This is the handout for Labs 7 and 8. Work on the questions in order (i.e., start with Q1, Q2, and Q3, and then move on to other questions). In order to get credit for Lab 8, you must show up to the lab and demonstrate that you have made progress

Question 1.

Consider the problem we introduced in lecture: we want to make up an amount of money using coins from a given set of denominations, using the least possible number of coins. For example, if our denominations are `[1c, 4c, 5c]`, the most efficient way to make up `8c` is with two coins, `4c + 4c`.

Part (a)

Suppose you have computed `OPT(0), OPT(1), OPT(2), ..., OPT(n-1)`, the numbers of coins that are needed to make up `0c, 1c, 2c, ..., (n-1)c`.

Write down an expression for `OPT(n)`, in terms of the denominations and `OPT(0), ...., OPT(n)`. Hint: if you want to use the 5c coin, the total number of coins you'll need is `OPT(n-5)+1`

Part (b)

Write code to compute `OPT(0)`, then `OPT(1)`, then `OPT(2)`, ..., then `OPT(n)`. You should write a function that takes in the denominations list and the target amount, and returns the `OPT` list.

Part (c)

Write code that returns a list of the coins you need to make up a target amount, when the least possible number of coins is used. (For example, the function would return `[4, 4]`, if the denominations are `[1c, 4c, 5c]` and the target is `8`).

Follow the example from `houses.py`. The first coin used, `denom`, should be such that `1 + OPT(n-denom)` is as small as possible. To determine the next coin, `denom2`, make `1 + 1 + OPT(n-denom-denom2)` as small as possible.

(Note: it is possible to keep track of the coins used when computing `OPT`, but here we are asking for you to do something analogous to `houses.py`).

Question 2.

In this question, you will familiarize yourself with working with images using `c_img.c` and `c_img.h`, files that were given to you as part of Project 2.

You can make an image brighter by multiplying all the pixel values by a constant larger than 1; and you can make an image darker by multiplying all the pixel values by a constant smaller than one.

The files `c_img.c`/`c_img.h` store red/green/blue pixel values as `uint8_t`'s – values between 0 and 255. If you are trying to make an image brighter, you may need to round the products you obtain by multiplying by a larger constant down to `255`.

Download the image
`https://www.cs.toronto.edu/~guerzhoy/190/labs/cannon.jpg`
Convert the image to a `bin` file using `https://constructor-s.github.io/esc190_bin_image/`.
Write C code to create five different versions of the image at different levels of brightness.
Display the images using `https://constructor-s.github.io/esc190_bin_image/`.
Please write a separate C file that will compile together with `c_img.c`.

Optional: if you are interested in how one might read the images using Python, you can look at `png2bin.py` Note: you need to be able to `import` the module `Image` from `PIL`. This will work on ECF, and will work for you if you are using Anaconda. If you are not using Anaconda, you need to run the following in the terminal:

```
python3 -m pip install numpy
python3 -m pip install scipy
python3 -m pip install PIL
python3 -m pip install Pillow
```

This should work on most systems.

Question 3.

In Python, implement a Heap class into which one can insert tuples like `(priority, entry)`, and use `extract_min` to remove and obtain the entry that corresponds to the highest priority (i.e, `priority` that's a small as possible).

For example, we would like to be able to use this as follows:

```
h = Heap()
h.insert((5, "a"))
h.insert((2, "b"))
h.insert((3, "c"))
print(h.extract_min()) # "b"
print(h.extract_min()) # "c"
print(h.extract_min()) # "a"
```

Note that you only need to change a few lines of code from our implementation of Heap in class.

Note that Python has a built-in `heapq` library. You should try it out, but not use it for this lab.

Question 4.

In Python, change your implementation from a "Min-Heap" (we retrieve the entry associated with the smallest priority number) to a "Max-Heap" (we retrieve the entry associated with the highest priority number). Again, you need to change just a few lines.

Question 5.

Implement a Heap that can store integer values in C. Make sure to `realloc` if you run out of space.

Implement the c file, the header file, and the testing code. Show the TA that you can compile and debug your heap in VS Code. (First, implement a bare-bones version that you can debug. Then add functionality.)

Question 6.

Work on Project 2.