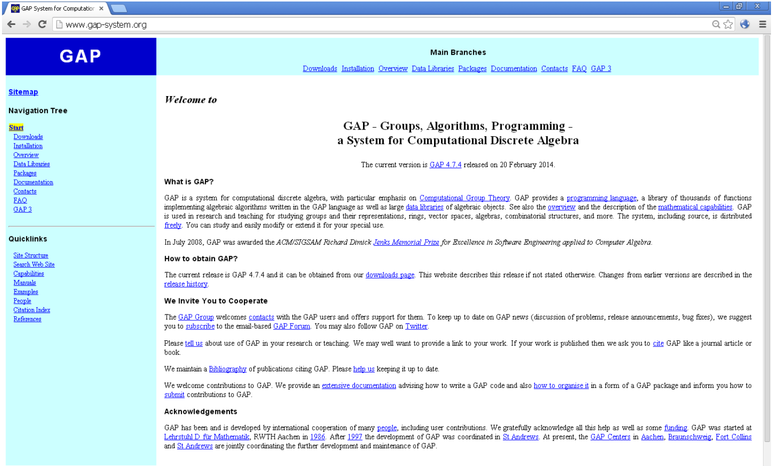
Group, Algebra, Programming

M. Farrokhi D. G.

Department of Pure Mathematics, Ferdowsi University of Mashhad

6th Group Theory Conference of Iran
GAP Workshop
Golestan University
March 12-13, 2014

GAP Website: www.gap-system.org



GAP System for Computation

www.gap-system.org

GAP

Main Branches

Downloads Installation Overview Data Libraries Packages Documentation Contacts FAQ GAP 3

Sitemap

Navigation Tree

Start
Downloads
Installation
Overview
Data Libraries
Packages
Documentation
Contacts
FAQ
GAP 3

Quicklinks

Site Structure
Search Web Site
Capabilities
Manuals
Examples
People
Citation Index
References

Welcome to

GAP - Groups, Algorithms, Programming - a System for Computational Discrete Algebra

The current version is GAP 4.7.4 released on 20 February 2014.

What is GAP?

GAP is a system for computational discrete algebra, with particular emphasis on Computational Group Theory. GAP provides a programming language, a library of thousands of functions implementing algebraic algorithms written in the GAP language as well as large data libraries of algebraic objects. See also the overview and the description of the mathematical capabilities. GAP is used in research and teaching for studying groups and their representations, rings, vector spaces, algebras, combinatorial structures, and more. The system, including source, is distributed freely. You can study and easily modify or extend it for your special use.

In July 2008, GAP was awarded the *ACM/SIGSAM Richard Dimick Jenks Memorial Prize for Excellence in Software Engineering applied to Computer Algebra.*

How to obtain GAP?

The current release is GAP 4.7.4 and it can be obtained from our downloads page. This website describes this release if not stated otherwise. Changes from earlier versions are described in the release history.

We Invite You to Cooperate

The GAP Group welcomes contacts with the GAP users and offers support for them. To keep up to date on GAP news (discussion of problems, release announcements, bug fixes), we suggest you to subscribe to the email-based GAP Forum. You may also follow GAP on Twitter.

Please tell us about use of GAP in your research or teaching. We may well want to provide a link to your work. If your work is published then we ask you to cite GAP like a journal article or book.

We maintain a Bibliography of publications citing GAP. Please help us keeping it up to date.

We welcome contributions to GAP. We provide an extensive documentation advising how to write a GAP code and also how to organise it in a form of a GAP package and inform you how to submit contributions to GAP.

Acknowledgements

GAP has been and is developed by international cooperation of many people, including user contributions. We gratefully acknowledge all this help as well as some funding. GAP was started at Lehrstuhl D für Mathematik, RWTH Aachen in 1986. After 1997 the development of GAP was coordinated in St Andrews. At present, the GAP Centers in Aachen, Braunschweig, Fort Collins and St Andrews are jointly coordinating the further development and maintenance of GAP.

Running GAP: C:/gap4r7/bin/gap.bat

/cygdrive/C/gap4r7/bin/gapw95.exe -l /cygdrive/C/gap4r7

```
 ┌──────┐    GAP, Version 4.7.2 of 01-Dec-2013 (free software, GPL)
 │ GAP  │    http://www.gap-system.org
 └──────┘    Architecture: i686-pc-cygwin-gcc-default32
 Libs used:  gmp, readline
 Loading the library and packages ...
 Components: trans 1.0, prim 2.1, small* 1.0, id* 1.0
 Packages:   AClib 1.2, Alnuth 3.0.0, AtlasRep 1.5.0, AutPGrp 1.5,
             Browse 1.8.3, CRISP 1.3.7, Cryst 4.1.12, CrystCat 1.1.6,
             CTblLib 1.2.2, FactInt 1.5.3, FGA 1.2.0, GAPDoc 1.5.1,
             IO 4.2, IRREDSOL 1.2.3, LAGUNA 3.6.4, Polenta 1.3.1,
             Polycyclic 2.11, RadiRoot 2.6, ResClasses 3.3.2,
             Sophus 1.23, SpinSym 1.5, TomLib 1.2.4
 Try '?help' for help. See also  '?copyright' and  '?authors'
gap>
```

How to define a group in GAP?

How to define a group in GAP?

1 Predefined groups

How to define a group in GAP?

1. Predefined groups
2. Small groups library

How to define a group in GAP?

1. Predefined groups
2. Small groups library
3. Presentations

How to define a group in GAP?

1. Predefined groups
2. Small groups library
3. Presentations
4. Permutations groups

How to define a group in GAP?

1. Predefined groups
2. Small groups library
3. Presentations
4. Permutations groups
5. Matrix groups

Groups known in GAP

Groups known in GAP

- TrivialGroup()

Groups known in GAP

- TrivialGroup()
- CyclicGroup(order)

Groups known in GAP

- TrivialGroup()
- CyclicGroup(order)
- AbelianGroup(order 1, ... , order n)

Groups known in GAP

- TrivialGroup()
- CyclicGroup(order)
- AbelianGroup(order 1, ... , order n)
- ElementaryAbelianGroup(order)

Groups known in GAP

- TrivialGroup()
- CyclicGroup(order)
- AbelianGroup(order 1, ... , order n)
- ElementaryAbelianGroup(order)
- DihedralGroup(order)

Groups known in GAP

- TrivialGroup()
- CyclicGroup(order)
- AbelianGroup(order 1, ... , order n)
- ElementaryAbelianGroup(order)
- DihedralGroup(order)
- QuaternionGroup(order)

Groups known in GAP

- TrivialGroup()
- CyclicGroup(order)
- AbelianGroup(order 1, ... , order n)
- ElementaryAbelianGroup(order)
- DihedralGroup(order)
- QuaternionGroup(order)
- DicyclicGroup(order)

Groups known in GAP

- TrivialGroup()
- CyclicGroup(order)
- AbelianGroup(order 1, ... , order n)
- ElementaryAbelianGroup(order)
- DihedralGroup(order)
- QuaternionGroup(order)
- DicyclicGroup(order)
- ExtraspecialGroup(order, exponent or type)

Groups known in GAP

- TrivialGroup()
- CyclicGroup(order)
- AbelianGroup(order 1, ... , order n)
- ElementaryAbelianGroup(order)
- DihedralGroup(order)
- QuaternionGroup(order)
- DicyclicGroup(order)
- ExtraspecialGroup(order, exponent or type)
- SymmetricGroup(degree)

Groups known in GAP

- TrivialGroup()
- CyclicGroup(order)
- AbelianGroup(order 1, ... , order n)
- ElementaryAbelianGroup(order)
- DihedralGroup(order)
- QuaternionGroup(order)
- DicyclicGroup(order)
- ExtraspecialGroup(order, exponent or type)
- SymmetricGroup(degree)
- AlternatingGroup(degree)

Groups known in GAP

- TrivialGroup()
- CyclicGroup(order)
- AbelianGroup(order 1, ... , order n)
- ElementaryAbelianGroup(order)
- DihedralGroup(order)
- QuaternionGroup(order)
- DicyclicGroup(order)
- ExtraspecialGroup(order, exponent or type)
- SymmetricGroup(degree)
- AlternatingGroup(degree)
- MathieuGroup(degree)

Groups known in GAP

- TrivialGroup()
- CyclicGroup(order)
- AbelianGroup(order 1, ... , order n)
- ElementaryAbelianGroup(order)
- DihedralGroup(order)
- QuaternionGroup(order)
- DicyclicGroup(order)
- ExtraspecialGroup(order, exponent or type)
- SymmetricGroup(degree)
- AlternatingGroup(degree)
- MathieuGroup(degree)
- SuzukiGroup(q)

Groups known in GAP

- TrivialGroup()
- CyclicGroup(order)
- AbelianGroup(order 1, ... , order n)
- ElementaryAbelianGroup(order)
- DihedralGroup(order)
- QuaternionGroup(order)
- DicyclicGroup(order)
- ExtraspecialGroup(order, exponent or type)
- SymmetricGroup(degree)
- AlternatingGroup(degree)
- MathieuGroup(degree)
- SuzukiGroup(q)
- ReeGroup(q)

Groups known in GAP

- Matrix groups

Groups known in GAP

- Matrix groups
  - GeneralLinearGroup(dimention, order of field)

Groups known in GAP

- Matrix groups
  - GeneralLinearGroup(dimention, order of field)
  - SpecialLinearGroup(dimention, order of field)

Groups known in GAP

- Matrix groups
  - GeneralLinearGroup(dimention, order of field)
  - SpecialLinearGroup(dimention, order of field)
  - GeneralUnitaryGroup(dimension, order of field)

Groups known in GAP

- Matrix groups
  - GeneralLinearGroup(dimention, order of field)
  - SpecialLinearGroup(dimention, order of field)
  - GeneralUnitaryGroup(dimension, order of field)
  - SpecialUnitaryGroup(dimension, order of field)

Groups known in GAP

- Matrix groups
  - GeneralLinearGroup(dimention, order of field)
  - SpecialLinearGroup(dimention, order of field)
  - GeneralUnitaryGroup(dimension, order of field)
  - SpecialUnitaryGroup(dimension, order of field)
  - SymplecticGroup(dimension, order of field)

Groups known in GAP

- Matrix groups
  - GeneralLinearGroup(dimention, order of field)
  - SpecialLinearGroup(dimention, order of field)
  - GeneralUnitaryGroup(dimension, order of field)
  - SpecialUnitaryGroup(dimension, order of field)
  - SymplecticGroup(dimension, order of field)
  - GeneralOrthogonalGroup(dimension, order of field)

Groups known in GAP

- Matrix groups
  - GeneralLinearGroup(dimention, order of field)
  - SpecialLinearGroup(dimention, order of field)
  - GeneralUnitaryGroup(dimension, order of field)
  - SpecialUnitaryGroup(dimension, order of field)
  - SymplecticGroup(dimension, order of field)
  - GeneralOrthogonalGroup(dimension, order of field)
  - SpecialOrthogonalGroup(dimension, order of field)

Groups known in GAP

- Matrix groups
  - GeneralLinearGroup(dimention, order of field)
  - SpecialLinearGroup(dimention, order of field)
  - GeneralUnitaryGroup(dimension, order of field)
  - SpecialUnitaryGroup(dimension, order of field)
  - SymplecticGroup(dimension, order of field)
  - GeneralOrthogonalGroup(dimension, order of field)
  - SpecialOrthogonalGroup(dimension, order of field)
  - ProjectiveGeneralLinearGroup(dimension, order of field)

Groups known in GAP

- Matrix groups
  - GeneralLinearGroup(dimention, order of field)
  - SpecialLinearGroup(dimention, order of field)
  - GeneralUnitaryGroup(dimension, order of field)
  - SpecialUnitaryGroup(dimension, order of field)
  - SymplecticGroup(dimension, order of field)
  - GeneralOrthogonalGroup(dimension, order of field)
  - SpecialOrthogonalGroup(dimension, order of field)
  - ProjectiveGeneralLinearGroup(dimension, order of field)
  - ProjectiveSpecialLinearGroup(dimension, order of field)

Groups known in GAP

- Matrix groups
  - GeneralLinearGroup(dimention, order of field)
  - SpecialLinearGroup(dimention, order of field)
  - GeneralUnitaryGroup(dimension, order of field)
  - SpecialUnitaryGroup(dimension, order of field)
  - SymplecticGroup(dimension, order of field)
  - GeneralOrthogonalGroup(dimension, order of field)
  - SpecialOrthogonalGroup(dimension, order of field)
  - ProjectiveGeneralLinearGroup(dimension, order of field)
  - ProjectiveSpecialLinearGroup(dimension, order of field)
  - ProjectiveGeneralUnitaryGroup(dimension, order of field)

Groups known in GAP

- Matrix groups
  - GeneralLinearGroup(dimention, order of field)
  - SpecialLinearGroup(dimention, order of field)
  - GeneralUnitaryGroup(dimension, order of field)
  - SpecialUnitaryGroup(dimension, order of field)
  - SymplecticGroup(dimension, order of field)
  - GeneralOrthogonalGroup(dimension, order of field)
  - SpecialOrthogonalGroup(dimension, order of field)
  - ProjectiveGeneralLinearGroup(dimension, order of field)
  - ProjectiveSpecialLinearGroup(dimension, order of field)
  - ProjectiveGeneralUnitaryGroup(dimension, order of field)
  - ProjectiveSpecialUnitaryGroup(dimension, order of field)

Groups known in GAP

- Matrix groups
    - GeneralLinearGroup(dimention, order of field)
    - SpecialLinearGroup(dimention, order of field)
    - GeneralUnitaryGroup(dimension, order of field)
    - SpecialUnitaryGroup(dimension, order of field)
    - SymplecticGroup(dimension, order of field)
    - GeneralOrthogonalGroup(dimension, order of field)
    - SpecialOrthogonalGroup(dimension, order of field)
    - ProjectiveGeneralLinearGroup(dimension, order of field)
    - ProjectiveSpecialLinearGroup(dimension, order of field)
    - ProjectiveGeneralUnitaryGroup(dimension, order of field)
    - ProjectiveSpecialUnitaryGroup(dimension, order of field)
    - ProjectiveSymplecticGroup(dimension, order of field)

Groups known in GAP

GAP

```
gap> G:=AbelianGroup([2,4,8,16]);
<pc group of size 1024 with 4 generators>
gap> StructureDescription(G);
"C16 x C8 x C4 x C2"
```

Groups known in GAP

GAP

```
gap> G:=AbelianGroup([2,4,8,16]);
<pc group of size 1024 with 4 generators>
gap> StructureDescription(G);
"C16 x C8 x C4 x C2"
```

Groups known in GAP

GAP

```
gap> G:=AbelianGroup([2,4,8,16]);
<pc group of size 1024 with 4 generators>
gap> StructureDescription(G);
"C16 x C8 x C4 x C2"
gap> G:=SuzukiGroup(8);
Sz(8)
gap> Order(G);
"29120"
```

Groups known in GAP

GAP

```
gap> G:=AbelianGroup([2,4,8,16]);
<pc group of size 1024 with 4 generators>
gap> StructureDescription(G);
"C16 x C8 x C4 x C2"
gap> G:=SuzukiGroup(8);
Sz(8)
gap> Order(G);
"29120"
gap> G:=GL(2,7);;
gap> C:=Center(G);;
gap> StructureDescription(C);
"C6"
```

All groups of

- order at most 2000 except 1024 (423164062 groups);

All groups of

- order at most 2000 except 1024 (423164062 groups);
- cubefree order at most 50000 (395703 groups);

All groups of

- order at most 2000 except 1024 (423164062 groups);
- cubefree order at most 50000 (395703 groups);
- order p^7 for the primes $p = 3, 5, 7, 11$ (907489 groups);

All groups of

- order at most 2000 except 1024 (423164062 groups);
- cubefree order at most 50000 (395703 groups);
- order p^7 for the primes $p = 3, 5, 7, 11$ (907489 groups);
- order p^n for $n \leq 6$ and all primes p

All groups of

- order at most 2000 except 1024 (423164062 groups);
- cubefree order at most 50000 (395703 groups);
- order p^7 for the primes $p = 3, 5, 7, 11$ (907489 groups);
- order p^n for $n \leq 6$ and all primes p
- order pq^n for q^n dividing 28, 36, 55 or 74 and all primes p with $p \neq q$

All groups of

- order at most 2000 except 1024 (423164062 groups);
- cubefree order at most 50000 (395703 groups);
- order p^7 for the primes $p = 3, 5, 7, 11$ (907489 groups);
- order p^n for $n \leq 6$ and all primes p
- order pq^n for q^n dividing 28, 36, 55 or 74 and all primes p with $p \neq q$
- squarefree order

All groups of

- order at most 2000 except 1024 (423164062 groups);
- cubefree order at most 50000 (395703 groups);
- order p^7 for the primes $p = 3, 5, 7, 11$ (907489 groups);
- order p^n for $n \leq 6$ and all primes p
- order pq^n for q^n dividing 28, 36, 55 or 74 and all primes p with $p \neq q$
- squarefree order
- order pqr.

How to use small group library?

- SmallGroup(order,index)

How to use small group library?

- SmallGroup(order,index)
- NumberSmallGroups(order)

How to use small group library?

- SmallGroup(order,index)
- NumberSmallGroups(order)
- IdSmallGroup(group)

How to use small group library?

- SmallGroup(order,index)
- NumberSmallGroups(order)
- IdSmallGroup(group)

GAP

```
gap> NumberSmallGroups(1024);
49487365422
```

How to use small group library?

- SmallGroup(order,index)
- NumberSmallGroups(order)
- IdSmallGroup(group)

GAP

```
gap> NumberSmallGroups(1024);
49487365422
```

How to use small group library?

- SmallGroup(order,index)
- NumberSmallGroups(order)
- IdSmallGroup(group)

GAP

```
gap> NumberSmallGroups(1024);
49487365422
gap> G:=SmallGroup(120,15);;
gap> StructureDescription(G);
"C5 x SL(2,3)"
```

How to use small group library?

- SmallGroup(order,index)
- NumberSmallGroups(order)
- IdSmallGroup(group)

GAP

```
gap> NumberSmallGroups(1024);
49487365422
gap> G:=SmallGroup(120,15);;
gap> StructureDescription(G);
"C5 x SL(2,3)"
gap> G:=AlternatingGroup(5);;
gap> IdSmallGroup(G);
[ 60, 5 ]
```

- Step 1. Define a free group:

- Step 1. Define a free group:
  - F:=FreeGroup(n);

- Step 1. Define a free group:
  - F:=FreeGroup(n);
- Step 2. Divide the free group by relators.

- Step 1. Define a free group:
  - F:=FreeGroup(n);
- Step 2. Divide the free group by relators.
  - G:=F/[relators];

- Step 1. Define a free group:
  - F:=FreeGroup(n);
- Step 2. Divide the free group by relators.
  - G:=F/[relators];

GAP

```
gap> F:=FreeGroup(2);
gap> G:=F/[F.1^4,F.2^2,F.1^F.2*F.1];
gap> StructureDescription(G);
"D8"
```

Some permutations

- (1,2,3)

Some permutations

- (1,2,3)
- (1,2,3)(4,5)(6,7,8,9)

Some permutations

- (1,2,3)
- (1,2,3)(4,5)(6,7,8,9)

GAP

```
gap> (1,2,3,4,5,6)^2;
(1,3,5)(2,4,6);
```

Some permutations

- (1,2,3)
- (1,2,3)(4,5)(6,7,8,9)

GAP

```
gap> (1,2,3,4,5,6)^2;
(1,3,5)(2,4,6);
```

Some permutations

- (1,2,3)
- (1,2,3)(4,5)(6,7,8,9)

GAP

```
gap> (1,2,3,4,5,6)^2;
(1,3,5)(2,4,6);
gap> G:=Group((1,2,3),(1,2));;
gap> StructureDescription(G);
"S3"
```

Finite fields

Finite fields and their elements

Finite fields

Finite fields and their elements

`Z(q)`	A generator of multiplicative group
`0*Z(q)`	Additive neutal element
`Z(q)^0`	Multiplicative neutal element

Finite fields

Finite fields and their elements

`Z(q)`	A generator of multiplicative group
`0*Z(q)`	Additive neutal element
`Z(q)^0`	Multiplicative neutal element
`GF(q)`	Field of order q
`0*GF(q)`	Additive neutal element
`GF(q)^0`	Multiplicative neutal element
`PrimitiveRoot(GF(q))`	A generator of multiplicative group

Some matrices

$$\begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} \longrightarrow \texttt{[[1,2],[0,1]]}$$

Some matrices

$$\begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} \longrightarrow \texttt{[[1,2],[0,1]]}$$

$$\begin{bmatrix} 1 & \lambda \\ 0 & 1 \end{bmatrix} \longrightarrow \texttt{[[Z(q)\^{}0,Z(q)],[0,Z(q)\^{}0]]}$$

Some matrices

$$\begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} \longrightarrow \texttt{[[1,2],[0,1]]}$$

$$\begin{bmatrix} 1 & \lambda \\ 0 & 1 \end{bmatrix} \longrightarrow \texttt{[[Z(q)\^{}0,Z(q)],[0,Z(q)\^{}0]]}$$

GAP

```
gap> Order([[1,2],[0,1]]);
infinity
```

Some matrices

$$\begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} \longrightarrow \texttt{[[1,2],[0,1]]}$$

$$\begin{bmatrix} 1 & \lambda \\ 0 & 1 \end{bmatrix} \longrightarrow \texttt{[[Z(q)\^{}0,Z(q)],[0,Z(q)\^{}0]]}$$

GAP

```
gap> Order([[1,2],[0,1]]);
infinity
```

Some matrices

$$\begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} \longrightarrow \texttt{[[1,2],[0,1]]}$$

$$\begin{bmatrix} 1 & \lambda \\ 0 & 1 \end{bmatrix} \longrightarrow \texttt{[[Z(q)\^{}0,Z(q)],[0,Z(q)\^{}0]]}$$

GAP

```
gap> Order([[1,2],[0,1]]);
infinity
gap> Order([[Z(5)^0,Z(5)],[0,Z(5)^0]]);
5
```

1. TrivialSubgroup(G)

1. TrivialSubgroup(G)
2. Center(G)

1. TrivialSubgroup(G)
2. Center(G)
3. DerivedSubgroup(G)

1. TrivialSubgroup(G)
2. Center(G)
3. DerivedSubgroup(G)
4. FrattiniSubgroup(G)

1. TrivialSubgroup(G)
2. Center(G)
3. DerivedSubgroup(G)
4. FrattiniSubgroup(G)
5. FittingSubgroup(G)

1. TrivialSubgroup(G)
2. Center(G)
3. DerivedSubgroup(G)
4. FrattiniSubgroup(G)
5. FittingSubgroup(G)
6. SylowSubgroup(G, p)

1. TrivialSubgroup(G)
2. Center(G)
3. DerivedSubgroup(G)
4. FrattiniSubgroup(G)
5. FittingSubgroup(G)
6. SylowSubgroup(G, p)
7. HallSubgroup(G, P)

1. TrivialSubgroup(G)
2. Center(G)
3. DerivedSubgroup(G)
4. FrattiniSubgroup(G)
5. FittingSubgroup(G)
6. SylowSubgroup(G, p)
7. HallSubgroup(G, P)
8. AllSubgroups(G)

1. TrivialSubgroup(G)
2. Center(G)
3. DerivedSubgroup(G)
4. FrattiniSubgroup(G)
5. FittingSubgroup(G)
6. SylowSubgroup(G, p)
7. HallSubgroup(G, P)
8. AllSubgroups(G)
9. MaximalSubgroups(G)

1. TrivialSubgroup(G)
2. Center(G)
3. DerivedSubgroup(G)
4. FrattiniSubgroup(G)
5. FittingSubgroup(G)
6. SylowSubgroup(G, p)
7. HallSubgroup(G, P)
8. AllSubgroups(G)
9. MaximalSubgroups(G)
10. NormalSubgroups(G)

1. TrivialSubgroup(G)
2. Center(G)
3. DerivedSubgroup(G)
4. FrattiniSubgroup(G)
5. FittingSubgroup(G)
6. SylowSubgroup(G, p)
7. HallSubgroup(G, P)
8. AllSubgroups(G)
9. MaximalSubgroups(G)
10. NormalSubgroups(G)
11. MaximalNormalSubgroups(G)

1. TrivialSubgroup(G)
2. Center(G)
3. DerivedSubgroup(G)
4. FrattiniSubgroup(G)
5. FittingSubgroup(G)
6. SylowSubgroup(G, p)
7. HallSubgroup(G, P)
8. AllSubgroups(G)
9. MaximalSubgroups(G)
10. NormalSubgroups(G)
11. MaximalNormalSubgroups(G)
12. MinimalNormalSubgroups(G)

1. TrivialSubgroup(G)
2. Center(G)
3. DerivedSubgroup(G)
4. FrattiniSubgroup(G)
5. FittingSubgroup(G)
6. SylowSubgroup(G, p)
7. HallSubgroup(G, P)
8. AllSubgroups(G)
9. MaximalSubgroups(G)
10. NormalSubgroups(G)
11. MaximalNormalSubgroups(G)
12. MinimalNormalSubgroups(G)

1. TrivialSubgroup(G)
2. Center(G)
3. DerivedSubgroup(G)
4. FrattiniSubgroup(G)
5. FittingSubgroup(G)
6. SylowSubgroup(G, p)
7. HallSubgroup(G, P)
8. AllSubgroups(G)
9. MaximalSubgroups(G)
10. NormalSubgroups(G)
11. MaximalNormalSubgroups(G)
12. MinimalNormalSubgroups(G)

GAP

```
gap> G:=DihedralGroup(8);;
gap> C:=Center(G);;
gap> D:=DerivedSubgroup(G);;
gap> F:=FrattiniSubroup(G);;
gap> C=D;D=F;F=C;
true
true
true
```

1. TrivialSubgroup(G)
2. Center(G)
3. DerivedSubgroup(G)
4. FrattiniSubgroup(G)
5. FittingSubgroup(G)
6. SylowSubgroup(G, p)
7. HallSubgroup(G, P)
8. AllSubgroups(G)
9. MaximalSubgroups(G)
10. NormalSubgroups(G)
11. MaximalNormalSubgroups(G)
12. MinimalNormalSubgroups(G)

GAP

```
gap> G:=DihedralGroup(8);;
gap> C:=Center(G);;
gap> D:=DerivedSubgroup(G);;
gap> F:=FrattiniSubroup(G);;
gap> C=D;D=F;F=C;
true
true
true
```

1. TrivialSubgroup(G)
2. Center(G)
3. DerivedSubgroup(G)
4. FrattiniSubgroup(G)
5. FittingSubgroup(G)
6. SylowSubgroup(G, p)
7. HallSubgroup(G, P)
8. AllSubgroups(G)
9. MaximalSubgroups(G)
10. NormalSubgroups(G)
11. MaximalNormalSubgroups(G)
12. MinimalNormalSubgroups(G)

GAP

```
gap> G:=DihedralGroup(8);;
gap> C:=Center(G);;
gap> D:=DerivedSubgroup(G);;
gap> F:=FrattiniSubroup(G);;
gap> C=D;D=F;F=C;
true
true
true
gap> G:=SymmetricGroup(5);
gap> P:=SylowSubgroup(G,2);
gap> StructureDescription(P);
"D8"
```

1. ConjugacyClass(G,g)

1. ConjugacyClass(G,g)
2. ConjugacyClasses(G)

1. ConjugacyClass(G,g)
2. ConjugacyClasses(G)
3. NrConjugacyClasses(G)

1. ConjugacyClass(G,g)
2. ConjugacyClasses(G)
3. NrConjugacyClasses(G)
4. Representative(class)

1. ConjugacyClass(G,g)
2. ConjugacyClasses(G)
3. NrConjugacyClasses(G)
4. Representative(class)

GAP

```
gap> G:=SymmetricGroup(5);;
gap> NrConjugacyClasses(G);
7
```

1. ConjugacyClass(G,g)
2. ConjugacyClasses(G)
3. NrConjugacyClasses(G)
4. Representative(class)

GAP

```
gap> G:=SymmetricGroup(5);;
gap> NrConjugacyClasses(G);
7
```

1. ConjugacyClass(G,g)
2. ConjugacyClasses(G)
3. NrConjugacyClasses(G)
4. Representative(class)

GAP

```
gap> G:=SymmetricGroup(5);;
gap> NrConjugacyClasses(G);
7
gap> ConjugacyClass(G,(1,2));
(1,2)^G
gap> List(ConjugacyClass(G,(1,2)));
[ (1,2), (1,3), (1,4), (1,5), (2,3), (2,4), (2,5), \
  (3,4), (3,5), (4,5) ]
```

1. ConjugacyClass(G,g)
2. ConjugacyClasses(G)
3. NrConjugacyClasses(G)
4. Representative(class)

GAP

```
gap> G:=SymmetricGroup(5);;
gap> NrConjugacyClasses(G);
7
gap> ConjugacyClass(G,(1,2));
(1,2)^G
gap> List(ConjugacyClass(G,(1,2)));
[ (1,2), (1,3), (1,4), (1,5), (2,3), (2,4), (2,5), \
  (3,4), (3,5), (4,5) ]
gap> List(ConjugacyClasses(G),Representative);
[ (), (1,2), (1,2)(3,4), (1,2,3), (1,2,3)(4,5), \
  (1,2,3,4), (1,2,3,4,5) ]
```

1. RightCoset(G,x)

1. RightCoset(G,x)
2. RightCosets(G,H)

1. RightCoset(G,x)
2. RightCosets(G,H)
3. RightTransversal(G,H)

1. RightCoset(G,x)
2. RightCosets(G,H)
3. RightTransversal(G,H)

GAP

```
gap> G:=SymmetricGroup(4);;
gap> H:=AlternatingGroup(3);;
gap> C:=RightCoset(H,(1,2));;
gap> List(C);
[ (1,2), (1,3), (2,3) ]
```

1. RightCoset(G,x)
2. RightCosets(G,H)
3. RightTransversal(G,H)

GAP

```
gap> G:=SymmetricGroup(4);;
gap> H:=AlternatingGroup(3);;
gap> C:=RightCoset(H,(1,2));;
gap> List(C);
[ (1,2), (1,3), (2,3) ]
```

1. RightCoset(G,x)
2. RightCosets(G,H)
3. RightTransversal(G,H)

GAP

```
gap> G:=SymmetricGroup(4);;
gap> H:=AlternatingGroup(3);;
gap> C:=RightCoset(H,(1,2));;
gap> List(C);
[ (1,2), (1,3), (2,3) ]
gap> R:=RightTransversal(G,H);;
gap> List(R);
[ (), (2,3), (1,4), (1,4)(2,3), (1,4,2), (1,4,2,3), \
  (1,4,3), (1,4,3,2) ]
```

1. FactorGroup(G, N)

1. FactorGroup(G, N)
2. CommutatorFactorGroup(G)

1. FactorGroup(G, N)
2. CommutatorFactorGroup(G)

GAP

```
gap> G:=DihedralGroup(8);;
gap> N:=Center(G);;
gap> F:=FactorGroup(G,N);;
gap> StructureDescription(F);
"C2 x C2"
```

1. IsCyclic(G)

1. IsCyclic(G)
2. IsAbelian(G)

1. IsCyclic(G)
2. IsAbelian(G)
3. IsElementaryAbelian(G)

1. IsCyclic(G)
2. IsAbelian(G)
3. IsElementaryAbelian(G)
4. IsNilpotentGroup(G)

1. IsCyclic(G)
2. IsAbelian(G)
3. IsElementaryAbelian(G)
4. IsNilpotentGroup(G)
5. IsSolvableGroup(G)

1. IsCyclic(G)
2. IsAbelian(G)
3. IsElementaryAbelian(G)
4. IsNilpotentGroup(G)
5. IsSolvableGroup(G)
6. IsPerfectGroup(G)

1. IsCyclic(G)
2. IsAbelian(G)
3. IsElementaryAbelian(G)
4. IsNilpotentGroup(G)
5. IsSolvableGroup(G)
6. IsPerfectGroup(G)
7. IsSimpleGroup(G)

1. IsCyclic(G)
2. IsAbelian(G)
3. IsElementaryAbelian(G)
4. IsNilpotentGroup(G)
5. IsSolvableGroup(G)
6. IsPerfectGroup(G)
7. IsSimpleGroup(G)

GAP

```
gap> G:=SmallGroup(120,5);;
gap> IsPerfectGroup(G);
true
```

1. IsCyclic(G)
2. IsAbelian(G)
3. IsElementaryAbelian(G)
4. IsNilpotentGroup(G)
5. IsSolvableGroup(G)
6. IsPerfectGroup(G)
7. IsSimpleGroup(G)

GAP

```
gap> G:=SmallGroup(120,5);;
gap> IsPerfectGroup(G);
true
```

1. IsCyclic(G)
2. IsAbelian(G)
3. IsElementaryAbelian(G)
4. IsNilpotentGroup(G)
5. IsSolvableGroup(G)
6. IsPerfectGroup(G)
7. IsSimpleGroup(G)

GAP

```
gap> G:=SmallGroup(120,5);;
gap> IsPerfectGroup(G);
true
gap> G:=AlternatingGroup(7);;
gap> IsSImpleGroup(G);
true
```

1. Order(G)

1. Order(G)
2. Exponent(G)

1. Order(G)
2. Exponent(G)
3. NilpotencyClassOfGroup(G)

1. Order(G)
2. Exponent(G)
3. NilpotencyClassOfGroup(G)
4. CommutatorLength(G)

1. Order(G)
2. Exponent(G)
3. NilpotencyClassOfGroup(G)
4. CommutatorLength(G)

GAP

```
gap> G:=SymmetricGroup(3);;
gap> Order(G);
6
```

1. Order(G)
2. Exponent(G)
3. NilpotencyClassOfGroup(G)
4. CommutatorLength(G)

GAP

```
gap> G:=SymmetricGroup(3);;
gap> Order(G);
6
```

1. Order(G)
2. Exponent(G)
3. NilpotencyClassOfGroup(G)
4. CommutatorLength(G)

GAP

```
gap> G:=SymmetricGroup(3);;
gap> Order(G);
6
gap> Exponent(G);
6
```

1. Order(G)
2. Exponent(G)
3. NilpotencyClassOfGroup(G)
4. CommutatorLength(G)

GAP

```
gap> G:=SymmetricGroup(3);;
gap> Order(G);
6
gap> Exponent(G);
6
gap> G:=DihedralGroup(8);;
gap> NilpotencyClassOfGroup(G);
2
```

1. GroupHomomorphismByImages(G,H, Generator of G, Images)

1. GroupHomomorphismByImages(G,H, Generator of G, Images)
2. Kernel(hom)

1. GroupHomomorphismByImages(G,H, Generator of G, Images)
2. Kernel(hom)
3. Image(hom)

1. GroupHomomorphismByImages(G,H, Generator of G, Images)
2. Kernel(hom)
3. Image(hom)
4. Image(hom,g) or g^hom

1. GroupHomomorphismByImages(G,H, Generator of G, Images)
2. Kernel(hom)
3. Image(hom)
4. Image(hom,g) or g^hom
5. PreImage(hom,h)

1. GroupHomomorphismByImages(G,H, Generator of G, Images)
2. Kernel(hom)
3. Image(hom)
4. Image(hom,g) or g^hom
5. PreImage(hom,h)

GAP

```
gap> G:=SymmetricGroup(4);; H:=SymmetricGroup(3);;
gap> Ggens:= [(1,2),(1,2,3,4)];;Hgens:=[(1,2),[(1,3)];
gap> hom:=GroupHomomorphismByImages(G,H,Ggens,Hgens);
gap> Image(hom,(1,2,4,3));
(2,3)
```

1. GroupHomomorphismByImages(G,H, Generator of G, Images)
2. Kernel(hom)
3. Image(hom)
4. Image(hom,g) or g^hom
5. PreImage(hom,h)

GAP

```
gap> G:=SymmetricGroup(4);; H:=SymmetricGroup(3);;
gap> Ggens:= [(1,2),(1,2,3,4)];;Hgens:=[(1,2),[(1,3)];
gap> hom:=GroupHomomorphismByImages(G,H,Ggens,Hgens);
gap> Image(hom,(1,2,4,3));
(2,3)
```

1. GroupHomomorphismByImages(G,H, Generator of G, Images)
2. Kernel(hom)
3. Image(hom)
4. Image(hom,g) or g^hom
5. PreImage(hom,h)

GAP

```
gap> G:=SymmetricGroup(4);; H:=SymmetricGroup(3);;
gap> Ggens:= [(1,2),(1,2,3,4)];;Hgens:=[(1,2),[(1,3)];
gap> hom:=GroupHomomorphismByImages(G,H,Ggens,Hgens);
gap> Image(hom,(1,2,4,3));
(2,3)
gap> StructureDescription(Kernel(hom));
"C2 x C2"
gap> StructureDescription(Image(hom));
"S3"
```

1. AutomorphismGroup(G)

1. AutomorphismGroup(G)
2. InnerAutomorphism(G, g)

1. AutomorphismGroup(G)
2. InnerAutomorphism(G, g)
3. InnerAutomorphismsAutomorphismGroup(autgroup)

1. AutomorphismGroup(G)
2. InnerAutomorphism(G, g)
3. InnerAutomorphismsAutomorphismGroup(autgroup)
4. IsomorphismGroups(G, H)

1. AutomorphismGroup(G)
2. InnerAutomorphism(G, g)
3. InnerAutomorphismsAutomorphismGroup(autgroup)
4. IsomorphismGroups(G, H)
5. AllHomomorphisms(G, H)

1. AutomorphismGroup(G)
2. InnerAutomorphism(G, g)
3. InnerAutomorphismsAutomorphismGroup(autgroup)
4. IsomorphismGroups(G, H)
5. AllHomomorphisms(G, H)
6. AllEndomorphisms(G)

1. AutomorphismGroup(G)
2. InnerAutomorphism(G, g)
3. InnerAutomorphismsAutomorphismGroup(autgroup)
4. IsomorphismGroups(G, H)
5. AllHomomorphisms(G, H)
6. AllEndomorphisms(G)
7. AllAutomorphisms(G)

1. AutomorphismGroup(G)
2. InnerAutomorphism(G, g)
3. InnerAutomorphismsAutomorphismGroup(autgroup)
4. IsomorphismGroups(G, H)
5. AllHomomorphisms(G, H)
6. AllEndomorphisms(G)
7. AllAutomorphisms(G)

GAP

```
gap> G:=AlternatingGroup(5);;
gap> Aut:=AutomorphismGroup(G);;
gap> StructureDescription(Aut);
S5
```

1. AutomorphismGroup(G)
2. InnerAutomorphism(G, g)
3. InnerAutomorphismsAutomorphismGroup(autgroup)
4. IsomorphismGroups(G, H)
5. AllHomomorphisms(G, H)
6. AllEndomorphisms(G)
7. AllAutomorphisms(G)

GAP

```
gap> G:=AlternatingGroup(5);;
gap> Aut:=AutomorphismGroup(G);;
gap> StructureDescription(Aut);
S5
```

1. AutomorphismGroup(G)
2. InnerAutomorphism(G, g)
3. InnerAutomorphismsAutomorphismGroup(autgroup)
4. IsomorphismGroups(G, H)
5. AllHomomorphisms(G, H)
6. AllEndomorphisms(G)
7. AllAutomorphisms(G)

GAP

```
gap> G:=AlternatingGroup(5);;
gap> Aut:=AutomorphismGroup(G);;
gap> StructureDescription(Aut);
S5
gap> Inn:=InnerAutomorphismsAutomorphismGroup(Aut);
gap> StructureDescription(Inn);
A5
```

```
for x in [objects] do
    commands
od;
```

```
for x in [objects] do
    commands
od;

while expression do
    commands
od;
```

```
for x in [objects] do
    commands
od;

while expression do
    commands
od;

repeat
    commands
until expression;
```

for x in [objects] do
    commands
od;

while expression do
    commands
od;

repeat
    commands
until expression;

Notepad - MyProgram.g

```
n:=4;;
i:=0;;
G:=SmallGroup(n,i+1);
while IsSimpleGroup(G)=false do
 i:=(i+1) mod NumberSmallGroups(n);
 n:=n+2*Maximum(1-i,0);
 G:=SmallGroup(n,i+1);
od;
Print(StructureDescription(G));
```

for x in [objects] do
    commands
od;

while expression do
    commands
od;

repeat
    commands
until expression;

Notepad - MyProgram.g

```
n:=4;;
i:=0;;
G:=SmallGroup(n,i+1);
while IsSimpleGroup(G)=false do
 i:=(i+1) mod NumberSmallGroups(n);
 n:=n+2*Maximum(1-i,0);
 G:=SmallGroup(n,i+1);
od;
Print(StructureDescription(G));
```

GAP

```
gap> Read("c:/MyProgram.g");
A5
```

```
If expression 1 then
    commands
elif expression 2 then
    commands
        ⋮
elif expression n then
    commands
else
    commands
fi;
```

If expression 1 then
    commands
elif expression 2 then
    commands
    ⋮
elif expression n then
    commands
else
    commands
fi;

Notepad - MyProgram.g

```
G:=AlternatingGroup(5);
counter:=0;
for x in G do
 for y in G do
  if x*y=y*x then
   counter:=counter+1;
  fi;
 od;
od;
Print(counter/Order(G)^2);
```

If expression 1 then
    commands
elif expression 2 then
    commands
    ⋮
elif expression n then
    commands
else
    commands
fi;

Notepad - MyProgram.g

```
G:=AlternatingGroup(5);
counter:=0;
for x in G do
 for y in G do
  if x*y=y*x then
   counter:=counter+1;
  fi;
 od;
od;
Print(counter/Order(G)^2);
```

GAP

```
gap> Read("c:/MyProgram.g");
1/12
```

```
name:=function(arguments)
  local variables;
    commands
        ⋮
    commands
  return results;
end;
```

name:=function(arguments)
  local variables;
    commands
    ⋮
    commands
  return results;
end;

Notepad - MyProgram.g

```
PComm:=function(G)
local x,y,counter;
counter:=0;
for x in G do
 for y in G do
  if x*y=y*x then
   counter:=counter+1;
  fi;
 od;
od;
return counter/Order(G)^2;
end;
```

```
name:=function(arguments)
  local variables;
    commands
        ⋮
    commands
  return results;
end;
```

Notepad - MyProgram.g

```
PComm:=function(G)
local x,y,counter;
counter:=0;
for x in G do
 for y in G do
  if x*y=y*x then
   counter:=counter+1;
  fi;
 od;
od;
return counter/Order(G)^2;
end;
```

GAP

```
gap> Read("c:/MyProgram.g");
gap> PComm(AlternatingGroup(5));
1/12
```

Lists

1. List[i]

Lists

1. List[i]
2. Number(L)

Lists

1. List[i]
2. Number(L)
3. Add(L,l)

Lists

1. List[i]
2. Number(L)
3. Add(L,l)
4. Remove(L,position)

Lists

1. List[i]
2. Number(L)
3. Add(L,l)
4. Remove(L,position)
5. Unique(L)

Lists

1. List[i]
2. Number(L)
3. Add(L,l)
4. Remove(L,position)
5. Unique(L)
6. Sort(L)

Lists

1. List[i]
2. Number(L)
3. Add(L,l)
4. Remove(L,position)
5. Unique(L)
6. Sort(L)
7. Append(L1,L2)

Lists

1. List[i]
2. Number(L)
3. Add(L,l)
4. Remove(L,position)
5. Unique(L)
6. Sort(L)
7. Append(L1,L2)
8. Concatelation(L1,L2)

Lists

1. List[i]
2. Number(L)
3. Add(L,l)
4. Remove(L,position)
5. Unique(L)
6. Sort(L)
7. Append(L1,L2)
8. Concatelation(L1,L2)
9. Filtered(L,x->statement)

Lists

1. List[i]
2. Number(L)
3. Add(L,l)
4. Remove(L,position)
5. Unique(L)
6. Sort(L)
7. Append(L1,L2)
8. Concatelation(L1,L2)
9. Filtered(L,x->statement)
10. Intersection(L1,L2)

Lists

1. List[i]
2. Number(L)
3. Add(L,l)
4. Remove(L,position)
5. Unique(L)
6. Sort(L)
7. Append(L1,L2)
8. Concatelation(L1,L2)
9. Filtered(L,x->statement)
10. Intersection(L1,L2)

GAP

```
gap> L1:=[2,3,5];;
gap> L1[3];
5
```

Lists

1. List[i]
2. Number(L)
3. Add(L,l)
4. Remove(L,position)
5. Unique(L)
6. Sort(L)
7. Append(L1,L2)
8. Concatelation(L1,L2)
9. Filtered(L,x->statement)
10. Intersection(L1,L2)

GAP

```
gap> L1:=[2,3,5];;
gap> L1[3];
5
```

Lists

1. List[i]
2. Number(L)
3. Add(L,l)
4. Remove(L,position)
5. Unique(L)
6. Sort(L)
7. Append(L1,L2)
8. Concatelation(L1,L2)
9. Filtered(L,x->statement)
10. Intersection(L1,L2)

GAP

```
gap> L1:=[2,3,5];;
gap> L1[3];
5
gap Add(L1,7);L1;
[ 2, 3, 5, 7 ]
```

Lists

1. List[i]
2. Number(L)
3. Add(L,l)
4. Remove(L,position)
5. Unique(L)
6. Sort(L)
7. Append(L1,L2)
8. Concatelation(L1,L2)
9. Filtered(L,x->statement)
10. Intersection(L1,L2)

GAP

```
gap> L1:=[2,3,5];;
gap> L1[3];
5
gap Add(L1,7);L1;
[ 2, 3, 5, 7 ]
gap> L2:=[11,13,17,19];;
gap> Append(L1,L2);L1;
[ 2, 3, 5, 7, 11, 13, 17, 19 ]
gap> Number(L1);
8
```

Lists

1. List[i]
2. Number(L)
3. Add(L,l)
4. Remove(L,position)
5. Unique(L)
6. Sort(L)
7. Append(L1,L2)
8. Concatelation(L1,L2)
9. Filtered(L,x->statement)
10. Intersection(L1,L2)

GAP

```
gap> L1:=[2,3,5];;
gap> L1[3];
5
gap Add(L1,7);L1;
[ 2, 3, 5, 7 ]
gap> L2:=[11,13,17,19];;
gap> Append(L1,L2);L1;
[ 2, 3, 5, 7, 11, 13, 17, 19 ]
gap> Number(L1);
8
gap> Filtered(L1,x->x mod 3=1);
[ 7, 13, 19 ]
```

Lists

1. List[i]
2. Number(L)
3. Add(L,l)
4. Remove(L,position)
5. Unique(L)
6. Sort(L)
7. Append(L1,L2)
8. Concatelation(L1,L2)
9. Filtered(L,x->statement)
10. Intersection(L1,L2)

GAP

```
gap> L1:=[2,3,5];;
gap> L1[3];
5
gap Add(L1,7);L1;
[ 2, 3, 5, 7 ]
gap> L2:=[11,13,17,19];;
gap> Append(L1,L2);L1;
[ 2, 3, 5, 7, 11, 13, 17, 19 ]
gap> Number(L1);
8
gap> Filtered(L1,x->x mod 3=1);
[ 7, 13, 19 ]
gap> Intersection(L1,[5,6,7,8,9]);
[ 5, 7 ]
```

Shorten functions

GAP

```
gap> Q:=QuaternionGroup;;
gap> Aut:=AutomorphismGroup;;
gap> SD:=StructureDescription;;
gap> G:=Q(8);;
gap> SD(Aut(G));
S4
```

Shorten functions

GAP

```
gap> Q:=QuaternionGroup;;
gap> Aut:=AutomorphismGroup;;
gap> SD:=StructureDescription;;
gap> G:=Q(8);;
gap> SD(Aut(G));
S4
```

Shorten functions

GAP

```
gap> Q:=QuaternionGroup;;
gap> Aut:=AutomorphismGroup;;
gap> SD:=StructureDescription;;
gap> G:=Q(8);;
gap> SD(Aut(G));
S4
gap> SG:=SmallGroup;;
gap> SD(SG(81,15));
"C3 x C3 x C3 x C3"
gap> SD(Aut(SG(81,15)));
"GL(4,3)"
```

gaprc

Write your own commands and functions that will be run at the begining of GAP

Notepad - C:/gap4r7/gaprc

```
SG:=SmallGroup;
NrSG:=NumberSmallGroups;
D:=DihedralGroup;
Q:=QuaternionGroup;
S:=SymmetricGroup;
A:=AlternatingGroup;
F:=FreeGroup;
Aut:=AutomorphismGroup;
Inn:=InnerAutomorphismsAutomorphismGroup;
SD:=StructureDescription;
```

Thank You for Your Patience!