LECTURE 8

LIST METHODS AND COMPREHENSIONS

MCS 260 Fall 2021

David Dumas

REMINDERS

- Worksheet 3 solutions posted
- Homework 3 posted (due Tues 10am)
- Project 1 due Fri Sep 17 at 6pm CDT

PROJECT 1 DISCUSSION

321, 174, 258, 385, 448, 480, 496, 612, 414, ...

ITERABLES

Recall a thing that can appear in a for loop in Python is called an **iterable**. So iterables include:

- Sequences (strings, lists, tuples*)
- **`range(...)`**, **`enumerate(...)`**
- Other built-in types we'll discuss soon (dict, set)

LIST METHODS

Lists in Python have many useful features we haven't talked about.

Any list, say **L**, comes with its own set of functions (called **methods**) that operate directly on the list.

```
L.append(x)     # Add x to the end of the list
L.insert(i,x)   # Insert x at position i
L.remove(x)     # Remove first instance of x in L
L.pop()         # Remove and return the last item of L
L.index(x)      # Find x in L, return its index
```

All except **`index()`** change the list.

Example: Suppose `L` is a list of strings representing integers, and we need to convert it to a list `M` of ints.

A for loop can be used to do this:

```
L = ["42", "16", "15", "8", "4"]
M = []
for s in L:
    M.append( int(s) )
# now M == [42, 16, 15, 8, 4]
```

This pattern is very common: Iterate over a list doing something to each element, producing a new list.

This pattern is so common that Python has a more compact way of writing it. The code:

```
M = []
for s in L:
    M.append( int(s) )
```

Can instead be written:

```
M = [ int(s) for s in L ]
```

The expression `[ ... for ... in ... ]` is called a **list comprehension**. It is a *compact way of writing a common type of for loop.*

COMPREHENSION EXAMPLES

The basic comprehension syntax is:

```
[ expression for varname in iterable ]
```

For example:

```
[ x**2 for x in range(5) ]
# Gives [0, 1, 4, 9, 16]

[ s[1:] for s in ["cat", "spot", "blot"] ]
# Gives ["at", "pot", "lot"]

[ float(s[:-1]) for s in ["6C", "12.5C", "25C"] ]
# Gives [6.0, 12.5, 25.0]
```

The variable name in a comprehension can be anything, it just needs to be used consistently.

These are all equivalent:

```
[ x**2 for x in range(5) ]
[ t**2 for t in range(5) ]
[ apple**2 for apple in range(5) ]
```

The name in a comprehension is not assigned to anything outside the comprehension:

```
>>> [ x**2 for x in range(5) ]
[0, 1, 4, 9, 16]
>>> x
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
```

FILTERING

There is another common type of for loop, where elements are not just transformed but also *filtered*.

```
words = [ "alpha", "bridge", "assemble", "question" ]
a_words = []
for s in words:
    if s[0] == "a":
        a_words.append(s)
# Now a_words is [ "alpha", "assemble" ]
```

This too can be done in a comprehension:

```
a_words = [ s for s in words if s[0]=="a" ]
```

The general form is

```
[ expression for name in iterable if condition ]
```

FILTERING EXAMPLES

Consider:

```
[ x+x**2 for x in range(5) if x!=2 ]
```

In words: Start with the integers $0 \ldots 4$, consider only the ones that are not equal to 2, and for each of those, add the number to its square. Make a list of the results.

```
# range(5) gives       [0, 1, 2, 3, 4]
# !=2 gives            [0, 1, 3, 4]
# add to square gives [0+0, 1+1, 3+9, 4+16]
# Final result:
[0, 2, 12, 20]
```

A list of lists of names and salutations:

```
namepairs = [ ["Mr.","Nabil Weber"],
              ["Ms.","Janet Leon"],
              ["Ms.","Mariana Wang"],
              ["Dr.","Lisa Young"] ]
```

Tip: as we do here, lists can be split between lines. Indenting is not required.

What if we want a greeting (as *salutation name*) of the people with salutation "Ms."?

```
[ sal+" "+name for sal,name in namepairs if sal=="Ms." ]
# Gives ["Ms. Janet Leon","Ms. Mariana Wang"]
```

Equivalent for loop:

```
mss = []
for sal,name in namepairs:
    if sal=="Ms.":
        mss.append(sal+" "+name)
```

Convert every digit from the input string to an int, and make a list of these:

```
[ int(c) for c in input() if c in "0123456789" ]
```

If the keyboard input is **`I like 0 more than 157`**, then the above will evaluate to

```
[ 0, 1, 5, 7 ]
```

REFERENCES

- In *Downey*:
    - Section 19.2 discusses list comprehensions

REVISION HISTORY

- 2021-09-09 Initial publication