CS 61A Structure and Interpretation of Computer Programs
Summer 2023

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?

```
Oski Bear
```

(b) What is your student ID number?

```
123456789
```

(c) What is your @berkeley.edu email address?

```
oski@berkeley.edu
```

(d) Sign (or type) your name to confirm that all work on this exam will be your own. The penalty for academic misconduct on an exam is an F in the course.

```
Oski the Bear
```

1. (11.0 points) What Would You Display

(a) (5.0 points) WWPD

For each expression below, choose the correct option or write the output displayed by the interactive Python interpreter when the expression is evaluated. The output may have multiple lines.

- If an error occurs, write "Error", but include all output displayed before the error.
- If evaluation would run forever, write "Forever".
- To display a function value, write "Function".
- If the evaluated expression wouldn't display anything, write "Nothing". The interactive interpreter displays the value of a successfully evaluated expression, unless it is None.
- If you need to write a level of indentation, please draw an arrow, like −>, to represent one singular level of indentation. Draw one arrow for each level of indentation you want to indicate. This may be a tab (\t) or multiple whitespaces.
- 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.
- Assume the Tree class has been defined (as written on the Study Guide provided)
- Assume the Tree data abstraction has been defined (as written on the Study Guide provided)

i. (0.5 pt)

```
>>> print("luck!", print("good"))
```

```
good
luck! None
```

ii. (1.0 pt)

```
>>> t = Tree("hi", [Tree("you"), Tree("there")])
>>> print(t)
```

```
hi
−> you
−> there
```

iii. (1.0 pt)

```
>>> t
```

```
Tree("hi", [Tree("you"), Tree("there")])
```

iv. (1.0 pt)

```
>>> t.branches.append(Tree("hey", ["hello"]))
```

```
Error
```

v. (0.5 pt)

```
>>> t.label = "hello"
>>> t.branches[0].label = "friend"
```

```
Nothing
```

vi. (1.0 pt) Do any of the above expressions in this question break the abstraction barrier?

○ YES

[✓ NO]

(b) (3.0 points) WWSD

For each expression below, choose the correct option or write the output displayed by the interactive Scheme interpreter when the expression is evaluated. The output may have multiple lines.

- If an error occurs, write "Error", but include all output displayed before the error.
- If evaluation would run forever, write "Forever".
- To display a function value, write "Function".
- If the evaluated expression wouldn't display anything, write "Nothing". The interactive interpreter displays the value of a successfully evaluated expression, unless it is undefined.
- Assume the Tree data abstraction has been defined (as written on the Study Guide provided)

i. (0.5 pt)

```
scm> (define t (tree 4 (list (tree 6 nil) (tree 2 (list (tree 7 nil))))))
```

```
t
```

ii. (1.0 pt)

```
scm> t
```

- ○ `(4 6 2 7)`
- ○ `(4 (6 2 (7)))`
- [✓ `(4 (6) (2 (7)))`]
- ○ `(tree 4 (list (tree 6 nil) (tree 2 (list (tree 7 nil))))`
- ○ `(tree 4 (tree 6 nil) (tree 2 (tree 7 nil)))`
- ○ Error
- ○ Forever
- ○ Function
- ○ Nothing

iii. (1.0 pt)

```
scm> (define f (lambda (x) (if (null? (cdr x)) (car x) (+ (car x) (f (cdr x))))))
f
scm> (f '(1 2 3))
6
```

Given this error, **select all** expressions that could have been the input to the interactive Scheme interpreter.

```
Traceback (most recent call last):
 0   (+ (car x) (f (cdr x)))
Error: operand 0 ((quote hi)) is not a number
```

- ☐ `(f '('hi))`
- ☐ `(f '(hi))`
- ☐ `(f (hi))`
- ☐ `(f (quote ((quote hi))))`
- [✓] `(f '('hi 'hello))`
- [✓] `(f '('hi 2))`
- ☐ `(f '(2 'hi))`
- ☐ `(f (2 'hi))`

iv. (0.5 pt)

```
scm> (or #f 0 2 (and 3 1 (or 5)))
```

0

(c) (2.0 points) WWRD

For each RegEx expression below, choose all options that would fully match if put as test strings in Regex101.

i. (2.0 pt)

`^b[^b-z]{1}[arken]+(bi)*es?`

- [✓] `bakes`
- [✓] `baarkenbie`
- [✓] `barre`
- ☐ `bbarbies`
- [✓] `barbie`
- ☐ `ken`
- ☐ `barbiess`
- [✓] `bakbie`

(d) (1.0 points) WWSQLD

Assume the records table that you've seen in lecture, discussion, and labs so far has been implemented in your interactive SQL interpreter so that it is this table:

name	division	title	salary	supervisor
Alyssa P Hacker	Computer	Programmer	40000	Ben Bitdiddle
Ben Bitdiddle	Computer	Wizard	60000	Oliver Warbucks
Cy D Fect	Computer	Programmer	35000	Ben Bitdiddle
Eben Scrooge	Accounting	Chief Accountant	75000	Oliver Warbucks
Lana Lambda	Administration	Executive Director	610000	Lana Lambda
Lem E Tweakit	Computer	Technician	25000	Ben Bitdiddle
Louis Reasoner	Computer	Programmer Trainee	30000	Alyssa P Hacker
Oliver Warbucks	Administration	Big Wheel	150000	Oliver Warbucks
Robert Cratchet	Accounting	Scrivener	18000	Eben Scrooge

i. (1.0 pt) Given this SQL query:

```
sql> SELECT AVG(a.salary) FROM records AS a, records AS b WHERE a.name = b.supervisor
...> GROUP BY a.division ORDER BY a.division LIMIT 2;
```

which outputs the table below, what is being displayed and represented by this new table? In other words, translate this SQL query into English pseudocode. Please summarize in 1-2 sentences. Responses with more than 2 sentences will be docked points.

AVG(a.salary)
75000
265000

This shows the average salary for supervisors who are in the accounting and administration divisions.
OR
The first two rows of the average salary column of the records table joined with the records table where the kept salary is from a supervisor grouped by the division ordered by the division in ascending order.
Any attempt to describe a table, such as, "a table with two rows and 1 column of the average salary" will receive 0 points.

2. (12.0 points) Slipknot

(a) (2.0 points) Product

Implement `product`, a function that takes in a linked list `s` and returns the product over all the values in the linked list. Assume that the initial list `s` is not `Link.empty`.

```
def product(s):
    """
    Takes in a linked list S and returns the product over all the values
    in the linked list.

    >>> product(Link(2, Link(5))) # 2 * 5 = 10
    10
    >>> product(Link(7, Link(2, Link(7)))) # 7 * 2 * 7 = 98
    98
    >>> product(Link(5))
    5
    """
    if s is Link.empty:
        return _________
                   (a)
    return _________
               (b)
```

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

```
1
```

ii. (1.5 pt) Fill in blank (b)

```
s.first * product(s.rest)
```

(b) (5.0 points) Untangle

Implement `untangle`, a function that takes in a linked list `s` with strictly positive values. The function **mutates** the linked list to only contain 1s, where the length of the resulting linked list is equal to the sum of the values in the original list. `untangle` potentially modifies the values in `s` and adds new `Link`s to `s`.

For example, calling `untangle` on `<3 1 2>` will mutate the linked list to become `<1 1 1 1 1 1>`. See the doctests for more examples.

```
def untangle(s):
    """
    Takes in a linked list S and mutates S to only contain 1s.
    The length of the resulting list is equal to the sum of the
    values in the original list. Assume the values in S are strictly
    positive.

    >>> a = Link(5)
    >>> untangle(a)
    >>> print(a)
    <1 1 1 1 1>
    >>> b = Link(3, Link(2, Link(4)))
    >>> untangle(b)
    >>> print(b)
    <1 1 1 1 1 1 1 1 1>
    >>> c = Link(1)
    >>> untangle(c)
    >>> c
    Link(1)
    """
    if _________:
          (a)
        return
    next = _________
              (b)
    while _________:
             (c)
        s.rest = _________
                   (d)
        s.first = _________
                    (e)
    _________
       (f)
```

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

```
s is Link.empty
```

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

```
s.rest
```

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

- ○ `s is not Link.empty`
- ○ `s.rest is not Link.empty`
- ○ `s is not Link.empty and s.rest is not Link.empty`
- ○ `s is not Link.empty or s.rest is not Link.empty`
- ○ `s.first > 0`
- ○ `s.first < 0`
- [✓ `s.first > 1`]
- ○ `s.first < 1`

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

- ○ `1`
- ○ `s`
- ○ `Link(1)`
- ○ `Link(1, s)`
- [✓ `Link(1, s.rest)`]
- ○ `Link(s, Link(1))`
- ○ `Link(s.first, Link(1))`

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

- ○ `s`
- ○ `s.rest`
- ○ `s.rest.first`
- [✓ `s.first - 1`]
- ○ `s.first + 1`
- ○ `-1 * s.first`

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

```
untangle(next)
```

(c) (5.0 points) Knots

Given a linked list with only 1s, we can create knots inside the list. A **knot** is formed by merging multiple Links with 1s into a single Link, where the resulting Link stores the number of 1 nodes that were merged together.

For example, given a linked list of <1 1 1 1>, we can generate the following knotted linked lists:

```
<2 1 1> # Linked list with <(1 + 1), 1, 1>
<1 2 1> # Linked list with <1, (1 + 1), 1>
<1 1 2> # Linked list with <1, 1, (1 + 1)>
<2 2> # Linked list with <(1 + 1), (1 + 1)>
<3 1> # Linked list with <(1 + 1 + 1), 1>
<1 3> # Linked list with <1, (1 + 1 + 1)>
<4> # Linked list with <(1 + 1 + 1 + 1)>
```

Implement knot, a function that takes in a linked list s with only 1s and returns a **new** linked list with one or more knots. knot should return the knotted linked list that has the largest product over all of its values. If there is a tie between multiple linked lists, return any list. Assume s has at least 2 nodes.

You may use functions implemented in previous subparts (product, untangle) to solve this question.

```
def knot(s):
"""
    Takes in a linked list S with only 1s and returns a new knotted linked list.
    The returned knotted linked list has the largest product over all of its values.
    If there is a tie between multiple linked lists, return any list.
    Assume S has at least 2 nodes.

    >>> a = Link(3)
    >>> untangle(a) # <1 1 1>
    >>> knot(a) # <1 1 1> --> <3>
    Link(3)
    >>> b = Link(5)
    >>> untangle(b) # <1 1 1 1 1>
    >>> knot(b) # <1 1 1 1 1> --> <2 3> or <3 2>
    Link(2, Link(3))
    >>> c = Link(10)
    >>> untangle(c) # <1 1 1 1 1 1 1 1 1 1>
    >>> knot(c) # <1 1 1 1 1 1 1 1 1 1> --> any list with two 2s and two 3s
    Link(2, Link(2, Link(3, Link(3))))
    """
    if _________:
          (a)
        return s
    next = s.rest
    keep_first = Link(_________, _________)
                        (b)        (c)
    merge_first = knot(_________)
                         (d)
    return _________([keep_first, merge_first], key=_________)
              (e)                                     (f)
```

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

- ○ `s is Link.empty`
- [✓ `s.rest is Link.empty`]
- ○ `s is Link.empty and s.rest is Link.empty`
- ○ `s.first == 1`
- ○ `s.first != 1`
- ○ `s.first <= 1`

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

```
s.first
```

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

```
knot(next) -OR- knot(s.rest)
```

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

- ○ `Link(next.first, next)`
- ○ `Link(next.first, next.rest)`
- ○ `Link(s.first, next)`
- ○ `Link(s.first, next.rest)`
- ○ `Link(s.first + next.first, next)`
- [✓ `Link(s.first + next.first, next.rest)`]
- ○ `Link(next.first + next.rest.first, next)`
- ○ `Link(next.first + next.rest.first, next.rest)`

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

- ○ `knot`
- ○ `sum`
- [✓ `max`]
- ○ `min`
- ○ `map`
- ○ `filter`

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

```
product
```

3. OOPeration

(a) (3.0 points) Patient

Leena is a hospitalist and is dissatisfied with the current system that organizes patient information at her hospitals, Kryser and Stanville, which have different admit and discharge protocols. Since some patients are admitted at multiple hospitals (somehow), the system gets confused.

You suggest representing each patient with a specific set of attributes - `name` (a string), patient `ID` (an integer), `age` (an integer), and `admitted`, a boolean that indicates whether they are currently admitted at a hospital.

A valid patient ID must be a seven-digit number, with the first three digits being less than 4 and the last four digits being between 0-9.

Start by implementing the `Patient` class.

```
class Patient:
    """
    >>> kim = Patient("Kim", 212764, 12)
    Invalid Patient ID entered. Try again.
    >>> kim.ID
    >>> kim.set_ID(3337656)
    Valid Patient ID
    >>> kim.ID
    3337656
    >>> tina = Patient("Tina", 2219542, 93)
    Valid Patient ID
    """
    def __init__(self, name, ID, age):
        self.name = name
        self.age = age
        self.admitted = False
        _________
           (a)

    def set_ID(self, ID):
        if re.match(_________, f"{ID}"):
                       (b)
            self.ID = ID
            print("Valid Patient ID")
        else:
            _________
               (c)
            print("Invalid Patient ID entered. Try again.")
```

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

```
self.set_ID(ID)
```

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

```
r"^[0-3]{3}\d{4}$"
```

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

```
self.ID = None
```

(b) (6.0 points) Hospital

Implement the `Hospital` class. A `Hospital` instance has two instance attributes: `name` and `patient_list`, which keeps track of all the patients (`Patient` instances) admitted to that hospital.

```
class Hospital:
    """
    >>> clinic42 = Hospital("StanKrys")
    >>> neel = Patient("Neel", 3125456, 63)
    Valid Patient ID
    >>> clinic42.admit(neel)
    >>> clinic42.admit_from_list([Patient("Ne", 3125456, 63), Patient("Sam", 1125432, 14)])
    Valid Patient ID
    Valid Patient ID
    Patient with ID 3125456 is already admitted
    >>> clinic42.patient_IDs_list()
    [3125456, 1125432]
    >>> clinic42.discharge(clinic42.find_patient(3125456))
    >>> clinic42.patient_IDs_list()
    [1125432]
    """
    hospitals = {}
    def __init__(self, name):
        self.patient_list = []
        _________
           (a)

    def admit(self, patient):
        IDs = _________
                 (b)
        if patient.ID in IDs:
            print(f"Patient with ID {patient.ID} is already admitted")
        else:
            _________
               (c)
            patient.admitted = True

    def admit_from_list(self, patient_list):
        for p in patient_list:
            _________
               (d)

    def discharge(self, patient):
        if patient.ID in self.patient_IDs_list():
            _________
               (e)
            patient.admitted = False

    def patient_IDs_list(self):
        return _________
                  (f)

    def find_patient(self, ID):
        for p in self.patient_list:
            if ID == p.ID:
                return p
```

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

○ `self.hospitals = self.hospitals.append(self)`

○ `hospitals[name] = self`

[✓ `Hospital.hospitals[name] = self`]

○ `hospitals.append(self)`

○ `self.hospitals = self.hospitals.extend(self)`

○ `Hospital.hospitals = {self.name: self}`

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

○ `self.patient_list`

○ `patient_IDs_list(self)`

○ `self.patient_IDs_list`

○ `self.hospitals.extend(self)`

○ `self.patient_IDs_list(self.patient_list)`

[✓ `self.patient_IDs_list()`]

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

```
self.patient_list.append(patient)
```

iv. (1.0 pt) Select all that apply for blank (d)

☐ `self.admit_from_list(p)`

[✓ `self.admit(p)`]

☐ `patient_IDs_list`

☐ `self.patient_list.append(patient)`

[✓ `Hospital.admit(self,p)`]

☐ `self.patient_list.extend(patient)`

☐ `p.admit(self)`

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

```
self.patient_list.remove(patient)
```

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

```
[p.ID for p in self.patient_list]
```

(c) (7.0 points)

Implement the `Kryser` and `Stanville` classes.

Here are the protocols:

- An ID is unique to each patient so `Patient("Sam", 1234511, 61)` is the same as `Patient("Samantha", 1234511, 60)`. This ensures the system doesn't have duplicate medical charts for a patient.
- Stanville will **not** discharge patients who are also admitted at Kryser
- Kryser will **not** admit patients who are also admitted at Stanville
- There is only one Kryser Hospital and one Stanville Hospital with the names "Kryser" and "Stanville," respectively.
- Assume these are the only two hospitals that patients can be admitted to

```
class Kryser(Hospital):

    """
    >>> k = Kryser("Kryser")
    >>> s = Stanville("Stanville")
    >>> nora = Patient("Nora", 3125456, 63)
    Valid Patient ID
    >>> jim = Patient("Jim", 2125496, 34)
    Valid Patient ID
    >>> s.admit(nora)
    >>> nora.admitted
    True
    >>> k.admit(nora) # Kryser will NOT admit Stanville patients
    Error: Patient with ID 3125456 is already admitted at Stanville
    >>> k.admit(jim)
    >>> k.admit(Patient("Jimmy", 2125496, 36)) # same patient
    Valid Patient ID
    Patient with ID 2125496 is already admitted
    >>> k.patient_IDs_list()
    [2125496]
    >>> s.admit(jim) # Stanville is okay admitting Kryser patients
    >>> s.discharge(jim)
    Error: Patient with ID 2125496 hasn't been discharged from Kryser
    >>> k.discharge(jim)
    >>> s.discharge(jim)
    >>> jim.admitted
    False
    """


    def admit(self, patient):
        if _________ and not patient in self.patient_list:
              (a)
            print(f"Error: Patient with ID {patient.ID} is already admitted at Stanville")
        else:
            _________.admit(patient)
               (b)

    def discharge(self, patient):
        _________
           (c)
```

```
class Stanville(_________):
                   (d)

    def admit(self, patient):
        _________
           (e)

    def discharge(self, patient):
        kryser = _________
                      (f)
        if patient.admitted and patient.ID in kryser.patient_IDs_list():
            print(f"Error: Patient with ID {patient.ID} hasn't been discharged from Kryser")
        else:
            _________
               (g)
```

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

`patient.admitted`

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

- ○ `self`
- ○ `Hospital`
- [✓ `super()`]
- ○ `self.Hospital`
- ○ `super`

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

`super().discharge(patient)`

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

- [✓ `Hospital`]
- ○ `Kryser`
- ○ `Hospital, Kryser`
- ○ `Kryser, Hospital`
- ○ `None`
- ○ `Patient`
- ○ `None of the answers are correct`

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

```
super().admit(patient)
```

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

```
Hospital.hospitals["Kryser"]
```

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

- ○ `self.discharge(patient)`
- [✓ `super().discharge(patient)`]
- ○ `Kryser.discharge(patient)`
- ○ `Kryser.discharge(self, patient)`
- ○ `None`
- ○ `patient`
- ○ `None of the answers are correct`

(d) (3.0 points) Find Patient

The system at Stanville glitched and some of the patient information (`name`, `ID`, and `age`) got shuffled. Leena urgently needs the `ID` of the patient she was going to see next. Luckily, she knows the patient's age, that they were only admitted at Stanville, and that the product of their `ID` digits equals their `age`. Implement `find_patient`, which takes in the patient's age `age` and returns a function, `checker`, to which Leena can feed in the shuffled IDs until she finds the correct ID to return the `Patient` object.

You may use the `Hospital` class and assume the `Stanville` hospital object exists.

```
def find_patient(age):
    """
    >>> s = Stanville("Stanville")
    >>> grace = s.admit(Patient("grace", 1121717, 98))
    Valid Patient ID
    >>> marie = s.admit(Patient("marie", 2121141, 16))
    Valid Patient ID
    >>> machine = find_patient(98)
    >>> machine(1211311)(1023209)(1121717)           # 1 * 1 * 2 * 1 * 7 * 1 * 7 = 98
    <__main__.Patient object>
    >>> find_patient(16)(1211311)(1123113)(1111111)(2121141)
    <__main__.Patient object>

    """
    def checker(ID):
        prod = 1
        parse_digits = ID
        while parse_digits > 0:
            parse_digits, last = parse_digits // 10, parse_digits % 10
            prod *= last
        if prod == age:
            stanville = _________
                           (a)
            return stanville._________
                                (b)
        else:
            return _________
                      (c)
    return checker
```

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

```
Hospitals.hospitals["Stanville"]
```

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

```
find_patient(ID)
```

iii. (1.0 pt) Select all that apply for blank (c)

- ☐ `checker(age)`
- ☐ `find_patient`
- ☐ `find_patient()`
- [✓ `find_patient(age)`]
- [✓ `checker`]
- ☐ `checker(x)`
- ☐ `find_patient(x)`

4. (8.0 points) Ratios

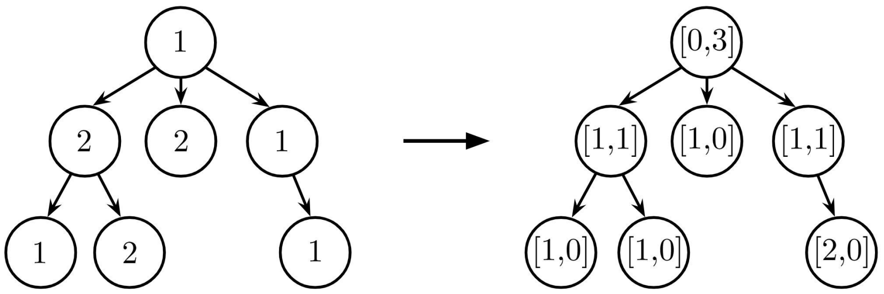
(a) Definition: The number of labels **above** tree `t` is the number of labels along the path from this node to the root of `t` **excluding the current node**.

Definition: The number of labels **below** tree `t` is the number of labels within a subtree `t` **excluding the current node**.

Implement `ratio`, which takes a Tree `t` and predicate function `f`, and **mutates** `t` into a Ratio Tree. A Ratio Tree replaces the value at each label with a 2-element list. Index 0 holds the number of labels above this node for which `f(label)` returns true. Index 1 holds the number of labels below this node for which `f(label)` returns true.

For example, consider the label 2 in the first branch of the root of the tree shown below. 2 has one label along the path to the root that is odd (1). 2 has one label within its subtrees that are odd (1). As a result, the label is replaced with the list [1, 1]

```
>>> t = Tree(1, [Tree(2, [Tree(1), Tree(2)]), Tree(2), Tree(1, [Tree(1)])])
>>> is_odd = lambda x: x % 2 == 1
>>> ratio(t, is_odd)
```



```
def ratio(t, f):
    """Mutate each label of t to the ratio of the number of labels above
       it to the number of labels below it, represented as a 2-element
       list - provided that f(label) == true.
    >>> t = Tree(15, [Tree(1, [Tree(8)]), Tree(5), (Tree(4, [Tree(6), Tree(7, [Tree(9)])]))])
    >>> pred = lambda x: x % 2 == 1 # odd?
    >>> ratio(t, pred)
    >>> t
    Tree([0, 4], [Tree([1, 0], [Tree([2, 0])]), Tree([1, 0]), Tree([1, 2],
    [Tree([1, 0]), Tree([1, 1], [Tree([2, 0])])])])
    """
    def helper(t, filtered_depth):
        above = _________
                    (a)
        if _________:
              (b)
            _________
               (c)
        below = _________
                    (d)
        for b in t.branches:
            _________
               (e)
        below += _________
                     (f)
        t.label = [above, below]
    return helper(t, 0)
```

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

```
filtered_depth
```

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

- ○ `t.label`
- ○ `t.is_leaf()`
- [✓ `f(t.label)`]
- ○ `f(t.branches)`
- ○ `any([f(b.label) for b in t.branches])`
- ○ `all([f(b.label) for b in t.branches])`

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

- ○ `return f(t.label)`
- ○ `return above`
- ○ `t.label = [above, 0]`
- ○ `t.label = [0, 0]`
- [✓ `filtered_depth += 1`]
- ○ `above += 1`

iv. **(1.5 pt)** Fill in blank (d)

- [✓ `len([b for b in t.branches if f(b.label)])`]
- ○ `sum([b for b in t.branches if f(b.label)])`
- ○ `sum([b.label for b in t.branches])`
- ○ `sum([helper(b, filtered_depth) for b in t.branches])`
- ○ `len([helper(b, filtered_depth) for b in t.branches])`
- ○ `all([helper(b, filtered_depth) for b in t.branches])`

v. **(2.0 pt)** Fill in blank (e). You **cannot** use `and`, `or`, `if`, or `else`.

```
helper(b, filtered_depth)
```

vi. (1.5 pt) Fill in blank (f)

- ○ `len([b for b in t.branches if f(b.label)])`
- ○ `sum([b for b in t.branches if f(b.label)])`
- [✓ `sum([b.label[1] for b in t.branches])`]
- ○ `len([b.label[0] for b in t.branches if f(b.label)])`
- ○ `sum([helper(b, filtered_depth) for b in t.branches])`
- ○ `len([helper(b, filtered_depth) for b in t.branches])`

5. (8.0 points) Kirby Consume

(a) Consumable?

Kirby is stuck in a forest of Whispy Woods. In order to get out, he needs to consume Maxim Tomatoes from vines along the way. However, tomato vines are picky about temperature and only grow tomatoes above or below a certain threshold temperature.

In fact, each vine can be represented using these three attributes: indicator, threshold, and tomatoes. A vine grows tomatoes if one of the following is true:

- It has a **positive** indicator and the current temperature is strictly **above** the threshold.
- It has a **negative** indicator and the current temperature is strictly **below** the threshold.

You may **assume that an indicator is never 0**.

If the vine can grow tomatoes, Kirby will consume a certain number of tomatoes from the vine, indicated by the third attribute of a vine object.

We choose to use the following Scheme data abstraction to represent a vine:

Constructor: `(make-vine ind thres tomatoes)` creates an vine object with the given indicator, threshold, and tomatoes.

Selectors:

- `(get-ind vine)` returns the vine's indicator
- `(get-thres vine)` returns the vine's threshold
- `(get-tomatoes vine)` returns the vine's number of tomatoes

Note: You may assume that the vine ADT has been properly implemented already.

Implement `consumable?`, which takes in an vine object, `vine`, and the current temperature, `temp`. It returns `#t` if the vine can be consumed from given the current temperature, and `#f` otherwise. `(consumable? (make-vine -1 31 10) 0)` evaluates to `#t`, as the current temperature of `0` is below the threshold of `31` and the indicator of `-1` is negative.

```
; doctests
(define v (make-vine -1 31 10))
(expect (consumable? v 0) #t) ; -1 is negative and 0 < 31
(expect (consumable? v 32) #f) ; -1 is negative and 32 > 31

(define (consumable? vine temp)
    ( _________
        temp
        (get-thresh vine))
)
```

i. (2.0 pt) Fill in the blank. **Hint:** Your expression should evaluate to an operator.

```
(if (< (get-ind vine) 0) < >)
```

(b) (6.0 points) Kirby Consume

Implement `kirby-consume`, which takes in a list of vine objects `vines`, the current temperature `temp`, and `min-tomatoes`, the minimum amount of tomatoes Kirby needs to consume from the vines in order to get out. It returns `#t` if Kirby was able to consume enough tomatoes to get out and `#f` otherwise.

Note: You may assume that `consumable?` has been implemented correctly regardless of your solution to the previous part, and you may call it in your implementation for this part. You **may leave lines blank** and can specify so by answering `Blank` for the question. Your implementation **must be tail recursive**. Any non-tail recursive solution will receive no credit.

Hint: First try writing a tail recursive solution without looking the multiple choice options.

```
; doctests
(define v1 (make-vine 1 20 2))
(define v2 (make-vine 1 25 5))
(define v3 (make-vine -1 30 3))
(define vines (list v1 v2 v3))
(expect (kirby-consume vines 20 8) #f) ; 0 + 0 + 3 < 8
(expect (kirby-consume vines 40 7) #t) ; 2 + 5 + 0 = 7
(expect (kirby-consume vines 27 9) #t) ; 2 + 5 + 3 > 9

(define (kirby-consume vines temp min-tomatoes)
    (if (null? vines)
        _________
           (a)
        (begin
            (define v (car vines))
            (_________
                (b)
            _________
               (c)
            _________
               (d)
            _________))))
               (e)
```

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

- ○ `0`
- ○ `#t`
- ○ `#f`
- ○ `min-tomatoes`
- ○ `(> min-tomatoes 0)`
- [✓ `(<= min-tomatoes 0)`]

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

- ○ `cons`
- [✓ `kirby-consume`]
- ○ `and`
- ○ `or`
- ○ `temp`
- ○ `<`
- ○ `>`
- ○ `Blank`

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

- [✓ `(cdr vines)`]
- ○ `vines`
- ○ `(car vines)`
- ○ `nil`
- ○ `temp`
- ○ `(>= min-tomatoes 0)`
- ○ `(<= min-tomatoes 0)`
- ○ `Blank`

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

- ○ `0`
- [✓ `temp`]
- ○ `(get-ind (car vines))`
- ○ `(get-tomatoes (car vines))`
- ○ `(kirby-consume (cdr vines) temp (- min-tomatoes (get-tomatoes v)))`
- ○ `(kirby-consume (cdr vines) temp min-tomatoes)`
- ○ `Blank`

v. (2.0 pt) Fill in blank (e).

- ○ `(+ min-tomatoes (get-tomatoes v))`
- ○ `(- min-tomatoes (get-tomatoes v))`
- ○ `(if (consumable? v temp) min-tomatoes 0))`
- ○ `(and (consumable? v temp) (kirby-consume (cdr vines) temp min-tomatoes))`
- ○ `(or (consumable? v temp) (kirby-consume (cdr vines) temp min-tomatoes))`
- [✓ `(- min-tomatoes (if (consumable? v temp) (get-tomatoes v) 0))`]
- ○ `(+ min-tomatoes (if (consumable? v temp) (get-tomatoes v) 0))`
- ○ `Blank`

6. (9.0 points) Matchmaking

(a) (6.0 points) do_match_form

Modify scheme to support the new special form `match`. A `match` expression has the following form: `(match <subject> <clause1> <clause2> ... )`

A `<clause>` has the following form `(<case> <consequent>)`. A `<subject>` is any expression.

The `match` expression first evaluates `<subject>`.

Starting with the first clause, evaluates `<case>`. If it is scheme equal to the evaluated subject, i.e `(equal? <subject> <case>)` is true, evaluate the `<consequent>` and return it. If they are not scheme equal, proceed to the next clause. If there are no more clauses, the return value is undefined.

```
scm> (define op 'add)
scm> (match op
...>        ('mul (* 1 2))
...>        ('add (+ 1 2))
...>        ('sub (- 1 2)))
3
scm> (match op
...>        ((print 'mul) (print '2))
...>        ('add (+ 1 2))
...>        ('(print 'sub) (- 1 2)))
mul
3
scm> (match op ('mul (* 1 2) (* 3 4))) ; More than two elements in a <clause>
Traceback (most recent call last):
  0     (match op ('mul (* 1 2) (* 3 4)))
Error: too many operands in form
```

Assume that the rest of the Scheme interpreter has been implemented correctly and only `do_match_form` needs to be implemented. You may use the following functions defined in the Scheme interpreter project: `validate_form`, `scheme_equalp`, `scheme_eval`

```
def do_match_form(expressions, env):
    """Evaluate a match form.
    >>> env = create_global_frame()
    >>> do_match_form(read_line("( 'c ('a 1) ('b (+ 1 1)) ('c 3))"), env)
    3
    """
    validate_form(expressions, 1) # Ensures there's at least 1 operand
    subject = _________
                 (a)
    val = scheme_eval(subject, env)
    expressions = expressions.rest
    while _________:
             (b)
        pair = expressions.first
        _________
           (c)
        match_term = _________
                          (d)
        if _________(val, match_term):
              (e)
            return scheme_eval(_________, env)
                                    (f)
        expressions = expressions.rest
```

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

```
expressions.first
```

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

- ● `expressions is not nil`
- ○ `expressions.rest is not nil`
- ○ `expressions.first`
- ○ `len(expressions) > 0`
- ○ `len(expressions) >= 0`

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

- ○ `scheme_eval(subject, env)`
- ○ `scheme_eval(pair, env)`
- ○ `scheme_equal(expressions, val)`
- ○ `scheme_equal(pair, val)`
- ○ `validate_form(expressions, 2, 2)`
- ● `validate_form(pair, 2, 2)`

iv. (1.5 pt) Fill in blank (d)

```
scheme_eval(pair.first, env)
```

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

- ○ `scheme_eval`
- ○ `validate_form`
- ● `scheme_equalp`
- ○ `do_match_form`
- ○ `min`

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

```
pair.rest.first
```

(b) (3.0 points) Match Counter

Assume the interpreter supports the `match` special form and `do_match_form` is implemented correctly. Let us consider interpreting the following expressions:

```
scm> (define op 'add)
op
scm> (match op
...>        ('mul (* 1 2))
...>        ('add (+ 1 2))
...>        ('sub (- 1 2)))
```

The following questions relate to the previous `match` expression. Each clause is in context of the given `match` expression.

i. (1.0 pt) How many times is `scheme_eval` called for **just** this clause of the given `match` special form:

`('mul (* 1 2))`

```
1
```

ii. (1.0 pt) How many times is `scheme_eval` called for **just** this clause of the given `match` special form:

`('add (+ 1 2))`

```
5
```

iii. (1.0 pt) How many times is `scheme_eval` called for **just** this clause of the given `match` special form:

`('sub (- 1 2))`

```
0
```

7. (9.0 points) Hills

(a) (9.0 points) Hill

Implement the generator function `hill` which takes in a positive integer `n` and returns a generator that yields every subsequence of `n` where each digit is exactly 1 away from its adjacent digits. The order in which numbers are yielded does not matter. Assume all digits in the number are unique.

```
def hill(n):
    """
    Accepts a positive integer N, and returns a generator that
    yields every subsequence of N where each digit is exactly 1
    away from its adjacent digits.

    >>> sorted(list(hill(354)))
    [3, 4, 5, 34, 54]
    >>> sorted(list(hill(246))) # individual digits are hills themselves
    [2, 4, 6]
    >>> sorted(list(hill(32451)))
    [1, 2, 3, 4, 5, 21, 32, 34, 45, 321, 345]
    """
    _________
       (a)
    if n >= 10:
        _________:
           (b)
            _________
               (c)
            if _________ == 1:
                  (d)
                _________
                   (e)
```

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

```
yield n % 10
```

ii. (1.5 pt) Fill in blank (b).

○ `for x in hill(n - 1)`

[✓ `for x in hill(n // 10)`]

○ `for x in range(n)`

○ `for x in range(n + 1)`

○ `while True`

○ `while n > 0`

○ `if n > 0`

○ `if n % 10`

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

```
yield x
```

iv. (1.5 pt) Fill in blank (d).

- ○ `x // pow(10, n)`
- ○ `n // x`
- ○ `abs(x - n % 10)`
- ○ `abs(x // 10 - n % 10)`
- [✓ `abs(x % 10 - n % 10)`]
- ○ `abs(n)`
- ○ `x % 10 - n // 10`

v. (2.0 pt) Fill in blank (e).

```
yield x * 10 + n % 10
```

8. (10.0 points) List Methods

(a) (5.0 points) Pad

Implement `pad` which takes in a list `s` of length `m`, and adds `el` to the end of the list until there are exactly `n` elements. If length `m` is greater than `n`, evaluate to the same list. Example: `(pad '(1 2 3) 6 4)` evaluates to `(1 2 3 4 4 4)`. `(pad '(1 2 3) 0 5)` evaluates to `(1 2 3)`. You **cannot use** `and`, `or`, `begin`, `cond`, or `if` in the blanks.

```
(define (pad s n el)
    (cond
        ((= n 0) _________)
                    (a)
        ((null? s) (cons _________ _________))
                            (b)       (c)
        (else (cons _________ _________))))
                      (d)       (e)
```

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

```
s
```

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

- ○ `s`
- ○ `(cdr s)`
- ○ `(car s)`
- [✓ `el`]
- ○ `n`

iii. (1.5 pt) Fill in blank (c)

```
(pad nil (- n 1) el)
```

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

- ○ `s`
- ○ `(cdr s)`
- [✓ `(car s)`]
- ○ `el`
- ○ `n`

v. (1.5 pt) Fill in blank (e)

```
(pad (cdr s) (- n 1) el)
```

(b) (5.0 points) Shorten

The `list-concat` procedure takes in a list `a` and a list `b`, and returns the result of appending each element from list `b` to the end of `a`. Below is the implementation for reference:

```
(define (list-concat a b)
    (if (null? a)
    b
    (cons (car a) (list-concat (cdr a) b))))
```

Implement shorten which takes in a list `s` of length `m`, and removes elements from the end of the list until there are exactly `n` elements. If length `m` is less than `n`, return the same list. `(shorten (list 1 2 3 4 5) 3)` evaluates to `(1 2 3)`. You **must** use the `list-concat` procedure defined previously. You **cannot use** `and`, `or`, `begin`, `cond`, or `if` in the blanks.

```
(define (shorten s n)
    (define (shorten-tail s n t)
        (if _________
               (a)
            _________
               (b)
            (shorten-tail (cdr s) (- n 1) _________)))
                                              (c)
    (shorten-tail s n nil))
```

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

● `(or (null? s) (= n 0))`

○ `(and (null? s) (= n 0))`

○ `(and s (= n 0)`

○ `(null? s)`

○ `(= n 0)`

○ `(< n 0)`

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

```
t
```

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

```
(list-concat t (list (car s)))
```

iv. (1.5 pt) What is the space complexity of `shorten` with respect `n`? Assume `n` < `m`.

○ $\theta(\log(n))$

○ $\theta(1)$

● $\theta(n)$

○ $\theta(n^2)$

○ $\theta(2^n)$

v. (1.5 pt) What is the time complexity of `shorten` with respect `n`? Assume `n` < `m`.

- ○ $\theta(\log(n))$
- ○ $\theta(1)$
- ○ $\theta(n)$
- [✓ $\theta(n^2)$]
- ○ $\theta(2^n)$

9. (10.0 points) S QUBE L

(a) (10.0 points)

The (abridged version) of the table below is named `ranks`. It stores the best result that a competitor has achieved in competition for each event in official competitive Rubik's Cube speedsolving competitions. The three columns have the following descriptions:

- `personId`: An identifier for each individual person in the `ranks` table. All rows are stored as a **string**
- `eventId`: The corresponding event of each row. `333` refers to regular 3x3x3 Cube speedsolving, while `333oh` refers to 3x3x3 Cube speedsolving with one hand. All rows are stored as a **string**
- `result`: The result (in **centiseconds**) of each row. All rows are stored as an **integer**

personId	eventId	result
2018KHAN28	222	101
2017TSVE02	222	112
2022VISH01	222	113
...	...	...
2018WIDJ01	333	669
2013BALI01	333	670
2016OLZI01	333	670
...	...	...
2016SONG03	333oh	1276
2015WILS05	333oh	1276
2016SMUL01	333oh	1277
...	...	...

Write a query that outputs all competitors with a **faster result** in One-Handed solving (`333oh`) than regular 3x3x3 speedsolving (`333`). Your query should *also* output the following:

- The `personId` corresponding to each `result`
- The fastest 3x3x3 Cube One Handed `result` in **seconds**, with a column name of `333oh_best`
- The fastest 3x3x3 Cube `result` in **seconds**, with a column name of `333_best`
- The output should be ordered in ascending order by the fastest One Handed `result`.

Your output will look similar to the following:

personId	333oh_best	333_best
2017TUNG13	9.28	9.35
2021TAKA01	14.38	18.07
2015HEER02	14.98	15.38
2013ZHAN31	16.9	19.03
...	...	...

You may assume that every row in the `ranks` table will have a valid `result`.

You may *not* use `LIKE` in your solution.

Hint: To convert centiseconds to seconds, you need to divide the column by 100.

```
SELECT _______, _______, _______
         (a)      (b)      (c)
FROM _______ AS reg, _______ AS oh
       (d)             (e)
WHERE _______________
            (f)
ORDER BY _______;
           (g)
```

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

```
reg.personId
```

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

```
oh.result AS 333oh_best
```

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

```
reg.result AS 333_best
```

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

```
ranks
```

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

```
ranks
```

vi. (5.0 pt) Fill in blank (f).

```
reg.personId = oh.personId AND reg.eventId = '333' AND oh.eventId =
'333oh' AND reg.result > oh.result
```

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

```
oh.result
```

10. (3.0 points) Special Topics

(a) (1.0 pt) ComP(ile)ython

In **CPython**, what is the order of intermediate results when going from source code to output?

○ `source->AST->output`

○ `source->bytecode->output`

○ `source->machine code->bytecode->output`

○ `source->AST->tokens->bytecode->output`

[✓ `source->tokens->AST->bytecode->output`]

○ `source->tokens->AST->machine code->bytecode->output`

(b) (1.0 pt) BlAST OFF

Assume your query sequence is `TACTGCGATA` which encodes for the insulin gene.

What is displayed when you run the BLAST algorithm (like we did on the NCBI website) on your query sequence and a database of 10 other insulin gene target sequences?

○ **Sequence** with the *highest* genetic similarity to the query sequence using **local** alignment

○ **Sequence** with the *highest* genetic similarity to the query sequence using **global** alignment

○ **Sequence** with the *highest* genetic similarity to the query sequence using **local or global** alignment

○ **Highest genetic similarity** of the query sequence with the target sequences using *local alignment*

○ **Highest genetic similarity** of the query sequence with the target sequences using *global alignment*

[✓ **Genetic similarities** of the query sequence with *all* target sequences using *local* alignment]

○ **Genetic similarities** of the query sequence with *all* target sequences using *global* alignment

○ **Top 3 highest genetic similarities** of the query sequence with target sequences using *global* alignment

(c) (1.0 pt) Pickle Tree

Given the Tree class on the Study Guide provided, select the option to fill in blank (a) the `__reduce__` function that would be used by calling `pickle.dumps(tree_obj)`:

```
def __reduce__(self):
    return _________
                (a)
```

[✓ `(self.__class__, (self.label, self.branches))`]

○ `(__class__, (label, branches))`

○ `[self.__class__, [self.label, self.branches]]`

○ `[__class__, [label, branches]]`

○ `self`

○ `str(self)`

○ `repr(self)`

11. (0.0 points) A+ Questions

(a) (0.0 points) A+ Levels

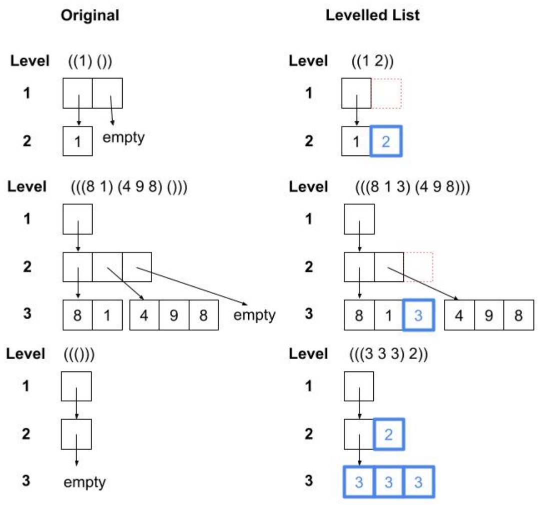
This A+ question is not worth any points. It can only affect your course grade if you have a high A and might receive an A+. Finish the rest of the exam first!

Definition: The level of a list refers to how deeply nested it is. Example: `(1)` has a level of 1. `((2))` has a level of 1. The inner list of `(2)` has a level of 2.

Implement `levels` that takes in a list `s`. It returns a new levelled list such that at each depth level L of the list there are exactly L elements. If there are more than L elements. Keep only the first L elements. If there are fewer than L elements. Add the depth level L as an element until there are exactly L elements.

You **can use** `shorten` and `pad` in your implementation. You **cannot use** `and`, `or`, `begin`, `cond`, or `if` in the blanks. Examples:

```
scm> (levels '(((8 1) (4 9 8) ()))) ; add 3 at depth 3. Remove empty list at depth 2.
(((8 1 3) (4 9 8)))
scm> (levels '((()))) ; add 3s at depth 3. Add 2s at depth 2.
(((3 3 3) 2))
```



```
(define (levels s)
    (define (helper s level)
        (cond
            ((integer? s) _________)
                                (a)
            ((null? s) _________)
                            (b)
            (else
                (begin
                    (define first _________)
                                      (c)
                    (define rest _________)
                                     (d)
                    _________))))
                       (e)
    (helper s 1))
```

i. (0.0 pt) Fill in blank (a)

```
s
```

ii. (0.0 pt) Fill in blank (b)

```
(pad s level level)
```

iii. (0.0 pt) Fill in blank (c)

```
(helper (car s) (+ level 1))
```

iv. (0.0 pt) Fill in blank (d)

```
(helper (cdr s) level)
```

v. (0.0 pt) Fill in blank (e)

```
(shorten (cons first rest) level)
```

(b) (0.0 points) A+ SQL

This A+ question is not worth any points. It can only affect your course grade if you have a high A and might receive an A+. Finish the rest of the exam first!

You are given the following tables:

`results`

eventId	result	personName	personId
...	...	...	...
333oh	1272	Andrew Bae	2014BAEA01
333oh	1318	Justin Mallari	2010MALL01
333oh	1592	Eric Zhao	2010ZHAO19
...	...	...	...
444	2917	Sebastian Weyer	2010WEYE02
444	2995	Sebastian Weyer	2010WEYE02
444	2973	Feliks Zemdegs	2009ZEMD01
...	...	...	...
skewb	1205	Riley Woo	2007WOOR01
skewb	1327	Tiffany Chien	2012CHIE01
skewb	1347	Jeong-Soo Park	2014PARK03
...	...	...	...

`persons`

name	country	id
...	...	...
Barrett Maun	USA	2022MAUN01
Ben Yin	USA	2022YINB01
Bowen Wang	USA	2022WANG56
...	...	...

`results` is a table that has 4 columns:

- `eventId`: The corresponding event of each row. Similarly to the previous SQL question, `333oh` refers to 3x3x3 Cube speedsolving with one hand. All rows are stored as a **string**
- `result`: The result (in **centiseconds**) of each row. All rows are stored as an **integer**
- `personName`: The name for each individual person in the `results` table. All rows are stored as a **string**
- `personId`: A **unique** identifier for each individual person in the `results` table. All rows are stored as a **string**

`persons` is a table that has 3 columns:

- `name`: The name for each individual person in the `persons` table. All rows are stored as a **string**
- `country`: The country that a person represents in competition. All results are stored as a **string**
- `id`: A **unique** identifier for each individual person in the `persons` table. All results are stored as a **string**

Assume that personId in `results` will always have a match for id in `persons`

Write a query that outputs the number of times that a person has achieved a `result` strictly under 10 seconds (1000 centiseconds) for the **333oh eventId**. Your query should also output the following:

- The `personName` corresponding to each `result`.
- The `personId` corresponding to each `result`.
- The country that each person represents.
- The fastest 3x3x3 Cube One-Handed `result` that the corresponding competitor achieved in **seconds**, with a column name of `best_result`

- The number of times that the corresponding competitor achieved a `result` under 10 seconds.
- The output should be sorted in descending order by the number of times someone has achieved a `result` under 10 seconds.
- There should be 8 rows outputted.

Your output will look similar to the following:

```
| personName               | personId   | country     | best_result | num_times_sub_10 |
| ------------------------ | ---------- | ----------- | ----------- | ---------------- |
| Max Park                 | 2012PARK03 | USA         | 8.76        | 63               |
| Patrick Ponce            | 2012PONC02 | USA         | 8.65        | 18               |
| Luke Garrett             | 2017GARR05 | USA         | 9.18        | 12               |
| Dwyane Ramos             | 2019RAMO05 | New Zealand | 8.88        | 9                |
| Sean Patrick Villanueva  | 2017VILL41 | Philippines | 9.13        | 8                |
| Juliette Sébastien       | 2014SEBA01 | France      | 9.23        | 7                |
| Magnus Lensch            | 2019LENS01 | Germany     | 9.33        | 6                |
| Zhouheng Sun             | 2008SUNZ01 | China       | 9.81        | 4                |
```

```
SELECT _______________
             (a)
FROM _______, _______
       (b)      (c)
WHERE _______________
            (d)
GROUP BY _______
           (e)
ORDER BY _______
           (f)
LIMIT _______;
        (g)
```

i. (0.0 pt) Fill in blank (a).

```
personName, personId, country, min(result) / 100 AS best_result,
count(*) AS num_times
```

ii. (0.0 pt) Fill in blank (b).

```
results
```

iii. (0.0 pt) Fill in blank (c).

```
persons
```

iv. (0.0 pt) Fill in blank (d).

```
results.personId = persons.id AND eventId='333oh' AND average < 1000
```

v. (0.0 pt) Fill in blank (e).

```
personId
```

vi. (0.0 pt) Fill in blank (f).

```
count(*) DESC
```

vii. (0.0 pt) Fill in blank (g).

```
8
```

12. (0.0 points) Fun Question

(a) **(0.0 pt) WWYD** with the `Link.rest` of your summer. `(draw '(a picture))`. Alternatively, feel free to draw a picture about 61A!

:)

No more questions.