CS 1132 Fall 2016 Assignment 1a due 10/25 at 11:59 pm

Adhere to the Code of Academic Integrity. You may discuss background issues and general strategies with others and seek help from course staff, but the implementations that you submit must be your own. In particular, you may discuss general ideas with others but you may not work out the detailed solutions with others. It is never OK for you to see or hear another student's code and it is never OK to copy code from published/Internet sources. If you feel that you cannot complete the assignment on you own, seek help from the course staff.

When submitting your assignment, follow the instructions summarized in Section 4 of this document.

Do not use the `break` or `continue` statement in any homework or test in CS1132.

1 Pokemon Babies

Pokemon originally began as a video game starring 150 fictional creatures that you could catch, train, and evolve into new creatures. Since then, the list of pokemon has grown to about 721 in the sixth generation of the game. There are 18 that hatch from eggs, as shown below:

Pokedex Num	Pokemon Name
172	Pichu
173	Cleffa
174	Igglybuff
175	Togepi
236	Tyrogue
238	Smoochum
239	Elekid
240	Magby
298	Azurill
360	Wynaut
406	Budew
433	Chingling
438	Bonsly
439	Mime Jr.
440	Happiny
446	Munchlax
447	Riolu
458	Mantyke

In the game, there is a pokedex—a kind of catalog—that provides information on each pokemon. Each pokemon is listed by an index number, as seen in the left-hand column above.[1]

By the end of this assignment, you will have hatched eggs, trained pokemon, and made a Tyrogue evolve. This requires using the built-in functions `rand` and `fprintf`, writing `for` loops, and creating, getting, and setting values in 1-D arrays (i.e., vectors). You will also be able to create and run functions and scripts.

Download the files `assignment1a.m` and `getPokeBabyName.m` from the course website. Function `getPokeBabyName` is completely implemented for your use. You will write three other functions from scratch and fill in the script `assignment1a`. See the next section for the details.

[1] If you are interested, you can get more information about these pokemon at this URL: `http://bulbapedia.bulbagarden.net/wiki/Baby_Pokemon`.

2 Function and Script Explanations

2.1 Function `getPokeBabyName`

This function is given and should NOT be modified. You will not submit it as we will use our own copy of the function to run your code. The function provides a mapping between the pokedex numbers and the pokemon names given in the chart in section 1. Read the provided function: you should be able to write this function yourself, but it is admittedly tedious to type in all 18 pokemon babies and is therefore provided for your convenience. What you do need to do is make sure you know how to use the function. Below is the description of the function:

```
function pokeBabyName = getPokeBabyName(pokedexNum)
% Returns the name of a pokemon baby given its pokedex number.
% pokedexNum - the index of a pokemon in the pokedex (a device used to
%              provide information on the various pokemon in the game)
% pokeBabyName - the name of the pokemon associated with the given pokedex
%                number. If the pokedex number does not match any pokemon
%                baby, the name is set to 'Not Found'.
```

The code relies on an `if` statement to determine which pokemon name to return. However, the code can alternatively be written using a construct called a `switch` statement. The `switch` statement is *not* a required part of this course, but if you are interested in learning about the `switch` statement, read the section of code that has been commented out in the provided function.

2.2 Function `hatchEgg`

The thing about pokemon eggs is that you never know what pokemon you are going to get! Implement the following function:

```
function pokedexNum = hatchEgg()
% Returns the pokedex number of one Pokemon baby chosen randomly out of
%   the 18 possibilities.
% pokedexNum - the index of the chosen pokemon in the pokedex
```

Start by creating an array with the pokemon index numbers of all 18 pokemon babies. Your code should choose randomly and with equal likelihood one of those pokemon index numbers to return.

A long statement should be broken into multiple lines

Statements that are too long—require horizontal scrolling when reading on an editor or wrap to the next line when printed—reduce the readability of a program. You should continue a too-long statement onto the next line by using the ellipsis symbol `...`, i.e., dot-dot-dot. Here is an example:

```
x= [3 4 9 4 ...
    5 6 1 3 ...
    4 8];
```

The single assignment statement above was broken into three lines. (That was just a demonstration; that particular statement was not too long.) What is a too-long statement? You do not need to count—on the MATLAB editor, a faint vertical gray line on the right of the window marks the width of 74 characters. Generally, your code should not go past the gray line.

Function comments

You should always document your code. When you write a function, the convention is that the file begins with the `function` keyword, i.e., the function header is the first line in the file. Below that is the function

comments, also called the function specification. The function comments describe concisely the purpose of the function and the input and output parameters. In this assignment, we provide the exact function specification that you should use, so be sure to copy both the function header and the complete function comment into the file that you will write and submit.

2.3 Script `assignment1a`

Follow the instructions written in the code documentation of `assignment1a.m`. Write the relevant code underneath each commented section. It is possible that code from one question can be used in subsequent questions (e.g., an array created in a prior question could potentially be used in the next few questions without recreating the array each time).

Answer questions Q1.1 to Q1.3 in the `assignment1a` script now by writing code. These questions relate to the `hatchEggs` function that you have just written and the given `getPokeBabyName` function. By answering these questions, you are testing your function to make sure that it works as specified!

You will answer the remaining questions after implementing the relevant functions. Indeed, you will switch back and forth between writing functions and calling them in the `assignment1a` script. *Test each function thoroughly after you have implemented it!* This is standard practice and it will save you a lot of work in the long run if you catch mistakes early.

The `assignment1a` script is divided into sections using the %% operator. When you are working on a particular section, the background will be highlighted in yellow. If a section line appears while you are typing, such as when you start a `for` loop, no worries - just finish off the loop structure by typing the `end` keyword, and the line should go away. For your convenience, you can click on the **Run Section** button to execute an individual part of the assignment. Click on the **Run** button to run the entire script from start to finish. (You can also click **Run and Advance** repeatedly to sequentially execute each part.)

2.4 Function `train`

Every pokemon starts at level 1 with zero attack points and zero defense points. A pokemon can "train" to level up and gain attack and defense points. The higher the number of points, the stronger the pokemon is. Implement the following function:

```
function [finalAttackPts, finalDefensePts] = train(numLevels, ...
                                                   startAttackPts, startDefensePts)
% Increases a pokemon's level by a specified number of levels.  For each
% time that the pokemon levels up, the attack and defense points are each
% independently increased by a uniformly random integer value in [0..10],
% i.e., any integer in [0..10] is equally likely to occur.
% numLevels - the number of times that leveling up occurs
% startAttackPts - the attack strength that the pokemon already had
%                  before training
% startDefensePts - the defense strength that the pokemon already had
%                   before training
% finalAttackPts - the attack strength of the pokemon after training
% finalDefensePts - the defense strength of the pokemon after training
```

Answer questions Q2.1 to Q2.8 in the `assignment1a` script by writing code.

2.5 Function `evolveTyrogue`

Many pokemon are able to evolve into a different pokemon, but they occur through different means. Some evolve at a particular level, some evolve after receiving special stones, and some never evolve at all. In Tyrogue's case, at level 20, it can change into one of three pokemon depending on its own attack and defense statistics just when it is about to evolve. Implement the following function:

```
function [pokedexNum, pokeName, attackPts, defensePts] = evolveTyrogue()
% Trains a Tyrogue from level 1 to level 20 and determines whether it
% evolves into Hitmonlee, Hitmonchan, or Hitmontop:
%   106 - Hitmonlee - when attacks points are greater than its defense points
%   107 - Hitmonchan - when attack points are less than its defense points
%   237 - Hitmontop - when attack points are the same as its defense points
% pokedexNum - the index of the newly evolved pokemon in the pokedex (106,
%              107, or 237)
% pokeName - the name of the newly evolved pokemon (Hitmonlee, Hitmonchan,
%            or Hitmontop)
% attackPts - the attack strength of the newly evolved pokemon
% defensePts - the defense strength of the newly evolved pokemon
```

The attack and defense strength of the newly evolved pokemon is the same as the attack and defense strength of Tyrogue at the moment that it evolves.

Answer question Q3 in the script `assignment1a`.

3 Self-check list

The following is a list of the minimum *necessary* criteria that your assignment must meet in order to be considered *satisfactory*. Failure to satisfy any of these conditions will result in an immediate request to resubmit your assignment. Save yourself and the graders time and effort by going over it before submitting your assignment for the first time.

Note that, although all of these are necessary, meeting all of them might still not be *sufficient* to consider your submission satisfactory. We cannot list everything that could possibly be wrong with any particular assignment!

- Δ Comment your code! If any of your functions is not properly commented, regarding function purpose and input/output arguments, you will be asked to resubmit.
- Δ Suppress all unnecessary output by placing semicolons (;) appropriately. At the same time, make sure that all output that your program intentionally produces is formatted in a user-friendly way.
- Δ Make sure your functions' names are *exactly* the ones we have specified, *including* case.
- Δ Check that the number and order of input and output arguments for each of the functions matches exactly the specifications we have given.
- Δ Test each one of your functions independently, whenever possible, or write short scripts to test them.
- Δ Check that your scripts do not crash (i.e., end unexpectedly with an error message) or run into infinite loops. Check this by running each script several times in a row. Before each test run, you should type the commands `clear all; close all;` to delete all variables in the workspace and close all figure windows.

4 Submission instructions

1. Upload files `assignment1a.m`, `hatchEgg.m`, `train.m`, and `evolveTyrogue.m` to CMS in the submission area corresponding to Assignment 1a before the deadline.
2. When the scores are released read the grader's feedback carefully.
3. If you need to resubmit, fix all the problems and go back to Step 1! Otherwise you are done with this assignment. Well done!