Lecture 06

Debugging and Reading Others' Code

21 Pluviôse, Year CCXXX

Song of the day: ***The (After) Life of the Party*** *by Fall Out Boy (2007).*

Practice Problem: *Curb your (graduation) privilege.*

This problem will focus on **debugging**. You may have heard the word before, and for good reason: knowing how to debug properly is one of the most important skills in any programmer's toolbox.

In computer programming and software development, debugging is the process of finding and resolving bugs (defects or problems that prevent correct operation) within computer programs (**source**). While some bugs can be identified by simply reading your code, as programs get longer and more complex, more often than not you will end up debugging your programs using several sample values, observing the resulting behavior, and then checking if you are getting the expected results.

The program you will be debugging is simple; it basically prints `True` if a student meets the requirements needed to graduate, and `False` if not. The requirements are as follows:

A student may graduate if:

- They have accumulated 64 credits **and** are approved seniors, or:
- They have accumulated 40 credits **and** have special permission from their advisor, or:
- They have special permission from the dean to graduate, regardless of how many credits they have.

For this program, you may assume that the user will **always** enter `y` for "yes", and `n` for "no".

You may find the buggy version of this program in the file **graduation_status.py**. Once you fix all the bugs in the program, your program must behave as follows:

- *Example 1*:

```
Do you have permission from the dean? [y/n] n
Do you have permission from your advisor? [y/n] n
Do you hold senior status? [y/n] y
How many credits do you have? 20
This student can graduate: False
```

- *Example 2*:

```
Do you have permission from the dean? [y/n] n
Do you have permission from your advisor? [y/n] n
Do you hold senior status? [y/n] y
How many credits do you have? 65
This student can graduate: True
```

- *Example 3*:

```
Do you have permission from the dean? [y/n] n
Do you have permission from your advisor? [y/n] y
Do you hold senior status? [y/n] n
How many credits do you have? 50
This student can graduate: True
```

These are, of course, not the only combinations possible, so give a couple more a try when debugging your program to make sure that you've covered all the possibilities!

Step 1: Run the program without changing anything

Before you even start trying to make a program look right, a good first step would be to run the program a couple of times in its current state and how it behaves. This will already tell you a lot, like:

1. Does the program crash?
2. If the program crashes, what kind of error does it give me and in which line is it?
3. If the program doesn't crash, does it always produce the same value?
4. If the program produces the same value, is it the value that you want?

And so on and so forth. The key here is to **observe** the program's behaviour. This is helpful because, once you start making your own changes to it, you can see if any of your changes

result in different behaviour (i.e the correct answer, or a different error. The latter often happens when there's several errors in a program, and fixing one allows the other to actually happen).

Let's run our current program a couple of times:

- *Run 1*:

```
Do you have permission from the dean? [y/n] n
Do you have permission from your advisor? [y/n] n
Do you hold senior status? [y/n] y
How many credits do you have? 20
Traceback (most recent call last):
  File "graduation_status.py", line 13, in <module>
    condition_one = accumulated_credits > 60 and has_advisor_permission
TypeError: '>' not supported between instances of 'str' and 'int'
```

- *Run 2*:

```
Do you have permission from the dean? [y/n] n
Do you have permission from your advisor? [y/n] n
Do you hold senior status? [y/n] y
How many credits do you have? 65
Traceback (most recent call last):
  File "graduation_status.py", line 13, in <module>
    condition_one = accumulated_credits > 60 and has_advisor_permission
TypeError: '>' not supported between instances of 'str' and 'int'
```

- *Run 3*:

```
Do you have permission from the dean? [y/n] n
Do you have permission from your advisor? [y/n] y
Do you hold senior status? [y/n] n
How many credits do you have? 50
Traceback (most recent call last):
  File "graduation_status.py", line 13, in <module>
    condition_one = accumulated_credits > 60 and has_advisor_permission
TypeError: '>' not supported between instances of 'str' and 'int'
```

Alright. It's pretty clear that we have an impassable `TypeError` on **line 13**. Generally speaking, **the last line of the error message will tell us what the most obvious error is**. This leads us to step 2.

Step 2: Find and fix the most obvious errors

Line 13 looks as follows:

```
condition_one = accumulated_credits > 60 and has_advisor_permission
```

In other words, a boolean expression. The last line of the error message reads as follows:

```
TypeError: '>' not supported between instances of 'str' and 'int'
```

What might this mean? Well, it tells us that somewhere in this line, the `>` (less-than) operator is being used on two operands of incompatible types (a `str` and an `int`). Well, 60 is very clearly an integer, which means that the string in question must be whatever value is being stored inside the variable `accumulated_credits`. This variable is defined on line 5:

```
accumulated_credits = input("How many credits do you have? ")
```

As we know, the `input()` function will **always return a** `str`. This is a problem here because `accumulated_credits` is supposed to store a numerical value (i.e. how many credits this student currently has). Using either the `int()` or `float()` function will hopefully take care of this issue:

```
accumulated_credits = float(input("How many credits do you have? "))
```

Once you have done this, we can move on to the next step.

Step 3: Re-run and observe

When debugging, any time you make a change to your code—regardless of how small—you should always re-run your program and see if it behaves any differently. If you perform too many changes at once without seeing how they affect your program's behaviour, you run the risk of either breaking your program even more, or potentially even fixing the error and adding another one.

Let's run our code now that we performed step 2:

- *Run 1*:

```
Do you have permission from the dean? [y/n] n
Do you have permission from your advisor? [y/n] n
Do you hold senior status? [y/n] y
```

```
How many credits do you have? 20
This student can graduate: False
```

Okay, this looks promising. Our `TypeError` is gone.

- *Run 2*:

```
Do you have permission from the dean? [y/n] n
Do you have permission from your advisor? [y/n] n
Do you hold senior status? [y/n] y
How many credits do you have? 65
Do you have permission from the dean? [y/n] n
Do you have permission from your advisor? [y/n] n
Do you hold senior status? [y/n] y
How many credits do you have? 65
This student can graduate: False
```

- *Run 3*:

```
Do you have permission from the dean? [y/n] n
Do you have permission from your advisor? [y/n] y
Do you hold senior status? [y/n] n
How many credits do you have? 50
This student can graduate: False
```

While our program is not crashing, however, we can see that the output is incorrect. The student in runs 2 and 3 should be able to graduate by our rule-set.

Now comes the hard part.

Step 4: Analyze your code from top to bottom

If you've reached a point where your program is no longer crashing, but it is giving you the incorrect output, then there is very little to do but to analyze your program part-by-part, and often line-by-line. This is where `print()` statements come extremely handy. Showing the result of different operations/expressions throughout the program is crucial to finding, diagnosing, and ultimately debugging your code. This may seem a bit like overkill in many situations, but when you reach a point where you have 300+ lines of code, it is very likely that you will end up missing some crucial part of the error by sheer virtue of there being *so much to look at*. Let's practice with our program here and see if we can't find what's giving us an error.

Our inputs looks pretty good, but just in case, we can check their values and their types using the `type()` function:

```
# Asking for user input
dean_permission_input = input("Do you have permission from the dean? [y/n] ")
advisor_permission_input = input("Do you have permission from your advisor? [y/n]
senior_status_input = input("Do you hold senior status? [y/n] ")
accumulated_credits = float(input("How many credits do you have? "))

print("dean_permission_input:", dean_permission_input, type(dean_permission_input
print("advisor_permission_input:", advisor_permission_input, type(advisor_permiss
print("senior_status_input:", senior_status_input, type(senior_status_input))
print("accumulated_credits:", accumulated_credits, type(accumulated_credits))
```

Output:

```
Do you have permission from the dean? [y/n] n
Do you have permission from your advisor? [y/n] n
Do you hold senior status? [y/n] y
How many credits do you have? 20
dean_permission_input: n <class 'str'>
advisor_permission_input: n <class 'str'>
senior_status_input: y <class 'str'>
accumulated_credits: 20.0 <class 'float'>
```

That all looks correct. Our permissions are strings either holding `'y'` or `'n'`, and our accumulated credits are indeed converted into a float equivalent. You can either commend or delete these `print()` statements now.

Next, we have to confirm that the `Student information` section is resulting in the correct `bool` values (i.e. `true` or `false`). We'll again make use of `print()` statements to check both the value resulting from the boolean expression, as well as their types (just in case they are not being stored as `str` values for whatever reason; you never know).

```
# Student information
has_dean_permission = dean_permission_input == 'y'
has_advisor_permission = has_dean_permission == 'n'
is_approved_senior = senior_status_input == 'y'

print("has_dean_permission:", has_dean_permission, type(has_dean_permission))
print("has_advisor_permission:", has_advisor_permission, type(has_advisor_permiss
print("is_approved_senior:", is_approved_senior, type(is_approved_senior))
```

Output:

```
Do you have permission from the dean? [y/n] n
Do you have permission from your advisor? [y/n] n
Do you hold senior status? [y/n] y
How many credits do you have? 20
has_dean_permission: False <class 'bool'>
has_advisor_permission: False <class 'bool'>
is_approved_senior: True <class 'bool'>
```

These, again, look pretty good. Dean permission was entered as `n`, meaning `False`, a `bool`. This is the same case for advisor permission, and senio status was entered as `y`, corresponding to `True`, another `bool` value. Looking good.

We'll now have to check our `Generating permission` section. You may have already figured that this is most likely where the error lies, as this section contains the most "complicated" logic in the entire program. Since we have three conditions for graduation, it would be helpful check that all three are being properly accounted for in the current boolean expressions.

The first condition is:

> They have accumulated 64 credits **and** are approved seniors.

In other words, `accumulated_credits` must be **greater than or equal** to 64, and `senior_status_input` must be `True`. `condition_two` seems to include the use of both `accumulated_credits` and `senior_status_input`. Let's take a look at it:

```
condition_two = accumulated_credits > 40 or is_approved_senior
```

So yeah, a couple of errors here. First of all, the number of accumulated credits must be greater than or equal to 64, not greater than 40. The second error here is the fact that we have an `or` operator instead of `and`. So, `condition_two` must instead look like:

```
condition_two = accumulated_credits >= 64 and is_approved_senior
```

Notice that I typed `is_approved_senior` instead of something like `is_approved_senior == True`. Both way are actually correct but, in most modern programming languages, adding the `== True` is repetitive. If the value of `is_approved_senior` will always be either `True` or

`False`, then you probably don't want to say `True == True` or `False == True`, since they just simplify to `True` and to `False`.

Anyway, that's our first rule. Our second rule reads as follows:

> They have accumulated 40 credits **and** have special permission from their advisor.

`condition_one` seems to be using `advisor_permission_input`. Let's take a look at it:

```
condition_one = accumulated_credits > 60 and has_advisor_permission
```

Yep, this one also doesn't look quite right. `> 64` should instead be `>= 40`. The rest looks okay.

Finally, we have:

> They have special permission from the dean to graduate, regardless of how many credits they have.

This one is a bit of a "tricky" one. This basically just means that if `dean_permission_input` is `True`, literally none of the other conditions matter. So, `condition_three` can be simplified from:

```
condition_three = has_dean_permission and accumulated_credits > 64
```

To simply:

```
condition_three = has_dean_permission
```

There's not a huge need to have a separate variable saving a simple value, but you are free to create it to keep things consistent.

Now, the line actually determining if this student is the following:

```
can_graduate = condition_one and condition_two and condition_three
```

This is incorrect. Remember that a student can graduate if they fulfill **any** of the conditions, not if they fulfill **all** the conditions. The correct operator would thus be `or`, since it will be `True` if **any** of the operands is `True`:

```
can_graduate = condition_one or condition_two or condition_three
```

Let's run our program now and see how it shakes up:

```
Do you have permission from the dean? [y/n] n
Do you have permission from your advisor? [y/n] n
Do you hold senior status? [y/n] y
How many credits do you have? 20
This student can graduate: False
```

```
Do you have permission from the dean? [y/n] n
Do you have permission from your advisor? [y/n] n
Do you hold senior status? [y/n] y
How many credits do you have? 65
This student can graduate: True
```

```
Do you have permission from the dean? [y/n] n
Do you have permission from your advisor? [y/n] y
Do you hold senior status? [y/n] n
How many credits do you have? 50
This student can graduate: True
```

Cool. All of our examples seem to work. This is a relatively simple example, but follow the same approach to debugging throughout the rest of the semester. The more you do it, the more used to it you will become and the more honed your instincts will become.

Solution.