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

Maddy Reid
Renee Noble
Emma Krantz

Testers

Olivia Gorton
Anton Black
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 own 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 out 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

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 https://sites.google.com/site/girlsprogrammingnetwork/markov-chains-book-2
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.

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 strings

☐ 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 word you have, not 100 times?

Task 2.2: Accessing each item in a list

Instead of printing out number, 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.

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?

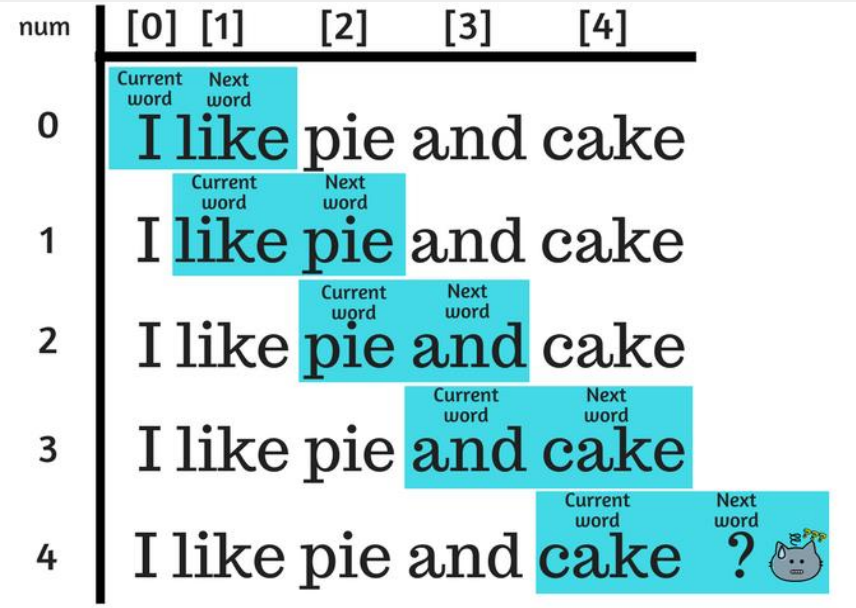
Task 2.4: List index out of range

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.

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 confused at the end of the list or words!



☑ CHECKPOINT ☑

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

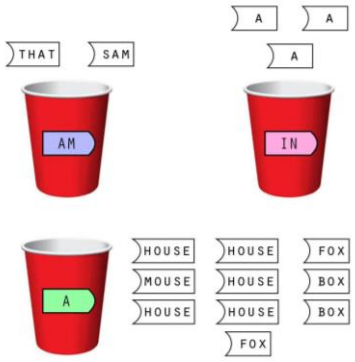
☐ Loop through each word in split_text, storing it in a variable

☐ Store the next word in a variable as well

☐ Stop your loop from trying to access past the end of the list

More

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.

Hint

To add something to the end of a list, we do this:

```
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

Task 4.1: Put it all together

Uncomment your code from the first workbook, and run your program. You should end up with some sentences generated from the text you chose.

Task 4.2: 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

☐ You've tried three different source texts

Files

5. Extension: longer texts

Task 5.1: Download a text

Instead of copying and pasting the text, let's make python do the reading for us! We'll get our text from a file!!

1. Go back to the link from before , and download a file that looks interesting. Save it as something like source_text.txt or harry_potter.txt.
   https://sites.google.com/site/girlsprogrammingnetwork/markov-chains-book-2

2. Make python read it's contents and store it in a variable called source_text variable.

Hint

Reading from python files is a two step process.

Step 1: If we had a file called "cats.txt" (which is in the same folder as your code) we could use:

```
f = open("cats.txt", "r")
```

Step 2: But that will just open the file. To put the text inside it in a variable, we then have to use:

```
text_inside_cats = f.read()
```

Task 5.2: Run your code!

What do the sentences you generate look like now? Are there any differences in the general readability or meaningfulness in the sentences? Why do you think that is?

☑ CHECKPOINT ☑

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

☐ Your program generates random sentences

☐ You've tried three different source texts

6. Extension: Messier texts!

Task 6.1: Find your own files!

You can find texts you like on the internet to use in your markov chain making!

1. File some text you like on the internet. Copy and paste it to a file (using a text editor on your computer like Notepad), save it as SOMETHING.txt

2. Update your code to read text from your new file. Run it and see what it looks like!

Task 6.2: Find your own files!

Did you notice anything when you ran your code? Did the text you generated have crazy punctuation and capitalisation?

The text you chose might be a little messier than the ones we've given you.
You might need to remove:

- Capital letters
- Punctuation
- New lines/Enters

We can can replace all these things we don't want in our text!

Use replace() to try removing different kinds of punctuation from you text. Look at the results and decide what you like best

Hint

You can replace text in a string like this:

This would replace a question mark with nothing (the empty string).

```
my_string = "Hi, how are you doing?"
my_string = my_string.replace("?", "")
```

This would replace a a new line/ENTER with a space.

```
my_string = """Hi, how are you doing?
               I really like cats"""
my_string = my_string.replace("\n", " ")
```

Enter is represented by \n, the N stands for New line.

Extensions - Trigram Generator

What are Tri-grams?

What we were writing before were bi-grams. This is where we had:
“Cat” -> “in”
“in” -> “the”

Now let’s do tri-grams! They are better than bi-grams, because when we chose a random word, they will have more context of what we have already said.
“The Cat” -> “in”
“Cat in” -> “the”

As you can see, we still have a key and a value pair, however they are better and contain more knowledge than before.

Ext. 1: Using Tri-grams

If you have a look at your previous code, you can see that we were creating bi-grams (which where 1 word leads to another word, a pair of words).

Try to modify your code so that you are creating tri-grams (where 2 words leads to a new word, at trio of words)! You will need to update your cup making code and your text generating code.

Check out the helper dictionary below to see what your cups dictionary will look like now!

Hint

When starting the program, ask for two words instead of one (which we did last time).

Keep track of the last 2 words you have used to randomly choose the next word!
You will need to add the two words together to use as your key!

Watch out when you create your dictionary not to look past the end of your list of words!

Hint

You can find the helper dictionary on the website as indicated by your tutor.

It will look something like this...

```
{'said Loonquawl': ['with', 'nervously,'],
'came to': ['life'],
'even so': ['they'],
'Loonquawl nervously,': ['do']... }
```