4. DATA TYPES

4.1 Sequences

In MAPLE, sequences take the form

$$expr1,\ expr2,\ expr3,\ \ldots,\ exprn.$$

```
> x := 1,2,3;
```

$$x := 1, 2, 3$$

```
> y := 4,5,6;
```

$$y := 4, 5, 6$$

```
> x,y;
```

$$1, 2, 3, 4, 5, 6$$

We observe that in MAPLE, `x,y` concatenates the two sequences x and y. There are two important functions used to construct sequences: `seq` and the repetition operator `$`.

```
> f:='f':   seq(f(i), i=1..6);
```

$$f(1), f(2), f(3), f(4), f(5), f(6)$$

```
> seq(i^2, i=1..5);
```

$$1, 4, 9, 16, 25$$

```
> x:= 'x':
> x$4;
```

$$x, x, x, x$$

In general, `seq(f(i), i=1..n)` produces the sequence

$$f(1),\ f(2),\ \ldots,\ f(n)$$

and `x$n` produces a sequence of length n

$$x,\ x,\ \ldots,\ x$$

The `op` function can be used to create sequences.

```
> b:='b':  c:='c':
> L := a+b+2*c+3*d;
```

$$L := a + b + 2c + 3d$$

```
> op(%);
```

$$a,\ b,\ 2c,\ 3d$$

`op(expr)` produces a sequence whose elements are the operands in `expr`.

```
> nops(L);
```

$$4$$

```
> op(3,L);
```

$$2c$$

`nops(expr)` gives the length of the sequence `op(expr)` and `op(j,expr)` gives the jth term in the sequence `op(expr)`.

If s is a sequence, then the jth term of the sequence is $s[j]$.

```
> s := 1, 8, 27, 64, 125;
```

$$s := 1,\ 8,\ 27,\ 64,\ 125$$

```
> s[3];
```

$$27$$

4.2 Sets

We have already seen the set data type in Section 3.2.2 when solving systems of equations. In MAPLE, a *set* takes the form

$$\{expr1,\ expr2,\ expr3,\ \ldots,\ exprn\}.$$

In other words, a set has the form $\{S\}$ where S is a sequence. A set is a set in the mathematical sense — order is not important.

```
> y := 'y':   {x,y,z,y};
```

$$\{x,\ y,\ z\}$$

Observe that $\{x, y, z, y\} = \{x, y, z\}$. MAPLE can perform the usual set operations: union, intersection, and difference.

```
> a := {1,2,3,4}; b := {2,4,6,8};
```

$$a := \{1,\ 2,\ 3,\ 4\}$$
$$b := \{2,\ 4,\ 6,\ 8\}$$

```
> a union b;
```

$$\{1,\ 2,\ 3,\ 4,\ 6,\ 8\}$$

```
> a intersect b;
```

$$\{2,\ 4\}$$

```
> a minus b;
```

$$\{1,\ 3\}$$

We can also determine whether a given expression is an element of a set using the function `member`.

```
> member(2,a);
```

$$true$$

```
> member(5,a);
```

$$false$$

```
> a[3];
```

$$3$$

So `member(x,A)` returns the value `true` if x is an element of A and `false` otherwise. Also, the jth element of the set A is `A[j]`.

4.3 Lists

In MAPLE, a *list* takes the form

$$[expr1,\ expr2,\ expr3,\ \ldots,\ exprn].$$

Here order is important.

```
> a:='a':  b:='b':
> L1 := [x,y,z,y]; L2 := [a,b,c];
```

$$L1 := [x,\ y,\ z,\ y]$$
$$L2 := [a,\ b,\ c]$$

```
> L := [op(L1),op(L2)];
```

$$L := [x,\ y,\ z,\ y,\ a,\ b,\ c]$$

```
> L[5];
```

$$a$$

We observe that the lists $L1$ and $L2$ can be concatenated by the command `[op(L1),op(L2)]` and that `L[j]` gives the jth item in the list L. Lists can be created from sequences:

```
> s := seq( i/(i+1), i=1..6);
```

$$s := 1/2,\ 2/3,\ 3/4,\ 4/5,\ 5/6,\ 6/7$$

```
> M := [s];
```

$$M := [1/2,\ 2/3,\ 3/4,\ 4/5,\ 5/6,\ 6/7]$$

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

$$[2/3,\ 3/4,\ 4/5,\ 5/6]$$

So, `M[i..j]` gives the ith through jth elements of the list M.

4.4 Tables

In MAPLE, a *table* is an array of expressions whose indexing set is not necessarily a set of integers. Sounds bizarre? Let's look at some examples. Tables are created by the `table` function.

```
> T := table([a,b]);
```

$$T := \text{table}([\ 1 = a,\ 2 = b\ ])$$

```
> T[2];
```

$$b$$

So, if L is a list, then `table(L)` converts L into a table. The jth element of this table T is given by `T[j]`. Try

```
> S := table([(1)=A,(3)=B+C,(5)=A*B*C]);
> S[3];
> S;
> op(S);
```

For the table S, the indexing set is $\{1, 3, 5\}$ and thus does not necessarily have to be a set of consecutive integers. See `?table` for more bizarre examples. In your session you should have found that `S` did not return the table, but that `op(S)` did.

4.5 Arrays

In MAPLE, an *array* is a special kind of a table. It most resembles a matrix. Let's look at some examples.

```
> A := array(1..2,1..3);
```

$$A := \text{array}(1..2, 1..3, [\ ])$$

```
> op(A);
```

$$\begin{bmatrix} ?_{1,1} & ?_{1,2} & ?_{1,3} \\ ?_{2,1} & ?_{2,2} & ?_{2,3} \end{bmatrix}$$

```
> B := array(1..2,1..2,1..2);
```

$$B := \text{array}(1..2, 1..2, 1..2, [\ ])$$

```
> op(B);
```

$$\text{array}(1..2, 1..2, 1..2, [$$
$$(1, 1, 1) = ?_{1,1,1}$$

$$(1,1,2) = ?_{1,1,2}$$
$$(1,2,2) = ?_{1,2,2}$$
$$(2,1,1) = ?_{2,1,1}$$
$$(2,1,2) = ?_{2,1,2}$$
$$(2,2,1) = ?_{2,2,1}$$
$$(2,2,2) = ?_{2,2,2}$$

])

We see that the array A corresponds to a 2×3 matrix. The array B corresponds to $2 \times 2 \times 2$ matrix or, if you like, a table with indexing set

$$\{(1,1,1),\ (1,1,2),\ \ldots,\ (2,2,2)\}.$$

We can insert entries into an array by using subscripts (or indices).

```
>  C:=array(1..2,1..2):
>  C[1,1]:=1:  C[1,2]:=2:  C[2,1]:=3:  C[2,2]:=7:
>  op(C);
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 7 \end{bmatrix}$$

```
>  print(C);
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 7 \end{bmatrix}$$

Observe that we can print out an array using the `print` command. An alternative method for entering arrray entries is given below.

```
>  F:=array(1..2,1..3,[[1,2,3],[5,9,7]]);
```

$$F := \begin{bmatrix} 1 & 2 & 3 \\ 5 & 9 & 7 \end{bmatrix}$$

4.6 Data conversions

The function `type` checks the data type of an object.

```
>  A := {1,2,3}:
>  s := 1,2,3:
>  L := [1,2,3]:
>  T := table([1,2,3]):
>  M := array(1..3,[1,2,3]):
>  type(L,list);
```

true

```
>  type(T,set);
```

false

The function `convert` can be used to convert from one data type to the other.

```
> convert(A,list);
```

$$[1, 2, 3]$$

```
> convert(L,set);
```

$$\{1, 2, 3\}$$

The `whattype` function is used find the type of an expression.

```
> whattype(A);
```

$$set$$

```
> whattype(s);
```

$$exprseq$$

```
> whattype(L);
```

$$list$$

```
> whattype(T);
```

$$symbol$$

```
> whattype(op(T));
```

$$table$$

```
> whattype(M);
```

$$symbol$$

```
> whattype(op(M));
```

$$array$$

See `?whattype` for more information.

4.7 Other data types

In this chapter we have seen a small sample of MAPLE's data types. To see a complete list, try

```
> ?type
```