CS 61A Structure and Interpretation of Computer Programs
Summer 2025

Midterm 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?

(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?

```
1: a = [[13], 8]
2: b = a + [[2]]
3: b[1] -= 2
4: a[1] -= b[1]
5:
6: print(a)
7:
8: a[0] += [7]
9: b[b[2][0]] += [a.pop(1)]
10:
11: print(b)
12:
13: c = b[-1]
14: c.append(a[0])
15: c.append(b[0].extend([c[0]]))
16:
17: print(c)
```

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.

Hint: Draw it out!

(a) (1.0 pt) What is displayed by the call to `print` in line 6?

```
[[13], 2]
```

(b) (2.0 pt) What is displayed by the call to `print` in line 11?

- ○ `[[13], 2, [2]]`
- ● `[[13, 7], 6, [2, 2]]`
- ○ `[[13], 6, [2]]`
- ○ `[[13, 7], 2, [2]]`
- ○ `[[13, 7], 6, [4]]`
- ○ `[[13, 7], 6, [2, [2]]]`
- ○ `[[13], 6, [2, 2]]`

(c) (2.0 pt) What is displayed by the call to `print` in line 17?

- ○ `[2, 2, [13, 7], None]`
- ○ `[2, 2, [13], [2, 2]]`
- ○ `[2, 2, [13, 2], [13]]`
- ○ `[2, 2, [13], 2]`
- ● `[2, 2, [13, 7, 2], None]`
- ○ `[2, 2, [13, 7, 2], [None]]`

2. (10.0 points) Surf's Up!

Answer the questions about the code below. Use the free space or scratch paper to draw the diagram to help you answer the questions, however any drawn diagrams will not be graded.

```
1: def wave(twin):
2:     n = 3
3:     def twin():
4:         print(wave, n)
5:         return print('Shred') or n
6:         print('Wipeout')
7:     return twin()
8:
9: def twin():
10:     def twin(wave):
11:         while wave(twin):
12:             return print('Riptide')
13:         print('Tide')
14:     return twin
15:
16: twin = twin()(lambda twin: print('Twin Wave') or wave(twin))
```

If an error occurs, write `"Error"`. If a function is printed, write `"Function"`.

(a) (4.0 pt) What is printed after evaluating the above code?

Twin Wave
Function 3
Shred
Riptide

(b) (1.0 pt) How many frames are opened (not including the Global Frame)?

- ○ `3`
- ○ `4`
- ● `5`
- ○ `6`

(c) (2.0 pt) What is the parent frame of `lambda twin:` in line 16?

- ● `Global`
- ○ `f1`
- ○ `f2`
- ○ `None`

(d) (1.0 pt) How many calls are made to the function `wave` defined in line 1?

- ○ `0`
- ● `1`
- ○ `2`
- ○ `3`
- ○ `None`

(e) (2.0 pt) What is `twin` assigned to in the Global Frame?

`None`

3. (17.0 points) Python in python?

Definition. A **python** is a sequence of digits where the following are true:

- All digits are equal to some digit `d`
- All digits are two places apart (difference between indices equals 2)

In other words, a python is a sequence of digits where a digit `d` occurs in every other digit place, and the length of the python is the number of times `d` appears consecutively.

(a) (9.0 points)

Implement `python`, which takes in a number `n` and a digit `d`, and returns the length of the longest **python** in `n` where all digits equal `d`

Assume `n > 0`

```
def python(n, d):
    """
    Return the length of the longest python of digit d in integer n.
    >>> python(2125262, 2)      # positions of 2s: [0,2,4,6] -> all 2 apart -> length 4
    4
    >>> python(2020211202, 2)   # positions of 2s: [0,2,4,7,9] -> longest python is 3 (at 0,2,4)
    3
    >>> python(20211120202, 2)  # positions of 2s: [0,2,6,8,10] -> longest python is 3 (at 6,8,10)
    3
    >>> python(1010, 0)
    2
    >>> python(123456, 7)       # no 7 exists within n
    0
    """
    lengths = []
    _______:
      (a)
        copied_n = _______
                     (b)
        length = 0
        _______ and copied_n > 0:
          (c)
            length += 1
            _______
              (d)
        lengths._______
                  (e)
        _______
          (f)
    return max(lengths)
```

i. (1.0 pt) Fill in blank (a). Select all that apply.

- ☐ `while n // 10 < 0:`
- ■ `while n > 0:`
- ☐ `while n >= 10:`
- ☐ `while d == n:`

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

```
n
```

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

```
while copied_n % 10 == d
```

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

- ◯ `n = n // 100`
- ◯ `n = copied_n // 100`
- ● `copied_n = copied_n // 100`
- ◯ `copied_n = n // 100`

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

- ■ `insert(0,length)`
- ☐ `append([length])`
- ☐ `extend(list(length))`
- ■ `extend([length])`

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

```
n = n // 10
```

(b) (8.0 points)

Implement `longest_python`, which takes in a number n and returns the digit and length for the digit with the longest **python** in n. This function explicity returns the digit, followed by its corresponding length.

If there is a tie for longest python, return the numerically greater digit.

Assume `python` from part (3a) is implemented correctly.

```
def longest_python(n):
    """
    Return the digit and length of the longest python for any digit d
    If there is a tie, return the larger digit.
    >>> longest_python(2324252)   # 2 has len 4, others have len 1
    (2, 4)
    >>> longest_python(1212121)   # 1 has len 4, 2 has len 3,
    (1, 4)
    >>> longest_python(56565656)  # 5 has len 4, 6 has len 4, 6 > 5
    (6, 4)
    >>> longest_python(67676565)  # 6 has len 4, 7 has len 2, 5 has len 2
    (6, 4)
    >>> longest_python(12345)     # all have len 1, return largest digit
    (5, 1)
    """
    d = {}
    for x in _______:
                (g)
        _______
          (h)
    key = _______
            (i)
    return d[key], _______
                     (j)
```

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

- ○ `n`
- ○ `range(n)`
- ○ `range(len(n))`
- ● `range(10)`

ii. (3.0 pt) Fill in blank (h). You may not use `for`, `in`, `if`, or `lambda` in your solution.

```
d[python(n,x)] = x
```

iii. (2.0 pt) Fill in blank (i). You may not use `for`, `in`, `if`, or `lambda` in your solution.

```
max(d) OR max(d.keys())
```

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

`key` **OR** `max(d)`

4. (17.0 points) Valid Number

Definition. A number is valid if it satisfies either of the following critera:

- It has 2 or fewer digits, or
- Excluding the last two digits, for every digit, there is another digit 1 or 2 places to the right with the same **parity** (even or odd)

For example, 3412 is valid as 3 and 1 have the same parity, and 4 and 2 have the same parity

(a) (7.0 points)

Implement `is_valid`, which takes in a number `n` and returns True if `n` is valid, `False` otherwise.

Assume `n > 0`

```
def is_valid(n):
    """
    >>> is_valid(3412)   # 1 is two digits to the right of 3, 2 is two digits to the right of 4.
    True
    >>> is_valid(43)     # 2 or fewer digits
    True
    >>> is_valid(3443)   # No odd number within two digits of leftmost 3.
    False
    >>> is_valid(213)    # No even number to the right of 2.
    False
    """
    def helper(last, second, n):
        if _______:
             (a)
            return True
        curr = n % 10
        if _______ and _______:
             (b)          (c)
            return False
        return helper(_______)
                       (d)

    return helper(n % 10, n // 10 % 10, n // 100)
```

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

- (●) `n == 0`
- ( ) `n >= 0`
- ( ) `n < 0`
- ( ) `n == 1`

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

- ( ) `curr % 2 == 0`
- ( ) `last % 2 == 0`
- (●) `curr % 2 != last % 2`
- ( ) `(curr + last) % 2 == 0`

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

```
curr % 2 != second % 2 OR (curr + second) % 2 == 1
```

iv. (3.0 pt) Fill in blank (d).

```
second, curr, n // 10 OR second, n % 10, n // 10
```

(b) (10.0 points)

Implement `valid_subseq`, which takes in some number `n`, and returns the number of subsequences of n that are valid numbers.

A subsequence of a number is a combination of digits whose relative order remains the same as in the original number. In other words, a subsequence of a number is the number with some (or no) digits removed.

For example, 213 has the following subsequences: 2, 1, 3, 21, 23, 13, 213

Assume `is_valid` from part (4a) is implemented correctly.

Additionally, we have provided you with a function `reverse`, which takes a number and reverses its digits.

Hint True + True == 2

```
def valid_subseq(n):
    """
    Return the number of valid subsequences of n.
    >>> valid_subseq(12)    # possible subseqs: 1, 2, 12 - all are valid
    3
    >>> valid_subseq(123)
    7
    >>> valid_subseq(213)   # possible subseqs: 2, 1, 3, 21, 23, 13, 213 - 6 of which are valid.
    6
    >>> valid_subseq(3412)
    14
    """

    def reverse(x):
        """
        This function gives the reverse of a number
        >>> reverse(3412)
        2143
        """
        # IMPLEMENTATION NOT SHOWN

    def helper(num, rest):
        if _______ and rest == 0:
             (e)
            return _______
                      (f)
        elif _______:
               (g)
            return _______
                      (h)
        return helper(_______, _______) + helper(_______,_______)
                        (i)      (j)               (k)     (l)
    return helper(0, reverse(n))
```

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

- ◯ `n >= 0`
- ◯ `num > 0`
- ● `num == 0`
- ◯ `n < 0`

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

- [x] `0`
- [ ] `1`
- [x] `num`
- [x] `rest`

The intended solution was `0`. However, since the `if` statement ensures that both `num` and `rest` are `0`, they are also valid options.

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

```
rest == 0
```

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

- [ ] `0`
- [ ] `1`
- [ ] `is_valid(rest)`
- [x] `is_valid(num)`

v. **(2.0 pt)** Fill in blank (i).

```
num * 10 + rest % 10
```

vi. **(1.0 pt)** Fill in blank (j).

```
rest // 10
```

vii. **(1.0 pt)** Fill in blank (k).

- [ ] `n`
- [x] `num`
- [ ] `rest`
- [ ] `num // 10`

viii. **(1.0 pt)** Fill in blank (l).

```
rest // 10
```

5. (15.0 points) Family Trees

(a) (7.0 points)

Implement `descendants`, which takes in a tree `t` and depth `d`, and returns a list of labels for all children at least depth `d` away from the root.

```
def descendants(t, d):
    """Return a list of all labels of descendants at least d levels away from the root node.

    >>> t1 = tree(1, [tree(2, [tree(3), tree(4)]), tree(5, [tree(6), tree(7)])])
    >>> print(t1)
    1
        2
            3
            4
        5
            6
            7
    >>> descendants(t1, 1)
    [2, 3, 4, 5, 6, 7]
    >>> descendants(t1, 2)
    [3, 4, 6, 7]
    >>> descendants(t1, 3)
    []
    """
    lst = []
    if _______:
         (a)

        _______
          (b)
    for b in branches(t):
        lst = _______
                (c)
    return _______
             (d)
```

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

- ○ `d == 0`
- ○ `d >= 0`
- ○ `d < 0`
- ● `d <= 0`

ii. (2.0 pt) Fill in blank (b). Select all that apply.

- ☐ `lst + label(t)`
- ☐ `lst.append(t)`
- ☐ `lst += [t]`
- ■ `lst.extend([label(t)])`

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

```
lst + descendants(b, d-1)
```

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

```
lst
```

(b) (8.0 points)

Definition. A tree is an ancestor if all paths from the root node to a depth `d` have strictly decreasing labels.

Implement `is_ancestor`, which takes in a tree `t` and a depth `d` and returns `True` if the tree is an ancestor, `False` otherwise.

```
def is_ancestor(t, d):
    """
    Return True if t is an ancestor with respect to a depth d, False otherwise.
    >>> t1 = tree(2, [tree(1)])
    >>> print_tree(t1)
    2
        1

    >>> is_ancestor(t1, 0)
    True
    >>> is_ancestor(t1, 1)    # 2 -> 1: strictly decreasing
    True

    >>> t2 = tree(4, [t1, tree(3, [tree(5), tree(3)])])
    >>> print_tree(t2)
    4
        2
            1
        3
            5
            3
    >>> is_ancestor(t2, 0)
    True     # 4
    >>> is_ancestor(t2, 1)    # 4 -> 2 and 4 -> 3 are both strictly decreasing
    True
    >>> is_ancestor(t2, 2)    # 4 -> 3 -> 5 and 4 -> 3 -> 3 are not strictly decreasing
    False
    """
    if _______:
         (e)
        return True
    for b in branches(t):
        if _______ _______ _______:
             (f)     (g)     (h)
            return _______
                      (i)
    return _______
             (j)
```

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

- (●) `d == 0`
- ( ) `d < 0`
- ( ) `d == 1`
- ( ) `d >= 0`

ii. **(2.0 pt)** Fill in blank (f). You may not use `and`, `or`, or `not`.

```
label(b) >= label(t)
```

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

- ◯ `and`
- ● `or`
- ◯ `not`
- ◯ `==`

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

```
not is_ancestor(b, d-1)
```

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

- ◯ `not b`
- ◯ `True`
- ◯ `is_ancestor(b, d-1)`
- ● `False`

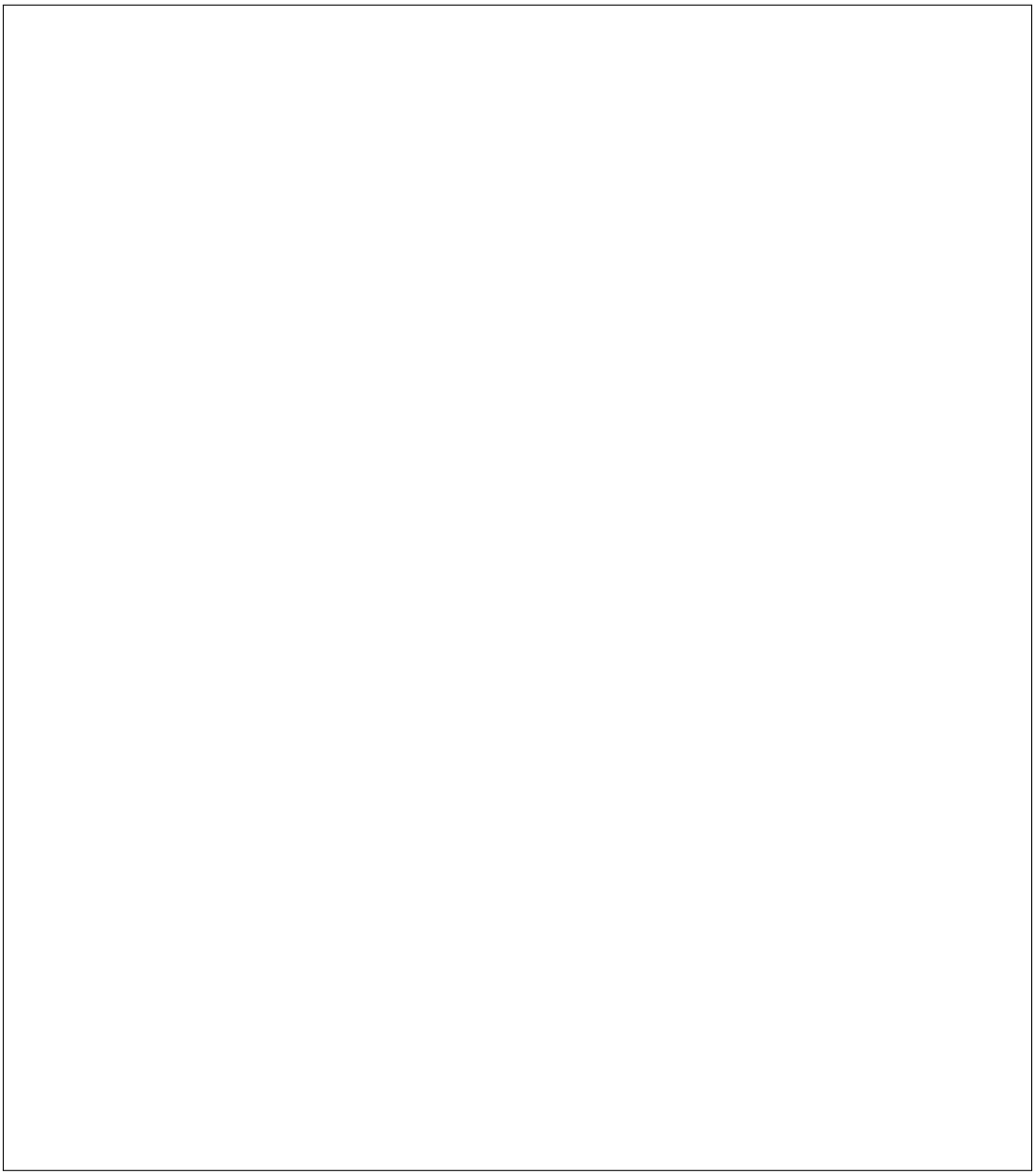
vi. **(1.0 pt)** Fill in blank (j).

```
True
```

6. (0.0 points) Just for Fun

This is not for points and will not be graded.

(a) Optional: Draw how this exam made you feel!



No more questions.