The purpose of this assignment is to get you to think about a problem in terms of **tests**.

Introduction

This assignment is about lines of succession to a throne. A "line of succession" is the order of people who are next on the throne after the current person on the throne dies. The method by which the order of the line of succession is determined is called the **rules of succession**. In the real world, the rules are really complicated, based on who gave birth to who, and when. [Here is the current line of succession to the British throne (Wikipedia)](http://en.wikipedia.org/wiki/Line_of_succession_to_the_British_throne" \l "The_early_part_of_the_line_of_succession" \t "_blank)

When we say a "line of succession" in this assignment, we mean an ordered list of objects that represent people, where the first object/person in the list is the next on the throne.

In this assignment, you will be modifying the \_\_cmp\_\_() function of several classes that inherit from the supplied Knight object, based on different rules of succession. Given two knights, your \_\_cmp\_\_() function will have to determine which should succeed to the throne first. Don't worry - the rules we will give you to implement will be much more simple than the rules used in any real-world monarchy.

Files To Download

Here's a listing of all the files to download:

* [knight.py](http://www.cdf.utoronto.ca/~csc148h/summer/assignments/a3/knight.py)
* [redknight.py](http://www.cdf.utoronto.ca/~csc148h/summer/assignments/a3/redknight.py)
* [greenknight.py](http://www.cdf.utoronto.ca/~csc148h/summer/assignments/a3/greenknight.py)
* [yellowknight.py](http://www.cdf.utoronto.ca/~csc148h/summer/assignments/a3/yellowknight.py)
* [purpleknight.py](http://www.cdf.utoronto.ca/~csc148h/summer/assignments/a3/purpleknight.py)
* [pinkknight.py](http://www.cdf.utoronto.ca/~csc148h/summer/assignments/a3/pinkknight.py)
* [successions.py](http://www.cdf.utoronto.ca/~csc148h/summer/assignments/a3/successions.py)
* [test\_successions.py](http://www.cdf.utoronto.ca/~csc148h/summer/assignments/a3/test_successions.py)

Types of Knights

In the universe of our assignment, there are several parallel worlds, where knights have the same name, date of birth and gender. However, the rules of succession differ in each world. In the "plain" world, the oldest knights (those than were born earliest) get to go on the throne first. The rules get stranger in the other worlds.  
The Knight class, representing the plain knights, is already done for you and you do not need to modify it. Every other knight class inherits from it. The RedKnight class is also already done for you. See how it uses the \_\_cmp\_\_(self,other) method. I n the class for each knight, you only need to implement the \_\_cmp\_\_ method, which overloads the \_\_cmp\_\_ method in the parent class (Knight). You do not need to implement anything else, although you may find 1 or 2 helper functions useful. Do NOT overload the \_\_init\_\_ method, as there is no reason to change its functionality.   
You can read the standard Python documentation on the \_\_cmp\_\_() method [here](http://docs.python.org/reference/datamodel.html#object.__cmp__).

Here are the rules for each type of knight in their separate universes:

Plain Knight

Older knights go on the throne first. The order between knights of the same age does not matter.

Red Knight

Red knights are the same as plain knights. However, if there is a man and a woman of the same age, the woman goes first.   
For example, if we had the following red knights:

|  |  |  |
| --- | --- | --- |
| Name | Date of Birth | Gender |
| A | 1990 | M |
| B | 1995 | M |
| D | 1998 | F |
| C | 1995 | F |

The final ordering of the line of succession would be: A,C,B,D.

Pink Knight

These knights are odd, but practical. They prefer knights who have the shorter name over knights that have a longer name in the line of succession. When we say "name", we mean the full name, so "Bartholomew the Short" (len=21) is a shorter name than "Ted the Dubiously Dressed" (len=25). Pink Knights to not distinguish between knights whose names are the same length (meaning, the \_\_cmp\_\_() method should return 0).

Green Knight

No women are allowed on the throne until all men have been on the throne, except for women whose name starts with 'S'. Otherwise, older knights are preferred.  
For example, if we had the following green knights:

|  |  |  |
| --- | --- | --- |
| Name | Date of Birth | Gender |
| Ted | 800 | M |
| Sam | 850 | F |
| Paul | 866 | M |
| Cheryl | 832 | F |
| Lucy | 798 | F |
| Bob | 888 | M |

The final ordering of the line of succession would be: Ted, Sam, Paul, Bob, Lucy, Cheryl.

Yellow Knight

Superstitiously, these knights prefer people born in even-numbered years first. Only after people born in even-numbered years are gone, do they want people born in odd-numbered years. Otherwise, the oldest knights come first. For example, we had the yellow knights:

|  |  |  |
| --- | --- | --- |
| Name | Date of Birth | Gender |
| Ted | 800 | M |
| Sam | 801 | F |
| Paul | 802 | M |
| Cheryl | 803 | F |
| Lucy | 804 | F |
| Bob | 805 | M |

The final ordering of the line of succession would be: Ted, Paul, Lucy, Sam, Cheryl, Bob.

Purple Knight

The purple knights do not like names that start with the letter R. Everyone is ranked by date of birth, with older knights preferred first. However, knights whose name starts with the letter R are treated as if they are 10 years younger.  
For example, if we had the following purple knights:

|  |  |  |
| --- | --- | --- |
| Name | Date of Birth | Gender |
| Ted | 800 | M |
| Roderick | 801 | F |
| Paul | 802 | M |
| Cheryl | 803 | F |
| Lucy | 814 | F |
| Rodriguez | 803 | M |

The final ordering of the line of succession would be: Ted, Paul, Cheryl, Roderick, Rodriguez, Lucy.

Testing

You will be use the nose module as shown in lecture to test your code. We are marking your \_\_cmp\_\_ method and the way you test with equal weight! The test file is[test\_successions.py](http://www.cdf.utoronto.ca/~csc148h/summer/assignments/a3/test_successions.py). For each type of knight, you will implement a test\_x\_simple and a test\_x\_knights. The simple test method should test on pairs of knights that you instantiate in that test method. The knights test method should test on the entire line of succession for that type of knight as constructed in [successions.py](http://www.cdf.utoronto.ca/~csc148h/summer/assignments/a3/successions.py). We have already implemented tests for the plain, red and pink knights; look at what we've done to get a sense of what you should do. When you run the test code the first time, the pink knight test will obviously fail, since you haven't implemented PinkKnight's \_\_cmp\_\_ function yet.  
[test\_successions.py](http://www.cdf.utoronto.ca/~csc148h/summer/assignments/a3/test_successions.py) runs [successions.py](http://www.cdf.utoronto.ca/~csc148h/summer/assignments/a3/successions.py). If you open [successions.py](http://www.cdf.utoronto.ca/~csc148h/summer/assignments/a3/successions.py), you will see that it loads several knights of different names, dates of birth and genders into lines of succession, and calls python's built-in sort method on them. If you have implemented \_\_cmp\_\_ correctly in each knight class, the ordering should be correct according to each type of knight's rules. Note that each type of "line of succession" will have knights of with the exact same name, date of birth and gender. It's best to think of these as "alternate universes".   
Keep in mind the following:

* You tests should NOT compare the entire list of knights, (e.g. assert green\_line[i] == expected\_green\_line[i]). We may use a different listing of knights than found in the [successions.py](http://www.cdf.utoronto.ca/~csc148h/summer/assignments/a3/successions.py). Look at how the tests for plain, red and pink only compare between two adjacent knights. Use that format.
* Your assert statements messages should be meaningful and helpful. See how the message in test\_plain\_knights() is   
  'Dates of birth not ordered correctly in plain line: '\  
  + str(knight1) + " " + str(knight2)
* You should implement the following test functions:
  + test\_green\_simple()
  + test\_green\_knights()
  + test\_yellow\_simple()
  + test\_yellow\_knights()
  + test\_purple\_simple()
  + test\_purple\_knights()

[successions.py](http://www.cdf.utoronto.ca/~csc148h/summer/assignments/a3/successions.py) has a helpful function print\_line. Pass a list of knight objects to this to have them printed in order. This is useful when debugging errors.   
Finally, we recommend you write the tests first, rather than the sorts. This makes it easier to check if your sorts are working correctly, and you have to do it anyway.

Additional requirements

1. Your methods must have absolutely no user-input (i.e. nothing that the user types). Also, your methods must have absolutely no output -- no print statements; no messages to the user (not even a "welcome!"); and no displaying of pictures, or else **you will receive zero for your assignment**.
2. You should not do any error checking for this assignment. Assume that all input values to the methods you design are within the bounds outlined by the assignment description.