CS 61A
Spring 2021

Control and Environments

Lost 1: January 29, 2021

1 Introduction

1.1 What is a LOST section?

LOST sections are optional, slower-paced sections which will be held weekly to complement orientations and tutorials. LOST sections will not necessarily consist of *easier* problems; rather, we will simply spend more *time* discussing and reviewing them. The point of LOST sections is to help students master the material, which cannot happen if we focus exclusively on easier problems. We will start simple but build up in complexity each section.

1.2 Who should attend a LOST section?

LOST sections are designed for students who feel 61A is moving too quickly for them and would like extra practice with the material.

1.3 What will a LOST section consist of?

The design of these sections is experimental, so they may change from week-to-week based on student feedback. The current plan is to begin with a mini-lecture of relevant topics, go through some basic examples, complete 1-2 review problems from orientation/tutorial, complete 1-2 new problems, and complete 1 exam-level problem. **Note: LOST sections will consist mostly of the TA lecturing and going through practice problems. Students will also have time to work on problems alone before they are reviewed by the TA. Because the size of these sections is expected to be large, LOST sections are not an ideal environment for collaborative work.**

2 Tips for Success in 61A

Attend lecture, orientation, and tutorial. This one is fairly straightforward. Especially if you are new to programming, it will take time for the concepts to sink in. The course is specifically designed to give you extensive practice with the material, and you should take advantage of it. Note that *attend* here has a loose definition–maybe you don't go to orientation live each week, but you do the worksheet on your own and refer to the recording as needed for help. That's fine; the point is, keep up with the material.

Don't fall behind. See above. Please don't put yourself in a position where you have to watch 10 lectures the day before the midterm.

Learn actively. If this is your first programming course, it probably will not be helpful to take notes as if this were a history or biology course. Simply copying down the lecture slides or frantically trying to write down everything said in lecture is unlikely to be productive. The best way to learn programming is to *practice*, *practice*, and *practice*. This is especially true when it comes to exams; many first-time programming students spend a lot of time re-watching lectures and not enough time practicing programming (both in an actual coding environment and on paper via practice problems). Be wary of doing so, and remember that there is no substitute for actively engaging with the material yourself.

Understand what the different course components are meant for. Lectures will give you a high-level introduction to the material. Labs and homeworks will give you practice coding and prepare you for projects, which in turn will help you become a comfortable programmer. That said, labs/homeworks/projects are **not** the best way to prepare for exams, which are more conceptual. Prepare for exams primarily by reviewing paper-based problems from orientations/tutorials and taking multiple practice exams in timed settings.

Have fun! Although this class is intense, it is also quite enjoyable if you put in the work. Computer science is a beautiful subject, and we couldn't be more excited to introduce you to it!

3 Orientation/Tutorial Review

3.1 Write a function that returns `True` if a positive integer `n` is a prime number and `False` otherwise.

A prime number n is a number that is not divisible by any numbers other than 1 and n itself. For example, 13 is prime, since it is only divisible by 1 and 13, but 14 is not, since it is divisible by 1, 2, 7, and 14.

Hint: use the `%` operator: `x % y` returns the remainder of `x` when divided by `y`. **Extra for Experience**: How could you make your solution more efficient?

```
def is_prime(n):
    """
    >>> is_prime(10)
    False
    >>> is_prime(7)
    True
    """
```

3.3 **Tutorial:** Draw the environment diagram that results from executing the code below.

```
def f(x):
        return x

def g(x, y):
        if x(y):
                return not y
        return y

x = 3
x = g(f, x)
f = g(f, 0)
```

3.2

4 Additional Practice

4.1 Define a function, ordered_digits, which takes in a positive integer, x, and returns True if the (base 10) digits of x are in non-decreasing order, and False otherwise.

```
def ordered_digits(x):
    '''
    >>> ordered_digits(5)
    True
    >>> ordered_digits(11)
    True
    >>> ordered_digits(127)
    True
    >>> ordered_digits(1357)
    True
    >>> ordered_digits(21)
    False
    >>> result = ordered_digits(1375) # Return, don't print
    >>> result
    False
    '''
```

4.2 Draw the environment diagram for evaluating the following code.

```
cap = 9
hulk = 3

def marvel(cap, thor, avengers):
    marvel = avengers
    iron = hulk + cap
    if thor > cap:
        def marvel(cap, thor, avengers):
            return iron
    else:
        iron = hulk
    return marvel(thor, cap, marvel)

def iron(man):
    hulk = cap - 1
    return hulk

marvel(cap, iron(3), marvel)
```

5 Exam-Level Practice

5.1 **Fall 2018 Midterm 1, Question 4**. Implement `rect`, which takes two positive integer arguments, `perimeter` and `area`. It returns the **integer** length of the longest side of a rectangle with integer side lengths ℓ and h which has the given perimeter and area. If no such rectangle exists, it returns `False`.

The perimeter of a rectangle with sides ℓ and h is $2\ell + 2h$. The area is $\ell \cdot h$.

Hint: The built-in function **`round`** takes a number as its argument and returns the nearest integer. For example, **`round`**`(2.0)` evaluates to `2`, and **`round`**`(2.5)` evaluates to `3`.

```
def rect(area, perimeter):
    """Return the longest side of a rectangle with area and perimeter that has integer sides.
    >>> rect(10, 14)  # A 2 x 5 rectangle
    5
    >>> rect(5, 12)   # A 1 x 5 rectangle
    5
    >>> rect(25, 20)  # A 5 x 5 rectangle
    5
    >>> rect(25, 25)  # A 2.5 x 10 rectangle doesn't count because sides are not integers
    False
    >>> rect(25, 29)  # A 2 x 12.5 rectangle doesn't count because sides are not integers
    False
    >>> rect(100, 50) # A 5 x 20 rectangle
    20
    """
    side = 1

    while side * side ____________________________________________________________________ area:

        other = round(_____________________________________________________________________________)

        if _____________________________________________________________________________________:

            _____________________________________________________________________________

        side = side + 1

    return False
```