CSCA20 Exercise 2

Deadline(s)

There will be a pre-grading run on Friday September 28 at 11:59pm. If you submit your work before this deadline, you will receive feedback from the automarker on what grade you would receive on your submission, as well as any errors in your code we've found.

The actual deadline for this exercise will be Sunday September 30 at 11:59pm. The latest version of e2.py you submit before this deadline (including possibly anything you've submitted before the pre-grading deadline) will be examined to determine your final grade for this exercise.

Deadline-related suggestions

I suggest you treat the pre-grading deadline as your actual deadline; ie. try to get all of your work done before then. If you earn a perfect grade, then you're done! And if you happen to make any errors, then you'll have two days to correct them in order to increase your final grade. Not many courses give you this opportunity, so make the best of it.

We will mark the newest version submitted not later than the deadline. Don't submit just once; instead, submit at least once well in advance, so that you know what you're doing, and then keep submitting new versions as you do more of the exercise. That way, if you run out of time on the last function that you just can't do, you'll still get marks for the others.

What you'll be required to do

David is too lazy to do his taxes himself, so he's planning to get his students to do his taxes for him. In this exercise, you'll be implementing functions to calculate various figures related to income tax. You can find all necessary taxation information in the following link https://www.canada.ca/en/revenue-agency/services/tax/individuals/frequently-asked-questions-individuals/canadian-income-tax-rates-individuals-current-previous-years.html in the section "Federal tax rates for 2018"

You can find (unfinished) definitions for three functions in e2.py alongside docstrings describing the arguments they take, the value(s) they should return, and anything else they should do. You will be required to complete the functions by writing code for their bodies. The areas where you are supposed to write your code are clearly denoted with comments.

The docstring for each function describes what each function is supposed to do and/or return. It's up to you to implement them to do what their docstrings require. Do *not* modify these docstrings. And also, do *not* modify the format of the functions' arguments.

Note that the last function you'll be required to implement must be done using a single line of code (not including comments). See the comments within the body of that function for more details.

Make sure to keep any printing lines inside the `if __name__ == '__main__'` block at the bottom of e2.py. Specifically, *do not print inside your functions unless specifically instructed to*! Such print statements could mess up the automarker's reports.

You can assume that we will be testing your code using sensible values for the arguments of your functions. For example, we won't be testing with negative incomes. Additionally, you can assume that the function argument values will be of the correct type; eg. if the docstring says that an argument is a float, we will only test by giving it floats.

What you *should also* do
In the comments of e2.py, you'll see that I've given suggestions for testing your functions. While we won't be grading you on how well you test your functions, we *will* be grading you on how well your functions *work*. And the best way for you to make sure your functions work is to test them thoroughly.

The guidelines for testing from exercise 1 still hold. In addition, you may have noticed that your functions should behave differently depending on different income levels. For example, the calculation for income taxes at an income of $46605.00 is not the same as the one for $46605.01[1]. In the discipline of Software Engineering, we call these kinds of values *boundary values*.

You should try to identify all of the boundary values and, for each boundary, make sure to test using a pair of incomes; one income higher than the boundary, and one income lower.

Attention to detail
Remember that Python cannot make any intelligent corrections for you. It will only do exactly what you tell it to. Any typos or logical errors will make your code behave completely differently from what you want it to! So the best way to make sure you haven't made any of these errors is to test! (See above)

Style
The adherence of your modified e2.py to the PEP8 style guide will count towards your grade. After you've implemented and tested your code, run it through the online style-checker provided in the Resources section of the course website and then fix all of the style errors it will no doubt find in your code.

It may seem tedious to write according to the PEP8 style guide when Python will understand non-PEP8-compliant code just as well. But remember that when you're programming, you're programming for humans as well. Writing code which follows style conventions will help humans read it. And as you practice following these conventions, they will become habit and you might eventually find yourself doing it by default.

Thus, the second last thing you should do before you submit is to run your code through the PEP8 style checker. The last thing you should do is the sanity check (see below).

Sanity check

---

[1] If you don't understand why the calculations will be different, take a look at the link to the tax information

If nothing else, make sure your code runs *without any syntax errors*. In lecture, you've seen how easy it is to forget a bracket and end up with invalid Python code. Even the best programmers make these mistakes regularly. If the code you eventually submit has these kinds of errors, you *will get a 0* on your exercise grade, no matter how well you did everything else! So the last thing you should do before you submit is a final sanity check to make sure you can at least run your code without any errors!

What to hand in

Submit your completed e2.py on MarkUs
https://markus.utsc.utoronto.ca/csca20f18/?locale=en