Computer algebra, week 1, lecture 1: Assignments in Maple, sequences, lists, and sets.

Assigning a value to a variable in Maple can be done with :=
An assignment is a Maple command, and any Maple command ends with a colon or a semicolon, with : or ;

Examples of assignments:

```
> A:=3;
```

$$A := 3$$

```
> B:=4;
```

$$B := 4$$

To view the value of a variable in Maple, just type the name of the variable, and end with a semicolon

```
> A;
```

$$3$$

```
> B;
```

$$4$$

You can assign two (or more) variables at the same time:

```
> (A, B) := (5, 7);
```

$$A, B := 5, 7$$

Spaces have no meaning, so this command was the same as:

```
> (A,B):=    (   5,7)   ;
```

$$A, B := 5, 7$$

```
> A;
```

$$5$$

```
> B;
```

$$7$$

Simultaneously assigning several variables at the same time is convenient if you want to switch the values of A and B:

```
> (A,B) := (B,A);
```

$$A, B := 7, 5$$

```
> A;
```

$$7$$

```
> B;
```

$$5$$

Variables can also be used even when they have not yet been assigned a value:

```
> x;
```

$$x$$

```
> y;
```

$$y$$

```
> P:=x^3+x^5+12*x;
```

$$P := x^3 + x^5 + 12\, x$$

When x is not assigned, you can still use x. Here P is a **polynomial** in x. If x had a value, say, an integer, then P wouldn’t have become a polynomial but it would have been an integer. You can compute values of P at x = 1, 2, 3 by substitution:

```
> subs(x=1,P);
```

$$14$$

```
> subs(x=2,P);
```

$$64$$

```
> subs(x=3,P);
```

$$306$$

or with the eval command (eval is short for: evaluation)

```
> eval(P, x=3);
```

$$306$$

We could do a sequence of substitutions by the command seq (stands for sequence) as follows. Lets say we want to substitute i=1..6 for x:

```
> seq( subs(x=i,P) ,i=1..6);
```

$$14, 64, 306, 1136, 3310, 8064$$

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

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

The output of the seq command is a datastructure that’s called a sequence in Maple. In Maple a sequence is something of the form:
   expression, expression, ..., expression
So **a sequence is a number of expressions seperated by commas.**

Instead of computing the sequence subs(x=1,P), subs(x=2,P), .. we can also compute the sum: subs(x=1,P)+subs(x=2,P)+... with the add command:

```
> add( subs(x=i,P), i=1..6);
```

$$12894$$

```
> add(i, i=1..100);
```

$$5050$$

Multiplication:

```
> mul(x-i, i=1..5);
```

$$(x-1)\,(x-2)\,(x-3)\,(x-4)\,(x-5)$$

```
> mul(i, i=1..30);
```

$$265252859812191058636308480000000$$

```
> 30! ;
```

$$265252859812191058636308480000000$$

```
> ifactor(%);
```

$$(2)^{26}\ (3)^{14}\ (5)^{7}\ (7)^{4}\ (11)^{2}\ (13)^{2}\ (17)\ (19)\ (23)\ (29)$$

The command **ifactor** stands for **integer factorization**. Every Maple command must be ended with ; or with : and until now we've only used ;   Let's look at the difference between ; and :

```
> A:=2;
```

$$A := 2$$

View the contents of variable A as follows:

```
> A;
```

$$2$$

Now end the command with a colon instead of a semi-colon

```
> A:=12^12:
```

You see that with a colon, Maple prints nothing. However, the command (in this case, an assignment) did get executed, see:

```
> A;
```

$$8916100448256$$

In Maple, every command is followed by a colon : or by a semi-colon ;
If you use : or ; that makes no difference for what Maple does, the only difference is that ; causes something to be written to the screen and : does not.
Lets look some more at sequences in Maple. The following is an example of a sequence:

```
> A := 3,4,5,2,3,3,3,1;
```

$$A := 3, 4, 5, 2, 3, 3, 3, 1$$

Now you can extend the sequence to the left, or to the right, or both, as follows:

```
> A:=A,20,30;
```

$$A := 3, 4, 5, 2, 3, 3, 3, 1, 20, 30$$

```
> A:=hello,how,are,you,A,1000;
```

$$A := \mathit{hello}, \mathit{how}, \mathit{are}, \mathit{you}, 3, 4, 5, 2, 3, 3, 3, 1, 20, 30, 1000$$

```
> A[1];
```

$$\mathit{hello}$$

```
> A[2];
```

$$\mathit{how}$$

```
> A[3];
```

$$\mathit{are}$$

There is one special sequence in Maple, namely the empty sequence, denoted by NULL. See what happens:

```
> A:=NULL,NULL,A,NULL;
```

$$A := \mathit{hello}, \mathit{how}, \mathit{are}, \mathit{you}, 3, 4, 5, 2, 3, 3, 3, 1, 20, 30, 1000$$

Maple sequences are very handy. There is one problem with sequences though, and that is:
You can't tell the difference between:

1) **An expression a.**
2) **A sequence with just 1 element a.**

Because of this problem, it is often more convenient to work with lists in Maple than with sequences.
In Maple,

a list is a sequence between square brackets.

We can **turn a sequence into a list by putting square brackets around the sequence and using an assignment,** as follows:

```
> A:=[A];
```

$$A := [hello, how, are, you, 3, 4, 5, 2, 3, 3, 3, 1, 20, 30, 1000]$$

We can again look at individual elements, A[1], A[2] and such. The following command tells you the number of elements in the list:

```
> nops(A);
```

$$15$$

```
> A[15];
```

$$1000$$

```
> A[nops(A)];
```

$$1000$$

To get the last element, you can do A[nops(A)]; A shorter syntax, which does exactly the same, is A[-1], which gives you the last element. A[-2] is the second last, etc.

```
> A[-1]; # last element
```

$$1000$$

```
> A[-2]; # second last element
```

$$30$$

```
> A[-3]; # third last element, etc.
```

$$20$$

Note that everything after # are considered comments (so everything after # will be ignored in Maple computations).

The seq command is very useful for operations with: sequences, lists and sets.

If we want to let i be all elements of a list or a set A we can do:

seq(.... , i=A);

and if we want to let i be all elements of a sequence S then we can do:

seq(.... , i=[S]);

For example, compute the sequence of squares of the elements of the list A:

```
> seq(i^2, i=A);
```

$$hello^2, how^2, are^2, you^2, 9, 16, 25, 4, 9, 9, 9, 1, 400, 900, 1000000$$

Or the list of squares:

```
> [seq(i^2, i=A)];
```

$$[hello^2, how^2, are^2, you^2, 9, 16, 25, 4, 9, 9, 9, 1, 400, 900, 1000000]$$

Or the set of squares.

A set in Maple is a sequence between curly brackets {}.

Of course, if you have elements that appear more than once, the extra ones will be removed. Furthermore, the order has no meaning for sets. If S is a non-empty set, then S[1] is an element of that set but you can not predict in advance which element you will get. This is the reason that redoing the same computation in Maple can lead to a different (but mathematically equivalent) result.

```
> {seq(i^2, i=A)};
```

$$\{1, 4, 9, 16, 25, 900, hello^2, how^2, are^2, you^2, 400, 1000000\}$$

We can use the A[-1], A[-2], construction to reverse the list:

```
> [ seq(A[-i], i=1..nops(A)) ];
```

$$[1000, 30, 20, 1, 3, 3, 3, 2, 5, 4, 3, you, are, how, hello]$$

We have seen how you can convert a sequence to a list, with [ ], or to a set, with {}. The converse is also possible. You can remove the [ ] or the { } brackets by the op command. This is also useful if you want to convert a list to a set or a set to a list.

```
> A;
```

$$[hello, how, are, you, 3, 4, 5, 2, 3, 3, 3, 1, 20, 30, 1000]$$

```
> B:=op(A);
```

$$B := hello, how, are, you, 3, 4, 5, 2, 3, 3, 3, 1, 20, 30, 1000$$

```
> C:={op(A)};
```

$$C := \{1, 2, 3, 4, 5, 20, 30, are, how, you, 1000, hello\}$$

```
> nops(A), nops(C); # C has fewer elements because multiple elements
  are removed.
```

$$15, 12$$

```
> nops(B); # The nops command allows a list or set but not a
  sequence:
Error, wrong number (or type) of parameters in function nops
> nops([B]); # Now the input of nops is [B] which is a list.
```

$$15$$

As you can see, we converted list A to a sequence B, and to a set C. Since C had fewer elements than A, we can see that A must have elements that appear more than once. Note that nops does not work for a sequence, but it does work for lists and sets. So, to count the number of elements in a sequence, we have to put brackets [ ] around it before we can count with nops. Given a list, we can compute sublists as follows, using the double dot ..

```
> A[1..1];
```

$$[hello]$$

```
> A[1..4];
```

$$[hello, how, are, you]$$

```
> A[1..-1];
```

$$[hello, how, are, you, 3, 4, 5, 2, 3, 3, 3, 1, 20, 30, 1000]$$

```
> A[-3..-1];
```

$$[20, 30, 1000]$$

```
> A[13..15];
```

$$[20, 30, 1000]$$

```
> A[13..-1];
```

$$[20, 30, 1000]$$

```
> A[-3..15];
```

$$[20, 30, 1000]$$

```
> C;
```

$$\{1, 2, 3, 4, 5, 20, 30, are, how, you, 1000, hello\}$$

```
> C[1..4];
```

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

```
> C minus C[1..4];
```

$$\{5, 20, 30, are, how, you, 1000, hello\}$$

The commands minus and union can only be used for sets.

```
> B;
```

$$hello, how, are, you, 3, 4, 5, 2, 3, 3, 3, 1, 20, 30, 1000$$

```
> B[4..6];
```

$$you, 3, 4$$

We have seen before that if you have an expression involving a variable x, you can replace x by something else with a subs. How to do that for sequences?

```
> P;
```

$$x^3 + x^5 + 12\,x$$

```
> subs(x=0,P);
```

$$0$$

```
> subs(x=1,P);
```

$$14$$

```
> A;
```

$$[hello, how, are, you, 3, 4, 5, 2, 3, 3, 3, 1, 20, 30, 1000]$$

```
> subsop(1=Hi,A);
```

$$[Hi, how, are, you, 3, 4, 5, 2, 3, 3, 3, 1, 20, 30, 1000]$$

```
> A;
```

$$[hello, how, are, you, 3, 4, 5, 2, 3, 3, 3, 1, 20, 30, 1000]$$

Note that this subsop command doesn’t actually change A, it just returns A with its first entry replaced by Hi.

To change A we have to do an assignment:

```
> A := subsop(1=Hi,A);
```

$$A := [Hi, how, are, you, 3, 4, 5, 2, 3, 3, 3, 1, 20, 30, 1000]$$

```
> A := subsop(2=there, A);
```

$$A := [Hi, there, are, you, 3, 4, 5, 2, 3, 3, 3, 1, 20, 30, 1000]$$

```
> A := subsop(3=fox, A);
```

$$A := [Hi, there, fox, you, 3, 4, 5, 2, 3, 3, 3, 1, 20, 30, 1000]$$

```
> A:=subsop(4=NULL, A);
```

$$A := [Hi, there, fox, 3, 4, 5, 2, 3, 3, 3, 1, 20, 30, 1000]$$

```
> nops(A);
```

$$14$$

Here I replaced entry A[4] by the empty sequence NULL, so this is the way **you can remove an entry from a list**. For sets there are different ways to do it.

```
> C;
```

```
{1, 2, 3, 4, 5, 20, 30, are, how, you, 1000, hello}
```

```
> subs(hello=NULL,C);
```

```
{1, 2, 3, 4, 5, 20, 30, are, how, you, 1000}
```

```
> C;
```

```
{1, 2, 3, 4, 5, 20, 30, are, how, you, 1000, hello}
```

```
> C minus {hello};
```

```
{1, 2, 3, 4, 5, 20, 30, are, how, you, 1000}
```

The best way to remove an element e from a set S is by doing:
S := S minus {e};

Note that if A,B are sets then in "A minus B" the set B can be any set, it does not need to be a subset of A. Example:

```
> {hello,Hello} minus C;
```

```
{Hello}
```

In Maple, the variable Hello is different from hello because lower case and upper case is not the same.

Another example with sets: Compute all numbers that are a square of a positive integer <= 100, and that are also a sum of two squares. How to do that?
Well, first take all squares of i=1..100.
Then let i = all elements of that set of squares and let j = all elements of that set of squares, and for all of those, take i+j, and then take the set of all of that.
Then take the intersection of these two sets.

```
> squares:={seq(i^2,i=1..100)}:
> sum_squares := { seq(seq(  i+j  ,i=squares),j=squares) }:
```

I used a colon instead of semi-colon to end the command, to prevent Maple from printing lots of numbers. To find the answer to the question, we will have to compute the intersection of those two sets:

```
> squares intersect sum_squares;
```

```
{25, 100, 169, 225, 900, 400, 289, 625, 676, 841, 1156, 1225, 1369, 1521, 1600, 1681, 2025,
    2500, 2601, 2704, 2809, 3025, 3364, 3600, 10000, 3721, 4225, 4624, 4900, 5329, 5476, 5625,
    6084, 6400, 6724, 7225, 7569, 7921, 8100, 8281, 9025, 9409}
```

We want to sort this set so it’s easier to read. However, sets don’t have an order. To sort it we should make it a list:

```
> L := [op(squares intersect sum_squares)]:
> L := sort( L );
```

```
L := [25, 100, 169, 225, 289, 400, 625, 676, 841, 900, 1156, 1225, 1369, 1521, 1600, 1681, 2025,
    2500, 2601, 2704, 2809, 3025, 3364, 3600, 3721, 4225, 4624, 4900, 5329, 5476, 5625, 6084,
    6400, 6724, 7225, 7569, 7921, 8100, 8281, 9025, 9409, 10000]
```

Recalling previous expressions with the percentage sign.

The percentage sign % in Maple refers to the last thing Maple evaluated. And %% means the one before that, and %%% the one before that (there is no %%%%).

```
> a:=1;
```

$$a := 1$$

```
> b:=2;
```

$$b := 2$$

```
> c:=3;
```

$$c := 3$$

```
> d:=4;
```

$$d := 4$$

```
> %%;
```

$$3$$

At this point, the last one is 3, and so the second last is no longer 3, it's 4.

```
> %%;
```

$$4$$

```
> a;
```

$$1$$

```
> b;
```

$$2$$

```
> c;
```

$$3$$

```
> %%%;
```

$$1$$

```
> %%%;
```

$$2$$

```
> %%%;
```

$$3$$

```
> a;
```

$$1$$

```
> b;
```

$$2$$

```
> c;
```

$$3$$

```
> %;
```

$$3$$

```
> %%%;
```

$$2$$

```
>
```