Storing data - Lists

Lists hold a list of values. For example, here `subjects` is a list of strings:

```
subjects = ["maths", "physics", "engineering", "chemistry", "biology"]
print(subjects)
```

The values can then be accessed by their numeric ordering. In the code below, what value do you expect printing `months[3]` to return? Run it and see.

```
months = ["January", "February", "March", "April", "May", "June", "July", "Aug
print(months[3])
```

It is important to remember that a lot of counting in computing starts at `0`. Failure to remember this can lead to a program doing something one place different from where you expected, a so-called 'off by one error'.

You can also access the values starting from the end by using negative numbers. For example, the following will print the last and third from last elements of the list.

```
months = ["January", "February", "March", "April", "May", "June", "July", "Aug
print(months[-1])
print(months[-3])
```

Random choice

First import the `random` module.

```
from random import choice
```

You can use `choice` to select a random item from a list. For example, here we choose a random item from the `months` list we made above.

```
choice(months)
```

You can also shuffle the elements in a list. Use `shuffle` to rearrange the items in the current list variable, for example:

```
numbers = [1,2,3,4,5,6]
print(numbers)
shuffle(numbers)
print(numbers)
```

Slicing

You can also access a range of values by slicing from the list. The following example uses code comments to indicate what is happening. Note the notation `a[start:end]` slices the values from the list from `start` to `end-1` (i.e. the value in the `end` position is not returned). Leaving either `start` or `end` blank tells Python to use the start or end of the list.

```
a = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]
print(a[0:2]) # first and second
print(a[1:4]) # second to fourth
print(a[-3:]) # last three
print(a[2:]) # everything from the third onwards
print(a[:7]) # everything from the start until the seventh
```

You can slice through the list in different step-lengths by passing a third parameter, `step`. In the examples above, the slice took every one step, i.e. they returned every value between `start` and `end`. In the examples below, different step lengths are used. Note the notation `a[start:end:step]` slices the values from the list from `start` to `end-1` by `step`. Run the code below and modify it to explore the behaviour of this feature.

```
a=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]
print(a[:10:2]) # every second item from the start to the tenth
print(a[4:14:3]) # every third item from the fifth to the fourteenth
```

You can also slice backwards through a list by using a negative `step`.

```
a=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]
print(a[::-1]) # every item backwards
```

You can also change the value of an item in a list. See what happens when you run this code - it will print out `ingredients` before and after making a change to one of the values.

```
ingredients = ["courgette","mushroom","aubergine","tomato","herbs","chilli"]
print(ingredients)
ingredients[2] = "peppers"
print(ingredients)
```

You can add values to the end of a list using `append()`.

```
mos_eisley = ["Luke","Ben","C3PO","R2D2"]
print(mos_eisley)
mos_eisley.append("Han")
mos_eisley.append("Chewie")
print(mos_eisley)
```

You can remove items from a list in a few ways.

Using `del` simply removes an item or several items from a list by their index or slice. Run the following example to see this in action.

```
engines = ["Thomas","Edward","Henry","Gordon","James","Percy","Toby","Duck","D
print(engines)
```

```
del engines[1]
print(engines)
del engines[2:4]
print(engines)
del engines[-3:]
print(engines)
```

Using `remove` searches through the list and removes the first matching value. Run the following example to see this in action.

```
staff=["Peter","Alex","Angharad","Alex","Ros","Alex"]
print(staff)
staff.remove("Alex")
print(staff)
staff.remove("Alex")
print(staff)
```

Using `pop` removes a specific item *and* returns it so you can store it somewhere else or do something else with it. Run the following example to see this in action.

```
a = [1,2,3,4,5,6,7,8,9,10]
print(a)
removed1 = a.pop(1)
removed2 = a.pop(4)
removed3 = a.pop(-3)
print(f"The sum of the removed values is {removed1+removed2+removed3}")
print(a)
```

Sorting

You can generate a copy of a list in alphanumeric order by value using `sorted`. The following example make a copy of a list in alphabetical order.

```
original_order = ["England", "Scotland", "Wales", "Northern Ireland"]
alphabetical_order = sorted(original_order)
print(original_order)
print(alphabetical_order)
```

You can sort a list by its values (rather than creating a copy) using `sort`, as in the following example. The disadvantage of this is that it modifies the existing list, losing whatever order it was in.

```
original_order = ["England", "Scotland", "Wales", "Northern Ireland"]
original_order.sort()
print(original_order) # original ordering is lost and cannot be recovered
```

You can sort in reverse order by passing a parameter `reverse=True`.

```
names = ("Monica", "Chandler", "Ross", "Rachel", "Joey", "Phoebe")
reverse_alphabetical = sorted(names, reverse=True)
print(reverse_alphabetical)
```

Sorting is a little complicated so you need to be careful. Try running the following code. What do you notice?

```
letters = ["A","a","C","B","c","b"]
print(sorted(letters))
```

When using sorted, you can pass a second parameter to the function called `key` which specifies a function to be called on each element before making comparisons. Here we use `key=str.lower` to tell it to convert each element to lower case before comparing (though leaving the original in the original case at the end).

```
letters = ["A","a","C","B","c","b"]
print(sorted(letters, key=str.lower))
```

Here is another strange example. We make a list of strings representing numbers. What do you notice about the way they are sorted?

```
numbers = ["0", "1", "3.14157", "2.71828", "1.41421", "10", "121", "1024"]
print(sorted(numbers))
```

Numbers as strings are sorted alphanumerically, i.e. strings starting with 0 first, then 1, and so on. To get `sorted` to sort the numbers according to their values, use the `key` parameter to convert the values to decimal numbers (called floats).

```
numbers = ["0", "1", "3.14157", "2.71828", "1.41421", "10", "121", "1024"]
print(sorted(numbers, key=float))
```

Related: A really useful way of storing dates is to use the format `YYYY-MM-DD`, i.e. four-digit year, then two-digit month, then two-digit day. This format will sort alphanumerically into chronological order. This can be really useful when naming different versions of a document, for example.

```
dates = ["1985-10-25", "1955-11-05", "1955-11-12", "1985-10-26", "2015-10-21",
print(sorted(dates))
```

Size

You can find the number of entries in a list using the `len` function.

```
flavours = ["up","down","strange","charm","bottom","top"]
print(len(flavours))
```

Dimensions

Note that the items within a list can be themselves lists, meaning that multi-dimensional lists can be created. For example, here we create a list and to it we add six other lists, each of which contains another list.

```
avengers = []
avengers.append(["Tony Stark", "Iron Man", ["Genius", "Billionare", "Armored s
avengers.append(["Steve Rogers", "Captain America", ["Super strength", "Accele
avengers.append(["Bruce Banner","Hulk",["Genius", "Invulnerability", "Super st
avengers.append(["Thor", "Thor", ["Super strength", "Mjölnir", "Electric/weath
avengers.append(["Natasha Romanoff", "Black Widow", ["Super spy"]])
avengers.append(["Clint Barton", "Hawkeye", ["Master archer"]])
print("all Avengers, a list:")
print(avengers)
print("one of the Avengers, also a list:")
print(avengers[4])
print("the powers of one of the Avengers, also a list:")
print(avengers[3][2])
```