**BCPR301 Assignment Two Documentation**

**Smell Detection**

**Smell #1**

**Name:** Large Class

**Location:** BCPR301 Assignment Two - controller.py - Controller - Lines 123 - 149

**Reasons:**

1. The controller in my project has more functionality than what is needed which has caused the class to become very large. Extracting these functionalities to other classes will give my controller a more specific purpose and responsibility.
2. Having a large controller class decreases the readability of it, which makes it more difficult to understand and maintain. Keeping my code clean will help to ensure the controller class has these attributes.

**Strategies/Approaches:** As the class I am refactoring is my controller class, it still needs to have the brains behind my program but it currently has too many responsibilities. Each the methods (display, load, save, pickle, welcome and validate) have too much code in their functions which is adding too much functionality to the controller class. To fix this I will be extracting all of this information into other classes, which will simplify the controller and bring those extra responsibilities to the individual classes that will be created. This will make the controller class a lot easier to read and refactor in later versions.

**Smell #2**

**Name:** Switch Case Smell

**Location:** BCPR301 Assignment Two - display.py - Display - Lines 123 - 149

**Reasons:**

1. The display function in my method will always be called but the output will differ depending on the type of display the user needs. Applying polymorphism allows very similar conditional statements to disappear, which gets rid of duplicate code.
2. Polymorphism uses subclasses to add a varied output when called. In my case, if I want to create a new way of displaying the data, I can easily just create a new sub class to change what occurs when the same abstract method is called. This makes adding new code very simple while also maintaining a high code readability.

**Strategies/Approaches:** To refactor my repetitive conditional statements, I will have to create a base class with an abstract method inside of it. For each conditional statement that I had before, I will create a subclass that inherits the base class previously created. Now whenever a specific display type is needed, the correct subclass can be called using the same abstract method created earlier. This will provide the specific output that the user needs.

**Smell #3**

**Name:** Long Method

**Location:** BCPR301 Assignment Two - validater.py - Display - Lines 60 - 113

**Reasons:**

1. Having a long method decreases the readability as it makes it difficult to understand what is being accomplished. Splitting it into smaller methods will make it easier to understand what that smaller group of code is accomplishing
2. Long methods increase complexity as more and more statements are being added to one section of code. This makes it harder to read and refactor the code later on down the track when changes are needed as full knowledge of the large method needs to be known to make correct changes.
3. In my case, it adds many conditional statements to one big method, which starts to make another bad smell, “conditional complexity”, become more of an issue. By splitting it into smaller statements helps to reduce the complexity as there are less conditions to comprehend in the section of code.

**Strategies/Approaches:** The check\_data method checks data depending on values to do with the row item in the row. This means the checks done for each item or row can be put into separate methods to shorten the length of the main method. I sections that I will be splitting my code into smaller methods will be:

* When the row of data has a correct length, call a method that receives that valid row and then checks if the user ID has been repeated or not.
* This data is then sent to another method that checks if each item complies to the regex.
* If the item in the row is the date index, send that to a date checker to check its regex and age validity.

**Smell #4**

**Name:** Switch Case Smell

**Location:** BCPR301 Assignment Two - load.py - Load - Lines 7 - 100

**Reasons:**

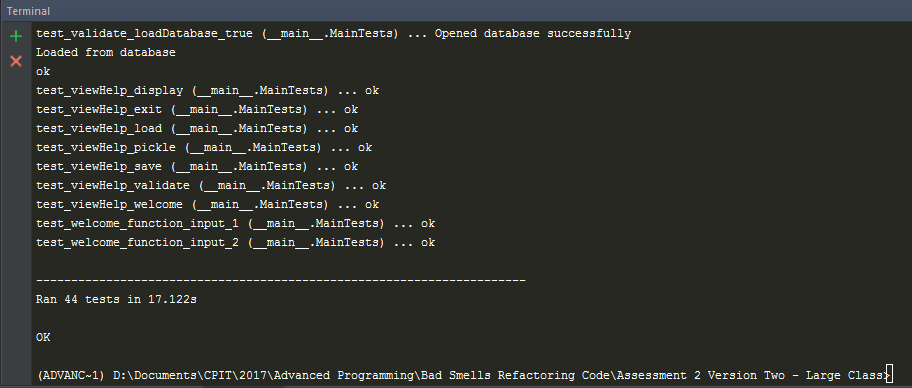
1. Similar to the old display method, the load method will have a different output based on a set amount of possible inputs. This was originally solved with switch statements but this used duplicate conditional statements that can be solved by using polymorphism.
2. By changing my code from using switch statements to polymorphism, I will easily be able to create a new specialised sub class from LoadData each time a new load type needs to be added.

**Strategies/Approaches:** Just like my Display method, I will be using polymorphism to refactor this affected area of code. A base class called LoadData will be created with an abstract method called load\_data(). Specialised sub classes will be created for each type of varied output that is needed from the program. These will be selected by using the users input as a key value for a dictionary containing each of the sub classes then that specific sub class will be run.

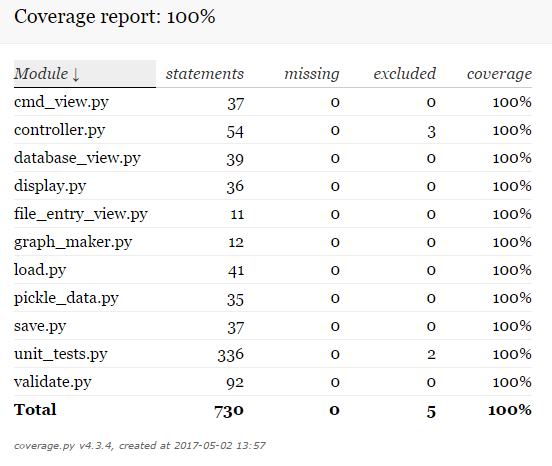
**Test Development**

For the PEP8 checks, I used <http://pep8online.com/>

**Smell #1 - Long Class**

**Unit Tests Running:**

**Coverage Report for Full Program:**



**Test coverage in affected file:**

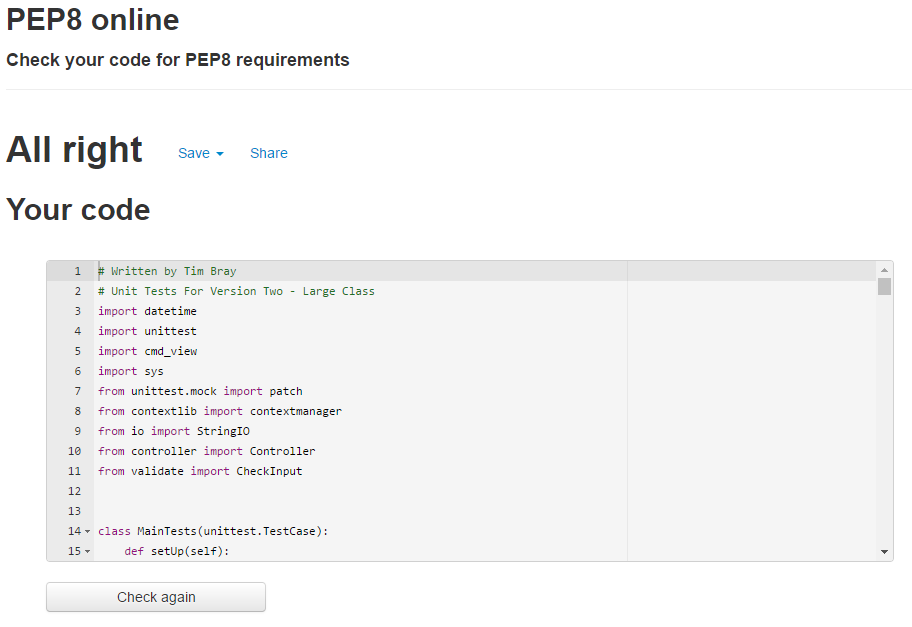


The three excluded lines in controller and validate are:

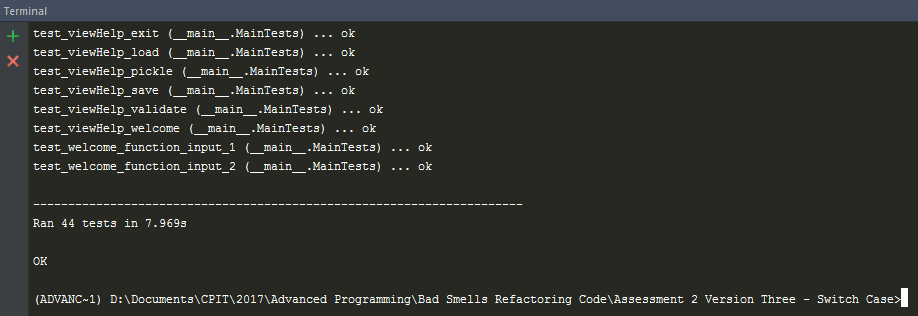


These lines are excluded in a .coveragerc file as it just creates an instance of controller and starts go() (the main method). The go() method has been tested with other unit tests to ensure it is fully functioning so all of its contents are still being fully tested.

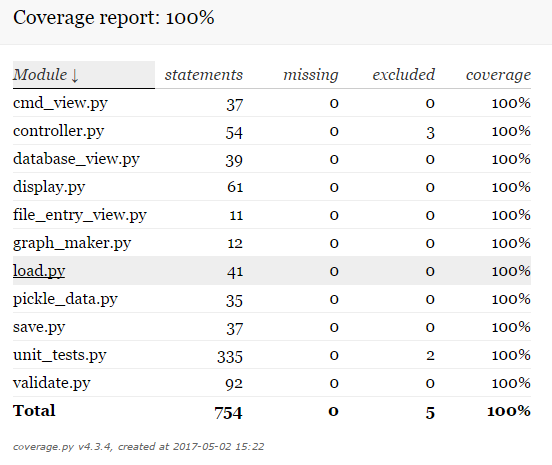
**Passing PEP8:**



**Smell #2 - Switch Case - Display**

**Unit Tests Running:**

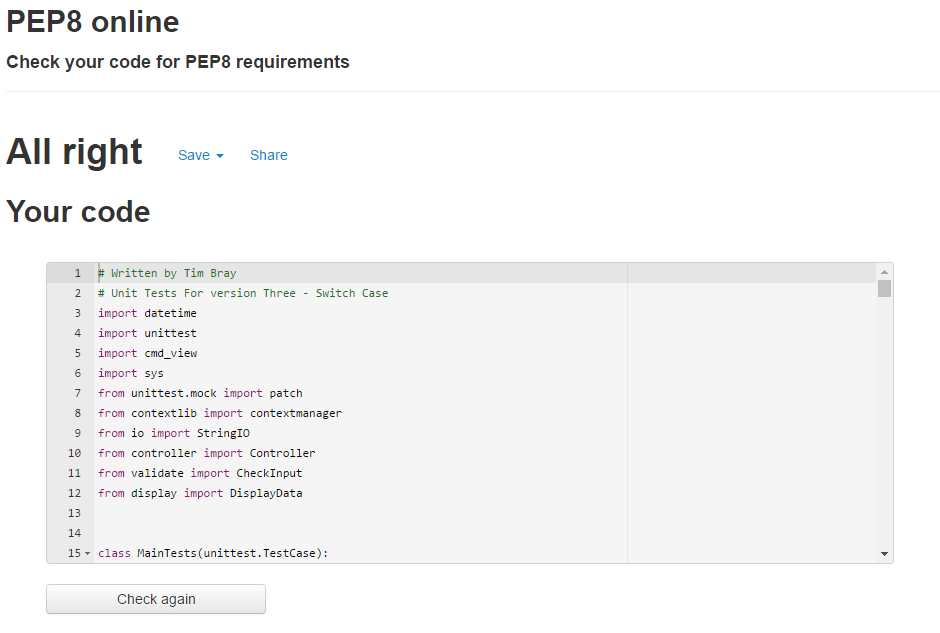
**Coverage Report for Full Program:**



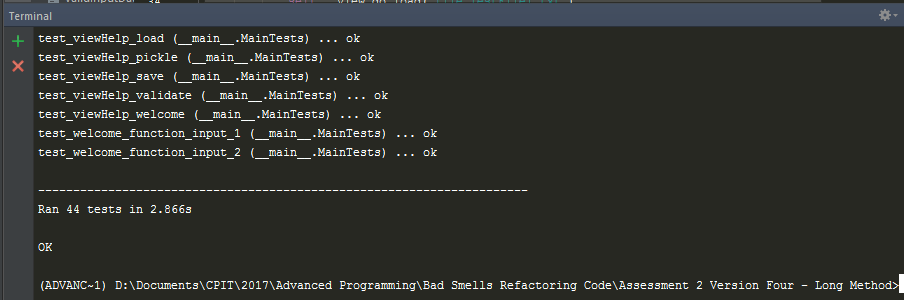
**Coverage for affected file:**



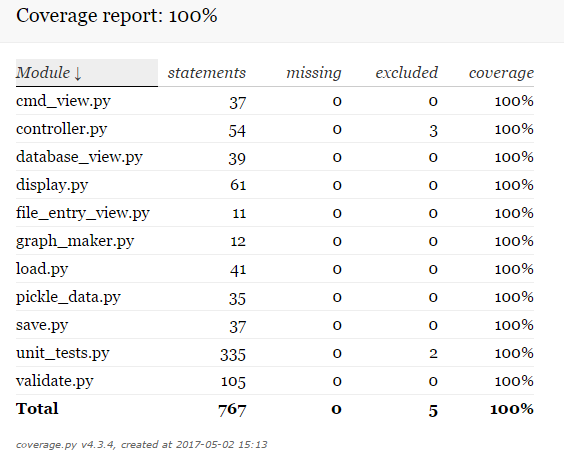
**Passing PEP8:**



**Smell #3 - Long Method - Validate**

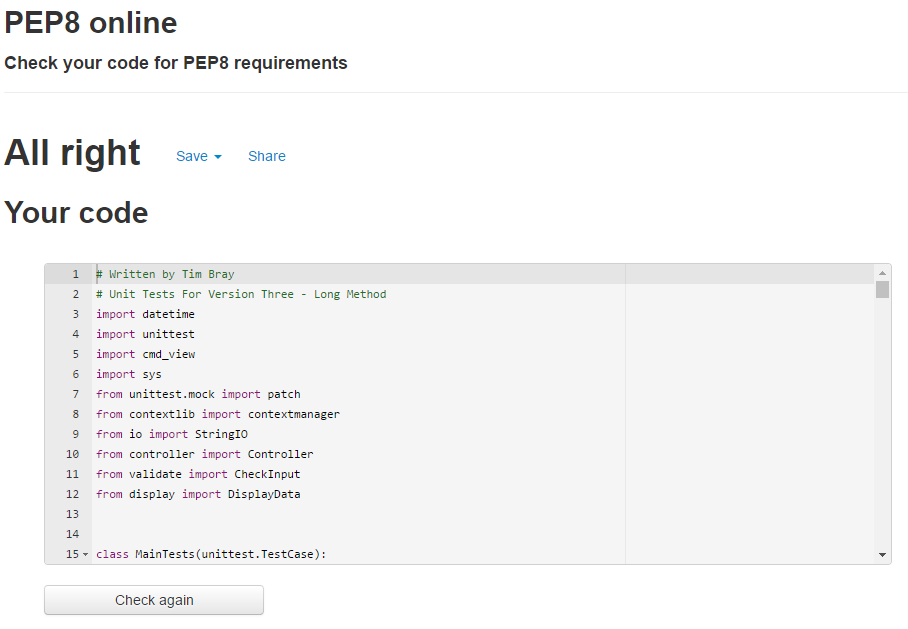
**Unit Tests Running:**

**Coverage Report for Full Program:**

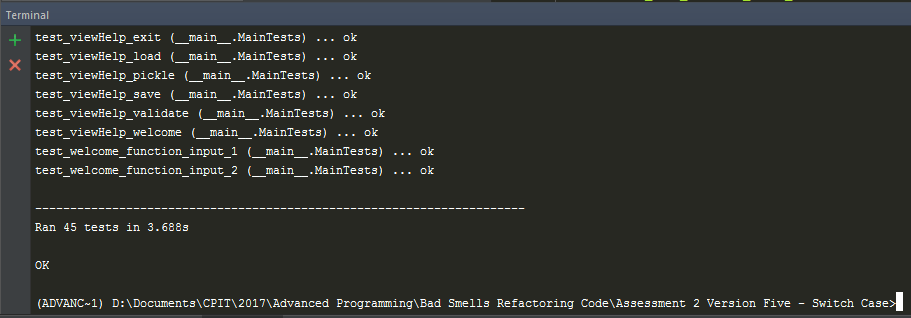


**Coverage for affected file:**

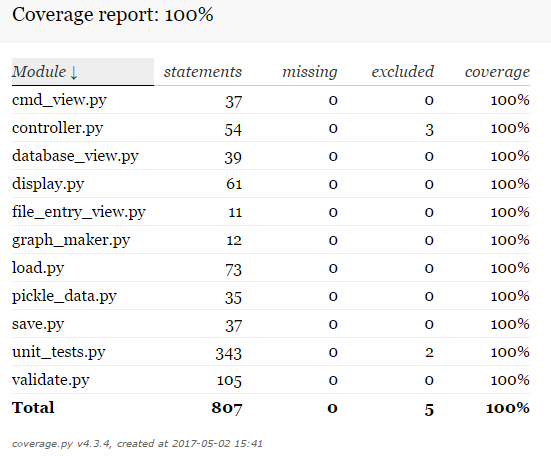
**Passing PEP8:**

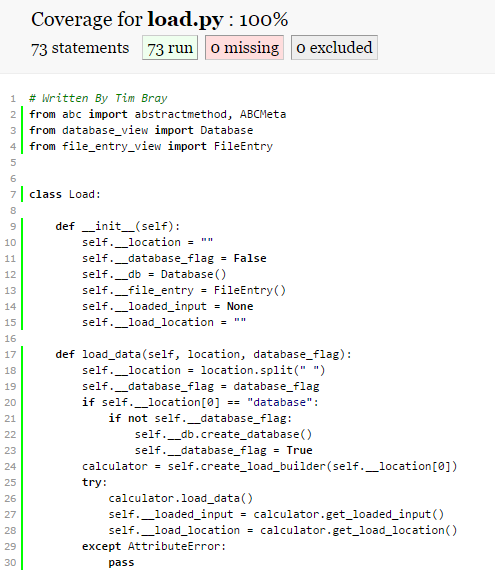


**Smell #4 - Switch Case - Load**

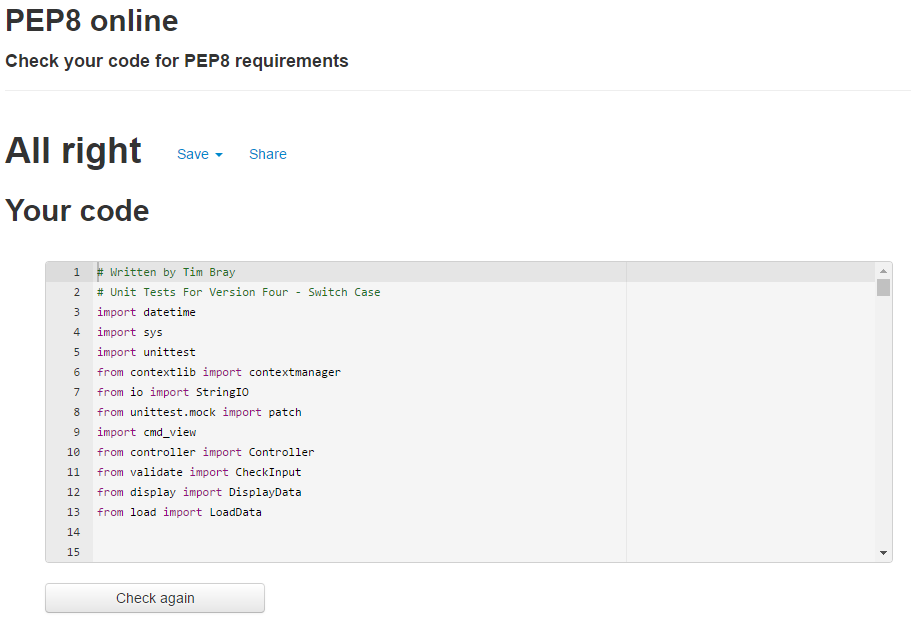
**Unit Tests Running:**

**Coverage Report for Full Program:**



**Coverage for affected file:**

**Passing PEP8:**



**Refactoring**

**Smell #1 – Large Class**

**Why Which Smell Was the Worst**

The worst smell that I have identified in my code is my large controller class. I believe that this was the worst smell as it had the largest amount of logical lines of code and also because the controller class is used every time the user inputs something into the CMD. Due to it being the controller class, each time a new feature is added, the class became longer and longer over time. This heavily reduced the readability of the class and made it more difficult to maintain. Extracting classes out of it helped to reduce the logical lines of code inside of it and will make it much easier to add any new code when needed.

**Version Control**

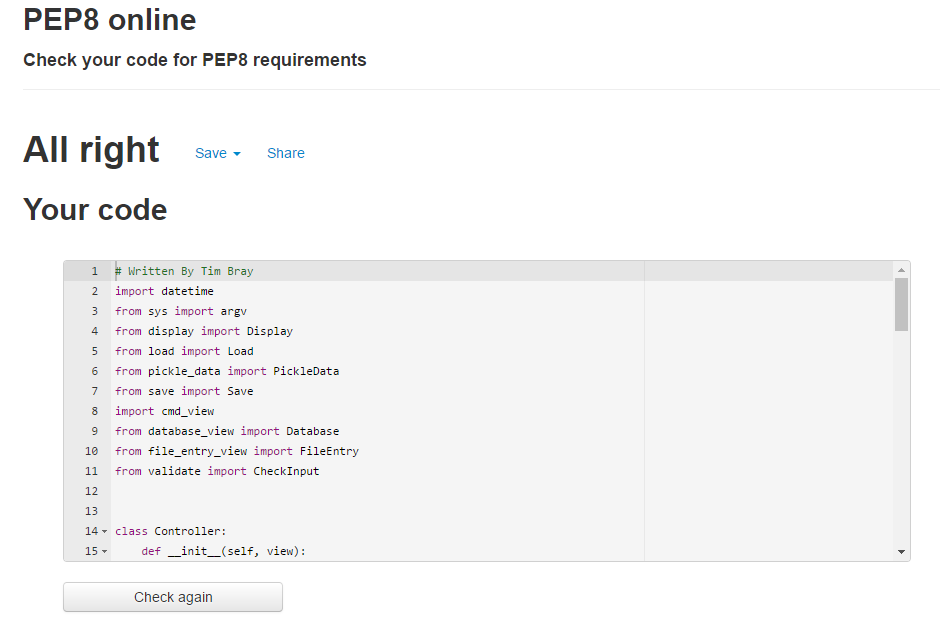
Version control was completed using a GitHub. Link: <https://github.com/TimBray2/BCPR301-Assignment-Two/commits/master>

**Modifications Made and PEP8 Compliance**

GitHub Link to version branch: <https://github.com/TimBray2/BCPR301-Assignment-Two/commits/Version-Three---Switch-Case-%231>

This shows the modifications done in the Controller class and the creation of the new classes.

**PEP8 Compliance:**



**Evaluations**

I believe that I successfully fixed this bad smell with the use of extracting classes to lower the amount of responsibilities the controller class has. Extracting the classes from the controller class allowed me to move the logic to a different class to lower the Controller’s responsibilities and complexity. This made the logical lines of code in the controller class heavily decrease, making it much easier to read and maintain. Although I did fix the bad smell, I do think that I could have fixed it better. Implementing the extracted classes as sub classes of the controller would have been a better way to fix the smell.

**Smell #2 – Switch Case**

**Why Which Smell Was the Worst**

**Refactor #2**

The next biggest smell that I have come across since my last refactor was the switch case statements in my display method. I was using several conditional statements to check whether the input was one of the four possible valid inputs before executing specific code. This was adding a lot of duplicate code to my program, which was ultimately decreasing the readability of it as well.

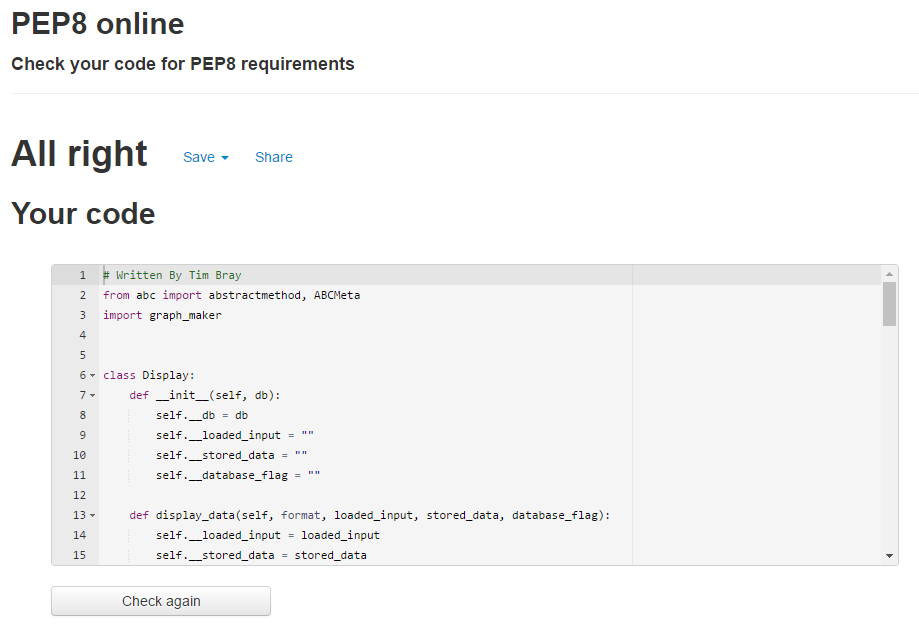
**Version Control:** Version control was completed using a GitHub. Link: <https://github.com/TimBray2/BCPR301-Assignment-Two/commits/master>

**Modifications Made and PEP8 Compliance**

GitHub Link to version: <https://github.com/TimBray2/BCPR301-Assignment-Two/commits/Version-Three---Switch-Case-%231>

This shows the modifications done in the Controller class and the creation of the new classes.

**PEP8 Compliance:**



**Evaluations:**

Using polymorphism on my switch case statements allowed me to successfully fix my switch case bad smell. By converting my switch case statements to a dictionary that has values of specialised sub classes, I was able to correctly implement a more efficient way of dealing with a set number of entries. This allowed me to get rid of my duplicate conditional statements and made it easier to add new possible entries just by adding more sub classes to the base class.

**Smell #3 – Long Method**

**Why Which Smell Was the Worst**

The next worst smell that I had found was my long method, check\_data(), in my CheckEntry class. This class consisted of many logical lines of code with a lot of conditional statements and loops. That made it very difficult to read and maintain the method whenever changes were necessary. Shortening this method will help improve these issues and get rid of the smell.

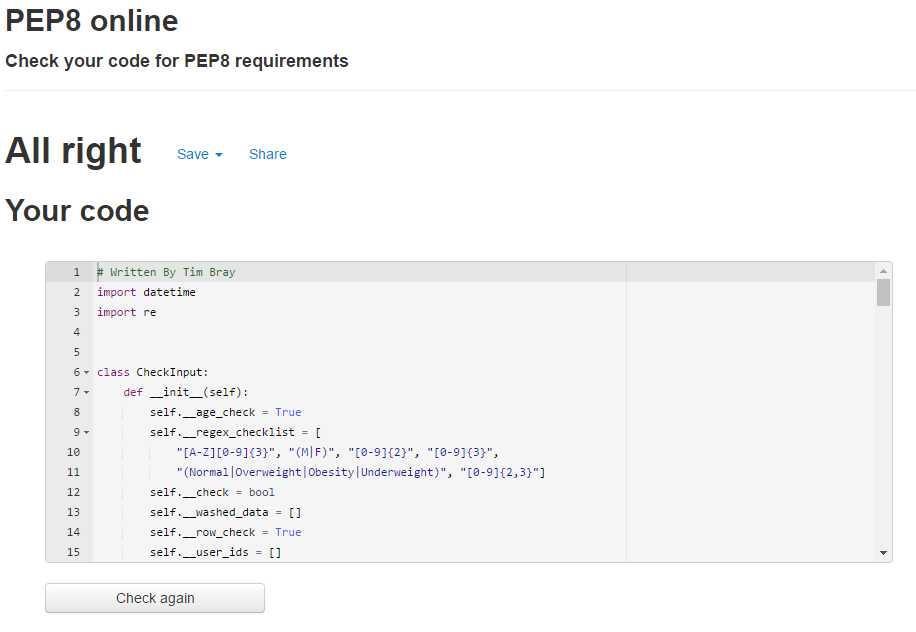
**Version Control:** Version control was completed using a GitHub. Link: <https://github.com/TimBray2/BCPR301-Assignment-Two/commits/master>

**Modifications Made and PEP8 Compliance**

GitHub Link to version: <https://github.com/TimBray2/BCPR301-Assignment-Two/commits/Version-Four---Long-Method>

This shows the modifications done in the Controller class and the creation of the new classes.

**PEP8 Compliance:**



**Evaluations:**

I believe that I successfully removed the long method “check\_data” by extracting smaller methods out of it. Doing this allowed me to create five individual methods that had their own individual purpose. This benefits the program by increasing the readability and decreasing the complexity of the method. Troubleshooting any future errors will also become easier as I will be able to narrow down where the error will be occurring much easier than if it was in one long method.

However, I do believe I could have fixed the bad smell in a better way as I have created another bad smell after fixing the old one. The code now has a message chain bad smell since check\_data will eventually be calling check\_date through three other methods. This could be fixed using a hide delegate later on.

**Smell #4 – Switch Case**

**Why Which Smell Was the Worst**

The next worst smell that I had come across was another switch case statement. I used similar switch case statements in my load\_data() method where I was checking results against a set number of valid inputs. This was again adding duplicate code and made it very messy when I needed to add new entry types. There was not as many valid inputs as the display method but the load function is a heavily used function so it made the smell worse.

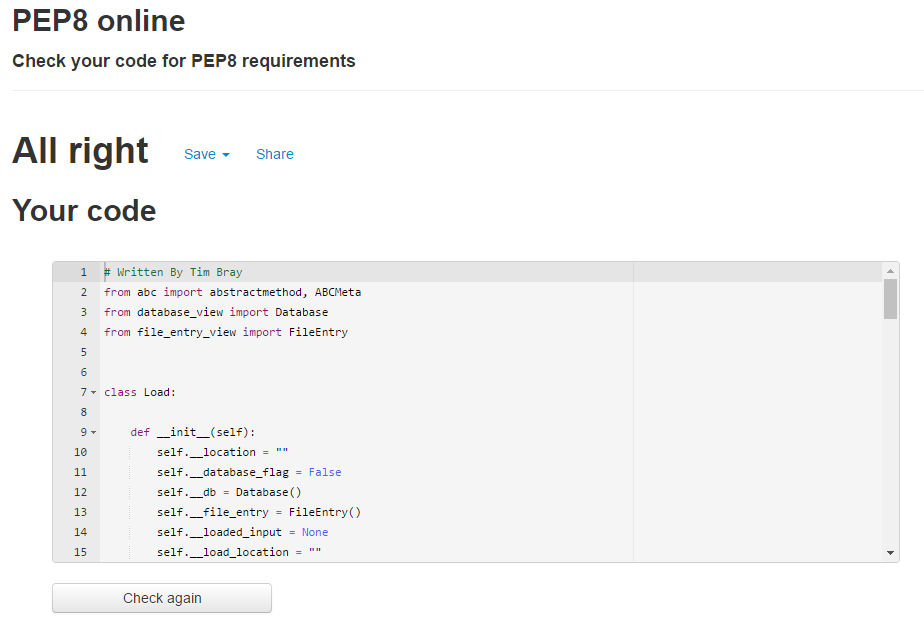
**Version Control:** Version control was completed using a GitHub. Link: <https://github.com/TimBray2/BCPR301-Assignment-Two/commits/master>

**Modifications Made and PEP8 Compliance**

GitHub Link to version: <https://github.com/TimBray2/BCPR301-Assignment-Two/commits/Version-Five---Switch-Case-%232>

This shows the modifications done in the Controller class and the creation of the new classes.

**PEP8 Compliance:**



**Evaluations:**

I believe that I correctly applied polymorphism to my load\_data method to get rid of a switch case bad smell. Instead of using if statements to check against a set number of inputs, I now correctly use a dictionary to call a specific sub class. This makes it much easier to maintain my code as errors will be easier to locate and extra valid inputs will be easy to add to the program. I do think that this smell is not an extremely bad smell due to the possible valid inputs being small; however, this means there will be no switch case bad smell if new load functions were to be added to the program. Adding more valid inputs with my previous code would have made the code very hard to maintain.