Summary: This is a rough draft of the overall function breakdown of the project. Each function lists the expected inputs, the return values, and a rough description of what the code should accomplish. The exact details of the program are left up to the person assigned to write the function. The purpose of this document is so that if you need to call one of these defined functions in your code you can assume it will follow this format.

Title: fetchWeb
Inputs: 1. the url of the page to be fetched 2. A Boolean value telling whether images should be loaded
Description: This function will fetch the web page of the provided url from the internet with or without the image data. If successful it will load the retrieved data into a CacheObj and return it, otherwise it will return None.
Return: CacheObj or None

Title: fetchCache
Inputs: 1. the url of the page to be fetched 2. A Boolean value telling whether images should be loaded
Description: This function will fetch the web page of the provided url from the Cache with or without the image data. If successful it will load the retrieved data into a CacheObj and return it, otherwise it will return None.
Return: CacheObj or None

Title: storeCache
Inputs: 1. The CacheObj to be stored
Description: This function will store the provided CacheObj in the cache. If there is already an existing copy of the file in the cache it should rename that file and store a new copy.
Return: True if successful or False if it has failed for some reason

Title: compare
Inputs: 1. CacheObj1 2. CacheObj2
Description: Compares the two provided CacheObj and returns if a difference exists.
Return: True if they are the same and False if they are different

Title: notify
Inputs: 1. url of the page that has shown a change
Description: This function will throw a notification to Jamey's program
Return:

Title: HTTPFetch
Inputs: 1. url of the page to be fetched
Description: This program will attempt to fetch the page from the cache and return if successful (This may require checking a list of outstanding notifications and flagging them either as valid or invalid). Otherwise it will fetch the page from the web and store it in the cache. Then return the CacheObj
Return: CacheObj of the fetched page

Title: newComicCheck
Inputs: 1. A comic id of the site to be checked.
Description: this site will fetch each of the last 3 pages of the comic from the web and from the cache, compare the two and send a notification if they are different.
Return: None

Title: historyCheck
Inputs: 1. A url or a set of urls to be checked
Description: This function fetch the url from the web and from the cache, compare the two and send a notification if there is a difference. (Depending on our decision it may or may not store the data in the cache)
Return: None

Title: predictor
Inputs: 1. Some type of update information (what exactly this is remains to be determined)
Description: This function will use the provided information to update the list of comics to be checked when. The schedule of new comics to be checked should be maintained either in a file or in the database if we decide to have one.
Return: none

Title: scheduler
Inputs: none
Description: This function will run the list of new comics to be checked at a given time as well as call the history checks when new comic checks are not being run.
Return: none

Questions:

1. What form should Boolean values take in our code? Options include the tradition true/false, 1/0 or several others. Keep in mind that Python treats everything except None, False, a zero, or an empty sequence as being true.
2. How should we store the Cache? A database provides some options regarding additional information we can store as well as interesting sorting options. However it adds quite a bit of complication and makes compression more difficult. Another option is to store the pages directly to disk using a file system based on the url. The server name would be the root directory with whatever subdirectories are specified and finally the filename provided in the url. This avoids keys and a lot of other complications, but would possibly make the storage and retrieval code a bit more complicated since it would need to decode the url. This seems like it might be a worthwhile trade-off to me.
3. Do we want to update the cache when we find a changed page in the history, or do we want to throw a notification and wait for Jamey to request the page if he finds that the change is a valid one? The former requires less interaction and simplifies the HTTPFetch code, but the later will probably make for less redundant pages in the cache history. It would require us to keep track of what history changes we have notified Jamey of. This is probably a good decision so we can use this data to update the predictor so it doesn't get inaccurate information.
4. How are we going to define a comic id? If we use a database it would be the key, but how is that determined. In a file system we might want to store the last 3 pages separate from the rest of the history and use the directory as the id. But how do we tell which are the last 3 pages?
5. Exactly what information do we need to give to the predictor in order to keep it as accurate as possible?
6. How should the predictor maintain the list of pages that should be updated when?