1 Introduction

The first step in using Maple to solve problems and understand mathematical objects is to familiarize yourself with the basic features of the language. This lesson is a brief introduction to various parts of Maple. The later chapters contain a detailed description of these and additional features of Maple.

When you first start Maple, you are presented with a blank screen and a > prompt. The prompt indicates that Maple is ready to accept your commands (that is, it is in input mode). This may vary from system to system, so if your version of Maple does not start this way, then please check with your system administrator or User's Guide.

At startup, you will be presented with a prompt as below.

```
>
```

Now, you can enter commands. Let us briefly explore some commands that allow us do simple arithmetic.

```
> 2 + 7;
                                    9
```

The output is displayed in the center of the window, below the command. You can enter any arithmetic expression involving the operators +, -, *, /, ^ (for exponentiation). Use parenthesis '(' and ')' for grouping expressions to reflect the algebraic structure.

```
> 2^17 -1;
                                 131071
```

To compute , enter the following;

```
> (2^24 - 1)/(2^8-1);
                                  65793
```

Sometimes, we want do the computations, but not display all the results. This is necessary when there are many intermediate computations or the output is very large. This is achieved by placing a colon instead of a semicolon at the end of the expression.

```
> 2^100 -1:
```

A colon or semicolon tells Maple that an expression should be evaluated. A long expression or statement can be split over several input lines and Maple will evaluate everything only after the colon or semicolon.

```
>  2+ 3 *(
>  5 + 7)
>  - 11* 5;
                                   −17


> 2+ 3*(5+7) - 11*5;
                                   −17
```

The multiplication operator is necessary to indicate multiplication. If you simply type 3(5+7), then Maple doesn't know that multiplication is implied in the expression.

```
> 3( 5+7);
                                    3
```

The exclamation ! represents factorial.

```
> 5!/(3! * 2!);
                                   10
```

Let us repeat the basic rules for entering expressions.

1. Each expression is terminated by a colon or a semi-colon. If you don't enter these then, Maple just sits there waiting for input and you will be wondering why you are not getting any output.

2. Use parenthesis to reflect the structure of the arithmetical expression. Multiplication is not implied by the use of parenthesis. Use the * operator explicitly.

3. You can enter any number of commands in a line. Maple waits for a carriage return or enter key to evaluate the line.

```
> 22/7; 335/113;
```

$$\frac{22}{7}$$
$$\frac{335}{113}$$

2 Variables

Variables can be defined by assigning quantities to them. The assignment operator in Maple is := and not = as in many programming languages. This is a frequent source of error, as Maple does not usually give error messages when = is used in place of :=. The symbol = is used for testing equality of two expressions.

A variable's value can be dislayed by typing the variable name followed by a semi-colon.

```
> x:= 2 + 9;
```

$$x := 11$$

```
> x;
```

$$11$$

```
> y =2;
```

$$y = 2$$

```
> y;
```

$$y$$

The use of equality above does not assign any value to y. We use = to test for equality as follows. The evalb function can be used to check the truth of the statement. The evalb is part of the family of eval functions that are discussed below in the section on functions.

```
> evalb(x= 11);
```

$$\mathrm{true}$$

Since Maple is a symbolic language, you can use variable names in all your expressions. Maple will evaluate the statements if the variables have a value. An expression will also be evaluated later when the variables have a value.

```
> speed := DistanceTraveled/ Time;
```

$$speed := \frac{DistanceTraveled}{Time}$$

```
> DistanceTraveled := 200;
```

$$DistanceTraveled := 200$$

```
> Time:= 3.0;
```

$$Time := 3.0$$

```
> speed;
```

$$66.66666667$$

Maple remembers all expressions and evaluates them as far back as possible. Hence repeated use of the same variable name can have unintended consequences.

```
> poly:= 2*x^2 - 3* x +5;
```

$$poly := 2\,x^2 - 3\,x + 5$$

```
> x:=2:
```

We have defined an expression for poly involving x and then we set the value of x. Maple will then evaluate poly.

```
> poly;
```

$$7$$

Assigned values can be unassigned or cleared by enclosing the variable name within backquotes.

```
> x:='x';
```

$$x := x$$

This unassigns the value of x. But the value of poly is still 7 as it was evaluated before. Clearing a value can also be achieved by using the evaln function.

```
> poly;
```

$$7$$

We will clear poly using the evaln function.

```
> poly:=evaln(poly); poly;
```

$$poly := poly$$

$$poly$$

3 Functions

The power of a computer algebra system such as Maple comes from its vast library of built-in functions. A major part of learning to use Maple is in getting familiar with these built-in functions. Maple has all the standard mathematical functions and many more speciliazed functions. The following is a brief introduction. In later chapters, we will introduce more functions as we need them.

Here are some arithmetic functions and examples of their usage.

modp(a, m) : Gives the remainder when a is divided by m;

This can also be used as

a mod m;

```
> modp( 37, 7);
```

$$2$$

```
> 37 mod 7;
```

$$2$$

```
>  37 * 2 mod 7;
```

$$4$$

The expression 37*2 mod 7 is equivalent to modp(37*2, 7).

The binary operator a mod b evaluates both the expressions a and b and then evaluates the remainder. It is important to use parenthesis to group expressions properly to reflect their mathematical structure.

```
> 2^17 -1 mod 11;
```

$$6$$

```
> modp( 2^17 -1, 11);
```

$$6$$

This is very different from $2^11 - (1 mod 11)$.

```
>
```

The floor and ceiling functions are represented by floor(x) and ceil(x) respectively.

```
>
> floor(3.7);
```

$$3$$

```
> ceil( 3.7);
```

$$4$$

A graph of the floor or ceiling functions can be observed using the plot command. The format is the following. The range for the variable t is specified by two dots, 0..5.

```
> plot( floor(t), t= 0..5);
```

The modp function can be used to define a quotient function. A function is defined by specifying a rule. We want to have quotient(a,m) = (a- modp(a,m))/m.

```
> quotient:= (a,m) -> ( a- modp(a,m) )/m;
```

$$quotient := (a, m) \to \frac{a - \mathrm{modp}(a, m)}{m}$$

```
> quotient( 37, 5);
```

$$7$$

```
> quotient( -37, 5);
```

$$-8$$

This definition of quotient is compatible with the standard requirement that the remainder be positive.

The remainder and quotient for integers is also computed using the functions irem and iquo. You can find out more about built-in functions using the command ?function name, such as ?irem.

```
> ?irem
> irem(89, 11);
```

$$1$$

A few additional functions that are useful in number theory are the following.

ithprime(k) gives the kth prime.

isprime(p) returns true if p is prime and false if p is composite.

ifactor(n) gives the prime factorization of n (whenever computationally feasible).

igcd and ilcm give the greatest common divisor and least common multiple of a set of numbers.

```
> isprime( 919);
```

true

```
> igcd( 17, 89);
```

$$1$$

```
>
```

4 Getting Help

One of the first things you should do is to familiarize yourself with the function browser. The browser is opened from the Help menu. The browser contains the complete list of functions in a format that is easy to navigate. In addition, the syntax of all expressions are outlined.

For example, the number theory functions are in Mathematics/NumberTheory. Selecting a function and then press help. This will bring up a window that contains a detailed description of the function and examples of its usage.

You can also bring up the help window for any function by using the ? operator.

?function name; will bring up the help window for the function name.

```
> ?irem;
>
>
```