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 platform 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. The reason behind this is that one of the core packages called Goose. Goose is a Python2 package, and instead of being compatible with both Python2 and Python3, its developers opted to make a new package for Python3. Because of this, only one version can be supported without separate imports dependant of the interpreter version.

Implementing Object Oriented Design

The entire structure of the package has been changed. The new structure follows standard design patterns and promotes proper implementation of components within the package. Previously the package just used a completely linear style, quite pythonic, but that does not conform to modern software design standards. This package use no object oriented design principles.

The package was completely reworked to follow a clear object oriented design. This split the package into three core components. The Summerizer, which handles the initial input of the user and binds the package together. The Scorer, responsible for managing how words/sentences are rated and weighted for choosing what is to be in the final summary. And lastly the text_parser. This is responsible for splitting the different articles up into different parts which can be used for scoring. These three components are their own classes and now have very clear, defined ways communicating with other components.

The changes to the structure of the package are consistent with standard, object oriented design principles. By using widely accepted standards there should be no reason for anyone to contest the changes. Especially as there are no significant downsides to adopting an object oriented design in python. These changes should help new developers learn the proper practices for writing software in python and there is a clear structure between the components. This will make it clear to new developers how everything is communicating and makes it very easy to replace components, as well as to add new features.

Evaluation

The main concern affected is the modernisation and the component design of the project. There has been a big drive to move package off Python2 because of its legacy nature. Having a project still written to use it doesn't set a good precedent for its users, and would potentially put people off using the project. By updating the package to Python3, it shows

that it is made to work on current systems and sets a precedent for being up to date and functional. Previously there was no component design, no use of object orientation, no real structure to the package. With the package in this manner, it was much harder for others to pick I up and make their own contributions. This was due to the steep learning curve required to come to grips with what the package was actually doing in its different methods. Let alone how it was working. By properly structuring the package, it will become much easier for developers to gain a detailed understanding of how the package works. With a clear indication of what components communicate and in what way. Its sets a precedent for good design patterns that will hopefully continued throughout the packages development.

Due to the nature of the changes, almost every stakeholder will be affected. Developers will be affected due to the dramatic changes in the packages structure and layout. Testers will have a new API to talk to, requiring them to change tests to suit. Users of the package will also need to issue minor updates to their usage of the package as the API has changes slightly

From a developers perspective, the changes have greatly improved the readability and structure of the project. It is clear and well defined what the roles of each component are and how it fits into the system. This will make expansion and modification much easier. The changes have been made by following good design practises in order to present a system with high cohesion and low coupling. Probably the biggest improvement will be for any testing working on the project. Previously, the package was difficult to test and there were no real components. You could not accurately test as there was too much going on when attempting to only test a single method. By splitting each component up properly, then each component can by individually tested, as well as have proper acceptance tests. It is key for a tester to be able to understand the flow of the system in order to properly test it. And with these design patterns in place, it will be far easier for a new tester to contribute effectively to the project. Some users will not necessarily get an improvement from this architectural change. Primarily to do with porting from Python2 to Python3. Users that require Python2 support will not be able to make use of the new changes if they are not able to update their current system to Python3. This can be solved with one of two solutions, someone adds support to use the old Goose package for systems running Python2. Or the project owner create a new, separate package to support the Python3 changes. To the end user, the structural changes do not change much, they will have to make small changes to how they communicate with the library but using objects gives them a large amount of more functionality. This is due to how they can manipulate an existing object and gain information and about different components at runtime.