**Chapter 8 Short Summary**

\* To *Define a Function* in maple

e.g. Write in Maple, to mean

\* *Map* and *Tilde*

The function *map( )* applies the value of its first argument to every element of its second argument and passes any subsequent arguments as subsequent arguments to its first argument, like this:

**> **



**> **



Alternatively use the tilde keyboard-character ~ (*not* the one in the palette)

**> **



\* *Select, Remove, Selectremove*

*select()* and *remove()*, selects or removes elements for which a predicate evaluates to true for every element of a data structure.

e.g.

**> **



**> **



The set denoted mathematically by , where  is a predicate, can be constructed explicitly in Maple as .

*selectremove()* splits it the data structure into two:

e.g.

**> **



This can also be done as a mapping as follows:

**> **



(\* *Mapping* vs *Expression*.)

Maple understands "function algebra". For example,  = ,  =  and  = .

\**Domain*, *Codomain*, *Range*

To find the range of a function , where A is a finite set, use 

or .

\*Specifying *Functions* *on* *Multidimensional* *Sets*

(A multidimensional set is the Cartesian product of 2 or more sets.)

e.g. (The function )

**> **



**> **



This can also be implemented using vectors.

\**Piecewise* *Functions*

Option 1) Use the expressions palette.

To add rows to a piecewise template, select the template and press Ctrl+Shift+R.

e.g. (absolute value function)

**> **

Option 2) Use the function .

The conditions are evaluated in order until  evaluates to , in which case the value of the piecewise-defined expression is . If no condition evaluates to  then the value of the expression is the final unpaired value, if there is one. Otherwise it evaluates to 0.

\*Defining *Finite Functions* in various ways:

Option 1)

**> **











Option 2) (parallel assignment)

**> **



Option 3) (map or tilde)

**> **



or

**> **



Option 4) (using assign() function)

**> **



\* *Plotting Functions* on *Finite* sets

e.g.

**> **



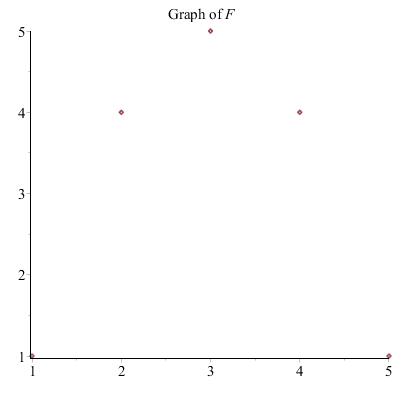
**> **



**> **



**> **



(\* function equality)

\* Checking *Injective*, *Surjective*, *Bijective* (for a finite function):

For functions on finite sets we have that

is injective ⇔,

 is surjective ⇔ .

Thus define the following functions:

**> **

**> **

**> **



(Remember how to implement: suppose we have defined the sets , and the function , then you need to type , which will return a logical value, to test for surjectivity)

\* Constructing *the Inverse function* on a *finite* set. (using a loop or using assign)

(Remember the inverse only exists for bijective functions)

Option 1) (use loop)

e.g.

Suppose  is defined as follows:

**> **



**> **



**> **



Implement the requirement that  for all as follows:

**>** 















Option 2) (use assign)

Construct a sequence of equations that specify the inverse function:

**> **



Then apply the function  to this sequence of equations to define the function :

**> **