

Part 05 - Containers and Casting

Part 05 - Containers and Casting

Lists

**Definition: List**

A linked list that can hold a variable amount of objects.

Lists are mutable, which means that the List can be changed, as well as its children.

lists

```
print([0, 'alpha', 4.5, char('d')])
print List('abcdefghij')
l = List(range(5))
print l
l[2] = 5
print l
l[3] = 'banana'
print l
l.Add(100.1)
print l
l.Remove(1)
print l
for item in l:
    print item
```

Output

```
[0, alpha, 4.5, d]
[a, b, c, d, e, f, g, h, i, j]
[0, 1, 2, 3, 4]
[0, 1, 5, 3, 4]
[0, 1, 5, 'banana', 4]
[0, 1, 5, 'banana', 4, 100.1]
[0, 5, 'banana', 4, 100.1]
0
5
'banana'
4
100.1
```

As you can see, Lists are very flexible, which is very handy.

Lists can be defined two ways:

1. by using brackets []
2. by creating a new List wrapping an IEnumerable, or an array.

Slicing

Slicing is quite simple, and can be done to strings, Lists, and arrays.

It goes in the form `var[start:end]`. both start and end are optional, and must be integers, even negative integers.

To just get one child, use the form `var[position]`. It will return a char for a string, an object for a List, or the specified type for an array.

Slicing counts up from the number 0, so 0 would be the 1st value, 1 would be the 2nd, and so on.

slicing

```
list = List(range(10))
print list
print list[:5] // first 5
print list[2:5] // starting with 2nd, go up to but not including the 5
print list[5:] // everything past the 5th
print list[:-2] // everything up to the 2nd to last
print list[-4:-2] // starting with the 4th to last, go up to 2nd to last
print list[5] // the 6th
print list[-8] // the 8th from last
print '---'
str = 'abcdefghij'
print str
print str[:5] // first 5
print str[2:5] // starting with 3rd, go up to but not including the 6th
print str[5:] // everything past the 6th
print str[:-2] // everything before the 2nd to last
print str[-4:-2] // starting with the 4th to last, to before the 2nd to last
print str[5] // the 6th
print str[-8] // the 8th from last
```

Output

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 1, 2, 3, 4]
[2, 3, 4]
[5, 6, 7, 8, 9]
[0, 1, 2, 3, 4, 5, 6, 7]
[6, 7]
5
2
---
abcdefghij
abcde
cde
ghij
abcdefgh
gh
f
c
```

I hope you get the idea. Slicing is very powerful, as it allows you to express what you need in a minimal amount of space, while still being readable.

Arrays



Definition: Array

Arrays are simple objects that hold equally-sized data elements, generally of the same data type.

Arrays, unlike Lists, cannot change their size. They can still be sliced, just not added on to.

Arrays can be defined three ways:

1. by using parentheses (`()`)
 - If you have 0 members, it's declared: `(,)`
 - If you have 1 member, it's declared: `(member,)`
 - If you have 2 or more members, it's declared: `(one, two)`
2. by creating a new array wrapping an `IEnumerator`, or an `List`.
3. by creating a blank array with a specified size: `array(type, size)`

arrays

```
print((0, 'alpha', 4.5, char('d')))  
print array('abcdefghij')  
l = array(range(5))  
print l  
l[2] = 5  
print l  
l[3] = 'banana'
```

Output

```
(0, alpha, 4.5, d)  
(a, b, c, d, e, f, g, h, i, j)  
(0, 1, 2, 3, 4)  
(0, 1, 5, 3, 4)  
ERROR: Cannot convert 'System.String' to 'System.Int32'.
```

Arrays, unlike Lists, do not necessarily group objects. They can group any type, in the case of `array(range(5))`, it made an array of ints.

List to Array Conversion

If you create a List of ints and want to turn it into an array, you have to explicitly state that the List contains ints.

list to array conversion

```
list = []
for i in range(5):
    list.Add(i)
    print list
a = array(int, list)
for a_s in a:
    print a_s
a[2] += 5
print a
list[2] += 5
print list[2]
```

Output

```
[0]
[0, 1]
[0, 1, 2]
[0, 1, 2, 3]
[0, 1, 2, 3, 4]
(0, 1, 2, 3, 4)
(0, 1, 7, 3, 4)
ERROR: Operator '+' cannot be used with a left-hand side of type
'System.Object' and a right-hand side of type 'System.Int32'
```

This didn't work, because the List still gives out objects instead of ints, even though it only holds ints.

Casting



Definition: Typecast

The conversion of a variable's data type to another data type to bypass some restrictions imposed on datatypes.

To get around a list storing only objects, you can cast an object individually to what its type really is, then play with it like it should be. Granted, if you cast to something that is improper, say a string to an int, Boo will emit an error.

There are two ways to cast an object as another data type.

1. using `var as <type>`
2. using `var cast <type>`

casting example

```
list = List(range(5))
print list
for item in list:
    print ((item cast int) * 5)
print '---'
for item as int in list:
    print item * item
```

Output

```
[0, 1, 2, 3, 4]
0
5
10
15
20
---
0
1
4
9
16
```



Recommendation

Try not to cast *too* much.
If you are constantly *cast*-ing, think if there is a better way to write the code.



Upcoming feature: Generics

Generics, which will be part of the .NET Framework 2.0, will allow you to create a `List` with a specified data type as its base. So soon enough, there will be a way to not have to *cast* a `List`'s items every time.

Hashes



Definition: Hash

A `List` in which the indices may be objects, not just sequential integers in a fixed range.

Hashes are also called "dictionaries" in some other languages.

Hashes are very similar to `Lists`, except that the key in which to set values can be an object, though usually an `int` or a `string`.

Hashes can be defined two common ways:

1. by using braces `{}`
2. by creating a new `Hash` wrapping an `IEnumerator`, or an `IDictionary`.

hash example

```
hash = {'a': 1, 'b': 2, 'monkey': 3, 42: 'the answer'}
print hash['a']
print hash[42]
print '---'
for item in hash:
    print item.Key, '=>', item.Value

# the same hash can be created from a list like this :
ll = [ ('a',1), ('b',2), ('monkey',3), (42, "the answer") ]
hash = Hash(ll)
```

Output

```
1
the answer
---
a => 1
b => 2
monkey => 3
42 => the answer
```

Exercises

1. Produce a List containing the fibonacci sequence that has 1000 values in it. (See if you can do it in 4 lines)

Go on to [Part 06 - Operators](#)