CS 61A Structure and Interpretation of Computer Programs
Spring 2022

MIDTERM 2

INSTRUCTIONS

This is your exam. Complete it either at 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.

(a) What is your full name?

(b) What is your student ID number?

(c) Below is a breakdown of the points per question, to help you budget your time.

Question	Points
Q1	5 pts
Q2	3 pts
Q3	2 pts
Q4	6 pts
Q5	4 pts
Q6	3 pts
Q7	6 pts
Q8	21 pts

1. (5.0 points) A Bug's Life

Thank you to our course staff members Richard, Melanie, and Amritansh for donating their bug collection to the midterm.

(a) (1.0 points) Wasp

Richard is trying to write a generator function that maps a given iterable using a one-argument function. Here's what he comes up with:

```
1 def map_gen(fn, iterable):
2     for it in iterable:
3         yield from fn(it)
```

i. (0.5 pt) Richard tests out his function by running the code below. He expected to see `[2, 4, 6, 8]` but instead gets an error when he runs these lines in an interpreter:

```
>>> data = [1, 2, 3, 4]
>>> fn = lambda x: x * 2
>>> list(map_gen(fn, data))
```

Which line of code in the function caused the code to break?

◯ 1

◯ 2

◯ 3

ii. (0.5 pt) Rewrite the buggy line of code you identified above to fix the error.

(b) (1.5 points) Moth

Melanie is trying to write some code to represent a cylinder in Python. This is her code:

```
1  class Cylinder:
2      def __init__(self, radius, height):
3          self.radius = radius
4          self.height = height
5
6      def volume(self):
7          volume = 3.14 * self.radius ** 2 * self.height
8          return volume
9
10     def percent_filled(self, volume_of_water):
11         return (volume_of_water / self.volume) * 100
```

i. (0.5 pt) Melanie runs the following code expecting to see it return 50.0, since 392.5 is a half-empty `cyl` (or is it half-full?).

```
>>> cyl = Cylinder(5, 10)
>>> cyl.percent_filled(392.5)
```

Unfortunately, the code results in this error instead:

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/Users/melanie/cylinder.py", line 11, in percent_filled
    return (volume_of_water / self.volume) * 100
TypeError: unsupported operand type(s) for /: 'int' and 'method'
```

According to this traceback, which method contains an error?

◯ `__init__`

◯ `volume`

◯ `percent_filled`

ii. (1.0 pt) Rewrite the buggy line of code from the method that you identified above to fix the error.

(c) (2.5 points) Bumblebee

Amritansh steps up with a challenge of his own. He's writing some code to flatten lists, but he seems to be running into some errors. Here's his code:

```
1 def flatten(lst):
2     res = []
3     for el in lst:
4         if isinstance(el, list):
5             res += [res.extend(flatten(el))]
6         else:
7             res += [el]
8     return res
```

i. (1.0 pt) He runs this on a nested list, as below:

```
>>> my_data = ["Mialy", ["Daphne", "Jordan"]]
>>> flatten(my_data)
```

The result he expects is `["Mialy", "Daphne", "Jordan"]`. What is the actual result?

ii. (0.5 pt) What line number causes the unexpected output to appear?

iii. (1.0 pt) Rewrite the buggy line you identified above so that Amritansh gets the expected output.

2. (3.0 points) What Would Python Display?

Assume the following code has been run (you may find a box-and-pointer diagram useful):

```
>>> lst1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> number_one = lst1.remove(1)
>>> some_number = lst1.pop()
>>> lst2 = lst1
>>> lst3 = lst2[:]
>>> lst1.insert(4, lst3)
>>> lst3.extend([lst1.pop() for _ in range(3)])
>>> lst2.append(lst1.remove(lst3))
```

For each part below, write the output of running the given code. Assume that none of the lines below will error.

(a) (1.0 pt)

```
>>> lst1[4:] + [number_one, some_number]
```

(b) (1.0 pt)

```
>>> lst3[5:9]
```

(c) (0.5 pt)

```
>>> next(reversed(lst3[:5]))
```

(d) (0.5 pt)

```
>>> lst1 is lst2
```

3. (2.0 points) A Different Kind of Dictionary

The `DictionaryEntry` class stores entries for an English dictionary, using instance variables for the `word` and its `definition`.

```
class DictionaryEntry:
    """
    >>> euph = DictionaryEntry("euphoric", "intensely happy")
    >>> avid = DictionaryEntry("avid", "enthusiastic")
    >>> [euph, avid]
    [DictionaryEntry('euphoric', 'intensely happy'), DictionaryEntry('avid', 'enthusiastic')]
    >>> f'Today we are learning {euph}'
    'Today we are learning euphoric: "intensely happy"'
    """
    def __init__(self, w, d):
        self.word = w
        self.definition = d

    def __repr__(self):
        ----------
           (a)

    def __str__(self):
        ----------
           (b)
```

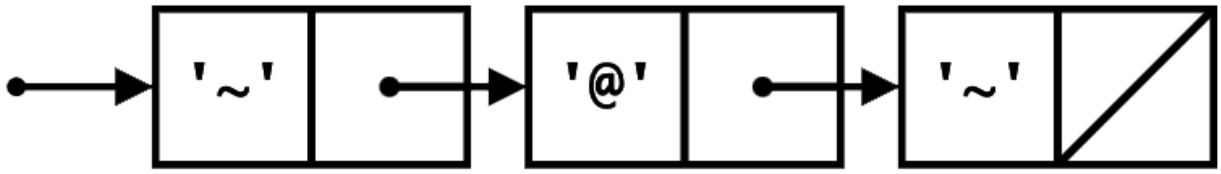
(a) (1.0 pt) Fill in blank (a).

(b) (1.0 pt) Fill in blank (b).

4. (6.0 points) Beadazzled

Implement `make_necklace`, a function that creates a linked list where each value comes from a given list of `beads`, and the beads are repeated in order to make a necklace of a given `length`.

For example, if `make_necklace` is called with `["~", "@"]` and a `length` of 3, then the linked list will contain `'~'`, then `'@'`, then `'~'`. Here's a diagram of that list:



See the docstring and doctests for further details on how the function should behave.

```
def make_necklace(beads, length):
    """
    Returns a linked list where each value is taken from the BEADS list,
    repeating the values from the BEADS list until the linked list has reached
    LENGTH. You can assume that LENGTH is greater than or equal to 1, there is
    at least one bead in BEADS, and all beads are string values.

    >>> wavy_ats = make_necklace(["~", "@"], 3)
    >>> wavy_ats
    Link('~', Link('@', Link('~')))
    >>> print(wavy_ats)
    <~ @ ~>
    >>> wavy_ats2 = make_necklace(["~", "@"], 4)
    >>> print(wavy_ats2)
    <~ @ ~ @>
    >>> curly_os = make_necklace(["}", "O", "{"], 9)
    >>> print(curly_os)
    <} O { } O { } O {>
    """
    if __________:
           (a)
        return __________
                    (b)
    return Link(__________, make_necklace(__________, __________))
                    (c)                        (d)         (e)
```

(a) (1.0 pt) Fill in blank (a).

(b) (1.0 pt) Fill in blank (b).

(c) (1.0 pt) Fill in blank (c).

(d) (2.0 pt) Fill in blank (d).

(e) (1.0 pt) Fill in blank (e).

5. (4.0 points) File Tree-tionaries

We can represent files and folders on a computer with Trees:

```
Tree("C:", [Tree("Documents", [Tree("hw05.py")]), Tree("pwd.txt")])
```

We can also represent the same folder layout with dictionaries:

```
{"C:": {"Documents": {"hw05.py": "FILE"}, "pwd.txt": "FILE"}}
```

Notice that in this model, we still treat files as dictionary keys, but with the value `"FILE"`.

Complete the implementation of `filetree_to_dict` below, which takes in a Tree `t` representing files and folders, and converts it to the dictionary representation.

```
def filetree_to_dict(t):
    """
    >>> filetree_to_dict(Tree("hw05.py"))
    {"hw05.py": "FILE"}
    >>> filetree = Tree("C:", [Tree("Documents", [Tree("hw05.py")]), Tree("pwd.txt")])
    >>> filetree_to_dict(filetree)
    {"C:": {"Documents": {"hw05.py": "FILE"}, "pwd.txt": "FILE"}}
    """
    res = {}
    if t.is_leaf():
        res[t.label] = ______________
                            (a)
    else:
        nested = {}
        for branch in t.branches:
            nested[branch.label] = ______________(branch)[______________]
                                        (b)                    (c)
        ______________ = nested
             (d)
    return res
```

(a) (1.0 pt) Fill in blank (a)

(b) (1.0 pt) Fill in blank (b)

(c) (1.0 pt) Fill in blank (c)

(d) (1.0 pt) Fill in blank (d)

6. (3.0 points) Mapping Time and Space

(a) The goal of the `maplink_to_list` function below is to map a linked list `lnk` into a Python list, applying a provided function `f` to each value in the list along the way. The function is fully written and passes all its doctests.

```
def maplink_to_list1(f, lnk):
    """Returns a Python list that contains f(x) for each x in Link LNK.
    >>> square = lambda x: x * x
    >>> maplink_to_list1(square, Link(3, Link(4, Link(5))))
    [9, 16, 25]
    """
    new_lst = []
    while lnk is not Link.empty:
        new_lst.append(f(lnk.first))
        lnk = lnk.rest
    return new_lst
```

i. (1.0 pt) What is the order of growth of `maplink_to_list1` in respect to the size of the input linked list `lnk`?

- ◯ Constant
- ◯ Logarithmic
- ◯ Linear
- ◯ Quadratic
- ◯ Exponential

(b) The next function, `maplink_to_list2`, serves the same purpose but is implemented slightly differently. This alternative implementation also passes all the doctests.

```
def maplink_to_list2(f, lnk):
    """Returns a Python list that contains f(x) for each x in Link LNK.
    >>> square = lambda x: x * x
    >>> maplink_to_list2(square, Link(3, Link(4, Link(5))))
    [9, 16, 25]
    """
    def map_link(f, lnk):
        if lnk is Link.empty:
            return Link.empty
        return Link(f(lnk.first), map_link(f, lnk.rest))

    mapped_lnk = map_link(f, lnk)
    new_lst = []
    while mapped_lnk is not Link.empty:
        new_lst.append(mapped_lnk.first)
        mapped_lnk = mapped_lnk.rest
    return new_lst
```

i. (1.0 pt) What is the order of growth of the alternative function, `maplink_to_list2`, in respect to the size of the input linked list `lnk`?

- ◯ Constant
- ◯ Logarithmic
- ◯ Linear
- ◯ Quadratic
- ◯ Exponential

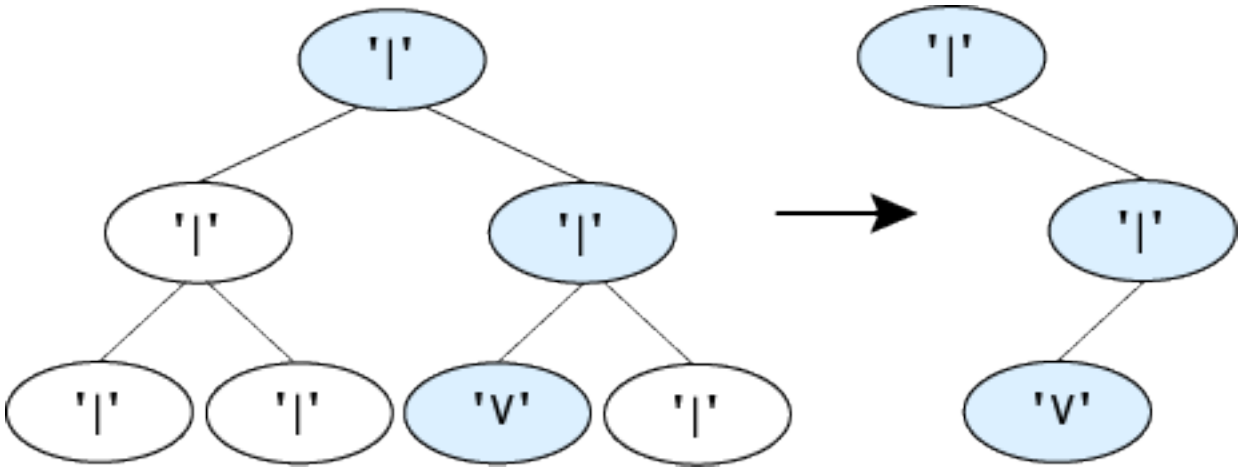
(c) (1.0 pt) Which of the functions requires more **space** to run?

- ◯ `maplink_to_list1`
- ◯ `maplink_to_list2`
- ◯ They both require the same amount of space.

7. (6.0 points) Tulip Mania

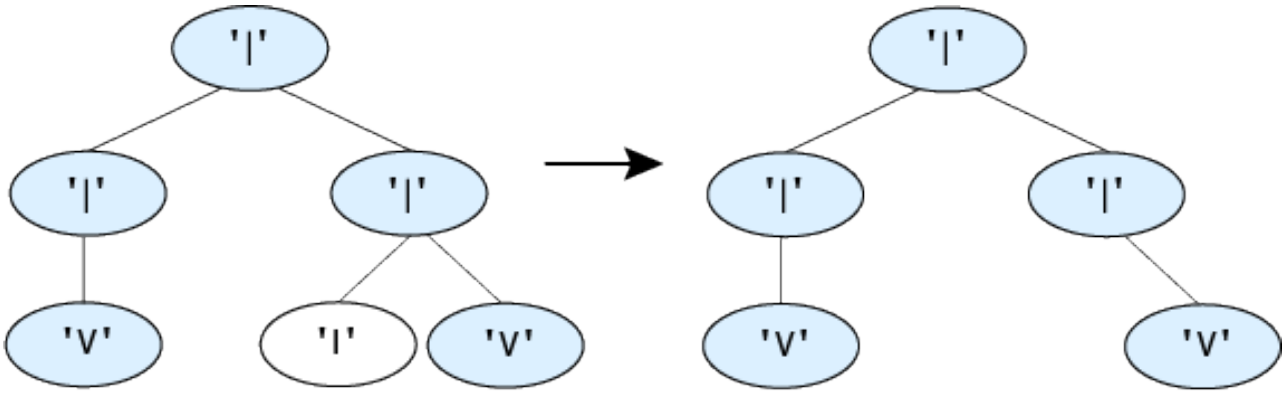
Implement `flower_keeper`, a function that mutates a tree `t` so that the only paths that remain are ones which end in a leaf node with a Tulip flower ('V').

For example, consider this tree where only one path ends in a flower. After calling `flower_keeper`, the tree has only three nodes left, the ones that lead to the flower.

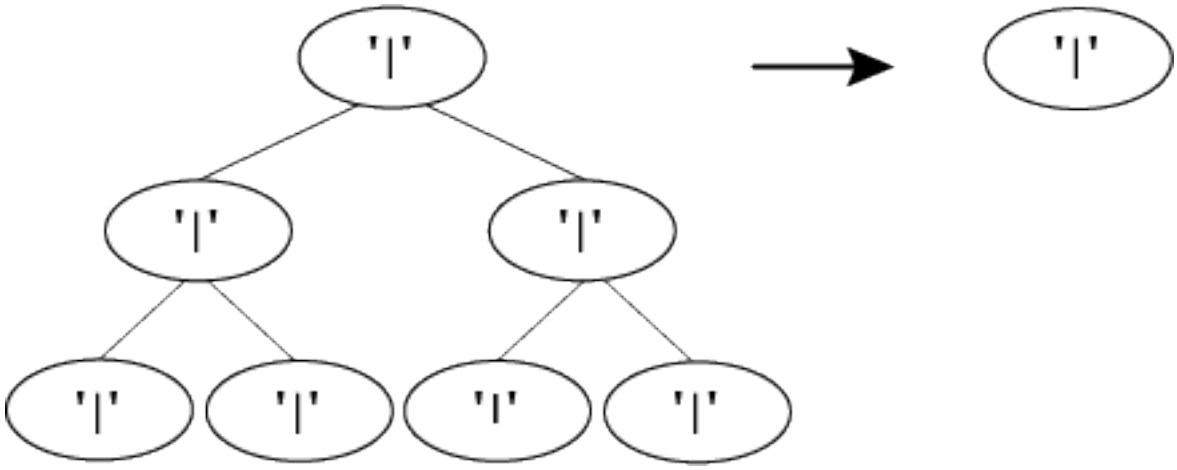


The shaded nodes in the diagram indicate paths that end in flowers.

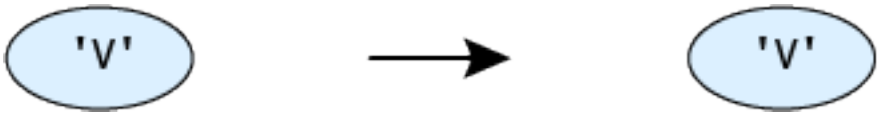
For this tree where two paths end in flowers, the tree keeps both paths that lead to flowers.



For a tree where none of the nodes are flowers, the function removes every branch except the root node.



For a tree with only a single node that is a flower, the function does not remove anything.



Read through the docstring and doctests for additional guidance on completing the implementation of the function. The examples shown above are from the doctests.

```
def flower_keeper(t):
    """
    Mutates the tree T to keep only paths that end in flowers ('V').
    If a path consists entirely of stems ('|'), it must be pruned.
    If T has no paths that end in flowers, the root node is still kept.
    You can assume that a node with a flower will have no branches.

    >>> one_f = Tree('|', [Tree('|', [Tree('|'), Tree('|')]), Tree('|', [Tree('V'), Tree('|')])])
    >>> print(one_f)
    |
      |
        |
        |
      |
        V
        |
    >>> flower_keeper(one_f)
    >>> one_f
    Tree('|', [Tree('|', [Tree('V')])])
    >>> print(one_f)
    |
      |
        V
    >>> no_f = Tree('|', [Tree('|', [Tree('|'), Tree('|')]), Tree('|', [Tree('|'), Tree('|')])])
    >>> flower_keeper(no_f)
    >>> no_f
    Tree('|')
    >>> just_f = Tree('V')
    >>> flower_keeper(just_f)
    >>> just_f
    Tree('V')
    >>> two_f = Tree('|', [Tree('|', [Tree('V')]), Tree('|', [Tree('|'), Tree('V')])])
    >>> flower_keeper(two_f)
    >>> two_f
    Tree('|', [Tree('|', [Tree('V')]), Tree('|', [Tree('V')])])
    """
    for b in __________:
                (a)
        __________
            (b)
    __________ = [__________ for b in __________ if __________]
        (c)           (d)                 (e)           (f)
```

(a) (0.5 pt) Fill in blank (a).

(b) (1.0 pt) Fill in blank (b).

(c) (1.0 pt) Fill in blank (c).

(d) (1.0 pt) Fill in blank (d).

(e) (0.5 pt) Fill in blank (e).

(f) (2.0 pt) Fill in blank (f).

8. (21.0 points) CS 61A Presents The Game of Hoop.

When we introduced the Hog project to you, we designed the rules of the game in a way that would make it possible to implement without using more advanced concepts like OOP and iterators/generators. Now that we've learned these concepts, you can rewrite parts of the game and expand it! The new version, called Hoop, will support *any* number of players greater than or equal to 2.

Assume the following changes from Hog:

- All rules have been taken out except for the Sow Sad rule, which is reiterated below.
- Each player's `strategy` is a function `strategy(own_score, other_scores)` that takes in that player's score and a *list* of the other players' scores, and outputs the number of times that player wishes to roll the dice during their turn. Order doesn't matter for the list of other players' scores.
- You don't have to keep track of the leader, and there is no commentary mechanism.

Sow Sad: If a player rolls any number of 1s in their turn, their score for that set of rolls is a 1 regardless of whatever else they rolled.

To start, take a look at the `HoopPlayer` class, which represents one player in our game:

```
class HoopPlayer:
    def __init__(self, strategy):
        """Initialize a player with STRATEGY, and a starting SCORE of 0. The
        STRATEGY should be a function that takes this player's score and a list
        of other players' scores.
        """
        self.strategy = strategy
        self.score = 0
```

Every game of Hoop will involve rolling a dice as well. Take a look at the `HoopDice` class, which represents a rollable dice that takes a set of values. You'll fill in the blanks in the first part of this question.

```
class HoopDice:
    def __init__(self, values):
        """Initialize a dice with possible values VALUES, and a starting INDEX
        of 0. The INDEX indicates which value from VALUES to return when the
        dice is rolled next.
        """
        self.values = values
        self.index = 0

    def roll(self):
        """Roll this dice. Advance the index to the next step before returning.
        """
        value = ______________
        ______________ = (______________) % ______________
        return value
```

(a) (4.0 points) The Dice: Making it Rollable

Begin by making the `HoopDice` rollable. Fill in the skeleton for the `HoopDice.roll` method below. See above for the rest of `HoopDice`.

```
class HoopDice:
    ...
    def roll(self):
        """Roll this dice. Advance the index to the next step before
        returning.
        >>> five_six = HoopDice([5, 6])
        >>> five_six.roll()
        5
        >>> five_six.index
        1
        >>> five_six.roll()
        6
        >>> five_six.index
        0
        """
        value = _______________
                    (1a)
        _______________ = (_______________) % _______________
            (1b)               (1c)               (1d)
        return value
```

i. (1.0 pt) Fill in blank (1a)

ii. (1.0 pt) Fill in blank (1b)

iii. (1.0 pt) Fill in blank (1c)

iv. (1.0 pt) Fill in blank (1d)

(b) (2.0 points) The Game: Looping Through Players

For the rest of the question, assume that `HoopDice.roll` is implemented correctly, regardless of your answers above.

In the rest of the question, you will work on making the game itself. Take a look at the `HoopGame` class, which takes in the setup for your game and makes it playable. You'll fill in the blanks shortly.

```
class HoopGame:
    def __init__(self, players, dice, goal):
        """Initialize a game with a list of PLAYERS, which contains at least one
        HoopPlayer, a single HoopDice DICE, and a goal score of GOAL.
        """
        self.players = players
        self.dice = dice
        self.goal = goal

    def next_player(self):
        """Infinitely yields the next player in the game, in order."""
        yield from ______________
        yield from ______________

    def get_scores(self):
        """Collects and returns a list of the current scores for all players
        in the same order as the SELF.PLAYERS list.
        """
        # Implementation omitted. Assume this method works correctly.

    def get_scores_except(self, player):
        """Collects and returns a list of the current scores for all players
        except PLAYER.
        """
        return [______________ for pl in ______________ if ______________]

    def roll_dice(self, num_rolls):
        """Simulate rolling SELF.DICE exactly NUM_ROLLS > 0 times. Return sum of
        the outcomes unless any of the outcomes is 1. In that case, return 1.
        """
        outcomes = [______________ for x in ______________]
        ones = [______________ for outcome in outcomes]
        return 1 if ______________(ones) else ______________(outcomes)

    def play(self):
        """Play the game while no player has reached or exceeded the goal score.
        After the game ends, return all players' scores.
        """
        player_gen = self.next_player()
        while max(self.get_scores()) < self.goal:
            player = ______________(player_gen)
            other_scores = self.get_scores_except(player)
            num_rolls = ______________(player.score, other_scores)
            outcome = self.roll_dice(num_rolls)
            ______________ += outcome
        return self.get_scores()
```

To see what the expected behavior of each method is, take a look at its doctests. These will be provided for the relevant methods in each subpart for the rest of the question.

In all following subparts, assume that the following has already been run to set up the game:

```
>>> roll_once_strategy = lambda pl, ops: 1
>>> roll_twice_strategy = lambda pl, ops: 2
>>> always_5 = HoopDice([5])
>>> player1 = HoopPlayer(roll_twice_strategy)
>>> player2 = HoopPlayer(roll_once_strategy)
>>> player3 = HoopPlayer(lambda pl, ops: 6)
>>> game = HoopGame([player1, player2, player3], always_5, 55)
>>> # since we omit the implementation of HoopGame.get_scores, here's what it
>>> # should output:
>>> game.get_scores()
[0, 0, 0]
```

The first `HoopGame` method you'll implement is `HoopGame.next_player`, which gives the game a way to iterate over all of the players playing, in order to give them each a turn in order.

```
class HoopGame:
    ...
    def next_player(self):
        """Infinitely yields the next player in the game, in order.
        >>> player_gen = game.next_player()
        >>> next(player_gen) is player1
        True
        >>> next(player_gen) is player3
        False
        >>> next(player_gen) is player3
        True
        >>> next(player_gen) is player1
        True
        >>> next(player_gen) is player2
        True
        >>> new_player_gen = game.next_player()
        >>> next(new_player_gen) is player1
        True
        >>> next(player_gen) is player3
        True
        """
        yield from ______________
                        (2a)
        yield from ______________
                        (2b)
```

i. (1.0 pt) Fill in blank (2a)

ii. (1.0 pt) Fill in blank (2b)

(c) (3.0 points) The Game: Getting Scores

The `HoopGame.get_scores` method is already fully implemented and provides a way to get a list of the scores of *all* players.

However, each player's strategy relies on the player's *opponents'* scores, so `HoopGame` also needs a method that gives the scores of all players *except* the current one. Fill in the skeleton for `HoopGame.get_scores_except` below. Keep in mind that multiple players may have the same score.

```
class HoopGame:
    ...
    def get_scores_except(self, player):
        """Collects and returns a list of the current scores for all players
        except PLAYER.
        >>> game.get_scores_except(player2)
        [0, 0]
        """
        return [_______________ for pl in _______________ if _______________]
                    (3a)                    (3b)              (3c)
```

As a reminder, here's what the `HoopPlayer` class looks like:

```
class HoopPlayer:
    def __init__(self, strategy):
        """Initialize a player with STRATEGY, and a starting SCORE of 0. The
        STRATEGY should be a function that takes this player's score and a list
        of other players' scores.
        """
        self.strategy = strategy
        self.score = 0
```

i. (1.0 pt) Fill in blank (3a)

ii. (1.0 pt) Fill in blank (3b)

iii. (1.0 pt) Fill in blank (3c)

(d) (5.0 points) The Game: Rolling Dice

Next, give `HoopGame` the ability to roll its dice some given number of times and return the total from rolling all those times. Recall that the dice in the game is just a single dice, represented by the `HoopDice` instance at `self.dice`. Fill in the skeleton for `HoopGame.roll_dice` below:

```
class HoopGame:
    ...
    def roll_dice(self, num_rolls):
        """Simulate rolling SELF.DICE exactly NUM_ROLLS > 0 times. Return sum of
        the outcomes unless any of the outcomes is 1. In that case, return 1.
        >>> game.roll_dice(4)
        20
        """
        outcomes = [_____________ for x in _____________]
                        (4a)                    (4b)
        ones = [_____________ for outcome in outcomes]
                    (4c)
        return 1 if _____________(ones) else _____________(outcomes)
                        (4d)                    (4e)
```

i. (1.0 pt) Fill in blank (4a)

ii. (1.0 pt) Fill in blank (4b)

iii. (1.0 pt) Fill in blank (4c)

iv. (1.0 pt) Fill in blank (4d)

v. (1.0 pt) Fill in blank (4e)

(e) (3.0 points) The Game: Putting it all Together

Finally, it's time to play the game! Recall that a player's strategy depends on two inputs: the player's own score, and a list of the scores of the player's opponents. Using existing attributes wherever possible, fill in the skeleton for `HoopGame.play` below:

```
class HoopGame:
    ...
    def play(self):
        """Play the game while no player has reached or exceeded the goal score.
        After the game ends, return all players' scores.
        >>> game.play()
        [20, 10, 60]
        """
        player_gen = self.next_player()
        while max(self.get_scores()) < self.goal:
            player = _______________(player_gen)
                        (5a)
            other_scores = self.get_scores_except(player)
            num_rolls = _______________(player.score, other_scores)
                           (5b)
            outcome = self.roll_dice(num_rolls)
            _______________ += outcome
                (5c)
        return self.get_scores()
```

i. (1.0 pt) Fill in blank (5a)

ii. (1.0 pt) Fill in blank (5b)

iii. (1.0 pt) Fill in blank (5c)

(f) (4.0 points) The Dice: It's Broken!

Now that you've finished implementing the game, you'll implement a new kind of `BrokenHoopDice`. This dice extends `HoopDice`, but is broken! That is, it alternates between behaving like a normal dice and returning `when_broken` value between turns.

For this part of the question, you may not use `=` or `lambda`.

```
class BrokenHoopDice(HoopDice):
    def __init__(self, values, when_broken):
        ______________(values)
             (6a)
        self.when_broken = when_broken
        self._______________ = False
                  (6b)

    def roll(self):
        """
        >>> broken = BrokenHoopDice([5, 6, 7], 11)
        >>> broken.roll()
        5
        >>> [broken.roll() for _ in range(6)]
        [11, 6, 11, 7, 11, 5]
        """
        if self.is_broken:
            self.is_broken = not self.is_broken
            return _______________
                        (6c)
        else:
            self.is_broken = not self.is_broken
            return _______________()
                        (6d)
```

i. (1.0 pt) Fill in blank (6a)

ii. (1.0 pt) Fill in blank (6b)

iii. (1.0 pt) Fill in blank (6c)

iv. (1.0 pt) Fill in blank (6d)

9. (0.0 points) Just for Fun

Draw some cool dice faces for Hoop! This is not for points and will not be graded.

No more questions.