# Nested data and iteration

## Warm-up

**1.** Write a function called `word_freq_counts` that takes in a string and then prints a table of words in the string in alphabetical order together with the number of times each word occurs. Case should be ignored. A sample run of the program might look this this:

```
freq_counts('ThiS is String with Upper and lower case Letters String is awesome')
```

```
and 1
awesome 1
case 1
is 2
letters 1
lower 1
string 2
this 1
upper 1
with 1
```

```python
def word_freq_counts(text: str) -> None:
    freq_dict = {}
    for word in text.lower().split():
        if word not in freq_dict:
            freq_dict[word] = 0
        freq_dict[word] +=1
    for k in sorted(freq_dict):
        print(k, freq_dict[k])
```

# Model 1 Nested Lists

Elements in a list can be of sequence type (string or list), for example, in a list of words, each element is a string type. Similarly, here is an example of a list of lists:

```
states = [
    ['AL','AK','AZ','AR'],
    ['CA', 'CO','CT'],
    ['DC','DE'],
    ['FL'],
    ['GA'],
    ['HI'],
    ['ID','IL','IN','IA']
]
```

The `states` list contains sub-lists with states that start with the same letter.

**2.** Evaluate each expression in order and record the output for each line in the second column.

| Python code | Output |
|---|---|
| `print(states[0])` | ['AL','AK','AZ','AR'] |
| `print(states[-1])` | ['ID','IL','IN','IA'] |
| `print(states[4][-1])` | `'GA'` |
| `print(states[5][0])` | `'HI'` |
| `print(len(states))` | 7 |
| `print(len(states[1]))` | 3 |
| `print(len(states[3]))` | 1 |
| `print(len(states[3][0]))` | 2 |
| `print(len(states[3][1]))` | Index Error (Runtime) |
| `print(states[3][0][0])` | `'F'` |

**3.** What does the following code snippet print?

```
1 for sublist in states:
2     letters = ''
3     for state in sublist:
4         letters += state[1]
5     print(letters)
```

```
LKZR
AOT
CE
L
A
I
DLNA
```

**4.** Modify the code in the previous problem to print all the letters inside the list, that is: `'ALAKAZARCACOCTDCDEFLGAHIIDILINIA'`

```
letters = ''
for sublist in states:
    for state in sublist:
        letters += state

print(letters)
```

**5.** Write a function called `max_states` that takes in the list of states and returns the maximum size of its sublists.

```
def max_states(states: list) -> int:
    max = 0
    for sublist in states:
        if len(sublist) > max:
            max = len(sublist)
    return max
```

**6.** Write a function called `min_states` that takes in the list of states and returns the first sublist with minimum size.

```
def min_states(states: list) -> list:
    min_list = states[0]
    for sublist in states:
        if len(sublist) < len(min_list):
            min_list = sublist
    return min_list
```

**7.** **Challenging:** Modify the code in the previous problem to print all the unique letters inside the list, that is: `'ACDFGHILKZROTEN'`

```
letters = ''
for sublist in states:
    if sublist[0][0] not in letters:
        letters += sublist[0][0]

for sublist in states:
    for state in sublist:
        if state[1] not in letters:
            letters += state[1]

print(letters)
```

# Model 2 Nested Dictionaries

Collections/containers (sequence-type like strings and lists, and dictionaries/maps) can be nested in arbitrary ways. For example, the following data could be described as a "dictionary of dictionaries of integers and lists of strings":

```
movies = {
    "Casablanca": {
        "year": 1942,
        "genres": ["Drama", "Romance", "War"],
    },
    "Star Wars": {
        "year": 1977,
        "genres": ["Action", "Adventure", "Fantasy"],
    },
    "Groundhog Day": {
        "year": 1993,
        "genres": ["Comedy", "Fantasy", "Romance"],
    },
}
```

**8.** Evaluate the following expressions in the order that they are listed:

| Python code | Output |
|---|---|
| `movies` | prints all of `movies` without any formatting |
| `movies["Casablanca"]` | {'genres': ['Drama', 'Romance', 'War'], 'year': 1942} |
| `movies["Casablanca"]["year"]` | 1942 |
| `movies["Casablanca"]["genres"]` | ['Drama', 'Romance', 'War'] |
| `type(movies)` | <class 'dict'> |
| `type(movies["Casablanca"])` | <class 'dict'> |
| `type(movies["Casablanca"]["year"])` | <class 'int'> |
| `type(movies["Casablanca"]["genres"])` | <class 'list'> |
| `len(movies)` | 3 |
| `len(movies["Casablanca"])` | 2 |
| `len(movies["Casablanca"]["year"])` | TypeError: object of type 'int' has no len() |
| `len(movies["Casablanca"]["genres"])` | 3 |
| `for key in movies:`<br>`  print(key)` | prints the keys: Casablanca, Groundhog Day, Star Wars |
| `for key, val in movies.items():`<br>`  print(key, val)` | prints each individual movie (the inner dictionaries) |

**9**. Explain the `TypeError` you encountered.

The expression `movies["Casablanca"]["year"]` is an integer, and you can't get the "length" of an integer.

**10**. In the expression `movies["Casablanca"]["genres"]`, describe the purpose of the strings `"Casablanca"` and `"genres"`.

They are keys to their corresponding dictionaries. The first string selects a particular movie, and the second string selects the corresponding movie data.

**11**. When iterating a dictionary using a `for` loop (i.e., `for x in movies`), what gets assigned to the variable?

The keys of the dictionary.

**12**. What is wrong with the following code that attempts to `print` each movie?

```
for i in range(len(movies)):
    print(movies[i])
```

You cannot iterate a dictionary by index number; it is not a sequence. Running this code results in `KeyError: 0`.

**13**. Write nested loops that outputs (prints) every *genre* found under the `movies` dictionary. Trace your code to ensure that it outputs a total of nine lines.

```
for key in movies:
    movie = movies[key]
    for genre in movie["genres"]:
        print(genre)
```

**14**. Each movie in Model 2 has a title, a year, and three genres.

a) Is it necessary that all movies have the same format? No

b) Name one advantage of storing data in the same format: It simplifies the code

c) Show how you would represent The LEGO Movie (2014) with a runtime of 100 min and the plot keywords "construction worker" and "good cop bad cop".

```
    "The LEGO Movie": {
        "year": 2014,
        "runtime": "100 min",
        "keywords": ["construction worker", "good cop bad cop"],
    },
```