Girls' Programming Network

Markov Chains

Workbook 2

Tutors Only

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

Maddy Reid
Renee Noble
Emma Krantz

Testers

Olivia Gorton
Anton Black
Josie Ang
Maddie Jones
Kassandra di Bona
Isabel Brison

A massive thanks to our sponsors for supporting us!

ATLASSIAN

Part 1: New ingredients for new sentences

Task 1.0: Making some room for new code

In the first Markov Chain workbook, you used a dictionary to generate random things.

Now we're going to write code to create our own dictionary. This means we can use different pieces of text to generate things!

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. You can bulk comment lines of code by selecting them and then pressing **ctrl** + **/**

Hint : An empty dictionary looks like this: fav_foods = {}

We now start coding above the commented code block.

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 girlsprogramming.network/markov-files
2. Store it as a string in a variable. Give your variable a good, descriptive name like `source_text`.
3. Make sure your new variable is below your new `cups` variable and above the commented out code.

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.
"""
```

TUTOR TIPS

- If you want to stop the program before it's done (e.g. if you've loaded a very large file), you can use Ctrl+C to quit the program early. Or you can go through the menu to "Shell" -> "Interrupt Execution" to do the same thing.
- The code should now look like this:

```
cups = {}
source_text = """
<a bunch of text here>
"""
```

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`. This will split the text every time it sees a space, making a list of all the individual words.

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']
```

TUTOR TIPS

- The code should now look like this:

```
cups = {}
source_text = """
<a bunch of text here>
"""
split_text = source_text.split()
```

☑ CHECKPOINT ☑

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

☐ 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

☐ Store some text in a variable

☐ Split the text into a list of strings

TUTOR TIPS

- The code should now look like this:

```
cups = {}
source_text = """
<a bunch of text here>
"""
split_text = source_text.split()
num_words = len(split_text)
```

2. Loopy for lists

Task 2.1: Word count

Now we need to make a new variable called `num_words` to store the number of words in `split_text` to do this we're going to find it's length. To make sure it's working, you should then print `num_words` and see if it makes sense based on what's in `split_text` (You can delete this once you see it works!)

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.

Task 2.2: 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?

TUTOR TIPS

- Make sure the correct variables are being referenced!
- The code should now look like this:

```
cups = {}
source_text = """
<a bunch of text here>
"""
split_text = source_text.split()
num_words = len(split_text)
for i in range(num_words):
    print(i)
```

Task 2.3: 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]
cats
```

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

TUTOR TIPS

- Make sure the correct variables are being referenced!
- The code should now look like this:

```
cups = {}
source_text = """
<a bunch of text here>
"""
split_text = source_text.split()
num_words = len(split_text)
for i in range(num_words):
    current_word = split_text[i]
    print(current_word)
```

Task 2.4: 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?

TUTOR TIPS

- This code will cause an IndexError on the last iteration - that's fixed in the next task, so don't be alarmed.
- The code should now look like this:

```
cups = {}
source_text = """
<a bunch of text here>
"""
split_text = source_text.split()
num_words = len(split_text)
for i in range(num_words):
    current_word = split_text[i]
    next_word = split_text[i+1]
    print(next_word)
```

Task 2.5: List index out of range

You should have gotten an error that says:

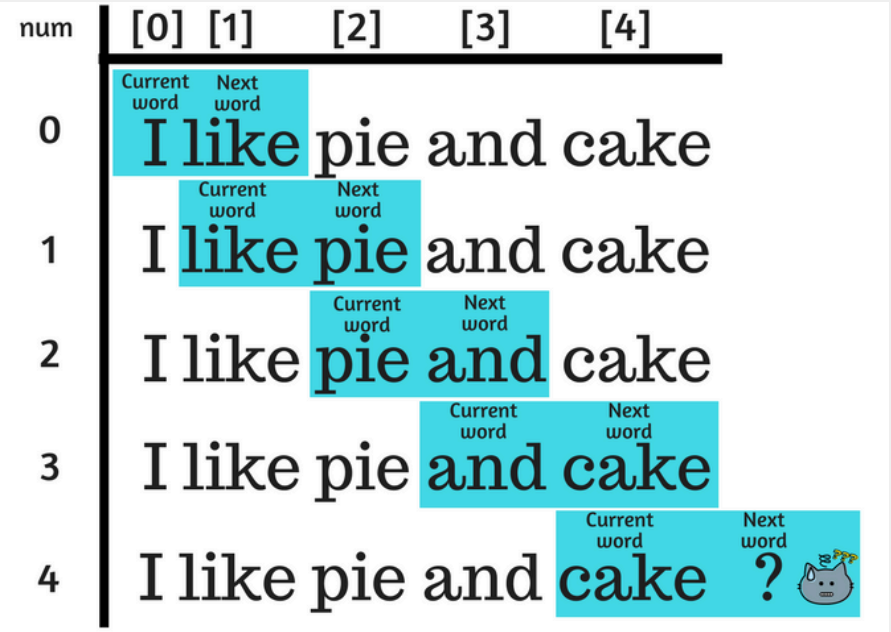
```
IndexError: list index out of range
```

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 spots in the list you look at. We want to fix the error but for now, don't worry if the first word is not printed.

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 of words!



☑ CHECKPOINT ☑

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

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

☐ Your code loops through each word in split_text, storing it in a variable

☐ Your code stores the next word in a variable as well

☐ Your code stops your loop from trying to access past the end of the list

TUTOR TIPS

- Make sure that the students are **using the index number (i)** to prevent the IndexError. **They shouldn't be checking the *contents* of the current word** - this method is "brittle" because it breaks as soon as you change the source text.
- The code should now look like this:

```
cups = {}
source_text = """
<a bunch of text here>
"""
split_text = source_text.split()
num_words = len(split_text)
for i in range(num_words - 1):
    current_word = split_text[i]
    next_word = split_text[i+1]
    print(next_word)
```

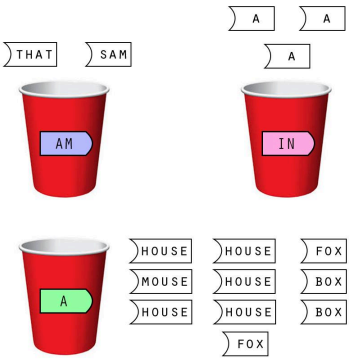
3. Fill the cups with words

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`.

TUTOR TIPS

- The code should now look like this:

```
cups = {}
source_text = """
<a bunch of text here>
"""
split_text = source_text.split()
num_words = len(split_text)
for i in range(num_words - 1):
    current_word = split_text[i]
    next_word = split_text[i+1]
    cups[current_word] = next_word
```

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

As you might have noticed, something isn't quite right. What happens when 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. Above where you add to the `cups` dictionary in 3.1, check if `present_word` is not in the `cups` dictionary (You can use not in just like you use in!)
2. If `present_word` is not in `cups`, modify your 3.1 code to add an entry to the dictionary. The key is the `present_word`, the value is a list containing one item, the `next_word`.
3. Add an elif or else statement to your if. In the case that `present_word` is in cups, append the `next_word` to the value list that's already there.
4. 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).

TUTOR TIPS

- The code should now look like this:

```
cups = {}
source_text = """
<a bunch of text here>
"""
split_text = source_text.split()
num_words = len(split_text)
for i in range(num_words - 1):
    current_word = split_text[i]
    next_word = split_text[i+1]

    if current_word not in cups:
        cups[current_word] = [next_word]
    else:
        cups[current_word].append(next_word)
print(cups)
```

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.

Hint

Hint: Remember you can use ctrl + / to bulk comment and un-comment code!

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 goes 100 times. Remember to indent your if statement inside of the for loop, and to indent all the code below it to move the code inside your if statement.
If you still have a section that prints something like "Sorry, you can't use that as a starting word", delete it.
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.

Now what happens if the `current_word` is "another"? Can you think of a reason why it might be breaking for a word that shouldn't be the end?

Which parts of the code will need to be lowercase in order to make sure there's no key errors due to capitalisation?

Hint

Hint: computers are really strict about the difference between capital and lower case letters. To make all letters in a string lowercase we need to use my_string.lower()

Task 4.4: New texts

Try out your code with different source texts and see what happens!
We've uploaded a bunch of files into this code space, to find them click the files button.

Hint

Hint: Here's a list of all the files you should have…

- `beatles.txt` lyrics from "The Yellow Submarine" by The Beatles
- `disney.txt` lyrics from "You're Welcome" the song from Moana
- `don't_panic.txt` an excerpt from "The Hitchhiker's Guide to the Galaxy" by Douglas Addams
- `feel-it-still.txt` lyrics from "Feel it Still" by Portugal
- `forklift_manual.txt` an excerpt from a forklift manual
- `harry_potter.txt` an excerpt from "Harry Potter and the Order of the Phoenix" by J. K. Rowling
- `lorde.txt` lyrics from "Royals" by Lorde
- `poe.txt` an excerpt from "The Raven" by Edgar Allen Poe
- `queen_b.txt` lyrics from "Run the World(Girls)" by Beyonce
- `recipes.txt` a collection of recipes
- `shakespeare_sonnets.txt` an excerpt of "Sonnet 18" by Shakespeare
- `taylor-swift.txt` lyrics from "Shake it Off" by Taylor Swift
- `terry_pratchet.txt` an excerpt from "The Colour of Magic" by Terry Pratchet
- `winnie_the_pooh.txt` an excerpt from "Winnie the Pooh" by A. A. Milne

☑ 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

TUTOR TIPS

- **Common mixups:** Depending on whether the student's commented code was left at the top or the bottom of their program, their program won't

necessarily start working straight away.
Look carefully and make sure that:
    - Their **name comment** is at the top
    - The **import** is at the top
    - The **rest of the old code** is ***below*** the new code
- Also, make sure they aren't overwriting the **cups** dictionary they've generated.
- The code should now look like this:

```
# <the student's name>
import random
cups = {}
source_text = """
<a bunch of text here>
"""
split_text = source_text.split()
num_words = len(split_text)
for i in range(num_words - 1):
    current_word = split_text[i]
    next_word = split_text[i+1]

    if current_word not in cups:
        cups[current_word] = [next_word]
    else:
        cups[current_word].append(next_word)

print("I am a markov chain generator")
current_word = input("What word do you want to start with? ")
print(current_word)

for i in range(100):
    if current_word in cups:
        next_word_options = cups[current_word]
        next_word = random.choice(next_word_options)
        print(next_word, end=" ")
        current_word = next_word
```