Girls' Programming Network

Markov Chains

Workbook 2

This project was created by GPN Australia for GPN sites all around Australia!

This workbook and related materials were created by tutors at:

Sydney, Canberra and Perth

Girls' Programming Network

If you see any of the following tutors don't forget to thank them!!

Writers	**Testers**
Maddy Reid	Olivia Gorton
Renee Noble	Anton Black
Emma Krantz	Josie Ang
	Maddie Jones
	Kassandra di Bona
	Isabel Brison

Part 0: Setting up

In the first Markov Chain workbook, you used a dictionary to generate random things. **Now we're going to write code to create your own dictionary.** This means we can use different pieces of text to generate things!

Task 0.1: Here's one I prepared earlier...

Make sure your file from before is still open in IDLE. The file should be called markov_chains.py

Task 0.2: Making room for new code

Let's get started by making space for our new code!

1. Comment out the dictionary called `cups` by putting a '#' at the beginning of it.
2. Below that, make a brand new **empty** dictionary to replace it. Call that one cups as well.
3. Press enter a few times so that you have some space between that and the rest of your code.
4. Comment out the other code below that too, we'll use it later.

Hint

You can quickly comment out a block of code in IDLE by selecting it and clicking `Format > Comment Out Region` from the menu, or by using the keyboard shortcut `Alt+3`. To remove the comment you can use the keyboard shortcut `Alt+4`.

☑ CHECKPOINT ☑

If you can tick all of these off you can go to Part 1:

☐ You should have a file open with the Markov chain code you wrote last time

☐ You've commented out the dictionary at the top of the file and made a new empty one.

☐ The code you wrote last time is commented out

More Lists and Dicts

Part 1: New ingredients for new sentences

Task 1.1: Get some new source text

Let's give our program some new inspiration for what to generate.

1. Pick one of the texts from www.girlsprogramming.network/markov-files
2. Store it as a string in a variable. Give your variable a good, descriptive name like `source_text`.

Hint

Remember, in Python we call chunks of text *strings.* To tell python something is a string, we need to wrap it in quotes like
`'Hello, I am a string'` or `"Hello, I am a string"`.

Multiline strings use triple quotes:

```
"""
This is a multiline string.
I can put 'quotes' inside quotes.
"""
```

Task 1.2: Split the text into a list

Before we can do anything else, we need to split our source text into words—currently it's just one big blob of letters!

1. Turn our `source_text` string into a list of strings, where each string is a single word in a variable called split_text.

Hint

We can use .split() to do this.

```
blob = "Peter Piper picked a peck of pickled peppers"
blob.split()
['Peter', 'Piper', 'picked', 'a', 'peck', 'of', 'pickles',
'peppers']
```

Task 1.3: Word count

Store the number of words in split_text in a variable called num_words.

Hint

We could count each word ourselves. But python gives us a handy shortcut! len() will tell you the length of a list.

```
print(len(['I', 'love', 'cats']))
```
will print out 3.

☑ CHECKPOINT ☑

If you can tick all of these off you can go to Part 2:

☐ Store some text in a variable

☐ Split the text into a list of stringscss

☐ Use `len()` to count how many words are in the list and store that number in `num_words`

2. Loopy for lists

Task 2.1: Use a for loop to make the computer count for us

Using a for loop, get your program to count from 0 along the length of the `split_text` list.

If the length of `split_text` is 4, your program should print:

```
0
1
2
3
```

Hint

Remember using this in the first workbook?

```
for num in range(100):
    # The thing you want to do 100 times goes here
```

How could you change that to make it go for the number of words you have, not 100 times?

Task 2.2: Accessing each item in a list

Instead of printing out numbers, we want to print out the word in the list we are up to. Use `num` to access that correct spot in the list and print it out!

1. Store the word you're about to print in a variable called `current_word`
2. Print it out!

Hint

You can get items out of the list like this:

```
>>> words = ["I", "love", "cats"]
>>> words[2]
```

You can use the variable `num` to change which spot in the list you are grabbing every loop.`cats`

Task 2.3: Accessing the next item

Each time we loop, we also want to **look one word ahead** in our list.

1. Create a variable called `next_word`, set it to the word in the list after current_word
2. Print out `next_word`
3. What happens when you run your code?

Task 2.4: List index out of range

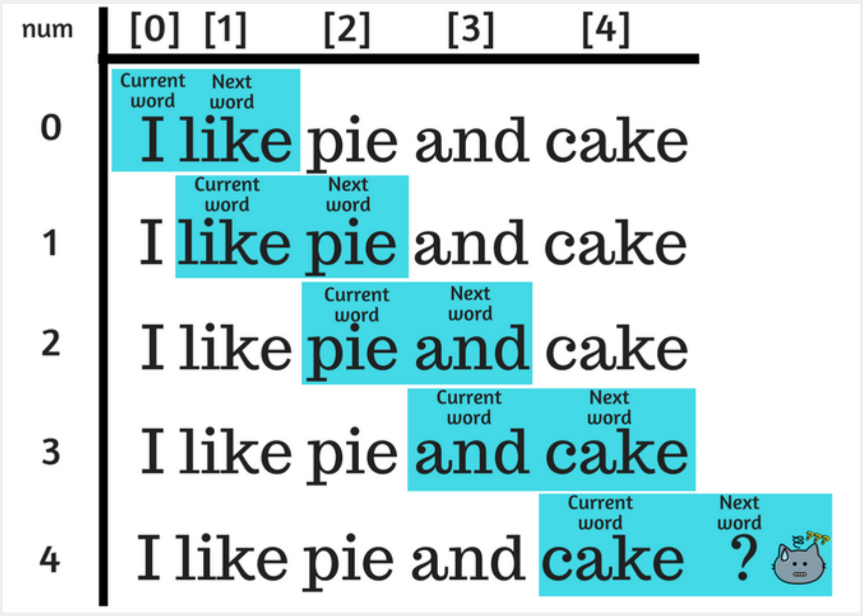
```
IndexError: list index out of range
```

Whoops! This error is what happens when we try to access something past the end of the list. The index we're trying to access is bigger than the number of things in the list.

See if you can figure out how to fix this, by changing how many times your loop runs

Hint

See what happens if we try to get the current word and the next word, for every word in "I like pie and cake". It gets a bit confusing at the end of the list or words!

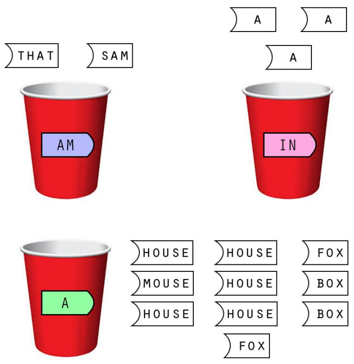


☑ CHECKPOINT ☑

If you can tick all of these off you can go to Part 3:

- ☐ Loop through each word in split_text, storing it in a variable
- ☐ Store the next word in a variable as well
- ☐ Stop your loop from trying to access past the end of the list

3. Fill the cups with words



Let's look at our cups again

Instead of actual cups, our program uses a **Python dictionary.**

The **keys** of the dictionary are the words written on the outside of the cup

The **values** of the dictionary are lists of words that come after the word written on the cup

Task 3.1: Add new words

For every different word in the text, we want to make a cup for it to hold all the possible words that might come after it.

1. Inside your loop, add `current_word` as a key to cups, and set the value to `next_word`.
2. After your loop, print out cups.

Hint

To create an entry in a dictionary called `phone`, you can use:

```
phone["Sam"] = 1023
```

Each dictionary entry is identified by a **key**, `"Sam"`, and a **value**, `1023`.

Task 3.2: When we see a word again...

As you might have noticed, something isn't quite right. What if we run into the same word twice? Our cup is only ever going to have one word in it. Instead of a single string, we need to store a list.

In your loop:

1. Remove your code from Task 3.1, we're going to replace it with something better!
2. Check `if current_word` is `not in` the `cups` dictionary
   (You can use `not in` just like you use `in`!)

3. **If it's not there**, add an entry to the dictionary.
   The **key** is the `current_word`, the **value** is a `list` containing one item, the `next_word`.

4. **Otherwise** append the `next_word` to the list that's already there.
5. Then check by printing out your cups dictionary outside the loop.

Hint

To add something to the end of a list, we use .append() as seen below:

```
favourite_things = ["raindrops on roses", "whiskers on
kittens"]
favourite_things.append("bright copper kettles")
```

Hint

Remember your green eggs and ham dictionary? It looks something like this (but longer):

```
cups = {'am': ['sam', 'that'],
        'in': ['a', 'a', 'a'],
        'a' : ['house', 'mouse', 'house', 'mouse', 'box',
               'fox', 'box', 'fox', 'house', 'mouse']
        ....}
```

The new dictionary you generate should look similar, but with different words in it.

☑ CHECKPOINT ☑

If you can tick all of these off you can go to Part 4:

☐ Your dictionary containing lists possible next words for each word in the text

☐ You've printed out the dictionary and made sure that it looks similar to the example one (but with different words).

4. Generating sentences

In the last step we finished our code that creates our dictionary from any text. Now we want to combine this with the code we wrote in Workbook 1, where we used a dictionary. Putting these two bits of code together means we can create our dictionary and then use it!

Task 4.1: Put it all together

Comment out the last print-statement. Uncomment your code from the first workbook, and run your program.

Task 4.2: Code that sometimes work

We will try out our code with different source texts and see what happens:

1. Run your code with the following text (source_text):

```
"""
i see whats happening yeah
 youre face to face with greatness and its strange
 you dont even know how you feel its adorable
 well its nice to see that humans never change
 open your eyes lets begin yes its really me
 its maui breathe it in
 i know its a lot the hair the bod
 when youre staring at a demigod what can i say except youre welcome
 for the tides the sun the sky
 hey its okay its okay youre welcome
"""
```

This text should work.

2. Now we try another text. Run your code again with the following text. Generate a text with at least 100 words.
(Hint: The for-loop should be: `for i in range(100)`):

```
"""
Another thing that got forgotten was the fact that against all probability a sperm whale had suddenly been called into existence several miles above the surface of an alien planet. And since this is not a naturally tenable position for a whale, this poor innocent creature had very little time to come to terms with its identity as a whale before it then had to come to terms with not being a whale any more. This is a complete record of its thoughts from the moment it began its life till the moment it ended it.
"""
```

What happens if you run your code several times? Did you get a `KeyError: 'it.'` ?

3. Think about why this is broken!

Hint

What happens if a word does not have any subsequent words, e.g. like the last word? Let's say we have three words in our text: "I love cats" The next_word for each of the words are:
I > love
love > cats
cats > ????
Since no word comes after the word cats it doesn't go into our dictionary, so our program breaks. Our dictionary `cups` would look like this:

```
>>> cups = {"I":  ["love"],
            "love" : ["cats"]
           }
>>> cups["cats"]

KeyError: "cats"
```

Task 4.3: Let's fix this!

We can prevent this by checking if the `current_word` is in our dictionary `cups`. Add an if statement inside the for loop that checks this. The following code should then be indented.

Now, run it again with the text that broke it last time! You might get a generated text that is shorter than 100 words but you won't get an error anymore.

Task 4.4: New texts

Try out your code with different source texts and see what happens!

☑ CHECKPOINT ☑

If you can tick all of these off you can go to Part 5:

☐ Your program generates random sentences without getting a KeyError for the last word in the text.
☐ You've tried three different source texts