Motivation

This week we took an in depth look at our first testing technique: *black box testing*. One of the key elements of black box testing is that the tester does not have access to the source. In this assignment, you will be given a specification and list of requirements. This will give you experience writing unit tests in a black box testing scenario.

Course Learning Outcome(s):

- **Apply** testing techniques, including black-box and white-box techniques, automatic testing activities, and regression testing (CLO 4)

Module Learning Outcome(s):

- **Apply** black box testing techniques

Description

For this assignment, you will be writing unit tests based on the following specification/requirements.

You will write a series of unit tests to test a function called `credit_card_validator` (written for you) that is passed a sequence of digits as a `string` that represents as credit card number. This function will return `True` if it is a valid credit card number, otherwise it will return `False`.

Depending on the credit card issuer, the length of a credit card number can range between 10 and 19 digits. The first few digits of the number are the *issuer prefix*. Each credit card issuer has an assigned range of numbers. For example, only Visa credit card numbers may begin with `4`, while American Express card numbers must begin with either a `34` or `37`. Sometimes, credit card providers are assigned multiple ranges. For example, MasterCard card numbers must start with the numbers between `51` through `55` or `2221` through `2720` (inclusive).

The last digit of the number is referred to as the *check digit* and acts as a *checksum*. Most credit cards calculate this *check digit* using the *Luhn algorithm* (see resources below for how this is calculated).

In order to limit the scope of this assignment, we are going to limit the number of credit card issuers to 3: Visa, MasterCard, and American Express. Each has their own prefixes and length requirements.

- **Visa**
    - Prefix(es): 4
    - Length: 16
- **MasterCard**
    - Prefix(es): 51 through 55 and 2221 through 2720
    - Length: 16
- **American Express**
    - Prefix(es): 34 and 37
    - Length: 15

Your task is to create a series of tests that attempt to reveal **bugs** in the implementation. As this is black box testing, you will not have access to the source so you must use what you have learned this week to generate test cases.

You will be submitting your code to Gradescope which will auto grade your tests. In order to get full credit on the assignment, you will need to locate *all* **6** bugs in the code (refer to the rubric for full details). Some are easier than others. Bug 5 is easy to miss without using Partition Testing and *Bug 6* requires using what you know about common errors to design your tests.

You are free to determine how you generate your tests cases. You may do it completely manually, or use an automated tool like the TSLgenerator. No matter how you generate your test cases, in your file testing file (`tests.py`), you need to include a comment that describes how you chose your test cases.

You also need to ensure you have test cases that do a good job covering the input domain. This means that at the very least, you need to have a test case for **each** of the prefix ranges listed above.

Finally, your test suite needs to be free of linting errors using the PEP8 standard; this will be important later when working on shared repositories. If you are unfamiliar with linting, please see the resources below. The easiest way to accomplish this is to ensure that there are no "squiggly" lines under your code in PyCharm (You will need to change PyCharm's default line length to 79 to match PEP8). You can also use the PEP8 Online tool below to copy and paste your code to verify it has no errors.

Hints

You will need to include

```
if __name__ == '__main__':
    unittest.main()
```

-
- It is best to only have a single assert in any test. Once one fails, the rest of the code in the test isn't executed.
- You may assume only strings of digits are being sent to `credit_card_validator`
- I used the TSLgenerator to create roughly 35 test cases and then picked some to break
- Use what you learned about *Error Guessing* and *Boundary Values* to find tricky bugs
- You will need to use `from credit_card_validator import credit_card_validator` in your `tests.py`

To ensure your tests are correctly importing the function for testing, you may put the following dummy code into a file called `task.py`

```
def credit_card_validator(num):
    pass
```

- You may submit as many times as you want to Gradescope to check how well your test suite performs

What to turn in

Submit to Gradescope your testing suite; it must be named `tests.py`

- This file will include *at the top* a comment describing your test generation methodology
- This file will contain tests that cover *all* prefix ranges
- This file will be free of PEP8 linting errors