In this problem set 6, we built a program to monitor news feeds over the internet. It filtered the news, alerting the user when it notices a news story that matches the user's interests. Concepts used from Python were to implement classes and their attributes, understanding class methods, understanding inheritance, telling the difference between a class and an instance of that class, and utilizing libraries as black boxes. The purpose of doing this project was as follows. Many web sites have content that is updated on an unpredictable schedule. News sites such as Google News, are a good example of this. One tedious way to keep track of this changing content is to load a website up in your browser, and periodically hit the refresh button. This process can be streamlined and automated by connecting to the web site's RSS feed, using an RSS feed reader instead of a web browser. An RSS (Really Simple Syndication) reader will periodically collect and draw your attention to updated content. An RSS feed consists of (periodically changing) data stored in an XML-format file residing on a web server. For this project we did not need to know XML or how to access these files over the network. We used a special Python module to deal with these lower level details. The higher level details of the structure of the Google News RSS feed was our focus of this problem set. URL: http://news.google.com/?output=rss. If we loaded this into a browser, we would see the browser's interpretation of the XML code generated by the feed. You can view the XML source with the "View Page Source" function. Abstractly, when you connect to Google News RSS feed, you receive a list of items. Each entry in this list represents a single news item. In a Google News feed, every entry has the following fields: guid = globally unique identifier for this news story, title, subject, summary, link. The goal was to create an application that aggregates several RSS feeds from various sources and can act on all of them in the exact same way all in one place. So we needed a unified, standard representation to store a news story. Parsing is the process of turning a data stream into a structured format that is more convenient to work with. We were given code that retrieved and parsed the Google and Yahoo news feeds. Parsing all this information from Google, Yahoo, New York Times, etc, gives us no small feat. To tackle the easy part of the problem, we pretended that someone has already done the specific parsing, and has left us with variables that contain the following information for a news story: guid, title, subject, summary, link to more content (all strings). We had to store this information in an object NewsStory. The NewsStory object had methods getGuid, getTitle, getSubject, getSummary, getLink, and each method returned the appropriate element of the instance. The next step was given a set of news stories, our program had to generate alerts for a subset of those stories. Stories with alerts were displayed to the user and others discarded. Alerting Rules were known as triggers. A trigger is a rule that is evaluated over a single news story and may fire to generate an alert. It could match a word in a title, summary, for many news stories. A more specific trigger would fire for 2 words that match in a summary. To simplify our code, we used object polymorphism. We defined a trigger interface and then implemented a number of different classes that implement that trigger interface in different ways. Each trigger class implemented the evaluate method that takes an news item (NewsStory object) as an input and returns True if an alert should be generated. The Trigger class was not directly used and it threw an exception if anyone tried to use it. Subclasses inherited from the Trigger class and they had to write their own evaluate method, but if they did not, the super class Trigger from which it inherited would use its evaluate method and throw an exception. The Trigger class inheritance was as follows. WordTrigger , NotTrigger, AndTrigger, OrTrigger, PhraseTrigger inherited directly from the super class Trigger. TitleTrigger, SubjectTrigger, SummaryTrigger inherited from WordTrigger's class. We used the split method to split strings on a specified character such as a punctuation mark, and the replace method to put something else in a string's spot instead. We used lower and upper on the strings to make them all the same case. These triggers were specified in the Python code and to change them, you have to edit the code, not a user-friendly way of doing it. To improve, you have a trigger.txt file read in each time the code starts, and the triggers are specified there.