Note: The worksheet below is already old but it contains a number of useful things. I've added a few comments (Note:..) to this document.

Introduction to Maple

This worksheet is intended to get you started using Maple. By reading this worksheet, executing all commands and studying the output of those commands you will become familiar with some of the possibilities of Maple. At least you will then be able to work with Maple and explore it further.

Some very usefull books concerning Maple are:

B.W. Char, K.O. Geddes, G.H. Gonnet, B.L. Leong, M.B. Monagan, S.M. Watt: First Leaves: A Tutorial Introduction to Maple,

B.W. Char, K.O. Geddes, G.H. Gonnet, B.L. Leong, M.B. Monagan, S.M. Watt: Maple V Language Reference Manual

and

B.W. Char, K.O. Geddes, G.H. Gonnet, B.L. Leong, M.B. Monagan, S.M. Watt: Maple V Library Reference Manual.

There are many more books on Maple which are very worthwhile.

Besides the text in this worksheet you will also see lines starting with a greater than sign (>). These lines are command lines and the text on those lines are Maple commands. By putting the cursor on such a line and hitting the Return-key the command on that line is executed by Maple and the output is shown on the screen. After executing a Maple command in such a way the cursor is automaticcally placed on the next line containing a Maple command.

Maple contains of course an ordinary pocket calculator. This is shown by the following Maple commands.

```
> 23+59;
> 5*12-13;
> 2^10;
```

```
> 3^(1/2);
> sqrt(5);
> sin(1/2*Pi);
```

You immediately may notice the following:

- A Maple command is always ended by a semicolon (;) (or a colon (:) as you will see later).
- Multiplication is denoted by an asterisk (*), division by a slash (/).
- Maple also knows functions like sqrt, sin, cos, tan, arcsin and so on.
- Maple knows some constants like Pi, E and gamma.

It may happen that you forget to type the semicolon at the end of your input line and hit the Return-key. In that case Maple won't do anything since it considers the input not yet to be completed. You can then still complete the input line by typing a semicolon on the next line and hitting the Return key. The following example will illustrate this.

```
> 45-16
> ;
```

The previous example is in fact an example of a command distributed over several input lines. Maple will read all input lines until it encounters a semicolon. It will then concatenate all input lines and execute the command this will give. The following is an example of this.

```
> 345+523*
> (23-45/3);
```

In your input you may go to the next line at all places except when entering a number or a function name. If however you want to go to the next line inside a number or function name you can use the backslash (\ ).

```
> 31312321321
> 6765767231;
> 34324323432\
> 423423432;
```

The following examples will show a very important characteristic of computer algebra systems: exact arithmetic. This means that expressions are not approximated but are

computed exactly.

```
> 3*(1/3);                    # Compare with 0.99999999
> (sqrt(5))^2;
```

In this example we encountered the #-symbol. Everything behind this symbol is ignored by Maple. This can be used to put comments between your computations.

It is however possible to approximate expressions by floating point numbers using the ‘evalf‘ command..

```
> 1/3;
> evalf(%);
```

In the example above you have already seen an example of the double quote (") operator (or % in Maple V Release 5). This operator stands for the previous result. In the same way two double quotes stand for the second previous result and you can even use triple double quotes.

The command evalf (evaluate to floating point number) evaluates its operand to a floating point number with accuracy determined by the Maple variable ’Digits’ (default value 10). By changing the value of ’Digits’ the accuracy can be modified.

```
> Digits:=40;
> evalf(1/3);
> Digits:=10;     # its default value
```

In the last example you have seen how you can change the value of a variable in Maple. You can create your own variables, give them values and use them inside expressions.

```
> number_of_people:=10;
> number_of_people;
> number_of_people+1;
```

In this last example we have assigned the value 10 to the variable ’number_of_people’. In the second line we asked for the value of number_of _people and in the third line we have used the variable in an expression. You see that variable names may consist of many characters. Also digits may be used but a variable name may not start with a digit.

It is a good habit to give your variables names that express what they mean. This will enlighten your calculation for yourself and for other people.

If you have assigned a value to a variable it will keep this value until you change it to another value. If you want a variable to have no value anymore you can use the single quote (’) as in the following example.

```
> number_of_people:=’number_of_people’;
> number_of_people;
```

Another important feature of Maple is its on-line help. By clicking on the word ’Help’ in the upper right corner of this Maple window you will see a menu which is the entrance to the on-line help facility of Maple. The best way to learn how to use this on-line help is experimenting by yourself.

Now we will show you a lot of Maple commands. By executing these commands you will get an idea of some of the capabilities of Maple. Notice however that this will show only a very small part of the power of Maple.

Integers

```
> 5^(5^5);
> length(%);
> 30!;
> ifactor(%);
> 2^89-1;
> isprime(%);
> a:=121932009755;
> b:=80780187944;
> igcdex(a,b,’s’,’t’);
> a*s+b*t;
> a:=’a’; b:=’b’;
```

In this last example you see that you can put more commands on one input line.

Rational numbers

```
> (1/3+1/5)*3/6;
```

Real numbers and floating point numbers

```
> (34/25)^(21/10);
> evalf(%);
> 1/3; 1/3.0;
> Digits:=40;
> evalf(E);
> evalf(sqrt(2));
> Digits:=10;
```

Complex numbers

```
> c:=2+3*I;
> (3*c^5+2*c^3+10)/(7*c^3+2*c^2-5);
> abs(c);
> Re(c);
> argument(c);
> c:='c':
```

Here you see that you can also end a command with a colon (:). In that case Maple will not show the output.

Modular arithmetic

```
> 47 mod 5;
> 5^(5^5) mod 7;
> 1/11 mod 7;
```

Algebraic numbers

```
> alias(w=RootOf(_Z^3+_Z^2+2));
> w^2;
> evala(%);
> w^3;
> evala(%);
> (w^5+3*w+1)/(2*w^4-5*w^3+2);
> evala(%);
> alias(w=w):        # unaliassing
```

Note: you should use alias only for interactive working with Maple. Never use alias inside a program because that leads to tricky problems. Instead of doing:
alias(w = RootOf(...));
you can also do:
w := RootOf(...));
Then everything computes in the same way. The only difference you will notice is that in the latter approach, the RootOf(...) will appear as RootOf(...) on the screen, whereas if you use alias it will look like w on the screen. So the alias function alters the way things look on the screen. That is the main difference with simply doing: w:=RootOf(...). Personally, I never use alias, I always use w:=RootOf(...), because alias may lead to problems.

Polynomials

This shows another important feature of computer algebra systems: symbolic computation

```
> f:=7*x^4-3*x^3+7*x^2-3*x;
> g:=5*x^5+3*x^3+x^2-2*x+1;
> f+g;
> f*g;
> expand(%);
```

You see that you must force Maple to expand the product (see also lecture on data representation).

```
> gcdex(f,g,x,'s','t');
> expand(s*f+t*g);
> factor(21*x^5-35*x^4*y+14*x^3*y^3+18*y*x^2-30*x*y^2+12*y
  ^4+9*x^3*y^2-15*x^2*y^3+6*x*y^5);
> factor(x^105-1);
> Factor(x^105-1) mod 7;
> f:=x^6+6*x^5+12*x^4+8*x^3+2*x^2+4*x-4;
> factor(f);
> alias(w=RootOf(_Z^3+_Z^2+2));
> evala(Factor(f,w));
> randpoly([x,y],terms=20,degree=7);
> f:='f': g:='g': s:='s': t:='t': alias(w=w):
```

Rational functions

```
> f:=7*x^4-3*x^3+7*x^2-3*x;
> g:=5*x^5+3*x^3+x^2-2*x+1;
> f/g;
> normal(%);
> f/g+g/f;
> normal(%);
> convert(1/f,parfrac,x);
> f:='f': g:='g':
```

General functions

```
> exp(x+y);
> expand(%);
> simplify(%);
> sin(x+y);
> expand(%);
> F:=x->x^3+5*sin(x^2)+sqrt(x);
> F(10);
> F:='F':
```

Differentiation and integration

```
> f:=7*x^4-3*x^3+7*x^2-3*x;
> diff(f,x);
> diff(x^(x^x),x,x);
> diff(log(x/(x^2+1)),x,x);
> normal(%);
> F:=x->x^3-sin(x);
> op(F);
```

The operator ‘op‘ returns the operands of an expression.

```
> diff(F(x),x);
> D(F);
> D(sin);
> 'int(7*x^4+3*x^3-5*x+11,x)';
> %;
> diff(%,x);
> 'int(1/(a+b*sin(x)),x)';
> %;
> diff(%,x);
> simplify(%);
> f:='f': F:='F':
```

Taylor series, Laurent expansions

```
> taylor(sin(x),x=0,15);
> series(cot(x),x=0,15);
```

Solving equations

```
> solve({26*x^2-y^3+1=0,2*x-y=-1},{x,y});
> subs(%[1],26*x^2-y^3+1);
> solve({x+y=3,a*x+b*y=3*a},{x,y});
```

This example shows that one has to be careful. If a=b in the system above the solution Maple gives is not the only solution. What are the other solutions?

```
> solve(x^5-x+1=0,x);
> fsolve(x^5-x-1,x);
```

We see that when Maple cannot solve an equation symbolically, you can still solve it numerically.

Solving differential equations

```
> eqn:=diff(y(x),x,x)-y(x);
> dsolve(eqn=0,y(x));
> eqn:='eqn':
```

Linear algebra

```
> with(linalg):
Warning: new definition for   norm
Warning: new definition for   trace
```

```
> A:=matrix([[3,2,1],[4,3,1],[5,4,2]]);
> B:=matrix([[3,1,0],[4,2,1],[5,2,2]]);
> evalm(A&*B);
> evalm(A+B);
> inverse(A);
> A;
> eval(A);
```

You see that the command ‘A‘ only gives ‘A‘ as an answer. To get the value of the variable ‘A‘ you must explicitly ask for it by using the function ‘eval‘. This same feature appears when the value of a variable is an array, table, procedure or function. Compare this to the variable ‘number_of_people‘ above.

```
> b:=vector([1,2,3]);
> linsolve(A,b);
```

```
> A:='A':B:='B':b:='b':
```

Plotting, 2- and 3-dimensional

```
> plot(sin(x),x=0..5*Pi);
> plot(sin(1/x),x=0.01..0.1);
> f1:=x; f3:=x-x^3/3!; f5:=x-x^3/3!+x^5/5!;
> plot({sin(x),f1,f3,f5},x=0..Pi);
```

Here you see how the succeeding polynomials are better and better approximations of sin(x) in the origin.
Do some experiments using the menus in the windows of the graphs.

```
> plot3d(x^2+3*BesselJ(0,y^2)*exp(1-x^2-y^2),x=-2..2,y=-2.
  .2);
> with(plots):
> polarplot(sin(2*t),t=0..2*Pi);
```

Now we will show some features of Maple as a programming language.

Data types

- Several kinds of numbers
- Polynomials
- Rational functions
- Series

and further

```
> 1,4,7,2,4;

  # Sequence
> [4,7,3,9,3,3];
```

```
  # List
> {3,4,4,4,5,3,2,1};

  # Set
> ‘This is a string containing the numbers 1 and 2‘;

  # String
> array(3..6,[4,8,3,1]);

  # Array
> table([(peter)=‘11-09-58‘,(mary)=‘05-12-61‘]);

  # Table
> 34,‘Hello‘,[array([[1,2],[5,6]]),Pi];
```

Note: Maple 6 has strings of the form "hello", with quotes ". Maple 5 only has strings with quotes like in the above example.

Assignment, evaluation

```
> a:=3;
> b:=a;
> b;
> a:=9;
> b;
> c:=d;
> d:=5;
> c;
> d:=7;
> c;
> ’’a’’;
> %;
> %;
> a:=’a’: b:=’b’: c:=’c’: d:=’d’:
```

Control structures

```
> i:=5; b:=3*i+1;
  # Sequence
> for i from 2 to 4 do print(i^2) od;
  # Repetition
> i:=17; while i>=5 do i:=i-5 od;
  # Repetition
> i:=-5; if i>=0 then i else -i fi;
  # Choice
> maximum:=proc(m,n)
    if m>n then
      m
    else
      n
    fi
  end;
  # Procedures
> maximum(9,4);
> maximum;
> eval(maximum);
```

You see the same feature as with matrices.

```
> i:='i':b:='b':maximum:='maximum':
```

Many, many, many,..... built-in procedures (see the on-line help).

An example: Fibonacci's numbers

```
> fib1:=proc(n)
    if n<2 then
      1
    else
      fib1(n-2)+fib1(n-1)
    fi
  end;
> fib1(20);
```

This is very slow since fib1(n) is computed over and over again (also in the procedure body). To avoid this one can use Maple's remember option.

```
> fib2:=proc(n)
  option remember;
    if n<2 then
      1
    else
      fib2(n-2)+fib2(n-1)
    fi
  end;
> fib2(50);
```

Now each computed value fib2(n) is stored and will be retrieved by table lookup.

```
> fib1:='fib1': fib2:='fib2':
```

Parameters

When you define a procedure like

f:=proc(a,b) procedure-body end;

the parameters a and b are called formal parameters. When you call this procedure like

f(c,d);

the parameters c and d are called actual parameters.

Maple uses the call-by-value paramter mechanism. This means that in the call f(c,d); first c and d are evaluated, then the respective values are assigned to the formal parameters a and b and then the procedure-body is executed. When, in a procedure-body, you want to assign a value to an actual parameter you must be sure the value passed to the procedure is a name (variable). This can be achieved using the single quote to prevent evaluation. An example:

```
> f:=proc(a,b,m)
    if a>b then
      m:=a
    else
      m:=b
    fi
  end;
> f(5,2,maximum);
> f(23,6,maximum);
```

In the first call maximum evaluates to the name maximum since it has not yet a value. The value 5 is then assigned to maximum during execution of the procedure body. In the second call of f the name maximum evaluates to 5 and this value, i.e 5, is passed to the procedure in stead of the name maximum. During execution of the procedure-body an attempt is made to assign 23 to the value 5 and this yields an error message. One can prevent this by not using maximum as actual parameter but 'maximum' (this evaluates to the name maximum).

```
> f(23,6,'maximum');
```

Notice that evaluation inside a procedure-body is different from outside a procedure-body. Outside a procedure body full evaluation is used, i.e. a name is evaluated as far as possible (except for names for arrays, tables, procedures, etc). Inside a procedure-body however we only have one-step evaluation. The following example will make this clear.

```
> x:=y;
> y:=3;
> x;
> f:=proc()
  local x,y;
    x:=y;
    y:=3;
    x
  end:
> f();
```

To force further evaluation one can use the eval function.

```
> f:=proc()
  local x,y;
    x:=y;
    y:=3;
    eval(x,2)
  end:
> f();
> f:='f':maximum:='maximum':x:='x':y:='y':
```

Names (variables)

Maple distinguishes between local and global names. A name outside a procedure-body is global. A name inside a procedure-body is local unless stated otherwise. A local name overrules the same global name. Some examples.

```
> x:=5;   # x is a global name
> f:=proc() local x; x:=3; print(x) end;    # x is local
  inside the procedure-body
> f();
> x;   # this is again the global x
> g:=proc() global x; x:=7; print(x) end;    # x is now a
  global name, even inside the procedure-body
> g();
> x;   # g() has changed the value of the global name x
```

Notice that global and local names are completely different.

```
> h:=proc() local z; z end;
> h()-z;
> x:='x':f:='f':g:='g':h:='h':
```

args and nargs

It is not necessary to actually mention all formal parameters when defining a procedure.

This is very handy when one does not know the number of parameters in advance. An example.

```
> maximum:=proc()
  local max;
   if nargs=1 then
      args[1]
    else
      max:=maximum(args[2..nargs]);
      if max > args[1] then
        max
      else
        args[1]
      fi
    fi
  end:
> maximum(34,-6,22,45,12,-9);
```

We have used the Maple-routines args and nargs. Inside a procedure-body nargs returns the number of actual parameters when the procedure is called. The routine args returns the list of actual parameters itself.

```
> maximum:='maximum':
```

As we have seen before you have to use the eval command if you want to see the definition of a procedure, defined by yourself, on the screen. This does not work for routines inside Maple, for example:

```
> eval(gcd);
```

If you want to see the definition of a Maple routine you first have to give the interface variabale ‘verboseproc‘ the right value.

```
> interface(verboseproc=2);
> eval(gcd);
```

If you want to monitor the execution of a computation you can increase the printlevel.

```
> igcd(56749620128,13112621504);
> printlevel:=10;
> igcd(56749620128,13112621504);
> printlevel:=1;              # the default value
>
```