CSCI 1100 — Computer Science 1
Fall 2019
General Comments: Read This Before Starting the Homework

Overview

Welcome to CS-1 homework assignments. This short document should be used as a reference for all your homework submissions.

Submission of homework assignments: test first, but you can submit many times

As you learned in Lab 1, all homework will be submitted electronically through Submitty, the Department of Computer Science homework submission and auto-grading server. This link is also available on the Course Website. The homework submission server for a particular assignment will typically be available no later than two days before the homework is due.

For homework assignments we will use a combination of auto-grading and manual verification. For the autograded part, we will use both *open* and *hidden* tests. Open tests will show you your score and the corresponding output on the submission page shortly after you submit your work. For hidden tests, you will not be able to see our output, the expected output, or your points when you submit. Once grades are released, you will be able to see how many points your code scored, but the rest will remain hidden. Manual verifucation looks at other aspects of your code, You will not see the manually graded portion until the TA has looked at your submission and the grades are released.

Make a habit of testing your program by running it using `spyder`. When you do, first make sure that it can run. Then, go through the logic to make sure that it is correct. Then thoroughly test your code using all the different types and values of input data to make sure that it works correctly according to the specification. Learning to test your code is a big part of learning to program and it will help you when we use hidden tests or unexpected input. Use `Submitty` only after are confident your code is working. You will not be penalized for submitting a program multiple times, but you are generally graded on the last solution you submit.

Submit your own work

Prior to HW3 we will distribute *Collaboration Guidelines* that describe in detail what is and what isn't allowed, but for now just make sure you submit your own work. We have tools that allow us to flag code that is too similar and we will use them.

Late homework policy

When a homework has multiple parts, the submission time is the time that the last part is submitted. For example, if you submit parts 1 and 2 three days before the due date and part 3 one hour after the submission deadline, you will have used up one late day. Reread the late homework policy in the syllabus. You have a total of three full or part days you can use for late homework assignments in the whole semester and up to two days can be used for a single homework, assuming you have them available. It is highly recommended that you save these for later homework assignments that will be harder to complete and longer.

Spyder vs. Homework Submission server

On occasion, when the homework submission server runs your code it will produce different output than `spyder` does. Most of the time this is because we are using different test cases or different data sets then what we gave you to test against. Sometimes it is an error you need to debug. Regardless, In such cases pay careful attention to the server's output and try to figure out what happened. If there is an error, think about what could be causing problems. Usually you've done something wrong in the way you wrote, submitted or executed your program. If you can't fix the problem, check the Discussion Forum on `Submitty` for a relevant discussion. Only after checking should you post a Forum question (and remember do not post your code!) This is also a good topic for help during office hours.

Input format for homework assignments in this class

In all homework assignments, we will use a convention specific to CS 1. If we ask you to read an input, then after you read it you must immediately print it. For example, the following is the correct method to input the name string and the email address string:

```
name = input('Enter a name ==> ')
print(name)
email = input('Enter an email ==> ')
print(email)
```

The reason for this is explained in the *Common errors* section below.

How will you be graded?

- Program correctness will be the most important determinant of your grade. In some cases, especially later in the semester, we will test your code with many different inputs, not just the ones we provide as sample test cases in the homework description. Test your code thoroughly.
- Submitty uses a character matching algorithm to assign the suto-graded points. Which means that for full credit, you need to match the expected output exactly. This is not a bad thing. Making sure your output matches the sample output teaches you to use the `print` function and strings effectively. If we feel Submitty is being overly harsh, we **may** choose to give you back some credit for small differences in output spacing, but you should not count on it.
- Your code will be checked to ensure that it is well-written and has clear, correct program logic. Refer to the *Coding style conventions* section for a list of conventions you should follow. We will start out being gentle with this on HW 1, but by HW 3 you should be following the guidelines closely, and by HW 5 you should be experts.
- *We read your code every time.* If you are tempted replace the calculations and algorithms with hardcoded `print` statements that mimic the expected output **do not do it.** This is not a valid submission, the TAs look for it, and you will get a **0**.
- Remember that unless you explicitly change the version you want graded on the Submitty submission page, the last version you submitted is the one that will be graded. You can make a past submission the active one if you want, but you need to specify this explicitly on the server. This must be done by the submission deadline. It is your responsibility to manage it.
- We often get the question, **can I use a programming construct that we have not learned yet?** Unless we give you an explicit warning, you will not lose points for this.

However, we strongly urge against it. We design homework assignments to target the concepts we are learning right now. While there may be other methods to solve a specific problem, if you feel you need to use them you are not properly learning and applying the concepts we are teaching.

Appendix

Code style conventions

We expect your code to follow certain code style conventions outlined below (we haven't covered many Python features yet, so do not worry if something does not sound familiar but make sure to revisit this list as we move forward with Python):

1. Write docstrings (multiline comments using `''' '''` or `""" """`) for all public programs, modules, functions, classes, and methods.
2. Use descriptive names for all variables, functions, modules, classes, etc.
3. Make effective use of comments and blank lines to set off blocks of code and to clarify complex or non-standard constructs.
4. Structure your program appropriately so that it easily understood. A well structured program should generally have the program components in the following order
   (a) A general comment describing the program.
   (b) All import statements.
   (c) All function definitions.
   (d) Protect your program with a guard *(don't worry about this until Lecture 11)*:

   ```
   if __name__ == "__main__":
       # Main body of your program or module
   ```

   (e) The main body of your program.

   Well structured programs are easy to read and debug. We will work hard to help you develop good habits early on.
5. Envision your code as having two main parts: the main body and the functions that help the main code.
   (a) Make sure your functions accomplish one well-defined task. This makes them both easy to test and useful in many places.
   (b) Move repeated blocks and complex blocks of code into functions.
   (c) When the primary purpose of your code is to provide functionality as a module, it is best to use the code in the main body to test the module functions.

Common errors

- `Error`: **EOFError: EOF when reading a line**

`Explanation:` You are trying to read too many or too few inputs. Read the problem specification carefully. Your program must read the same number of inputs as are required according to the given problem. For example, if we tell you to read a name first and an email address next, that means your program must have two `input` statements. If it does not, you will get an error like:

```
EOFError:  EOF when reading a line
```

- `Error:` **Please enter a name==> Please enter your email==>** (Your Submitty input statements all show up in one line.)

  Remember that for CS1 homework, you should always print out the value you read immediately after reading it as shown below:

```
name = input('Enter a name ==> ')
print(name)
email = input('Enter an email ==> ')
print(email)
```

The above program will produce a different output in spyder and on Submitty:

```
Enter a name ==> Rensselaer
Rensselaer

Enter an email ==> info@rpi.edu
info@rpi.edu
```

Spyder output
(what you see on your computer)

```
Enter a name ==> Rensselaer
Enter an email ==> info@rpi.edu
```

Homework submission server output
(what you see on the submission server)

If you forgot to add the print function calls, you would actually see something like this in the submission server which will be considered incorrect:

```
Please enter a name==> Please enter your email==>
```

The difference is that for each part of the homework, we place all the input into a file and do what's called "running from the command-line." In the above example, we are using an input file (let's assume it is called `input.txt`) that contains the two strings `Rensselaer` and `info@rpi.edu` on two lines.

When a program is run from a command shell, we use a command-line of the form:

```
python part1.py < input.txt
```

I did a little googling and I am not finding an easy way to do this with spyder. Doing it outside of spyder is possible on all systems, but then you lose the advantages of using an IDE. Still, if you want to learn how, you can ask us during office hours. Anyway, here is the bottom line:

Anytime you read input, print it out immediately afterwards.

Note that this is only required for submissions to `Submitty` . It is not necessary in labs or on tests; although, if you echo the input back in those situations, you will not be penalized.