ECMM455 Python Worksheet 10: Lists

Prof Hywel Williams

September 12, 2019

1 Aims

- Learn about lists

2 Lists

A list is a container for multiple pieces of data – these can be of mixed types. Lists can be created by declaration (writing them out in square brackets, separating items by commas):

mylist = ['fish', 'cat', 'dog', 23]

Lists are "iterable" objects in Python, meaning that their elements can be accessed using an index. For example:

print mylist[2]

There are a number of useful methods and functions associated with lists. For a list called mylist:

- *mylist.append(x)* - adds *x* to end of list
- *mylist.sort()* - sorts list alphanumerically
- *mylist.reverse()* - reverses list order
- *del mylist[3]* - deletes item at index 3

There are also some useful operators:

- *mylist1 + mylist2* - concatenates *mylist1* and *mylist2*
- *mylist*3* - concatenates three copies of *mylist*

To check whether an item is present in a list, the *in* operator allows a membership check:

- *x in mylist* - logical statement, evaluates *True* if *x* is in *mylist*

Other functions on lists:

- *max(mylist)* - returns largest element in *mylist*
- *min(mylist)* - returns smallest element in *mylist*
- *len(mylist)* - returns number of elements in *mylist*

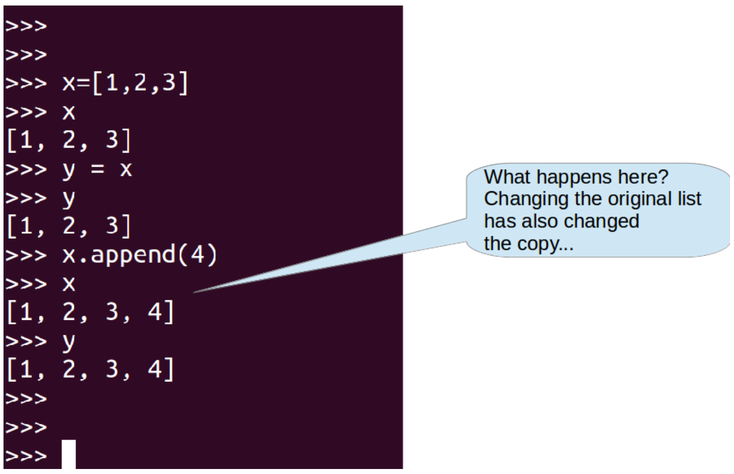
3 Exercises

In interactive mode:

1. Create a list of 4 types of vehicle, then display it in alphabetical order.
2. Create a list of 6 numbers, then display it in descending order.
3. Declare a list of three cheeses. Append two more cheeses. Delete the second cheese. What indexes do the three original cheeses have now? Is "stilton" in your cheese list?
4. Declare an empty list. Append some numerical values to it. What are the maximum and minimum values? How long is your list?

4 How to copy a list

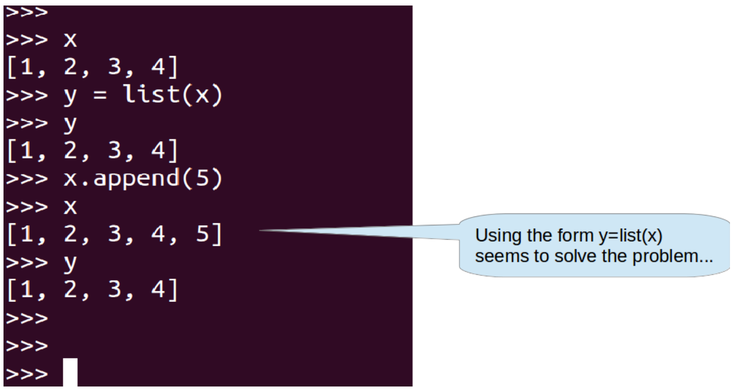
List copying is not as straightforward as it might appear. Look at the code below where a list x is "copied" to a new list y by simple assignment:



```
>>>
>>>
>>> x=[1,2,3]
>>> x
[1, 2, 3]
>>> y = x
>>> y
[1, 2, 3]
>>> x.append(4)
>>> x
[1, 2, 3, 4]
>>> y
[1, 2, 3, 4]
>>>
>>>
>>>
```

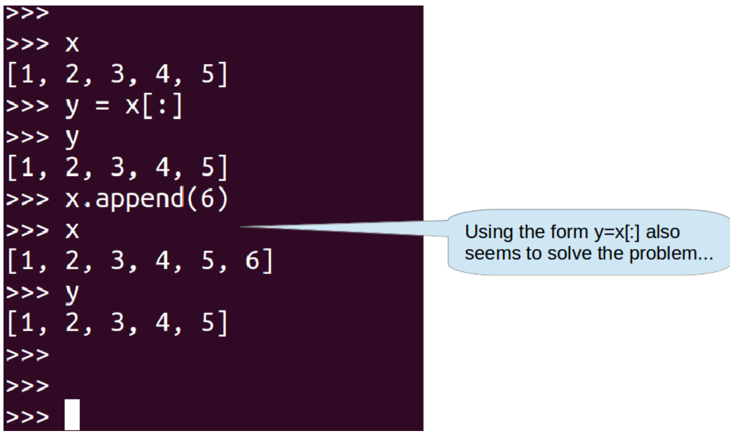
What happens here? Changing the original list has also changed the copy...

This did not have the effect we might have expected, i.e. creation of two independent lists. Now look at the code below, which copies list *x* into list *y* using the *list()* function:



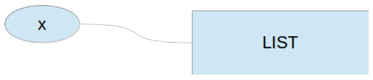
```
>>>
>>> x
[1, 2, 3, 4]
>>> y = list(x)
>>> y
[1, 2, 3, 4]
>>> x.append(5)
>>> x
[1, 2, 3, 4, 5]
>>> y
[1, 2, 3, 4]
>>>
>>>
>>>
```

This version seems to have worked. Also, look what happens below when we copy using a slice operation:

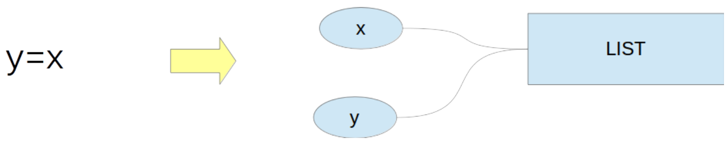


```
>>>
>>> x
[1, 2, 3, 4, 5]
>>> y = x[:]
>>> y
[1, 2, 3, 4, 5]
>>> x.append(6)
>>> x
[1, 2, 3, 4, 5, 6]
>>> y
[1, 2, 3, 4, 5]
>>>
>>>
>>>
```

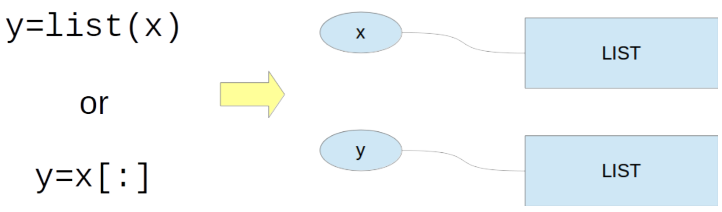
The reason for this Python weirdness is to do with how variables and memory are linked “under the bonnet”. Variables in Python are really just tags for data objects:



Copying a list variable using simple assignment (e.g. y = x) just associates another tag with the same object:



So altering either x or y affects both, since they point to the same object. However, using the list() or slice methods has the desired effect. First it creates a second list object. Second it populates it with data copied from the first list.



5 Exercise

Play around with the three methods of copying lists until you understand how they work.