CS 61A Structure and Interpretation of Computer Programs
Summer 2022 Final

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?

1. (8.0 points) What Would Python Display?

For each part of this question, you will be shown code that is executed in the Python interpreter, and you should write what is printed to the interpreter after executing the last line.

- If a `StopIteration` exception occurs as a result of executing a line, you should write `StopIteration`
- If an iterator object is displayed as the result of executing a line, you should write `Iterator`, unless that object is a generator. If a generator object is displayed as the result of executing a line, you should write `Generator`
- If an error arises as a result of executing a line that is *not* a `StopIteration`, you should write `Error`
- If nothing is displayed as the result of executing a line, you should write `Nothing`

You should assume that each part is executed in order—that all the code from the previous parts have been executed before the next part.

(a) (1.0 pt)

```
>>> a = [1, 2, 3, 5, 8, 13]
>>> i = iter(a)
>>> j = iter(a)
>>> next(i) + next(j)
```

(b) (2.0 pt)

```
>>> next(i) + next(i)
```

(c) (2.0 pt)

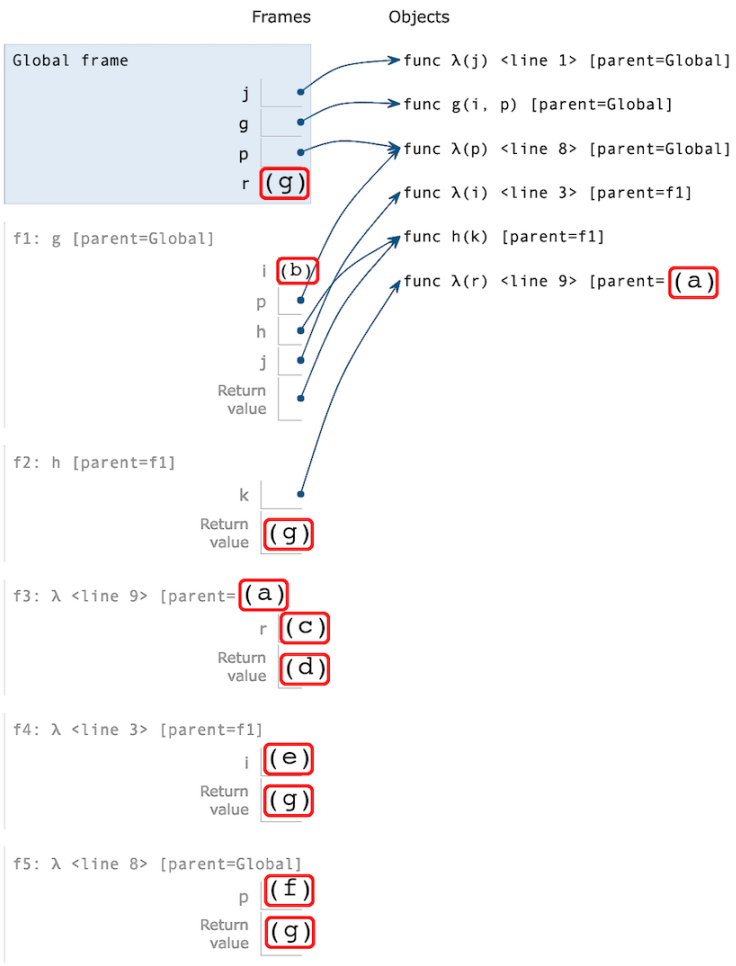
```
>>> def mystery(n):
...     i = 1
...     while i < n:
...         if n % i == 0:
...             yield i
...         i += 1
...
>>> m = mystery(8)
>>> m
```

(d) (3.0 pt) `>>> for x in range(10): ...     print(next(m)) ...`

2. (10.0 points) Lambdistinction

Fill in the blanks of the environment diagram corresponding to executing the following code block.

```
1| j = lambda j: not j
2| def g(i, p):
3|     j = lambda i: p(i - 1)
4|     def h(k):
5|         return j(k(i + 2) - 1)
6|     return h
7|
8| p = lambda p: not p
9| r = g(-4, p)(lambda r: r * -1)
```



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

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

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

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

(e) **(2.0 pt)** Fill in blank (e)

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

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

3. (15.0 points) Pruning and Sprouting

(a) (8.0 points) Prune Below

Fill in the definition of the function `prune_below`, which takes in one argument `t`, that is a `Tree`. `prune_below` also takes in a second optional parameter, `seen_before`, which is a list containing all of the values the function has seen in `t`'s parent.

`prune_below` mutates `t` by removing all subtrees whose labels are equal to `t.label`, and then mutates all remaining subtrees of `t` by the same process.

```
def prune_below(t, seen_before=[]):
    """
    >>> t = Tree(5, [Tree(4, [Tree(3), Tree(4), Tree(5)]), Tree(5, [Tree(6), Tree(7)])])
    >>> print(t)
    5
      4
        3
        4
        5
      5
        6
        7
    >>> prune_below(t)
    >>> print(t)
    5
      4
        3
    """
    seen_before = __________
                    (a)
    new_branches = [b for b in t.branches if __________]
                                                 (b)

    __________
        (c)
    for b in t.branches:
        __________
            (d)
```

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

- ◯ `seen_before.append(t.label)`
- ◯ `seen_before[1:]`
- ◯ `seen_before + [t.label]`
- ◯ `[t.label]`
- ◯ `Tree(t.label)`
- ◯ `list(seen_before)`
- ◯ `seen_before + t.label`
- ◯ `seen_before + [b.label for b in t.branches]`
- ◯ `seen_before.extend([b.label for b in t.branches])`
- ◯ `seen_before += [t.label]`

ii. (3.0 pt) Fill in blank (b)

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

- ◯ `t.branches.extend(new_branches)`
- ◯ `t.branches = [b.label for b in t.branches]`
- ◯ `return Tree(t.label, new_branches)`
- ◯ `t.branches.remove(new_branches)`
- ◯ `prune_below(t, new_branches)`
- ◯ `t = Tree(t.label, new_branches)`
- ◯ `new_branches.remove(t.label)`
- ◯ `t.branches = new_branches`

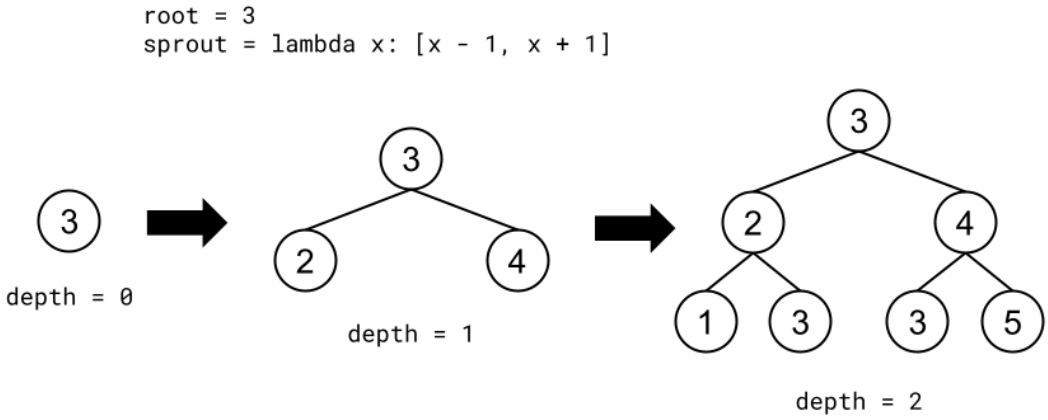
iv. (1.0 pt) Fill in blank (d)

(b) (7.0 points) Sprout Tree

In this problem, we'll write a function that "sprouts" new trees from a given `root`, which can be any integer. We'll also use a function `sprouter`, that takes in a single integer as an argument and returns a list. To sprout a new tree, we need to construct an instance of the `Tree` class whose `label` is `root`. We then call `sprouter` on `root` to get a list of numbers that will serve as the labels of our tree's branches.

Depending on how deep we want our tree to be, we can repeat the process on the `labels` of each of our new `branches`, using the same `sprouter` function.

For example, the below diagram illustrates how to sprout a tree with a `root` of `3` and a `sprouter` function of `lambda x: [x - 1, x + 1]`.



Fill in the definition of `sprout_tree` below, which takes in three parameters: `root`, `sprouter`, and `d`, and returns a new tree of depth `d` that is sprouted from `root` by applying the `sprouter` function. You can assume that `d` will always be `>= 0`.

```
def sprout_tree(root, sprouter, d):
    """
    >>> minus_plus_one = lambda x: [x - 1, x + 1]
    >>> print(sprout_tree(3, minus_plus_one, 0))
    3
    >>> print(sprout_tree(3, minus_plus_one, 1))
    3
      2
      4
    >>> print(sprout_tree(3, minus_plus_one, 2))
    3
      2
        1
        3
      4
        3
        5
    """
    if __________:
           (a)
        return __________
                   (b)
    new_branches = [__________ for i in __________]
                        (c)                 (d)
    return __________
              (e)
```

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

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

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

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

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

4. (15.0 points) Linked List Comprehension

In Python, we can use list comprehensions to quickly generate lists from other lists—list comprehensions include an expression that is applied to each element in the list, and can optionally include an `if` clause.

```
>>> a = [1, 2, 3, 4, 5]
>>> [i ** 2 for i in a if i % 2 == 0]
[4, 16]
```

We often refer to the expression as a "mapping expression", and the `if` clause as a "filter clause". `i ** 2` is the mapping expression in the above example, and `i % 2 == 0` is the filter clause.

In this problem, we will write a function that works similarly for linked lists:

```
>>> b = Link(1, Link(2, Link(3, Link(4, Link(5)))))
>>> link_comp(b, lambda x: x ** 2, lambda x: x % 2 == 0)
Link(4, Link(16))
```

You will implement this function both recursively and iteratively.

(a) (7.0 points) Recursive Version

Fill in the definition of the below function `link_comp_recur` to execute a "list comprehension" over a linked list `lnk`, using the functions `map_func` and `filter_func`.

`map_func` is a function that takes in one argument and returns one value, and `filter_func` is a one-argument function that will always return a boolean value.

`link_comp_recur` should return a *new* linked list that is identical to `lnk`, but only keeping each `Link` for whom calling `filter_func` on the `first` of that `Link` returns `True`. Additionally, the `first` of each `Link` in your new list should be equal to the `first` of the corresponding `Link` in `lnk` with `map_func` applied.

Note: `filter_func` is always applied *before* `map_func`—we check whether a link should be filtered before applying our mapping function to it.

```
def link_comp_recur(lnk, map_func, filter_func):
    """
    >>> lnk = Link(1, Link(2, Link(3, Link(4, Link(5)))))
    >>> print(lnk)
    <1 2 3 4 5>
    >>> add_one = lambda x: x + 1
    >>> is_even = lambda x: x % 2 == 0
    print(link_comp_recur(lnk, add_one, is_even))
    <3 5>
    >>> square = lambda x: x ** 2
    >>> greater_than_2 = lambda x: x > 2
    print(link_comp_recur(lnk, square, greater_than_2))
    <9 16 25>
    """
    if __________:
          (a)
        return Link.empty
    new_rest = __________
                  (b)
    if __________:
          (c)
        return __________
                  (d)
    else:
        return new_rest
```

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

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

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

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

(b) (7.0 points) Iterative Version

Next, fill in the behavior of `link_comp_iter`. `link_comp_iter` should behave exactly the same as `link_comp_recur`, but internally it is implemented using iteration rather than recursion. For simplicity, you can assume that `lnk` will never be equal to `Link.empty`, and that we will never filter out *every* value of the list.

```
def link_comp_iter(lnk, map_func, filter_func):
    """
    >>> lnk = Link(1, Link(2, Link(3, Link(4, Link(5)))))
    >>> print(lnk)
    <1 2 3 4 5>
    >>> add_one = lambda x: x + 1
    >>> is_even = lambda x: x % 2 == 0
    print(link_comp_iter(lnk, add_one, is_even))
    <3 5>
    >>> square = lambda x: x ** 2
    >>> greater_than_2 = lambda x: x > 2
    print(link_comp_iter(lnk, square, greater_than_2))
    <9 16 25>
    """
    while __________:
            (a)
        lnk = lnk.rest
    front = Link(map_func(lnk.first))
    end = front
    lnk = lnk.rest
    while lnk is not Link.empty:
        if __________:
              (b)
            end.rest = __________
                           (c)
            __________
                (d)
        __________
           (e)
    return front
```

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

ii. (2.0 pt) Fill in blank (b)

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

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

- ◯ `lnk = lnk.rest`
- ◯ `lnk.first = map_func(lnk.first)`
- ◯ `end = end.rest`
- ◯ `end, lnk = end.rest, lnk.rest`
- ◯ `end.first = map_func(end.first)`
- ◯ `end, lnk = lnk, end`

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

(c) (1.0 points) Efficiency

i. (1.0 pt) Which of the following describes the efficiency of `link_comp_recur` and `link_comp_iter`, relative to the length of the linked list `lnk`? Efficiency is the same for both functions.

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

5. (15.0 points) Tails is my Pal!

(a) (3.0 points) Reverse car

Implement `rev-car`, a procedure that takes in a Scheme list and returns the last value in the list. You may assume the input list is not empty.

```
scm> (rev-car '(1 2 3))
3
scm> (rev-car '(1))
1

(define (rev-car lst)
  (if (__________)
          (a)
     ----------
        (b)
     ----------
        (c)
  )
)
```

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

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

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

(b) (3.0 points) Reverse cdr

Implement `rev-cdr`, a procedure that takes in a Scheme list and returns a new list with all but the last element. You may assume the input list is not empty.

```
scm> (rev-cdr '(1 2 3))
(1 2)
scm> (rev-cdr '(1))
()

(define (rev-cdr lst)
  (if __________
         (a)
    __________
       (b)
    __________
       (c)
  )
)
```

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

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

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

(c) (5.0 points) Reverse cdr Tail

Now, implement `rev-cdr` tail recursively. `reverse` is a one-argument procedure that takes in a Scheme list and reverses it. You may assume `reverse` is implemented tail recursively and correctly for you. However, your implementation may **not** call `reverse` besides the one place where it is already written for you.

```
scm> (reverse '(1 2 3))
(3 2 1)
scm> (rev-cdr '(1 2 3))
(1 2)
scm> (rev-cdr '(1))
()
```

```
(define (rev-cdr lst)
  (define (helper lst sofar)
    (if __________
           (a)
      __________
         (b)
      (helper __________ __________)
                 (c)        (d)
    )
  )
  (reverse __________)
              (e)
)
```

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

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

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

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

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

(d) (4.0 points) Is Palindrome?

Implement `is-pal`, a procedure that takes in a Scheme list of numbers and returns `#t` if the list is a palindrome. That is, the input list reads the same forward and backward. Otherwise, it should return `#f`. Your implementation should work for lists of odd and even length. Your implementation must use `rev-car` and `rev-cdr`.

```
scm> (is-pal '(1 2 3 3 2 1))
#t
scm> (is-pal '(1 2 3 2 1))
#t
scm> (is-pal '(1 2 3))
#f
scm> (is-pal '(1))
#t
scm> (is-pal '())
#t

(define (is-pal lst)
  (cond
    (__________ #t)
        (a)
    (__________ #t)
        (b)
    ((= __________) __________)
           (c)          (d)
    (else #f)
  )
)
```

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

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

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

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

6. (11.0 points) Sche(min) some Data Abstract(ions)

(a) (3.0 points) Minion Dominion

Let's make a Scheme data abstraction to represent everybody's favorite fictional yellow creatures! A minion is described by its name, energy level, and the names of the missions it has completed.

- The name of a minion can be any symbol.
- The energy level of a minion is an integer.
- A minion's completed missions are represented as a list of **mission names**.

A mission is also represented by a data abstraction. Each mission has a mission name and energy cost. The mission data abstraction implementation is omitted, but you may use the `get-mission-name` and `get-mission-energy` procedures.

- The name of a mission can be any symbol.
- The energy cost of a mission is an integer.

Here is how a minion data abstraction can be constructed and used:

```
scm> (define stuart (make-minion 'Stuart 1 nil))
stuart
scm> (get-name stuart)
Stuart
scm> (get-energy stuart)
1
scm> (get-missions stuart)
()
```

`make-minion` is implemented for you. Complete the `get-name`, `get-energy`, and `get-missions` procedures.

```
(define (make-minion name energy missions)
  (list name energy missions)
)

(define (get-name minion)
  ----------
      (a)
)

(define (get-energy minion)
  ----------
      (b)
)

(define (get-missions minion)
  ----------
      (c)
)
```

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

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

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

(b) (6.0 points) You mission or mishout?

Minions are good for one thing: completing missions. Implement `complete-mission`, which takes in a **minion** abstraction and **mission** abstraction and returns a **new minion** abstraction. If the minion's **energy level** is at least as much as the mission's **energy cost**, that mission's **name** should be added to the **end** of the minion's list of completed missions, and the minion's **energy level** should be reduced by the **energy cost** of the completed mission. If the minion does not have enough energy to complete the mission, nothing about the minion should change.

In both cases, you should be returning a minion abstraction.

Do not break the abstraction barrier.

```
scm> (define kevin (make-minion 'Kevin 9 nil))
kevin
scm> (define mission-1 (make-mission 'Mission1 7))
mission1
scm> (define mission-2 (make-mission 'Mission2 5))
mission2
scm> (define mission-3 (make-mission 'Mission3 1))
mission3
scm> (define kevin (complete-mission kevin mission-1))
kevin
scm> (define kevin (complete-mission kevin mission-2))
kevin
scm> (define kevin (complete-mission kevin mission-3))
kevin
scm> (get-energy kevin)
1
scm> (get-missions kevin)
(Mission1 Mission3)
```

The procedure signatures for the mission abstraction are shown below:

```
(define (make-mission name energy-cost)
  ; IMPLEMENTATION OMITTED
)

(define (get-energy-cost mission)
  ; IMPLEMENTATION OMITTED
)

(define (get-mission-name mission)
  ; IMPLEMENTATION OMITTED
)

(define (complete-mission minion mission)
  (if (>= __________)
             (a)
    __________
        (b)
    __________
        (c)
  )
)
```

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

ii. **(4.0 pt)** Fill in blank (b)

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

(c) (2.0 points) M&Ms need R&R

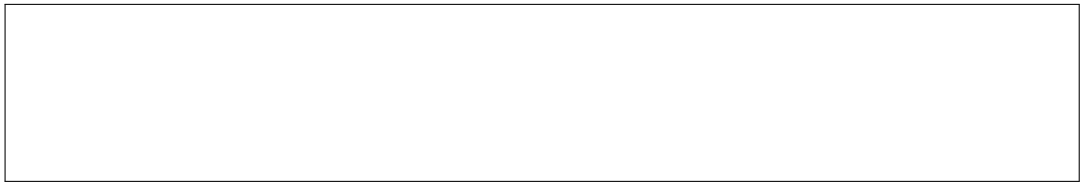
Minions need to take a break and replenish their energy after a long day of completing missions. Implement the `take-break` procedure which takes in a **list of minion abstractions** and an integer `n` and returns new list of **new** minion abstractions where each minion has `n` more energy. The names and completed mission lists of each minion should be unaffected.

Do not break the abstraction barrier.

```
scm> (define bob (make-minion 'Bob 2 nil))
bob
scm> (define mission-A (make-mission 'MissionA 1))
mission-A
scm> (define bob (complete-mission bob mission-A))
bob
scm> (define otto (make-minion 'Otto 0 nil))
otto
scm> (define mission-B (make-mission 'MissionB 2))
mission-B
scm> (define otto (complete-mission otto mission-B))
otto
scm> (define minions (list bob otto))
minions
scm> (define new-minions (take-break minions 12))
new-minions
scm> (define bob (car new-minions))
bob
scm> (define otto (car (cdr (new-minions))))
otto
scm> (get-energy bob)
13
scm> (get-energy otto)
12
scm> (get-missions bob)
(MissionA)
scm> (get-missions otto)
()

(define (take-break minions n)
  (map
    ----------
        (a)
    minions
  )
)
```

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



7. (6.0 points) Regular Expressongs

(a) Richard Roggenrigo

Richard is obsessed with Olivia Rodrigo's song "good 4 u" and wants to be able to match all the valid spellings of the title. He wants to write a function in Python that will return whether the title is contained within the lyrics passed in. The only variations he has seen include:

- "good 4 u"
- "Good 4 u"
- "good for you"
- "Good for you"

Here is the function Richard wants to use:

```
def good_4_u(lyrics):
    """
    >>> good_4_u("good 4 u")
    True
    >>> good_4_u("Good 4 u")
    True
    >>> good_4_u("good for you")
    True
    >>> good_4_u("Good for you")
    True
    >>> good_4_u("We think that's good for you!") # can contain more than the song title
    True
    >>> good_4_u("Good for your") # can't be similar words or have extra characters
    False
    >>> good_4_u("cGood for you")
    False
    >>> good_4_u("food for you")
    False
    >>> good_4_u("godd for you")
    False
    >>> good_4_u("Good four you")
    False
    >>> good_4_u("good 4 you") # can't combine 4 and you
    False
    >>> good_4_u("good for u") # can't combine for and u
    False
    """
    return bool(re.search(r'_____________________________', lyrics))
```

(a)

i. (3.0 pt) Fill in blank a such that all the doctests behave as expected. Check all that apply.

- ☐ `\b(good|Good) (4 u|for you)\b`
- ☐ `^(good|Good) (4 u|for you)$`
- ☐ `\b(g|G)ood (4 u|for you)\b`
- ☐ `(\bg|G)ood( 4 u| for you\b)`
- ☐ `\b(g|G)ood( 4 u| for you)\b`
- ☐ `[^g|G]ood( 4 u| for you)\b`
- ☐ `\b[Ggodfr4yu\s]{8, 12}\b`
- ☐ `\b[Ggod]{4} [4for]{3} [you]{3}\b`
- ☐ `.*[Ggod]{4} [4for]{1,3} [you]{1,3}.*`
- ☐ `\b[Ggod]{4} (4 u|for you)\b`

(b) Cooper Belongs with Me

Cooper, on the other hand, is obsessed with Taylor Swift's song "You Belong With Me" and wants to be able to match all the valid spellings of this song's title. They have written a function in Python that will return whether the lyrics passed in are exactly a valid title.

```
def you_belong_with_me(lyrics):
    return bool(re.search(r'^([Yy]ou) [Bb]elong\s((W|w)ith|w\/) (M|m)e(e{3})*$', lyrics))
```

i. (3.0 pt) Select all possible inputs for `lyrics` where `you_belong_with_me` would return True.

- ☐ `"You Belong With Me"`
- ☐ `"You belong With me"`
- ☐ `"You Belong With Mee"`
- ☐ `"You Belong With Meee"`
- ☐ `"You Belong With Meeeeeee"`
- ☐ `"you belong with me"`
- ☐ `"you belong with me!"`
- ☐ `"Oh You belong with me"`
- ☐ `"You Belong w/ Me"`
- ☐ `"You belong with me (Taylor's Version)"`

8. (13.0 points) SQL: The Sequel

(a) (3.0 pt) Write a query that finds all the Disney or Illumination movies. The output table should have 1 column with the movie names.

You must reference the `studios` tables (pictured below) in your query.

name	studio
Shrek 2	Dreamworks
Kung Fu Panda 2	Dreamworks
Top Gun: Maverick	Skydance
Star Wars: Attack of the Clones	Disney
Home Alone 2	20th Century
Percy Jackson: Sea of Monster	20th Century
Harry Potter: Chamber of Secrets	Universal
Minions: The Rise of Gru	Illumination
Frozen 2	Disney

Your solution should work on any table with the same columns.

(b) (4.0 pt) The below table is named `movies`. All values are in millions.

title	budget	boxoffice
Shrek 2	150	929
Kung Fu Panda 2	150	665
Top Gun: Maverick	170	1287
Star Wars: Attack of the Clones	115	654
Home Alone 2	28	359
Percy Jackson: Sea of Monster	90	201
Harry Potter: Chamber of Secrets	100	880
Minions: The Rise of Gru	80	654
Frozen 2	150	1450

Write a query that finds the movie with the highest earnings ratio, the ratio between how much the movie makes in the box office compared to the budget. For example, a movie that has a budget of 50 and has a box office of 400 should have a ratio of 8.

The output table should have 1 column for the movies.

Your output should be a table with one row with the value "Home Alone 2". You must reference the `movies` table in your query. `SELECT "Home Alone 2";` is not a valid solution. Your solution should work on any table with `title`, `budget`, and `boxoffice` as the columns.

You are not allowed to use aggregate functions on this problem.

(c) (6.0 points)

Using the `movies` and `studios` tables from the last 2 problems, write a query that find the top 2 studios with regards to average box office.

The table should be sorted from highest average to lowest. You should only include studios that have 2 or more movies made. The output table should have 2 columns, one for the studio and one for the average box office.

Your solution should work on any tables with the same columns.

```
SELECT (a) FROM movies, studios WHERE (b)
GROUP BY (c) HAVING (d) ORDER BY (e) DESC (f);
```

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

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

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

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

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

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

9. (3.0 points) Special Topics

(a) (1.0 pt) Theory Query

What does NP stand for?

◯ Non-computable Problem

◯ Non-deterministic Polynomial

◯ Not Polynomial

◯ Non-decidable Problem

(b) (1.0 pt) Hear-y Queery

What is the process of numerically measuring the properties of speech sounds called?

◯ Segmentation

◯ Phonological Areality

◯ Sociolinguistics

◯ Acoustic Analysis

(c) (1.0 pt) Avengers, Ensemble!

A _____ model aggregates the results from a lot of randomized _____ models.

◯ Pandas, Linear Regression

◯ Random Lines, Linear Regression

◯ Random Forest, Decision Trees

◯ Random Forest, Machine Learning

No more questions.