ECS 170: Spring 2021
Homework Assignment 4

Due Date:

No later than Monday, May 31, 11:59pm PDT. The grace period extends to 12:15am PDT, Tuesday, June 1. (Note: that's 12:15 in the very early morning, **not** 15 minutes after noon.)

You are expected to do this assignment on your own, not with another person.

This assignment looks really long. Don't panic. Most of this document is just examples.

The assignment:

Recall the perceptron learning model described in Episode 30. (If you haven't watched Episode 30, what follows won't make much sense, so stop now and watch that episode.) Your task is to use Python to write a function that implements the perceptron described in that episode. Your function should be named `perceptron`.

Your function should expect five arguments, in this order, left-to-right:

A number representing the threshold
A number representing the adjustment factor
A list of numbers representing the initial weights on the inputs
A list of examples for training the perceptron
A number representing the number of passes the perceptron should make through the list of examples

Let's map these descriptions onto the actual values used in the example:

In the example, the threshold was 0.5
The adjustment factor was 0.1
If contained in a list, the initial weights were [-0.5, 0, 0.5, 0, -0.5]
The list of examples was:

```
[[True,  [1,1,1,1,0]],
 [False, [1,1,1,1,1]],
 [False, [0,0,0,0,0]],
 [False, [0,0,1,1,0]],
 [False, [1,0,1,0,1]],
 [False, [1,0,1,0,0]],
 [False, [0,1,0,1,1]],
 [False, [0,1,0,1,0]],
 [False, [0,0,1,0,0]],
 [False, [0,0,0,1,0]]]
```

The number of passes through the list of examples was 4

Each example in the list of examples has two parts. The first part (True or False) indicates whether this example is a positive example (True) of the concept to be learned or a negative example (False). The second part is a list of 1's or 0's. This list (the 1's and 0's) must be the same length as the list of initial weights.

Now let's say that the variable `weights` is bound to the list of weights shown above, and that the variable `examples` is bound to that list of examples above. When your function is called like this:

```
>>> perceptron(0.5, 0.1, weights, examples, 4)
```

your function should print (not return) the following output (sorry, it's really long, just like in Episode 30):

```
Starting weights:  [-0.5, 0, 0.5, 0, -0.5]
Threshold:  0.5     Adjustment:  0.1

Pass  1

inputs:  [1, 1, 1, 1, 0]
prediction:  False    answer:  True
adjusted weights:  [-0.4, 0.1, 0.6, 0.1, -0.5]
inputs:  [1, 1, 1, 1, 1]
prediction:  False    answer:  False
adjusted weights:  [-0.4, 0.1, 0.6, 0.1, -0.5]
inputs:  [0, 0, 0, 0, 0]
prediction:  False    answer:  False
adjusted weights:  [-0.4, 0.1, 0.6, 0.1, -0.5]
inputs:  [0, 0, 1, 1, 0]
prediction:  True     answer:  False
adjusted weights:  [-0.4, 0.1, 0.5, 0.0, -0.5]
inputs:  [1, 0, 1, 0, 1]
prediction:  False    answer:  False
adjusted weights:  [-0.4, 0.1, 0.5, 0.0, -0.5]
inputs:  [1, 0, 1, 0, 0]
prediction:  False    answer:  False
adjusted weights:  [-0.4, 0.1, 0.5, 0.0, -0.5]
inputs:  [0, 1, 0, 1, 1]
prediction:  False    answer:  False
adjusted weights:  [-0.4, 0.1, 0.5, 0.0, -0.5]
inputs:  [0, 1, 0, 1, 0]
prediction:  False    answer:  False
adjusted weights:  [-0.4, 0.1, 0.5, 0.0, -0.5]
inputs:  [0, 0, 1, 0, 0]
prediction:  False    answer:  False
adjusted weights:  [-0.4, 0.1, 0.5, 0.0, -0.5]
inputs:  [0, 0, 0, 1, 0]
prediction:  False    answer:  False
adjusted weights:  [-0.4, 0.1, 0.5, 0.0, -0.5]

Pass  2

inputs:  [1, 1, 1, 1, 0]
prediction:  False    answer:  True
adjusted weights:  [-0.30000000000000004, 0.2, 0.6, 0.1, -0.5]
inputs:  [1, 1, 1, 1, 1]
prediction:  False    answer:  False
adjusted weights:  [-0.30000000000000004, 0.2, 0.6, 0.1, -0.5]
inputs:  [0, 0, 0, 0, 0]
prediction:  False    answer:  False
```

```
adjusted weights:  [-0.30000000000000004, 0.2, 0.6, 0.1, -0.5]
inputs:  [0, 0, 1, 1, 0]
prediction:  True     answer:  False
adjusted weights:  [-0.30000000000000004, 0.2, 0.5, 0.0, -0.5]
inputs:  [1, 0, 1, 0, 1]
prediction:  False    answer:  False
adjusted weights:  [-0.30000000000000004, 0.2, 0.5, 0.0, -0.5]
inputs:  [1, 0, 1, 0, 0]
prediction:  False    answer:  False
adjusted weights:  [-0.30000000000000004, 0.2, 0.5, 0.0, -0.5]
inputs:  [0, 1, 0, 1, 1]
prediction:  False    answer:  False
adjusted weights:  [-0.30000000000000004, 0.2, 0.5, 0.0, -0.5]
inputs:  [0, 1, 0, 1, 0]
prediction:  False    answer:  False
adjusted weights:  [-0.30000000000000004, 0.2, 0.5, 0.0, -0.5]
inputs:  [0, 0, 1, 0, 0]
prediction:  False    answer:  False
adjusted weights:  [-0.30000000000000004, 0.2, 0.5, 0.0, -0.5]
inputs:  [0, 0, 0, 1, 0]
prediction:  False    answer:  False
adjusted weights:  [-0.30000000000000004, 0.2, 0.5, 0.0, -0.5]

Pass  3

inputs:  [1, 1, 1, 1, 0]
prediction:  False    answer:  True
adjusted weights:  [-0.20000000000000004, 0.30000000000000004, 0.6, 0.1, -0.5]
inputs:  [1, 1, 1, 1, 1]
prediction:  False    answer:  False
adjusted weights:  [-0.20000000000000004, 0.30000000000000004, 0.6, 0.1, -0.5]
inputs:  [0, 0, 0, 0, 0]
prediction:  False    answer:  False
adjusted weights:  [-0.20000000000000004, 0.30000000000000004, 0.6, 0.1, -0.5]
inputs:  [0, 0, 1, 1, 0]
prediction:  True     answer:  False
adjusted weights:  [-0.20000000000000004, 0.30000000000000004, 0.5, 0.0, -0.5]
inputs:  [1, 0, 1, 0, 1]
prediction:  False    answer:  False
adjusted weights:  [-0.20000000000000004, 0.30000000000000004, 0.5, 0.0, -0.5]
inputs:  [1, 0, 1, 0, 0]
prediction:  False    answer:  False
adjusted weights:  [-0.20000000000000004, 0.30000000000000004, 0.5, 0.0, -0.5]
inputs:  [0, 1, 0, 1, 1]
prediction:  False    answer:  False
adjusted weights:  [-0.20000000000000004, 0.30000000000000004, 0.5, 0.0, -0.5]
inputs:  [0, 1, 0, 1, 0]
prediction:  False    answer:  False
adjusted weights:  [-0.20000000000000004, 0.30000000000000004, 0.5, 0.0, -0.5]
inputs:  [0, 0, 1, 0, 0]
prediction:  False    answer:  False
adjusted weights:  [-0.20000000000000004, 0.30000000000000004, 0.5, 0.0, -0.5]
inputs:  [0, 0, 0, 1, 0]
prediction:  False    answer:  False
adjusted weights:  [-0.20000000000000004, 0.30000000000000004, 0.5, 0.0, -0.5]

Pass  4
```

```
inputs:  [1, 1, 1, 1, 0]
prediction:  True     answer:  True
adjusted weights:  [-0.20000000000000004, 0.30000000000000004, 0.5, 0.0, -0.5]
inputs:  [1, 1, 1, 1, 1]
prediction:  False    answer:  False
adjusted weights:  [-0.20000000000000004, 0.30000000000000004, 0.5, 0.0, -0.5]
inputs:  [0, 0, 0, 0, 0]
prediction:  False    answer:  False
adjusted weights:  [-0.20000000000000004, 0.30000000000000004, 0.5, 0.0, -0.5]
inputs:  [0, 0, 1, 1, 0]
prediction:  False    answer:  False
adjusted weights:  [-0.20000000000000004, 0.30000000000000004, 0.5, 0.0, -0.5]
inputs:  [1, 0, 1, 0, 1]
prediction:  False    answer:  False
adjusted weights:  [-0.20000000000000004, 0.30000000000000004, 0.5, 0.0, -0.5]
inputs:  [1, 0, 1, 0, 0]
prediction:  False    answer:  False
adjusted weights:  [-0.20000000000000004, 0.30000000000000004, 0.5, 0.0, -0.5]
inputs:  [0, 1, 0, 1, 1]
prediction:  False    answer:  False
adjusted weights:  [-0.20000000000000004, 0.30000000000000004, 0.5, 0.0, -0.5]
inputs:  [0, 1, 0, 1, 0]
prediction:  False    answer:  False
adjusted weights:  [-0.20000000000000004, 0.30000000000000004, 0.5, 0.0, -0.5]
inputs:  [0, 0, 1, 0, 0]
prediction:  False    answer:  False
adjusted weights:  [-0.20000000000000004, 0.30000000000000004, 0.5, 0.0, -0.5]
inputs:  [0, 0, 0, 1, 0]
prediction:  False    answer:  False
adjusted weights:  [-0.20000000000000004, 0.30000000000000004, 0.5, 0.0, -0.5]
```

During Pass 4, the perceptron predicts correctly for each example and no weights are adjusted. If the examples represent positive and negative instances of the arch concept, then we can say that the perceptron has learned what an arch is.

The perceptron would not be very interesting if the only thing it could learn was the concept of an arch. Let's try to teach the perceptron the concept of logical-or (i.e., 1 or 1 is true, 0 or 0 is false, 0 or 1 is true, 1 or 0 is true). Here are the examples:

```
examples = [[True, [1,1]],
            [False, [0,0]],
            [True, [0,1]],
            [True, [1,0]]]
```

Now let’s say that the variable `examples` is bound to the list of examples shown above, and that the variable `weights` is bound to this list of initial weights:

```
weights = [0.3, -0.6]
```

Now when your function is called like this:

```
>>> perceptron(0.4, 0.09, weights, examples, 10)
```

your function should now print the following output :

```
Starting weights:  [0.3, -0.6]
Threshold:  0.4     Adjustment:  0.09

Pass  1

inputs:  [1, 1]
prediction:  False    answer:  True
adjusted weights:  [0.39, -0.51]
inputs:  [0, 0]
prediction:  False    answer:  False
adjusted weights:  [0.39, -0.51]
inputs:  [0, 1]
prediction:  False    answer:  True
adjusted weights:  [0.39, -0.42000000000000004]
inputs:  [1, 0]
prediction:  False    answer:  True
adjusted weights:  [0.48, -0.42000000000000004]

Pass  2

inputs:  [1, 1]
prediction:  False    answer:  True
adjusted weights:  [0.57, -0.33000000000000007]
inputs:  [0, 0]
prediction:  False    answer:  False
adjusted weights:  [0.57, -0.33000000000000007]
inputs:  [0, 1]
prediction:  False    answer:  True
adjusted weights:  [0.57, -0.24000000000000007]
inputs:  [1, 0]
prediction:  True     answer:  True
adjusted weights:  [0.57, -0.24000000000000007]

Pass  3

inputs:  [1, 1]
prediction:  False    answer:  True
adjusted weights:  [0.6599999999999999, -0.15000000000000008]
inputs:  [0, 0]
prediction:  False    answer:  False
adjusted weights:  [0.6599999999999999, -0.15000000000000008]
inputs:  [0, 1]
prediction:  False    answer:  True
adjusted weights:  [0.6599999999999999, -0.06000000000000008]
inputs:  [1, 0]
prediction:  True     answer:  True
adjusted weights:  [0.6599999999999999, -0.06000000000000008]

Pass  4

inputs:  [1, 1]
prediction:  True     answer:  True
adjusted weights:  [0.6599999999999999, -0.06000000000000008]
inputs:  [0, 0]
prediction:  False    answer:  False
adjusted weights:  [0.6599999999999999, -0.06000000000000008]
```

```
inputs:  [0, 1]
prediction:  False    answer:  True
adjusted weights:  [0.6599999999999999, 0.029999999999999916]
inputs:  [1, 0]
prediction:  True     answer:  True
adjusted weights:  [0.6599999999999999, 0.029999999999999916]

Pass  5

inputs:  [1, 1]
prediction:  True     answer:  True
adjusted weights:  [0.6599999999999999, 0.029999999999999916]
inputs:  [0, 0]
prediction:  False    answer:  False
adjusted weights:  [0.6599999999999999, 0.029999999999999916]
inputs:  [0, 1]
prediction:  False    answer:  True
adjusted weights:  [0.6599999999999999, 0.11999999999999991]
inputs:  [1, 0]
prediction:  True     answer:  True
adjusted weights:  [0.6599999999999999, 0.11999999999999991]

Pass  6

inputs:  [1, 1]
prediction:  True     answer:  True
adjusted weights:  [0.6599999999999999, 0.11999999999999991]
inputs:  [0, 0]
prediction:  False    answer:  False
adjusted weights:  [0.6599999999999999, 0.11999999999999991]
inputs:  [0, 1]
prediction:  False    answer:  True
adjusted weights:  [0.6599999999999999, 0.2099999999999999]
inputs:  [1, 0]
prediction:  True     answer:  True
adjusted weights:  [0.6599999999999999, 0.2099999999999999]

Pass  7

inputs:  [1, 1]
prediction:  True     answer:  True
adjusted weights:  [0.6599999999999999, 0.2099999999999999]
inputs:  [0, 0]
prediction:  False    answer:  False
adjusted weights:  [0.6599999999999999, 0.2099999999999999]
inputs:  [0, 1]
prediction:  False    answer:  True
adjusted weights:  [0.6599999999999999, 0.29999999999999993]
inputs:  [1, 0]
prediction:  True     answer:  True
adjusted weights:  [0.6599999999999999, 0.29999999999999993]

Pass  8

inputs:  [1, 1]
prediction:  True     answer:  True
adjusted weights:  [0.6599999999999999, 0.29999999999999993]
inputs:  [0, 0]
```

```
prediction:  False    answer:  False
adjusted weights:  [0.6599999999999999, 0.29999999999999993]
inputs:  [0, 1]
prediction:  False    answer:  True
adjusted weights:  [0.6599999999999999, 0.3899999999999999]
inputs:  [1, 0]
prediction:  True     answer:  True
adjusted weights:  [0.6599999999999999, 0.3899999999999999]

Pass  9

inputs:  [1, 1]
prediction:  True     answer:  True
adjusted weights:  [0.6599999999999999, 0.3899999999999999]
inputs:  [0, 0]
prediction:  False    answer:  False
adjusted weights:  [0.6599999999999999, 0.3899999999999999]
inputs:  [0, 1]
prediction:  False    answer:  True
adjusted weights:  [0.6599999999999999, 0.47999999999999987]
inputs:  [1, 0]
prediction:  True     answer:  True
adjusted weights:  [0.6599999999999999, 0.47999999999999987]

Pass  10

inputs:  [1, 1]
prediction:  True     answer:  True
adjusted weights:  [0.6599999999999999, 0.47999999999999987]
inputs:  [0, 0]
prediction:  False    answer:  False
adjusted weights:  [0.6599999999999999, 0.47999999999999987]
inputs:  [0, 1]
prediction:  True     answer:  True
adjusted weights:  [0.6599999999999999, 0.47999999999999987]
inputs:  [1, 0]
prediction:  True     answer:  True
adjusted weights:  [0.6599999999999999, 0.47999999999999987]
```

This time, the perceptron requires 10 passes through the examples to learn the concept of logical-or. For different problems, the perceptron will require different numbers of passes to learn the concept. The values chosen for the threshold, the adjustment, and the initial weights are not magic numbers. Changing these numbers will change the final set of weights determined by the perceptron, but if the examples can be separated into two classes, the perceptron will converge on a set of weights that will enable it to correctly predict the classification for every example in the training set.

Your task is to use Python to implement the perceptron described in Episode 30. Your perceptron should generate the same output that you see above when given the same input. Write your program without using any packages like NumPy that do much of the work for you. You should be able to write all the code to implement your perceptron in less than 100 lines without resorting to external packages. It's really not that difficult. (I wrote it all in about 60 lines, and I'm a fairly wordy coder.) *Strive to make your program's output match what you see above. Don't round any numeric values.*

Feel free to explore learning other concepts, with different values. For example, try learning logical-and. That should work. Try learning exclusive-or. That probably won't converge.

As always:

1) We may need to modify the specifications a bit here or there in case we forgot something. Try to be patient and flexible.

2) Just like with previous assignments, we want to see well-abstracted, well-named, cohesive, and readable functions. Comments are required at the beginning of every function and must provide a brief description of what the function does, what arguments are expected by the function, and what is returned by the function.