CS 88 Fall 2021 — Computational Structures in Data Science

Final SOLUTIONS

INSTRUCTIONS

- You have 180 minutes to complete the exam. **Do NOT open the exam until you are instructed to do so!**
- You **must not** collaborate with anyone inside or outside of CS88.
- 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 CS88 Reference Sheet

Full Name	
Student ID Number	
Official Berkeley Email (@berkeley.edu)	<EMAILADDRESS>
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.
    - **Online Exams: You may start you exam as soon as you are given the password.**
    - **You may have a digitial version of the CS88 Reference Sheet, or the PDF, but no other files.**
    - Open Reference Sheet Part 1
    - Open Reference Sheet Part 2

1. (7.0 points) WWPD

For each of the expressions in the table 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 a function is returned, write "Function". If nothing is returned, write "Nothing". Your answers must fit within the boxes provided. Work outside the boxes will not be graded.

```
f = lambda y, z: 3 * (y - z)

def fun(f, n):
    if n < 10:
        n += 2
    return lambda x: f(x, n)

class Holiday:
    def __init__(self, name, date):
        self.date = date
        self.name = name

    def find_date(self):
        return self.name + " is on " + self.date

    def celebrate(self):
        print("It's time to celebrate " + self.name)

class Birthday(Holiday):
    guest_list = []
    def __init__(self, name, date):
        Holiday.__init__(self, name, date)
        self.guest_list = []

    def add_guests(self, guests):
        return self.guest_list.extend(guests)

    def celebrate(self, attendance):
        gifts = []
        for name, gift in attendance.items():
            if name not in self.guest_list:
                print("surprise!")
            gifts.append(gift)
        return gifts
```

(a) (1.0 pt)

```
>>> fun(f, 4)(8)
```

```
6
```

(b) (0.5 pt)

```
>>> christmas = Holiday("christmas", "12/25")
>>> christmas.celebrate()
```

```
It's time to celebrate christmas
```

(c) (0.5 pt)

```
>>> jenny_birthday = Birthday("Jenny's birthday", "8/15")
```

```
Nothing
```

(d) (1.0 pt)

```
>>> jenny_birthday.find_date()
```

```
"Jenny's birthday is on 8/15"
```

(e) (1.0 pt)

```
>>> print(jenny_birthday.add_guests(["lukas", "chi", "matt"]))
```

```
None
```

(f) (1.0 pt)

```
>>> Birthday.guest_list
```

```
[]
```

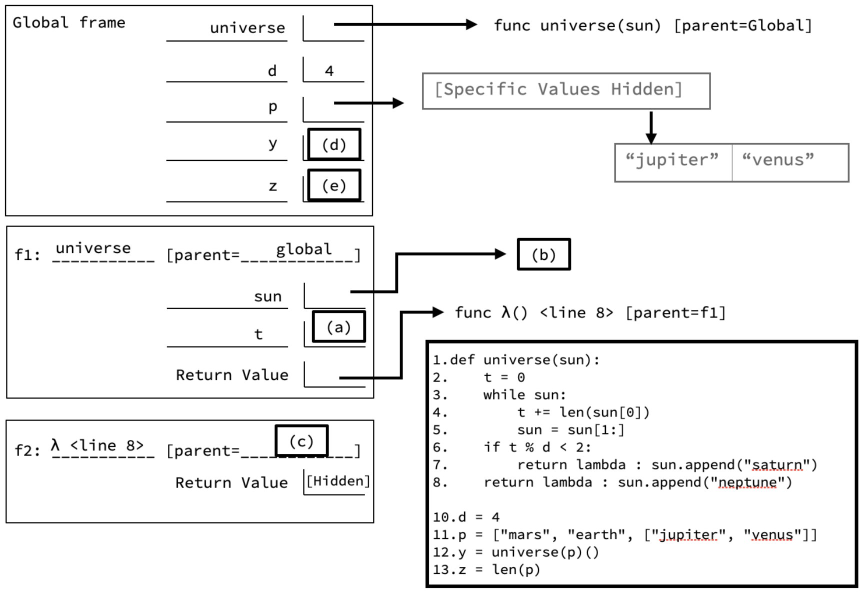
(g) (2.0 pt)

```
>>> Birthday.guest_list = ["shreya", "lukas", "tommy"]
>>> gifts = {"shreya": "candles", "lukas": "headphones", "nick": "cake"}
>>> jenny_birthday.celebrate(gifts)
```

```
surprise!
surprise!
['candles', 'headphones', 'cake']
```

2. (8.0 points) The Sun And All Its Planets Analysis

Fill in the blanks to complete the environment diagram and complete the questions about the environment diagram below.



Envronment Diagram

```
def universe(sun):
    t = 0
    while sun:
        t += len(sun[0])
        sun = sun[1:]
    if t % d < 2:
        return lambda : sun.append("saturn")
    return lambda : sun.append("neptune")

d = 4
p = ["mars", "earth", ["jupiter", "venus"]]
y = universe(p)()
z = len(p)
```

(a) (2.0 pt) What is the value of `t` in the f1 frame when the environment diagram is complete?

11

(b) (2.0 pt) What is the value of `sun` in the f1 frame when the environment diagram is complete?

["neptune"]

(c) (1.0 pt) What frame is the parent of the f2 frame?

f1 frame: universe

(d) (1.0 pt) What is the value of `y` in the global frame when the environment diagram is complete?

```
None
```

(e) (2.0 pt) What is the value of `z` in the global frame when the environment diagram is complete?

```
3
```

3. (5.0 points) Equation Solver

Complete the `solve_eqn` function that, given an equation `eqn` represented as a list, will compute the final value of evaluating the equation from left to right.

Think of an equation as an arithmetic expression with each integer or function in the equation stored at a separate index in the list (see the doctests for an example). Assume that `eqn` is a non-empty list that will be in a valid, processable format. For example, `[1, add, 1, 100]` would not be a valid input as it would translate to the equation `1 + 1 100`, but there should be an operation (like `add`) applied between `1` and `100` for it to be a valid equation.

```
def solve_eqn(eqn):
    """
    >>> from operator import add, sub
    >>> eqn = [1, add, 1] # 1 + 1
    >>> solve_eqn(eqn)
    2
    >>> eqn = [1, add, 2, sub, 3] # 1 + 2 - 3
    >>> solve_eqn(eqn)
    0
    """
    if __________________:
        return eqn[0]
    else:
        ans = ___________________
        return __________________
```

(a) (5.0 pt) Complete the skeleton code. You may not add, change, or delete lines from the skeleton code.

```
def solve_eqn(eqn):
    if len(eqn) == 1:
        return eqn[0]
    else:
        ans = eqn[1](eqn[0], eqn[2])
        return solve_eqn([ans] + eqn[3:])
```

4. (6.0 points) Remove-Multiple

We have created functions in the past that can remove every other node in a linked list, but Lukas wants a custom linked list mutated according some new specifications. He gives you a list of numbers, and tasks you with removing nodes using that list as a reference. For example, if he gives you a list containing [2, 7], you would keep the first node, remove the next two, keep the node after that, and then remove the next seven nodes, keeping any nodes afterwards.

Complete the `remove_multiple` function that takes in a linked list `lnk` and list `number_to_remove` and mutates the link that is passed in by following the above process. If the sum of the numbers in `number_to_remove` is more than the number of nodes in `lnk`, remove as many nodes as you can as dictated by the `number_to_remove` (see doctests).

```
def remove_multiple(lnk, number_to_remove):
    """
    >>> lnk = Link('a', Link('b', Link('c', Link('d', Link('e', Link('f'))))))
    >>> remove_multiple(lnk, [1, 2])
    >>> lnk
    Link('a', Link('c', Link('f')))
    >>> lnk = Link('a', Link('b', Link('c', Link('d', Link('e', Link('f'))))))
    >>> remove_multiple(lnk, [1, 2, 3, 4])
    >>> lnk
    Link('a', Link('c', Link('f')))
    """
    if ____________________________________________:
        return
    else:
        for i in range(_______________):
            if ______________________:
                _____________________________
            else:
                return
        remove_multiple(______________________________)
```

(a) (6.0 pt) Complete the skeleton code. You may not add, change, or delete lines from the skeleton code.

```
def remove_multiple(lnk, number_to_remove):
    if lnk is Link.empty or lnk.rest is Link.empty or not number_to_remove:
        return
    else:
        for i in range(number_to_remove[0]):
            if lnk.rest is not Link.empty:
                lnk.rest = lnk.rest.rest
            else:
                return
        remove_multiple(lnk.rest, number_to_remove[1:])
```

(b) (1.0 pt) As the length of `lnk` grows, what is the time complexity of the `remove_multiple` function?

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

5. (6.0 points) Divides Debugging

You've decided to write a function that returns `True` if `x` divides `y` and `False` otherwise. `x` divides `y` if there is an integer `c` such that `x*c` = `y`. The `divides` function should only accept integer values for both `x` and `y`, and your function should raise a `TypeError` if any other data type is passed.

Unfortunately, your code has some bugs in it: find at least 3 bugs and explain what they are! There can be bugs in any of the given lines in the body of the `divides` function.

```
def divides(x, y):
    """
    >>> divides(2, 4)
    True
    >>> divides(4, 2)
    False
    >>> 0 % 5
    0
    >>> divides(0, 5)
    File "<stdin>", line 1, in <module>
      File "<stdin>", line 2, in divides
    AssertionError: denominator should not be 0
    >>> divides(”two”, “four”)
    Traceback (most recent call last):
      File "<stdin>", line 1, in <module>
    TypeError: divides only takes integers
    """
    assert y != 0, “denominator should not be 0”
    if type(x) != int and type(y) != int:
        raise TypeError(“divides only takes integers”)
    return y % x != 0
```

In each box: Identify *one* of the 3 unique bugs and explain how to fix each bug. After all the bugs are fixed, the function should work as intended.

(a) (2.0 pt)

`assert y != 0` **should be** `assert x != 0` **because** `x` **is the denominator, and the denominator should not be 0.**

(b) (2.0 pt)

`if type(x) != int and type(y) != int` **should be an** `if type(x) != int or type(y) != int`**, so that if either** `x` **OR** `y` **is not an integer, an error is thrown**

(c) (2.0 pt)

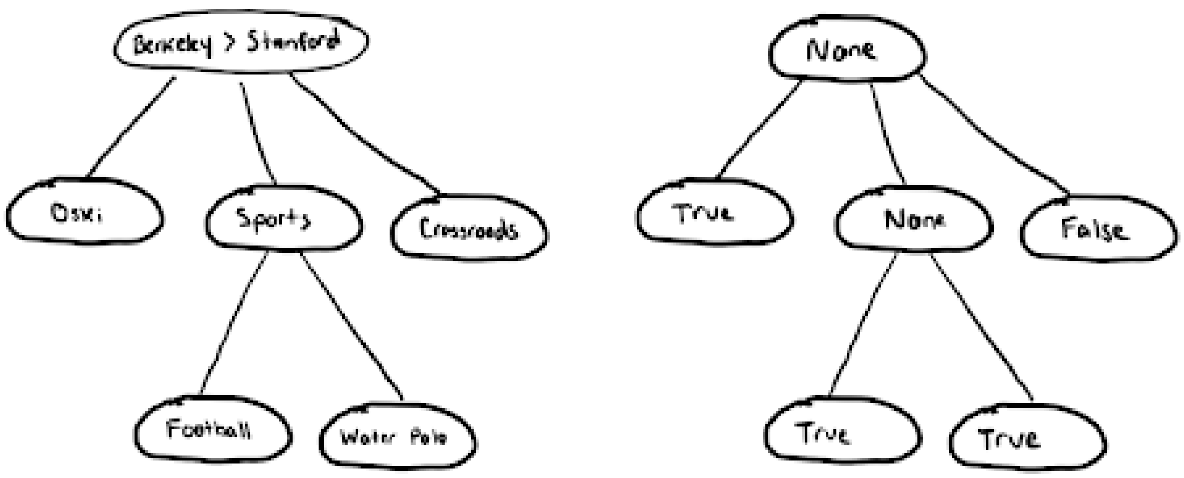
`y % x != 0` **should be** `y % x == 0` **as** `x` **divides** `y` **if the remainder IS 0**
Note that the following bug was also accepted:
Swap the first and second lines so that we first check that the type of the two arguments passed in are integers before asserting that the value of the denominator is not `0` **(better coding practice!)**

6. (7.0 points) Objective Judge

A debate judge is in charge of declaring the winner of a two-sided debate. In order to be more objective, the judge starts by drawing out all the points (nodes) in the debate as a "concept tree." A point's children are sub-points that are related to that point.

Then the judge creates another tree of identical structure called the "winner's tree" where non-leaf nodes are marked as `None` and leaf nodes are marked as `True` if the "for" side has won that point or `False` if the "against" side has won that point.

Complete the function `judge` which takes in a winner's tree `t` corresponding to a debate and returns `True` if the "for" side is the winner and `False` if the "against" side is the winner. The judge decides the "for" side has won a given point if they have won the majority of points directly below that point. If there is a tie, the "against" side wins that point. Using these rules, the judge can start at the leaves and move up to determine who won the point that is the root node and therefore the debate.



Concept Tree (NOT used in function) | Winner's Tree (used in function)

```
def judge(t):
    """
    >>> point1 = Tree(True)
    >>> point2 = Tree(None, [Tree(True), Tree(True)])
    >>> point3 = Tree(False)
    >>> point4 = Tree(None, [Tree(False), Tree(True), Tree(False)])
    >>> judge(Tree(None, [point1, point2, point3]))#debate1
    True
    >>> judge(Tree(None, [point1, point4, point3]))#debate2
    False
    """
    if _________________________________:
        return t.value
    else:
        points_won = 0
        for b in t.branches:
            if ____________________________________:
                ___________________________________
        if ________________________________________________:
            return True
        else:
```

(a) **(7.0 pt)** Write the fully *completed* `judge` function below using the skeleton code provided. You may not add, change, or delete lines from the skeleton code.

```
def judge(t):
    if t.is_leaf()::
        return t.value
    else:
        points_won = 0
        for b in t.branches:
            if judge(b):
                points_won += 1
        if 2 * points_won > len(t.branches):
            return True
        else:
            return False
```

7. (7.0 points) MultiMerger

Write a generator `merge` which takes in a list of sorted lists that each contain integers. It yields the integers in each of the lists in sorted order.

```
def merge(lst):
    """
    >>> list(merge([[1,3,5],[2,4,6]]))
    [1, 2, 3, 4, 5, 6]
    >>> list(merge([[1,2,3,4,5,6,7]]))
    [1, 2, 3, 4, 5, 6, 7]
    >>> list(merge([[1,2,5],[4],[6]]))
    [1, 2, 4, 5, 6]
    """
    indices = [0] * len(lst)
    while True:
        smallest = float('inf') # a number that is infinity
        smallest_index = -1
        for ____________:
            index = ____________
            if ____________:
                smallest = ____________
                smallest_index = ____________
        if ____________:
            return
        indices[smallest_index] += 1
        yield ____________
```

(a) (7.0 pt) Complete the skeleton code. You may not add, change, or delete lines from the skeleton code.

```
def merge(lst):
    indices = [0] * len(lst)
    while True:
        smallest = float(“inf”)
        smallest_index = -1
        for i in range(len(indices)):
            index = indices[i]
            if index < len(lst[i]) and lst[i][index] <= smallest:
                smallest = lst[i][index]
                smallest_index = i
        if smallest_index == -1:
            return
        indices[smallest_index] += 1
        yield smallest
```

8. (8.0 points) Swaps

Complete the `swap_pairs` function that takes a linked list `lnk` and two argument function `f` as input. This function creates a new linked list that is the same length as `lnk` by processing the given `lnk` in pairs of 2 `Link` nodes (we'll call the first node in the pair Node A and the second node in the pair Node B).

- If calling `f` on Node A and Node B returns `True`, add two `Link` nodes to the new linked list with the order swapped, so the first `Link` node added will have the value of Node B and the second `Link` node will have the value of Node A.
- Otherwise, add the two `Link` nodes to the new linked list in the same order as they were in the original `lnk`, so the first `Link` node added will have the value of Node A and the second `Link` node will have the value of Node B.

For every pair of 2 `Link` nodes, let the first `Link` node's value be the first argument for `f` and the second `Link` node's value be the second argument.

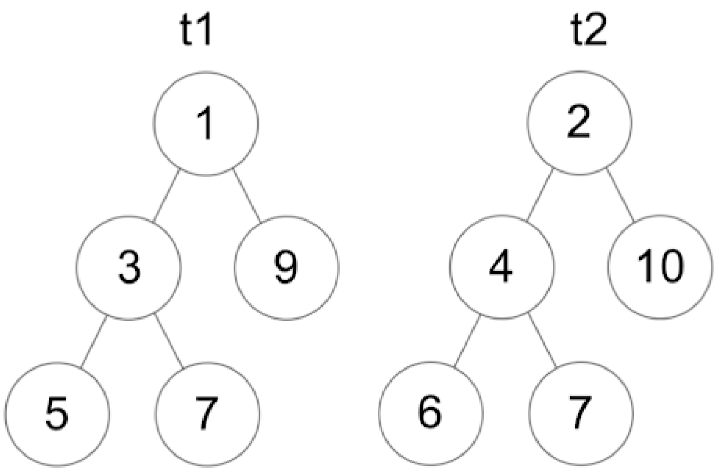
```
def swap_pairs(lnk, f):
    """
    >>> f = lambda x, y: x < y
    >>> g = lambda x, y: x != y
    >>> swap_pairs(Link(1), f)
    Link(1)
    >>> lnk = Link(1, Link(2))
    >>> swap_pairs(lnk, f)
    Link(2, Link(1))
    >>> swap_pairs(Link(2, Link(1)), g)
    Link(1, Link(2))
    >>> swap_pairs(Link(1, Link(2, Link(4, Link(3)))), f)
    Link(2, Link(1, Link(4, Link(3))))

    """
    if ________________________:
        return ________________
    elif ______________________:
        return ________________
    else:
        rest = swap_pairs(_____________, _____________)
        if __________________:
            return ________________________________________
        else:
            return _________________________________________
```

(a) (8.0 pt) Complete the skeleton code. You may not add, change, or delete lines from the skeleton code.

```
def swap_pairs(lnk, f):
    if lnk is Link.empty:
        return  Link.empty
    elif lnk.rest is Link.empty:
        return Link(lnk.first)
    else:
        rest = swap_pairs(lnk.rest.rest, f)
        if f(lnk.first, lnk.rest.first):
            return Link(lnk.rest.first, Link(lnk.first, rest))
        else:
            return Link(lnk.first, Link(lnk.rest.first, rest))
```

9. (12.0 points) Tree Farm

You've decided to get into the tree growing business! All the trees you grow have the same structure as each other but may have different values. You want to detect the nodes that are in the same position in two given trees but have different values. Write a function that takes in two trees, `t1` and `t2`, with the same structure and yields the mismatching node values as a tuple.




```
def tree_mismatches(t1, t2):
    """
    >>> t1 = Tree(1, [Tree(3, [Tree(5), Tree(7)]), Tree(9)])
    >>> t2 = Tree(2, [Tree(4, [Tree(6), Tree(7)]), Tree(10)])
    >>> a = tree_mismatches(t1, t2)
    >>> next(a)
    (1, 2)
    >>> next(a)
    (3, 4)
    >>> next(a)
    (5, 6)
    >>> next(a)
    (9, 10)
    >>> next(a)
    Traceback (most recent call last):
      File "<stdin>", line 1, in <module>
    StopIteration
    """
    if __________________:
        ________________________
    n = __________________________
    for i in range(n):
        branch_mismatches = _______________________
        for _________________________:
            _________________________
```

(a) (6.0 pt) Complete the skeleton code. You may not add, change, or delete lines from the skeleton code.

```
def tree_mismatches(t1, t2):
    if t1.value != t2.value:
        yield t1.value, t2.value
    n = len(t1.branches)
    for i in range(n):
        branch_mismatches = tree_mismatches(t1.branches[i], t2.branches[i])
        for val in branch_mismatches:
            yield val
```

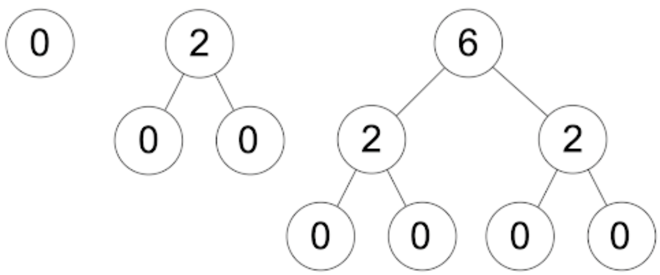
(b) (6.0 pt) What's better than identical trees? Big trees! You want to create the `tree_generator_2000` function that accepts an integer `n` and yields a new, larger tree whenever `next` is called. `next` can be called infinitely many times.

You are provided a function `tree_copy` which accepts a tree `t` as an argument and returns a copy of `t`.

The trees generated by the `tree_generator_2000` have a very specific structure:

- Every non-leaf node has `n` branches
- The value stored at each node is equal to the number of nodes beneath it
- The depth of the tree increases by one each time `next` is called

First, second, and third trees generated when n = 2:




```
def tree_copy(t):
    ‘“”’Helper function that accepts a tree t as an argument and returns a copy of t‘“”’
    <implementation hidden>

def tree_generator_2000(n):
    ‘“”’
    >>> binary_tree = tree_generator_2000(2)
    >>> next(binary_tree)
    Tree(0)
    >>> next(binary_tree)
    Tree(2, [Tree(0), Tree(0)])
    >>> next(binary_tree)
    Tree(6, [Tree(2, [Tree(0), Tree(0)]), Tree(2, [Tree(0), Tree(0)])])
    >>> next(binary_tree)
    Tree(14, [Tree(6, [Tree(2, [Tree(0), Tree(0)]), Tree(2, [Tree(0), Tree(0)])]), Tree(6, [Tree(2
    >>> ternary_tree = tree_generator_2000(3)
    >>> next(ternary_tree)
    Tree(0)
    >>> next(ternary_tree)
    Tree(3, [Tree(0), Tree(0), Tree(0)])
    >>> next(ternary_tree)
    Tree(12, [Tree(3, [Tree(0), Tree(0), Tree(0)]), Tree(3, [Tree(0), Tree(0), Tree(0)]), Tree(3,
    ‘“”’
    tree = _________________________________
    while __________________________________:
        _________________________________________
        new_branches = ___________________________
        tree = _______________________________
```

Complete the skeleton code. You may not add, change, or delete lines from the skeleton code.

```
def tree_generator_2000(n):
```

```
    tree = Tree(0)
    while True:
        yield tree
        new_branches = [tree_copy(tree) for i in range(n)]
        tree = Tree(tree.value*n+n, new_branches)
```

10. (11.0 points) Mario Kards

Friends from the world of Mario Kart have gathered to play some cards!

The players table contains information about the player id, player name, and the number of card games that player has won in the past. The cards table contains information about each playing card including the suit of the card (either "club", "diamond", "heart", or "spade""), the card value (a number from 1 - 13), and the player id of the player that holds that card (or -1 if no player has that card in their hand right now).

players

id	name	games_won
1	"Peach"	4
2	"Mario"	3
3	"Bowser"	4
4	"Luigi"	4
5	"Toad"	3

cards

suit	value	dealt_to
"club"	5	4
"diamond"	8	3
"spade"	9	2
"heart"	13	-1
"heart"	11	1
"diamond"	9	2
"club"	12	2
"club"	4	-1
"club"	1	3

Use the SQL skeleton to complete each question.

(a) (2.0 pt) Find the card suit and card value for all cards that have not been dealt to any player.

suit	value
"heart"	"13"
"club"	"4"

```
SELECT suit, value
  FROM cards
  WHERE dealt_to = -1;
```

name1	name2

(b) **(3.0 pt)** You're trying to find players who have similar playing levels to other players. Find all unique pairs of player names where both players in the pair have won the same number of games. The player names within each pair can appear in any order in the resulting table.

name1	name2
"Bowser"	"Peach"
"Bowser"	"Luigi"
"Luigi"	"Peach"
"Mario"	"Toad"

```
SELECT a.name, b.name
  FROM players as a, players as b
  WHERE a.name < b.name
  AND a.games_won = b.games_won;
```

(c) **(3.0 pt)** For each player that has at least one card, find the total card value for all cards that they currently hold.

name	total_value
"Peach"	11
"Mario"	30
"Bowser"	9
"Luigi"	5

```
SELECT p.name, SUM(c.value)
  FROM cards as c, players as p
  WHERE c.dealt_to = p.id
  GROUP BY p.name;
```

(d) **(3.0 pt)** In this game, any player wins if they have at least one pair of cards that add up to exactly 21. Find the name of all players who win according to this rule. **Note: There may be a solution which does not use all lines.**

name
"Mario"

```
SELECT p.name FROM cards as c1, cards as c2, players as p
  WHERE c1.dealt_to = c2.dealt_to
    AND c1.value + c2.value == 21
    AND c1.dealt_to = p.id
  GROUP by p.name;
```

11. (15.0 points) Trusty Triangles

Fill in the `Polygon` and `Triangle` classes below according to the problem descriptions and the doctests. Polygons are represented as a list of points, and triangles are polygons represented by a list of exactly 3 points.

```
class Polygon:
    shape_counts = {}
    def __init__(self, points)
        # part (a)

    def count_by_sides(num_sides)
        # part (a)

    def find_area(self)
        # part (c)

class Triangle(Polygon):
    def num_tris_created()
        # part (b)

    def find_area(self)
        # Assume this function is already correctly implemented!
```

(a) (6.0 pt) First, complete the `__init__` so that the class attribute `shape_counts` stores key-value pairs where the key is the number of points and the value is the number of Polygons created with that number of points. Also every polygon instance should have a single instance attribute `points` which is bound to the list of points representing the polygon.

Second, complete the `count_by_sides` using the class attribute `shape_counts` to return the number of polygons that have been created with `num_sides` sides.

```
>>> sq = Polygon([(0, 0), (4, 0), (4, 4), (0, 4)])
>>> sq2 = Polygon([(0, 0), (0, 2), (2, 2), (2, 0)])
>>> tr = Triangle([(0, 0), (4, 0), (0, 3)])
>>> Polygon.count_by_sides(4)
2
>>> Polygon.count_by_sides(10)
0

class Polygon:
    shape_counts = {}
    def __init__(self, points):
        ____________________________________________________________
        if ____________________________________________________________:
            ____________________________________________________________
        else:
            ______________________________________________________________
    def count_by_sides(num_sides):
        if ____________________________________________________________:
            return ____________________________________________________
        else:
            return ____________________________________________________
```

```
def __init__(self, points):
    self.points = points
    if len(self.points) in self.shape_counts:
        Polygon.shape_counts[len(self.points)] += 1
    else:
        Polygon.shape_counts[len(self.points)] = 1

def count_by_sides(num_sides):
    if num_sides in Polygon.shape_counts::
        return Polygon.shape_counts[num_sides]
    else:
        return 0
```

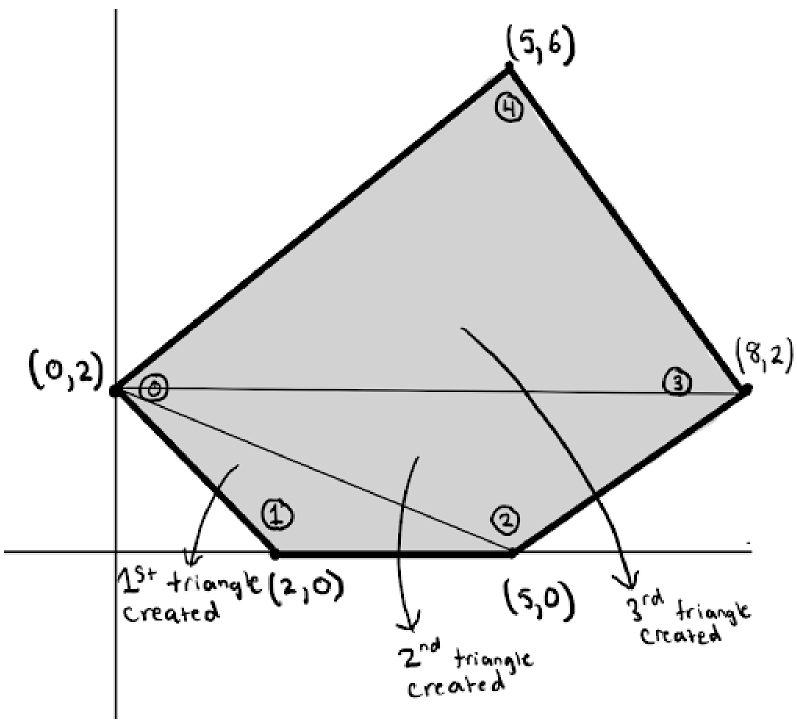
(b) (2.0 pt) Complete `num_tris_created` so that it returns the number of triangles that have ever been created. Note that you may not need to use all the lines for a correct solution.

```
>>> Polygon.shape_counts = {}
>>> sq = Polygon([(0, 0), (4, 0), (4, 4), (0, 4)])
>>> tr = Triangle([(0, 0), (4, 0), (0, 3)])
>>> Triangle.num_tris_created()
1
>>> tr2 = Triangle([(0, 0), (4, 0), (0, 3)])
>>> Triangle.num_tris_created()
2

def num_tris_created():
    ______________________________________________________
    ______________________________________________________
    ______________________________________________________
```

```
def num_tris_created():
  return Polygon.count_by_sides(3)
```

(c) **(7.0 pt)** Complete the `find_area` function so that it returns the area of the polygon. Crucially you may assume that the list of points in the polygon are ordered in the counterclockwise direction (the circled integers in the drawing below indicate the index corresponding to the point in the points list). Also since all polygons can be broken into triangles, you may find the already correctly implemented `find_area` function in the `Triangle` class useful.

```
>>> sq = Polygon([(0, 0), (4, 0), (4, 4), (0, 4)])
>>> round(sq.find_area())
16
>>> pent = Polygon([(0, 2), (2, 0), (5, 0), (8, 2), (5, 6)]) # See visual below
>>> round(pent.find_area())
27
>>> line = Polygon([(0, 0), (4, 0)])
>>> round(line.find_area())
0

def find_area(self):
    if len(self.points) <= __________________:
        return _______________________________________________________
    first_tri = ______________________________________________________
    points_copy = ____________________________________________________
    points_copy.pop(_____)
    return ____________________________ + ____________________________
```




```
def find_area(self):
    if len(self.points) <= 2:
        return 0
    first_tri = Triangle(self.points[0:3])
    points_copy = self.points[:]
    points_copy.pop(1)
    return first_tri.find_area() + Polygon(points_copy).find_area()
```

No more questions.