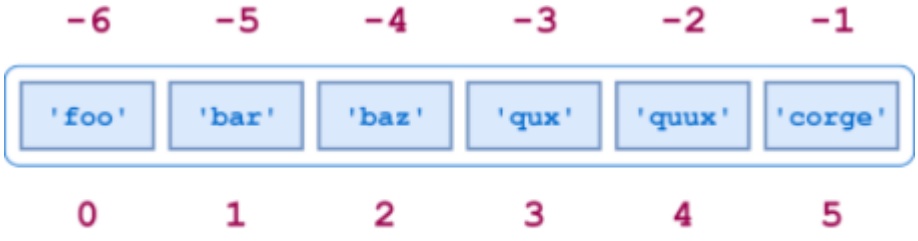
Here is Python code to access some elements of a list:

```
[27]: a[0]
[27]: 'foo'
[28]: a[2]
[28]: 'baz'
[29]: a[5]
[29]: 'corge'
```

Virtually everything about string indexing works similarly for lists. For example, a negative list index counts from the end of the list:



Negative List Indexing


```
[30]: a[-2]
[30]: 'quux'
[31]: a[-5]
[31]: 'bar'
```

2.1.4 List Slicing

Slicing also works. If a is a list, the expression a[m:n] returns the portion of a from index m to, but not including, index n:

Example: a[start : end]

```
[131]: #slicing
       a=['foo','bar','baz','qux','quux','corge']
       a[2:5]
[131]: ['baz', 'qux', 'quux']
```

Other features of string slicing work analogously for list slicing as well:

If s is a string, s[:] returns a reference to the same object:

```
In [144]: s='food bar'
          s[:]
Out[144]: 'food bar'

In [145]: s[:]  is s
Out[145]: True
```

Conversely, if a is a list, a[:] returns a new object that is a copy of a:

```
In [148]: a=['foo','bar','baz','qux','quux','corge']
          a[:]
Out[148]: ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']

In [149]: a[:]  is a
Out[149]: False
```

Several Python operators and built-in functions can also be used with lists in ways that are analogous to strings:

The in and not in operators:

```
In [151]: 'qux' in a
Out[151]: True

In [152]: 'thud ' not in a
Out[152]: True
```


The concatenation (+) and replication (*) operators:

```
In [153]: a+['graulr', 'graply']
Out[153]: ['foo', 'bar', 'baz', 'qux', 'quux', 'corge', 'graulr', 'graply']

In [155]: print(a*2)
          ['foo', 'bar', 'baz', 'qux', 'quux', 'corge', 'foo', 'bar', 'baz', 'qux', 'quux', 'corge']
```

The `len()`, `min()`, and `max()` functions:

```
[156]: len(a)

[156]: 6

[158]: max(a)

[158]: 'qux'

[159]: min(a)

[159]: 'bar'
```

x[0], x[2], and x[4] are strings, each one character long:

```
[34]: print(x[0], x[2], x[4])
      a g j
```

But x[1] and x[3] are sub lists:

```
[166]: x[1]
[166]: ['bb', ['ccc', 'ddd'], 'ee', 'ff']
[167]: x[3]
```

To access the items in a sub list, simply append an additional index:

```
[168]: x[1][0]
[168]: 'bb'
[169]: x[1][1]
[169]: ['ccc', 'ddd']
[170]: x[1][2]
[170]: 'ee'
```


x[1][1] is yet another sub list, so adding one more index accesses its elements:

```
[170]: x[1][2]
[170]: 'ee'
[171]: x[1][1][0],x[1][1][1]
[171]: ('ccc', 'ddd')
```

There is no limit, short of the extent of your computer's memory, to the depth or complexity with which lists can be nested in this way.

All the usual syntax regarding indices and slicing applies to sub lists as well: