Practical 3: Foundations (Part 2)

Getting to grips with Dictionaries, LOLs and DOLs

Table of contents

In this notebook we are exploring basic (in the sense of fundamental) data structures so that you understand both how to manage more complex types of data and are prepared for what we will encounter when we start using `pandas` to perform data analysis. To achieve that, you will need to be 'fluent' in "nested" lists and dictionaries; we will focus primarily on lists-of-lists and dictionaries-of-lists, but note that file formats like JSON can be understood as dictionaries-of-dictionaries-of-lists-of-... so this is just a *taster* of real-world data structures.

> **Tip**
>
> You should download this notebook from GitHub and then save it to your own copy of the repository. I'd suggest adding it (`git add ...`) right away and then committing (e.g. `git commit -m "Just added."`). Do this again at the end of the class (`git add ...`, `git commit -m ...`) and you'll have a record of everything you did. Then you can `git push` everything to GitHub.

> **Group Sign-Up**
>
> While there's no benefit to signing up early, if you've already formed a group then you can now Sign Up!

1 From Lists to Data (Little Steps)

We're going to start off using lists and dictionaries that *we* define right at the start of the 'program', but the *real* value of these data structures comes when we build a list or dictionary *from* data such as a file or a web page... and that's what we're going to do below!

First, here's a reminder of some useful methods (*i.e.* functions) that apply to lists which we covered in the lecture and practical in Week 2:

Method	Action
`list.count(x)`	Return the number of times x appears in the list
`list.insert(i, x)`	Insert value `x` at a given position `i`
`list.pop([i])`	Remove and return the value at position `i` (`i` is optional)
`list.remove(x)`	Remove the first element from the list whose value is `x`
`list.reverse()`	Reverse the elements of the list in place
`list.sort()`	Sort the items of the list in place
`list.index(x)`	Find the first occurence of `x` in the list
`list[x:y]`	Slice the list from index `x` to `y-1`

This should all be revision... because it's how we finished things up *last week*. But I want to go over it *briefly* again because we're going to build on it this week.

> **Hint**
>
> As before, `??` will highlight where one or more bit of code are missing and need to be filled in...

1.1 List Refresher

> **Difficulty: Low.**

To complete these tasks, all of the methods that you need are listed above, so this is about testing yourself on your understanding *both* of how to read the help *and* how to index elements in a list.

The next line creates a list of city names (each element is a string):

```
cities = ["Bristol", "London", "Manchester", "Edinburgh", "Belfast", "York"]
```

1.1.1 List Arithmetic

Replace the `??` so that it prints `Belfast`.

Question

```
print(cities[?? + 2])
```

1.1.2 Negative List Arithmetic

Use a **negative** index to print `Belfast`:

Question

```
print(cities[??])
```

1.1.3 Finding a Position in a List

Replace the `??` so that it prints the *index* for Manchester in the list.

Question

```
print("The position of Manchester in the list is: " + str( ?? ))
```

1.2 Looking Across Lists

> **Connections**
>
> This section draws on the LOLs lecture and you will also find Code Camp's Loops session useful here.

> **Difficulty: Medium.**

Notice that the list of `temperatures` below is the same length as the list of `cities`, that's because these are (roughly) the average temperatures for each city.

```
cities = ["Bristol", "London", "Manchester", "Edinburgh", "Belfast", "York"]
temperatures = [15.6, 16.5, 13.4, 14.0, 15.2, 14.8]
```

1.2.1 Lateral Thinking

Given what you know about `cities` and `temperatures`, how do you print:

> "The average temperature in Manchester is 13.4 degrees"

But you have to do this *without* doing any of the following:

1. Using a list index directly (*i.e.* `cities[2]` and `temperatures[2]`) or
2. Hard-coding the name of the city?

To put it another way, **neither** of these solutions is the answer:

```
print("The average temperature in Manchester is " + str(temperatures[2]) + " degrees
# ...OR...
city=2
print("The average temperature in " + cities[city] + " is " + str(temperatures[city]
```

> **Tip**
>
> You will need to combine some of the ideas above and also think about the fact that the list index is that we need is the same in both lists... Also, remember that you'll need to wrap a `str(...)` around your temperature to make it into a string.

Question

```
city="Manchester" # Use this to get the solution...

# This way is perfectly fine
print("The average temperature in " + ?? + " is " + str(??))

# This way is more Python 3 and a bit easier to read
print(f"The average temperature in {??} is {??}")
```

1.2.2 Double-Checking Your Solution

You'll know that you got the 'right' answer to the question above if you can copy+paste your code and change only **one** thing in order to print out: "The average temperature in Belfast is 15.2 degrees"

Question

```
city="Belfast"
print(??)
```

1.2.3 Loops

Now use a `for` loop over the cities to print out the average temperature in each city:

Question

```
for c in cities:
    ??
```

The output should be:

```
The average temperature in Bristol is 15.6
The average temperature in London is 16.5
The average temperature in Manchester is 13.4
The average temperature in Edinburgh is 14.0
The average temperature in Belfast is 15.2
The average temperature in York is 14.8
```

2 Dictionaries

> **Connections**
>
> This section draws on the Dictionaries lecture and Code Camp Dictonaries session.

> **Difficulty: Low.**

Remember that dictionaries (a.k.a. dicts) are like lists in that they are data structures containing multiple elements. A key difference between dictionaries and lists is that while elements in lists are ordered, dicts (in most programming languages, though not Python) are unordered. This means that whereas for lists we use integers as indexes to access elements, in dictonaries we use 'keys' (which can multiple different types; strings, integers, etc.). Consequently, an important concept for dicts is that of key-value pairs.

2.1 Creating an Atlas

The code below creates an Atlas using a dictionary. The dictionary `key` is a city name, and the `value` is the latitude, longitude, and main airport code.

```
cities = {
    'San Francisco': [37.77, -122.43, 'SFO'],
    'London': [51.51, -0.08, 'LDN'],
    'Paris': [48.86,2.29, 'PAR'],
    'Beijing': [39.92,116.40 ,'BEI'],
}
```

2.1.1 Adding to a Dict

Add a record to the dictionary for Chennai (data here)

Question

```
??
```

2.1.2 Accessing a Dict

In *one* line of code, print out the airport code for Chennai (`MAA`):

Question

```
??
```

2.2 Dealing With Errors

Check you understand the difference between the following two blocks of code by running them.

```
try:
    print(cities['Berlin'])
except KeyError as e:
    print("Error found")
    print(e)
```

```
Error found
'Berlin'
```

```
try:
    print(cities.get('Berlin','Not Found'))
except KeyError as e:
    print("Error found")
    print(e)
```

```
Not Found
```

Notice that trying to access a non-existent element of a dict triggers a `KeyError`, while asking the dict to `get` the *same element* does not, it simply returns `None`. Can you think why, depending on the situtation, *either* of these might be the 'correct' answer?

2.3 Thinking Data

This section makes use of both the Dictionaries lecture and the DOLs to Data lecture.

> **Tip**
>
> In this section you'll need to look up (i.e. Google) and make use of a few new functions that apply to dictionaries: `<dictionary>.items()`, `<dictionary>.keys()`. *Remember*: if in doubt, add `print(...)` statements to see what is going on!

2.3.1 Iterating over a Dict

Adapting the code below, print out the city name and airport code for every city in our Atlas.

Question

```
for c in cities.keys():
    print(??)
```

The output should look something like this:

```
San Francisco -> SFO
London -> LDN
Paris -> PAR
Beijing -> BEI
Chennai -> MAA
```

2.3.2 More Complex Dicts

How would your code need to change to produce the *same output* from this data structure:

```
cities = {
    'San Francisco': {
        'lat': 37.77,
        'lon': -122.43,
        'airport': 'SFO'},
    'London': {
        'lat': 51.51,
        'lon': -0.08,
        'airport': 'LDN'},
    'Paris': {
        'lat': 48.86,
        'lon': 2.29,
        'airport': 'PAR'},
    'Beijing': {
        'lat': 39.92,
        'lon': 116.40,
        'airport': 'BEI'},
    'Chennai': {
        'lat': 13.08,
        'lon': 80.28,
        'airport': 'MAA'}
}
```

Question

```
for c in cities.keys():
    print(??)
```

2.3.3 More Dictionary Action!

And how would it need to change to print out the name and latitude of every city?

Question

```
for c in cities.keys():
    print(??)
```

The output should be something like this:

```
San Francisco is at latitude 37.77
London is at latitude 51.51
Paris is at latitude 48.86
Beijing is at latitude 39.92
Chennai is at latitude 13.08
```

2.3.4 And Another Way to Use a Dict

Now produce the *same output* using this new data structure:

```
cities_alt = [
    {'name':     'San Francisco',
     'position': [37.77, -122.43],
     'airport':  'SFO'},
    {'name':     'London',
     'position': [51.51, -0.08],
     'airport':  'LDN'},
    {'name':     'Paris',
     'position': [48.86, 2.29],
     'airport':  'PAR'},
    {'name':     'Beijing',
     'position': [39.92, 116.40],
     'airport':  'BEI'},
    {'name':     'Chennai',
     'position': [13.08, 80.28],
      'airport': 'MAA'}
]
```

Question

```
for c in cities_alt:
    print(??)
```

The output should be something like this:

```
San Francisco is at latitude 37.77
London is at latitude 51.51
Paris is at latitude 48.86
Beijing is at latitude 39.92
Chennai is at latitude 13.08
```

Answer

```
for c in cities_alt:
    print(f"{c['name']} is at latitude {c['position'][0]}")
```

```
San Francisco is at latitude 37.77
London is at latitude 51.51
Paris is at latitude 48.86
Beijing is at latitude 39.92
Chennai is at latitude 13.08
```

2.3.5 Think Data!

What are some of the main differences that you can think of between `cities` and `cities_alt` *as* data? There is no right answer.

> **Tip**
>
> I just want you to think about these *as data*! If you were trying to use `cities` and `cities_alt` as data what differences would you find when accessing one or more 'records'?

- Point 1 here.
- Point 2 here.
- Point 3 here.