Data C88C Fall 2024

Almeda, DeNero, Tsang Final

Print Your Name: ______________________________

Print Your Student ID: ______________________

Print Your @berkeley.edu email address: ______________________________

You have 170 minutes. There are 6 questions of varying credit (88 points total).

Question:	1	2	3	4	5	6	Total
Points:	10	11	24	28	15	0	88

For questions with **circular bubbles**,
you may select only one choice.

○ Unselected option (completely unfilled)

● Only one selected option (completely filled)

✓ Don't do this (it will be graded as incorrect)

- The exam is closed book, closed notes, closed computer, closed calculator, except three 8.5" x 11" pages of your own creation and the provided midterm and final study guide.
- Anything you write outside the answer boxes or you ~~cross out~~ will not be graded.
- If you write multiple answers, your answer is ambiguous, or the bubble is not entirely filled in, we will grade the worst interpretation.
- You may use built-in Python functions that do not require import, such as `pow`, `len`, `abs`, `bool`, `int`, `float`, `str`, `round`, `max`, `min`, `list`, `tuple`, `sum`, `all`, `any`, `map`, `filter`, `zip`, `sorted`, and `reversed`.
- You may **not** use example functions defined on your study guide unless a problem clearly states you can.
- Unless otherwise specified, you are allowed to reference functions defined in previous parts of the same question.
- You may not use ; to place two statements on the same line.
- Unless otherwise specified, do **not** hardcode function implementations to match example outputs.
- You may use the `Link` and `Tree` class defined on the midterm and final study guide.

Sign (or type) your name to confirm that all work on this exam will be your own. The penalty for academic misconduct on an exam is an F in the course.

Sign (or print) your name: ______________________________

Q1 *WWPD* (10 points)

Suppose we have the following Python code. There are no errors that occur when running this program.

```python
class Foo:
    a = []
    def __init__(self, a):
        self.a = a

    def func(self, a):
        self.a.append(a)
        return lambda a: self.a.pop(a)

    def func2(self, a):
        x = Foo.a.extend(a)
        print(x)
        return lambda a: Foo.a.pop(a)

class Bar(Foo):
    def __init__(self, a):
        1 + 2

    def func2(self, a):
        print(len(self.a))
        return super().func2(a)

f1 = Foo([])
x = f1.func([1, 2])
f1.func([3, 4])
print(x(1))
y = f1.func2([5, 6])
print(y(0))
f2 = Bar([])
z = f2.func2([7, 8])
print(z(0))
```

Q1.1 (2 points) What is printed out after line 26: `print(x(1))`?

- ○ [1, 2]
- ○ [3, 4]
- ○ [5, 6]
- ○ [7, 8]
- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5
- ○ 6
- ○ 7
- ○ 8
- ○ None

Q1.2 (2 points) What is printed out after line 27: `y = f1.func2([5, 6])`?

- ○ [1, 2]
- ○ [3, 4]
- ○ [5, 6]
- ○ [7, 8]
- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5
- ○ 6
- ○ 7
- ○ 8
- ○ None

Q1.3 (2 points) What is printed out after line 28: `print(y(0))`?

- ○ [1, 2]
- ○ [3, 4]
- ○ [5, 6]
- ○ [7, 8]
- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5
- ○ 6
- ○ 7
- ○ 8
- ○ None

Q1.4 (2 points) What is printed out after line 30: `z = f2.func2([7, 8])`?

Clarification: Only select what is printed out in LINE 20 when evaluating the call expression f2.func2([7, 8]), NOT LINE 12.

- ○ 0
- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5
- ○ 6
- ○ 7
- ○ 8
- ○ None

Q1.5 (2 points) What is printed out after line 31: `print(z(0))`?

- ○ [1, 2]
- ○ [3, 4]
- ○ [5, 6]
- ○ [7, 8]
- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5
- ○ 6
- ○ 7
- ○ 8
- ○ None

Q2 *SQL* **(11 points)**

Shm's family loves 2 things: frozen dumplings, and great deals! Complete the SQL queries using the two tables below.

Note: You may use SQL keywords in the blanks.

The `dumplings` table contains data on different frozen dumpling packs: the net **weight** of each package in ounces, and the **count** of dumplings in each pack.

The `prices` table contains data on the prices for each dumpling pack at two different stores, **storeA** and **storeB**.

The brands are stored as Strings, and all other data are stored as floats.

brand	**weight**	**count**
Bibigo Mandu	24.0	20.0
Shirakiku Gyoza	21.0	28.0
Mila Soup Dumpling	8.8	9.0
Twin Marquis	16.0	21.0
Surasang Gyoza	35.2	32.0
Wei Chuan Pork and Cabbage	21.0	28.0
Trader Joe's Soup Dumplings	6.0	6.0

Table 1: dumplings

brand	**storeA**	**storeB**
Bibigo Mandu	7.49	6.98
Shirakiku Gyoza	5.99	6.99
Mila Soup Dumpling	8.49	7.99
Twin Marquis	6.49	4.99
Surasang Gyoza	4.49	6.99
Wei Chuan Pork and Cabbage	8.19	5.49
Trader Joe's Soup Dumplings	3.49	3.99

Table 2: prices

(5 points) Fill in the SQL query that displays only the names of the top 3 brands with the best **ounces-per-dumpling** ratio, in order from highest ratio (i.e. thickest dumplings) to lowest. Your query should work correctly for any data that might appear in the dumplings and prices table.

Hint: The ratio can be calculated as weight divided by count.

This is the expected output for the example given:

brand
Bibigo Mandu
Surasang Gyoza
Trader Joe's Soup Dumplings

Table 3: The Three Thickest Dumplings

```
SELECT                                                                                 ;
```

(6 points) Shm will be shopping at their favorite grocery store ”storeB” later today, and wants to get the best dumpling deals there!

Fill in this SQL query that returns ONLY the dumpling brands that are cheaper at storeB than at storeA, in order from the lowest cost-per-ounce to highest cost-per-ounce (based on the price at storeB).

This is the expected output for the example given:

brand
Wei Chuan Pork and Cabbage
Bibigo Mandu
Twin Marquis
Mila Soup Dumpling

Table 4: Delicious Savings!

SQL Query:

```
SELECT (a) FROM dumplings as a, prices as b WHERE (b) ORDER BY (c);
```

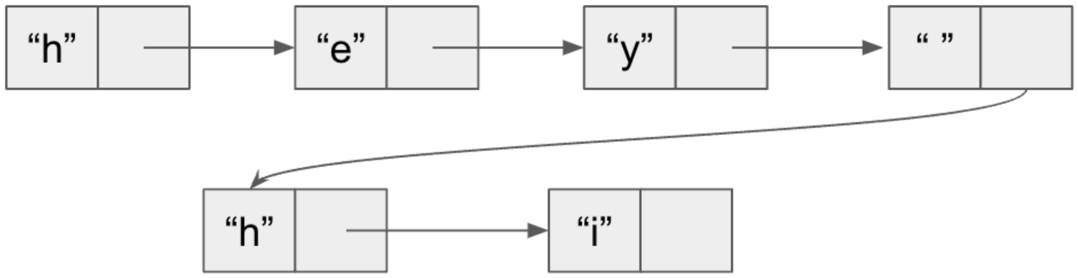
What should go in **(a)**?

What should go in **(b)**?

What should go in **(c)**?

Q3 *Linked Lists* **(24 points)**

Suppose we represent words using a Linked List, where each node contains a letter within the word. For example, the phrase "hey hi" could be stored as follows:



Implement `num_words`, which returns the number of words that are in a given Linked List. You may assume that words are separated by the space " " character.

```
def num_words(word_list):
    """Returns the number of words present in the word_list.
    A word is defined as a sequence of non-space characters.
    Each word is separated by spaces.
    >>> lst = Link("h", Link("e", Link("y", Link(" ", Link("h")))))
    >>> num_words(lst)
    2
    >>> lst = Link(" ", Link("h", Link("i", Link(" "))))
    >>> num_words(lst)
    1
    """
    count = 0
    in_word = False
    while BLANK ONE:
        if BLANK TWO and BLANK THREE:
            count += 1
            in_word = BLANK FOUR
        BLANK FIVE:
            in_word = BLANK SIX
        BLANK SEVEN
    return count
```

Q3.1 (2 points) What should go in BLANK ONE?

Q3.2 (1 point) What should go in BLANK TWO?

- ○ word_list != “ ”
- ○ word_list == “ ”
- ○ word_list.first != “ ”
- ○ word_list.first == “ ”
- ○ word_list.rest != “ ”
- ○ word_list.rest == “ ”

Q3.3 (1 point) What should go in BLANK THREE?

- ○ in_word
- ○ not in_word
- ○ True
- ○ False

Q3.4 (1 point) What should go in BLANK FOUR?

- ○ in_word
- ○ True
- ○ False

Q3.5 (1 point) What should go in BLANK FIVE?

- ○ else
- ○ elif word_list != “ ”
- ○ elif word_list == “ ”
- ○ elif word_list.first != “ ”
- ○ elif word_list.first == “ ”
- ○ elif word_list.rest != “ ”
- ○ elif word_list.rest == “ ”

Q3.6 (1 point) What should go in BLANK SIX?

- ○ in_word
- ○ True
- ○ False

Q3.7 (2 points) What should go in BLANK SEVEN?

(10 points) Implement `exists_in`, which returns whether a word `w` exists inside of a Linked List `word_list`. You may assume that all inputs to `w` are valid words.

```
def exists_in(word_list, w):
    """Returns whether a word w exists inside of a linked list word_list.

    A word is defined as a sequence of non-space characters.
    Each word is separated by a space.

    >>> lst = Link("h", Link("e", Link("y", Link(" ", Link("h", Link("i"))))))
    >>> exists_in(lst, "hey")
    True
    >>> exists_in(lst, "hi")
    True
    >>> exists_in(lst, "hello")
    False
    >>> exists_in(lst, "y")
    False
    """
    def helper(word_list, current_word):
        if word_list is Link.empty:

            return ______________________________________________
                                      Q3.8


        if ____________________________________________ == " ":
                                 Q3.9

            return ________________________ or ________________________
                             Q3.10                        Q3.11
        else:

            return ______________________________________________
                                      Q3.12

    return helper(word_list, "")
```

Q3.13 (2.5 points) Suppose we would like to add the String "aloha" to the **start** of a given Linked List. What would the fastest runtime of this algorithm be, with respect to the number of elements in the Linked List?

- ○ Constant
- ○ Logarithmic
- ○ Linear
- ○ Quadratic
- ○ Exponential

Q3.14 (2.5 points) Suppose we would like to add the String "aloha" to the **end** of a given Linked List. What would the fastest runtime of this algorithm be, with respect to the number of elements in the Linked List?

- ○ Constant
- ○ Logarithmic
- ○ Linear
- ○ Quadratic
- ○ Exponential

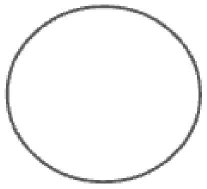
Q4 *Trees* **(28 points)**

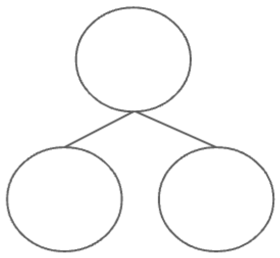
(5 points) Implement `count_nodes`, which counts the total number of nodes in a given Tree. You may assume that the Tree is non-empty.

```
def count_nodes(tree):
    """Counts the total number of nodes in the tree.
    >>> t1 = Tree(0, [Tree(1), Tree(1)])
    >>> count_nodes(t1)
    3
    """

    return ________([_______________________ for b in tree.branches])________
               Q4.1              Q4.2                                       Q4.3
```

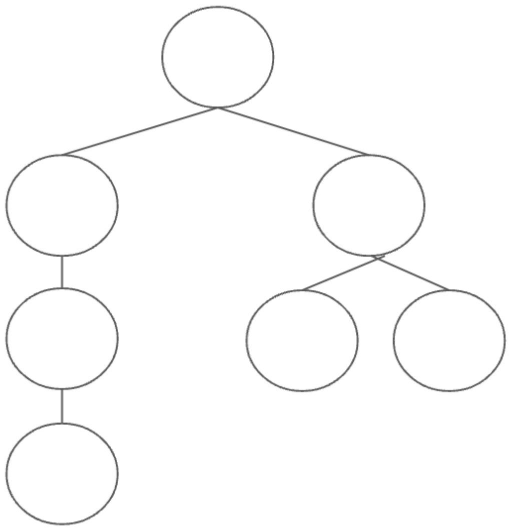
We define a Tree to be **symmetric** if every node at depth `d` shares the same number of children.

For example, the following Trees are symmetric:





And the following Tree is **not** symmetric (i.e. asymmetric):



The above tree has the following properties:

One node at depth d = 0 with 2 children

One node at depth d = 1 with 1 child

One node at depth d = 1 with 2 children

One node at depth d = 2 with 1 child

One node at depth d = 2 with 0 children

One node at depth d = 2 with 0 children

One node at depth d = 3 with 0 children

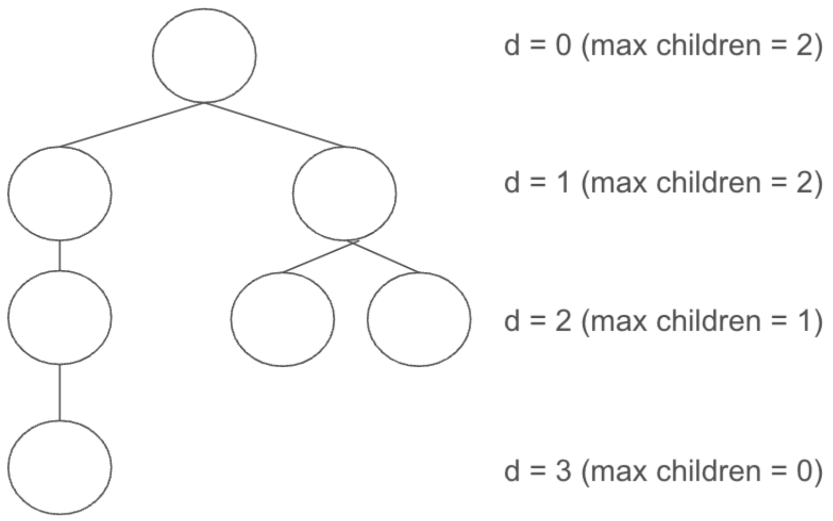
Since not all nodes in depth d = 1 and d = 2 share the same number of children, this tree is considered asymmetric.

(7 points) Let's start by implementing `max_children_at_depth`, which returns a dictionary mapping a given depth to the maximum number of children at that depth. For example, the asymmetric tree on the previous page should return the following dictionary:

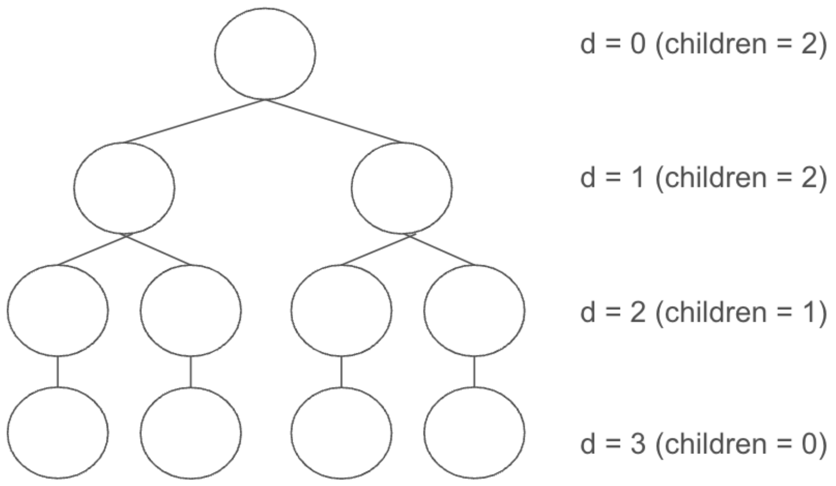
{0: 2, 1: 2, 2: 1, 3:0}

```
def max_children_at_depth(tree):
    def helper(node, depth, depth_dict):

        if ______________________________:
                       Q4.4
            depth_dict[depth] = 0

        depth_dict[depth] = max(depth_dict[depth], ___________________)
                                                          Q4.5
        for b in node.branches:

            helper(__________________________________)
                                 Q4.6
    depth_dict = {}
    helper(tree, 0, depth_dict)
    return depth_dict
```

(5 points) Next, implement `symmetrical`, which uses `max_children_at_depth` to determine how many nodes would be in a **symmetrical** Tree with the maximum number of children at each depth. For example, the following Tree:



would look like the following Tree if it was symmetrical with the maximum number of children at each depth:



Therefore, symmetrical would return (1) + (1 * 2) + (2 * 2) + (4 * 1) = 1 + 2 + 4 + 4 = 11 nodes.

```
def symmetrical(tree):
    depth_dict = max_children_at_depth(tree)
    nodes_in_layer, total_nodes = 1, 1
    for d in BLANK ONE:
        multiplier = depth_dict[d]
        total_nodes = BLANK TWO
        nodes_in_layer = BLANK THREE
    return total_nodes
```

Q4.7 (1 point) What should go in BLANK ONE?

Hint: Assume that dictionaries are unordered in Python.

Clarification: The answer to this question is range(len(depth_dict)).

- ○ depth_dict
- ○ range(len(depth_dict))
- ○ depth_dict.values()

Q4.8 (2 points) What should go in BLANK TWO?

Q4.9 (2 points) What should go in BLANK THREE?

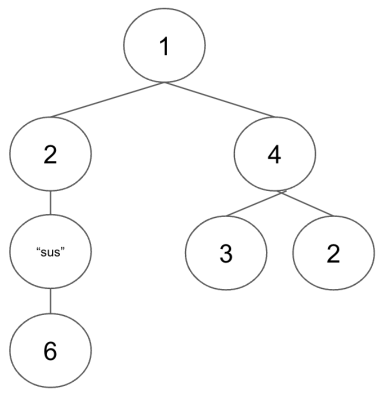
(3 points) Finally, let's put it all together! Implement `imbalance`, which returns the minimum number of nodes we would need to add to a given Tree to make it symmetrical. You may use any of the functions implemented earlier, assuming a correct implementation.

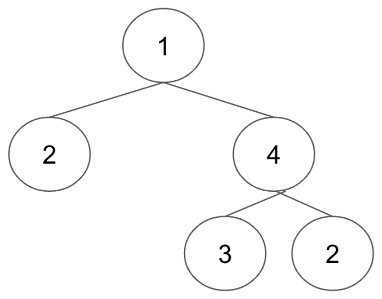
For example, the previous two trees would have an imbalance of 4, since the original tree had 7 nodes, and the symmetrical version of that tree has 11 nodes.

```
def imbalance(tree):

    return ____________________________________________
                            Q4.10
```

(8 points) Oh no! It looks like some imposters have invaded our Tree. Implement `eject`, which removes all nodes containing the label “sus” (imposter nodes) from a given Tree. All subtrees under imposter nodes are also removed as well.

For example, this is a Tree before (left) and after (right) ejecting the imposter nodes:





Clarification: The output of the expression eject(t3) in the doctest has been omitted.

```
def eject(t):
    """Ejects imposter nodes (and all nodes under them) from t.
    >>> t3 = Tree(0, [Tree("sus", [Tree(2, [Tree(3)]), Tree(2)]),
                  Tree(1, [Tree("sus")]),
                  Tree(1)])
    >>> print(t3)
    0
       sus
          2
             3
          2
       1
          sus
       1
    >>> eject(t3)
    >>> print(t3)
    0
       1
       1
    """
    if t.label == "sus":
        return BLANK ONE
    elif t.is_leaf():
        return BLANK TWO
    else:
        BLANK THREE = [BLANK FOUR for b in t.branches if BLANK FIVE]
        return t
```

Q4.1 (1.5 points) What should go in BLANK ONE?

- ○ 0
- ○ t.label
- ○ None
- ○ Tree(None)
- ○ t

Q4.2 (1.5 points) What should go in BLANK TWO?

- ○ 0
- ○ t.label
- ○ None
- ○ Tree(None)
- ○ t

Q4.3 (2 points) What should go in BLANK THREE?

- ○ t
- ○ t.label
- ○ t.branches

Q4.4 (1 point) What should go in BLANK FOUR?

- ○ eject(b)
- ○ eject(t.branches)
- ○ eject(t)

Q4.5 (2 points) What should go in BLANK FIVE?

Q5 *OOP* **(15 points)**

Suppose we want to map out all of the courses within the Data Science curriculum. Implement the `Course` class with the following specifications:

The `Course` class contains a class attribute `course_catalog`, which is a dictionary with course IDs (Integers) as keys and `Course` instances as values. Every time a new `Course` is created, we modify this class attribute.

Each `Course` has the following instance attributes:

- ID: an Integer uniquely identifying the course
- prereqs: a list of IDs representing the courses that must be taken before this one

Additionally, implement the `recommended_next_courses` method, which receives a list of course IDs `courses_taken`, and returns a list of recommended next course IDs, determined by the following criteria:

- This course is a prerequisite for the potential next course.
- Every prerequisite for the potential course has been satisfied.
- This potential next course has not been taken yet.

Note: For `recommended_next_courses`, courses_taken will not include the current course ID when called. See the code below for more details.

HINT: The dictionary `values()` method returns an object that provides access to the dictionary's values. You can use a for loop to traverse the object returned by this method.

Below is an example use case of the classes:

```
>>> data8 = Course(8)
>>> data8.ID
8
>>> list(Course.course_catalog.values())
[<__main__.Course object at 0x102ee46e0>] #The Data 8 Course Instance
>>> c88c = Course(88, [8])
>>> c88c.prereqs
[8]
>>> data100 = Course(100, [88, 54])
>>> math54 = Course(54)
>>> math54.recommended_next_courses([])
[]
>>> math54.recommended_next_courses([88, 100])
[]
>>> math54.recommended_next_courses([88])
[100]
```

```
class Course:

    ____________________________________________________________
                                  Q5.1

    def __init__(self, ID, prereqs = []):

        ______________________________ = ______________________________
                     Q5.2                              Q5.3

        ______________________________ = ______________________________
                     Q5.4                              Q5.5

        ____________________________________________________________
                                  Q5.6

    def recommended_next_courses(self, courses_taken):
        """
        Returns a list of recommended next course IDs after this course.
        See problem description for more details.
        Note: courses_taken will not include the current course ID
        """

        return [course.ID for course in ______________________________
                                                     Q5.7
               #Is this course a prereq?

               if __________________________________________________
                                          Q5.8
               #Is every prereq satisfied?

               and __________(
                      Q5.9

                   [________________________________________
                                     Q5.10

                    for prereq in ________________________________________
                                                  Q5.11

                    if __________________________________________________]
                                              Q5.12
                   )
               #We haven't taken this course yet.

               and course.ID ________________________________________]
                                             Q5.13
```

Q6 *The Finish Line* **(0 points)**

These questions will not be assigned credit; feel free to leave them blank.

Q6.1 (0 points) If there's anything else you want us to know, or you feel like there was an ambiguity in the exam, please put it in the box below.

For ambiguities, you must qualify your answer and provide an answer for both interpretations. For example, "if the question is asking about A, then my answer is X, but if the question is asking about B, then my answer is Y". You will only receive credit if it is a genuine ambiguity and both of your answers are correct. We will only look at ambiguities if you request a regrade.

(Question 6 continued...)