CS 61A Structure and Interpretation of Computer Programs
Spring 2020

Final Exam

INSTRUCTIONS

This is your exam. Complete it either at exam.cs61a.org or, if that doesn't work, on Google Forms. If either tool stops working, switch to the other one and continue taking the exam. We will merge your solutions together at the end of the exam, taking Google Form submissions in preference to submissions at exam.cs61a.org.

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.

1. (12 points) What Does This Function or Method Do?

Complete the description of each function or method so that its behavior is described correctly.

(a) (4 points) Game

```
def time(hour, minute, second):
    """Create a time value using data abstraction."""
    <implementation omitted>

def format(t):
    """Return a string that formats a time such that the hours, minutes,
    and seconds all have two digits.

    >>> format(time(3, 30, 5))
    '03:30:05'
    """
    <implementation omitted>

class Game:
    time = None
    def __init__(self, time):
        Game.time = time
    def __str__(self):
        return format(self.time)
```

Assume that `time` takes numbers that can represent the hours, minutes, and seconds of a valid time and returns a value that can be passed to `format`. Assume that `format` behaves as described in its docstring. A `Game` instance is constructed from the return value of a call to `time`.

Complete this description:

`print(Game(time(2, 10, 0)), Game(time(3, 0, 0)))` will display ...

i. (4 pt)

- ◯ ... `02:10:00 03:00:00`
- ◯ ... `03:00:00 03:00:00`
- ◯ ... `'02:10:00' '03:00:00'`
- ◯ ... `'03:00:00' '03:00:00'`
- ◯ ... `Game(time(2, 10, 0)) Game(time(3, 0, 0))`
- ◯ ... `Game(time(3, 0, 0)) Game(time(3, 0, 0))`
- ◯ ... `'Game(time(2, 10, 0))' 'Game(time(3, 0, 0))'`
- ◯ ... `'Game(time(3, 0, 0))' 'Game(time(3, 0, 0))'`

(b) (4 points) Mystery

```
def mystery(t):
    def e(r, y):
        assert type(r.label) == int
        myst = [e(b, max(y, r.label)) for b in r.branches]
        if r.label > y:
            myst.append(r.label)
        return sum(myst)
    return e(t, 0)
```

Assume that `mystery` is called on a `Tree` instance with integer labels. Assume that the sum of an empty set of labels is 0. The `Tree` class is defined on the Midterm 2 Study Guide.

An *ancestor* is a parent, or parent's parent, or parent's parent's parent, etc.

A *descendant* is a child, or child's child, or childs' child's child, etc.

Complete this description: `mystery(t)` returns the sum of ...

i. (2 pt)

◯ ... all leaf labels in `t` ...

◯ ... all labels in `t` ...

◯ ... all positive leaf labels in `t` ...

◯ ... all positive labels in `t` ...

ii. (2 pt)

◯ ... that are either the root label or larger than their parent label.

◯ ... that are either the root label or larger than all their ancestor labels.

◯ ... that are either leaf labels or larger than all their child labels.

◯ ... that are either leaf labels or larger than all their descendant labels.

(c) (4 points) Add

```
class Add:
    s = 2
    def __init__(self, s):
        assert isinstance(s, Link) or s is Link.empty
        self.t = s
        s = self.s + 1
    def this(self, v):
        def f(t):
            if t is Link.empty or t.first >= v:
                return Link(v, t)
            else:
                return Link(t.first, f(t.rest))
        for i in range(self.s):
            self.t = f(self.t)
```

Assume `Add` is called on `Link.empty` or a `Link` instance containing numbers, and the `this` method is called on a number. The `Link` class is defined on the Midterm 2 Study Guide.

Complete this description: For an instance `a = Add(s)`, the expression `a.this(v)` inserts `v` into `a.t` ...

i. (2 pt)

◯ ... once ...

◯ ... twice ...

◯ ... three times ...

◯ ... a number of times equal to two plus the number of Add instances ever constructed ...

ii. (2 pt)

◯ ... at the latest position within `a.t` where `v` is larger than all previous elements in `a.t`.

◯ ... at the earliest position within `a.t` where `v` is smaller than all subsequent elements in `a.t`.

◯ ... at the latest position within `a.t` where `v` is smaller than all previous elements in `a.t`.

◯ ... at the earliest position within `a.t` where `v` is larger than all subsequent elements in `a.t`.

2. (6 points) Multiples

Implement `multiples`, a generator function that takes positive integers `k` and `n`. It yields all positive multiples of `k` that are smaller than `n` in decreasing order.

```
def multiples(k, n):
    """Yield all positive multiples of k less than n in decreasing order.

    >>> list(multiples(10, 50))
    [40, 30, 20, 10]
    >>> list(multiples(3, 25))
    [24, 21, 18, 15, 12, 9, 6, 3]
    >>> list(multiples(3, 3))
    []
    """
    if _________:
          (a)

        for eye in _________:
                       (b)

            yield _________
                      (c)

        yield k
```

(a) (2 pt) Which expression completes blank (a)?

- ◯ `k < n`
- ◯ `k > 0`
- ◯ `n > 0`
- ◯ `k > 0 and n > 0`

(b) (2 pt) Which expression completes blank (b)?

- ◯ `multiples(k, n // 10)`
- ◯ `multiples(k, n - 1)`
- ◯ `multiples(k, n - k)`
- ◯ `multiples(k, n / k)`

(c) (2 pt) What expression completes blank (c)?

3. (16 points) Meeting in Place

Implement the methods of the `User` and `Meeting` classes as follows:

- When a `User` decides to `attend` a `Meeting` for the first time, if they are the `host` of the `Meeting`, they will be added to the end of the `joined` list; otherwise they will be added to the end of the `pending` list.
- When a `User` attempts to `attend` a meeting again, they are not added to any list. A string is returned stating that the `User` is already attending.
- A `Meeting`'s `admit` method takes a function `f` that takes a `User` and returns whether they should be admitted. The `admit` method moves all `pending` `Users` for which `f` returns a true value from the `pending` list to the end of the `joined` list.

```
class User:
    """A User can attend a Meeting.

    >>> john = User('denero@berkeley')
    >>> oski = User('oski@berkeley')
    >>> jack = User('jack@junioruniversity')
    >>> section = Meeting(john)
    >>> for x in [john, oski, jack]:
    ...     x.attend(section)
    >>> section.joined
    [User('denero@berkeley')]
    >>> section.pending
    [User('oski@berkeley'), User('jack@junioruniversity')]

    >>> oski.attend(section)
    oski@berkeley is already attending

    >>> section.admit(lambda x: 'berkeley' in x.identifier)
    >>> section.joined
    [User('denero@berkeley'), User('oski@berkeley')]
    >>> section.pending
    [User('jack@junioruniversity')]

    >>> oski.attend(section)
    oski@berkeley is already attending
    >>> User('denero@berkeley').attend(section) # A different user with the same identifier can attend
    >>> section.pending
    [User('jack@junioruniversity'), User('denero@berkeley')]
    """
    def __init__(self, identifier):

        self.identifier = identifier

    def attend(self, meeting):

        if _________ in _________:
              (a)          (b)

            print(self.identifier, 'is already attending')

        else:

            users = _________
                       (c)
```

```
            if _________:
                  (d)

                users = _________
                            (e)

            users.append(_________)
                           (f)

    def __repr__(self):

        return 'User(' + repr(self.identifier) + ')'

class Meeting:
    """A Meeting can admit pending Users."""
    def __init__(self, host):
        self.pending = []
        self.joined = []
        self.host = host

    def admit(self, f):

        for x in self.pending:

            if _________:
                  (g)

                self.joined.append(x)

        self.pending = _________
                           (h)
```

(a) **(2 pt)** What expression completes blank (a)?

(b) **(2 pt)** Which expression completes blank (b)?

- ◯ `meeting.pending + meeting.joined`
- ◯ `[meeting.pending, meeting.joined]`
- ◯ `meeting.pending.extend(meeting.joined)`
- ◯ `meeting.pending.append(meeting.joined)`

(c) **(2 pt)** What expression completes blank (c)?

(d) **(2 pt)** Which expression completes blank (d)?

- ◯ `self.identifier in meeting`
- ◯ `self in meeting`
- ◯ `self.identifier in meeting.host`
- ◯ `self in meeting.host`
- ◯ `self.identifier in meeting.joined`
- ◯ `self in meeting.joined`
- ◯ `self.identifier == meeting.host`
- ◯ `self is meeting.host`

(e) **(2 pt)** What expression completes blank (e)?

(f) **(2 pt)** What expression completes blank (f)?

(g) **(2 pt)** What expression completes blank (g)?

(h) **(2 pt)** What expression completes blank (h)?

No more questions.