COMPSCI 230 — Spring 2017
Homework Preparation and Submission Instructions

Carlo Tomasi

Short Version

1. Download the assignment from the homework page[a] and unzip it.
2. **The only files you need to edit are `solution.tex` and, if code is required, `code.py`.**
3. The answer to each problem **must** fit on a single page.
4. Put your full name in place of `Mandatory Author` in the command

   ```
   \authors{Mandatory Author}[][]
   ```

   at the top of `solution.tex`, and put the full names of any partners in the brackets.
5. Write any required text inside the `solution` environment for each problem in LaTeX.
6. Replace each instance of the comment

   ```
   # Your code here
   ```

   in `code.py` with your code.
7. Check the code outputs for correctness. These are either written automatically to `output.txt` or your code writes them to a file whose name is specified in the assignment.
8. Compile `solution.tex` to produce `solution.pdf`. Your code in `code.py` is automatically inserted where appropriate during compilation, and so are the outputs.
9. Double-check and proofread your solution carefully.
10. Each group submits **one** copy of `solution.pdf` to Gradescope and **one** copy of `code.py` to Sakai (if the assignment has code) before 10:05am on the due date. The Gradescope class code is `M3WGDM`. **Enter the names of your group peers on the Gradescope submission form!** Submit files to Sakai as attachments, and do *not* use the Sakai dropbox. No late assignments are accepted.

[a] In this document and in the assignments, cyan-colored text like this denotes a clickable link to a web page.

The instructions above are terse, and are intended for reference when you work on future assignments. The rest of this document is a longer version that explains what is going on in some detail.

Introduction

This document explains why we want you to care about what your assignment solutions for COMPSCI 230 look like, and how to achieve proper homework formatting. **Follow these instructions scrupulously for full credit.**

The last section of this document, in particular, takes you through the workflow for preparing your solution to assignment 1. Similar steps will hold for subsequent assignments, so you will likely refer back to this document (or at least the short version on the first page) in the future.

Why Format Matters

There are several reasons why we require carefully formatted homework assignments:

- Mathematics is all about clarity, and you cannot write clear math sloppily. Sloppy math is no math.
- If you become a computer scientist, preparing documents for others to read will be a major part of what you do. If your documents look professional, so do you.
- COMPSCI 230 is a large class, and grading assignments is much more efficient if they are formatted well, so we do not need to read scribbles and we know where to look for answers. You receive faster feedback as a result.
- We use Gradescope to grade assignments. This tool expects each answer to be in a very specific place in a PDF file. The macros we wrote for you make it easy to meet these expectations.
- I wrote a combination of LaTeX commands for a new LaTeX class implemented in file `homework.cls`, and Python code (in a file called `homework.py`) that make it straightforward to include code and code output into your solution.

I know that these reasons are enough for you, but here is one more, just in case:

- **Grade points will be deducted for every deviation from the required format.**

Who Hands in What, Where, and When

Homework Files. Every homework assignment shows up on the class homework page with a PDF file that describes the assignment itself, plus a ZIP archive file that contains all the files necessary for the assignment. The ZIP file in turn contains a directory[1] whose name is a two-digit string that identifies the assignment, starting with `01` for the first assignment.[2]
Do not change the names of any of the files in the ZIP archive. The only files you need to edit are `solution.tex` *and* `code.py`*. As you edit those files, make sure you do not disturb what is already in them, and make sure you write in the appropriate places as explained below.*

Partnering for an Assignment. You may work on an assignment alone, as part of a duo, or as part of a trio. You may change groups from assignment to assignment, but once you start working on an assignment you may not change partnership.[3]

Solution Submissions. Each group hands in *one* assignment solution that lists the names of all students (up to three) who worked on it. *Do not hand in multiple solutions for the same group. This would make our work much harder.*

All assignment solutions have some text, and some have Python 3 code. *No code written in earlier versions of Python will be accepted.* Version 3.6 is recommended for consistency with the version we run.

Every solution submission comes with a PDF file that contains both your answer text and appropriate listings of your code, if any, as explained later on. For assignments that involve code, a separate `code.py` file should also be submitted for us to run and verify.

The name of your PDF submission is `solution.pdf`. The name of your Python submission, if any, is `code.py`. *Do not change the names of these files, a template for which is provided in the ZIP archive.*

One person per team submits the PDF file for the team to Gradescope. The same person submits the Python file, if required, to Sakai.

Submission Deadline. *All assignments are due by beginning of class (10:05 am) on the due date.* The Sakai and Gradescope sites will close for submission at that time, so this is a hard deadline.

LaTeX. Your PDF submission is to be prepared in LaTeX, a document preparation language everyone in computer science, mathematics, and several other disciplines uses to write books, papers, and more. If you become a computer scientist, you will use LaTeX at some point. You might as well learn how to use it now. In exchange for 2-3 hours invested at the beginning of this semester, preparing your homework for submission will become much easier and less error-prone than it would be without LaTeX, as many of the necessary tasks can be automated. You will save both time and grade penalties with this initial investment.

[1] Directories are also called "folders" on some operating systems.

[2] Two digits are used so that your directories show up in the proper order if you keep them all together.

[3] If you were to start an assignment with A and finish it with B, you would be transferring information between A and B, which would be a violation of the rules of academic integrity. Joining one or two other students late is OK, because it involves no undue transfer of information. Splitting from a group is not OK, because the two new groups share information from before they split. You get the idea.

How to Prepare a Homework Solution

This Section takes you through the workflow of handing in homework 1. Similar steps, with obvious variations, will hold for future assignments.

It is assumed that you have read enough about LaTeX that you understand how to compile the `.tex` file you write into a `.pdf` file. It is also assumed that you have installed both Python 3 (with the IDLE development interface) and LaTeX, and know how to use them.

If not, stop now, and study the LaTeX intro and Python 3 intro. Install the software[4] and follow any links you need from those two documents to familiarize yourself with the basics. Then read on.

1. Download the assignment text `assignment.pdf` and the ZIP file `HW01.zip` from the homework page. If you use ShareLaTeX, you can just upload the entire ZIP file to that site and continue with step 5. If you have installed LaTeX on your computer, on the other hand, do steps 2-4 below.

2. Unzip the ZIP file. This will create a directory `HW01` with all the files needed for the assignment.

3. Move the file `assignment.pdf` to directory `HW01`, so all you need is in one place.[5]

4. Move the directory `HW01` to a place you will be able to find later: It is good practice to put all the homework directories for this course in the same directory on your computer, so when you study for exams you find everything easily. Also, some assignments may refer to older assignments, so it is good to know where to find those on your computer.

5. Open `assignment.pdf` to see what you are asked to do for the assignment. An assignment is divided into *parts*. A part is simply a way to group problems by topic. Often a part will have a (possibly long) preamble explaining various aspects of the corresponding topic. Each part is in turn divided into *problems*, and **each problem requires generally one solution, which is expected to fit on a single page** in the document `solution.pdf` you submit.

6. Each problem (not part!) title has a tag name in parentheses. For instance, problem 1.1 has tag **math**. For problems that involve no code, the tags are merely a convenient way to make the correspondence between problems and your solutions easier to notice than the problem numbers (1.1, 1.2 and so forth). For problems that involve code, on the other hand, tags are crucial for synchronizing LaTeX and Python files. More on this later on.

7. To start solving problem 1.1, open the provided file `solution.tex`. As you see, the structure of this LaTeX source file mirrors that of the assignment.[6] Let us look at the initial elements of this structure:

[4] If you prefer, you can use ShareLaTeX online, instead of downloading LaTeX to your computer. The latter solution is more efficient in the long term.

[5] The assignment PDF file is separate from the ZIP so it is viewable directly from the class homework page. It is not duplicated in the ZIP file to avoid possibly inconsistent versions.

[6] In fact, a program generates `solution.tex` automatically from the assignment.

- The instruction

  `\documentclass{homework}`

  tells LaTeX to read formatting and other instructions from the document class defined in the provided file `homework.cls`. This file remains more or less the same throughout the semester. It may change a little over time, and this is why a fresh copy is given with each assignment. Among other things, this file tells LaTeX to find other files in the homework directory, including `macros.sty`, which contains LaTeX command definitions that make it easier to write the assignment. Those definitions could be placed directly in `solution.tex`, but putting them in a separate file keeps things cleaner. Typically, hints will be given as to what commands are useful, but you can also explore `macros.sty` yourself.

- The line

  `\homework{COMPSCI 230}{Spring 2017}{1}`

  defines course name, semester, and homework number, so LaTeX knows how to build page headers and footers.

- The command

  `\authors{Mandatory Author}[][]`

  is important for your submission. **Replace the words `Mandatory Author` with your name. If you work with one or two more people on this assignment, add their names in the square brackets.** Do not change brackets or braces. Curly braces denote mandatory arguments in LaTeX, while brackets denote optional ones.

- The entire document (the part to be typeset) is included in a `document` environment, which starts with `\begin{document}` and ends with `\end{document}`.

- The command

  `\part{\latex}`

  merely provides a title for the first part. The `macros.sty` file, which LaTeX knows about through the `homework` class, contains a macro `\latex` so that it is easy to write "LaTeX" rather than just "latex."

- The command

  `\problem{math}`

  tells LaTeX that everything in the subsequent `solution` environment (between `\begin{solution}` and `\end{solution}`) refers to the *tag* **math**. Since this problem requires no Python coding, this tag is not too important here. It will be crucial in the second problem. Let us now go back to step-by-step instructions.

8. **Write text and LaTeX commands inside the `solution` environment** so that when you compile `solution.tex` the page for Problem 1.1, the file `solution.pdf` looks as similar as you can make it to the corresponding text in `assignment.pdf`. Compile your `solution.tex` file to make sure, and iterate until you are satisfied.

   LaTeX Errors and Warnings. The first time you compile `solution.tex`, you may get some error and warning messages. Error messages stop compilation, and you need to fix the problem before

recompiling. Please check that you have no unbalanced braces or dollar signs. Read the message that comes with the error: It is often quite informative. One way to make debugging much easier is to compile the LaTeX source file often as you write in it: If an error occurs, it must be in the last few lines you wrote since the previous compilation.

Warnings are more benign, and do not stop compilation. For instance, the following warning may occur:

```
Package hyperref Warning: Rerun to get /PageLabels entry.
```

This tells you that LaTeX collected some information about cross-references within the document, and needs a second pass to use that information to link document parts appropriately. If you see similar warnings, just recompile the file. This will fix the references, and the warning will go away.[7]

Another warning may be similar to the following:

```
Class homework Warning: Did not find file ./output.txt
in the current directory on input line 59.
```

This is because, for homework that requires code, LaTeX (or rather the `homework` class) knows to expect a file `code.py`, from which to read Python code, and a file `output.txt`, from which to read the output that this code generates. These files are expected to be in the same directory as `solution.tex`. The file `code.py` is already there because it is provided with the assignment, but `output.txt` is not there yet, because you have not yet run any Python code that makes `output.txt` and writes to it. The warning will go away only after `output.txt` shows up.

9. Part 2, with problem 2.1, is more interesting, because LaTeX will insert Python code and output from the code automatically into your LaTeX source. The key to this is the **slice** tag for this problem. Open `code.py` in IDLE and inspect it. The very first line imports various functions from `homework.py` (provided). These include `begin`, `end`, which are called around a comment that tells you to enter your code there, and `beginOutput`, `endOutput`, which are called around some code that tests the code you write.

   As you see, the argument to all four of these functions is the string `'slice'`, and this is how Python and LaTeX understand each other. If you look at the solution to Problem 2.1 in `solution.pdf`, you will see a box with the text

   ```
   # Your code here
   ```

   That text was generated by the first `\insertCode` command in `solution.tex`. The command `\problem{slice}` (no quotes here) tells LaTeX:

   > *If you encounter an* `\insertCode` *command in the next* `solution` *environment, look for file* `code.py` *in the current directory, extract any text between a line that contains* `begin('slice')` *(and nothing else) and a line that contains* `end('slice')`*, and insert that text* verbatim *in place of the* `\insertCode` *command.*

[7] In rare cases, you may have to compile the file three times.

So if you replace the comment in `code.py` by your code, a listing of that code will show up in `solution.pdf` the next time you compile `solution.tex`.

Take a minute to absorb what is going on here. It is important that you understand this.

10. Delete the comment in the Python file and replace it by your code. Once you are done, if your code runs without errors, the output it produces will show up in `output.txt`, because its name is contained in the variable `outputFileName`, and the test code in `code.py` writes to that file. The `'a'` argument to `open` tells Python to append the output to the file, so any output produced by any previous code you may have run is still in `output.txt`. If you had used the `'w'` argument instead, old contents in `output.txt` would be lost. The call `clearOutputFile()` at the top of `code.py` erases `output.txt` to start anew.

11. Once you are happy with your code, run `code.py`. If your code had errors, the test code fails. Since it is in a `try` block, the error causes execution to skip to what's after the `except` keyword. The `pass` command does nothing, and execution of `code.py` continues after that. All the subsequent test code fails as well, because you haven't solved subsequent coding problems yet, so nothing useful happens.

    If your code runs without errors, on the other hand, the test outputs are written to `output.txt`. Crucially, these outputs are again surrounded by `begin('slice')` and `end('slice')` lines. While in `code.py` these lines are code (and also double as tag delimiters for LaTeX), in `output.txt` they are just tag delimiters.

    The command `\insertOutput` in `solution.tex` is analogous to `\insertCode`, with the only difference that `\insertOutput` looks for file `output.txt` instead of `code.py` to get its text. As a result, LaTeX automatically retrieves the output from `output.txt` and displays it appropriately in `solution.pdf`.

12. Problem 3.1 (tag: **names**) is a bit different, because you are asked to write a function `switch` that writes to a file with a name given as argument, rather than to `output.txt`. So you cannot use `\insertOutput`. Instead, `solution.tex` contains the following commands:

    ```
    \insertFileIfExists{enumeration.txt}
    ```

    This command looks for a file `enumeration.txt`. If it finds it, it places its contents *verbatim* in place of the command itself. Otherwise, it issues a warning and continues compilation, without doing anything.

13. Another difference arises in Problem 3.2 (tag: **maze**), for which the file `solution.tex` contains the following commands:

    ```
    \begin{center}
    \inputFileIfExists{bigMaze}
    \end{center}
    ```

    The `center` environment centers its contents horizontally on the page. The command `\inputFileIfExists{bigMaze}` looks for a LaTeX file `bigMaze.tex`. If it finds it, it executes its contents *in lieu* of the command itself. Otherwise, it issues a warning and continues compilation, without doing anything.

The difference between `\insertFileIfExists`, used in Problem 3.1, and `\inputFileIfExists`, used in Problem 3.2, is that the former can take any file and paste its contents *verbatim* where the command appears, so what you see in `solution.pdf` looks exactly what you would see if you looked at the file. In contrast, `\inputFileIfExists` assumes that the given file is a `.tex` file (this is why you are not asked to specify the file extension), and executes what is in it. As an example, suppose that file `example.tex` contains the following:

```
The square root of 2,
\[
        \sqrt{2} \approx 1.414 \;,
\]
is an irrational number, so that $\sqrt{2} \notin \mathbb{Q}$.
```

Then, saying `\insertIFileIfExists{example.tex}` produces

```
The square root of 2,
\[
        \sqrt{2} \approx 1.414 \;,
\]
is an irrational number, so that $\sqrt{2} \notin \mathbb{Q}$.
```

in `solution.pdf`, while saying `\inputFileIfExists{example}` yields

The square root of 2,

$$\sqrt{2} \approx 1.414 \;,$$

is an irrational number, so that $\sqrt{2} \notin \mathbb{Q}$.

14. For problems that require short code snippets and test outputs, code and outputs are typically requested as part of a single problem. For tasks with longer answers, such as the maze problem, code and output may be requested as part of separate problems (3.2 and 3.3 in the example) with different tags (**maze** and **path**). Do *not* change these partitions, as LaTeX, Python, and Gradescope all depend on them.

15. When you are done with the assignment, proofread it accurately. Then submit `solution.pdf` to Gradescope and `code.py` to Sakai. **One** copy per group, but remember to **enter the names of your group peers on the submission form!**