```
>
```

1 Sequences, Sets and Lists

Beyond the elementary mathematical objects like numbers and strings, Maple has a wide variety of structures that are an integral part of the programming toolkit. The simplest of these are sequences, lists and sets.

1.1 Sequences

A sequence in Maple is the same as a finite sequence in mathematics. It can be simply defined as follows.

```
> x:= 1, 2, 4, 8, 16;
```

$$x := 1, 2, 4, 8, 16$$

```
>
```

This defines x to be a sequence of five elements. You can refer to the ith element of the sequence by x[i]

```
> x[1];
```

$$1$$

```
> x[5];
```

$$16$$

Sequences can be generated using the seq command. The format is

seq(function, range of function variable).

For example, to generate a sequence of the first ten squares, type:

```
> seq(i^2, i=1..10);
```

$$1, 4, 9, 16, 25, 36, 49, 64, 81, 100$$

```
> seq(ithprime(n), n=2..10);
```

$$3, 5, 7, 11, 13, 17, 19, 23, 29$$

```
>
```

As you can imagine, the function ithprime(i) gives the ith prime.

seq(f(i), i=n..m) generates the sequence f(n), f(n+1), ..., f(m).

An element can be added in front of or at the end of a sequence by simply appending the term.

```
> z:= x,32;
```

$$z := 1, 2, 4, 8, 16, 32$$

```
> z:= 0, z;
```

$$z := 0, 1, 2, 4, 8, 16, 32$$

A subsequence of consecutive elements can be selected using the range operation. z[2..5] will select z[2], z[3], z[4] and z[5]

```
> z[2..5];
```

$$1, 2, 4, 8$$

1.2 Lists

A sequence can be made into the standard list data structure by enclosing it in brackets.

list := [sequence].

Maple lists behave like lists in any other language. There are a few basic functions to manipulate lists and these can be combined to produce any list manipulation function.

```
> x:= [seq( 2^i, i=1..6)];
```

$$x := [2, 4, 8, 16, 32, 64]$$

```
> x[ 4];
```

$$16$$

```
>
```

A list can be converted back into a sequence by the op command op(list);

```
> op(x);
```

$$2, 4, 8, 16, 32, 64$$

This means that operations on lists can be performed using the corresponding operations on sequences. For example, to add something to the end of a list, convert it to a sequence, add a number, and then convert it back by enclosing it in brackets.

```
> x:= [op(x), 256];
```

$$x := [2, 4, 8, 16, 32, 64, 256]$$

Insert an element into a list by breaking it into two sequences and then combining them with the new element. To insert 128 between 64 and 256, do the following.

```
> x:= [ x[1..6], 128, x[7..nops(x)] ];
```

$$x := [2, 4, 8, 16, 32, 64, 128, 256]$$

```
>
```

Here we used the function nops to determine the number of elements in x. In general, the function nops gives the number of elements of a sequence, list, or set.

To replace an element in the list, we use subsop.

subsop (elementnumber = replacement value, list).

```
> subsop(  5= e,  [a,b,c,d, f] );
```

$$[a, b, c, d, e]$$

```
>
```

Later we will build a set of procedures for carrying out standard list operations like insertion, deletion, replacement and so on. These are built from the basic operations outlined above. Also, as you may have observed, there is essentially no difference between a sequence and a list. This is not the case with sets, which we now explore.

1.3 Sets

A set in Maple is represented in braces.

A:= { 561, 1729}

The basic difference between a set and a list is that elements can be repeated in a list. By its definition, an element occurs only once in a set. A sequence can be converted into a set by enclosing it in braces. This eliminates any duplicates.

```
> x:= 1,1,2,3,2,4,5,6:
> z:={x};
```

$$z := \{1,2,3,4,5,6\}$$

```
> x;
```

$$x := \ 1,2,3,4,5,6$$

The basic set operations are union, minus (for set difference) and intersect.

```
> { 1,1,2,3} union {5, 8,13};
```

$$\{1,2,3,5,13,8\}$$

```
> op(");
```

$$1,2,3,5,13,8$$

```
>
```

There is no particular order to the representation of elements in a set. Herell we used op to convert the set to a sequence and " refers to the previous output.

```
>  { 1,1, 2,3,5,8, 13,21} minus { 2,8};
```

$$\{1,3,5,13,21\}$$

```
> { 1, 1, 2, 3, 5, 8, 13, 21} intersect { 3, 5, 7, 11, 13, 17};
```

$$\{3,5,13\}$$

```
>
```

Other useful set related operations are member and select. The first allows us to test if an element is a member of a set or a list and the second allows us to choose a subset of elements satisfying the specified criterion.

Examples are the following.

```
> member ( 5, { 1, 1, 2, 3, 5, 8, 13}, 'pos');
```

$$\text{true}$$

```
>
> pos;
```

$$5$$

Here the third argument specified the variable pos which contained the place of the element in the set.

The select command works with a true/false function to pick elements out of a set. For example, isprime returns true if its argument is prime.

```
> isprime(29);
```

$$\text{true}$$

```
> select( isprime,  { 2, 3, 4, 9, 11, 17, 18} );
```

$$\{2,3,11,17\}$$