CS 61A Structure and Interpretation of Computer Programs
Spring 2025

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?

(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.

1. (5.0 points) What Would Python Display?

Assume the following code has been executed.

```
nest = [[[x, y] for x in range(y, 3)] for y in range(1, 3)]

def dip(s):
    if s:
        for x in dip(s[1:]):
            yield x + 1
        yield s[0]
```

Write the output that would be displayed by printing the result of each expression. If an error occurs, write ERROR.

(a) (1.0 pt) `next(map(lambda x: 6 // (x - 2), range(5)))`

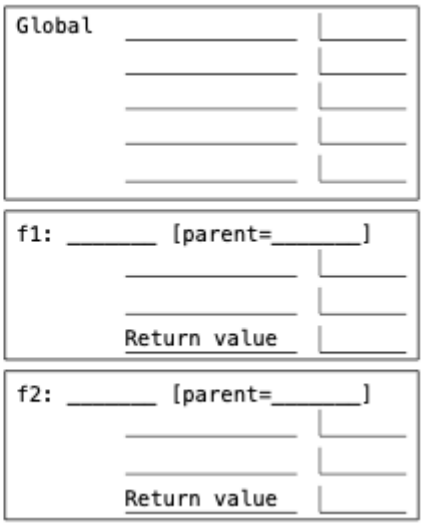
(b) (2.0 pt) `nest[1]`

(c) (2.0 pt) `list(dip([3, 4, 5]))`

2. (7.0 points) Square Bears on Stairs

Complete the environment diagram to answer the following questions. Only the questions will be scored.

```
 1: stairs = [2, [3, [4, 5]]]
 2: squares = [[9], [16], [25]]
 3: where = lambda i: squares[i]
 4:
 5: def bears(squares):
 6:     stairs[1].append(squares[1].pop())
 7:     squares[2] = 36
 8:     print(where(1))
 9:     print(stairs[1])
10:
11: bears(stairs[1] + squares[1])
```



(a) (2.0 pt) What is displayed by `print(where(1))` on line 8?

(b) (3.0 pt) What is displayed by `print(stairs[1])` on line 9?

(c) (1.0 pt) What is the order of growth of the time it takes to evaluate `skipsum(s)` in terms of the length of list `s`?

```
def skipsum(s):
    if len(s) <= 2:
        return max(s + [0])
    return max(s[0] + skipsum(s[2:]), s[1] + skipsum(s[3:]))
```

- ◯ logarithmic
- ◯ linear
- ◯ quadratic
- ◯ exponential

(d) (1.0 pt) What would be the order of growth of the time to evaluate `skipsum(s)` if `skipsum` were memoized?

- ◯ logarithmic
- ◯ linear
- ◯ quadratic
- ◯ exponential

3. (6.0 points) Swap Meet

Implement `swap`, which takes a list `s` and non-negative numbers `t` and `u`. It returns a list with the same elements as `s`, but with the elements at indices `t` and `u` swapped. Hint: `(a + b) - a == b` for any integers `a` and `b`.

```
def swap(s, t, u):
    """Return a new list like s but with the elements at positions t and u swapped.
    >>> letters = ['p', 'q', 'r', 's']
    >>> swap(letters, 0, 2)
    ['r', 'q', 'p', 's']
    >>> letters
    ['p', 'q', 'r', 's']
    """
    assert min(t, u) >= 0 and max(t, u) < len(s)
    def select(i):
        if _______:
             (a)
            return _______
                     (b)
        else:
            return _______
                     (c)
    return [select(x) for x in _______ ]
                                 (d)
```

(a) (2.0 pt) Fill in blank (a). **Select all that apply.**

- ☐ `i == t or u`
- ☐ `i == t and u`
- ☐ `i == t or i == u`
- ☐ `i == t and i == u`
- ☐ `i in [t, u]`

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

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

- ◯ `i`
- ◯ `x`
- ◯ `s[i]`
- ◯ `s[x]`

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

- ◯ `s`
- ◯ `s[t:u]`
- ◯ `range(s)`
- ◯ `range(len(s))`

4. (11.0 points) Tree Spree

(a) (5.0 points)

Implement `offspring`, which takes a `Tree` instance `t` and a value `x`. It returns a list of the labels of the nodes in `t` that are children of nodes labeled `x`. The returned list can have any order.

The `Tree` class appears on the left side of Page 2 of the Midterm 2 study guide.

```
def offspring(t, x):
    """List the labels of all nodes whose parents are labeled x.

    >>> a = Tree(5, [Tree(6, [Tree(7), Tree(8)]), Tree(9, [Tree(10)])])
    >>> sorted(offspring(a, 6))
    [7, 8]
    >>> b = Tree(3, [Tree(5, [Tree(3, [Tree(3), Tree(5)])]), Tree(7, [Tree(3)])])
    >>> sorted(offspring(b, 3))
    [3, 5, 5, 7]
    """
    result = []
    for b in t.branches:
        if _______:
             (a)
            result.append(_______)
                          (b)
        result._______
                 (c)
    return result
```

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

◯ `t.label == x`

◯ `b.label == x`

◯ `t.label == x.label`

◯ `b.label == x.label`

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

◯ `t`

◯ `t.label`

◯ `b`

◯ `b.label`

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

(b) (6.0 points)

Implement `graft`, which takes two `Tree` instances `t` and `u` and a value `x`. All labels in `t` and `u` are unique, and `x` is a label in `t`. It mutates `t`, replacing the node labeled `x` with `u`, so that `u` is now a node in `t`, and the node labeled `x` along with all of its descendants are removed from `t`.

Hint: The `index` method of a list takes a value `x` and returns the index of the first occurence of `x` in the list. `['a', 'b', 'c'].index('b')` evaluates to 1.

```
def graft(t, u, x):
    """Mutate Tree t so that the node rooted at label x is replaced by Tree u.

    >>> lemon = Tree(13, [Tree(14), Tree(15)])
    >>> lime = Tree(1, [Tree(2, [Tree(3), Tree(4, [Tree(5)])]), Tree(6, [Tree(7)])])
    >>> graft(lime, lemon, 4)
    >>> lime
    Tree(1, [Tree(2, [Tree(3), Tree(13, [Tree(14), Tree(15)])]), Tree(6, [Tree(7)])])
    >>> lime.branches[0].branches[1] is lemon
    True
    """
    s = _______
          (d)
    if x in s:
        _______
          (e)
    else:
        for b in t.branches:
            _______
              (f)
```

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

- ◯ `[t.label]`
- ◯ `t.branches`
- ◯ `[b.label for b in t.branches]`
- ◯ `[graft(b, u, x) for b in t.branches]`

ii. (4.0 pt) Fill in blank (e).

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

- ◯ `graft(b, u, x)`
- ◯ `return graft(b, u, x)`
- ◯ `b = u`
- ◯ `b.label = u.label`

5. (12.0 points) Row, Row, Row Your Boat

Each `Boat` has a number. Each `Rower` has a `name`. The `add` method of a `Boat` takes a list of `Rower` instances called `folks`. Then, for each `Rower` in `folks`, if they can `sit` in the `Boat`, they are assigned the lowest non-negative seat number that hasn't yet been assigned for that boat (starting at 0). The seat number for each `Rower` is stored in the `where` attribute of their boat: a dictionary whose keys are `Rower` instances and whose values are non-negative integers. The `sit` method of a `Rower` returns `True` if that `Rower` doesn't already have a seat, and prints a message otherwise.

A `Captain` is a `Rower` that won't sit in a boat with another `Captain`. The `sit` method of the `Captain` class first checks for another captain in the boat and prints a message and returns `None` if there is one, then checks that the `Captain` doesn't already have a seat in a boat (and prints a message and returns `None` if they do), and finally claims the boat so that no other captain will sit there (and returns `True`).

```
class Boat:
    """A boat full of rowers.
    >>> bella, finn, hank = Rower('Bella'), Captain('Finn'), Captain('Hank')
    >>> Boat(5).add([Rower('Ace'), bella, Rower('Charlie'), finn, Rower('Ginger')])
    >>> Boat(7).add([Rower('Daisy'), finn, hank, bella, Captain('Ellie')])
    Finn in Boat 5 Seat 3 already has a seat!
    Bella in Boat 5 Seat 1 already has a seat!
    Ellie won't sit. Hank is captain of Boat 7!
    >>> print(hank)
    Hank in Boat 7 Seat 1
    """
    def __init__(self, number):
        self.number = number  # The number of the boat
        self.where = {}       # A dict from Rowers to seat numbers
    def add(self, folks):
        for p in folks:
            if p.sit( _______ ):
                        (a)
                _______       # Assign them the lowest available seat number
class Rower:      (b)
    """A Rower in a Boat."""
    def __init__(self, name):
        self.name = name
        self.boat = None
    def sit(self, boat):
        if self.boat is None:
            self.boat = boat
            return True
        print(self, 'already has a seat!')
    def __str__(self):
        return f'{self.name} in Boat {self.boat.number} Seat { _______ }'
class Captain(Rower):                                          (c)
    """A captain will not sit in a boat that already has a captain."""
    claimed = {}
    def sit(self, boat):
        if boat not in Captain.claimed:
            if _______ :
                 (d)
                _______  # Claim the boat
                  (e)
                return True
        else:
            print(f"{self.name} won't sit. { _______ } is captain of Boat {boat.number}!")
```

```
(f)
```

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

- ◯ `self`
- ◯ `self.number`
- ◯ `Boat`
- ◯ `Boat.number`

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

(c) (3.0 pt) Fill in blank (c), which evaluates to the seat number assigned to the Rower when they sat.

(d) (2.0 pt) Fill in blank (d). **Select all that apply.**

- ☐ `self.boat is not None`
- ☐ `super().sit(boat)`
- ☐ `super(self).sit(boat)`
- ☐ `super().sit(self, boat)`
- ☐ `Rower.sit(boat)`
- ☐ `self.Rower.sit(boat)`
- ☐ `Rower.sit(self, boat)`

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

- ◯ `claimed = True`
- ◯ `self.claimed = True`
- ◯ `claimed[boat] = self`
- ◯ `self.claimed[boat] = self`
- ◯ `boat.captain = self`
- ◯ `boat.captain = True`

(f) (2.0 pt) Fill in blank (f), which evalautes to the name of the captain that claimed this boat.

6. (18.0 points) Arithmetic

(a) (8.0 points)

Implement `can_equal`, which takes a string `digits` containing a sequence of digits from 1 to 9 (no zeros) and an integer `n`. It returns `True` if there is a way to add zero or more `+` and `-` symbols to `digits` to form an expression that evaluates to `n`.

Hint: The built-in `int` can be called on a string to convert it to an integer. For example, `int('-42')` returns -42.

```
def can_equal(digits, n):
    """Whether adding + and - symbols to digits can form an expression that evaluates to n.

    >>> can_equal('45332', 14)  # 45-33+2 = 14
    True
    >>> can_equal('45332', 527) # -4+533-2 = 527
    True
    >>> can_equal('45332', 38)  # 4+5-3+32 = 38
    True
    >>> can_equal('45332', 287) # -45+332 = 287
    True
    >>> can_equal('45332', -287) # 45-332 = -287
    True
    >>> can_equal('45332', 45)  # Impossible
    False
    >>> can_equal('45332', 39)  # Impossible
    False
    """
    if _______ :
         (a)

        return True

    for k in range(1, _______ ):
                         (b)

        if _______ ([can_equal( _______ , n - sign * _______ ) for sign in [1, -1]]):
             (c)                   (d)                   (e)

            return True

    return _______
             (f)
```

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

◯ `digits == ''`

◯ `n == 0`

◯ `digits == '' and n == 0`

◯ `digits == '' or n == 0`

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

- ◯ `10`
- ◯ `int(digits)`
- ◯ `int(digits) + 1`
- ◯ `len(digits)`
- ◯ `len(digits) + 1`

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

- ◯ `any`
- ◯ `all`
- ◯ `list`
- ◯ `min`
- ◯ `iter`

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

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

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

- ◯ `True`
- ◯ `False`
- ◯ `n == 0`
- ◯ `n in digits`
- ◯ `str(n) in digits`

(b) (4.0 points)

The `Link` class appears on the left side of Page 2 of the Midterm 2 study guide.

i. (2.0 pt) Fill in blank (g) to implement `join_link`, a function that takes a linked list of strings `s`. It returns a string that contains all of the characters in all of the strings in `s` in order.

```
def join_link(s):
    """Return a string that concatenates all of the strings in linked list s.

    >>> join_link(Link('cs', Link('6', Link('1a'))))
    'cs61a'
    """
    if s is Link.empty:
        return ''
    return _______
             (g)
```

◯ `s + str(s.rest)`

◯ `s + join_link(s.rest)`

◯ `s + join_link(s.rest.first)`

◯ `s.first + str(s.rest)`

◯ `s.first + join_link(s.rest)`

◯ `s.first + join_link(s.rest.first)`

ii. (2.0 pt) Fill in blank (h) to implement `to_link`, a function that takes a finite iterator `t`. It returns a linked list containing all of the values that `t` iterates over.

```
def to_link(t):
    """Return a linked list containing the values in iterator t.

    >>> to_link(iter([3, 5, 7, 9]))
    Link(3, Link(5, Link(7, Link(9))))
    """
    try:
        return _______
                 (h)
    except StopIteration:
        return Link.empty
```

◯ `Link(t, to_link(t))`

◯ `Link(t, to_link(t.rest))`

◯ `Link(t, to_link(t).rest)`

◯ `Link(t, to_link(next(t)))`

◯ `Link(next(t), to_link(t))`

◯ `Link(next(t), to_link(t.rest))`

◯ `Link(next(t), to_link(t).rest)`

◯ `Link(next(t), to_link(next(t)))`

(c) (6.0 points)

Implement `terms`, a generator function that takes a string `digits` containing digits from 1 to 9 and an integer `n` for which `can_equal(digits, n)` returns `True`. It yields integers whose digits are the contents of `digits` and that sum to `n`. If there is more than one such sequence of numbers, yield any one of them. You may call `can_equal`, `join_link`, and `to_link`.

```
def terms(digits, n):
    """Yield the numbers in an expression that demonstrates can_equal(digits, n) is True.

    >>> list(terms('45332', 38))   # 4+5-3+32 = 38
    [4, 5, -3, 32]
    >>> list(terms('45332', 527))  # -4+533-2 = 527
    [-4, 533, -2]
    """
    assert can_equal(digits, n)
    digits = to_link(iter(digits))  # Create a linked list over one-digit strings
    first = 0
    while digits:
        first = _______
                  (i)
        rest = _______
                 (j)
        if can_equal(rest, n - first):
            yield first
            first, n = 0, n - first
        elif can_equal(rest, n + first):
            _______
              (k)
            first, n = 0, n + first
        digits = digits.rest
```

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

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

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

- ◯ `n = -n`
- ◯ `yield first`
- ◯ `yield -first`
- ◯ `yield n-first`
- ◯ `yield from terms(digits, -n)`

7. (10.0 points) Finally Some Scheme

(a) (6.0 points)

Implement `except-last`, a procedure that takes a list `s` and returns a list with all of the elements of `s` except the last one. If there is no last element because `s` is empty, return `nil`.

```
;;; Return all but the last element of s in a list.
;;;
;;; scm> (except-last nil)
;;; ()
;;; scm> (except-last '(2))
;;; ()
;;; scm> (except-last '(2 3 4 5))
;;; (2 3 4)
;;; scm> (except-last '(2 (3 3) 4 (5 6)))
;;; (2 (3 3) 4)

(define (except-last s)
  (if _______ nil
      (a)
    (cons (car s) _______ )))
                  (b)
```

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

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

- ◯ `s`
- ◯ `(cdr s)`
- ◯ `(cdr (cdr s))`
- ◯ `(except-last s)`
- ◯ `(except-last (cdr s))`
- ◯ `(except-last (cdr (cdr s)))`

iii. (3.0 pt) Fill in blank (c) below to implement `except-last-k`, which takes a non-negative integer `k` and a list `s`. It returns a list with all of the elements of `s` except the last `k` elements. If there are `k` or fewer elements, return `nil`. You may call `except-last`.

```
(define (except-last-k k s)
  (if (= k 0) s _______ ))
                 (c)
```

(b) (4.0 points)

Assume the following code had been evaluated and that `except-last` is implemented correctly.

```
(define-macro (mystery expr other)
  (append (except-last expr) (list other)))
```

What would Scheme display as the value of the following expression? Write `ERROR` if an error occurs.

i. (2.0 pt) `(cons 2 (mystery (cons 3 nil) '(4)))`

- ◯ `(2 4)`
- ◯ `(2 (4))`
- ◯ `(2 3 4)`
- ◯ `(2 3 (4))`
- ◯ `(2 (3 4))`
- ◯ `(2 (3 (4)))`
- ◯ `ERROR`

ii. (2.0 pt) `(mystery (* (+ 2 3) (+ 4 (/ 2 0)))  (- 5 2) )`

- ◯ `3`
- ◯ `5`
- ◯ `8`
- ◯ `15`
- ◯ `20`
- ◯ `35`
- ◯ `ERROR`

8. (6.0 points) WHERE Brainrot='Italian'

The `who` table has one row per character with its `name` (a unique string), `size` (a number), and `kind` (a string). The `kinds` table has one row per kind that shows whether it's `living` (`'yes'` or `'no'`) and `what` kind it is (a unique string).

who:

name	size	kind
tralalero tralala	800.0	reptile
bombardiro crocodillo	600.0	reptile
lirili larila	1500.0	mammal
capuchino assassino	60.0	object
brr brr patapim	80.0	mammal
chimpanzini bananini	90.0	mammal
tung tung tung sahur	200.0	object

kinds:

living	what
yes	mammal
yes	reptile
no	object

Result Part (a):

total
3070.0

Result Part (b):

name	number
bombardiro crocodillo	1
brr brr patapim	2
capuchino assassino	2
chimpanzini bananini	2
lirili larila	1
tralalero tralala	2

(a) (3.0 pt) Fill in blank (a) to create a one-row, one-column table containing the `total` size of all characters that have a `living` kind.

```
SELECT SUM(size) AS total FROM _______ ;
                                (a)
```

- ◯ `who JOIN kinds ON kind=what WHERE kind=living`
- ◯ `who JOIN kinds ON kind=what WHERE kind='living'`
- ◯ `who JOIN kinds ON kind=what WHERE living=yes`
- ◯ `who JOIN kinds ON kind=what WHERE living='yes'`
- ◯ `who JOIN kinds ON kind=kinds WHERE kind=living`
- ◯ `who JOIN kinds ON kind=kinds WHERE kind='living'`
- ◯ `who JOIN kinds ON kind=kinds WHERE living=yes`
- ◯ `who JOIN kinds ON kind=kinds WHERE living='yes'`

(b) (3.0 pt) Two characters can battle if the smaller one is at least half the size of the larger one (or they are the same size). Fill in blank (b) to create a table with one row per character that has two columns: the character's `name` and the `number` of other characters it can battle. For example, `lirili larila` can only battle `tralalero tralala` because all other characters are too small. Do not include rows for characters that cannot battle any other characters. **Select all that apply.**

```
SELECT a.name,COUNT(*) FROM who AS a, who AS b WHERE a.name!=b.name AND _______ GROUP BY a.name;
                                                                        (b)
```

- ☐ `a.size >= b.size * 0.5`
- ☐ `b.size >= a.size * 0.5`
- ☐ `a.size >= b.size * 0.5 OR b.size >= a.size * 0.5`
- ☐ `a.size >= b.size * 0.5 AND b.size >= a.size * 0.5`
- ☐ `MIN(a.size, b.size) >= b.size * 0.5`
- ☐ `a.size >= MAX(a.size, b.size) * 0.5`
- ☐ `MIN(a.size, b.size) >= MAX(a.size, b.size) * 0.5`
- ☐ `MIN(a.size / b.size, b.size / a.size) >= 0.5`
- ☐ `MAX(a.size / b.size, b.size / a.size) >= 0.5`

9. (0.0 points) A+

These two A+ questions are not worth any points. They can only affect your course grade if you have a high A and might receive an A+. Finish the rest of the exam first!

(a) (0.0 pt) Fill in the blank to implement `swap`, which takes a list `s` and non-negative numbers `t` and `u`. It returns a list with the same elements as `s`, but with the elements at indices `t` and `u` swapped. Do not write `if`, `and`, `or`, or `lambda`. You may not call functions from elsewhere on the exam. **Hint:** Try using a dictionary.

```
def swap(s, t, u):
    """Return a new list like s but with the elements at positions t and u swapped.
    >>> letters = ['p', 'q', 'r', 's']
    >>> swap(letters, 0, 2)
    ['r', 'q', 'p', 's']
    >>> letters
    ['p', 'q', 'r', 's']
    """
    return [ _______ for x in range(len(s)) ]
```

(b) (0.0 pt) Fill in the blank to implement `equal`, which takes a string `digits` containing digits from 1 to 9 and an integer `n` for which `can_equal(digits, n)` returns `True`. It returns a string `s` containing all of `digits` and zero or more `+` or `-` symbols such that `eval(s) == n`. You may call `eval` and `can_equal`. If there is more than one possible return value, return any of them. **Important:** For credit, your implementation must ensure that the generator returned by `f` only yields as many values as necessary for correctness (and no more).

```
def equal(digits, n):
    """Return a string containing digits and + and - that evaluates to n.

    >>> equal('45332', 527)
    '-4+533-2'
    >>> equal('45332', 38)
    '4+5-3+32'
    """
    assert can_equal(digits, n)
    def f(digits):
        yield digits
        yield '-' + digits
        for k in range(1, len(digits)):
            for rest in f(digits[k:]):
                yield digits[:k] + '+' + rest
                yield digits[:k] + '-' + rest
                yield '-' + digits[:k] + '+' + rest
                yield '-' + digits[:k] + '-' + rest
    return _______
```

No more questions.