In [1]:
```
# Homework 2: Markov Models of Natural Language

### Name: [Yang An]
### Collaborators: [None]
```

This homework focuses on topics related to string manipulation, dictionaries, and simulations.

I encourage collaborating with your peers, but the final text, code, and comments in this homework assignment should still be written by you. Please check the collaboration policy.

Submission instructions:

- Submit `HW2.py` and `HW2.ipynb` compressed in a one zip file on Gradescope under "HW2 - Autograder". Do **NOT** change the file name.
- Convert this notebook into a pdf file and submit it on GradeScope under "HW2 - PDF". Make sure your text outputs in the latter problems are visible.

Language Models

Many of you may have encountered the output of machine learning models which, when "seeded" with a small amount of text, produce a larger corpus of text which is expected to be similar or relevant to the seed text. For example, there's been a lot of buzz about the new GPT-3 model, related to its carbon footprint, bigoted tendencies, and, yes, impressive (and often humorous) ability to replicate human-like text in response to prompts.

We are not going to program a complicated deep learning model, but we will construct a much simpler language model that performs a similar task. Using tools like iteration and dictionaries, we will create a family of **Markov language models** for generating text. For the purposes of this assignment, an n-th order Markov model is a function that constructs a string of text one letter at a time, using only knowledge of the most recent n letters. You can think of it as a writer with a "memory" of n letters.

In [1]:
```
# This cell imports your functions defined in HW2.py

from HW2 import count_characters, count_ngrams, markov_text
```

Data

Our training text for this exercise comes from Jane Austen's novel *Emma*, which Professor Chodrow retrieved from the archives at (Project Gutenberg). Intuitively, we are going to write a program that "writes like Jane Austen," albeit in a very limited sense.

In [2]:
```
with open('emma-full.txt', 'r') as f:
    s = f.read()
```

Problem 1: Define `count_characters` in HW2.py

Write a function called `count_characters` that counts the number of times each character appears in a user-supplied string `s`. Your function should loop over each element of the string, and sequentually update a `dict` whose keys are characters and whose values are the number of occurrences seen so far. Your function should then return this dictionary.

You may know of other ways to achieve the same result. However, you are encouraged to use the loop approach, since this will generalize to the next exercise.

Note: While the construct `for character in s:` will work for this exercise, it will not generalize to the next one. Consider using `for i in range(len(s)):` instead.

Example usage:

```
count_characters("Torto ise!")
{'T': 1, 'o' : 2, 'r' : 1, 't' : 1, ' ': 1, 'i' : 1, 's' : 1, 'e' : 1, '!': 1}
```

Hint: Yes, you did a problem very similar to this one on HW1.

In [3]:
```
count_characters("Torto ise!")
```

Out[3]:
```
{'T': 1, 'o': 2, 'r': 1, 't': 1, ' ': 1, 'i': 1, 's': 1, 'e': 1, '!': 1}
```

How many times does 't' appear in Emma? How about '!'?

How many different types of characters are in this dictionary?

In [4]:
```
# Assuming 's' contains the content of "Emma"
emma_counts = count_characters(s)

t_count = emma_counts.get('t', 0)
exclamation_count = emma_counts.get('!', 0)
unique_characters = len(emma_counts)
```

```
print(f"The letter 't' appears {t_count} times in Emma.")
print(f"The exclamation mark '!' appears {exclamation_count} times in Emma."
print(f"There are {unique_characters} different types of characters in Emma.
```

```
The letter 't' appears 58067 times in Emma.
The exclamation mark '!' appears 1063 times in Emma.
There are 82 different types of characters in Emma.
```

Problem 2: Define `count_ngrams` in HW2.py

An `n`-*gram* is a sequence of `n` letters. For example, `bol` and `old` are the two 3-grams that occur in the string `bold`.

Write a function called `count_ngrams` that counts the number of times each `n`-gram occurs in a string, with `n` specified by the user and with default value `n = 1`. Your function should return the dictionary. You should be able to do this by making only a small modification to `count_characters`.

Example usage:

```
count_ngrams("tortoise", n = 2)

   {'to': 2, 'or': 1, 'rt': 1, 'oi': 1, 'is': 1, 'se': 1} #
   output
```

In [5]:
```
count_ngrams("tortoise", n=2)
```

Out[5]:
```
{'to': 2, 'or': 1, 'rt': 1, 'oi': 1, 'is': 1, 'se': 1}
```

How many different types of 2-grams are in this dictionary?

In [6]:
```
ngram_counts = count_ngrams("tortoise", n=2)
unique_ngrams = len(ngram_counts)
print(f"There are {unique_ngrams} different types of 2-grams in the dictiona
```

```
There are 6 different types of 2-grams in the dictionary.
```

Problem 3: Define `markov_text` in HW2.py

Now we are going to use our `n`-grams to generate some fake text according to a Markov model. Here's how the Markov model of order `n` works:

A. Compute (`n`+1)-gram occurrence frequencies

You have already done this in Problem 2!

B. Starting `n`-gram

The starting `n`-gram is the last `n` characters in the argument `seed`.

C. Generate Text

Now we generate text one character at a time. To do so:

1. Look at the most recent `n` characters in our generated text. Say that `n = 3` and the 3 most recent character are `the`.
2. We then look at our list of `n+1`-grams, and focus on grams whose first `n` characters match. Examples matching `the` include `them`, `the `, `thei`, and so on.
3. We pick a random one of these `n+1`-grams, weighted according to its number of occurrences.
4. The final character of this new `n+1` gram is our next letter.

For example, if there are 3 occurrences of `them`, 4 occurrences of `the `, and 1 occurrences of `thei` in the n-gram dictionary, then our next character is `m` with probabiliy 3/8, `[space]` with probability 1/2, and `i` with probability `1/8`.

Remember: the ***3rd***-order model requires you to compute ***4***-grams.

What you should do

Write a function `markov_text` that generates synthetic text according to an `n`-th order Markov model. It should have the following arguments:

- `s`, the input string of real text.
- `n`, the order of the model.
- `length`, the size of the text to generate. Use a default value of 100.
- `seed`, the initial string that gets the Markov model started. I used `"Emma Woodhouse"` (the full name of the protagonist of the novel) as my `seed`, but any subset of `s` of length `n` or larger will work.

It should return a string with the length of `len(seed) + length`.

Demonstrate the output of your function for a couple different choices of the order `n`.

Expected Output

Here are a few examples of the output of this function. Because of randomness, your results won't look exactly like this, but they should be qualitatively similar.

```
markov_text(s, n = 2, length = 200, seed = "Emma Woodhouse")
```

```
Emma Woodhouse ne goo thimser. John mile sawas amintrought
will on I kink you kno but every sh inat he fing as sat buty
aft from the it. She cousency ined, yount; ate nambery quirld
diall yethery, yould hat earatte
```

```
markov_text(s, n = 4, length = 200, seed = "Emma Woodhouse")
```

```
Emma Woodhouse!”—Emma, as love,           Kitty, only this
person no infering ever, while, and tried very were no do be
very friendly and into aid,    Man's me to loudness of
Harriet's. Harriet belonger opinion an
```

```
markov_text(s, n = 10, length = 200, seed = "Emma Woodhouse")
```

```
Emma Woodhouse's party could be acceptable to them, that if
she ever were disposed to think of nothing but good. It will
be an excellent charade remains, fit for any acquainted with
the child was given up to them.
```

Notes and Hints

Hint: A good function for performing the random choice is the `choices()` function in the `random` module. You can use it like this:

```
import random

options = ["One", "Two", "Three"]
weights = [1, 2, 3] # "Two" is twice as likely as "One", "Three"
three times as likely.

random.choices(options, weights)
```

```
['One'] # output
```

The first and second arguments must be lists of equal length. Note also that the return value is a list -- if you want the value *in* the list, you need to get it out via indexing.

Note: For grading purposes, the `options` should be the possible `n+1`-grams in the order of first appeareance in the text. If you are working through the strings from beginning to end, you will not have issues with this, as dictionary keys are ordered. Please do NOT use `random.seed()` in your function -- the autograder code will do it. You are welcome to try it out in your notebook for reproducible results if you are interested.

Hint: The first thing your function should do is call `count_ngrams` above to generate the required dictionary. Then, handle the logic described above in the main loop.

In [7]:
```
print(markov_text(s, n=2, length=200, seed="Emma Woodhouse"))
print(markov_text(s, n=4, length=200, seed="Emma Woodhouse"))
print(markov_text(s, n=10, length=200, seed="Emma Woodhouse"))
```

```
Emma Woodhouse coms by talwarrietemaked the usterfalwas sit Mis grom sustion
along yould.”

The quit forty.”

“She ounat thim their Chur such all aft wever be whimagembink muck saing the
of thearaped, I go he ing h
Emma Woodhouse, her, the breakfast, with a very name, Mr. Weston oftentimate
r ill the me too much a come could had just be on be pay coming in to seeing
than a mildness to be glad writ ion—as you are the Weston almo
Emma Woodhouse—how was Mr. Woodhouse, you will hear presently added, with a
little labour and great girls in Highbury. I dared not stay five minutes, an
d had often been negligent or perverse, slighting Mr. Weston c
```

Problem 4

Using a `for`-loop, print the output of your function for `n` ranging from `1` to `10` (including 10).

Then, write down a few observations. How does the generated text depend on `n`? How does the time required to generate the text depend on `n`? Do your best to explain each observation.

What do you think could happen if you were to repeat this assignment but in unit of words and not in unit of characters? For example, 2-grams would indicate two words, and not two characters.

What heuristics would you consider adding to your model to improve its prediction performance?

write your observations and thoughts here

Dependency of Generated Text on n: Low n Values (1-2): The text will likely be less coherent with more randomness. It might not form valid words consistently as the model only "remembers" 1 or 2 characters back. Medium n Values (3-6): The coherence of the text improves. Words start to form more accurately, and some phrase structures may seem logical due to the model capturing more of the local context. High n Values (7-10): The generated text may begin to closely mimic the style and structure of the source text. The higher the n, the more the model can "remember," leading to outputs that can

contain whole phrases or sentences that were in the original text. Time Required to Generate Text: The computation time generally increases with n because the model needs to manage a larger set of n-grams, and the dictionary of possible continuations grows. The time complexity can increase due to more complex matching operations and a larger search space for the next character.

In [8]:
```
# Assuming 's' contains the text data, for example from "Emma" by Jane Auste
seed = "Emma Woodhouse"  # A consistent seed for comparison

for n in range(1, 11):  # Loop from n = 1 to n = 10
    generated_text = markov_text(s, n, length=200, seed=seed)
    print(f"Output for n={n}:\n{generated_text}\n")
```

```
Output for n=1:
Emma Woodhouse winaveat wivevexctte tero n.


“Ahatthucig alf omerting ouingme wowitha onconde?—I co toutouco hereck hieal
dy f at st ionofitot an and’s.. Knd shasulk sere h!—ay, erishand byo ha arrn
yormet agf omewo

Output for n=2:
Emma Woodhousetwout I he appy he her lefuld odhought wis untima kinat all t
h. W. I sher, they, anight Robserfave brociss but hing th and ot day,” re sh
eme con thavend oved re afte have no conly anigh youll hings, d

Output for n=3:
Emma Woodhouse mined Mrs. Comething one was, of it comfore was examinevent.
Mr. Eltoget contired been write in the my regular appine was voidersong Isab
le and have till up that, and key remed not luck as matinute h

Output for n=4:
Emma Woodhouse of the of insist.

“Well!—why should declining half any awkwards Williance I cannot at her elbo
w, in the degreeable to make hers are as a proceed no recome assion short of
that was perfection; but yo

Output for n=5:
Emma Woodhouse!” cried Harriet Smith; and you will that he is son be always
at Mr. Dixons that is now I did not I should be on slowly—“so good Mr. Cole,
a more the Campbells or to Mrs. Weston’s particle mention in

Output for n=6:
Emma Woodhouse together home, that home afforded herself making of Harriet w
as companion; but one night’s absence, there is April come. Name your resolv
ed to each to understand the friend’s, I suppose Mrs. Goddard

Output for n=7:
Emma Woodhouse?—what can it be, Miss Woodhouse little more height, and said
not make due allowed by dint of Mrs. Elton. He gave me a brief sentence, wit
hout a blush unseen,
    ‘And waste its chance of insensible g

Output for n=8:
Emma Woodhouse, always say what we liked to him strong as he has outstepped
the good indeed he is not as likely to happen before, but it is an Irish fas
hion. Shall we have no happier than I could not allow for Harr

Output for n=9:
Emma Woodhouse, it is natural, you know the party did not give any connected
it to her—shall I?”—and as clear as possible!—She could stand the door—I was
quite easy. She doubted, and understanding about farming:—Th

Output for n=10:
Emma Woodhouse, how you were the person to do it, even without proof.”

“Proof indeed!” cried Emma, smiling, “what is the foolish girl about?”
```

```
“Oh! that’s a great deal of the best of it—and, indeed!—I beg you will
```

Problem 5

Try running your program with a different text!

You can

- find any movie script from https://imsdb.com/ or a book by Shakespeare, Hemingway, Beowulf, O.Henry, A.A. Milne, etc from https://www.gutenberg.org/
- ctrl + a to select all text on the page
- copy paste into a new `.txt` file. let's call it `book.txt`.
- put `book.txt` in the same folder as emma-full.txt.
- run the following code to read the file into variable `s`.

```
with open('book.txt', 'r') as f:
    s = f.read()
```

- run `markov_text` on `s` with appropriate parameters.

Show your output here. Which parameters did you pick and why? Do you see any difference from when you ran the program with Emma? How so?

In [9]:
```
# Load the text from the file
with open('book.txt', 'r', encoding='utf-8') as f:
    s = f.read()

# Specify the seed, n value, and length for the Markov model
seed = "Osage murders"  # Choose an appropriate seed related to the text
n = 5  # Higher n for more coherent longer memory
length = 300  # Generate 300 characters of text

# Generate text using the Markov model
output = markov_text(s, n, length, seed)
print(output)
```

```
Osage murders.

          golf at all...
       ERNEST
                     Submit    |    CUT TO:



231   INT. PITTS BEATY
     officials. We can help of
                We see if your house. He, Blackie Thompson with Minnie
CU. LIZZIE
   Yes. You smoked the
         I'll cut – that you so long aro
```

Chosen Parameters for "Killers of the Flower Moon": Seed: "Osage murders", n-value: 5. This is suggested by the relative coherence and occasional complete phrases in the output. Length: 300. Why These Parameters? Seed Choice: "Osage murders" is a pivotal theme within the text, making it a significant starting point for generating related content. n-value: A moderate to high n helps produce more coherent outputs that maintain some semblance of the original text's structure and style. Given the complex and detailed narrative style typical of non-fiction texts like "Killers of the Flower Moon," a higher n helps in capturing longer dependencies in text, such as names, specific terms, and partial sentences. Length: Typically set to provide enough content to observe the model's behavior without overwhelming randomness or repetition. Differences from Running the Program with "Emma": Content Style and Coherence: Emma: As n increases, the outputs become significantly more coherent and less random. Phrases and even complete sentences appear to be well-formed and contextually appropriate, reflecting the nuanced and complex sentence structures of Jane Austen. Killers of the Flower Moon: The output, even at a presumably higher n, contains more fragmented and abrupt shifts, likely reflecting a more varied use of language and structure in the source text. Non-fiction might also include more data, names, or factual information, which could disrupt flow when taken out of context. Dependency on n: Higher n values lead to markedly improved text quality in both cases. However, the improvement curve might be steeper for literary texts like "Emma" due to its more uniform and stylized use of language compared to a non-fiction text that might contain more diverse vocabularies and sentence structures. Randomness and Legibility: Outputs from "Emma" become increasingly legible and less random with higher n, showcasing more of the book's narrative style. The non-fiction text might still exhibit abrupt topic shifts even at higher n values, indicating a broader range of topics and less predictability in sentence structure.