(['chipotle', 'la burrita'], ['burrito', 'taco'])
    [['la burrita', 9.78], [['chipotle', 11.96], ['la burrita',9.78]]

    >>> find_restaurant(['sliver','cheeseboard'], ['boba'])
    [[None found!, -1.0], ['sliver', -1.0]['cheeseboard', -1.0]]
    """
```

```
def findRestaurant(restaurants, order):
    """
    Function that returns the cheapest restaurant and price as the first
    element of a list followed by the prices for each of the restaurants.
    In the case that no restaurant can fulfill the order, the first
    element should be ['None found!', -1].
    Hint: Use totalCost! In the case that two restaurants have the same price,
    keep the first restaurant.
    >>> findRestaurant(['chipotle', 'la burrita'], ['burrito', 'taco'])
    [['la burrita', 9.78], [['chipotle', 11.96], ['la burrita',9.78]]
    >>> findRestaurant(['sliver','cheeseboard'], ['boba'])
    [[None found!, -1.0], ['sliver', -1.0]['cheeseboard', -1.0]]
    """
    placesList = [ [restaurant, totalCost(restaurant, order)]
                    for restaurant in restaurants ]
    minCost = -1.0
    cheapestPlace = "None found!"
    for place in range(placesList):
        if place[1] != -1.0 and (place[1] < minCost or minCost == -1.0):
            minCost = place[1]
            cheapestPlace = place[0]
    return [cheapestPlace, minCost] + placesList
```

Computational Structures in Data Science

Environment Diagrams

Berkeley
UNIVERSITY OF CALIFORNIA

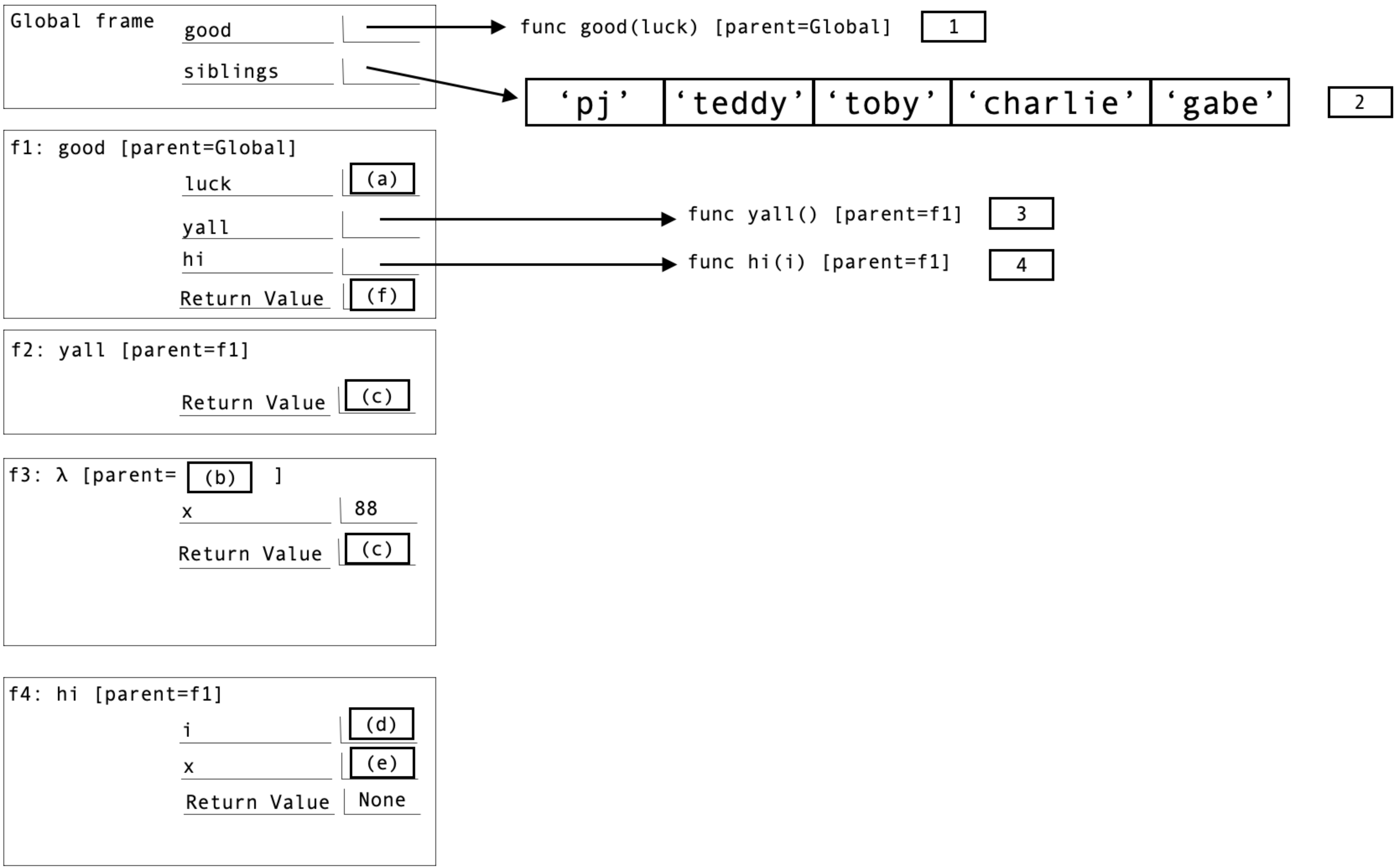
Michael Ball | UC Berkeley | https://c88c.org | © CC BY-NC-SA

Fall 2025 Q2

- Exam: https://c88c.org/sp26/assets/pdfs/exams/c88c/fa25-mt.pdf
- Solutions: https://c88c.org/sp26/assets/pdfs/exams/c88c/fa25-mt-sol.pdf
- Python Tutor

```python
def good(luck):
    def yall():
        return (lambda x : len(luck) // 2)(88)
    def hi(i):
        for x in range(len(luck)):
            if (i == len(luck)):
                i = 0
            print(luck[i])
            i += 1
    return hi(yall())

siblings = ['pj', 'teddy', 'toby', 'charlie', 'gabe']
good(siblings)
```



Global frame
good
siblings
func good(luck) [parent=Global]
1
'pj'
'teddy'
'toby'
'charlie'
'gabe'
2
f1: good [parent=Global]
luck
(a)
yall
hi
Return Value
(f)
func yall() [parent=f1]
3
func hi(i) [parent=f1]
4
f2: yall [parent=f1]
Return Value
(c)
f3: λ [parent=
(b)
]
x
88
Return Value
(c)
f4: hi [parent=f1]
i
(d)
x
(e)
Return Value
None

Computational Structures in Data Science

Environment Diagrams

Berkeley
UNIVERSITY OF CALIFORNIA

Michael Ball | UC Berkeley | https://c88c.org | © CC BY-NC-SA