Magma Tutorial ~~for pairing cryptographers~~. Part I - Introduction

Luis J Dominguez Perez

Cinvestav, Mexico

September, 2011

http://www.computing.dcu.ie/~ldominguez/module6/pairing-tutorial.pdf

Table of contents

Magma

Luis J Dominguez Perez

The Magma shell

- The typical way of running magma is using its interactive shell, which behaves similarly to those of Python, Perl or Sage.
- Magma does not provides a graphical interface, however, it is possible to integrate it into Sage to produce graphics.
- The command shell symbol is $>$, and the command delimiter is ;.
- Since we are using a delimiter, we can have several commands in the same line.

The Magma program

- We can have several Magma copies of the program at the same time.
- Magma uses one and only one core per copy of the program.
- Each copy of Magma runs in a non-intrusive environment. (We can run 3 copies of Magma in a Quad core and still have a responsible system, which is useful for running a test with several set of inputs).

Operators

Arithmetic Operators

- Assignment $:=$
- $+,-,*,/,\hat{},$ mod, div, cat, etc.
- $+:=,-:=,*:=$

Boolean operators

- eq, ne, not, and, or, in

For the Binary Operations, I convert the number into a string sequence of the bits.

Hands-on

Open magma and do the following exercise:

- $x \leftarrow 2$
- $y \leftarrow x$
- $x \leftarrow z$
- $z \leftarrow x^2$
- $a \leftarrow 1/2$
- $b \leftarrow a^{-1}$

Use “variable”; to display its value.

Display

More printing

- printf "A=%o\n",x; //as in c/c++
- Sprintf("A=%o\n",x);
- PrintFile("MyFile",Sprintf("%o,%o\n",3,5));

Use "%h" to display values in hexadecimal.

Loops

Luis J Dominguez Perez

For

- for i:=1 to 10 do ... end for;
- for i in [1..9] do ... end for;
- for i:= 10 to 1 by -1 do ... end for;

While

- while i lt 10 do ... end while;

Repeat until

- repeat ... until i lt 10;

Conditionals

if

- if i eq 10 then ... end if;
- if i eq true then ... else ... end if;
- if i eq 1 then ... elif i eq 0 then ... else ... end if;

switch

- case a: when: ... else: ... end case;

File

- L:=**Open**("NOTICE","r");
- while true do
  - s:=**Gets**(L);
  - if **IsEof**(s) then break; end if;
  - print s;
- end while;
- **Flush**(L);

Magma will do the cleaning, but it is always better to explicitly close a file (specially when writing in it).

Hands-on

Exercise:

- Create a file with the multiplication tables.

Sets and Sequences

The difference:

```
i:={IntegerRing() | 1,2,3}; i;
i:={IntegerRing() | 1,3,5}; i;

i:=[IntegerRing() | 1,5,3];
```

Autofilling it:

```
T := [ Integers() | x^2+x+1 :
    x in { -3 .. 2 by 1} ];
```

Accessing elements:

- a[1][2];
- a[1,2];

both are OK

More on sets and sequences

New operators:

- join
- meet
- cat

Modifying the set

- Append, Insert, Include, Exclude
- Prune, Remove
- Sort, Reverse, Rotate

Getting information

- Maximum, Minimum, #, Random, Index, Parent, Universe, Category, etc.

Matrices

Luis J Dominguez Perez

Generating a matrix:

```
Matrix(IntegerRing(), 2, 2, [0,0,0,0]);

Matrix(RationalField(), 5, 10, [<1,2,23>,
  <3,7,11>, <5,10,-1>]);

Matrix(IntegerRing(), 10, 10, [<2*i-1, 2*j-1, i*j>:
  i, j in [1..5]]);
```

Matrices II

Generation shorcuts:

- ZeroMatrix(Ring,m,n)
- DiagonalMatrix(Ring,n,Sequence)
- ScalarMatrix(n,value)
- SymmetricMatrix(Sequence)

Operators:

- NumberOfRows
- NumberOfColumns

Hands-on

Exercise:

- Create a file with the multiplication tables. (using matrices)

Functions

Two ways to declare a function:

- `f := function`
- `function f`

Both end with `end function;`

There's a difference though, one may need to use $$ to write a recursive function.

Actually, there's a third one:

- `f := func< x| x^2>`

Procedure

The same principle applies for the procedure, exempt that:

- It does not `return` statements
- It supports parameters as reference (∼a)

Optionally, we can forward a definition of a procedure with `forward f;`

Hands-on

Exercise:

- Create a function and a procedure to get the multiplication tables.

Package I

A package, is a function or procedure which will be compiled by Magma at loading time.

A package is much more faster than a regular function or procedure, since it requires the user to specify the data-types of the arguments.

We “Attach” or “Detach” at runtime the file containing our package.

Package II

The syntaxis is as follows:

```
intrinsic NAME(ARG-LIST) [ -> RET-LIST ]
{ COMMENT-TEXT }
statements
end intrinsic;
```

For example:

```
intrinsic myGCD(x::RngIntElt, y::RngIntElt)
-> RngIntElt
{ Return the GCD of x and y }
   return ...;
end intrinsic;
```

Please note that the documentation **is mandatory**

Associative Array

An associative array is a type of array with a named index. Useful for look up tables.

- AssociativeArray
- Remove
- Keys
- IsDefined

Hands-on

Exercise:

- Create a package with the multiplication tables as a lookup

Prime numbers

Generating a prime number

- NextPrime
- PreviousPrime
- NthPrime
- RandomPrime

Primality test:

- IsPrime
- IsProbablePrime
- IsPrimePower
- Factorisation

Hands-on

Exercises:

(write down a function to)

- Determine if a number is *almost* prime
- Compute the MCM of two numbers
- Compute the mcm of two numbers
- Compute the Euler totient function

More hands-on

Exercises:

- Toy-example of RSA

Verify:
$(a^e)^d \equiv a \bmod n$, $(a^d)^e \equiv a \bmod n$, and $a^{ed} \equiv a \bmod n$.
for any random a.

Solution

- RandomPrime(100)
- Setup e
- $d1 \leftarrow$InverseMod(e,p-1)
- $d2 \leftarrow$InverseMod(e,q-1)
- GCD(p-1,q-1)
- TrialDivision(p-1) for common factors
- d1 mod common factor
- d := CRT([d1,d2],[p-1,(q-1) div common factor]) or
- d := InverseMod(e,LCM(p-1,q-1));

Error Support

When magma encounters a runtime error, it stops the execution of the program; if the program was running for a long period of time, then this is catastrophic.

```
procedure always_fails(x)
  error Error(x);
end procedure;
try
  always_fails(1);
catch e
 error "Error",e`Object;
end try;
```

After catching a runtime error, Magma continues the execution of the program.

End

End of Part I

There's part II