Assignment 2

Recursive functions. GitHub issues.

You may need to refer to the Pervasives module[1] documentation occasionally. All the functions there are available in OCAML by default. You are NOT allowed to use everything that's there unless we have talked about it or unless the assignment question asks you to look it up.

In this assignment you are asked to implement 9 functions. Each function is worth two points, and the assignment has two extra points for "style", for a total of 20 points.

The assignment expects you to use GitHub and Git to keep track of your work. You have already set up GitHub on your first assignment. What we need to do now is "update" your GitHub with the new information. This is a somewhat delicate process, as your "branch" has deviated from the instructor's "branch", and we need to realign them.

1. The first thing we need to do is set up the instructor's repository as a remote repository you can access. You will only have to do this first step once, while the other steps you will have to repeat for each assignment.

   - From the terminal in the directory that is your repository, do: `git remote add instr https://g`
   - Now do `git remote -v` and you should see a list of 4 items, two called "instr" and two called "origin".

2. Now we need to download the updated version of the instructor's repository and merge the changes into ours.

   - Type: `git fetch instr`
   - Type: `git merge master instr/master`. This may give you a editor window to edit a commit message. You don't need to edit it, just "save" (writeOut) and exit. (This is probably not your prefered editor. See this page[2] for how to set up your favorite editor for use in these merge situations. I would recommend setting it to SublimeText).
   - If it reports no problems, you're ready.
   - If it reports merge conflicts, you will need to resolve those first, then make a commit. this page[3] has some instructions on how to do that. Or contact me.

3. From this assignment on, we will start developing an issue-driven methodology for your assignments. Every individual item you are working on should have a corresponding GitHub issue associated with it, and when you "commit" your work you would also be closing the issue. Here is the process for doing so:

[1] http://caml.inria.fr/pub/docs/manual-ocaml/libref/Pervasives.html
[2] https://help.github.com/articles/associating-text-editors-with-git/
[3] https://help.github.com/articles/resolving-a-merge-conflict-from-the-command-line/

- Start by going to your project's page in GitHub (and be logged in to GitHub). You should see a "Settings" button near the right side of the main project bar. Click on it, and make sure that under the "Features" section there is a checkbox next to "Issues". You only have to do this once.
- Now the project bar should have an item called "Issues", click on it. This is where you can see pending issues, and also create new ones.
- Click the "New Issue" button to create a new issue. Put as its title "create stubs for hw2 functions". Of course in future issues you'll need to create an appropriate title.
- In the "Leave a comment" section you can elaborate on the issue. This section understands GitHub Flavored Markdown[4] and you should start getting used to that by reading the information in that link. One neat feature is task lists[5], give them a go at some point, or even here by creating a checklist with all the functions you need to work on. Then as you implement a stub for each you can check them off.
- Click "Submit New Issue" when ready. This will create the new issue and give it an increasing number, starting from `#1`. You can use those numbers to refer to issues.
- Clicking on an issue shows you any discussion on that issue, and allows you to leave a comment. Do this now for practice. You can also close the issue from there, but we will show how to close issues via commits.
- Now you should go to the assignments file, read through the functions and for now implement "stubs" for them. So for each function you would write something like:

  ```
  let f (x : int) = (x, 1)
  ```

  where of course you need to make sure these are the "right things" type-wise. The point of this part is to get the function signature right. So the value you use on the right may be completely wrong, for now, but we just want it there to correctly identify the types of functions.
- As you do this for each function, do `#use "assignment2sub.ml;;"` to make sure the signature is right, and if it is then check off the corresponding checkbox in the task list we created in GitHub for this issue.
- When you have all function stubs ready, it is time to make a commit. Here are the steps for that:
  - `git add .` This "prepares" your changes for committing. You can also do `git add assignment2sub.ml` to add just the one file (otherwise it picks up all changes in that directory).
  - `git status` This should show you the one file ready for commit.
  - `git commit -m "Implement hw2 stubs. Close #1"` This creates a commit, and the last line should be typed exactly that way (with the number reflecting which issue you want to close).

[4] https://help.github.com/articles/github-flavored-markdown/
[5] https://github.com/blog/1825-task-lists-in-all-markdown-documents

- `git push origin master` This should upload your changes to GitHub, and also automatically close that issue.

4. From this point on you will repeat this process for each individual function:

   - Create an issue about "writing tests for function . . . ". Use the issue's comment space to write thoughts about what you should test.
   - Create an issue about "write function . . . ". Use the issue's comment space to write thoughts about how you might go about doing this.
   - Create an issue about "polish function . . . ".
   - Write tests for the function. Watch (most of) them fail with your current stub implementation.
   - Commit these tests and close the issue about writing tests.
   - Implement the function. Make sure it passes your tests.
   - Commit that function and close the issue about writing the function.
   - Now that you have a working code, take another look at it and polish it up, paying attention to line breaking, indentation, variable names, unnecessary code etc. Don't forget to consult the style guide[6].
   - Rinse and repeat with the each function.

5. You will find two files in this directory:

   - `assignment2sub.ml` This is the main submission file. It is where you will add your code for the various functions that you need to write. There are comments in that file to show you where to add your function definitions, and to tell you what your functions should do.
   - `assignment2tests.ml` This is a file with a small number of tests, and you should add plenty tests your own. "Tests" are arranged as lines `let ... = e` where `e` is a an expression that is meant to evaluate to a boolean indicating if the tests succeeded or not.

6. To "run" your tests, start an OCAML session in the terminal via `utop`, do:

   ```
   #use "assignment2hw.ml;;"
   #use "assignment2tests.ml;;"
   ```

   You should be able to use auto-completion.

   - The first `#use` should print for you the type signatures for all the functions you had to write. Make sure this matches the signatures described in the code file.
   - The second `#use` should print a bunch of `true` values out, for all the existing tests along with all the tests you added.

[6] ../notes/style.html