First Steps in Magma

John J. Cannon
Catherine Playoust

School of Mathematics and Statistics
University of Sydney
NSW 2006, Australia

August 1996

Typeset by computer using Donald Knuth's TEX, and the document preparation system LATEX developed by Leslie Lamport.

This booklet is a very first guide to MAGMA, suitable for use in the early stages of learning the system. For more information, see *An Introduction to Magma*, *Handbook of Magma Functions*, *Solving Problems with Magma*, and the online help system.

For enquiries about Magma, contact

The Secretary
Computational Algebra Group
School of Mathematics and Statistics
University of Sydney
NSW 2006
Australia

Email: magma@maths.usyd.edu.au
Telephone: +61–2–9351–3338
Fax: +61–2–9351–4534

The Magma Home Page may be found at

http://www.maths.usyd.edu.au:8000/comp/magma/Overview.html

1 Basic Ideas

1.1 How to Use Magma

The way in which you gain access to Magma depends on your course and institution; your tutor or local manual will advise you. Generally speaking, you can enter Magma simply by typing

```
magma
```

and pressing the key marked 'return' or 'enter', which finishes the line.

Once you are in Magma, you can type in your commands. Each complete command must finish with a semicolon (;), and you should then press the return key. Whenever the computer is ready for you to type a command, it puts a *prompt* symbol, usually looking like this:

```
>
```

on the left of the input line. For instance, if you want to find the sum of 2 and 4, you should type the command print 2 + 4; after the prompt symbol, then press the return key. Magma is an interactive language, so your commands are executed immediately. Your screen should look like this:

```
> print 2 + 4;
6
```

where the 6 is the output from Magma. If it doesn't (apart from the spaces around the plus sign, which are optional), you have probably forgotten the semicolon at the end of the line. Type in a semicolon now, and then press the return key again.

Actually, the word **print** is optional in this case, so a shorter way to evaluate this expression and display the result is to type:

```
> 2 + 4;
6
```

This abbreviation will be used from now on.

In this manual, any line beginning with a prompt symbol should be typed by you (not including the ">"), and any line with no prompt symbol at the start is Magma's response to what you have typed. For instance:

```
> P<x> := PolynomialRing(IntegerRing());
> P;
Univariate Polynomial Algebra in x over Integer Ring
> (x^6 - 5*x^2 + 2) * (17*x^3 - 1);
17*x^9 - x^6 - 85*x^5 + 34*x^3 + 5*x^2 - 2
> Factorization(x^8 - 1);
[
    <x - 1, 1>,
    <x + 1, 1>,
    <x^2 + 1, 1>,
    <x^4 + 1, 1>
]
```

To quit from Magma, type control-d at the beginning of a line. (Hold down the 'control' key and type 'd'.)

1.2 Online Help

You can obtain help from the MAGMA system itself. One way to use it is to type `?` followed by the word for which you need help. For instance, to learn about **div**, type

```
> ?div
5 matches:
     1   0   /magma/ring-field-algebra
     2   I   /magma/ring-field-algebra/distributive-multivariate-polynomial/\
             operation-element/division/div
     3   I   /magma/ring-field-algebra/integer/operation-element/arithmetic/div
     4   I   /magma/ring-field-algebra/univariate-polynomial/\
             operation-element/division/div
     5   I   /magma/ring-field-algebra/valuation/element/arithmetic/div
To view an entry, type ? followed by the number next to it
```

These entries mean an Overview on rings etc, and the Intrinsic **div** as an element operation for multivariate polynomial rings, the integer ring, univariate polynomial rings, and valuation rings. (Here "magma" means "algebraic structure", as contrasted with help on the system or the language.) To read about **div** for univariate polynomials, say, do this:

```
> ?4
============================================================================
PATH: /magma/ring-field-algebra/univariate-polynomial/operation-element/\
      division/div
KIND: Intrinsic
============================================================================
f div g : RngUPolElt, RngUPolElt -> RngUPolElt

    The quotient q of f by g, where f and g are in the polynomial ring
    P=R[x].  Magma calculates the polynomials q (quotient) and r (remainder)
                [etc.]
```

Type `?` by itself to find out more about the online help system.

1.3 Printing the Value of an Expression

As you have seen from the examples above, the MAGMA command used to produce output is **print**, followed by the *expression* which you wish to be evaluated. In many situations (at the "top-level", i.e. not inside a function or procedure definition), the word **print** is optional; you can simply type the expression and a semicolon.

1.3.1 Arithmetic

The examples so far have displayed several of MAGMA's arithmetic operators: $+$, $-$, $*$ (multiplication), and `^` [or $\uparrow$] (exponentiation). The division operators depend on the kind of objects you want to divide. For elements of Euclidean rings, such as integers and polynomials, "division" means finding the quotient and remainder. The operators for these are **div** and **mod** respectively. For example, the output of

```
> 23 div 4, 23 mod 4;
5 3
```

means that 23 divided by 4 is 5 with a remainder of 3. (Note that if several expressions are to be printed, they should be separated by commas.)

For elements of fields, such as reals and rationals, use the slash (/) operator to compute exact division. For instance:

```
> (2/3) / (4 + 7/9);
6/43
> 4.7 / 2.6;
1.807692307692307692307692307692
```

In a large expression, MAGMA uses the usual precedence rules for the different operators. For instance, multiplication comes before addition. Operators with the same precedence (such as + and −) are evaluated left-to-right, except for exponentiation. You can always force a particular evaluation order with round parentheses:

```
> 2 + 4 * 9, (2 + 4) * 9;
38 54
```

1.3.2 Output from Functions

MAGMA contains many standard pre-defined functions and procedures, known as *intrinsics*, including several simple integer functions; you will find that almost all intrinsics begin with a capital letter. When you want to evaluate a function for a particular argument, type the function name followed by the argument in parentheses. For instance, to calculate 18! enter the statement

```
> Factorial(18);
6402373705728000
```

If the function has several arguments they must be separated by commas. To find the greatest common divisor of 15130 and 3162, you would type

```
> GCD(15130, 3162);
34
```

1.3.3 Printing Text

You can also use the **print**-statement to display text. The text must be enclosed in double quotation marks ("), which will be available as a single key on your keyboard. Text like this is called a *string*. For example:

```
> "Number of permutations of five objects is", Factorial(5);
Number of permutations of five objects is 120
```

1.4 Variables

It is often useful to store a MAGMA object (e.g. a number, a set, or a group) in a "box" in the computer's memory and give it a label. For instance, you might want to store the number 1492478 with the name `rqA5`. The way to tell MAGMA this is to type

```
> rqA5 := 1492478;
```

This instruction is called an *assignment statement*, and `rqA5` is called a *variable* or *identifier*. The computer can then look in the box and find out the value of `rqA5` any time it is needed. For instance:

```
> rqA5 * 10;
14924780
```

MAGMA regards upper and lower case letters in a variable name as different, and so `rqa5` or `Rqa5` would be different variables.

The expression on the right side of an assignment statement sometimes involves the variable on the left side. One common instance of this is

```
> rqA5 := rqA5 + 1;
```

The statement simply means to look up the value of `Rqa5`, add 1 to it, and store the result back in `Rqa5`. This is why the MAGMA assignment symbol is ":=", not "=". When you see ":=", think of it as meaning "becomes" or "has assigned to it", rather than "is equal to" in the mathematical sense.

To remind yourself what identifiers are currently defined, type **ShowIdentifiers();** on a line by itself. To see their current values as well, type **ShowValues();** .

1.5 Loading Input from a File

The examples of MAGMA code you have seen so far have been very short. As you come to write longer programs, you will find that it can be convenient to develop code within a text file exterior to MAGMA. You can then ask MAGMA to **load** the contents of the file into MAGMA:

load "filename";

Here "filename" is the name of the file. MAGMA will use this input as if you had typed it in from the keyboard.

1.6 Booleans

Operator	Usage	Meaning
eq	x **eq** y	**true** if x is equal to y, else **false**
ne	x **ne** y	**true** if x is not equal to y, else **false**
lt	x **lt** y	**true** if x is less than y, else **false**
le	x **le** y	**true** if x is less than or equal to y, else **false**
gt	x **gt** y	**true** if x is greater than y, else **false**
ge	x **ge** y	**true** if x is greater than or equal to y, else **false**
not	**not** a	**true** if a is **false**, else **false**
and	a **and** b	**true** if both a and b are **true**, else **false**
or	a **or** b	**false** if both a and b are **false**, else **true**
xor	a **xor** b	**true** if exactly one of a and b is **true**, else **false**

Table 1: Relational and logical operators

The symbol **gt** in the above example is a *relational operator*. It compares two quantities x and y, and returns an answer of **true** or **false**, depending on whether x is greater than y. The example also illustrated a *logical operator*, **and**. Logical operators act on *Boolean* values, that is, either **true** or **false**, and return a Boolean value. Some of MAGMA's relational and logical operators are described in table 1.

There are many Boolean functions in MAGMA. Their names are typically **Is** followed, with no intervening space, by the description of the property to be tested. For example:

```
> IsPrime(357);
false
```

1.7 Conditionals

MAGMA's **if**-statement is used when you want the computer to perform different actions depending on whether a certain condition is true or false. The following example calculates the area of the triangle, given a, b, c as the lengths of its sides, using the formula $Area = \sqrt{s(s-a)(s-b)(s-c)}$, where $s = (a+b+c)/2$. However, it checks to see if the triangle is non-degenerate, that is, if the sum of any two sides is greater than the third side. (The symbol **gt** means "Greater Than".) If the condition after **if** is true, MAGMA does the statements between **then** and **else**, and if it is false, it does the statements between **else** and **end if**; :

```
> a := 6; b := 8; c := 10;
> if ((a+b) gt c) and ((b+c) gt a) and ((c+a) gt b) then
>     s := (a+b+c)/2;
>     "Area is", Sqrt(s*(s-a)*(s-b)*(s-c)), "square units.";
> else
>     "Triangle is degenerate.";
> end if;
Area is 24.000000000000000000000000 square units.
```

With this input the computer gives the correct area, 24 units^2. However, if c were changed to 15, say, then it would print `Triangle is degenerate.` , because $6 + 8 < 15$.

There are other kinds of conditional statement. For instance, you can leave out the **else** clause, and you can make a chain of **if**-statements using **elif**.

2 Sets, Sequences, Cartesian Products and Records

Sets and sequences are both collections of objects which are all in the same algebraic structure (the *universe*). A set is unordered, so an element can be in a set at most once. However, a sequence is ordered, so repetition is possible. Sets are bracketed with { } and sequences with [ ]. For example:

```
> t := { (-11)^2, (-7)^2, (-5)^2, (-3)^2, 3^2, 5^2, 7^2, 11^2 };
> q := [ (-11)^2, (-7)^2, (-5)^2, (-3)^2, 3^2, 5^2, 7^2, 11^2 ];
> t, q;
{ 9, 25, 49, 121 }
[ 121, 49, 25, 9, 9, 25, 49, 121 ]
```

The i^{th} entry of a sequence Q is $Q[i]$:

```
> q[10] := q[5] - 4*q[1]; q[2] := 1000;
> q;
[ 121, 1000, 25, 9, 9, 25, 49, 121, undef, -475 ]
```

MAGMA has special constructors for sets and sequences. The expression

$\{i..j \textbf{ by } k\}$ or $[i..j \textbf{ by } k]$

means the arithmetic progression $i, i+k, i+2k, \ldots, j$ as a set or sequence of integers, and

{ expression in x : x **in** D | condition } or [expression in x : x **in** D | condition]

means the set or sequence of the values of "expression in x" evaluated for all x in D such that the Boolean condition is **true**. For example, you could have created t like this:

```
> t := { n^2 : n in [-11..11 by 2] | IsPrime(n) };
```

The symbol & followed by a binary operator and a set or sequence S combines the elements of S using that operator. For instance, the following command prints $\sum_{i=1}^{10}(i!)^2$:

```
> &+[ Factorial(i) ^ 2 : i in [1..10] ];
13301522971817
```

To collect data of various kinds together, MAGMA has Cartesian products (and records, which are not discussed here). Elements of Cartesian products (*tuples*) have the form < $a_1, a_2, \ldots, a_t$ >. The factorization output on page 3 is a sequence of tuples. Another example is:

```
> { <a, b, c> : a, b, c in [1..10] | a^2 + b^2 eq c^2 };
{ <6, 8, 10>, <4, 3, 5>, <3, 4, 5>, <8, 6, 10> }
```

3 Iterative Statements (Loops)

MAGMA's iterative statements resemble those in other programming languages. They are:

- **while** Boolean expression **do**
  statements
  end while;
- **repeat**
  statements
  until Boolean expression;
- **for** variable **in** domain **do**
  statements
  end for;

MAGMA normally gives a special prompt symbol when you are in the middle of an iterative statement or other compound statement. For example:

```
> for g in DihedralGroup(8) do
for> if Order(g) eq 4 then
for|if> g;
for|if> end if;
for> end for;
(1, 3, 5, 7)(2, 4, 6, 8)
(1, 7, 5, 3)(2, 8, 6, 4)
```

4 User-Defined Functions and Procedures

Functions are first-class objects in MAGMA, so you can assign them to identifiers like any other object. There are two ways to define your own function in MAGMA. If the return value can be expressed simply in terms of the inputs, then use the **func** constructor. For instance, the function $f(n, q) = \prod_{i=1}^{n} q^n - q^{i-1}$ can be defined as follows:

```
> f := func< n, q | &*[q^n - q^(i-1) : i in [1..n]] >;
> f(5, 3);
475566474240
```

If you need several steps to compute the return value (or values), then use the longer form:

```
> counting := function(n, r)
>     x := Factorial(n);
>     y := Factorial(r);
>     z := Factorial(n-r);
>     p := x div z;    // integer division will be exact here
>     c := p div y;    // ditto
>     return p, c;
> end function;
> per, com := counting(5, 2);
> per, com;
20 10
```

Procedures are rather like functions, but they do not have return values. The following simple procedure prints n copies of the character c together in a line. *Since the printing is happening inside a procedure definition, you must use the word* **print**. To call the procedure, you put its name in a statement by itself, and supply values for c and n:

```
> SeparatingLine := procedure(c, n)
procedure> print c^n;
procedure> end procedure;
> SeparatingLine("#", 35);
###################################
```

A procedure can also modify a variable "in place", if you put a ~ tilde symbol in front of the variable. For instance, the procedure *SelectIntegers*($\tilde{}Q, b$) has a formal reference argument Q and a formal value argument b. Given a sequence Q of rational numbers and a Boolean value b, it removes from Q all the non-integral entries if b is **true**, and all the integral entries if b is **false**:

```
> SelectIntegers := procedure(~Q, b)
procedure>     if b then
procedure|if>         Q := [Q[i]: i in [1..#Q] | IsIntegral(Q[i])];
procedure|if>         return;
procedure|if>     end if;
procedure>     Q := [Q[i]: i in [1..#Q] | not IsIntegral(Q[i])];
procedure> end procedure;
> w := [4, 7/2, 1, 7, 8, 2/9];
> SelectIntegers(~w, true);
> w;
[ 4, 1, 7, 8 ]
```

Here the command **return** tells MAGMA to stop executing the procedure call immediately, without progressing through the rest of the procedure body down to **end procedure**.

5 Overview of Groups

MAGMA supports five kinds of groups: abelian, finitely-presented (fp), matrix, permutation, and soluble groups in power-conjugate/polycyclic (pc) representation. Here are examples of how to construct each of them:

```
> G<a,b> := AbelianGroup< a,b | 6*a, 14*b >; G;
Abelian Group isomorphic to Z/2 + Z/42
Defined on 2 generators
Relations:
    6*a = 0
    14*b = 0

> G<a,b,c> := Group< a,b,c | b*c = a^9, c^7 >; G;
Finitely presented group G on 3 generators
Relations
    b * c = a^9
    c^7 = Id(G)

> G<a,b> := MatrixGroup< 2, FiniteField(5) | [0,1,2,4], [1,1,0,3] >; G;
MatrixGroup(2, GF(5))
Generators:
    [0 1]
    [2 4]

    [1 1]
    [0 3]

> G<a,b,c,d,e> := PolycyclicGroup< a,b,c,d,e |
>     a^2 = c, b^2, c^2 = e, d^5, e^2, b^a = b*e >; G;
GrpPC : G of order 80 = 2^4 * 5
PC-Relations:
a^2 = c,
b^2 = Id(G),
c^2 = e,
d^5 = Id(G),
e^2 = Id(G),
b^a = b * e

> G<a,b,c> := PermutationGroup< 10 | (1,3,5,7,9), (2,3,4), (1,10) >; G;
Permutation group G acting on a set of cardinality 10
    (1, 3, 5, 7, 9)
    (2, 3, 4)
    (1, 10)
```

Other ways to construct groups include the **sub**, **quo**, and **ncl** constructors, and functions such as **Sym**, **Alt** and **DirectProduct**. Table 2 (page 11) lists a few of the many functions on groups.

6 Overview of Rings, Fields, and Algebras

MAGMA's rings, fields and algebras include **Z** and its residue classes, polynomial rings and their residue classes, power and Laurent series rings, valuation rings, finitely-presented algebras, matrix

MAGMA	Meaning
IsSoluble(G)	**true** if G is soluble
IsNormal(G, H)	**true** if H is normal in G
SylowSubgroup(G, p)	Sylow p-subgroup of G
H **meet** K	intersection of H and K
$H\hat{}u$	subgroup obtained by conjugating H by u
$H\hat{}G$	normal closure of H in G
Core(G, H)	maximal normal subgroup of G contained in the subgroup H
Centralizer(G, x)	$\{g \in G : x^g = x\}$
Normalizer(G, H)	$\{g \in G : H^g = H\}$
Index(G, H)	number of cosets of H in G
Transversal(G, H)	returns (i) indexed set of right coset representatives for H in G (ii) mapping $G \to H$ given by $x \mapsto t$ where $x \in Ht, t \in T$
CosetAction(G, H)	permutation representation L of G given by action of G on right cosets of H; returns (i) natural homomorphism $G \to L$ (ii) induced group L (iii) kernel of the action

Table 2: Some functions on groups

algebras, **Q**, **R**, **C**, finite fields, and number fields (including quadratic and cyclotomic). Some other examples are below:

```
> d, p, q := XGCD(21, 27); d, p, q; d eq (21*p + 27*q);
3 4 -3
true

> PQ<x> := PowerSeriesRing(RationalField());
> C := (1 - Sqrt(1-4*x)) / (2*x); Coefficients(C);
[ 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012,
742900, 2674440, 9694845, 35357670, 129644790, 477638700 ]

> P3<x> := PolynomialRing(GF(3)); f := x^4 + 2*x^3 + 2;
> R<y> := quo<P3 | f>;
> (y^2 + 3) ^ -1;
y^2 + 2*y

> K3<y> := FieldOfFractions(P3);
> (y^2 - 1) / (y^10 - 1);
1 / (y^8 + y^6 + y^4 + y^2 + 1)

> M := MatrixAlgebra(P3, 5); M;
Full Matrix Algebra of degree 5
over Univariate Polynomial Algebra in x over GF(3)
> M ! [x^i + x^j: i, j in [1..5]];
[      2*x   x^2 + x   x^3 + x   x^4 + x   x^5 + x]
[  x^2 + x     2*x^2 x^3 + x^2 x^4 + x^2 x^5 + x^2]
[  x^3 + x x^3 + x^2     2*x^3 x^4 + x^3 x^5 + x^3]
[  x^4 + x x^4 + x^2 x^4 + x^3     2*x^4 x^5 + x^4]
[  x^5 + x x^5 + x^2 x^5 + x^3 x^5 + x^4     2*x^5]

> C<i> := ComplexField();
```

```
> PolarToComplex(1, 2 * Pi(C) / 7);
0.623489801858733530525004884000 + 0.781831482468029808708444526665775860168*i
> Q7<sev> := CyclotomicField(7);
> C!sev;
0.623489801858733530525004884000 + 0.781831482468029808708444526665775860168*i

> PQ<x> := PolynomialRing(RationalField());
> IsIrreducible(x^3 - 2);
true
> N<w> := NumberField(x^3 - 2);
> PN<X> := PolynomialRing(N);
> Factorization(X^3 - 2);
[
    <X - w, 1>,
    <X^2 + w*X + w^2, 1>
]
> NN<z> := ext< N | X^2 + w*X + w^2 >; NN;
NumberField(x^6 + 108)
> NN ! w;
1/18*z^4
> PNN<XX> := PolynomialRing(NN);
> Factorization(XX^3 - 2);
[
    <XX - 1/18*z^4, 1>,
    <XX + 1/36*z^4 - 1/2*z, 1>,
    <XX + 1/36*z^4 + 1/2*z, 1>
]
```

Other ways to construct rings, fields and algebras include the **sub**, **quo**, and **ideal** constructors. For left and right ideals of non-commutative rings there are also **lideal** and **rideal**.