

# GAP Tutorial for the PREP workshop: Abstract Algebra with GAP, July 2003

This tutorial was written by Russell Blyth and Julianne Rainbolt. The intent of this tutorial is to provide a brief introduction to the software GAP and thus prepare participants for the PREP workshop “Abstract Algebra with GAP”. If you do not have a version of GAP already installed you need to download a version from:

`http://www-gap.dcs.st-and.ac.uk/~gap`

An extensive reference manual and tutorial for the software GAP are also available at this web site.

## Some Commands That You Will Use Often

- 1) To exit from GAP type `quit`;
- 2) To save your work in a file type `LogTo("filename");`
- 3) To stop saving your work in that file type `LogTo()`;
- 4) Type `<ctl>-p` to redisplay the previous commands.
- 5) If an error causes GAP to enter a loop (and gives you the prompt `brk>` ) you can exit the loop by typing `quit`;
- 6) Type `?` and then a subject name to get help about that subject.

The `?` command, listed as (6) above, is particularly useful. This can be used whenever you forget the exact command for something or when you wonder if GAP has a command for a particular operation. For example, suppose you wonder how to denote multiplication in GAP. Type

```
gap> ?multiplication
```

GAP returns a description of how to enter basic arithmetic commands.

**Careful:** GAP distinguishes between upper and lower case letters. (So, for example, `LogTo` is not the same as `Logto`.)

**Careful:** Always put a semicolon at the end of a command. Two semicolons at the end of a command will cause GAP to execute the command but not list the output of the command.

## Getting Use to GAP

We will now do some exercises to get you comfortable with GAP.

Open the software GAP. At the gap prompt type `LogTo("tutorial1");` This will save the following work in a file called "tutorial1". At the gap prompt type `(5 + 3) * 9;` then hit the enter key. Your screen should look like this:

```
gap> LogTo("tutorial1");
gap> (5 + 3) * 9;
72
```

At the prompt type `(5 + 3) * 9` (without the semicolon) and then hit enter.

```
gap> (5 + 3) * 9
>
```

Notice nothing happens. GAP does not know you have finished the command because there is no semicolon. Now type `;` and then hit return. You should now get the 72.

```
gap> (5 + 3) * 9
>;
72
gap>
```

Now type `(5 + 3 * 9;` and hit enter. Your screen should look like

```
gap> (5 + 3 * 9;
Syntax error: ) expected
```

An error message occurred because the number of left parentheses and right parentheses does not match. Type `<ct1>-p` to redisplay the previous command. Then use your arrow keys to insert the needed parenthesis.

```
gap> (5 + 3 * 9;
Syntax error: ) expected
```

```
gap> (5 + 3) * 9;
72
gap>
```

In general, basic arithmetic commands are entered just as they would be on a calculator. The only difference is you need to include a semicolon at the end of the command. **GAP** can also be used to test the equality of two values. At the **GAP** prompt type `6=9;` and then hit enter.

```
gap> 6=9;
false
```

**GAP** returned the value of `false` since 6 is not equal to 9. Now use **GAP** to complete the following exercises. Don't forget to put a semicolon at the end of a command.

1. Compute  $3^{121}$ .
2. Determine if  $2^{25} + (45 * 51,777)$  is greater than 34 million.

You can assign values to variables in **GAP** by using `:=`. This allows you to refer to an object with a name. In the following example `a` is an identifier.

```
gap> a := (10 + 7) * (9 - 6);
51
gap> a;
51
gap> a * (a-1);
2250
gap> a:=14;;
gap> a * (a-1);
182
```

Notice when an identifier is assigned a value that value is echoed on the next line. Because two semicolons occurred in the line `a:= 14;;` the value of `a` was not echoed. Almost any sequence of letters and digits containing at least one letter can be used as an identifier. Use **GAP** to complete the following exercise.

3. Assign the value 4 to the identifier `b`. Then assign the value 522456 to the identifier `bignumber`. Have GAP compute `b + bignumber`.

Now we will print off these first three exercises. At the gap prompt type `LogTo()`; . This causes GAP to stop saving your work to the file “tutorial1”. Exit GAP by typing `quit`; at the gap prompt. Then print the file “tutorial1”.

## Properties of Integers

The software GAP contains many predefined functions. Functions in GAP begin with a capital letter. For example, the function `Gcd` gives the *-r--t-st* --- - - - *-v-s-r* of two nonzero integers. The function `Lcm` gives the *--st* --- - - - *-u-t---* of two nonzero integers. Examples:

```
gap> Gcd(123, 456);
3
gap> Lcm(123,456);
18696
```

The function `Gcdex` provides the numbers needed to write the gcd of  $a$  and  $b$  as a linear combination of  $a$  and  $b$ . An example:

```
gap> Gcdex(4,15);
rec( gcd := 1, coeff1 := 4, coeff2 := -1, coeff3 := -15,
coeff4 := 4)
```

The above output tells us that the gcd of 4 and 15 is 1 and that  $\text{gcd}(4, 15) = 4 * 4 + (-1) * 15$ . That is, `coeff1` and `coeff2` are the  $s$  and  $t$  needed to write  $\text{gcd}(a, b) = as + bt$ . (The output `coeff3` and `coeff4` are integers  $m$  and  $n$  such that  $0 = am + bn$ .)

Before starting the following exercises type, at the gap prompt, `LogTo("tutorial2");`. This will save your work in a file named “tutorial2”. When you are done with the exercises type `LogTo()`; and print “tutorial2”. Don’t forget functions begin with capital letters.

4. Compute the gcd of 8701 and 10057.
5. Compute the lcm of 25 and 80.

6. Use `Gcdex` to find integers  $s$  and  $t$  so that  $\gcd(8701, 10057) = 8701s + 10057t$ .

## Modular Arithmetic and Functions

GAP can perform modular arithmetic. For example

```
gap> 23 mod 6;  
5
```

**Careful:** Blank spaces are needed around `mod`. (`23mod6` will be viewed as an identifier by GAP.)

You can create your own functions within GAP. One way to do this is to use the `--s-t-` operator `->`. This is a minus sign and a greater than sign with no blank space between. The following is an example of creating a function called `square` which takes a number and squares it.

```
gap> square:=x -> x^2;  
function( x ) ... end  
gap>
```

Now we can use this function:

```
gap> square(2);  
4  
gap> square(5);  
25
```

Use GAP to do the following exercises. Save your work in a file.

7. Determine which, if any of the following numbers are congruent to 3 mod 7: 17, 19, 101, 121.

8. Define a function `SumFirstnInt` which takes as input a positive integer  $n$  and outputs the sum  $1 + 2 + 3 + \dots + n$ . Then use this function to find this sum for  $n = 100$  and  $n = 987$ . (Hint: Use the fact that the sum of the first  $n$  positive integers equals  $n * (n + 1) / 2$ .)

## Predefined Groups Example - Dihedral Groups

There are many ways to define groups in GAP. Some groups are already set up in GAP. For example the dihedral group of order  $2n$ , described as a permutation group, is obtained by typing `DihedralGroup(IsPermGroup,2n)`. For example to get  $D_4$ , the dihedral group of order 8, we type:

```
gap> d4:= DihedralGroup(IsPermGroup,8);
      Group([ (1,2,3,4), (2,4) ])
gap> Elements(d4);
      [ (), (2,4), (1,2)(3,4), (1,2,3,4), (1,3), (1,3)(2,4),
        (1,4,3,2), (1,4)(2,3) ]
```

The command `Elements` lists the elements in the group. The identity is denoted by `()`. The command `Size` gives the number of elements in the group (that is, the order of the group).

```
gap> Size(d4);
      8
```

Elements can be multiplied:

```
gap> (1,4)(2,3)*(2,4);
      (1,2,3,4)
```

**Note:** GAP multiplies these permutations from left to right.

The command `DihedralGroup(2n)` also creates the dihedral group of order  $2n$  by listing the elements in terms of a collection of generators:

```
gap> e4:= DihedralGroup(8);
      <pc group of size 8 with 3 generators>
gap> Elements(e4);
      [ <identity> of ..., f1, f2, f3, f1*f2, f1*f3, f2*f3, f1*f2*f3 ]
```

9. Using GAP take a reflection in  $D_5$  and multiply it by a reflection in  $D_5$ . What rotation do you get?

We can also define subgroups of a given group. The following creates the subgroup of  $D_8$  generated by the permutation  $(1,2,3,4,5,6,7,8)$ :

```
gap> G:= DihedralGroup(IsPermGroup,16);
Group([ (1,2,3,4,5,6,7,8), (1,8)(2,7)(3,6)(4,5) ])
gap> Elements(G);
[( ), (2,8)(3,7)(4,6), (1,2)(3,8)(4,7)(5,6), (1,2,3,4,5,6,7,8),
(1,3)(4,8)(5,7), (1,3,5,7)(2,4,6,8), (1,4)(2,3)(5,8)(6,7),
(1,4,7,2,5,8,3,6), (1,5)(2,4)(6,8), (1,5)(2,6)(3,7)(4,8),
(1,6)(2,5)(3,4)(7,8), (1,6,3,8,5,2,7,4), (1,7)(2,6)(3,5),
(1,7,5,3)(2,8,6,4), (1,8,7,6,5,4,3,2), (1,8)(2,7)(3,6)(4,5) ]
gap> a:= (1,2,3,4,5,6,7,8);
(1,2,3,4,5,6,7,8)
gap> b:= (1,8)(2,7)(3,6)(4,5);;
gap> H:= Subgroup(G, [a]);
Group([ (1,2,3,4,5,6,7,8) ])
gap> Elements(H);
[( ), (1,2,3,4,5,6,7,8), (1,3,5,7)(2,4,6,8), (1,4,7,2,5,8,3,6),
(1,5)(2,6)(3,7)(4,8), (1,6,3,8,5,2,7,4), (1,7,5,3)(2,8,6,4),
(1,8,7,6,5,4,3,2)]
```

The first command above assigns the name  $G$  to the dihedral group of order 16 (the group of symmetries of a regular 8-gon). The next two commands assign the names  $a$  and  $b$  to two elements in  $G$ . The next command assigns the name  $H$  to the cyclic subgroup of  $G$  generated by  $a$ . Notice that  $H$  is a subgroup of  $G$  of order 8.

```
gap> K:= Subgroup(G, [a,b]);
Group([ (1,2,3,4,5,6,7,8), (1,8)(2,7)(3,6)(4,5) ])
gap> Size(K);
16
gap> c:= (1,5)(2,6)(3,7)(4,8);
(1,5)(2,6)(3,7)(4,8)
gap> L:= Subgroup(G, [a,c]);
Group([ (1,2,3,4,5,6,7,8), (1,5)(2,6)(3,7)(4,8) ])
gap> Size(L);
8
```

This first command above assigns the name  $K$  to the subgroup of  $G$  generated by  $a$  and  $b$ . Notice that after this command `GAP` echoes the generators of the subgroup. By typing in `Elements(K)` or `Size(K)` we can see that  $K = G$ . Notice that the subgroup  $L$  is a proper subgroup of  $G$ .

The following is a list of some other group theory commands that you might find useful.

- 1) The command to find the center of the group  $G$  is `Center(G)`.
- 2) The command to find the centralizer of an element  $a$  in a group  $G$  is `Centralizer(G,a)`.
- 3) The command to find the order of an element  $a$  in a group  $G$  is `Order(a)`.
- 4) The command `IsAbelian(G)` tells you whether or not the group  $G$  is Abelian.
- 5) The command `IsCyclic(G)` tells you whether or not the group  $G$  is cyclic.

**\*At this point we suggest you play around with these commands in GAP.\***

### Other Predefined Groups

There are many other predefined groups in `GAP`. Several that we will be using are:

- 1) `Integers mod n` denotes in `GAP` the group of integers mod  $n$ . (You have to specify  $n$ .) The spaces before and after `mod` are needed here.
- 2) `CyclicGroup(IsPermGroup, n)`; denotes the cyclic group of order  $n$  as a permutation group. (Again you have to specify the  $n$ .)
- 3) `SymmetricGroup(n)`; denotes the symmetric group  $S_n$ .
- 4) `AlternatingGroup(6)`; denotes the alternating group  $A_n$ .

## Generators and Relations

Groups defined using generators and relations can be easily created in GAP. If you want to create an  $n$ -generated group, start with a free group on  $n$  generators. Then create the group by “moding out by” the relations. For example,  $D_4$  is a 2-generated group with relations  $a^4 = b^2 = (ab)^2 = e$ , where  $a$  and  $b$  are the generators. The following creates the group  $D_4$  in GAP using generators and relations:

```
gap> f:=FreeGroup(2);
<free group on the generators [ f1, f2 ]>
gap> d4:=f/[f.1^4, f.2^2, (f.1*f.2)^2];
<fp group on the generators [ f1, f2 ]>
gap> Elements(d4);
[ <identity ...>, f2, f1^3*f2, f1, f1^2*f2, f1^2, f1^3, f1*f2 ]
```

The first command above creates a free group with two generators. The element `f.1` denotes the first generator and `f.2` denotes the second generator in the free group `f`:

```
gap> f.1;
f1
gap> f.2;
f2
```

The line:

```
gap> d4:=f/[f.1^4, f.2^2, (f.1*f.2)^2];
```

creates  $D_4$  as the free group on two generators mod the relations  $f.1^4 = f.2^2 = (f.1 * f.2)^2 = e$ , the identity.

We can now use the group theory GAP commands discussed above on `d4`. For example:

```
gap> IsAbelian(d4);
false
gap> Size(d4);
8
gap> Center(d4);
Group([ f1^2 ])
```

The `Factorization` command can be used to express the image of a word in the free group as an element in the factor group:

```
gap> a:=d4.1;;
gap> b:=d4.2;;
gap> Factorization(d4,a*b*a^3*b*a^5);
x1^3
```

The first two commands assign the letters  $a$  and  $b$  to the image of the two generators of the free group. The `Factorization` output tells us that  $aba^3ba^5$  reduces to  $a^3$  in  $D_4$ . (GAP will denote the  $i$ th generator of the factor group by `xi`.) The `Factorization` command does not work for all groups.

10. Let  $G = \langle a, b \mid a^8 = b^2 = e, baba^3 = e \rangle$ .
- Use GAP to find  $|G|$ .
  - Find the order of  $ab$ .
  - Find the center of  $G$ .

## Vectors and Matrices in GAP

A vector in GAP is entered by listing the components within square brackets. For example the vector  $v = (1, 3, 7)$  is entered as:

```
gap> v:= [1,3,7];
[ 1, 3, 7 ]
```

A matrix can be entered as a list of row vectors. For example, the matrix

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

is entered into GAP by typing

```
gap> M:= [ [1,2,3], [4,5,6], [7,8,9] ];
[ [ 1, 2, 3 ], [ 4, 5, 6 ], [ 7, 8, 9 ] ]
```

If you prefer to exhibit the matrix  $M$  as a 3 by 3 array use the command `PrintArray`:

```
gap> PrintArray(M);
[ [ 1, 2, 3 ],
  [ 4, 5, 6 ],
  [ 7, 8, 9 ] ]
```

The notation  $M[i][j]$  denotes the entry in the  $i$ th row and  $j$ th column of  $M$ . Similarly, the notation  $M[i]$  denotes the  $i$ th row of  $M$ .

```
gap> M[2][3];
6
gap> M[1];
[ 1, 2, 3 ]
```

To multiply  $v$  and  $M$  type:

```
gap> v*M;
[ 62, 73, 84 ]
gap> M*v;
[ 28, 61, 94 ]
```

That is,  $vM = (62, 73, 84)$  and  $Mv = (28, 61, 94)$ . Notice that GAP automatically treats  $v$  as a row vector in the multiplication  $vM$  but as a column vector in the multiplication  $Mv$ .

11. Create a 4 by 4 matrix  $B$  and a 4-tuple  $u$ , then multiply  $Bu$  and  $uB$ .

## Finite Fields

GAP has a function  $Z$  which produces a generator for the multiplicative group of nonzero elements of a finite field. The following creates the multiplicative group of nonzero elements in the fields  $\mathbf{Z}_5$  and  $\mathbf{Z}_{25}$ .

```
gap> g:=Group([Z(5)]);
<group with 1 generators>
gap> Elements(g);
[ Z(5)^0, Z(5), Z(5)^2, Z(5)^3 ]
gap> g2:=Group([Z(5^2)]);
<group with 1 generators>
gap> Elements(g2);
[ Z(5)^0, Z(5), Z(5)^2, Z(5)^3, Z(5^2), Z(5^2)^2, Z(5^2)^3,
```

$Z(5^2)^4, Z(5^2)^5, Z(5^2)^7, Z(5^2)^8, Z(5^2)^9, Z(5^2)^{10},$   
 $Z(5^2)^{11}, Z(5^2)^{13}, Z(5^2)^{14}, Z(5^2)^{15}, Z(5^2)^{16}, Z(5^2)^{17},$   
 $Z(5^2)^{19}, Z(5^2)^{20}, Z(5^2)^{21}, Z(5^2)^{22}, Z(5^2)^{23}$  ]

## Matrix Groups

We can now define groups of matrices over  $\mathbf{Z}_{p^n}$ . The following GAP commands create the group  $g$  which is the matrix group defined over  $\mathbf{Z}_5$  generated

by  $a = \begin{bmatrix} 1 & 1 & 2 \\ 2 & 0 & 2 \\ 0 & 2 & 0 \end{bmatrix}$  and  $b = \begin{bmatrix} 2 & 2 & 1 \\ 2 & 0 & 2 \\ 1 & 0 & 2 \end{bmatrix}$ . (Here the entries of  $a$  and  $b$  are viewed mod 5 and we are taking 2 mod 5 as the generator of the multiplicative group of nonzero elements in  $\mathbf{Z}_5$ .)

```

gap> z:= Z(5);;
gap> a:=[[z^0,z^0,z],[z,0*z,z],[0*z,z,0*z]];
      [[ Z(5)^0, Z(5)^0, Z(5) ], [ Z(5), 0*Z(5), Z(5) ],
      [ 0*Z(5), Z(5), 0*Z(5) ] ]
gap> b:=[[z,z,z^0],[z,0*z,z],[z^0,0*z,z]];
      [[ Z(5), Z(5), Z(5)^0 ], [ Z(5), 0*Z(5), Z(5) ],
      [ Z(5)^0, 0*Z(5), Z(5) ] ]
gap> g:= Group(a,b);
      Group([ [ [ Z(5)^0, Z(5)^0, Z(5) ], [ Z(5), 0*Z(5), Z(5) ],
      [ 0*Z(5), Z(5), 0*Z(5) ] ], [ [ Z(5), Z(5), Z(5)^0 ],
      [ Z(5), 0*Z(5), Z(5) ], [ Z(5)^0, 0*Z(5), Z(5) ] ] ] )

```

**Careful:** The nonzero elements of  $\mathbf{Z}(p^n)$  are denoted as powers of  $Z(p^n)$ . (Thus, for example, 1 mod 5 above is denoted by  $Z(5)^0$ .) Also, 0 mod  $p^n$  is denoted by  $0* Z(p^n)$ , not just 0.

**Careful:** The last command above echoes only the generators of the group (not all the elements). Don't ask GAP to list all the elements of the group  $g$ ; it is too big:

```

gap> Size(g);
744000

```

12. Create a matrix group with three generators over  $\mathbf{Z}_7$ . Find the order of your group and the order of its center.

This completes the tutorial.