CS 61A
Fall 2023

Midterm Review

Discussion 9: October 25, 2023

Getting Started

Someone from your group should join Discord.

To get help from a TA, send a message to the `discuss-queue` channel with the @discuss tag and your discussion group number.

If you have only 1 or 2 people in your group, you can join the other group in the room with you.

Pro tip: Any of you can type a question into your group's Discord channel's text chat with the `@discuss` tag, and a member of the course staff will respond.

Q1: Ice Breaker

Everybody say your name, and then figure out who most recently pet a dog. They're your group's facilitator for the day. (Feel free to share dog photos. Even cat photos are acceptable.)

Reminder: To pet Prof. DeNero's dog, come to office hours on Memorial Glade next Monday 10/30 9am-10am.

Facilitator: If you ever don't want to be facilitator anymore, you can ask for a new facilitator. Just say, "Would someone else be facilitator for a while?" You can even do that now.

Objects

Q2: Lock

A locker is locked by placing a lock on it, and it can only be opened using a key for that lock. Two different keys with the same code (called grooves in real life) will open the same locks.

Fill in the blank in the doctest with an expression that **doesn't contain any numbers** so that the doctests are correct **without changing the implementation** of any of the classes. But first…

Facilitator: Give everyone a minute to read the code, then ask for volunteers to describe to the group what each of the following bits of code does. (E.g., ask, "Who wants to explain the first one?") Try to have different people explain each one. If the group isn't sure about one or more of these lines, click the "Hint: Answers" below.

1. `box.ie = self` in the `__init__` method of `Lock`.
2. `code = self.code` in the `__init__` method of `Key`.
3. The long line starting with `type(self).code = ...`
4. `self.code = code` in the `__init__` method of `Key`.
5. The expression `self.ie.stick` in the `open` method of `Locker`.

1. `box.ie = self` assigns a `Lock` object (called `self`) to the `ie` attribute of a `Locker` object called `box`. 2. `code = self.code` assigns the class attrbute `Key.code` to the local name `code`. 3. `type(self).code = ...` changes the class attribute `Key.code`. The last digit becomes the first digit. 4. `self.code = code` assigns an instance attribute for the `Key` object called `self` to `code`, which is the old `Key.code` value before its digits were rearranged. 5. In `open`, `self` is a `Locker` object, `self.ie` is the `Lock` object most recently placed on that locker, and `self.ie.stick` is the code for the key that opens the lock.

When that's finished, ask if anyone has ideas for how to make a key with the right code to open `x`. Once everyone knows the idea, it's time to work on code.

Did you know? To "pop a lock" means to open it without a key.

```
def open_locker(x):
    """Lock, and then open, a locker.

    >>> open_locker(Locker())
    Opened!
    """
    Lock(x).make_key()
    x.open(_______)

class Lock:
    "A lock on a locker that opens with a key with the right code."
    def __init__(self, box):
        box.ie = self

    def make_key(self):
        self.stick = Key().code

class Key:
    code = 5678
    def __init__(self):
        code = self.code
        type(self).code = code // 10 + (code % 10) * 1000
        self.code = code

class Locker:
    def open(self, key):
        if key.code == self.ie.stick:
            print('Opened!')
```

The first four `Key` objects created with `Key()` have codes 5678, 8567, 7856, and 6785. What code will the fifth `Key()` have?

Try creating a list of key objects and using `pop` to return the last of them.

Facilitator: Ask someone to post your solution in your group's Discord channel's text chat.

Recursion

Q3: Paths List

(Adapted from Fall 2013) Implement `paths`, a function that takes two integers `x` and `y` that are both greater than 1. It returns a list of every list that starts with `x`, ends with `y`, and has elements after the first that are either one more or double the previous element.

Facilitator: Say, "Look! All the lists start with x, but the second element is either `x+1` or `2*x`." Then your group should talk about what the recursive case needs to do before writing any code.

```
def paths(x, y):
    """Return a list of ways to reach y from x by repeated incrementing or doubling.

    >>> paths(3, 5)
    [[3, 4, 5]]
    >>> sorted(paths(3, 6))
    [[3, 4, 5, 6], [3, 6]]
    >>> sorted(paths(3, 9))
    [[3, 4, 5, 6, 7, 8, 9], [3, 4, 8, 9], [3, 6, 7, 8, 9]]
    >>> paths(3, 3)  # A path can have just one element if x and y are equal.
    [[3]]
    >>> paths(5, 3)  # There is no way to start with 5 and ends with 3.
    []
    """
    if _______:
        return _______
    elif _______:
        return _______
    else:
        return [_______ for p in _______ + _______]
```

The last two doctests show examples of the base cases: when `x == y` and when `x > y`.

A recursive call to `paths(2*x, y)` will return a list of paths that start with `2*x`. A second recursive call is needed to return a list of paths that start with `x+1`.

Facilitator: Once your group has an answer, please ask, “Does anyone have questions about how this works?”

Presentation Time: Describe how the `paths` implementation would have to change if every double had to come after an increment (so `[2, 3, 6, 7, 14, 15]` would be ok, but neither `[2, 4]` nor `[2, 3, 4, 8, 16, 17]` would be allowed). You don’t have to write the code for this change; just describe what you would do. Once your group agrees on an answer (or wants help), send a message to the `#discuss-queue` channel with the `@discuss` tag, your discussion group number, and the message “Stay on the path!” and a member of the course staff will join your voice channel to hear your explanation and give feedback.

Linked Lists

Q4: Insert

Implement `insert`, which takes a non-empty linked list `s`, a `value`, and a non-negative integer `index` that is less than or equal to the length of `s`. It mutates `s` by inserting the `value` into `s` at the given `index`. The index of `s.first` is 0.

Facilitator: Give everyone a minute to read the code and doctests, then take a vote to decide whether to use recursion (calling `insert`) or iteration (a `while` loop). Either way works (and the solutions are about the same complexity/difficulty).

```
def insert(s, value, index):
    """Insert value into non-empty linked list s at the given index.

    >>> s = Link(1, Link(2, Link(3)))
    >>> print(s)
    <1 2 3>
    >>> insert(s, 4, 0)  # Insert 4 at the start
    >>> print(s)
    <4 1 2 3>
    >>> insert(s, 5, 2)  # Insert 5 in the middle
    >>> insert(s, 6, 5)  # Insert 6 at the end
    >>> print(s)
    <4 1 5 2 3 6>
    """
    assert 0 <= index and index <= length(s)
    "*** YOUR CODE HERE ***"















def length(s):
    "Return the length of linked list s."
    if s is Link.empty:
        return 0
    else:
        return 1 + length(s.rest)
```

There are two ways to insert: - At `index==0`, create a new link for the original `s.first` just after `s`, then change `s.first` to `value`. - At `index==1`, create a new link for `value` just after `s`.

Use both ways in order to be able to insert at both the beginning and the end.

Insert `value` into `s.rest` at `index-1`.

Facilitator: If you have more than 15 minutes left after your group solves this problem, say: "How would we write it the other way?" Then, have a discussion about how you might solve it using iteration instead of recursion, or vis versa. If you're not sure how to change from one version to the other, ask `@discuss` for help on Discord.

Document the Occasion

Please all fill out the attendance form (one submission per person per week).

Here are some tips for the exam: 1. Get some rest tonight! 2. The doctest examples can help you figure out solutions. It's often helpful to consider example values. 4. Once you finish a question, check your work using the doctests. 5. Call functions on the right type(s) of value(s) and return the right type of value. 6. Make sure you know the type of value that results from evaluating each expression you write.

Good luck!