LIST COMPREHENSIONS

Quick Recap

You should already know what list comprehensions are, but let's quickly recap their syntax and how they work:

goal → generate a list by transforming, and optionally filtering, another iterable

- start with some iterable
- create empty new list
- iterate over the original iterable
- skip over certain values (filter)
- transform value and append to new list

```
other_list = ['this', 'is', 'a', 'parrot']

new_list = []

for item in other_list:

        if len(item) > 2:

            new_list.append(item[::-1])
```

List comprehension:

```
new_list = [item[::-1] for item in other_list if len(item) > 2]
```

transformation — iteration — filter

Formatting the Comprehension Expression

If the comprehension expression gets too long, it can be split over multiple lines

For example, let's say we want to create a list of squares of all the integers between `1` and `100` that are not divisible by `2`, `3` or `5`

```
sq = [i**2 for i in range(1, 101) if i%2 and i%3 and i%5]
```

We could write this over multiple lines:

```
sq = [i**2
      for i in range(1, 101)
      if i%2 and i%3 and i%5]
```

Comprehension Internals

Comprehensions have their own local scope – just like a function

We should think of a list comprehension as being wrapped in a function that is created by Python that will return the new list when executed

```
sq = [i**2 for i in range(10)]
```

RHS

When the RHS is compiled: Python creates a temporary function that will be used to evaluate the comprehension

```
def temp():
    new_list = []
    for i in range(10):
        new_list.append(i**2)
    return new_list
```

When the line is executed:

Executes `temp()`

Stores the returned object (the list) in memory

Points `sq` to that object

We'll disassemble some Python code in the coding video to actually see this

Comprehension Scopes

So comprehensions are basically functions

They have their own local scope:

```
[item ** 2 for item in range(100)]
```

local symbol

But they can access global variables:

```
# module1.py

num = 100

sq = [item**2 for item in range(num)]
```

global symbol

local symbol

As well as nonlocal variables:

```
def my_func(num):
    sq = [item**2 for item in range(num)]
```

nonlocal symbol

Closures!!

Nested Comprehensions

Comprehensions can be nested within each other

And since they are functions, a nested comprehension can access (nonlocal) variables from the enclosing comprehension!

```
[ [i * j for j in range(5)] for i in range(5)]
```

Nested Loops in Comprehensions

We can have nested loops (as many levels as we want) in comprehensions.

This is not the same as nested comprehensions

```
l = []
for i in range(5):
    for j in range(5):
        for k in range(5):
            l.append((i, j, k))
```

```
l = [(i, j, k) for i in range(5) for j in range(5) for k in range(5)]
```

Note that the order in which the for loops are specified in the comprehension correspond to the order of the nested loops

Nested Loops in Comprehensions

Nested loops in comprehensions can also contain `if` statements

Again the order of the `for` and `if` statements does matter, just like a normal set of `for` loops and `if` statements

```
l = []
for i in range(5):
    for j in range(5):
        if i==j:
            l.append((i, j))
```

```
l = []
for i in range(5):
    if i==j:
        for j in range(5):
            l.append((i, j))
```

won't work!

`j` is created here

`j` is referenced after it has been created

```
l = [(i, j) for i in range(5) for j in range(5) if i == j]
```

```
l = [(i, j) for i in range(5) if i == j for j in range(5)]
```

won't work!

Nested Loops in Comprehensions

```
l = []
for i in range(1, 6):
    if i%2 == 0:
        for j in range(1, 6):
            if j%3 == 0:
                l.append((i,j))
```

```
[(i, j)
  for i in range(1, 6) if i%2==0
  for j in range(1, 6) if j%3==0]
```

```
l = []
for i in range(1, 6):
    for j in range(1, 6):
        if i%2==0:
            if j%3 == 0:
                l.append((i,j))
```

```
[(i, j)
  for i in range(1, 6)
  for j in range(1, 6)
  if i%2==0
  if j%3==0]
```

```
l = []
for i in range(1, 6):
    for j in range(1, 6):
        if i%2==0 and j%3==0:
            l.append((i,j))
```

```
[(i, j)
  for i in range(1, 6)
  for j in range(1, 6)
  if i%2==0 and j%3==0]
```

Code Exercises