|  |
| --- |
| University of Sunderland |
| Python Code Quality Check |
| An overview of looking at my own personal code, to see if I can provide examples of Single Responsibility, WET, DRY, and any Stylistic issues. |

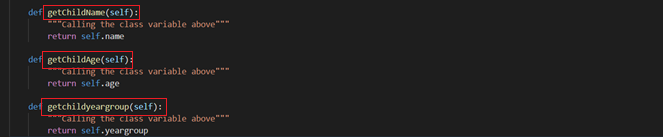
|  |
| --- |
| Heslop, Alan John  01/11/2020 |

### Code Quality: Improvements

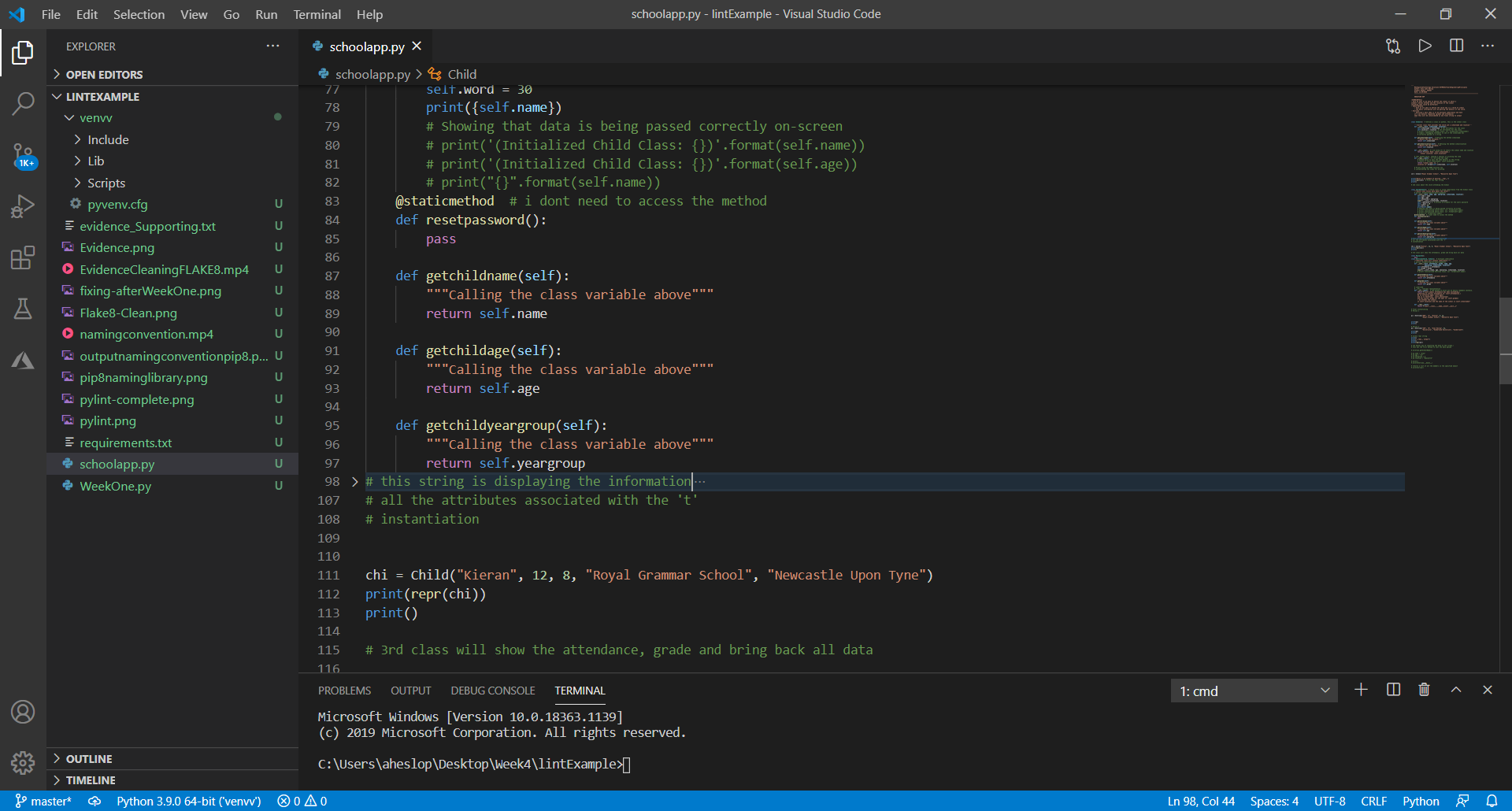
Code can run with errors and without, however you can always improve on the code and make it “cleaner”. By having cleaner code, you can reduce issues both stylistic and future issues, and conforming your code to the PEP8 Standard can leave your code easier to understand.

### Stylistic Issue

As you can see below, I was presented with errors relating to the naming conventions for stylistic issues. The problem with this is it runs and works as it should however when ran against a “linter”, issues can arise with the formatting and analysis of the application in question.



In order to fix this issue, I researched the naming conventions suitable to pass a linting test, and then re-formatted my code to mean it look “clean”. The fix was simply taking the code out of “camel case” and changing it to “lower case”, as below:



### Single Responsibility Principle

The **S**ingle Responsibility Principle — Classes should have a single responsibility and thus only a single reason to change. (<https://medium.com/@severinperez/writing-flexible-code-with-the-single-responsibility-principle-b71c4f3f883f>)

Text

Description automatically generatedIn the screenshot below I have created a password class which inherits its details from the Child class resetpassword function. I can now create a whole set of functions to determine how to act on the password class.

### WET Code

Wet code means ‘Write Everything Twice’ and is fundamentally the case of repeating yourself

In the screenshot below, I have instantiated the code twice meaning I’ve created 2 versions of the “Child” Kieran – This is not needed. I can correct this error by removing line 111-113, due to already giving a final instantiation on line 156 for Child Kieran.

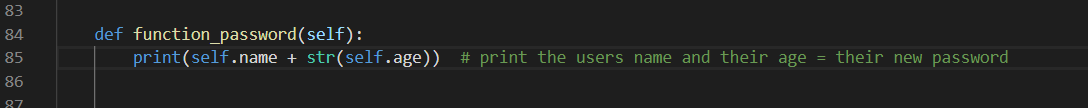
Text

Description automatically generated

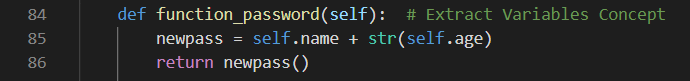
### Extract Variable Concept

Extract variables is also known as ‘Refactoring’, is a way of improving the design of the code after it’s been written. Sometimes you may enter more code than you start off with, however the code has been refactored and can save time in the future due to being easier to maintain and read.

At present this code below could be a prime example of using the extract variable concept, by formatting the code to be easier to read and understand.



This is the result of how the extract variable has been assigned to **newpass**, basically I have assigned the self.name +str(self.age) to the variable newpass and then returned it below. This means it is now stored in memory and can be called on easily and is much easier to understand.



### Extract Method Concept

The extract method function can give the developer an understandable conversion from a standard print to a new function that does the same but easier to understand.

Let’s say I want to create a function that displays the child’s name on top of their output, showing that the child belongs to that specific data.

In step 1, I have indicated yellow boxes, as to where I would like the child’s name to be apparent.

Step 1: Identify what you require | Step 2: Create a statement | Step 3: Implement an Extract Method

Text

Description automatically generated Text

Description automatically generated Text

Description automatically generated