List comprehensions

List comprehensions are one of the most powerful tools in Python, so people tend to think of them as advanced. They're not. Indeed, I've taken the liberty of littering previous examples with comprehensions and assuming you'd understand them. While it's true that advanced programmers use comprehensions a lot, it's not because they're advanced, it's because they're trivial, and handle some of the most common operations in programming.
Let's have a look at one of those common operations, namely, converting a list of items into a list of related items. Specifically, let's assume we just read a list of strings from a file, and now we want to convert it to a list of integers; we know every item in the list is an integer, and we want to do some activity (say, calculate an average) on those numbers. Here's one simple way to approach it:

```
input_strings = ['1', '5', '28', '131', '3']
output_integers = []
for num in input_strings:
      output_integers.append(int(num))
```

This works fine, it's only three lines of code. If you aren't used to comprehensions, you may not even think it looks ugly! Now, look at the same code using a list comprehension:

```
input_strings = ['1', '5', '28', '131', '3']
output_integers = [int(num) for num in input_strings]
```

We're down to one line and we've dropped an append method call. Overall, it's pretty easy to tell what's going on, even if you're not used to comprehension syntax.
The square brackets show we're creating a list. Inside this list is a for loop that loops over each item in the input sequence. The only thing that may be confusing is what's happening between the list's opening brace and the start of the for loop. Whatever happens here is applied to each of the items in the input list. The item in question is referenced by the `num` variable from the loop. So it's converting each such item to an `int`.

That's all there is to a basic list comprehension. They are not so advanced, after all! Comprehensions are highly optimized; list comprehensions are far faster than for loops when we are looping over a huge number of items. If readability alone isn't a convincing reason to use them as much as possible, then speed should be.
Converting one list of items into a related list isn't the only thing we can do with a list comprehension. We can also choose to exclude certain values by adding an `if` statement inside the comprehension. Have a look:

```
output_ints = [int(n) for n in input_strings if len(n) < 3]
```

I shortened the name of the variable from `num` to n and the result variable to `output_ints` so it would still fit on one line. Other than that, all that's different between this example and the previous one is the `if len(n) < 3` part. This extra code excludes any strings with more than two characters. The `if` statement is applied before the int function, so it's testing the length of a string. Since our input strings are all integers at heart, it refers to any number over ninety-nine. Now that is all there is to list comprehensions! We use them to map input values to output values, applying a
filter along the way to exclude any values that don't meet a specific condition.

Any iterable can be the input to a list comprehension; anything we can wrap in a `for` loop can also be placed inside a comprehension. For example, text files are iterable; each call to `__next__` on the file's iterator will return one line of the file. The contact file example we used earlier (to try out the `zip` function) can use a list comprehension instead:

```
import sys

filename = sys.argv[1]

with open(filename) as file:
      header = file.readline().strip().split('\t')
      contacts = [
            dict(
                  zip(header, line.strip().split('\t'))
            ) for line in file
      ]
```

This time, I've added some `whitespace` to make it more readable (list comprehensions don't have to fit on one line). This example is doing the same thing as the previous version: creating a list of dictionaries from the zipped header and split lines for each line in the file.