A: Lists in Python

Rhodri James Revision 1.8, May 14, 2001

Credits

Spam, spam, spam

Lists are collections of things. You make lists of what you want when you go shopping, lists of what's been cooked in the canteen, or lists of where your robots are on the screen. So how do you make a list in Python?

```
>>> menu = ['spam', 'eggs', 'chips', 'beans']
>>> numbers = [1, 2, 3]
>>> both = [2, "wombats", "are", 4, "going elsewhere"]
>>> empty = []
```

Yes, you just stick square brackets around whatever you want to turn into a list, even when that something is nothing at all! That's not as silly as it sounds, by the way; everything has to start from somewhere, so why not start a list with nothing on it, and add items as you think of it.

If you are used to other computer languages, you may already be thinking that Python's lists sound a lot like arrays. Congratulations, they are arrays, mostly. Keep reading, though; there are some little twists in thinking coming!

Slice of spam, sir?

Now we have this list, how do we find out what's on it? Python conveniently numbers everything on our list for us, and lets us find out what each thing is by writing its number in square brackets after the list's name. Try typing the following:

```
>>> menu[1]                                First thing on the list?
'eggs'                                     Not 'spam'. Oops!
```

In fact Python numbers items on the list from 0, not from 1, so our spam is actually `menu[0]`. You can also use negative numbers to read from the end of the list, rather than the beginning. The last item on the list is numbered `-1`, so this works a little more like you would expect!

```
>>> menu[-2]
'chips'
>>> numbers[-1]
3
```

Something else you can do with lists is to “slice” them, making a new list from part of the old one. Just tell Python where you want it to start taking things from and where you want it to stop, with a colon `:` in between them.

```
>>> menu[1:3]
[’eggs’, ’chips’]                           Take items 1 & 2, stop at 3.
>>> menu[:2]
[’spam’, ’eggs’]                            Start at the beginning if you don’t say.
>>> menu[2:]
[’chips’, ’beans’]                          End at the end if you don’t say.
>>> menu[:]
[’spam’, ’eggs’, ’chips’, ’beans’]          Silly!
```

We can do a few other things with lists that you haven’t seen yet.

```
>>> menu + numbers                          ‘‘Concatenation’’ works just like strings.
[’spam’, ’eggs’, ’chips’, ’beans’, 1, 2, 3]
>>> [’x’] * 3                               ‘‘Repetition’’, again just like strings.
[ ’x’, ’x’, ’x’]
>>> numbers.append(100)                     Add something to the end of the list.
>>> print numbers                           It didn’t print out for itself!
[1, 2, 3, 100]
>>> len(numbers)                            How long is my list?
4
>>> numbers.reverse()                       Turn the list upside-down.
>>> print numbers                           Again, it didn’t print for itself.
[100, 3, 2, 1]
>>> numbers.sort()                          Sort the list into order.
>>> print numbers
[1, 2, 3, 100]
>>> numbers.index(3)                        Where in the list is 3?
2                                           Position 2 (counting from 0)
>>> numbers[2]                              Check that position 2 really has the right thing
3                                           Yes!
>>> numbers.index(1234)                     What if it’s not there?
Traceback (innermost last):
  File "<stdin>", line 1, in ?
ValueError: list.index(x): x not in list
                                            Eeeek. So don’t do that, then.
>>> del numbers[2]                          Removing an item from the list
>>> numbers
[1, 2, 100]                                 It’s gone.
>>> range(3)                                Making useful lists.
[0, 1, 2]
>>> range(1,4)                              A bit like slicing in reverse.
[1, 2, 3]
```

(Some of those probably look rather weird – for instance, `numbers.sort()`. Don’t worry about it!)

The `range` function is very useful for `for` loops – see Sheet L (*Loops*).

There are lots more things that you can do with lists; you’ll discover some of them on the holiday!

Lists of lists of lists of . . .

Sometimes one list isn’t enough. Suppose you wanted to keep a list of what you had eaten for breakfast; it’s easy enough to write

```
>>> breakfast = [’coffee’, ’corn flakes’, ’toast’, ’marmalade’]
```

and carry on. But suppose you wanted to list what you had eaten for breakfast every day, and you don’t always eat the same things. What do you do then.

Fortunately, Python is very helpful about this. Remember that we said earlier that lists were just collections of things. Well, lists are things too, so making lists of lists is just like making lists of anything else!

```
>>> breakfast = [
...   'Monday', ['coffee', 'rice crispies'], Remember the spaces!
...   'Tuesday', ['orange juice', 'toast', 'marmalade'],
...   'Wednesday', ['coffee'] ]
```

(Look, I was in a hurry on Wednesday!)

Some people call lists of lists "2-dimensional arrays" or "tables", because you can write them out in rows and columns (two dimensions!) like a table. Unlike real tables (and many other computer languages), in Python you don't have to have every row the same length.

To get at the list items, just add "indices" (item numbers) in square brackets to the end of the list name. For a list in a list you need two sets of square brackets. For a list in a list in a list you need three, and so on.

```
>>> breakfast[3]
['orange juice', 'toast', 'marmalade']
>>> breakfast[3][0]
'orange juice'
```

A subtle trap

So far, we've talked about lists as collections of things. Actually, it's more accurate to say that Python's lists are really collections of "references" or "pointers" to things. For most purposes, this makes no difference at all, and you can happily ignore us being pedants here. However, when you make lists from variables or other lists, then it can become important.

For example, when you replicate a list, you get copies of the same *identical* list. They are all actually the same list, so when you change one, you actually change them all. So the following doesn't work quite the way you might have expected:

```
>>> wombat = [[1, 2, 3]] * 3
>>> print wombat
[[1, 2, 3], [1, 2, 3], [1, 2, 3]]
>>> wombat[0][2] = 4
>>> print wombat
[[1, 2, 4], [1, 2, 4], [1, 2, 4]]
```

There are ways around this; the easiest is do something that makes a new list from the oldest, such writing the list out separately each time or (if you are copying from a variable) taking a slice of the whole list. Maybe typing `numbers[:]` isn't so silly after all!

(If you didn't understand any of that last bit, don't worry.)