Acknowledgements

- Thanks to the organizers

  - especially Temur

- Thanks to other colleagues

  - especially Steve Linton and Alexander Konovalov

What is GAP?

- GAP stands for

  Groups, Algorithms and Programming
- GAP is a system for computational discrete algebra
- Emphasis* on computational group theory
- A tool for research and teaching
- Free and extensible

History

- Neubüser, Schönert

  Cast of thousands(?)

- Aachen, St Andrews
- Centres

  - also include Fort Collins and Braunschweig

- GAP Council
- Worldwide collaboration

Structure

- Kernel - C
- GAP programming language
- GAP Library
- Data Libraries
- Packages

Getting GAP

- Website

  http://www.gap-system.org

- Installation

  UNIX (including Mac OS X), MS Windows

- Alternatives

  Linux binary distribution via rsync

  'Experimental' GAP Installer for Windows

Starting GAP

- To start GAP

  Type gap or click icon

- Command line switches
- Then GAP will display a banner, some system information, and a prompt:

```
jjm.chrystal.mcs.st-and.ac.uk > gap

         #########           ######         ###########           ###
       #############         ######         ############        ####
      ##############         #######        #############       #####
     ###############         #######        #####   ######      #####
    ######         #         ########       #####    #####     ######
   ######                   #########      #####    #####     #######
   #####                    #####  ####     #####   ######   ########
   ####                     #####  #####    #############   ###  ####
   #####     #######        ####    ####    ###########    ####  ####
   #####     #######       #####    #####   ######        ####   ####
   #####     #######       #####    #####   #####         #############
    #####      #####      ################   #####         #############
    ######     #####      ################   #####         #############
    ################     ##################  #####                 ####
     ###############     #####        #####  #####                 ####
       #############     #####        #####  #####                 ####
        #########       #####          #####  #####                 ####

     Information at:  http://www.gap-system.org
     Try '?help' for help. See also  '?copyright' and  '?authors'

   Loading the library. Please be patient, this may take a while.
GAP4, Version: 4.4.10 of 02-Oct-2007, i686-pc-linux-gnu-gcc
Components:  small 2.1, small2 2.0, small3 2.0, small4 1.0, small5 1.0,
             small6 1.0, small7 1.0, small8 1.0, small9 1.0, small10 0.2,
             id2 3.0, id3 2.1, id4 1.0, id5 1.0, id6 1.0, id9 1.0, id10 0.1,
             trans 1.0, prim 2.1  loaded.
Packages:    AClib 1.1, Polycyclic 2.2, Alnuth 2.2.5, CrystCat 1.1.2,
             Cryst 4.1.5, Carat 2.0.2, AutPGrp 1.2, CRISP 1.3.1, CTblLib 1.1.3,
             TomLib 1.1.2, FactInt 1.5.2, GAPDoc 1.1, FGA 1.1.0.1, HAP 1.8.2,
             EDIM 1.2.3, IRREDSOL 1.1.2, LAGUNA 3.4, Sophus 1.23,
             Polenta 1.2.7, ResClasses 2.5.3  loaded.

gap>
```

The prompt

- We start using GAP at the prompt

```
gap>
```

- Type an expression, end it with a semicolon ';'
- Eg

```
gap> 2 + 1;
3
```

- And GAP prints the value of the expression

The prompt

- Multiline

```
gap> 2 + 1
> ;
3
```

- Suppressing printing

```
gap> n := 2^10000;;
gap> n;
[…]
```

The prompt

- Editing input

  emacs-like keys, eg Ctrl-B for back, or cursor keys

- Command history

  Ctrl-P or the ‘up arrow’ gives the previous line

- Tab completion

Help

- Online manual

```
gap> ?
[…]
```

or

```
gap> ?help
gap> ?Group
[…]
```

- Searching chapters

```
gap> ??
```

Help

- PDF or HTML
- Website

  Includes a search facility
- GAP Forum

  http://mail.gap-system.org/mailman/listinfo/forum
- Support list

  support@gap-system.org

Objects

- Integers and rationals

```
gap> 6411/33;
2137/11
```

- Finite field elements

```
gap> Z(3)^2;
Z(3)^0
```

- Permutations

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

Variables

- Identifiers bound to objects
- last, last2, last3

```
gap> Group((1,2));
Group([ (1,2) ])
gap> G:=last;
Group([ (1,2) ])
gap>
```

- You may not overwrite GAP system variables

Demo

- Basics

Breaks and errors

- The break loop

```
gap> Factorial(1/2);
Range: <last> must be an integer less than 2^28 (not
   a rational) at
return Product( [ 1 .. n ] );
 called from
<function>( <arguments> ) called from read-eval-loop
Entering break read-eval-print loop ...
you can 'quit;' to quit to outer loop, or
you can replace <last> via 'return <last>;' to
   continue
brk> quit;
```

Breaks and errors

- No Method Found

```
gap> IsNormal(SymmetricGroup(4),(1,2));
Error, no method found! For debugging hints type
  ?Recovery from NoMethodFound
[…]
```

- Other errors

```
gap> factorial(10);
Variable: 'factorial' must have a value

gap>
```

Reading

- We can input commands (programs) by reading a file

```
gap> Read("filename.g");
```

- The results are not printed (unless explicit '`Print`' commands are used)
- Comments

```
gap> Factorial(30); # This is a comment
265252859812191058636308480000000
```

Logging

- Log entire session

```
gap> LogTo("filename.g")
```

- End logging

```
gap> LogTo();
```

- Logs can be used as test files via

```
gap> ReadTest("filename.g")
```

- See also

`LogInputTo` and `LogOutputTo`

Saving

- Computed objects can be saved

  `PrintTo, AppendTo`

- Saving a workspace

```
gap> SaveWorkspace( "filename.ws");
gap> quit;
```

- Restart GAP with the saved workspace

```
gap -L filename.ws
```

Lists

- Lists use the `[ ]` constructor
- Lists can contain any GAP objects

```
gap> ll := [ 1, (1,2), true, GF(9) ];
```

- Lists may contain holes

```
gap> Unbind(ll[1]);
[ , (1,2), true, GF(3^2) ]
```

Lists

- List functions

```
Length( l ), Add( l, elm ), Append( l1, l2 )
List( l, x -> f(x) )
Filtered, First
```

and many more…

Lists

- Many objects represented as lists
- Vectors, Matrices (lists of lists)
- Sets are special lists
  - no holes or duplicates
  - sorted (comparable with `<`)
  - `Union, Intersection`

Records

- Store objects in one data structure
- Access components by name

```
gap> r:= rec( name1 := object1,
      name2 := object2, … );
gap> r.name2;
object2
```

Demo

- Lists etc.

GAP Packages

- Libraries and accepted vs deposited
- See the website for a list
- Packages archive available alongside 'core' GAP for download

  - but authors maintain packages

GAP Packages

- To use a package

```
gap> LoadPackage( "package-name" );
```

- Autoloaded packages
- No autoloading - start GAP with

```
gap -A
```

- `LoadAllPackages();`

Demo

- Packages, libraries and some object constructions

Programming

- See the following for a handy quick reference

`http://ukrgap.exponenta.ru/VUB/GAP/6lang/gapref.pdf`

Functions

- Syntax

  `function( <arguments> ) <statements> end`

- Functions are expressions, so may be assigned to variables

```
gap> sayhello:= function()
> Print(”Hello, world.\n");
> end;
gap> sayhello;
Hello, world
```

Functions

- May have any number of arguments

```
gap> printDigits := function ( arg )
>      Print( arg );
>      Print( "\n" );
>    end;
```

- Functions for functions

```
NumberArgumentsFunction( func )
NamesLocalVariablesFunction( func )
```

If

- The syntax of the `if-then-else` statement

```
if <bool-expr1> then
<statements1>
{ elif <bool-expr2> then
<statements2> }
[ else <statements3> ]
fi;
```

If

- An example

```
gap> sign:= function(n)
>         if n < 0 then
>            return -1;
>         elif n = 0 then
>            return 0;
>         else
>            return 1;
>         fi;
>     end;
```

For

- `for`

```
for <simple-var> in <list-expr> do <statements> od;
```

For

- An example

```
gap> repeats := function( x, n )
>    local l, i;
>    l := [ ];
>    for i in [1..n] do
>      Add( l, x );
>    od;
>    return l;
> end;
```

- What does it do?

While and repeat

- `while` syntax

```
while <condition> do <statements> od;
```

- `repeat` syntax

```
repeat <statements> until <bool-expr>;
```

Loop control

- `break`

  - causes an immediate exit from the innermost loop enclosing it.

- `continue`

  - causes the rest of the current iteration of the innermost loop enclosing it to be skipped.

Demo

- A demonstration of some simple elements of programming

Attributes

- Attributes
- Once computed, they are set
- Eg `Size( group )`

- but not `Random( group )`

Properties

- Boolean valued attributes
- Eg `IsAbelian( group )`

Operations and methods

- An operation is a special GAP function that bundles its *methods*
- Each method computes the same result
- Different methods used for different arguments

Method selection

- How is the right method chosen?
- Filters - depending on Families (see `?families`) and Properties
- The Family of an object determines its relationship to other objects (comparable)

Method selection

- `FamilyObj( obj )`
- `KnownAttributesOfObject( obj )`
- `KnownPropertiesOfObject( obj )`
- `KnownTruePropertiesOfObject( obj )`

Demo

- Operations and methods

Creating new operations

- `DeclareOperation( <name>, <args-filts> )`
- `InstallMethod( <opr>, […], <args-filts>, […], <method> )`

Creating new objects

- To create an entirely new type of object in GAP may require the creation of new families, filters, operations and methods and more
- See `?Creating New Objects`

Package authoring

- Example pkg
- GAPDoc?
  - documentation must integrate into the GAP online manual system
- Development version access
- Testing - name clashes, loading with others, etc
- Package refereeing