CS 61A Structure and Interpretation of Computer Programs
Spring 2021

Midterm 2 Solutions

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) By writing my name below, I pledge on my honor that I will abide by the rules of this exam and will neither give nor receive assistance. I understand that doing otherwise would be a disservice to my classmates, dishonor me, and could result in me failing the class.

1. (2.0 points) Tuple Trouble

Consider this partially implemented two-line program:

```
greeting_from_staff = ("Good Luck", "You Got This", "Just Breathe")
______ = "Once More with Feeling"
```

The full program results in this error:

```
AttributeError: 'tuple' object has no attribute '__setitem__'
```

(a) What code could fill in the blank to cause that error?

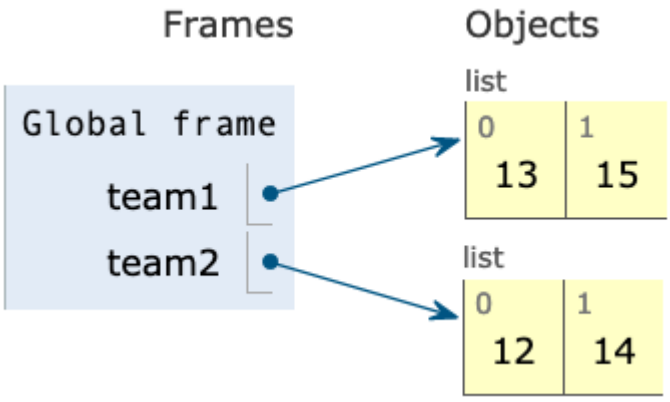
```
greeting_from_staff[2]
```

2. (4.0 points) Teenage Mutant Ninja Lists

A program starts with two list assignments:

```
team1 = [13, 15]
team2 = [12, 14]
```

That code corresponds to this environment diagram:

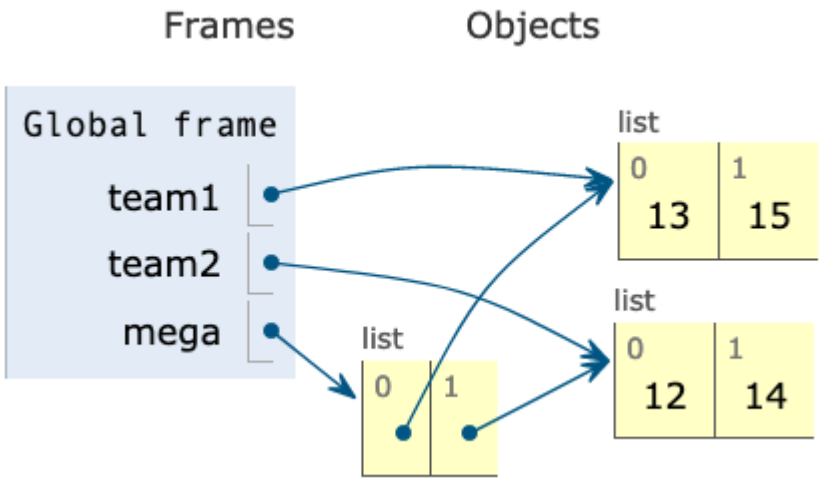


In this question, you'll suggest ways of mutating the lists to result in different environment diagrams.

(a) (1.0 pt)

```
team1 = [13, 15]
team2 = [12, 14]
mega = _____________
```

This modified program corresponds to this diagram:



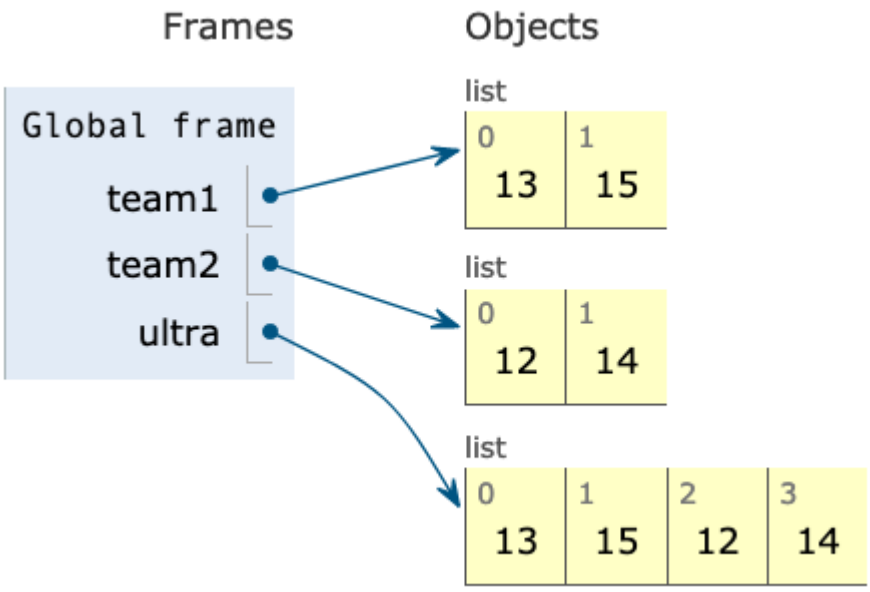
Which of these could fill in the blank? **Select all that apply**

- [x] `[team1, team2]`
- [x] `[team1] + [team2]`
- [ ] `[team1[0:], team2[0:]]`
- [ ] `[team1 + team2]`
- [ ] `[list(team1 + team2)]`
- [ ] `team1, team2`
- [ ] `[list(team1), list(team2)]`
- [ ] `team1 + team2`
- [ ] `team1 - team2`
- [ ] `team1 + team2[:]`
- [ ] `team1[:] + team2`
- [ ] `team1[:] + team2[:]`
- [ ] `team1 += team2`

(b) (1.0 pt)

```
team1 = [13, 15]
team2 = [12, 14]
ultra = _____________
```

This modified program corresponds to this diagram:



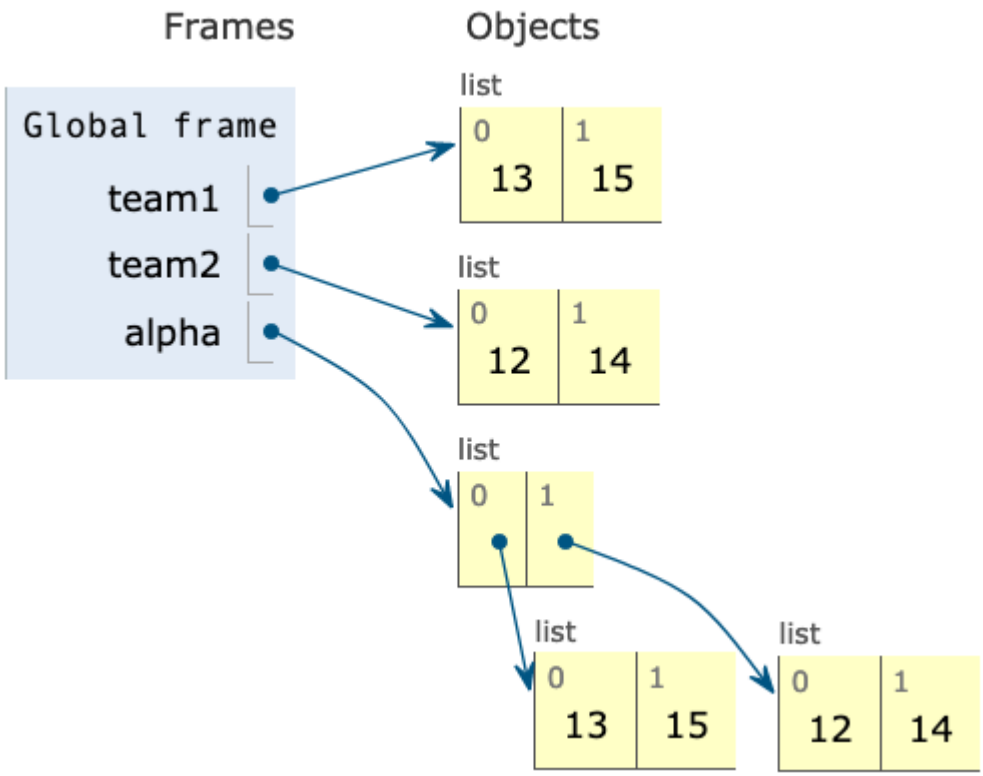
Which of these could fill in the blank? **Select all that apply**

- [x] `team1 + team2`
- [x] `team1[:] + team2[:]`
- [x] `team1 + team2[:]`
- [x] `team1[:] + team2`
- [ ] `[team1, team2]`
- [ ] `[team1[0:], team2[0:]]`
- [ ] `[team1 + team2]`
- [ ] `[list(team1 + team2)]`
- [ ] `team1, team2`
- [ ] `[list(team1), list(team2)]`
- [ ] `team1 - team2`
- [ ] `[team1] + [team2]`
- [ ] `team1 += team2`

(c) (1.0 pt)

```
team1 = [13, 15]
team2 = [12, 14]
alpha = _____________
```

This modified program corresponds to this diagram:

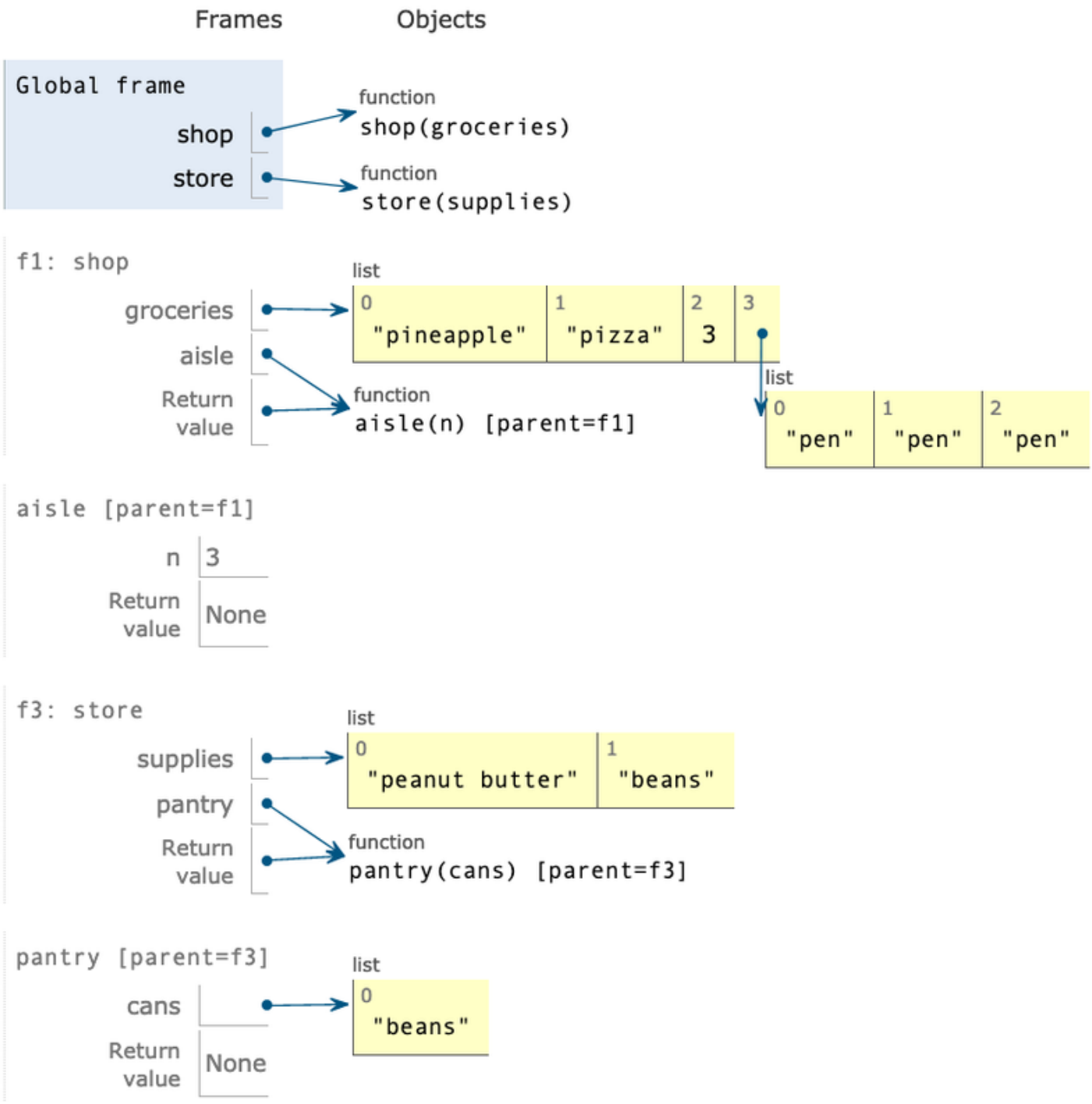


Which of these could fill in the blank? **Select all that apply**

- [x] `[list(team1), list(team2)]`
- [x] `[team1[0:], team2[0:]]`
- [ ] `[team1, team2]`
- [ ] `[team1 + team2]`
- [ ] `[list(team1 + team2)]`
- [ ] `team1, team2`
- [ ] `team1 + team2`
- [ ] `team1 - team2`
- [ ] `[team1] + [team2]`
- [ ] `team1 + team2[:]`
- [ ] `team1[:] + team2`
- [ ] `team1[:] + team2[:]`
- [ ] `team1 += team2`

3. (4.0 points) Higher Order Shopping

The following environment diagram was generated by a program:



Click here to open the diagram in a new window

In this series of questions, you'll fill in the blanks of the program that follows so that its execution matches the environment diagram.

```
def shop(groceries):

    def aisle(n):
```

```
        --------
        #  (a)
        --------
        #  (b)

    return aisle

def store(supplies):

    def pantry(cans):
        --------
        #  (c)
        --------
        #  (d)

    return pantry

shop(["pineapple", "pizza"])(3)
store(["peanut butter"])(["beans"])
```

(a) (2.0 pt) Which of these could fill in blank (a)? **Select all that apply!**

- [x] `groceries.extend([n])`
- [x] `groceries.append(n)`
- [ ] `nonlocal groceries`
- [ ] `nonlocal n`
- [ ] `nonlocal groceries += n`
- [ ] `nonlocal groceries.extend([n])`
- [ ] `nonlocal groceries.append(n)`
- [ ] `nonlocal groceries.extend(n)`
- [ ] `groceries.extend(n)`
- [ ] `groceries.append([n])`
- [ ] `groceries += n`
- [ ] `nonlocal n, groceries`
- [ ] `nonlocal groceries + n`

(b) (1.0 pt) Which of these could fill in blank (b)?

- [x] `groceries.append(["pen"] * n)`
- [ ] `groceries.append("pen" * n)`
- [ ] `groceries.extend("pen" * n)`
- [ ] `groceries.extend(["pen" * n])`
- [ ] `groceries += "pen" * n`
- [ ] `groceries += ["pen" for x in n]`
- [ ] `groceries.extend(["pen" for x in n])`
- [ ] `groceries.append(["pen" for x in n])`
- [ ] `groceries[3:n] = ["pen"]`
- [ ] `groceries[:3:n] = ["pen"]`
- [ ] `groceries[0:n] = ["pen"]`

(c) (1.0 pt) Which of these could fill in blank (c)?

- [x] `nonlocal supplies`
- [ ] `nonlocal store`
- [ ] `nonlocal n`
- [ ] `supplies = list(supplies)`
- [ ] `supplies = supplies[:]`
- [ ] `nonlocal supplies = list(cans)`
- [ ] `nonlocal supplies = list()`
- [ ] `nonlocal list supplies`
- [ ] `nonlocal(supplies)`

(d) (1.0 pt) Which of these could fill in blank (d)?

- [x] `supplies += cans`
- [ ] `nonlocal supplies += cans`
- [ ] `supplies.append(cans)`
- [ ] `nonlocal supplies.append(cans)`
- [ ] `supplies[:] = cans`
- [ ] `nonlocal supplies[:] = cans`
- [ ] `supplies = cans`
- [ ] `nonlocal supplies = cans`
- [ ] `supplies.extend([cans])`
- [ ] `nonlocal supplies.extend([cans])`

4. (8.0 points) Gotta Keep 'Em Separated

Implement `separate()`, a function that accepts two parameters (`separator` and `lnk`) and returns a new linked list where any two consecutive repeated numbers in `lnk` are separated by a link with the value of `separator`. All values in `lnk` will be numbers.

The function should assume the standard CS61A Link definition, viewable here: code.cs61a.org/link_class

```
def separate(separator, lnk):
    """Returns a new linked list (using Link) that separates
    any two consecutive repeated numbers in non-empty LNK
    by inserting a Link with the value of SEPARATOR.
    All values in LNK will be numbers.

    # Case 1: Separates list that starts with pair
    >>> separate(999, Link(1, Link(1)))
    Link(1, Link(999, Link(1)))

    # Case 2: Separates list with overlapping pairs
    >>> separate(999, Link(1, Link(1, Link(1))))
    Link(1, Link(999, Link(1, Link(999, Link(1)))))

    # Case 3: Doesn't mutate the input list
    >>> link = Link(1, Link(1))
    >>> separate(999, link)
    Link(1, Link(999, Link(1)))
    >>> link
    Link(1, Link(1))

    # Case 4: Only separates pairs, not non-pairs
    >>> separate(-999, Link(2, Link(2, Link(4, Link(5)))))
    Link(2, Link(-999, Link(2, Link(4, Link(5)))))

    # Case 5: Separates pairs at end of list
    >>> separate(-1, Link(2, Link(3, Link(4, Link(4)))))
    Link(2, Link(3, Link(4, Link(-1, Link(4)))))

    # Case 6: Returns same-valued list if no pairs found
    >>> separate(999, Link(1, Link(2)))
    Link(1, Link(2))

    # Case 7: Handles single element lists correctly
    >>> lnk = Link(1)
    >>> lnk2 = separate(999, lnk)
    >>> lnk
    Link(1)
    >>> lnk is not lnk2
    True
    """
```

(a) Here's an approximate structure of a recursive solution, if that helps guide your implementation. It does not reflect the exact number of lines or indentation. **Your solution must use a recursive approach, not an iterative approach. An iterative approach will receive 0 points.**

```
if ______:
  _____________
elif ____:
  _____________
elif ____:
  _____________
_______________
```

You can use code.cs61a.org to try out your code and see if it passes the doctests. You can then paste the code here.

```
def separate(separator, lnk):
   if lnk is Link.empty:
      return Link.empty
   elif lnk.rest is Link.empty:
      return Link(lnk.first)
   elif lnk.first == lnk.rest.first:
      return Link(lnk.first, Link(separator, separate(separator,
lnk.rest)))
   return Link(lnk.first, separate(separator, lnk.rest))
```

5. (8.0 points) Online Classes

This question is inspired by online education websites that offer a range of materials such as videos, articles, exercises, and quizzes.

We will represent two kinds of educational content as a Python class: `Video` and `Exercise`. Each of those classes inherits from the base class `LearnableContent`.

The partially implemented classes are shown below. You'll fill out the class definitions using inheritance and attributes. **Always use the definition from the base class if it's reasonable to do so.**

```
class LearnableContent:
    """A base class for specific kinds of learnable content.
    All kinds have title and author attributes,
    but each kind may have additional attributes.
    """

    def __init__(self, title, author):
        self.title = title
        self.author = author

    def __str__(self):
        return f"{self.title} by {self.author}"

class Exercise(LearnableContent):
    """
    >>> lambda_calc = Exercise("Lambda Calculus", "Rosie F", 5)
    >>> lambda_calc.title
    'Lambda Calculus'
    >>> lambda_calc.author
    'Rosie F'
    >>> lambda_calc.num_questions
    5
    >>> str(lambda_calc)
    'Lambda Calculus by Rosie F'
    """

    def __init__(self, title, author, num_questions):
        ________________________
        #          (a)
        ________________________
        #          (b)

class Video(LearnableContent):
    """
    >>> vid = Video("The Golden Ratio", "Sal Khan", 881)
    >>> Video.license
    'CC-BY-NC-SA'
    >>> vid.title
    'The Golden Ratio'
    >>> vid.author
    'Sal Khan'
    >>> vid.num_seconds
    881
    >>> str(vid)
    'The Golden Ratio by Sal Khan (881 seconds)'
    """
```

```
____________________
#      (c)

def __init__(self, title, author, num_seconds):
    ________________________
    #          (d)
    ________________________
    #          (e)

def __str__(self):
    ________________________
    #          (f)
```

Please check your work on code.cs61a.org and run the doctests there.

(a) (1.0 pt) What line of code could go in blank (a)? **You only need a single line of code.**

```
super().__init__(title, author)
```

(b) (1.0 pt) What line of code could go in blank (b)? **You only need a single line of code.**

```
self.num_questions = num_questions
```

(c) (1.0 pt) What line of code could go in blank (c)? **You only need a single line of code.**

```
license = 'CC-BY-NC-SA'
```

(d) (2.0 pt) What line of code could go in blank (d)? **You only need a single line of code.**

```
super().__init__(title, author)
```

(e) (1.0 pt) What line of code could go in blank (e)? **You only need a single line of code.**

```
self.num_seconds = num_seconds
```

(f) (2.0 pt) What line of code could go in blank (f)? **You only need a single line of code.**

You may use any form of string formatting that you are comfortable with, but your solution must use `super()`.

```
return super().__str__() + f" ({self.num_seconds} seconds)"
```

6. (6.0 points) Gas Composition

This question is inspired by research projects that are tracking the emissions of greenhouse gases into the environment so that they can identify the biggest sources of greenhouse gases and find ways to sustainably reduce them.

We have written two classes to help track gas emissions: `EmissionSource` and `EmissionsTracker`.

The `EmissionSource` class is fully implemented. Each instance stores a name, a dictionary of emission rates per hour per gas type, and the number of hours it's been running (initialized to 0). Read through the implementation below:

```
class EmissionSource:

    def __init__(self, name, co2, ch4, n2o):
        self.name = name
        self.emissions = {
            "carbon dioxide": co2,  # kg CO2/ton/hour
            "methane": ch4,         # g CH4/ton/hour
            "nitrous oxide": n2o,   # g N2O/ton/hour
        }
        self.hours = 0

    def run_for(self, num_hours):
        self.hours += num_hours

    def calc_emissions(self, gas):
        return self.emissions.get(gas, 0) * self.hours
```

The `EmissionsTracker` class is partially implemented, but has doctests that demonstrate how it should work. It's job is to track emission_source_list and calculate statistics about their emissions. Read through the doctests, docstrings, and partial implementation below:

```
class EmissionsTracker:
    """
    >>> tracker = EmissionsTracker()
    >>> pp1 = EmissionSource("Anthracite Coal", 2602, 276, 40)
    >>> pp2 = EmissionSource("Lignite Coal", 1389, 156, 23)
    >>> tracker.add_sources([pp1, pp2])
    >>> pp1.run_for(5)
    >>> pp2.run_for(6)

    # Case 1: Calculate max for 2 added so far
    >>> tracker.calc_emissions("methane")
    [('Anthracite Coal', 1380), ('Lignite Coal', 936)]
    >>> tracker.calc_max_source("methane")
    ('Anthracite Coal', 1380)

    # Case 3: Calculate emissions for all 3
    >>> pp3 = EmissionSource("Plastics", 2850, 1216, 160)
    >>> pp3.run_for(3)
    >>> tracker.add_sources([pp3])
    >>> tracker.calc_emissions("carbon dioxide")
    [('Anthracite Coal', 13010), ('Lignite Coal', 8334), ('Plastics', 8550)]
    >>> tracker.calc_emissions("methane")
    [('Anthracite Coal', 1380), ('Lignite Coal', 936), ('Plastics', 3648)]
    >>> tracker.calc_emissions("nitrous oxide")
    [('Anthracite Coal', 200), ('Lignite Coal', 138), ('Plastics', 480)]
```

```
# Case 4: Calculate max for all 3
>>> tracker.calc_max_source("carbon dioxide")
('Anthracite Coal', 13010)
>>> tracker.calc_max_source("methane")
('Plastics', 3648)
>>> tracker.calc_max_source("nitrous oxide")
('Plastics', 480)
"""
def __init__(self):
    self.emission_source_list = []

def add_sources(self, sources_to_add):
    """ Adds the list of SOURCES_TO_ADD to the end of self.emission_source_list"""
    ___________________________
    #         (a)

def calc_emissions(self, gas):
    """ Returns a list of tuples where the first element is the name of
    the emission_src and the second element is the total amount of emissions
    from that type of GAS."""
    ___________________________
    #         (b)

def calc_max_source(self, gas):
    """ Returns a tuple where the first element is the name of the emission_src
    with the highest emissions and the second element is the total amount
    of emissions from that emission_src for that type of GAS."""
    return max(_______________)
    #               (c)
```

The following questions will ask you to fill in the blanks for `EmissionsTracker`. You can try out the code on code.cs61a.org and run the doctests to verify that your proposed code does indeed work.

(a) **(2.0 pt)** What line of code could go in blank (a)? **You only need a single line of code.**

```
self.emission_source_list.extend(sources_to_add)
```

(b) **(2.0 pt)** What line of code could go in blank (b)? **You only need a single line of code. Your code must use methods from the `EmissionSource` class if it's reasonable to do so.**

```
return [ (source.name, source.calc_emissions(gas)) for source in
self.emission_source_list]
```

(c) **(2.0 pt)** What expression could go in blank (c)? You may want to consult the Python documentation for the max function.

```
self.calc_emissions(gas), key=lambda tup: tup[1]
```

7. (4.0 points) Business as Usual

Modern office supplies companies use software to track their sales inventory and can use object oriented programming to represent the types of products they sell.

We will represent one type of product as a Python class: `PaperReam`.

The class is partially implemented below. Please read through the code and doctests.

```
class PaperReam:
    """
    >>> ream = PaperReam("red", 200)
    >>> ream.color_name
    'red'
    >>> ream.num_sheets
    200
    >>> ream
    PaperReam('red', 200)
    """

    def __init__(self, color_name, num_sheets):
        self.color_name = color_name
        self.num_sheets = num_sheets

    --------------------
    #          (a)
        ----------------
        #      (b)
```

The following questions will ask you to fill in the blanks for `PaperReam`. You can try out the code on code.cs61a.org and run the doctests to verify that your proposed code does indeed work.

(a) (2.0 pt) What line of code could go in blank (a)? **You only need a single line of code, since it should be a function header.**

```
def __repr__(self):
```

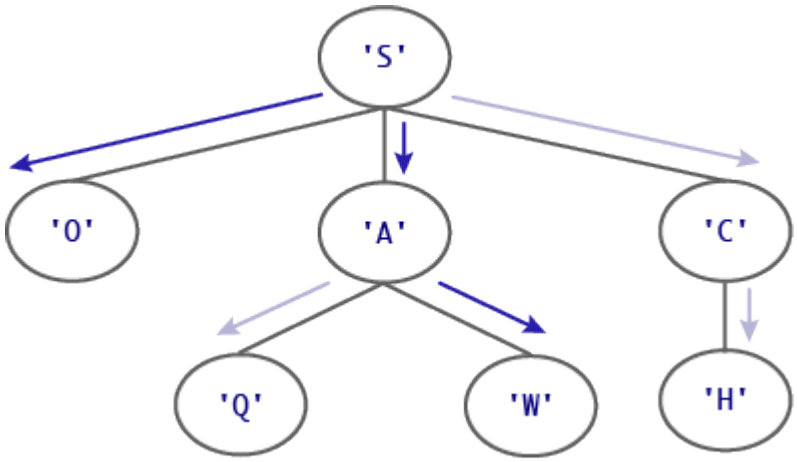
(b) (2.0 pt) What line of code could go in blank (b)? **You only need a single line of code.**

```
return f"PaperReam({repr(self.color_name)}, {repr(self.num_sheets)})"
OR
return f"PaperReam('{self.color_name}', {self.num_sheets})"
```

8. (7.0 points) The Tree of L-I-F-E

Implement `word_finder`, a generator function that yields each word that can be formed by following a path in a tree from the root to a leaf, where the words are specified in a list.

When given the tree shown in the diagram below and a word list that includes 'SO' and 'SAW', the function should first yield 'SO' and then yield 'SAW'.



Please read through the function header and doctests below. We have provided quite a few doctests to test different situations and demonstrate how the function should work. You can always call `draw(t)` on a particular tree object on code.cs61a.org to help you visualize its structure and understand the results of a doctest.

```
def word_finder(letter_tree, words_list):
    """ Generates each word that can be formed by following a path
    in TREE_OF_LETTERS from the root to a leaf,
    where WORDS_LIST is a list of allowed words (with no duplicates).

    # Case 1: 2 words found
    >>> words = ['SO', 'SAT', 'SAME', 'SAW', 'SOW']
    >>> t = Tree("S", [Tree("O"), Tree("A", [Tree("Q"), Tree("W")]), Tree("C", [Tree("H")])])
    >>> gen = word_finder(t, words)
    >>> next(gen)
    'SO'
    >>> next(gen)
    'SAW'
    >>> list(word_finder(t, words))
    ['SO', 'SAW']

    # Case 2: No words found
    >>> t = Tree("S", [Tree("I"), Tree("A", [Tree("Q"), Tree("E")]), Tree("C", [Tree("H")])])
    >>> list(word_finder(t, words))
    []

    # Case 3: Same word twice
    >>> t = Tree("S", [Tree("O"), Tree("O")] )
    >>> list(word_finder(t, words))
    ['SO', 'SO']

    # Case 4: Words that start the same
    >>> words = ['TAB', 'TAR', 'BAT', 'BAR', 'RAT']
    >>> t = Tree("T", [Tree("A", [Tree("R"), Tree("B")])])
    >>> list(word_finder(t, words))
```

```
['TAR', 'TAB']

# Case 5: Single letter words
>>> words = ['A', 'AN', 'AH']
>>> t = Tree("A")
>>> list(word_finder(t, words))
['A']

# Case 6: Words end in leaf
>>> words = ['A', 'AN', 'AH']
>>> t = Tree("A", [Tree("H"), Tree("N")])
>>> list(word_finder(t, words))
['AH', 'AN']

# Case 7: Words start at root
>>> words = ['GO', 'BEARS', 'GOB', 'EARS']
>>> t = Tree("B", [Tree("E", [Tree("A", [Tree("R", [Tree("S")])])])])
>>> list(word_finder(t, words))
['BEARS']

# Case 8: This special test ensures that your solution does *not*
#  pre-compute all the words before yielding the first one.
#  If done correctly, your solution should error only when it
#  tries to find the second word in this tree.
>>> words = ['SO', 'SAM', 'SAT', 'SAME', 'SAW', 'SOW']
>>> t = Tree("S", [Tree("O"), Tree("A", [Tree("Q"), Tree(1)]), Tree("C", [Tree(1)])])
>>> gen = word_finder(t, words)
>>> next(gen)
'SO'
>>> try:
...     next(gen)
... except TypeError:
...     print("Got a TypeError!")
... else:
...     print("Expected a TypeError!")
Got a TypeError!
"""
```

(a) Here is a skeleton of a correct solution:

```
def word_finder(letter_tree, words_list):
    def _________(____):
        _______________ # Optional
        if ____________:
            yield _______________
        for ____________:
            yield from _________

    yield from __________
```

We highly encourage you to follow the skeleton. You may diverge from the skeleton if your approach is correct and passes the doctests, but a non-working solution that doesn't follow the skeleton will not receive partial credit.

A correct solution should **not** first find all the words and then yield them; it should instead yield whenever it finds a new word. **Your solution may receive 0 points if it pre-computes the words.**

Your function should use the standard CS61A Tree definition, viewable here: code.cs61a.org/tree_class

Please use code.cs61a.org to try out your code. You can then paste the code here.

```
def word_finder(letter_tree, words_list):
   def string_builder(t, str):
      str += t.label
      if t.is_leaf() and str in words_list:
         yield str
      for b in t.branches:
         yield from string_builder(b, str)
   yield from string_builder(letter_tree, "")
```

9. (4.0 points) Light It Up

An individually addressed LED light strip allows programmers to write programs to change the color of each individual LED light.

We could represent a strip of lights using a `SingleLED` class and an `LEDLightStrip` class:

```
class SingleLED:

    def __init__(self, brightness, color):
        self.brightness = brightness
        self.color = color

class LEDLightStrip:

    def __init__(self, leds):
        self.leds = leds
        self.length = len(leds)
        self.is_on = False

    def toggle(self):
        self.is_on = not self.is_on
```

Then a short 3-light strip could be constructed like so:

```
light_strip = LEDLightStrip([SingleLED(0, "blue"), SingleLED(0, "red"), SingleLED(0, "green")])
```

We would like to be able to iterate through the light object, one light at a time, like so:

```
for led in light_strip:
    led.brightness += 20
```

But with our current code, we see this error:

```
TypeError: 'LEDLightStrip' object is not iterable
```

(a) (2.0 pt) We need to define a new method on `LEDLightStrip` to make it iterable. Here's a skeleton of that method:

```
    def ______________(self):
        #     (a)
        ______________
        ______________
        #     (b)
```

What method name should go in blank (a)?

`__iter__`

(b) (2.0 pt) What code could go in blank (b) such that the for loop works? *You can use a single line of code or multiple lines of code, as long as the code is correct.*

```
for led in self.leds:
yield led
OR:
return iter(self.leds)
```

10. (3.0 points) May I Take Your Order?

The following two functions can be used for inserting items into a linked list. The first function `insert_front()` always inserts at the front of the list while the second function `insert_back()` always inserts at the back of the list. Both functions use the standard CS61A Link definition, viewable here: code.cs61a.org/link_class

```
def insert_front(link, new_val):
    """Add NEW_VAL to the front of non-empty LINK, mutating the list.

    >>> link = Link(-1, Link(-3, Link(-5)))
    >>> insert_front(link, -0)
    >>> link
    Link(-0, Link(-1, Link(-3, Link(-5))))
    >>> insert_front(link, -3)
    >>> link
    Link(-3, Link(-0, Link(-1, Link(-3, Link(-5)))))
    """
    original_first = link.first
    link.first = new_val
    link.rest = Link(original_first, link.rest)

def insert_back(link, new_val):
    """Add NEW_VAL to the end of non-empty LINK, mutating the list.

    >>> link = Link(-1, Link(-3, Link(-5)))
    >>> insert_back(link, -0)
    >>> link
    Link(-1, Link(-3, Link(-5, Link(-0))))
    >>> insert_back(link, -3)
    >>> link
    Link(-1, Link(-3, Link(-5, Link(-0, Link(-3)))))
    """
    while link.rest is not Link.empty:
        link = link.rest
    link.rest = Link(new_val)
```

(a) (1.0 pt) What is the order of growth of `insert_front()` as the input list size changes?

- (●) (1)
- ( ) $(\log_2 n)$
- ( ) (n)
- ( ) $(n \log_2 n)$
- ( ) (n^2)
- ( ) (n^3)
- ( ) (2^n)
- ( ) $(n!)$

(b) (1.0 pt) What is the order of growth of `insert_back()` as the input list size changes?

- ○ (1)
- ○ $(\log_2 n)$
- ● (n)
- ○ $(n \log_2 n)$
- ○ (n^2)
- ○ (n^3)
- ○ (2^n)
- ○ $(n!)$

(c) (1.0 pt) Which of the following correctly describes the two functions?

- ● Both functions perform destructive operations on the input parameter, a mutable object.
- ○ Both functions perform non-destructive operations on the input parameter, a mutable object.
- ○ The first function performs a non-destructive operation while the second function performs a destructive operation.
- ○ The first function performs a destructive operation while the second function performs a non-destructive operation.

11. (1.0 points) Extra Point!

(a) (1.0 pt) What phrase might be suggested by "6 = Q. in the S. M."?

6 quarks in the standard model.

No more questions.