Exam: CS61A Summer 2020 Final

Name: Solution Key

Email: example_key

signature-page

January 01 2000 at 12:00:00am PDT (7531 days and 10 minutes before the deadline)

https://okpy.org/admin/grading/0

WITHIN SKELETON

The code passed the test cases and is within the skeleton

```
'[ommited docstring]'
'[ommited docstring]'
honor_code = '\nHONOR CODE GOES HERE\n'
addendum = '\nADDENDUM GOES HERE\n'
your_name = '\nYOUR NAME GOES HERE\n'
todays_date = '\nTODAYS DATE GOES HERE\n'
'[ommited docstring]'
mc_result_compvsinterp = 'False, you can turn an interpreter into a compiler via¶
 a Futamura projection'
'[ommited docstring]'
mc_result_pyfeatures = 'Lazy Evaluation'
'[ommited docstring]'
mc_result_regression = 'Orthogonal'
'[ommited docstring]'
mc_result_machine_learning = 'Determine f'
```

comments

January 01 2000 at 12:00:00am PDT (7531 days and 35 minutes before the deadline)

https://okpy.org/admin/grading/0

WITHIN SKELETON

The code passed the test cases and is within the skeleton

```
(define
  (cabinet-filter lst pred)
  (cond
    ((or (not (list? lst)) (null? lst)) lst)
    ((pred (car lst)) (cabinet-filter (cdr lst) pred))
    (else
      (cons (cabinet-filter (car lst) pred) (cabinet-filter (cdr lst) pred))
    )
  )
)(define-macro
  (remove-comments code)
  (define
    (helper x)
    (and (list? x) (equal? (car x) (quote comment-starts-here)))
  )
  (cabinet-filter code helper)
)
```

scheme-curry

January 01 2000 at 12:00:00am PDT (7531 days and 1 hour before the deadline)

https://okpy.org/admin/grading/0

WITHIN SKELETON

The code passed the test cases and is within the skeleton

```
(define (// numer denom) (floor (/ numer denom)))(define
  (ladder-locator newspaper)
  (define
    (helper newspaper output)
    (define current-digit (modulo newspaper 10))
    (define value (if (= current-digit 8) (list current-digit) current-digit))
    (if (zero? newspaper) output (helper (// newspaper 10) (cons value output)))¶

  )
  (helper newspaper nil)
)(define
  (take k s)
  (if (zero? k) nil (cons (car s) (take (- k 1) (cdr-stream s))))
)(define
  (studio-switch xv yq)
  (if
    (equal? (car xv) (quote guitar))
    (studio-switch yq (cdr-stream xv))
    (cons-stream (car xv) (studio-switch (cdr-stream xv) yq))
  )
)
```

same-length

January 01 2000 at 12:00:00am PDT (7531 days, 1 hour, and 25 minutes before the deadline)

https://okpy.org/admin/grading/0

WITHIN SKELETON

The code passed the test cases and is within the skeleton

```
'[ommited docstring]'
'[ommited docstring]'

def same_length(t, desired):
    '[ommited docstring]'
    if (desired == 0):
        t.branches = []
    elif t.is_leaf():
        t.branches = [Tree('hello')]
    for b in t.branches:
        same_length(b, (desired - 1))
'[ommited docstring]'

def shortest_no_hello(t):
    '[ommited docstring]'
    if all([(b.label == 'hello') for b in t.branches]):
        return [t.label]
    return ([t.label] + min([shortest_no_hello(b) for b in t.branches], key=len)¶
)

class Tree():
    '[ommited docstring]'

    def __init__(self, label, branches=[]):
        for b in branches:
            assert isinstance(b, Tree)
        self.label = label
        self.branches = list(branches)

    def is_leaf(self):
        return (not self.branches)

    def __str__(self):

        def print_tree(t, indent=0):
            tree_str = ((' ' * (4 * indent)) + str(t.label)) + '\n')
            for b in t.branches:
                tree_str += print_tree(b, (indent + 1))
            return tree_str
        return print_tree(self).rstrip()
```

symdiff

January 01 2000 at 12:00:00am PDT (7531 days, 1 hour, and 50 minutes before the deadline)

https://okpy.org/admin/grading/0

WITHIN SKELETON

The code passed the test cases and is within the skeleton

```
def symdiff(salmons):
    '[ommited docstring]'
    salmons = [salmon for salmon in salmons if (salmon is not Link.empty)]
    if (not salmons):
        return Link.empty
    first = min([salmon.first for salmon in salmons])
    new_salmons = []
    count = 0
    for salmon in salmons:
        if (salmon.first == first):
            (salmon, count) = (salmon.rest, (count + 1))
        new_salmons.append(salmon)
    if ((count % 2) == 0):
        return symdiff(new_salmons)
    else:
        return Link(first, symdiff(new_salmons))

class Link():
    '[ommited docstring]'
    empty = ()

    def __init__(self, first, rest=empty):
        self.first = first
        self.rest = rest

    def __getitem__(self, i):
        if (i == 0):
            return self.first
        else:
            return self.rest[(i - 1)]

    def __len__(self):
        return (1 + len(self.rest))

    def __repr__(self):
        if self.rest:
            rest_str = (', ' + repr(self.rest))
        else:
            rest_str = ''
        return 'Link({0}{1})'.format(repr(self.first), rest_str)
```

grocery-list

January 01 2000 at 12:00:00am PDT (7531 days, 2 hours, and 20 minutes before the deadline)

https://okpy.org/admin/grading/0

WITHIN SKELETON

The code passed the test cases and is within the skeleton

```
'[ommited docstr¶
ing]'

class Item():

    def __init__(self, name, price, type):
        '[ommited docstring]'
        self.name = name
        self.price = price
        self.type = type

    def __repr__(self):
        '[ommited docstring]'
        return self.name

class GroceryList():

    def __init__(self, items):
        '[ommited docstring]'
        self.items = items

    def buy_item(self, name):
        '[ommited docstring]'
        i = 0
        while (i < len(self.items)):
            item = self.items[i]
            if (item.name == name):
                self.items.pop(i)
            i += 1

    def buy_cheapest(self):
        '[ommited docstring]'
        cheapest = min(self.items, key=(lambda x: x.price))
        self.buy_item(cheapest.name)

    def generate_list(self, dollars_available, types):
        '[ommited docstring]'
        spent = 0
        i = 0
        while ((spent < dollars_available) and (i < len(self.items))):
```

```
            item = self.items[i]
            if ((item.type in types) and ((spent + item.price) <= dollars_availa¶
ble)):
                spent += item.price
                (yield item)
            i += 1
```

compress

January 01 2000 at 12:00:00am PDT (7531 days, 2 hours, and 45 minutes before the deadline)

https://okpy.org/admin/grading/0

WITHIN SKELETON

The code passed the test cases and is within the skeleton

```
def compress(lst1, lst2):
    '[ommited docstring]'
    if (lst1 == lst2):
        return True
    elif ((len(lst1) <= 1) or (len(lst2) == 0)):
        return False
    compress_add = compress(([(lst1[0] + lst1[1])] + lst1[2:]), lst2)
    compress_sub = compress(([(lst1[0] - lst1[1])] + lst1[2:]), lst2)
    compress_eq = (compress(lst1[1:], lst2[1:]) and (lst1[0] == lst2[0]))
    return (compress_eq or compress_add or compress_sub)
```

blockbuster

January 01 2000 at 12:00:00am PDT (7531 days, 3 hours, and 10 minutes before the deadline)

https://okpy.org/admin/grading/0

WITHIN SKELETON

The code did not pass the test cases but is within the skeleton

```
-- You are a DVD rental ¶
shop owner.

-- To run your shop smoothly, you created some tables
-- related to your shop.
--      `DVD` table to keep track of basic information about the DVDs
--      `inventory` table to keep track of the inventory in the shop.
--          NOTE: the total can be greater than the sum of available and rented,¶

--          meaning that there can be some missing copies.
--      `july_rental` table to keep track of rental history of the month of july¶

--
-- This question is in 2 parts.
--
-- NOTE: The tests for this question are NOT comprehensive, as they only
-- refer to the data tables as shown below. We will be grading this question
-- manually.
--
-- NOTE: In all scheme/sql questions you can put a multi-line answer
-- in a blank


CREATE TABLE DVD AS
    SELECT "DVD a" AS name, "*****" AS ratings, 2017 AS year UNION
    SELECT "DVD b" , "*" , 1988 UNION
    SELECT "DVD c" , "***" , 2010 UNION
    SELECT "DVD d" , "***" , 2018 UNION
    SELECT "DVD e" , "***" , 2020 UNION
    SELECT "DVD f" , "*****" , 2019;

CREATE TABLE inventory AS
    SELECT "DVD a" AS name, 1 AS available, 2 AS rented, 3 AS total UNION
    SELECT "DVD b" , 2 , 3 , 10 UNION
    SELECT "DVD c" , 4 , 1 , 9 UNION
    SELECT "DVD d" , 7 , 2 , 11 UNION
    SELECT "DVD f" , 0 , 12, 20;

CREATE TABLE july_rental AS
    SELECT "DVD c" AS name, 1 AS rented_date, 5 AS return_date UNION
```

```
    SELECT "DVD f" , 4 , 10 UNION
    SELECT "DVD a" , 5 , 8 UNION
    SELECT "DVD b" , 5 , 9 UNION
    SELECT "DVD d" , 9 , 13 UNION
    SELECT "DVD f" , 10 , 15 UNION
    SELECT "DVD a" , 18, 26;

-- Part a : Complete the SQL statement below to select a one-column table
-- of DVD names whose release year is 2010 or later and have at least 1 copy
-- missing. Order your results by the number of missing copies, biggest to
-- smallest.
--
-- To run tests just for this part run
--      python3 ok -q a

CREATE TABLE parta AS
    SELECT DVD.name FROM DVD, inventory
        WHERE
            DVD.name = inventory.name AND DVD.year >= 2010
            AND inventory.total - (inventory.available + inventory.rented) > 0
        ORDER
            BY inventory.total - (inventory.available + inventory.rented)
            DESC;

-- Part b : We are interested in seeing how renting behavior correlates with
--      rating.
--
-- Complete the SQL statement below to create a two-column table that shows
-- how many DVDs per star rating have been rented out for a period of more
-- than 3 days.
--
-- For example, if a DVD was rented out on july 3 and returned on july 6,
-- that would not count as being rented out for more than 3 days, but if it
-- was returned on july 7 that would count as more than 3 days.
--
-- Each row should contain the star rating and number of DVDs. Only include rati¶
ng
-- categories that have at least 2 DVDs satisfying the condition.
--
-- To run tests just for this part run
--      python3 ok -q b

CREATE TABLE partb AS
    SELECT DVD.ratings, COUNT(*) FROM DVD, july_rental
        WHERE DVD.name = july_rental.name AND return_date - rented_date > 3
        GROUP BY DVD.ratings
        HAVING COUNT(*) >= 2;
```