Manipulating lists

Lists

A string is a sequence of characters. A *list* is a sequence of anything. As a string is an *object*, so is a list.

A list value is specified by a pair of square brackets.

Example.

```
a = [1, 2, 3, 4]
```


Lists

Each element of a list is indexed by an integer, starting from 0.

To access a list `a`'s `i`th element, use `a[i]`.

Example.

```
>>> a = [1, 2, 3, 4]
>>> print a[2]
3
```

Lists

Unlike strings, lists are *mutable*.

Example.

```
>>> b = ['trinity', 'College']
>>> b[0] = 'Trinity'
>>> print b
['Trinity', 'College']
```


Appending elements

To *append* an element to a list,

```
>>> a = []
>>> a.append(1)
>>> a.append(2)
>>> print a
[1, 2]
```


Appending lists

To *append* a list to another list,

```
>>> a = [1, 2, 3]
>>> b = [4, 5]
>>> a.append(b)
>>> print a
[1, 2, 3, 4, 5]
[1, 2, 3, [4, 5]]
```

Extending lists

To *extend* a list by another list,

```
>>> a = [1, 2, 3]
>>> b = [4, 5]
>>> a.extend(b)
>>> print a
[1, 2, 3, 4, 5]
```


Concatenating lists

To *glue* (or *concatenate*) a list to another list,

```
>>> a = [1, 2, 3]
>>> b = [4, 5]
>>> c = a + b
>>> print c
[1, 2, 3, 4, 5]
```


Traversing lists

To traverse a list, use a *for* loop.

Example.

```
>>> cpsc115 = ['alyssa', 'ben', 'carl']
>>> for name in cpsc115:
      print name
alyssa
ben
carl
```

Traversing lists

To traverse a list, use a *for* loop.

Example.

```
>>> cpsc115 = [’alyssa’, ’ben’, ’carl’]
>>> for name in cpsc115:
      name = name + ’@trincoll.edu’
>>> print cpsc115
[’alyssa’, ’ben’, ’carl’]
```


Traversing lists

To modify a list, use indices.

Example.

```
>>> cpsc115 = [’alyssa’, ’ben’, ’carl’]
>>> for i in range(len(cpsc115)):
      cpsc115[i] = cpsc115[i] + ’@trincoll.edu’
>>> print cpsc115
['alyssa@trincoll.edu', 'ben@trincoll.edu',
 'carl@trincoll.edu']
```


Multiplying lists

To *multiply* a list,

```
>>> a = [0, 1]
>>> b = a * 3
>>> print b
[0, 1, 0, 1, 0, 1]
```

Slicing lists

Lists can also be *sliced* by specifying the first and last indices: `a[i:j]` is the sublist of `a` starting from `i` through `j-1`.

```
>>> a = [0, 1, 2, 3, 4, 5, 6, 7]
>>> a[1:4]
[1, 2, 3]
>>> a[:4]
[0, 1, 2, 3]
>>> a[4:]
[4, 5, 6, 7]
```


Slicing lists

Lists can also be *sliced* by specifying the first and last indices: `a[i:j]` is the sublist of `a` starting from `i` through `j-1`.

```
>>> a = [0, 1, 2, 3, 4, 5, 6, 7]
>>> a[1:4] = [8, 8, 8]
>>> print a
[0, 8, 8, 8, 4, 5, 6, 7]
>>> a[1:4] = [8, 8, 8, 8, 8]
>>> print a
[0, 8, 8, 8, 8, 8, 4, 5, 6, 7]
```


Reducing

Functions often reduce a value from list elements.

```
>>> def add_all(list):
      sum = 0
      for element in list:
        sum = sum + element
      return sum

>>> add_all([1, 2, 3, 4])
10
```

Mapping

Functions often apply (or map) an operation to list elements.

```
>>> def square(list):
      result = []
      for element in list:
        result.append(element * element)
      return result

>>> square([1, 2, 3, 4])
[1, 4, 9, 16]
```


Filtering

Functions often filter out list elements.

```
>>> def even(list):
      result = []
      for element in list:
        if element % 2 == 0:
          result.append(element)
      return result

>>> even([1, 2, 3, 4])
[2, 4]
```