INTRODUCTION TO CONSTRUCTING AND USING FUNCTIONS IN MATHEMATICA

Different Types of Equal Signs

We have discussed that there are a number of different types of equal signs in Mathematica. The single equal sign, as for example x = 3, immediately assigns the value of 3 to the variable x, and the value of x will remain equal to 3 until we redefine or clear the value of x. So, if we set x = 3 and then try to solve the trivial equation :

```
In[22]:= x = 3;
         Solve[x - 4 == 0, x]

         General::ivar : 3 is not a valid variable. >>

Out[23]= Solve[False, 3]
```

We get the error message that 3 is not a valid variable. If we clear the value of x :

```
In[24]:= Clear[x]
         Solve[x^2 - 5 x + 6 == 0, x]

Out[25]= {{x → 2}, {x → 3}}
```

We obtain the solutions we expect. In fact, we encountered a second type of equal sign, the double equal sign (==); we use this in *Mathematica* when we are testing whether one side of an expression equals the other side. This makes the double equal sign the appropriate symbol to use in solving equations

Defining Functions in Mathematica

(For more information on this topic, see functions in the online Doc Center.)

Suppose we want to write the Mathematica version of a simple function, say f (x) = x^2. We write this as:

```
In[28]:= Clear[x]
         f[x_] := x^2
```

It is important to put the underscore in the brackets on the left, but no underscore on the right. (In Mathematica jargon, this use of underscore is called a "blank"). In essence, we have created our own (albeit not very novel) function. We can evaluate this function :

In[31]:=
```
f[7]
```

Out[31]= 49

We can evaluate expressions as well as numbers :

In[35]:=
```
f[q]
f[q^2]
f[z^3 + z^2 + z + 1]
```

Out[35]= q^2

Out[36]= q^4

Out[37]= $\left(1 + z + z^2 + z^3\right)^2$

A short program involving functions

Let' s see how we might construct a function to calculate Fibonacci numbers. You may know that Fibonacci numbers are defined such that the nth Fibonacci number is equal to the sum of the two prior numbers, or mathematically :

$$F_n = F_{n-1} + F_{n-2}$$

In order to calculate these numbers, we need to establish what the first two Fibonacci numbers are, since we need a starting point for our calculations. Fibonacci numbers are defined in such a way that :

$$F_1 = F_2 = 1$$

What will we need to write a program to compute Fibonacci Numbers? We will need to construct a function that we can evaluate, and also will need to define the first two terms. If we call our function fib, we write :

```
Clear[fib]
fib[1] = 1; fib[2] = 1;
fib[n_] := fib[n - 1] + fib[n - 2]
```

Let' s review each step of this program. The first step clears the value of fib so that no possible prior values of fib will be carried through. The second line defines the first and second Fibonacci numbers to equal one. Notice the use of semi - colons; semi - colons in Mathematica are used to suppress output. By inserting semi - colons, the output will not include the input statements fib[1] = 1 or fib[2] = 1.

Finally, the third statement defines our function; the nth Fibonacci number is the sum of the two prior Fibonacci numbers. Let' s see if it all works; let' s calculate the next five Fibonacci numbers . We can do this inelegantly as :

```
In[57]:= fib[3]
         fib[4]
         fib[5]
         fib[6]
         fib[7]

Out[57]= 2

Out[58]= 3

Out[59]= 5

Out[60]= 8

Out[61]= 13
```

Or more compactly :

```
In[62]:= Do[Print[fib[i]], {i, 3, 7}]

2
3
5
8
13
```

Or with a bit more flourish :

```
In[86]:= Print[Style["i", Bold, FontSize → 16], "    ", Style["Fib. no.", Bold, FontSize → 16]]
         Do[Print[i, "       ", fib[i]], {i, 3, 7}]
```

```
i  Fib. no.
3      2
4      3
5      5
6      8
7      13
```

An Important Mathematica "Trick"

Now that we have a function written to compute Fibonacci numbers, we can, in principal, calculate any Fibonacci numbers we wish. As a test, calculate the value of the 35 th Fibonacci number ... be patient it will take a while. We can get a sense of the time it takes to produce this result using :

```
In[91]:= Timing[fib[35]]
Out[91]= {22.781, 9 227 465}
```

The solution set in output statement 91 consists of the time (in secs) required to compute the 35 th Fibonacci number. This computation took so long because *Mathematica* did not store any prior results. In other words, if we were to do this calcuation by hand, we would simply add the two most recent Fibonacci numbers to produce the next number in the sequence. In essence, by writing numbers on a page or storing values on our calculators, we store each value and merely need to add to previous values.

Not so with Mathematica; in order to get Mathematica to store prior results, we make use of this simple trick (uh, technique) :

```
In[92]:= Clear[fib]
         fib[1] = 1; fib[2] = 1;
         fib[n_] := fib[n] = fib[n - 1] + fib[n - 2]
         Timing[fib[35]]
Out[95]= {0., 9 227 465}
```

You might have noticed that calculation went a bit faster. What is the difference in the two programs? Look in the third line. Notice that we inserted "fib[n]" between the declaration of the function ("fib[n_]:=") and the definition of Fibonacci numbers ("fib[n-1]+fib[n-2]"). This insertion instructs Mathematica to store all previously calculated values of "fib[n]"; the computation of Fibonacci numbers occurs much more quickly when Mathematically does not have to start from n=1 each time a new number is computed.

Your turn

Try to write a program that will compute n!. Check your results against the values computed from the Mathematica function for factorial, e.g. :

In[119]:=
```
20!
```
Out[119]= 2 432 902 008 176 640 000