Computer Science CS134C (Fall 2018)
Duane A. Bailey
Laboratory 8
Setting Precedent in the Supreme Court (Due Thursday/Friday)

Objective. To experiment with sorting in Python.

This week, we'll look at one of the most carefully curated social networks—the network of Supreme Court majority decisions. In this network, the majority decision of a case is founded in the law by *citing* past decisions. Past decisions establish *precedent* by being cited by later cases.

A similar network is the network of academic publications. Publications cite prior work to establish legitimacy, and older publications become influential when they're cited by future work.

In academic circles, an author's influence can be estimated by *academic impact scores*. In this lab we'll attempt to apply academic impact scores to dockets of Supreme Court decisions.[1]

Background: The h-Index

A commonly used metric for determining the impact of authors is the *h-index*. The index is computed by counting the number of times each of the author's papers have been cited. Suppose Rita DeCoder has has written ten papers. Her list of citation counts might look something like this:

```
[0, 2, 15, 9, 7, 48, 4, 82, 14, 6]
```

Rita's first paper has been cited zero times, her second paper has been cited twice, etc. Rita's h-index is *the maximum* n *where her top* n *papers have been cited at least* n *times*. Looking at Rita's citation counts we see that her top 6 papers have been cited at least 6 times.

The process is more obvious when you sort the citation count list in descending order:

```
[82, 48, 15, 14, 9, 7, 6, 3, 2, 0]
```

It's easy to see that the first six elements are greater than or equal to 6, but the 7th is less than 7. We'll leave it to you to determine the relationship between the *values* of the list and their list *indices*. Authors with a high h-index have written many highly-cited papers and, over time, their impact score may improve as more and more people cite their works.

It may now be clear that we have to use Python's sorting methods. Python's main workhorse for sorting is a builtin procedure, `sorted(i)`, that takes an iterable (e.g. a `tuple`, `list`, `set`, or `range`, etc.), `i`, and returns a fresh list of values in their natural ascending order. For example, the `set` of `tuples`

```
>>> i = {(1,'one'), (7,'seven'), (5,'five'), (7,'magic'), (3, 'magic')}
>>> sorted(i)
```

would naturally be ordered by `tuple` comparison:[2]

```
[(1,'one'), (3,'magic'), (5,'five'), (7,'magic'), (7,'seven')]
```

Of course, you might chose to sort this same set by just the second component of each `tuple`. You can accomplish this by providing a function, `second`, that, given a `tuple` called `pair`, returns the actual *key* to be used for comparison:

---

[1] Thanks to Peter Tianlun Zhang for performing this initial research and to Carl Rustad for developing this lab.

[2] Recall that tuple a is less than b if in the first component that differs, a's component is less than b's.

```
>>> def second(pair):
...     return pair[1]
...
>>> sorted(i,key=second)
[(5,'five'), (7,'magic'), (3, 'magic'), (1,'one'), (7,'seven')]
```

Notice the relative positions of the two `'magic'` pairs. These two values have no reason to be swapped. We say that a sort is *stable* if values exchange places only if it's `required` by comparison. By the way, if we have a simple function like `second`, it's often easier to specify it as a *lambda*:

```
>>> sorted(t,key=lambda pair : pair[1])
[(5,'five'), (7,'magic'), (3, 'magic'), (1,'one'), (7,'seven')]
```

Only mutable order-preserving classes, like `lists`, can be sorted *in place* using the dedicated method, `sort`. Immutable objects, `str`, cannot be modified, so it is meaningless to support in-place sorting.

First steps.
Before you begin, you will need to clone your `lab8` repository. Assuming your username is `22xyz9`, type:

```
git clone ssh://22xyz9@davey.cs.williams.edu/~cs134/22xyz9/lab8.git ~/cs134/lab8
```

We hope to plot data, so you will also have to activate your virtual environment:

```
-> cd ~/cs134
-> source bin/activate
(cs134) -> cd lab8
```

You will find several CSV files in the lab folder. The primary data you will be using comes from Fowler and Jeon's interesting analysis of the 30,288 majority US Supreme Court decisions on dockets through 2002.[3] The raw data used for their work is found in `judicial.csv`. Each row in this file corresponds to a majority decision of the U.S. Supreme Court. There are many different columns in this file, but we are primarliy interested in are `caseid` (column 0), `year` (column 3), and `indeg` (column 8). The `caseid` is a serial number the authors have given to each case in the order they were decided. The `indeg` field is the number of later Supreme Court decisions that cite that particular case; we will refer to this statistic as the *citation count* for a decision. The `year` field is the date of the decision. This information is sufficient to compute an impact score for each year—each *docket*—of decisions made by the Supreme Court.

For those seeking to investigate the citation network more deeply, the file `allcites.csv` is a file with two columns of numbers: the left column is a case identifier for a decision making a citation, and the right column is the case cited.

In `chiefJustices.csv` we've listed all the chief justices along with the start and end year of each justice's term. This file may also be helpful in deeper investigations.

[3] J. H. Fowler, S. Jeon, "The authority of Supreme Court precedent", *Social Networks*, 30(1), January 2008, pp. 16-30. See also `fowler.ucsd.edu/judicial.htm`.

Required tasks.

In the python file `lab8.py`, you will find a number of incomplete methods that we expect you to finish. We suggest the following approach:

1. `readDecisions(filename)`. This is a function that reads the CSV data found in the indicated file (typically, `"judicial.csv"`) and returns a *decisions dictionary* summarizing each year's cases. This decisions dictionary maps a year to a **tuple** of citation counts for each majority decision, in decision order, made during that year. Using a tuple makes it difficult to accidentally modify this important data. There are 235 different years that appear in the Fowler database.

2. `hIndex(i)`. This function computes the integer h-index for a (finite) iterable collection of integer citation counts. Here are some important examples:

```
>>> hIndex( [] )
0
>>> hIndex( (0, 0) )
0
>>> hIndex( {1, 2, 3} )
2
>>> hIndex([0, 2, 15, 9, 7, 48, 4, 82, 14, 6])
6
```

3. `docketImpacts(decisions)`. This function constructs an ordered *list* of pairs of the form `(year, h-index)` that contains the h-index for every year in the decisions dictionary. Make sure that the list is sorted in increasing order by year. The h-indices associated with the years 1800, 1900, and 2000 are 3, 22, and 2, respectively.

4. `plotImpacts(impacts,plotfilename)`. This procedure plots the h-index for each year in chronological order (please use line-style plotting to suggest trends). Remember to activate your Python environment before using `matplotlib`. Your independent variable will be the year and the dependent variable will be the h-index. Please label your axes and add a descriptive title.

The work above is sufficient to get a good grade. When you complete your work, please add, commit, and push it to the server.

Pushing forward.

If you'd like to push this analysis a little bit further, you might consider one or more of the following possibilities. To ensure we understand what you have done, make sure that you fully document your efforts and add/commit/push any new files you create to the repository.

- ⋆ Implement an immutable `SCOTUS` class, described in a file `scotus.py`. It imports and uses the functions and procedures of `lab8` to support helper and accessor methods in the `SCOTUS` class. A typical use of this class might be:

```
>>> db = SCOTUS("judicial.csv")
>>> db._hIndex([4,4,4])
3
>>> db.impact(1900)
22
>>> db.plotImpacts("judicial.pdf")
```

- ⋆ Determining impacts of specific courts. As mentioned above, the CSV file `chiefJustices` lists the terms of all the Chief Justices of the United States. Because these powerful justices determine the cases to be heard, they often determine the "personality" of the court. Thus, for example, the *Rehnquist Court* was a conservative course that spanned the years 1986 through 2005. For the purposes of this assignment, let's consider a majority decision to be part of the a particular justice's court if it was decided in a year of their term. Write a Python procedure, `courtInfluence(decisions)` that reads the terms of each chief justice and computes the impact of each of their court's decisions found in the decisions dictionary, `decisions`. The procedure sorts and prints the names of the courts and their h-index in decreasing order of impact.

- ⋆ Evaluation the foundation of decisions in law. We used the data column `indeg` to determine how many cases a particular Supreme Court case was cited by. If you'd like to extend the lab, include a plot of "foundational" h-indices computed using the data from `outdeg`. This measures how many cases are referenced by the case in question. The h-indices made with this data give us an idea of how frequently the court made use of previous rulings each year. Courts that practice *judicial activism*, of course, make fewer references to prior law in their decision making and, as a result, might lead to lower foundational h-index scores. Notice that scores based on out-degree don't change over time.

We would be very impressed with *any* of these extensions to the basic lab. Any of this work would be an important extension to Fowler and Jeon's basic investigation!

⋆