Comprehensions and Generators

COMPREHENSIONS

- Comprehensions are a quick way of generating or altering Lists, Sets or Dictionaries. They provide a powerful range of functionality using a single line of code.

List	**[List of Values]**
Set	**{Unique Values}**
Dictionary	**{Label: Set}**

List Comprehensions

Let’s imagine we had a list (or array) of strings, as follows:

- `string_array = [“234", "75", "331", "73", "5"]`

If we wanted to take each element and covert them into integers, we could do:

```
Output1 = [int(num) for num in string_array]
           \_______/ \___________________/
 Convert num to integer  for each number in the list
```

Which would give us:

```
[234, 75, 331, 73, 5]
```

If we wanted to only convert strings less than three characters long:

```
output2 = [int(num) for num in string_array  if len(num) < 3]
           \_______/ \___________________/  \_____________/
 Convert num to integer  for each number in the list  if string length less than 3
```

Which would give us:

```
[75, 73, 5]
```

Set Comprehensions

A set is like a list but with no duplicate entries. We create a set is to use the `set()` constructor to convert a list into a set, but we can also use a set comprehension.

```
f-authors = {b.author for b in books if b.genre == 'fantasy'}
              \______/ \__________/ \_____________________/
   Add unique author  for each book in list  if genre attribute is “fantasy”
```

Which would give us the following output (removing any duplicate values):

```
{'Pratchett', 'Le Guin', 'Turner'}
```

Dictionary Comprehensions

A dictionary is a list that has a label at the start of it, we can use an existing dataset and convert it into a dictionary using a dictionary comprehension.:

```
f-titles = {b.title: b for b in books if b.genre == 'fantasy'}
            \______/ \_/ \__________/ \_____________________/
           Add label set for each book in list if genre attribute is “fantasy”
```

Now we have a dictionary we can look up using title.

Comprehensions and Generators

GENERATORS

- Generators have a similar synthax to Comprehensions, but work on Tuples instead of Lists, Sets and Dictionaries.

Generator **(List of Values)**

Generator Experssions

If we were processing a large log file that had lots of lines in it, some of which had the word "WARNING" in it. If the log file was very big (terabytes) and we were looking only for lines with the word "WARNING" in them, we shouldn't use a List Comprehension in this case because they would temporarily create a list containing every line and then search for the appropriate messages.

Instead if we use a generator, we avoid that issue because generators don't create a temporary list, they only write content out when they are instructed to, this is sometimes called *Lazy Evaluation*, and it allows you to start using the list immediately:

```
import sys
InName = "InputFile.txt"
OutName = "OutputFile.txt"

with open(InName) as infile:
  with open(OutName, "w") as outfile:
    warnings = (line for line in infile if 'WARNING' in line)
    for line in warnings:
        outfile.write(line)
    # ENDFOR;
# END.
```

So the most important line in the program is as follows:

$$\texttt{warnings = }\underbrace{\texttt{(line}}_{\text{Add this line in}}\ \underbrace{\texttt{for line in infile}}_{\text{for each line in the file}}\ \underbrace{\texttt{if 'WARNING' in line)}}_{\text{if 'WARNING' is in line}}$$

If we wanted to remove the word "WARNING" out of each line, we could do:

$$\texttt{W = }\underbrace{\texttt{(line.replace(' WARNING', '')}}_{\text{Replace 'WARNING' with blank}}\ \underbrace{\texttt{for line in file}}_{\text{for each line in the file}}\ \underbrace{\texttt{if 'WARNING' in line)}}_{\text{if 'WARNING' is in line}}$$

If we wanted to do the same thing in a more object-oriented manner, we could use the following code. Note that the `yield` command works exactly like the normal `return` command (when you return a value from a method), but it temporarily returns control to the calling method, and remembers where it was in a sequence in each new call.

```
def warnings_filter(insequence):
    for line in insequence:
        if 'WARNING' in line:
            yield line.replace(' WARNING', '')
```

This will return the same result as the previous code snippet.