

An Introduction to GAP

Alice Niemeyer
The University of Western Australia

September 2004

What is GAP?

GAP is a free Computer System for
Computational Discrete Algebra.

What does GAP mean?

GAP stands for

Groups **A**lgorithms and **P**rogramming.

Who uses GAP?

- students taking a first course in Algebra.
- honours, postgraduate students.
- lecturers teaching algebra units.
- researchers.
- probably others.

What is GAP (in a little more detail)?

- a programming language called GAP,
- a collection of programs written in GAP,
- libraries of data,
- contributed packages of programs,
- an extensive collection of documentation,
- programming tools.

Let's start

computer>

Let's start

computer> gap

First Steps

gap> ← is called the GAP prompt

First Steps

gap> GAP is waiting for us to type something

First Steps

```
gap> 1 + 1
```

First Steps

```
gap> 1 + 1
```

> GAP is waiting for us to tell it that we are finished

First Steps

```
gap> 1 + 1  
> ;  
gap> 2
```

Expressions

Expressions

When GAP displays the prompt `gap>` it waits for an expression which finishes with a `;`

Response

GAP prints the result of the expression.

Integers and Rationals

GAP can perform basic operations such as

- addition $1 + 1$;
- multiplication $2 * 4$;
- subtraction $2 - 4$;
- division $4/2$;
- exponentiation 2^4 ;
- comparison $2 < 4$;

Finite Fields

Defining $GF(q)$

```
gap> GF(9);  
GF(3^2)
```

Generator for $GF(q) \setminus \{0\}$

GAP calls $Z(q)$ the generator of the cyclic group of $GF(q) \setminus \{0\}$

Finite Fields

GAP can perform basic operations such as

- addition $\mathbb{Z}(9) + \mathbb{Z}(9)$;
- multiplication $\mathbb{Z}(9)^4 * \mathbb{Z}(9)$;
- multiplication $2 * \mathbb{Z}(9)$;
- subtraction $\mathbb{Z}(9)^5 - \mathbb{Z}(9)^0$;
- division $\mathbb{Z}(9) / \mathbb{Z}(9)^{-1}$;
- exponentiation $\mathbb{Z}(9)^4$;

Permutations

Disjoint Cycle Notation

```
gap> (1,2,5,6,7) (3,8);
```

Acting on the right

This permutation is interpreted as a function mapping 1 to 2 and 2 to 5 etc.

Permutations

GAP can perform basic operations such as

- multiplication $(1, 2, 5) * (3, 5, 7);$
- division $(1, 2, 7, 4) / (2, 4, 5);$
- exponentiation $(1, 3, 2)(4, 5)^4;$

Operations

Same symbols

Note that GAP decides from the context what symbols like $*$ mean and interprets them accordingly.

Variables

Assignment

The command

```
gap> xy := ( 1, 2, 3 );
```

assigns the variable `xy` the permutation $(1, 2, 3)$.

The variable `xy` can now be used instead of the permutation.

Data Structures: Lists

lists

Lists are an ordered collection of GAP objects.

```
gap> l := [ 1, 3, 5 ];
```

assigns the variable `l` the list of three integers.

accessing list elements

We can access the second element in a list via

```
gap> l[2];
```

```
gap> 3;
```

Data Structures: Lists

objects in lists

Lists can contain any GAP objects

```
gap> l := [ 1, (2, 3), GF(5) ];
```

Data Structures: Lists

Functions on Lists

- `Length(l);`
number of objects in `l`
- `Add(l,e);`
adds element `e` to the end of list `l`
- `Append(l, m);`
adds all elements of `m` to end of list `l`

Data Structures: Lists

More functions on Lists

- `List(l, i -> f(i));`
creates the list `[f(l[1]), .. , f(l[n])]`,
where `n = Length(l)`;
- `Filtered(l,f);`
the list of `e` in `l` for which `f(e)=true`
- `First(l, f);`
returns first `e` of `l` for which `f(e) = true`

Data Structures: Lists

Lists are versatile

Many objects in GAP are represented as lists, e.g.

- vectors are lists with entries in a field;
- matrices are lists of lists with entries in a field;

Data Structures: Sets

sets

Sets in GAP are special lists.

- list has no holes;
- elements are comparable with ' $<$ ';
- the list is strictly ordered

Data Structures: Sets

Creating a set

```
gap> s := Set( [ 1, 7, 4, 2 ] );  
[ 1, 2, 4, 7]
```

Data Structures: Sets

Functions on sets

- `Intersection(s, t);`
computes intersection of sets s and t
- `AddSet(s, e);`
adds element e to set s

Data Structures: Records

records

Records provide a way of storing objects in one data structure. Each object can be accessed by a unique *name*.

Data Structures: Records

Creating a record

```
gap> r := rec( degree := 3, gens := [
(1,2), (1,2,3)]);
```

Data Structures: Records

Creating a record

```
gap> r := rec( degree := 3,  
             gens := [ (1,2), (1,2,3)]);  
gap> r.degree;  
gap> 3  
gap> r.gens[1];  
gap> (1,2)
```

Domains

algebraic structures

GAP represents algebraic structures as *domains*.

Domains

Example domains

- Groups
- Fields
- vector spaces
- semigroups
- conjugacy classes

Domains

What are domains in GAP?

- 1 domains are sets with additional structure.
- 2 domains are closed under certain operations.

Domains

- 1 Group()
- 2 GF()
- 3 Ring()
- 4 VectorSpace()
- 5 ConjugacyClass()
- 6 SymmetricGroup()

Domains

- 1 GAP stores mathematical objects in Domains.
- 2 GAP collects additional information for mathematical objects.
- 3 `KnownPropertiesOfObject()` lists names of properties of an object for which GAP can determine whether or not the object possesses the property.

Programming in GAP

GAP - the language

A programming language especially suited for algebraic applications

Programming in GAP

if-statement

```
if i = 0 then  
    a := 1;  
else a := 2;  
fi;
```

Programming in GAP

for-loop

```
a := 10;  
for i in [ 1 .. 20] do  
    a := a + 1;  
od;
```

Programming in GAP

while-loop

```
a := 10;  
while a > 0 do  
    a := a - 1;  
od;
```

Programming in GAP

functions

```
myfn := function( a )  
local i;  
i := 1;  
while a > 0 do  
    i := i * a;  
    a := a - 1;  
od;  
return i;  
end;
```

Programming in GAP

What does myfn() compute?

functions

```
myfn := function( a )  
  local i;  
  i := 1;  
  while a > 0 do  
    i := i * a;  
    a := a - 1;  
  od;  
  return i;  
end;
```

Permutation groups in GAP

```
gap> grp := Group( (1, 2),(1,4)(2,3) );  
Defines the group generated by (1, 2) and  
(1, 4)(2, 3).
```

Elements of a group

```
gap> Size(grp);
```

8

This group is small enough for us to compute all elements

```
gap> Elements(grp);
```

```
[(),(3,4),(1,2),(1,2)(3,4),(1,3)(2,4),  
(1,3,2,4),(1,4,2,3), (1,4)(2,3) ]
```

The Symmetric Group

```
gap> grp := SymmetricGroup(15);
```

```
Sym([1..15])
```

```
gap> Size(grp);
```

```
1307674368000
```

This is too big to compute all of its elements

Properties of the group

```
gap> IsAbelian(grp);
```

```
false
```

```
gap> IsSimple(grp);
```

```
false
```

```
gap> Orbits(grp);
```

```
[ [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 ] ]
```

More properties of the group

```
gap> dgrp := DerivedSubgroup(grp);  
<permutation group with 13 generators> ;  
gap> IsSimple(dgrp);  
true  
gap> Size(dgrp);  
653837184000  
gap> Size(dgrp) = Factorial(15)/2;  
true
```

Random Elements in Groups

```
gap> g := PseudoRandom(grp);  
( 1, 2,12,14,15, 8, 3,13,10, 6, 9)( 5, 7,11)  
gap> g * g;  
( 1,12,15, 3,10, 9, 2,14, 8,13, 6)( 5,11, 7)
```

Membership in Groups

We can test whether a particular permutation is an element of grp:

```
gap> g in grp;
```

```
true
```

```
gap> (1,16) in grp;
```

```
false
```

```
gap> MovedPoints(grp);
```

```
[1 .. 15 ]
```

What does GAP know about grp?

```
gap> KnownPropertiesOfObject(grp) ;  
[ "IsEmpty", "IsTrivial", "IsNonTrivial", "IsFinite",  
  "IsGeneratorsOfMagmaWithInverses",  
  "IsAssociative",  
  "KnowsHowToDecompose",  
  "IsNaturalSymmetricGroup", "IsSymmetricGroup",  
  "IsPrimitiveAffine", ..]  
gap> IsTrivial( grp );  
false
```

Dihedral Groups

```
gap>d12:=DihedralGroup(IsPermGroup,12);  
gap> Elements(d12);  
[ (), (2,6)(3,5), (1,2)(3,6)(4,5), (1,2,3,4,5,6),  
(1,3)(4,6), (1,3,5)(2,4,6), (1,4)(2,3)(5,6),  
(1,4)(2,5)(3,6), (1,5)(2,4), (1,5,3)(2,6,4),  
(1,6,5,4,3,2), (1,6)(2,5)(3,4) ]
```

Dihedral Groups

```
gap> ccs := ConjugacyClasses(d12);  
[ ()^G, (2,6)(3,5)^G, (1,2)(3,6)(4,5)^G,  
(1,2,3,4,5,6)^G, (1,3,5)(2,4,6)^G, (1,4)(2,5)(3,6)^G  
]  
gap> Elements(ccs[3]);  
[ (1,2)(3,6)(4,5), (1,4)(2,3)(5,6), (1,6)(2,5)(3,4) ]
```

The break-loop

A break-loop is like a normal
gap> loop, but is printed as
brk>

The break-loop

Why do we need a break loop?

- to interrupt and check long computations
- to determine what caused an error
- to debug and test our GAP-functions

The break-loop

The break-loop is entered

- by typing `^C` (control-C)
- via the function `Error()`

Example

The following function contains an infinite loop:

```
myfn := function( a )  
local i;  
i := 1;  
while a > 0 do  
    i := i * a;  
    a := a + 1;  
od;  
return i;  
end;
```

Example

```
gap> f(1);
```

Example

```
gap> f(1);  
user interrupt at  
a := a + 1;  
<function>( <arguments> ) called from  
read-eval-loop  
Entering break read-eval-print loop, you  
can 'quit;' to quit to outer loop  
or you can return to continue  
brk>
```

Example

```
brk> a;
```

```
26381
```

```
brk> i;
```

```
<<an integer too large to be printed>
```

Example

```
myfn := function( a )  
local i;  
  
i := 1;  
while a > 0 do  
i := i * a;  
a := a + 1; ← mistake!  
od;  
return i;  
end;
```

Example

```
myfn := function( a )  
local i;  
  
i := 1;  
while a > 0 do  
i := i * a;  
a := a-1; ← correction!  
od;  
return i;  
end;
```

brk> loop

- Gives access to local variables in functions
- allows to change local variables in functions

brk> loop

- For debugging enter brk-loop by a call to `Error()`
- resume computation with `return;`
- stop computation with `quit;`

Example

```
myfn := function( a )  
local i;  
i := 1;  
while a > 0 do  
    i := i * a;  
    a := a + 1;  
    Error(‘‘my stopping point 1’’);  
od;  
return i;  
end;
```

Example

```
gap> f(1);  
Error my stopping point 1 at  
Error( my stopping point 1" );  
<function>( <arguments> ) called from  
read-eval-loop  
Entering break read-eval-print loop, you  
can 'quit;' to quit to outer loop  
or you can return to continue
```

Example

```
brk> i;
```

```
1
```

```
brk> a;
```

```
2
```

```
brk> return;
```

Example

```
gap> f(1);  
Error my stopping point 1 at  
Error( my stopping point 1" );  
<function>( <arguments> ) called from  
read-eval-loop  
Entering break read-eval-print loop, you  
can 'quit;' to quit to outer loop  
or you can return to continue
```

Example

```
brk> i;
```

```
2
```

```
brk> a;
```

```
3
```

```
brk> return;
```

Example

```
gap> f(1);  
Error my stopping point 1 at  
Error( my stopping point 1" );  
<function>( <arguments> ) called from  
read-eval-loop  
Entering break read-eval-print loop, you  
can 'quit;' to quit to outer loop  
or you can return to continue
```

Example

```
brk> i;
```

```
6
```

```
brk> a;
```

```
4
```

```
brk> quit;
```

GAP function Libraries

- functions written in GAP language
- implement many algorithms for groups, vector spaces, number theory
- in the directory `lib`
- readable and accessible

Group Libraries

- basic groups, e.g.
 - cyclic groups
 - dihedral groups
 - symmetric or alternating groups
- classical matrix groups
 - general, special and projective linear, symplectic, unitary and orthogonal groups
- transitive permutation groups of degree at most 23

Group Libraries

- groups of small order (at most 2000)
- finite perfect groups of size at most 10^6
- all primitive permutation groups of degree < 256 ,
- some primitive permutation groups of degree < 1000
- the irreducible solvable subgroups of $GL(n, p)$ for $n > 1$ and $p^n < 256$

Refereed Packages

- located in pkg directory
- some in GAP others in C
- refereed and tested by an expert

Contributed Code

Unrefereed code can also be contributed to GAP

Documentation

- GAP tutorial
Helps you get started and learn the basics
- GAP reference manual
Documents GAP functionality for users
- GAP programmer's manual
- Extending GAP manual
- New Features manual