Generating strings of letters

Here we are going to write a program to analyse the 12 strings we saw previously, work out the transition probabilities, and use this to generate some random strings that share the properties of this set.

We could count the letter beginnings and look for As, Bs and Cs, but instead it would be good to code this in a way that doesn't rely on us doing this work by hand, so we can swap it for different text without many changes.

So first we'll store the letters we are going to use in a list.

```
letters = ["A","B","C"]
```

We'll also store the 12 strings in a list.

```
strings = ["AAB", "AAC", "ABA", "ACA", "ACC", "BAA", "BAB", "BAC", "CAA", "CAB
```

We can loop through this list outputting the strings like this:

```
for string in strings:
    print(string)
```

We are interested in the first two letters from each string, which we can find using `string[0:2]`. We can print out the first two characters of each string like this:

```
for string in strings:
    print(string[0:2])
```

Aside: dictionaries

A dictionary is a more complicated data structure than a list. A list just stores a set of items in order. A dictionary gives each item a label, called a key, and you can use the key to refer to the item. You define a list using `{...}` and specific item keys and values using `key: value`.

Here we set up a dictionary `offices` which uses the names of three people as keys and their office numbers as the values.

```
offices = {"Angharad": "Norfolk 607", "Peter": "Norfolk 601", "Ros": "Norfolk
```

Having set this up, we can ask for people's office numbers using their names, rather than having to remember which order we put them into the dictionary.

```
print(offices["Ros"])
```

You can test whether a value `x` is in a list `a` using `if x in a:`. Here we can test whether a value is in the dictionary items using `values` and whether it is the name of one of the keys using `keys()`, like this:

```
if "Norfolk 601" in offices.values():
    print("Yes it is")

if "Peter" in offices.keys():
    print("Yes it is")
```

Back to our strings

For each string, we printed out the first two characters using `string[0:2]`. Actually, what we'd like is for each of these `string[0:2]` pairs of letters to count how many times they are followed by 'A', 'B' or 'C', the letters we stored in `letters`.

To do this, we will use a dictionary. This will use `string[0:2]` as keys and count how many times each occurs in the 12 strings.

First we make a blank dictionary called `starts`.

```
starts = {}
```

Now as we loop through all the strings we look to see if we've already stored its two-character start in `starts`. If we haven't (`not in`), then we add a new entry to `starts` with these two-characters `string[0:2]` as the key. For this new entry, we set up a dictionary with a count of `0` for each of the letters in `letters`.

```
for string in strings:
    if string[0:2] not in starts.keys():
        starts[string[0:2]] = {}
        for letter in letters:
            starts[string[0:2]][letter] = 0
```

This is quite complicated, but for example if `string` starts "`AB`" and this is the first time we've seen "`AB`", then a new dictionary is set up at `starts["AB"]` with the contents `{"A": 0, "B": 0, "C": 0}`.

Having made sure `starts[string[0:2]]` exists, we now want to look at what the third character is (`string[2]`) and add one (`+= 1`) to the count for that character. Adding that into the loop code above, we get this.

```
for string in strings:
    if string[0:2] not in starts.keys():
        starts[string[0:2]] = {}
        for letter in letters:
            starts[string[0:2]][letter] = 0

    starts[string[0:2]][string[2]] += 1
```

If you want to see what this produces, use

```
print(starts)
```

Hopefully you will see that this has made a first entry `AA` with counts of `0` for `A`, `1` for `B` and `1` for `C`, then a second entry `AB` with counts of `1`, `0` and `0`, and so on.

This variable `starts` is a representation of our transition table, and we can use it to build strings that loop like the 12 strings we started with.

For example, say we want to start a string with "`AA`" and see what happens next.

The counts for the number of times "`AA`" is followed by `A`, `B` and `C` are stored in `starts["AA"]`. We will get the next letter using `choices` from the `random` package, which makes a *weighted* random choice from a list. To do this we need the list of options and a list of weights.

The options are already stored in `letters`.

For the weights, these are stored in `starts["AA"]`. Because `choices` wants the weights to be a list, we convert this dictionary to a list like this:

```
weights = list(starts["AA"].values())
```

Now we can ask `choices` to return a random letter weighted with how often this letter follows "`AA`" in the 12 strings. The `[0]` is because `choices` returns a list and we just want the first item.

```
from random import choices
next_letter = choices(letters, weights)[0]
print("AA" + next_letter)
```

We can adjust this code to any starting pair of letters:

```
start = "AA"
weights = list(starts[start].values())
next_letter = choices(letters,weights)[0]
print(start + next_letter)
```

We can ask our program to produce longer strings than three letters. For example, to add ten new letters to the string following the pattern from the original 12 strings, we would use a for loop.

Here we build a string `output`, starting with `"AA"`. In each iteration of the loop, we take the last two characters (`output[-2:]`) and use the weights associated with that start to make a random choice for the next letter. At the end we print our `output` string.

```
output = "AA"
for i in range(10):
    start = output[-2:] # last two characters
    weights = list(starts[start].values())
    next_letter = choices(letters,weights)[0]
    output += next_letter

print(output)
```

Exercises

1. Change the possible letters to 'X', 'Y' and 'Z' and use these input letters:

```
XXY
XXZ
XYZ
XZX
XZY
YXX
YZX
YZZ
ZXX
ZYX
ZYZ
ZZY
```

2. Change the code to use four input letters of your choice and make some input strings using those four letters. Be careful: every pair of letters in your input strings must be the start of another input string.

3. Change the code to use strings of length four, predicting the fourth letter based on the preceeding three.