Girls' Programming Network

Markov Chains

Workbook 1

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
Caitlin Mangan

Testers

Vivian Dang
Annie Liu
Anton Black
Georgia Kirkpatrick-Jones
Olivia Gorton
Leesandra Fernandes
Josie Ang
Isabel Brison
Maddie Jones
Kassandra di Bona
Ashleigh Pribilovic
Lucy Wang

Intro

Part 0: Setting up

Task 0.1: Making a python file

Open the start menu, and type 'IDLE'. Select IDLE 3.5.

1. Go to the file menu and select 'new file'. This opens a new window.
2. Go to the file menu, select 'Save as' and save the file as 'markov_chain.py'

Task 0.2: You've got a blank space, so write your name!

At the top of the file use a comment to write your name!

Hint

Any line starting with # is a comment.

```
# This is a comment
```

☑ CHECKPOINT ☑

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

☐ You should have a file called markov_chain.py

☐ Your file has your name at the top in a comment

☐ Run your file with the F5 key and it does nothing!!

Part 1: Welcome message

Task 1.1: Print a message

We want to print a message to tell the user what our program does.

1. On the line after your name, use the `print` statement to display the following message:

```
I am a markov chain generator
```

2. Now **run your program** to see what happens!

Hint

You can print out a greeting like this:

```
print("Hello, world")
```

☑ CHECKPOINT ☑

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

☐ Print a message to the user

☐ Try running your code with the F5 key!

Part 2: The first word

Task 2.1: Get the user to choose the first word

1. Use `input` to ask the user for the first word in our sentence, and store their answer in a variable called `current_word`.

2. Use a `print` statement to display the `current_word`.

3. When you **run your program**, you should see something like this:

```
I am a markov chain generator
What word do you want to start with? a
a
```

Hint

You can get information from the user using `input`:

```
name = input("What is your name? ")
```

This will put the name the user enters into the variable called `name`.

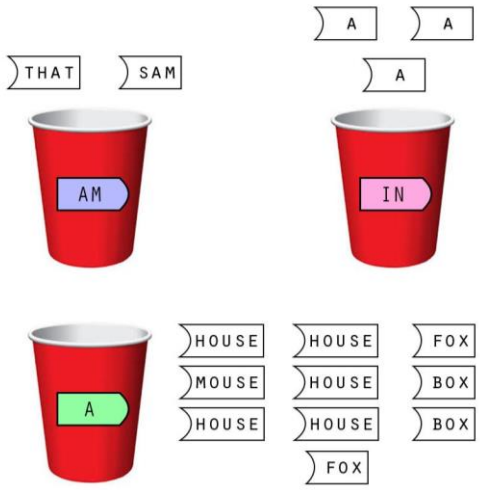
☑ CHECKPOINT ☑

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

☐ Get your program to print the word the user enters

☐ Run your program a few times and input a different word each time

Part 3: What comes next?



Remember the cups game from the start of the day, let's doing it with programming!

Instead of actual cups, our program will use a **Python dictionary** that tells us what's in the cup for each word. The label on each cup is the **key** of our dictionary. The matching **value** is a `list` of all the words that can come after it!

We'll give you all the cups/words to get you started

Task 3.1: Create the word cups

1. **Find the full dictionary at this link:** https://sites.google.com/site/girlsprogrammingnetwork/markov-chains-book-1
   The dictionary looks a bit like this (but longer):

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

2. **Copy and paste** the dictionary from the website. **Assign it to a variable called `cups`.**

3. Format the dictionary by going to top menu bar, click Format -> Format Paragraph. This will make the dictionary easier for us to read.

Hint: Understanding the cups dictionary

Each cup has a key-value pair (e.g. `'am': ['sam', 'that']`). The **key** (e.g. `'am'`) is the label on the cup and the **value** (e.g. `['sam', 'that']`) is a `list` of words inside the cup. Remember the words inside the cup are all the words that would possibly come next (after the label word).

Task 3.2: What could come after current_word?

Next we want to look up our `current_word` in our cups dictionary to find what could come next in the sentence we are making!

1. Look the `current_word` key up in the `cups` dictionary, and put the value in a variable called `next_word_options`.

2. Use a `print` statement to display the contents of `next_word_options`. (This is just to help us see what's happening, we will remove it later).

3. **Run your program** a few times using the following starting words and notice what happens. Write down the contents of `next_word_options` for each starting word below.

Starting word	next_word_options
i	
a	
do	

Hint

You can look at the **value** of a **key** in a dictionary using square brackets like this:

```
pet_names = {'dog': ['Spot', 'John'], 'cat': ['Snowball',
'Shadow'], 'mouse': 'Mr Squeaks'}
dog_name_suggestions = pet_names['dog']
```

Task 3.3: What if current_word doesn't exist?

Run your program using a starting word that is **not** in the `cups` dictionary, like `"dog"`.

Which of the following errors do you see? (We'll fix that later!)

A. `KeyError`
B. `IndexError`
C. `TypeError`

☑ CHECKPOINT ☑

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

- ☐ You have a variable with a dictionary of the cups
- ☐ You have another variable containing a list of all possible next words
- ☐ You've run your program using a few different starting words and seen what happens, and filled in the table in Task 3.2
- ☐ You know what kind of error happens if you start with a word that's not in cups

★ BONUS 3.4

You might have noticed that all the words in our cups dictionary are lower case. Because case is important in Python, we want to make the word the user enters lowercase too!

If the user puts "Sam" or "sAM" in as their start word we want to start with "sam" because "sam" is in the dictionary it wont' get an error!

1. Use `.lower()` on the input we get from the user to convert the word they enter to lowercase

2. **Run your program** and make sure words that are in the cups dictionary work even if they have some capital letters (like "sAM")

Python has a shortcut for making things lowercase, here is an example:

```
>>> name = input("What's your name? ")
KeLlY
>>> lowercase_name = name.lower()
>>> print(lowercase_name)
kelly
```

Rando

Part 4: Making choices

Task 4.1: Import Random Library

To get access to cool random things we need to import random!

1. At the top of your file, below your name, add this line:
   ```
   import random
   ```

Task 4.2: Choose a random word!

1. Randomly choose a word from `next_word_options`, and put that word in a new variable such as `next_word`.

2. Use `print` out `next_word`

Hint

If I wanted to choose a random food for dinner, I could use the `choice` function in the Python `random` library, like this:

```
dinner = random.choice(["pizza", "chocolate", "nutella", "lemon"]
```

Task 4.3:

Run your program several times, using **'a'** as the starting word each time, until you get 3 different words. **Write them down below!!**

Word #1: .....

Word #2: .....

Word #3: .....

☑ CHECKPOINT ☑

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

☐ The program randomly chooses a next word and prints it out

☐ Fill in the next word after 'a' for 3 different runs of your program

For

Part 5: Even more words

Task 5.1: Now let's do that 100 times!

That was so much fun we're going to do it 100 times!

Put all your code about choosing the next word (that's your code from Part 3 and 4) into a for loop that runs 100 times.

A **for loop** that runs 100 times looks like this:

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

Hint

When we put some code in an `if` or a `for` loop, we have to **indent** it. Indented lines have a tab (the blank space) at the start like this, only the indented things get repeated:

```
for blah in something:
    # THIS IS INDENTED
```

Task 5.2: Almost … but not quite

Run your program a few times using 'i' as the starting word. What do you notice?

Something's not quite right. We're always choosing from the first cup (the 'i' cup)! We need to update the current_word so we pick from the next cup!

Hint

Here's what 'i' looks like in the dictionary:

```
{..., 'i': ['am', 'am', 'do', 'do', 'do', 'would', 'would', 'do',
'do', 'do', 'do', 'do', 'do', 'do', 'do', 'would', 'would',
'would', 'do'], ...}
```

And here's what happens when we run our program starting with 'i'. We're only getting words that normally come after the word 'i':

```
I am a markov chain generator
What word do you want to start with? i
i do would would do would do am would do do do would would d
o do would would would do do do would am do am do do do do w
ould would do do do do do do do do do do do do do do do woul
d am do do do do do am am do would do do do do do do do do d
```

Task 5.3: Get the right cup

To fix the problem, we have to make sure we always look at the right cup. That means we have to *update* what the new current word is each time we print the next word.

1. At a line at the end of your for loop (don't forget your indenting!) set `current_word` to be the `next_word` we just printed.

2. When you **run your program** you should look a little like one of these! It should be different every time!

```
I am a markov chain generator
What word do you want to start with? i
i do not eat them here or there i am that sam-i-am would not like
green eggs and ham would you like them with a box would you like
them here or there i do not like them with a box would you like
them with a house ....
```

```
I am a markov chain generator
What word do you want to start with? i
i do not like green eggs and ham would you like them here or
there i do not like green eggs and ham i would not with a mouse i
am that sam-i-am i do not like them in a mouse i do not in a
mouse i do not with a mouse ...
```

```
I am a markov chain generator
What word do you want to start with? sam
sam i would you eat them sam-i-am i do not like green eggs and
ham would not like green eggs and ham i do not eat them anywhere
i do not like them sam-i-am i do not like them sam-i-am that sam-
i-am ...
```

Your program is writing the next word in the sentence based on the previous word to make a full sentence!

☑ CHECKPOINT ☑

If you can tick all of these off you're done!

☐ Your programs prints a silly sentence with 101 words (the starting word + 100 more)

☐ Each word is based on the previous word

★ BONUS 5.4: Too many lines!!

Woah that's hard to read!! Let's make it print on one line only!

When we use print("blah blah blah") it automatically does and ENTER! We don't want that.

But we can use tell it what to end the line with. This makes it end with 3 exclamation marks!

```
>>> print("blah blah blah", end="!!!")
blah blah blah!!!
```

Change the ending symbol of your print to make it print on one line! Don't forget to do it for when you print the first word too!

★ BONUS 5.5: Not so fast!!

This would look cooler if the computer wrote our story 1 word at a time. At the moment the computer goes so fast, it looks like it appears all at once!

1) At the top of your file write `import time`
   This will let us use what we need to use to make our program sleep for a few seconds.

2) Before any `print`, add a line that says `time.sleep(0.1)`
   This will make our program 'sleep' for a tenth of a second! You can adjust it to any time you want.

Part 6: I don't know that word!

If

Task 6.1: Check that current_word actually exists

We don't want an error if the user enters a starting word that is not in the `cups`!

1. **Use an `if` statement** to make sure the lines you added in Task 3.2 only run **if** `current_word` is `in` the `cups` dictionary.

2. **Run your program** again using a starting word that is not in the `cups` dictionary, such as `"dog"`. What happens now?

Hint

You can check if something exists in a dictionary using `if` ... `in` ... like this:

```
pet_names = {'dog': ['Spot', 'John'], 'mouse': 'Mr Squeaks'
             'cat': ['Snowball', 'Shadow']}
animal = input('Which animal do you have? ')
if animal in pet_names:
    print('I have some great name suggestions!')
```

Task 6.2: Remove the temporary print statement

Remove the `print` you added in Task 3.1 before moving on.

☑ CHECKPOINT ☑

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

☐ Your program doesn't give an error, even if the user enters a word not in the cups dictionary

★ BONUS 6.3:

If the user enters a starting word not in our cups dictionary, nothing happens!

Add an `else` statement to the `if` you added in Part 3.5. Use a `print` statement (remember your indenting!) so that if the user tries a word not in the dictionary, they will see a message like this:

```
Sorry, you can't use that as a starting word!
```

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).

Now try to modify your code so that you are creating tri-grams (where 2 words leads to a new word, at trio of words)!

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!

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']... }
```