Recall that the main built-in types in python are numerics, sequences, mappings, classes, instances and exceptions.

The **sequence** type is characterised by the concept of order. Broadly speaking a sequence has

- a first element
- a last element
- a way of accessing an arbitrary element
- a way of systematically accessing all elements *in order*.

Lists are one variety of mutable sequence type. Lists are a compound data type, used to group *related* data together. That is, they are a collection of data items. We enclose the items within square brackets and separate them by commas. Some examples are

```
>>> p = [2, 3, 5, 7, 11, 13]
>>> f = [1, 1, 2, 3, 5, 8, 13]
```

An empty list is denoted by `[]` and a new list is typically initialised to that.

Lists can be heterogenous; that is they can contain items of different data types. For example,

```
>>> mix = ["A", 12, 13.7, -2]
```

`mix` is a list containing 4 elements: the first is a string; the second and fourth are integers and the third is float.

Accessing an element

The elements of a list can be accessed by its *index* which is the position of the element in the list. The convention is to use the index number 0 for the first element. Thus a list of n elements is said to have elements in index 0 to index n-1.

```
>>> cubes = [1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
>>> print(cubes[2])
27
>>> print(cubes[9])
```



```
1000
>>> i = 3
>>> print(cubes[i])
64
```

Lists are mutable. So we can modify an element of a list by assigning a new value as in:

```
>>> alpha = ["A", "X", "X", "T"]
>>> alpha[2] = "Q"
>>> alpha
["A", "X", "Q", "T"]
```

As you can see in the above example, a list can have multiple elements with the same value.

But we cannot add an element by trying to access a non-existing index. See .

Negative indices

In languages like C, Java you need to know/compute the number of elements in an array, in order to access the last element. But python gives a convenient shorthand: -1 is the index of the last element, -2 is the index of the last but one and so on.

```
>>> cubes[-1] == 1000
True
>>> print(cubes[-2])
729
```

Invalid indices

It is an error to try to access an elemnent that is not in the list; that is trying to access cubes[10] or cubes[-11] etc., will give *IndexError*.

Slicing and striding

We can *slice*, that is select a part of a list by using the slicing operation. For example,

```
>>> cubes[1:5]
[8, 27, 64, 125]
```


The syntax of a slice is `aList[start:end]`. Note that this operation

- returns a new list
- the first element of that new list is `aList[start]`
- the last element is `aList[end - 1]`, that is `aList` has (`end - start`) elements
- aList is unchanged

```
>>> p = [2, 3, 5, 7, 11, 13, 17, 19]
>>> p[3:7]
[7, 11, 13, 17]
>>> tp = p[4:]
>>> print(tp)
[11, 13, 17, 19]
>>> print(p[:-2])
[2, 3, 5, 7, 11, 13]
```

As we can see from the above if either `start` or `end` is not specified it defaults to be the logical end point.

Striding

While creating a slice we can pick every n^{th} element instead of every element. That is, achieved by specifying `aList[start:end:n]`. This will return a new list whose starting elements are `aList[start]`, `aList[start + n]`, `aList[start + 2n]`, `aList[start + 3n]` ...till the `aList[end]`.

List methods

All lists support the various operations by *methods*. The general syntax is `list.method([param])`. Some methods mutate the list while others do not. It is important to be aware whether a method is mutating or not.

Adding elements

We noted that we cannot use a non-existing index to add an element to a list. There are many ways of adding elements to a list:

- Concatenate a list to another: `alpha = alpha + beta`, where `alpha` and `beta` are lists, results in the elements of `beta` being added to the end of existing elements in `alpha`
- Extend a list: `alpha.extend(beta)` has the same effect as above.
- To add only one element, use `alpha.append(newElement)`
- To add an element in the middle, say index n, use `alpha.insert(n, newElement)`

Removing Elements

Again there are different ways:

- Specify the value of the element to be removed: `alpha.remove(x)` where `x` is an elemnt in `alpha`. If there is more than one element with the value `x` the first is removed.
- We can remove a slice by assigning it to the empty list. Thus `alpha[p:q] = []` will remove the elements in indices `p, p+1 ... q-1`. That means the number of elements in the list is now less by `q - p`. Of course a single element at index p can be removed by, `alpha[p:p+1] = []`.
- The code `k = alpha.pop()` does two things: One it assigns the value of the first element of `alpha` to `k`; two it removes the first element from the list.
- `alpha.clear()` removes all the elements in alpha, making it the empty list.

Utility methods

- `alpha.sort()` sorts the elements of the list *in place*. That is it mutates the list.
- `alpha.reverse()` reverses the order of elements in the list. Again mutates the list.
- `alpha.count(x)` returns the number of times the value x occurs in the list.

