CS 362 / 562 - Assignment 1 Spring, 2026

Assignment 1 - Kids Running in the Yard

The goal of this assignment is to to start thinking about problems in artificial intelligence and to practice your abilities with Python. There are three parts of this assignment: Reflection, Implementation and Comparison. You must respond to all three parts of the assignment. Assignment administration discusses other important parts of this assignment and expectations for completing it successfully.

Reflection

For this part of the assignment, you will read Mariette Awad's 2024 / 2025 paper entitled "Types of AI and their use in science." (This paper is officially available at the IDRC Digital Library. You may also find it via https://council.science/wp-content/uploads/2025/10/Types-of-AI-and-their-use-in-science-2.pdf). Answer the following questions:

- Awad distinguishes between different "types" of AI. What classification scheme does the paper use and why do these types matter for scientific research?
- Does Awad make a clear distinction between AI as a tool and AI as a scientific collaborator? If so, what are the differences and what are some examples given to support the differences? Do these examples suggest a real shift in how science is conducted, or mostly an extension of existing methods?
- What are some limitations or risks of using AI in science? How do these relate to issues such as interpretability, bias, reproducibility, or theory formation?
- According to Awad's arguments, is AI more likely to *accelerate* scientific discovery or to *reshape* the scientific method itself? Do you agree or disagree?

Implementation

For this part of the assignment you will build a family tree. This family tree will begin with two people born in 1950 ("the first two people"), their children, grandchildren, etc. until no more children can be generated or until 2120, whichever comes first. Once generated, you will use this family tree for reporting. As a running example, we may assume the first two people born in 1950 are named "Desmond Jones" and "Molly Jones" (née "Molly Smith").

Each person in the family tree must have the attributes listed in Table 1. Your implementation may have other attributes as you deem useful or necessary. The data in the files available for this project is based in the USA and England, but may not be valid, may not describe the particular people or the culture of where it's from and definitely does not accurately describe cultural shifts (eg. conventions for last names) we can observe in these places.

Your implementation must assume all files are in the current directory. Your implementation may read the data files with:

- Basic python read loops — eg. as described in Stack Overflow posts
- Using a tool — eg. Pandas' read_csv

However, you are required to cite the sources of code you use from places outside class. You are also responsible for understanding how to use the code used in your submission. See the subsection entitled "Use of LLMs and the code review."

Attribute	**Associated Files(s)**	**Determination**
Year Born	N/A	The first two people must be born in 1950. The Year Born values for other people are described in the Children row.
Year Died	life_expectancy.csv	Based on the decade in which a person is born, that person has a life expectancy listed in this file. Randomly generate the length of the person's life based on the contents of this file, +/- 10 years. For example, a person born in 2067 will have a Year Died between 2138 - 2158 — i.e. a life expectancy of 81.51 years (2148), plus or minus 10 years.
First Name	first_names.csv *gender_name_probability.csv*	A new person's first name is based on Year Born, gender and the frequency of names found during that year. For example, a person assigned male at birth in the 1950s has more than a 13.9% probability of being named "James." For graduate students in CS 562, you must additionally implement the following functionality: a person has a probability of a gendered first name according to the file "gender_name_probability.csv". Male names are indicated in the appropriate column of the first_names.csv file.
Last Name	last_names.csv rank_to_probability.csv	The last name for a person is based on the following rules: 1. Any person directly descended from the first two people keep the last name of either of the people. 2. For any person not directly descended from the first two people, the last name may be found in the last_names.csv file. In this file, find the rank of the last name (1 - 30). Your implementation will use this rank to find the probability of each last name based on the rank from the rank_to_probability.csv file. You may notice that the sum of the probabilities is not 1.0.
Partner / Spouse	birth_and_marriage_rates.csv	A person has a probability of having a spouse or partner as defined in the "marriage_rate" column of this file. Your implementation must create a partner — a "new" person — for such a marriage or partnership and that the partner's Year Born is randomly within 10 years of the partner's. You may assume that any marriage or partnership does not involve a change of name, and that a person can have at most 1 partner.
Children	birth_and_marriage_rates.csv	A single person or a person with a partner / spouse may have a number of children as defined in the "birth_rate" column of this file. The number of potential children depends on the

		decade of the person's birth, +/- 1.5 children. For example, a person born in the 2010s (between 2010 and 2019) will have between 1 (2.05 - 1.5 = 0.55; round up to 1) and 4 (2.05 + 1.5 = 3.55; round up to 4) children. The Year Born field for the children will be distributed evenly from the elder parent's Year Born + 25 years through parent's Year Born + 45 years. For example, a parent born in 1983 will have the child / children born between 2008 and 2028. For graduate students in CS 562, you must additionally implement the following functionality: a person who does not have a partner or spouse will have 1 child fewer than a person who has a partner or spouse.

Table 1: Attributes, Determinations and Files for people in a family tree

Once the family tree has been generated, you will allow the user to query or interact with the tree, specifically allowing:

1. Total number of people in the tree
2. Total number of people in the tree by year
3. Duplicate names

You may add other functionality.

An example of the interaction follows, with the user's input indicated in bold:

```
Reading files...
Generating family tree...
Are you interested in:
(T)otal number of people in the tree
Total number of people in the tree by (D)ecade
(N)ames duplicated
> N
There are 2 duplicate names in the tree:
* Jennifer Smith
* John Li
Are you interested in:
(T)otal number of people in the tree
Total number of people in the tree by (D)ecade
(N)ames duplicated
> D
1950: 2
1960: 2
1970: 2
1980: 5
1990: 5
2000: 11
[Remainder not shown]
(T)otal number of people in the tree
Total number of people in the tree by (D)ecade
(N)ames duplicated
> T
The tree contains 857 people total
```

A Design

There are many possible object-oriented designs for this implementation. This subsection represents one such design. While the implementation must be completed using object-oriented python and must have the functionality described, it does not need to be completed using the decomposition detailed below.

Class	Description	Attributes & Methods
Person	The Person class is responsible for keeping details of each simulated person in the model. Each member of the family tree is an instance of a Person class.	Suitable attributes for this class are listed in Table 1. Accessor and Mutator methods are essential for this class.
PersonFactory	The PersonFactory class is responsible for reading the data files and for generating new instances of the Person class.	This class has few attributes. Suitable methods include: get_person(year_born) and read_files()
FamilyTree	The FamilyTree class is the “driver” and is responsible for keeping reference to all Person instances.	This class should have two Person attributes. Methods for generating the family tree and for responding to user queries are useful for this class.

A note on code style

Your implementation must use one consistent and widely-accepted style, specifically for consistent indentation, line length, whitespace around operators, snake case names for functions, etc. The style you choose must also be self-documenting names for variables and functions. Python programmers (at least some) have converged on a style standard called PEP 8, which defines conventions around the above as well as exception raising, line continuations and other style elements. Examples of some of these are shown in the “fibonacci” function below.

```
def fibonacci(n):
    """Return the nth Fibonacci number."""
    if n < 0:
        raise ValueError("n cannot be negative")

    if n in (0, 1):
        return n

    return fibonacci(n - 1) + fibonacci(n - 2)
```

If you choose to use a style OTHER than PEP 8, you must reference the style you use in your submission comments.

Comparison

Once your implementation is complete, use (a modified version of) the above requirements as input to one of the following LLMs: ChatGPT, Google Gemini, Anthropic Claude, Grok, or Cursor. Notice that running these LLMs may require that you sign up for an account or that you pay the companies some fee. Change the prompts until you are satisfied with the outcome. It may be useful for you to review materials on prompt engineering: Prompt engineering essentials: Getting better results from LLMs | Tutorial or similar, for example.

Answer the following questions:

- Which tool(s) did you use?
- If you used an LLM, what was your prompt to the LLM?
- What differences are there between your implementation and the LLM?
- What changes would you make to your implementation in general based on suggestions from the LLM?
- What changes would you refuse to make?

You may change your implementation based on the differences between your original implementation and the one suggested by the LLM; however, you must show your original version.

Assignment administration

This section discusses the non-functional rules for the project, i.e. submission, grading, resubmissions and the use of LLMs and the code review. Accept this GitHub Classroom Assignment.

Submission

Your repository must contain the following items:

- Your answers to the Reflection questions. This may be in the form of a link to a Google Document (readable by anyone with the link) -OR- may be a document in TXT or PDF format. Other formats will not be accepted.
- Your completed implementation in OO python, checked in directly to the repository. Do NOT upload any compiled files such as those ending in "`.pyc`".
- Answers to the five questions in the Comparison section. You may answer these questions in the README.md file in the repository.

Grading

As with Project 1, grading will take place in two phases:

1. Once you submit your implementation, an evaluator will determine whether your implementation passes correctness and other quality standards, as detailed in the bullets below. If your implementation meets or exceeds these standards, the evaluator will mark your submission as "Reviewable." You must sign up for a code review over zoom within 2 weeks of your implementation being marked "Reviewable."

Assignment grades are based on Elements of quality standards are as follows:

- Reflection: your answers to the reflection questions must indicate that you have understood the arguments in the paper and can engage with the arguments in a sophisticated manner.
- Implementation: your implementations must run successfully with valid input. The implementation must fail gracefully for invalid inputs.
- Efficiency: in the eyes of the evaluator, your implementation must be maximally efficient with respect to expected running time and required space.
- Decomposition: in the eyes of the evaluator, your implementation must demonstrate a reasonable object oriented decomposition — i.e. encapsulation.
- Style: in the eyes of the evaluator, your implementation should be well-commented and must consistently use intelligently-named variables and functions.
- Documentation: in the eyes of the evaluator, the github repository must be well-documented and must not contain extraneous data, files or information.
- Code review: graduate students with passing implementation and efficiency will be invited to a code review session to review the submission. Some undergraduate students may also be required to perform a code review. In some cases, your evaluator will be unable to provide a grade until after the code review has been reviewed by the instructor or a second evaluator.

Your grade for this project will be determined as follows:

	Exceptional	**Good**	**Partial**
Reflection	*Required*	*Required*	*Submission contains one or more of these elements, but does not contain all of them*
Implementation	*Required*	*Required*	
Efficiency	*Required*	*Required*	
Decomposition	*Required*		
Style	*Required*		
Documentation	*Required*		
Code Review	*Required (CS 562)* *Possible (CS 362)*	*Required (CS 562)* *Possible (CS 362)*	*Not offered*

Missing submissions receive an Unassessable grade.

Late submissions and resubmissions

Submissions are allowed past the due date until the solution has been posted. Any late submission costs one (1) token.

Once submissions are graded, resubmissions are permitted until the solution has been posted. Each resubmission costs one (1) token.

Use of LLMs and the code review

For this assignment, you are required to use some code assist tool or large language model for the implementation portion. However:

1. You must clearly indicate what items are not exclusively your own work. In other words, you must cite any external sources of work (eg. StackOverflow posts by URL) and you must indicate code you use which was generated by an LLM or other tool. This indication may be in the form of a comment.
2. You are responsible for everything you submit. In other words, you must clearly understand the design, implementation and style choices of all the project artifacts (code, comments, etc.) you submit.

Practically, this means that the instructor or TA may require that you thoroughly explain the design, implementation or style choices during a code review. These code reviews are required for graduate students enrolled in CS 562 but may also be required for some undergraduate students.