

LogTo("filename"); Log a session
Read("filename"); Run commands in file
?command Look up help for *command*
Tab Command completion
Input line uses some emacs style keybindings

Operators and Keywords

Comment Delimiter
:= Assignment
=, <>, >, <, <=, >= Comparison
+, -, *, /, mod, ^ Arithmetic
and, or, not Logical

Basic Data Types

Foo_Bar1; Variable (long names ok)
x:= [1, 2, 3]; List
 access: x[1]; or x{[1,2]};
x:= Set([1, 2, 3]); Set
x:= (1, 2, 3); Permutation
x:= rec(Foo := 1, bar := 2); Record
 access: x.Foo or x.("bar");
true, false Boolean
Unbind(var); Clear var from memory

Display and Output

View(objects); Short form
Print(objects); GAP readable form
Display(objects); Long human-readable form
PrintArray(matrix); Pretty print *matrix*

Functions

```
MyFunc := function(x, y)
  local A, b;
  ...
  return result;
end;

# use arg for functions of many arguments
MySum := function( arg )
  local sum, x; sum := 0;
  for x in arg do sum := sum + x; od;
  return sum;
end;
```

MySquare := x -> x^2;

Loops and Conditionals

break Exit any loop
continue Move to next loop iteration

```
if cond then ...
elif cond then ...
else ... fi;
```

```
for var in list do ... od;
for var in object do ... od;
```

while *cond* do ... od;

repeat ... until *cond*;

L:= List(list, func); New list of *func* values
Apply(list, func); Change values of *list*

Lists

x:= [1, 2, 3]; List
x:= [1..2000]; List
 access: x[1]; or x{[1..5]};
Add(L, x); Add x to end of L
L:= Concatenation(L₁, L₂, ...); Join lists
Append(L, L₁, ...); Changes L
L:= Flat(list); Flatten nested lists
L:= Reversed(list);
n:= Length(L);
b:= IsEmpty(L);
L:= List(list, func); New list of *func* values
Apply(list, func); Alter values of *list*
L:= Filtered(list, func); Elems satisfying *func*

Sets

x:= Set([1, 2, 3]); Set
object in set; Test membership
UniteSet(A, B); Changes A
IntersectSet(A, B); Changes A
SubtractSet(A, B); Changes A
These next four also work for objects such as lists and groups. The objects of *C* will be identical to the objects of *A* and *B*.
b:= IsSubset(A, B); $B \subseteq A$?
C:= Union(A, B, ...);
C:= Intersection(A, B, ...);
C:= Difference(A, B); $A - B$

The GAP program and full manual can be found at:
<http://www.gap-system.org/>
<http://www-gap.dcs.st-and.ac.uk/~gap/>

Matrices

M:= [[1,2], [0,1]]; Matrix
Matrices typically have their own functions, RankMat(M), DeterminantMat(M), DimensionsMat(M), TransposedMat(M), NullspaceMat(M), Inverse(M), Also “Destructive” forms: RankMatDestructive, ...
PrintArray(matrix); Pretty print *matrix*
M:= IdentityMat(n); $n \times n$ identity matrix
M:= RandomMat(n, m, Ring);
M:= RandomInvertibleMat(n, Ring);
M:= DiagonalMat(vec); V a vector
M:= PermutationMat(perm, dim);
x:= SolutionMat(mat, b); $xM = b$
M:= TriangulizeMat(mat);
M:= BaseMat(mat); row space
D:= JordanDecomposition(mat);
New in GAP 4.4: NullspaceIntMat, SolutionIntMat, ComplementIntMat, DiagonalizeIntMat, ...

Permutations

x:= (1, 2, 3); Permutation
a:= 3^x; Action on a point
L:= OnTuples(list, perm); Action on a list
L:= ListPerm(perm); Permutation \rightarrow list
p:= PermList(list); List \rightarrow permutation
a:= SignPerm(perm); ± 1
L:= MovedPoints(perm);

Numerical

IsInt, isRat, IsCyc Are the filters which test for integers, rationals, and finite cyclotomic numbers.
infinity; ∞
a:= E(n); Primitive *n*-th root of unity
a:= AbsoluteValue(num);
a:= SignInt(num); ± 1 (rationals allowed)
a:= GcdInt(int, int);
a:= Sqrt(num); Given as a sum of E(i)’s

Number Theory

IsEvenInt, IsPrimeInt, ... Filters
int mod int
list:= FactorsInt(int);
o:= OrderMod(n, m); Order of *n* mod *m*
n:= Legendre(int, int);
n:= MoebiusMu(int);

Combinatorics

Binomial(n, k); $\binom{n}{k}$
 Factorial(int);

The following with second parameter k restricts to length k results. Use `Nr_____` for simple count.

list:= Combinations($list$); Subsets
 list:= Arrangements($list$);

Subset permutations

list:= PartitionsSet($list$); Partitions
 list:= Tuples($list, k$); Elements of $list^k$

Built-in Groups

Type: ?AbelianGroup for more info and types

G:= AbelianGroup($list$); $\bigoplus_{n \in list} \mathbb{Z}_n$
 G:= FreeGroup(int);
 G:= SymmetricGroup(int);
 G:= AlternatingGroup(int);
 G:= DihedralGroup(int);

Elements

$elem$ in $object$; Test membership
 n:= Order($elem$); $One(elem) = elem^n$
 i:= Inverse($elem$);
 x:= Comm($elem, elem$); Commutator $a^{-1}b^{-1}ab$
 matrix:= MultiplicationTable($elems$);
 x:= Factorization($gens, elem$);

M:= Maximum($list$ or $elems$);
 m:= Minimum($list$ or $elems$);
 p:= Product($list$ or $elems$);
 s:= Sum($list$ or $elems$);

SetName($object, name$); Set display string

Words

b:= IsWord($object$); Word in a free object?
 w:= UnderlyingElement($elem$); $elem \rightarrow word$
 a:= ElementOfFpGroup(FamilyObj($elem$), $word$);
Make $word$ an element in same group as $elem$
See Also: MappedWord
 list:= LetterRepAssocWord($word$);
 $word \rightarrow$ list of indices

a:= AssocWordByLetterRep(FamilyObj($elem$), $list$);
List of indices \rightarrow word of same type as $elem$

Groups

$object$ in grp ; Test membership
 G.1, G.2, ... Generators of group G
 G:= Group($elems$); Construct by generators
 H:= Subgroup($grp, elems$);
 n:= Index($grp, subgroup$);
 list:= ConjugateSubgroups($grp, subgroup$);
 F:= FreeGroup(2); Free group for next line
 G:= F/[F.1^4, F.1^-2*F.2^3]; Quotient group
 G:= DirectProduct(G_1, G_2, \dots);
 n:= Size(grp);
 list:= Elements(grp);
 list:= GeneratorsOfGroup(grp);
 e:= One(grp or $element$); Get identity
 b:= IsAbelian(grp); Testing a filter
 list:= AbelianInvariants(grp); Sig. of G^{ab}
 list:= LowIndexSubgroupsFpGroup($grp, index$);
 matrix:= CosetTable($grp, subgroup$);
 C:= ConjugacyClass($grp, element$);
 list:= ConjugacyClasses(grp);
 list:= ChiefSeries(grp);
 list:= CompositionSeries(grp);
 list:= LowerCentralSeriesOfGroup(grp);

Group Homomorphisms

Assume each line begins with “f:= ”.

GroupHomomorphismByImages($G, H, gens, imgs$);
 GroupHomomorphismByFunction($G, H, func$);
 NaturalHomomorphism(G, N); $f: G \rightarrow G/N$
 MaximalAbelianQuotient(G); $f: G \rightarrow G^{ab}$
 ConjugatorIsomorphism(G, g); $f(h) = ghg^{-1}$
 IsomorphismGroups(G, H); Find iso: $G \rightarrow H$
 H:= Kernel(f);
 H:= Image(f);
 H:= Image($f, subgroup$ or $element$);
 H:= PreImage($f, subgroup$ or $element$);
 H:= CoKernel(f); Nonstandard! $h = f(1)$
 G:= AutomorphismGroup($object$);
 H:= AutomorphismDomain($aut grp$);

Group Conversions

Each gives $iso: grp \xrightarrow{\cong} H$ with H of shown type
 iso:= IsomorphismFpGroup(grp);
 iso:= IsomorphismPcGroup(grp);
 iso:= IsomorphismPermGroup(grp);
 H:= SimplifiedFpGroup(grp); Do Tietze moves

Identifying/Canonicalizing Groups

A “small” group generally has less than 2000 elements, though there are exceptions both ways.

id:= IdGroup($group$); Identify a group
 G:= SmallGroup(id); Retrieve group by id
 L:= AllSmallGroups($order$ or $properties$);
 n:= NumberSmallGroups($order$);
 SmallGroupsInformation($order$);

Filters

L:= KnownPropertiesOfObject($object$);
Filters whose value is known on $object$
 L:= KnownTruePropertiesOfObject($object$);
Filters the $object$ will satisfy

b:= filter($object$); true if $object$ has $filter$
 Some example filters: IsCommutative, IsFinite, IsFpGroup, IsFreeAbelian, IsFreeGroup, IsInt, IsMatrix, IsMutable, IsNormal, IsOne, IsPrime, IsSolvableGroup, IsSortedList, IsSurjective, IsTrivial