# PyTeaser

## Proposed Change

There are two significant changes that I propose to make to the PyTeaser project. Both of these changes will significantly add to the usability of the project in modern systems, as well as providing a strong planform for future development. The changes are updating from Python2 to Python3 and to restructure the package into an object oriented design. Both of these changes will modernise the project. Helping promote new developers to take on a well architecture project running on the latest platform available to the community.

### Updating from Python2 to Python3

PyTeaser was written in Python2, along with all the libraries that it took advantage of. Pythons2s official end of life has been set at the 1st January 2020. This is because Python3 has been available as a replacement since 2008. Python3 is updated regularly with features and fixes, where’s Python2 only get key bug fix updates. In order to modernise the system one of my proposed changes is to port the existing system to Python3. Along with this, also making use of any of the dependencies that have also been ported to Python3. By achieving this the whole project was able to be build and run on a Python 3.6 interpreter. By porting the project to use Python3, it shows that the project is properly updated with performance and general functionality updates. Also going as far as to show this project as something worth looking at if someone is looking for a solution in the domain.

### Object Orientation

The second proposed change is to update the project to take full advantage of object orientation in Python. Currently, the software is very linear as it is. It doesn’t take advantage of object at all and as a result just looks like a bunch of methods that get called in the right order. By splitting the software up into appropriate objects, the whole system can clearly be shown to use different components and interact in a clear, appropriate manner. Identifying the different components with objects makes it much clearer to other developers what each part does. This helps on multiple fronts, it will be much easier for someone to come in and write tests for the system. For both unit and acceptance tests. This is because the components can be properly tested in their individual functionality. Issues to do with project updates can be localised to a certain component, making it much clearer where a bug has been introduced. Using this class structure, components can also be swapped out and tested with either test components, to see if they are viable, or mock components, in order to test for specific action in different scenarios. Separate components also make is clear as to what development areas there are. For example, to make the scoring more efficient a new developer should only need to make changes to the scoring class. When a pull request is made, the changes are very localised, and their intentions are clear.

## Description of Changes

### Updating from Python2 to Python3

The main difficulty with porting Python packages from Python2 to Python3 are the dependencies on external libraries. Python has a built-in module known as setup tools which can be used to download and use the latest available package dependencies. The issue is that a lot of packages written for Python2 and not compatible for Python3. Fortunately, each of the packages dependencies have already been ported to Python3, so the only problem needed to be solved was finding the new Python3 versions and adding them to the modules setup script. In terms of porting the package itself, the process is well documented in the python docs: <https://docs.python.org/3/howto/pyporting.html>. Unfortunately, due to one of the package dependencies, there is no longer support for this package on Python2. The best-case Scenario would result in the package being supported on both Python2 and Python3.

### Implementing Object Oriented Design

## Evaluation