CS 61A Structure and Interpretation of Computer Programs
Summer 2024

Final 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) What is your @berkeley.edu email address?

(d) Name and SID of the person to your left (or N/A).

(e) Name and SID of the person to your right (or N/A).

(f) Sign your name to confirm that all work on this exam will be your own.

1. (8.0 points) Phrase Phonetics

Assume the following code has been executed. No error occurs when executing this code block.

```
phrases = ['sweet', 'dreams', 'good', 'night', '!']

def crowdstrike():
    while phrases:
        yield phrases.pop()

i1 = iter(phrases)
i2 = iter(phrases[1:])
```

`What Would Python Display`? Write the output displayed by evaluating each expression below.

- If an error occurs, write `"Error"`, but include all output displayed before the error.
- If evaluation would run forever, write `"Forever"`.
- To display an iterator object, write `"Iterator"`.
- To display a generator object, write `"Generator"`.

Assume the expressions are evaluated in order in the same interactive session, and so evaluating an earlier expression may affect the result of a later one.

Hint: Draw it out!

(a) (1.0 pt)

```
>>> next(i1) + next(i2)
```

```
'sweetdreams'
```

(b) (1.0 pt)

```
>>> phrases.insert(1, 'question')
>>> next(i2) + next(i1)
```

```
'goodquestion'
```

(c) (2.0 pt)

```
>>> c = crowdstrike()
>>> next(i2) + next(i1) + next(c)
```

```
'nightdreams!'
```

(d) (2.0 pt)

```
>>> list(c)
```

```
['night', 'good', 'dreams', 'question', 'sweet']
```

(e) (2.0 pt)

```
print(next(i2)) or print(next(i1))
```

```
!
StopIteration
```

2. (8.0 points) Sweet Diadreams

Draw the environment diagram for the code block below and then answer the questions that follow. Your diagram will not be graded.

If an error occurs, answer the following questions according to the environment diagram you drew up until the error.

```
def sweet(x, y):

    def dreams(z, f):
        return f(z)

    while x + y > 0:
        y = y - dreams(x + 2, lambda x: x - y)

    return x + y

a = 1
b = 2
a = sweet(a, b)
```

Blank Space for Diagram:

(a) (2.0 pt) What is the value of `a` in Global?

0

(b) (1.0 pt) What is the return value of `f2`?

1

(c) (1.0 pt) What is the return value of `f3`?

1

(d) (1.0 pt) What is the return value of `f4`?

2

(e) (1.0 pt) What is the return value of `f5`?

2

(f) (1.0 pt) Which frame is the parent frame of the lambda function? **Note: These options may not cover every frame that is opened**.

- ○ `Global`
- ● `f1`
- ○ `f2`
- ○ `f3`
- ○ None of the above.

(g) (1.0 pt) How many times is `dreams` called?

- ○ `0`
- ○ `1`
- ● `2`
- ○ `3`
- ○ `4` or more

3. (10.0 points) Movie Theater Seating

Laryn, Raymond, and Charlotte want to watch a movie in theaters together but can't figure out how to seat themselves.

Implement `movie_seating`, a function that takes in a list of strings, `people`, and a list of integers, `seats`. `movie_seating` returns a list of lists of all the possible ways to arrange the people amongst the open seats. In order to be considered a valid seating arrangement, all people must have a seat.

A seat with a value `0` is open. A seat with a value `-1` is not open. The arrangements can be returned in any order.

Hint: Use `remove_person`, which takes in a list of strings people (representing people) and a string `to_remove` (representing a person to remove). This function returns a new list that includes all the people from the original list except the specified person to remove.

```
def remove_person(people, to_remove):
    return [person for person in people if person != to_remove]

def movie_seating(people, seats):
    """
    >>> movie_seating(['L', 'R'], [0, 0])
    [['L', 'R'], ['R', 'L']]
    >>> movie_seating(['L', 'C'], [0, -1, 0])
    [['L', -1, 'C'], ['C', -1, 'L']]
    >>> movie_seating(['L', 'R', 'C'], [0, -1, 0])
    []
    >>> movie_seating(['L', 'R', 'C'], [0, 0, 0])
    [['L', 'R', 'C'], ['L', 'C', 'R'], ['R', 'L', 'C'],
    ['R', 'C', 'L'], ['C', 'L', 'R'], ['C', 'R', 'L']]
    >>> movie_seating(['R', 'C'], [0, 0, 0])
    [['R', 'C', 0], ['R', 0, 'C'], ['C', 'R', 0], ['C', 0, 'R'],
    [0, 'R', 'C'], [0, 'C', 'R']]
    """

    if not seats and people:

        return []

    if not people:

        return [seats]

    skip_first_seat = __________
                          (a)
    if seats[0] == -1:

        return [__________ for arrangement in skip_first_seat]
                    (b)
    ways = []

    for choice in people:

        use_first_seat = __________
                             (c)
        ways.__________([__________ for arrangement in use_first_seat])
               (d)          (e)
    ways.__________([__________ for arrangement in skip_first_seat])
```

```
        (f)          (g)
return ways
```

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

```
movie_seating(people, seats[1:])
```

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

```
[-1] + arrangement
```

(c) (3.0 pt) Fill in blank (c).

```
movie_seating(remove_person(people, choice), seats[1:])
```

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

- ○ `append`
- ● `extend`
- ○ `pop`
- ○ `remove`
- ○ `insert`

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

```
[choice] + arrangement
```

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

- ○ `append`
- ● `extend`
- ○ `pop`
- ○ `remove`
- ○ `insert`

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

```
[0] + arrangement
```

4. (11.0 points) Linked Max Composite Value Path

Implement `link_path_tree` which takes in a Tree object, `t`, and an integer, `val`. The labels of `t` are one-argument functions that take in an integer and return an integer. `link_path_tree` should mutate `t` such that each label of `t` is a Linked List containing a path from the current node to the leaf with maximal "composite value".

For a node `n`, the "composite value" of `n` is the result of successively passing `val` through each one-argument function in the path from the root to `n`. For example, if the path from root to `n` consists of 3 functions, `f` –> `g` –> `h` where `f` is the original label of the root node and `h` is the original label of `n`, the "composite value" of `n` is `h(g(f(val)))`.

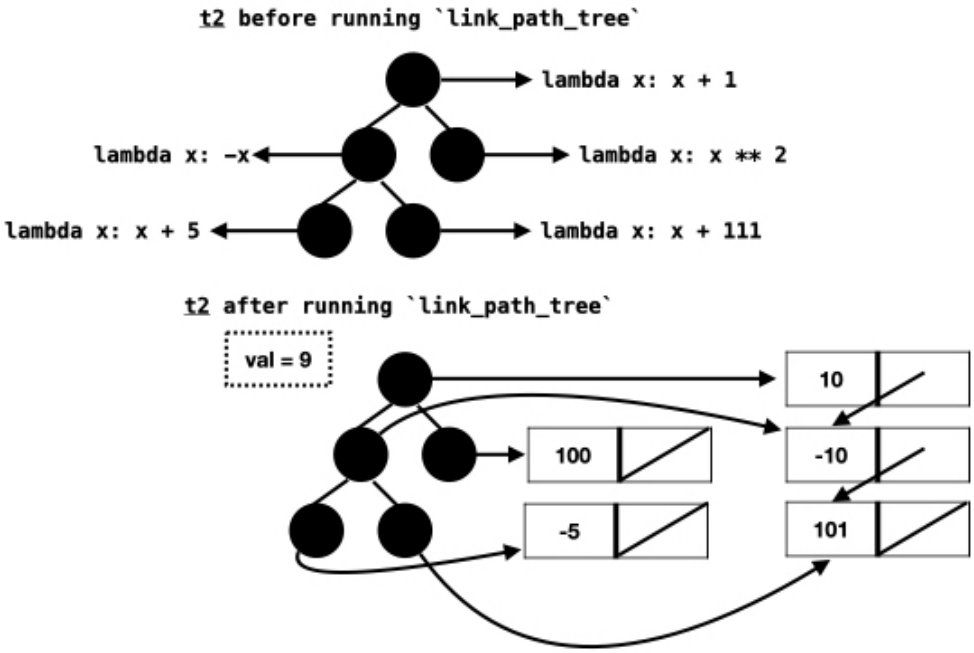
After `link_path_tree` finishes executing, the label of each `n` should be updated to be a Linked List where the first value is the "composite value" of `n`, and the rest of the Linked List is the path from `n` to the leaf in the subtree rooted at `n` with maximal "composite value".

Hint: Use `get_tail`. `get_tail` takes in a Linked List, `lnk`, and returns the value in the last Link. `lnk` must have at least one Link.

```
def get_tail(lnk):
    """
    >>> get_tail(Link(1))
    1
    >>> get_tail(Link(1, Link(2)))
    2
    """

    while lnk.rest is not Link.empty:
        lnk = lnk.rest
    return lnk.first
```

Here is a visualization of one of the doctests.



```
def link_path_tree(t, val):
    """
    >>> t = Tree(lambda x: x + 1, [Tree(lambda x: x + 2), Tree(lambda x: x + 3)])
    >>> link_path_tree(t, 0)
    >>> t # the path with maximal composite value starts from the root
    and ends at the second branch of the root
    Tree(Link(1, Link(4)), [Tree(Link(3)), Tree(Link(4))])
    >>> t.label.rest is t.branches[1].label
    True
    >>> t2 = Tree(lambda x: x + 1, [Tree(lambda x: -x, [Tree(lambda x: x + 5),
    Tree(lambda x: x + 111)]), Tree(lambda x: x ** 2)])
    >>> link_path_tree(t2, 9)
    >>> t2
    Tree(Link(10, Link(-10, Link(101))), [Tree(Link(-10, Link(101)), [Tree(Link(-5)),
    Tree(Link(101))]), Tree(Link(100))])
    >>> t2.label.rest is t2.branches[0].label
    True
    >>> t2.label.rest.rest is t2.branches[0].branches[1].label
    True
    """

    applied = __________
                  (a)
    if t.is_leaf():

        t.label = __________(applied)
                      (b)
    else:
        for b in t.branches:
            ----------
                (c)
        t.label = __________(__________, max(__________, key=__________))
                      (d)       (e)              (f)             (g)
```

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

```
t.label(val)
```

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

```
Link
```

(c) **(2.0 pt)** Fill in blank (c).

```
link_path_tree(b, applied)
```

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

```
Link
```

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

```
applied
```

(f) **(3.0 pt)** Fill in blank (f). **Hint**: Use a list comprehension.

```
[b.label for b in t.branches] -OR- map(lambda b: b.label. t.branches)
```

(g) **(2.0 pt)** Fill in blank (g).

```
get_tail
```

5. (21.0 points) CS 61A Web Browser

You are a talented web developer for CS 61A Inc. and have been tasked with modeling a web browser with object-oriented programming in Python. Fill out the classes below to satisfy the class descriptions and doctests.

(a) (7.0 points) Browser

`Browser`'s can visit pages which are represented as strings. `Browser`'s store their browsing history of visited pages in a Linked List so that when `str` is called on a `Browser` instance, the entire history of visited webpages can be displayed in order of most recently visited to least recently visited. `Browser`'s can also go back one webpage at a time, removing the most recently visited page from the browsing history each time.

The `visit` and `back` methods additionally return a zero-argument "undo" function that undoes the last action performed (either visiting or going back). "undo" functions themselves also return another "undo" function. Undoing an "undo" results in a net-zero effect (e.g., visiting a page, undoing the visit, then undoing the "undo" is the same as just visiting the page). Implement the `Browser` class.

```
class Browser:
    """
    >>> browser = Browser()
    >>> print(browser)

    >>> _ = browser.visit('cs61a.org')
    >>> _ = browser.visit('oh.cs61a.org')
    >>> print(browser)
    oh.cs61a.org<-cs61a.org
    >>> undo = browser.back()
    >>> print(browser)
    cs61a.org
    >>> undo = undo()
    >>> print(browser)
    oh.cs61a.org<-cs61a.org
    >>> undo = undo()
    >>> print(browser)
    cs61a.org
    >>> _ = undo()() # undo'ing an undo cancels it out and does nothing
    >>> print(browser)
    cs61a.org
    """

    def __init__(self):
        self.browsing_history = Link.empty

    def visit(self, page):
        self.browsing_history = __________
                                    (a)
        return __________
                  (b)
    def back(self):
        page = self.browsing_history.first
        self.browsing_history = __________
                                    (c)
        return __________
                  (d)
```

```
def __str__(self):
    display = ''
    head = self.browsing_history
    while head is not Link.empty:
        if __________ is not Link.empty:
               (e)
            display += __________
                           (f)
        else:
            display += __________
                           (g)
        head = head.rest
    return display
```

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

```
Link(page, self.browsing_history)
```

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

```
lambda: self.back() -OR- self.back
```

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

```
self.browsing_history.rest
```

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

```
lambda: self.visit(page)
```

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

- ◯ `head`
- ● `head.rest`
- ◯ `head.rest.rest`
- ◯ `self.browsing_history`
- ◯ `self.browsing_history.rest`
- ◯ `self.browsing_history.rest.rest`

vi. **(1.0 pt)** Fill in blank (f).

```
head.first + '<-'
```

vii. (1.0 pt) Fill in blank (g).

```
head.first
```

(b) (4.0 points) Chrome

`Chrome`'s are `Browser`'s that always begin their browsing from 'google.com'. Additionally, `Chrome`'s can interleave their browsing history with another browser, resulting in both browsers sharing the same browsing history that alternates between the webpages in each individual browser's original browsing history. Interleave operations cannot be undone. Implement the `Chrome` class.

```
class Chrome(Browser):
    """
    >>> browser = Chrome()
    >>> _ = browser.visit('cs61a.org')
    >>> _ = browser.visit('tutor.cs61a.org')
    >>> _ = browser.visit('go.cs61a.org')
    >>> browser2 = Chrome()
    >>> _ = browser2.visit('cs61a.org')
    >>> _ = browser2.visit('sections.cs61a.org')
    >>> _ = browser2.visit('code.cs61a.org')
    >>> _ = browser2.visit('oh.cs61a.org')
    >>> print(browser)
    go.cs61a.org<-tutor.cs61a.org<-cs61a.org<-google.com
    >>> print(browser2)
    oh.cs61a.org<-code.cs61a.org<-sections.cs61a.org<-cs61a.org<-google.com
    >>> browser.interleave_histories(browser2)
    >>> print(browser)
    go.cs61a.org<-oh.cs61a.org<-tutor.cs61a.org<-code.cs61a.org<-cs61a.org<-sections.cs61a.org
    <-google.com<-cs61a.org<-google.com
    >>> browser.browsing_history is browser2.browsing_history
    True
    """

    def __init__(self):
        self.browsing_history = __________
                                    (h)

    def interleave_histories(self, other):
        head = self.browsing_history
        other_head = other.browsing_history
        while head is not Link.empty and other_head is not Link.empty:
            head.rest, head, other_head = __________, __________, __________
                                              (i)         (j)         (k)
        other.browsing_history = self.browsing_history
```

i. (1.0 pt) Fill in blank (h).

```
Link('google.com') -OR- Link('google.com', Link.empty)
```

ii. (1.0 pt) Fill in blank (i).

```
other_head
```

iii. (1.0 pt) Fill in blank (j).

```
other_head
```

iv. (1.0 pt) Fill in blank (k).

```
head.rest
```

(c) (10.0 points) MemorySaver

`MemorySaver`'s are `Browser`'s that have a limit to the number of webpages they can store in their browsing history. Once their browsing history exceeds this limit, they begin removing webpages from their browsing history, starting with the earliest visited pages. Implement the `MemorySaver` class.

```
class MemorySaver(__________):
                      (l)
    """
    >>> browser = MemorySaver(2)
    >>> _ = browser.visit('cs61a.org')
    >>> _ = browser.visit('cs61bl.org')
    >>> print(browser)
    cs61bl.org<-cs61a.org
    >>> _ = browser.visit('cs61c.org')
    >>> print(browser)
    cs61c.org<-cs61bl.org
    >>> _ = browser.back()
    >>> print(browser)
    cs61bl.org
    >>> undo = browser.visit('eecs70.org')
    >>> print(browser)
    eecs70.org<-cs61bl.org
    >>> _ = undo()
    >>> print(browser)
    cs61bl.org
    """

    def __init__(self, limit):
        __________
            (m)
        self.limit = limit
        self.history_length = 0

    def visit(self, page):
        if self.history_length == self.limit:
            head = self.browsing_history
            while __________ is not Link.empty:
                      (n)
                head = __________
                           (o)
            __________
                (p)
        else:
            self.history_length = __________
                                      (q)
        return __________
                   (r)
    def back(self):
        __________
            (s)
        return __________
                   (t)
```

i. **(1.0 pt)** Fill in blank (l).

```
Browser
```

ii. **(2.0 pt)** Fill in blank (m).

- ☐ `Browser.__init__()`
- ■ `Browser.__init__(self)`
- ☐ `Browser.__init__(self, limit)`
- ☐ `Chrome.__init__()`
- ☐ `Chrome.__init__(self)`
- ☐ `Chrome.__init__(self, limit)`
- ■ `super().__init__()`
- ☐ `super().__init__(self)`
- ☐ `super().__init__(self, limit)`

iii. **(1.0 pt)** Fill in blank (n).

- ○ `head`
- ○ `head.rest`
- ● `head.rest.rest`
- ○ `self.browsing_history`
- ○ `self.browsing_history.rest`
- ○ `self.browsing_history.rest.rest`

iv. **(1.0 pt)** Fill in blank (o).

- ○ `head`
- ● `head.rest`
- ○ `head.rest.rest`
- ○ `self.browsing_history`
- ○ `self.browsing_history.rest`
- ○ `self.browsing_history.rest.rest`

v. **(1.0 pt)** Fill in blank (p).

```
head.rest = head.rest.rest -OR- head.rest = Link.empty
```

vi. **(1.0 pt)** Fill in blank (q).

```
self.history_length + 1
```

vii. **(1.0 pt)** Fill in blank (r).

```
super().visit(page)
```

viii. **(1.0 pt)** Fill in blank (s).

```
self.history_length -= 1
```

ix. **(1.0 pt)** Fill in blank (t).

```
super().back()
```

6. (16.0 points) Treequality

Help Scheme trees check for treequality!

(a) (2.0 points) all

`all` takes in a list, `s`, and returns `#t` if all the elements of the list are truth-y or if `s` has no elements. Otherwise, it returns `#f`.

```
; doctests
scm> (all (list 0 1 2 3 4 5))
#t
scm> (all (list 0 1 2 3 (< 4 2) 5))
#f
scm> (all '())
#t
```

```
(define (all s)
    (if (null? s)
        #t
        (and (car s) (all (cdr s)))
    )
)
```

i. (2.0 pt) Is `all` tail recursive?

- (●) Yes, it is tail recursive.
- ( ) No, it is not tail recursive.

(b) (5.0 points) zip

Implement `zip` which takes in two lists, `s0` and `s1`, and returns a list of lists where the nested list at index `i` contains exactly two elements: the element at index `i` in `s0` and the element at index `i` in `s1`. If `s0` and `s1` have different lengths, only zip together the first `k` elements where `k` is the length of the shorter list.

```
; doctests
scm> (zip '(1 2 3) '(-1 -2 -3))
((1 -1) (2 -2) (3 -3))
scm> (zip '(1 2 3) '(-1 -2 -3 -4 -5))
((1 -1) (2 -2) (3 -3))


(define (zip s0 s1)
    (if __________
            (c)
        nil
        (cons __________ (zip __________))
                 (d)             (e)
    )
)
```

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

```
(or (null? s0) (null? s1))
```

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

```
(list (car s0) (car s1))
```

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

```
(cdr s0) (cdr s1)
```

iv. (2.0 pt) Is `zip` tail recursive?

- ○ Yes, it is tail recursive.
- ● No, it is not tail recursive.

(c) (9.0 points) treequals?

Recall the `tree` Scheme data abstraction from lecture:

```
(define (tree label branches)
    (cons label branches)
)

(define (label t) (car t))

(define (branches t) (cdr t))

(define (is-leaf t) (null? (branches t)))
```

Implement `treequals?`, a Scheme procedure that takes in two tree abstractions, `t0` and `t1`, and returns `#t` if they have the exact same tree structure and same label values and returns `#f` otherwise.

Reminder, `all` returns `#t` when called on a list with no elements.

```
; doctests
scm> (define t (tree 1 (list (tree 2 nil) (tree 3 nil))))
t
scm> (treequals? t (tree 1 (list (tree 2 nil) (tree 3 nil))))
#t
scm> (treequals? t (tree 1 (list (tree 3 nil) (tree 2 nil))))
#f
scm> (treequals? t (tree 1 (list (tree 3 nil) (tree 3 nil))))
#f
scm> (treequals? t (tree 1 (list (tree 2 nil) (tree 4 nil))))
#f
scm> (treequals? t (tree 1 (list (tree 2 nil) (tree 3 (list (tree 4 nil))))))
#f
scm> (treequals? t (tree 1 (list (tree 2 nil))))
#f


(define (treequals? t0 t1)
    (cond
        ((not __________) #f)
                 (f)
        ((not __________) #f)
                 (g)
        (else (all (map
            (lambda (p) __________)
                             (h)
            __________)
                (i)
        ))
    )
)
```

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

```
(= (label t0) (label t1)) -OR- (= (length (branches t0)) (length
(branches t1)))
```

ii. (2.0 pt) Fill in blank (g). **Hint**: Use `length`.

```
(= (length (branches t0)) (length (branches t1))) -OR- (= (label t0)
(label t1))
```

iii. (3.0 pt) Fill in blank (h).

```
(subtree-matches? (car pair) (car (cdr pair)))
```

iv. (2.0 pt) Fill in blank (i). **Hint**: Use `zip`.

```
(zip (branches t0) (branches t1))
```

7. (8.0 points) Scheme Dictionary Abstraction

Implement a dictionary abstraction in Scheme. In this data abstraction, we represent a dictionary as a list of lists where each nested list has exactly two elements: the first element is the key and the second element is the value. `make-dict` is a zero-argument procedure that returns an empty dictionary abstraction and is implemented for you already. There are two procedures you must implement:

(1) `add-item` takes in a dictionary abstraction, `dict`, a key, `key`, and a `value`, and adds a new entry pointing from `key` to `value` at the end of the list. If `key` already exists as a `key` in `dict`, then the old key-value pair should be removed and the new key-value pair should be added to the end of the list.

(2) `get-item` takes in a dictionary abstraction, `dict`, and a key, `key`, and returns the value associated with `key`. If `key` does not exist in `dict`, `get-item` should error.

Hint: Use `cadr`, which is implemented for you below.

```
; doctests
scm> (define dict (make-dict))
dict
scm> (define dict (add-item dict 'a 'b))
dict
scm> dict
((a b))
scm> (get-item dict 'a)
b
scm> (define dict (add-item dict 'b 'c))
dict
scm> dict
((a b) (b c))
scm> (define dict (add-item dict 'a 'c))
dict
scm> dict
((b c) (a c))
scm> (get-item dict 'a)
c
scm> (get-item dict 'b)
c
scm> (get-item dict 'c)
Error


(define (cadr s) (car (cdr s)))

(define (make-dict) nil)

(define (add-item dict key value)
    (__________ (filter (lambda (p) __________) dict) (list __________))
         (a)                          (b)                    (c)
)

(define (get-item dict key)
    (__________ (__________ (filter (lambda (p) __________) dict)))
         (d)        (e)                            (f)
)
```

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

- ○ `car`
- ○ `cdr`
- ○ `cadr`
- ○ `cons`
- ○ `list`
- ● `append`
- ○ `map`

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

```
(not (eq? (car pair) key))
```

(c) (3.0 pt) Select all of the expressions below that could fill in blank (c). The options that use quotes or quasiquotes are explicitly noted for clarity.

- ☐ `(cons key value)`
- ■ `(cons key (cons value nil))`
- ☐ `(cons (cons key (cons value nil)) nil)`
- ■ `(list key value)`
- ☐ `(list key (list value))`
- ■ `(cons key (list value))`
- ☐ `(list (list key value))`
- ☐ `` `(key value) `` which uses `quasiquote`
- ■ `` `(,key ,value) `` which uses `quasiquote`
- ☐ `` (`(,key ,value)) `` which uses `quasiquote`
- ☐ `` `((,key ,value)) `` which uses `quasiquote`
- ☐ `` (list `(,key ,value)) `` which uses `quasiquote`
- ☐ `'(key value)` which uses `quote`
- ☐ `'((key value))` which uses `quote`

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

- ○ `car`
- ○ `cdr`
- ● `cadr`
- ○ `cons`
- ○ `list`
- ○ `append`
- ○ `map`

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

- (●) `car`
- ( ) `cdr`
- ( ) `cadr`
- ( ) `cons`
- ( ) `list`
- ( ) `append`
- ( ) `map`

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

```
(eq? (car pair) key)
```

8. (10.0 points) Team USA Basketball: The Sweet Dreams Team

The USA Basketball Men's National Team is competing in the 2024 Olympics and needs your help to analyze their players' performances. Complete the SQL queries using the two tables below.

Hint: You may use SQL keywords in the blanks.

The `box_scores` table contains data on how many minutes they spent playing in one game as well as how many points, rebounds, and assists each player got in that game. The `nba_data` table contains data on which National Basketball Association (NBA) team each player plays for as well as which position each player plays.

`box_scores`:

name	minutes	points	rebounds	assists
LeBron James	20.4	14	4	4
Anthony Edwards	20.6	13	2	1
Stephen Curry	21.8	13	2	2
Anthony Davis	17.6	12	10	1
Joel Embiid	16.6	10	7	3
Jrue Holiday	17.8	8	4	3
Bam Adebayo	17.6	8	5	1
Devin Booker	20.0	7	2	1
Jayson Tatum	17.8	6	3	2
Micah Potter	2.5	3	0	0
Tyrese Haliburton	14.8	2	2	3
Derrick White	11.3	1	2	2

`nba_data`:

name	nba_team	position
LeBron James	Lakers	Forward
Anthony Edwards	Timberwolves	Guard
Stephen Curry	Warriors	Guard
Anthony Davis	Lakers	Center
Joel Embiid	Sixers	Center
Jrue Holiday	Celtics	Guard
Bam Adebayo	Heat	Center
Devin Booker	Suns	Guard
Jayson Tatum	Celtics	Forward
Micah Potter	Jazz	Center
Tyrese Haliburton	Pacers	Guard
Derrick White	Celtics	Guard

(a) (2.0 points) Points-Per-Minute

Write a SQL query that returns the names of the top 5 players with the most points-per-minute (PPM) who played for at least 5 minutes in order from highest PPM to lowest PPM. PPM is calculated as `points` divided by `minutes`.

```
-- EXPECTED OUTPUT:
-- LeBron James
-- Anthony Davis
-- Anthony Edwards
-- Joel Embiid
-- Stephen Curry

SELECT name FROM __________;
                   (a)
```

i. (2.0 pt) Fill in blank (a).

```
box_scores WHERE minutes >= 5 ORDER BY points / minutes DESC LIMIT 5
```

(b) (4.0 points) More Rebounds = More Points?

Write a SQL query that returns the names of NBA teams and **average number of points** scored by players that play for that NBA team in order from most average rebounds to least average rebounds for the NBA teams that have a total of 5 or more rebounds across all their players. Only players who played for at least 5 minutes should be considered in these calculations.

```
-- EXPECTED OUTPUT:
-- Sixers|10.0
-- Lakers|13.0
-- Heat|8.0
-- Celtics|5.0

SELECT __________ FROM __________ WHERE __________ GROUP BY __________ ORDER BY __________;
          (b)             (c)              (d)              (e)                (f)
```

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

```
nba_team, AVG(points)
```

ii. (0.5 pt) Fill in blank (c).

- ○ `box_scores`
- ○ `nba_data`
- ○ `box_scores AS b1, box_scores AS b2`
- ○ `nba_data AS n1, nba_data AS n2`
- ● `box_scores AS b, nba_data AS n`

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

```
minutes >= 5 AND b.name = n.name
```

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

```
nba_team HAVING SUM(rebounds) >= 5
```

v. (1.0 pt) Fill in blank (f).

```
AVG(rebounds) DESC
```

(c) (4.0 points) Assists-Per-Minute

Write a SQL query that returns each position, the name of the one player with the most assists-per-minute (APM) in that position, and that player's APM value in order from highest APM to lowest APM. APM is calculated as `assists` divided by `minutes`. Only players who played for at least 5 minutes should be considered in these calculations.

```
-- EXPECTED OUTPUT:
-- Guard|Tyrese Haliburton|0.2027027027027027
-- Forward|LeBron James|0.19607843137254904
-- Center|Joel Embiid|0.18072289156626503

SELECT __________ FROM __________ WHERE __________ GROUP BY __________ ORDER BY __________;
         (g)             (h)             (i)              (j)              (k)
```

i. (1.0 pt) Fill in blank (g).

```
position, b.name, MAX(assists / minutes) -OR- position, n.name,
MAX(assists / minutes)
```

ii. (0.5 pt) Fill in blank (h).

- ○ `box_scores`
- ○ `nba_data`
- ○ `box_scores AS b1, box_scores AS b2`
- ○ `nba_data AS n1, nba_data AS n2`
- ● `box_scores AS b, nba_data AS n`

iii. (1.0 pt) Fill in blank (i).

```
WHERE minutes >= 5 AND b.name = n.name
```

iv. (0.5 pt) Fill in blank (j).

```
position
```

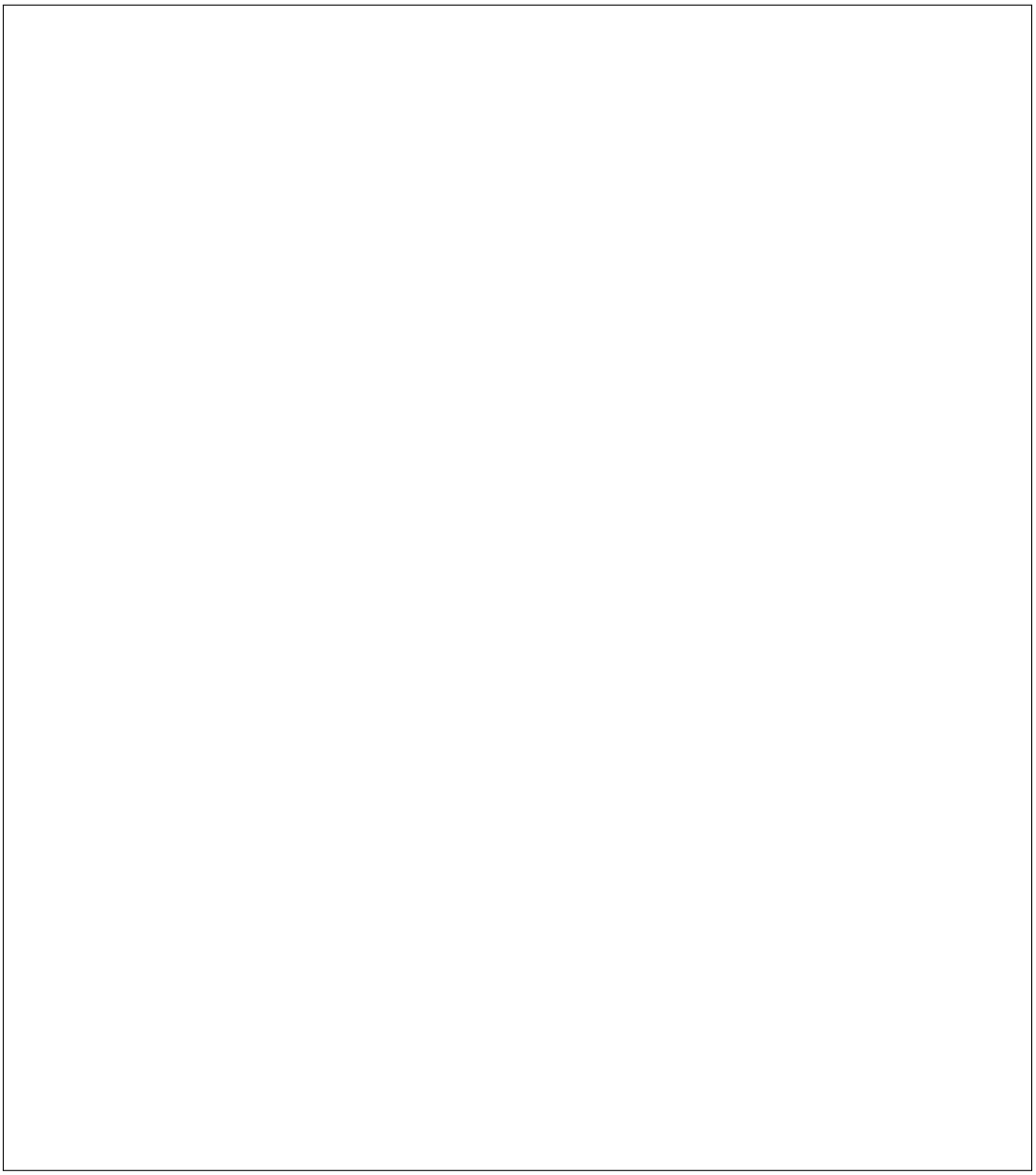
v. (1.0 pt) Fill in blank (k).

```
MAX(assists / minutes) DESC -OR- assists / minutes DESC
```

9. (0.0 points) Just for Fun

This is not for points and will not be graded.

(a) Optional: Draw something that encapsulates your summer!



No more questions.