README
==========

Author: Caleb Smith

Student ID: 1027644

November 9,2020

==========

Summary of file Structure:
There are 6 source files in this project:
    sandpile.py
    cylindrical.py
    hourglass.py
    tests.py
    main.py
    sandpilenumba.py

sandpile.py
==========
The file sandpile.py contains the base class for open boundary conditions BTW sandpile. Its constructor SandPile can be called with a width, a height, and optionally a threshold value for the grid and a boolean indicating
whether the grid should be initialized with random values.

The bulk of the code is composed of utility functions for graphing and otherwise recording the key statistics of the model.

```
Pile = SandPile(50, 50, random=True)
Pile.simulate(10000)
Pile.graph('output_directory')
```

The simulate function takes an integer representing the number of time steps to simulate and an optional tuple or list to specify a site to drop the sand on, instead of using a random site. The simulate function calls many other functions in the class, which can be seen examining the source code.

The graph function spits out files recording graphs and statistics of the quantities of interest. It takes two optional arguments: an output directory to save results in, and a boolean no_mass indicating whether mass loss statistics should be recorded. This boolean is helpful in situations where there is not enough data to accurately graph the mass loss as the system has not yet reached a critical state. If an error is occurring when attempting to produce a graph, setting this value to True may fix the

problem. If a nested output directory is given (e.g. ‘results/nested/output’), all but the last level of the directory must already exist for the output to be saved properly.

Another function of interest is the ensemble_simulate function. This function creates a large number of sandpiles and collects statistics on them independently of one another.
It has signature

```
ensemble_simulate(
      width,
      height,
      number_runs,
      n=1,
      site=None,
      output='ensemble/')
```

An example call would be

```
SandPile.ensemble_simulate(
      20,
      20,
      1000,
      output= “ensemble/”)
```

cylindrical.py
==========
This file contains the CylindricalSandPile class, which
inherits from SandPile. It overrides the needed methods to enforce cylindrical boundary conditions.

Its constructor takes the same arguments as the SandPile class and the same simulation methods can be used as in the SandPile case.

hourglass.py
==========
This file contains the HourGlassSandPile class, which
inherits from SandPile. It overrides the needed methods to enforce hourglass boundary conditions.

Its constructor takes the same arguments as the SandPile class and the same simulation methods can be used as in the SandPile case.

tests.py
==========
Tests to ensure that the classes are functioning properly and that they actually give the known results for simple cases.

This is a script which is designed to be run on the commandline. For example, after

moving to the proper directory, run in a terminal the command

```
python tests.py
```

The script will output a message describing the success or failure of the tests.

main.py

==========

A simple script used to generate results for the three types of sandpiles we considered, of various sizes. It uses multiprocessing to start a maximum of three processes to speed things up. te. Three was chosen because many personal computers have four physical cores, and it’s nice to be able to do something else while the results generate in the background. :)

On my personal computer, it takes about 15 minutes to comple

sandpilenumba.py

==========

Comparison of the original sandpile implementation with a refactored version using numba. Note this requires that the library numba be installed with a suitably recent version. I used the version 0.47.0.

This class was an experiment in trying to improve simulation performance. Much of the boilerplate from the original class is copied verbatim; the difference is in the simulation function. It does not include an ensemble_simulate function.

Since this class does not actually improve performance,

development was abandoned. It is included here as evidence of the attempt.

Example Usage:

```
from sandpilenumba import SandPile
pile = SandPile(20, 20)
pile.simulate(1000)
```

==========

There are many improvements which could be made to this software. However, the increase of simulation speed was given first priority in terms of development time. Thus, other values such as ease of use and code reuseability which were given lower priority. Some of the places where improvements in these areas could be made are noted in the comments of the relevant source files. Ultimately, these improvements were not made due to the need to get a working product out the door and the awareness that investing a great deal of time in improving code coherence and refactoring methods to be more discrete was not particularly good use of time in a project as simple as this.

For example, in a more complex project, it

would be desirable to break out the housekeeping and utility methods in the SandPile class which were not directly related to the SandPile’s function into a separate class to better promote encapsulation and cohesion. Instead, I chose to focus on attempting to improve the speed and accuracy of the results. This was a worthwhile tradeoff in my belief.