**This assignment falls under the standard cmsc201 academic integrity policy. To reinforce the obvious, this project should NOT be posted online to any forum or solution website, anyone who posts this project or uses a solution from an online site may receive an F for the course.**

Make sure that you have a complete file header comment at the top of each file, and that all of the information is correctly filled out.

**"""**

**File: FILENAME.py**

**Author: YOUR NAME**

**Date: THE DATE**

**Section: YOUR DISCUSSION SECTION NUMBER**

**E-mail: YOUR\_EMAIL@umbc.edu**

**Description:**

**DESCRIPTION OF WHAT THE PROGRAM DOES**

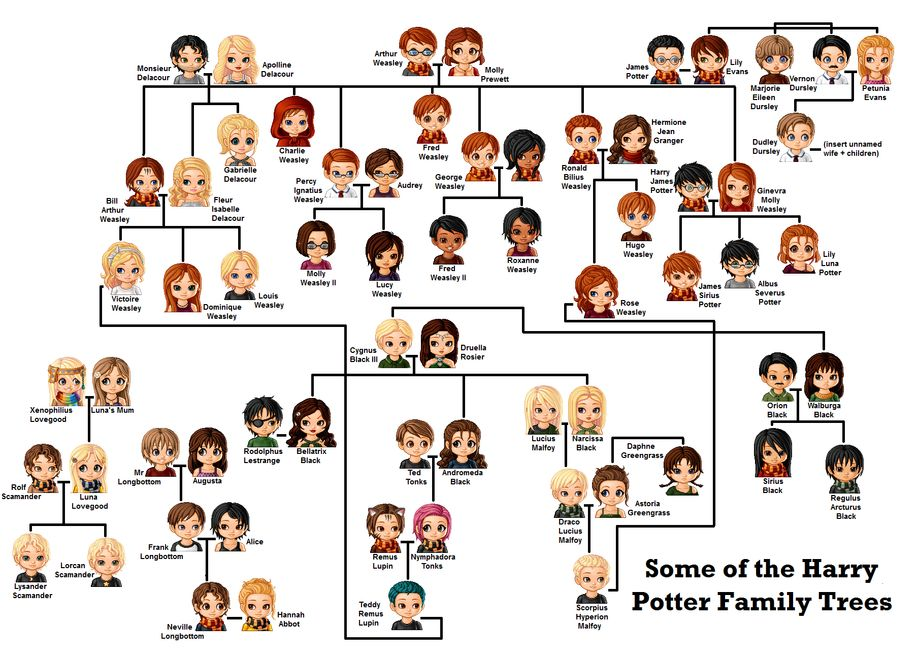
**"""**

# Project Description

In Project 3, you will build a family tree program. This program will allow you to enter people, and then query relations between various people in the family tree. Another goal is to learn a bit about json and how we can use it to load and save dictionary structures in files.

## What is a family tree?

A family tree is a map of relationships between people in a family. They're generally drawn out like this:



# Family Tree Rules

1. No Time Travel Paradoxes - No one will be their own parent, child, grandparent, grandchild, etc. A person will exist in a single "generation" only.
2. A person can have at most two parents. They can have one or zero parents, if their parents aren't known for instance. Their parents **cannot** be the same person.
3. A person can have an arbitrary number of children, probably less than 1,000.

# 

# Required Classes

You must implement the following two classes:

class Person:

This class must contain the information of parents, children, and the name of the person described by that class object.

This class must be implemented in a file called "person.py".

You must make meaningful use of this class in your overall program. Not doing so means a loss of 5 points.

You should use the line in order to import the class Person into your main project code:

from person import Person

class DynasticDescent:

This class must implement the following features:

1. You must implement the ability to get parents, grandparents, and other ancestors of degree n. (This should probably be done recursively). An ancestor of degree n is:
   1. Degree 1 are parents
   2. Degree 2 are grandparents
   3. Degree 3 are great-grandparents
   4. Degree 4 are great-great-grandparents.
   5. etc.
2. Similarly, you must be able to find children, grand-children, great-grandchildren. (This should probably be done recursively). You should find descendents based on "degree" so:
   1. Degree 1 are children
   2. Degree 2 are grandchildren
   3. Degree 3 are great-grandchildren
   4. Degree 4 are great-great-grandchildren
   5. etc.
3. Get Siblings of a given person. Siblings are people who have one of the two same parents, or both. Half-siblings are siblings and should be returned.
4. Add a new person's name. They should have no relations before we relate them with other people as parents or children.
5. Relate two people. You should be able to add a person as a child of another person (and set the second person's parent as their parent). If there are already two parents, then you should reject the new assignment.
6. Save a family tree to a file. You should decide on the file format.
7. Load a family tree from a file. You must use the same format to load and save.

All of this should be implemented inside of the DynasticDescent class which is implemented inside of a file called proj3.py.

# Testing Scripts

The testing script is worth 10 points.

Let's require tests of at least 4 functions, and at least 20 tests total for all of the functions tested.

Not every method or function needs to be tested.

You should write and test your load and save functionality first, then use it in the tests of other functions.

For instance, if you have a simple file that just has a few relations between parents and children you can test out the relations:

def test\_get\_parents():

dd = DynasticDescent()

dd.load\_file('test\_parents.json')

if dd.get\_parents('test person') == ['parent one', 'parent two']:

print('parent test 1 passed')

else:

print('parent test 1 failed')

Then you should add a few more tests to test other features, what if there is one parent (the other one is not added to the tree), or zero (if neither are added?). Think of the various types of conditions we can test to determine whether a get\_parents function works within the DynasticDescent class.

It may be hard to test your load or save methods, so you should test them very simply, with files you know exactly the people added and the relations are all simple, and generally should only be very few.

Just like in project 2, it's quite alright if all of your tests fail when you submit your script. If you create some test files, you should submit them along with your test script. By the time that you submit your final project, hopefully all of your tests will then pass.

## What to Submit

Submit your test script on the test script due date, along with any of your currently created test files and your mostly unfinished project. Even if your functions for the project are simply "pass" that's alright, as long as the functions that are being tested exist. Try to get it to the point where at least the test scripts run and fail properly.

As you work on your project, you'll probably want to modify your testing script to match and finally your project and script should both run properly, i.e. your project should work and your tests should pass on your project.

When you submit your final project, make sure to submit the updated testing script with tests that pass.

We will grade your first test submission based on 'effort,' i.e. how much did you think through your project? We can tell this because when you're writing tests, you can think about all the possibilities that you may have to test for to ensure that your project works, even if you don't really know how to solve all the problems yet.

Your final test script submission in the main project directory on the other hand should ideally run and pass on your project.

# Coding Standards

Prior to this assignment, **you should be familiar with the entirety of the Coding Standards**,

For the coding standards:

<https://docs.google.com/document/d/1hxv3Lp1TT4xe1HTm3nGBN7EXfq0pMjP40-yfrN4rKRI/edit?usp=sharing>

For the forbidden magic:

<https://docs.google.com/document/d/1IoMhl8Y1y9ZYFEQly7s2aKgAy9DOt_bG7XlhyJKihlo/edit?usp=sharing>

**You should be commenting your code, and using constants in your code (not magic numbers or strings).**

**Any strings with a meaning should be constants!**

**Any numbers other than 0 or 1 are magic numbers!**

You will **lose major points** if you do not follow the 201 coding standards.

If you have questions about commenting, whitespace, or any other coding standards, please come to office hours.

**TIP**: This would be a very good time to use incremental development! Incremental development is when you are only working on a small piece of the code at a time, and testing that the piece of code works before moving on to the next piece. This makes it a lot easier to fix any mistakes.

# More Details

# In order to get descendants and ancestors, you will almost certainly have to use a recursive solution.

You are encouraged to use the json package to convert dictionaries (lists, other primitive types) into a format which can be saved and loaded by files.

## JSON Files:

A JSON (JavaScript Object Notation) string is a string which is used to transmit data over the internet as well as to be saved in files for easy retrieval. There are other ways to send data over streams but this one is definitely very common and relatively easy. You can convert your data structures (dictionaries, lists, strings, ints, bools, floats) into JSON objects and then save, load, send them.

Two methods you can use are:

json.loads(a\_string)

This method will take a\_string (probably read out of a file) and convert it into a dictionary for your use in your program.

json.dumps(a\_dictionary)

This method will take a dictionary and return to you a string. This string, if written to a file, can be loaded again.

Hint: You may have to create a method that converts the data in your Person class to a JSON string.

If you want to know the documentation:

<https://docs.python.org/3/library/json.html>

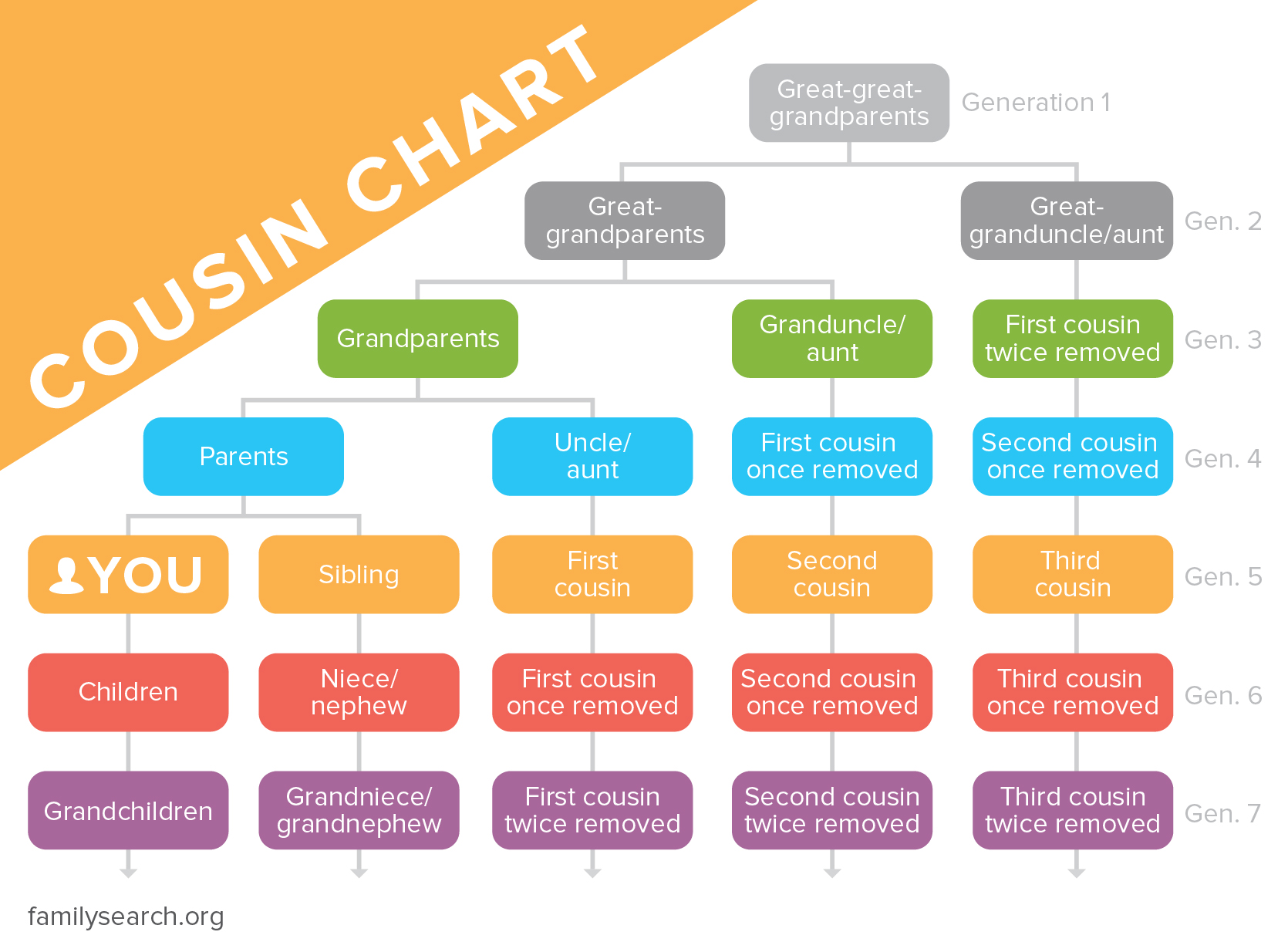
## Extra Credit (Find Cousins, 5 points)

Finding cousins in a family tree is an interesting challenge. In order to find cousins, in the same generation:

1. First cousins share the same grandparents but not the same parents.
2. Second cousins share the same great-grandparents but not the same grandparents.
3. Third cousins share the same great-great-grandparents but not the same great-grandparents.
4. And so on in that fashion.
5. You don't have to worry about the generations removed of cousins, just talk about 1st, 2nd, 3rd cousins.

In order to find the nth cousins you'll have to find the n+1st ancestors and then find the n + 1st descendants of the starting person.

There are multiple issues that you can run into, for instance, tracing back from a single grandparent can lead to repetitions in cousins, so you'll have to make sure each person is only represented in your solution once.



# Sample Output

[The Sample Output for Project 3 is here.](https://docs.google.com/document/d/1dUy5jdnEeUfQvY3taMJ0GCfvJmbygLHDrs5HJvy75AI/edit?usp=sharing)

# Points

The project is worth a total of 80 points. Of those points, 10 will be based on your testing script. The following is a general idea of how the distribution of points for this project will be given.

|  |  |
| --- | --- |
| Creating the Person class | 10 points |
| Add person functionality | 5 points |
| Relation functionality | 10 points |
| Implementing the Descendants Functionality | 8 points |
| Implementing the Ancestors Functionality | 8 points |
| Sibling Functionality | 4 points |
| Save Functionality | 5 points |
| Load Functionality | 5 points |
| Main Loop | 10 points |
| Coding Standards, etc | 5 points |

# Submitting

Once your **proj3.py** or **proj3\_test.py** file is complete, it is time to turn it in with the submit command. (You may also turn the design or project in multiple times, as you reach new milestones or complete each piece. To do so, run submit as normal.)

## BIG NOTE: We got around to changing the name of the project 3 design folder to tests, so it's a different submit this time.

To submit your design file (which is due Monday, May 4th, 2020 by 11:59:59 PM), use the command:

|  |
| --- |
| **linux1[4]% submit cmsc201 PROJ3\_TESTS proj3\_test.py**  **Submitting proj3\_test.py ...OK**  **linux1[5]% █** |

Submit any other files you want to use for testing as well.

To submit your project file (which is due Monday, May 11th, 2020 by 11:59:59 PM), use the command:

|  |
| --- |
| **linux1[17]% submit cmsc201 PROJ3 proj3.py person.py**  **Submitting proj3.py...OK**  **Submitting person.py...OK**  **linux1[18]% █** |

# If you don’t get a confirmation like the one above, check that you have not made any typos or errors in the command.

You can check that your project and/or design was submitted by following the directions in Homework 0. Double-check that you submitted your files correctly, **since an empty file will result in a grade of zero for this assignment.**