In general, a user does not want only a composition tree, but structural information, such as a list of subgroups.

The approach to compute these is the *Solvable Radical* (or *Trivial Fitting*) method, developed over the last 15-20 years *(HOLT, CANNON; H.)*
Structural Calculations
Radical Series

Let $R=\text{Rad}(G)$. Then $G/R$ has no solvable (or nilpotent, thus trivial Fitting) normal subgroup.

Let $R<S^*<G$ so that $S^*/R = \text{Soc}(G/R)$. We have that $S^*/R$ is a direct product of (nonabelian) simple groups.

$G$-conjugation permutes simple factors, kernel is $Pker$.

Action on $S^*/R$ is $\rho:G \to \text{Aut}(S^*/R) = \text{Aut}(T_1 \times \ldots \times T_k)$ with $T_i$ simple, nonabelian. Its kernel is $R$.

$T_i$ are (amongst) nonabelian factors, compute $\rho$ acting on comp. tree, representing the image as subgroup of (permutation or matrix group) $\times_i \text{Aut}(T_i) \triangledown S_{k_i}$. 
Radical Series

Let $R = \text{Rad}(G)$. Then $G/R$ has no solvable (or nilpotent, thus trivial Fitting) normal subgroup.

Let $R < S^* < G$ so that $S^*/R = \text{Soc}(G/R)$. We have that $S^*/R$ is a direct product of (nonabelian) simple groups.

$G$-conjugation permutes simple factors.

Action on $S^*/R$ is $\rho: G \to \text{Aut}(S^*/R)$.

with $T_i$ simple, nonabelian. Its kernel is $R$.

$T_i$ are (amongst) nonabelian factors, compute $\rho$ acting on comp. tree, representing the image as subgroup of (permutation or matrix group) $\times_i \text{Aut}(T_i) \wr S_{k_i}$.

Solvable Radical — largest solvable normal subgroup
Let \( R = \text{Rad}(G) \). Then \( G/R \) has no solvable (or nilpotent, thus \textit{trivial Fitting}) normal subgroup.

Let \( R < S^* < G \) so that \( S^*/R = \text{Soc}(G/R) \). We have that \( S^*/R \) is a direct product of (nonabelian) simple groups.

\( G \)-conjugation permutes simple factors, kernel is \( P\ker \).

Action on \( S^*/R \) is \( \rho: G \rightarrow \text{Aut}(S^*/R) = \text{Aut}(T_1 \times \ldots \times T_k) \) with \( T_i \) simple, nonabelian. Its kernel is \( R \).

\( T_i \) are (amongst) nonabelian factors, compute \( \rho \) acting on comp. tree, representing the image as subgroup of (permutation or matrix group) \( \times_i \text{Aut}(T_i) : \text{S}_{k_i} \).
Radical Series

Let $R=\text{Rad}(G)$. Then $G/R$ has no solvable (or nilpotent, thus *trivial Fitting* normal subgroup.

Let $R<S^*<G$ so that $S^*/R = \text{Soc}(G/R)$. We have that $S^*/R$ is a direct product of (nonabelian) simple groups.

$G$-conjugation permutes simple factors, kernel is $P\ker$.

Action on $S^*/R$ is $\rho: G \to \text{Aut}(S^*/R) = \text{Aut}(T_1 \times \ldots \times T_k)$ with $T_i$ simple, nonabelian. Its kernel is $R$.

$T_i$ are (amongst) nonabelian factors of $G$.

Collect isomorphic simple factors together, as $G$ can not mix these.

$\text{Aut}(T_1 \times \ldots \times T_k) = \bigtimes_i \text{Aut}(T_i) \wr S_{k_i}$

$G/P\ker$ is intransitive direct product of the $S_{k_i}$
Radical Series

Let \( R = \text{Rad}(G) \). Then \( G/R \) has no solvable (or nilpotent, thus trivial Fitting) normal subgroup.

Let \( R < S^* < G \) so that \( S^*/R = \text{Soc}(G/R) \). We have that \( S^*/R \) is a direct product of (nonabelian) simple groups.

\( G \)-conjugation permutes simple factors, kernel is \( P\text{ker} \).

Action on \( S^*/R \) is \( \rho: G \to \text{Aut}(S^*/R) = \text{Aut}(T_1 \times \ldots \times T_k) \) with \( T_i \) simple, nonabelian. Its kernel is \( R \).

\( T_i \) are (amongst) nonabelian factors, compute \( \rho \) acting on comp. tree, representing the image as subgroup of (permutation or matrix group) \( \times_i \text{Aut}(T_i) \triangleright S_k \).
Using Solvable Radical

Choose a series of $G$-normal subgroups

$$R = R_0 > R_1 > R_2 > \ldots > R_i = \langle 1 \rangle.$$ 

with $R_i/R_{i+1} \cong C_{p_i}^{n_i}$. (elementary abelian, $R$ is solvable.)

Then each factor $R_i/R_{i+1} \cong GF(p_i)^{n_i}$ can be interpreted as a vector space over a finite field, use linear algebra.

For this we need bases of the vector spaces.

(Use subgroups, elements of $G$ to represent factor groups.)
Lifting Paradigm

A Solvable-Radical algorithm now first computes the desired result in $G/R$, using its special structure and the fact that it is (comparatively) small.

Step by step, then lift the result from $G/R_i$ to $G/R_{i+1}$, using the fact that it is an elementary abelian normal subgroup. Usually this involves orbit calculations / linear algebra on $R_i/R_{i+1}$.

Use subgroups, elements of $G$ to represent factor group subgroups/elements.

**Describe single factor step with $N \triangleleft G$ elem. abelian.**

**Now:** $G \triangleright N$ elementary abelian. Result known for $G/N$. 
N-orbits

When computing group actions, use that the orbits of normal subgroup $N$ form blocks.

Instead of stabilizing a point $\omega$ under subgroup $U \supseteq N$, first stabilize the corresponding block $\Delta = \omega^N$. Let $A = \text{Stab}_U(\Delta)$ be this stabilizer.

Then $\forall \ a \in A, \ \exists \ n \in N$ such that $\omega^a = \omega^n$, so corrected element $a/n \in \text{Stab}_U(\omega)$ in same $N$-coset.

Thus $\text{Stab}_U(\omega)$ is generated by $\text{Stab}_N(\omega)$ with corrections of generators of $A$. 
Example: Conjugacy Classes

$G$- orbits under action of $G$ on itself by conjugation.
For each class store representative $x$ and centralizer $C_G(x)$.

**Step 1:** Almost simple group $T \triangleleft G \triangleleft \text{Aut}(T)$. Look up in predefined table (such as ATLAS webpages), or pick elements randomly, test for conjugacy.

**Step 2:** Direct Product $T_1 \times \cdots \times T_k$. Form cartesian product of conjugacy classes of factors. Same if automorphisms in factors.

**Step 3:** $T_1 \times \cdots \times T_k \triangleleft G \leq \text{Aut}(T_1 \times \cdots \times T_k)$. Fusion of conjugacy classes under component permutation, conj. tests.

**Assume:** Class representatives, centralizers for $G/N$ given.
Example: Conjugacy Classes

$G$-orbits under action of $G$ on itself by conjugation.
For each class store representative $x$ and centralizer $C_G(x)$.

**Step 1:** Almost simple group $T \triangleleft G \triangleleft \text{Aut}(T)$. Look up in predefined table (such as ATLAS webpages), or pick elements randomly, test for conjugacy.

**Step 2:** Direct Product of conjugacy classes

**Step 3:** $T_1 \times \ldots \times T_k$ classes under conjugacy.

**Assume:** Class representatives

---

ATLAS of Finite Group Representations - Version 3

About this ATLAS

This ATLAS of Group Representations has been prepared by Robert Wilson, Peter Walsh, Jonathan Tripp, Ibrahim Suleiman, Richard Parker, Simon Norton, Simon Nickerson, Steve Linton, John Bray, and Rachel Abbott (in reverse alphabetical order, because I'm fed up with always being last!)

Navigation
- Group families
  - Sporadic groups
  - Alternating groups
  - Linear groups
  - Classical groups
  - Exceptional groups of Lie type
  - Exceptional groups of small rank
  - Small simple groups
  - Small simple groups (character tables)
  - Small simple groups (automorphism groups)
  - Small simple groups (maximal subgroups)
  - Small simple groups (maximal subgroups of index 2)
  - Small simple groups (maximal subgroups of index 3)
  - Small simple groups (maximal subgroups of index 4)
  - Small simple groups (maximal subgroups of index 5)
  - Small simple groups (maximal subgroups of index 6)
  - Small simple groups (maximal subgroups of index 7)
  - Small simple groups (maximal subgroups of index 8)
  - Small simple groups (maximal subgroups of index 9)
  - Small simple groups (maximal subgroups of index 10)
  - Small simple groups (maximal subgroups of index 11)
  - Small simple groups (maximal subgroups of index 12)
  - Small simple groups (maximal subgroups of index 13)
  - Small simple groups (maximal subgroups of index 14)
  - Small simple groups (maximal subgroups of index 15)
  - Small simple groups (maximal subgroups of index 16)
  - Small simple groups (maximal subgroups of index 17)
  - Small simple groups (maximal subgroups of index 18)
  - Small simple groups (maximal subgroups of index 19)
  - Small simple groups (maximal subgroups of index 20)
  - Small simple groups (maximal subgroups of index 21)
  - Small simple groups (maximal subgroups of index 22)
  - Small simple groups (maximal subgroups of index 23)
  - Small simple groups (maximal subgroups of index 24)
  - Small simple groups (maximal subgroups of index 25)
  - Small simple groups (maximal subgroups of index 26)
  - Small simple groups (maximal subgroups of index 27)
  - Small simple groups (maximal subgroups of index 28)
  - Small simple groups (maximal subgroups of index 29)
  - Small simple groups (maximal subgroups of index 30)
  - Small simple groups (maximal subgroups of index 31)
  - Small simple groups (maximal subgroups of index 32)
  - Small simple groups (maximal subgroups of index 33)
  - Small simple groups (maximal subgroups of index 34)
  - Small simple groups (maximal subgroups of index 35)
  - Small simple groups (maximal subgroups of index 36)
  - Small simple groups (maximal subgroups of index 37)
  - Small simple groups (maximal subgroups of index 38)
  - Small simple groups (maximal subgroups of index 39)
  - Small simple groups (maximal subgroups of index 40)
  - Small simple groups (maximal subgroups of index 41)
  - Small simple groups (maximal subgroups of index 42)
  - Small simple groups (maximal subgroups of index 43)
  - Small simple groups (maximal subgroups of index 44)
  - Small simple groups (maximal subgroups of index 45)
  - Small simple groups (maximal subgroups of index 46)
  - Small simple groups (maximal subgroups of index 47)
  - Small simple groups (maximal subgroups of index 48)
  - Small simple groups (maximal subgroups of index 49)
  - Small simple groups (maximal subgroups of index 50)
  - Small simple groups (maximal subgroups of index 51)
  - Small simple groups (maximal subgroups of index 52)
  - Small simple groups (maximal subgroups of index 53)
  - Small simple groups (maximal subgroups of index 54)
  - Small simple groups (maximal subgroups of index 55)
  - Small simple groups (maximal subgroups of index 56)
  - Small simple groups (maximal subgroups of index 57)
  - Small simple groups (maximal subgroups of index 58)
  - Small simple groups (maximal subgroups of index 59)
  - Small simple groups (maximal subgroups of index 60)
  - Small simple groups (maximal subgroups of index 61)
  - Small simple groups (maximal subgroups of index 62)
  - Small simple groups (maximal subgroups of index 63)
  - Small simple groups (maximal subgroups of index 64)
  - Small simple groups (maximal subgroups of index 65)
  - Small simple groups (maximal subgroups of index 66)
  - Small simple groups (maximal subgroups of index 67)
  - Small simple groups (maximal subgroups of index 68)
  - Small simple groups (maximal subgroups of index 69)
  - Small simple groups (maximal subgroups of index 70)
  - Small simple groups (maximal subgroups of index 71)
  - Small simple groups (maximal subgroups of index 72)
  - Small simple groups (maximal subgroups of index 73)
  - Small simple groups (maximal subgroups of index 74)
  - Small simple groups (maximal subgroups of index 75)
  - Small simple groups (maximal subgroups of index 76)
  - Small simple groups (maximal subgroups of index 77)
  - Small simple groups (maximal subgroups of index 78)
  - Small simple groups (maximal subgroups of index 79)
  - Small simple groups (maximal subgroups of index 80)
  - Small simple groups (maximal subgroups of index 81)
  - Small simple groups (maximal subgroups of index 82)
  - Small simple groups (maximal subgroups of index 83)
  - Small simple groups (maximal subgroups of index 84)
  - Small simple groups (maximal subgroups of index 85)
  - Small simple groups (maximal subgroups of index 86)
  - Small simple groups (maximal subgroups of index 87)
  - Small simple groups (maximal subgroups of index 88)
  - Small simple groups (maximal subgroups of index 89)
  - Small simple groups (maximal subgroups of index 90)
  - Small simple groups (maximal subgroups of index 91)
  - Small simple groups (maximal subgroups of index 92)
  - Small simple groups (maximal subgroups of index 93)
  - Small simple groups (maximal subgroups of index 94)
  - Small simple groups (maximal subgroups of index 95)
  - Small simple groups (maximal subgroups of index 96)
  - Small simple groups (maximal subgroups of index 97)
  - Small simple groups (maximal subgroups of index 98)
  - Small simple groups (maximal subgroups of index 99)
  - Small simple groups (maximal subgroups of index 100)
  - Small simple groups (maximal subgroups of index 101)
  - Small simple groups (maximal subgroups of index 102)
  - Small simple groups (maximal subgroups of index 103)
  - Small simple groups (maximal subgroups of index 104)
  - Small simple groups (maximal subgroups of index 105)
  - Small simple groups (maximal subgroups of index 106)
  - Small simple groups (maximal subgroups of index 107)
  - Small simple groups (maximal subgroups of index 108)
  - Small simple groups (maximal subgroups of index 109)
  - Small simple groups (maximal subgroups of index 110)
  - Small simple groups (maximal subgroups of index 111)
  - Small simple groups (maximal subgroups of index 112)
  - Small simple groups (maximal subgroups of index 113)
  - Small simple groups (maximal subgroups of index 114)
  - Small simple groups (maximal subgroups of index 115)
  - Small simple groups (maximal subgroups of index 116)
  - Small simple groups (maximal subgroups of index 117)
  - Small simple groups (maximal subgroups of index 118)
  - Small simple groups (maximal subgroups of index 119)
  - Small simple groups (maximal subgroups of index 120)
  - Small simple groups (maximal subgroups of index 121)
  - Small simple groups (maximal subgroups of index 122)
  - Small simple groups (maximal subgroups of index 123)
  - Small simple groups (maximal subgroups of index 124)
  - Small simple groups (maximal subgroups of index 125)
  - Small simple groups (maximal subgroups of index 126)
  - Small simple groups (maximal subgroups of index 127)
  - Small simple groups (maximal subgroups of index 128)
  - Small simple groups (maximal subgroups of index 129)
  - Small simple groups (maximal subgroups of index 130)
  - Small simple groups (maximal subgroups of index 131)
  - Small simple groups (maximal subgroups of index 132)
  - Small simple groups (maximal subgroups of index 133)
  - Small simple groups (maximal subgroups of index 134)
  - Small simple groups (maximal subgroups of index 135)
  - Small simple groups (maximal subgroups of index 136)
  - Small simple groups (maximal subgroups of index 137)
  - Small simple groups (maximal subgroups of index 138)
  - Small simple groups (maximal subgroups of index 139)
  - Small simple groups (maximal subgroups of index 140)
  - Small simple groups (maximal subgroups of index 141)
  - Small simple groups (maximal subgroups of index 142)
  - Small simple groups (maximal subgroups of index 143)
  - Small simple groups (maximal subgroups of index 144)
  - Small simple groups (maximal subgroups of index 145)
  - Small simple groups (maxima
Example: Conjugacy Classes

$G$- orbits under action of $G$ on itself by conjugation.
For each class store representative $x$ and centralizer $C_G(x)$.

Step 1: Almost simple group $T < G < \text{Aut}(T)$. Look up in predefined table (such as ATLAS webpages), or pick elements randomly, test for conjugacy.

Step 2: Direct Product $T_1 \times \ldots \times T_k$. Form cartesian product of conjugacy classes of factors. Same if automorphisms in factors.

Step 3: $T_1 \times \ldots \times T_k < G < \text{Aut}(T_1 \times \ldots \times T_k)$. Fusion of conjugacy classes under component permutation, conj. tests.

Assume: Class representatives, centralizers for $G/N$ given.
Conjugacy Classes Lifting

In step, $N \triangleleft G$ elementary abelian. Know classes of

If $g \sim h$ in $G$ then $Ng \sim Nh$ in $G/N$. So each $G/N$-class of (images of) $G$-classes.

Let $Nx$ be class representative with centralizer $C/Nx$.

$c \in C$ acts on elements $nx \in Nx$ by $(nx)c = n^c x^c = n^c [C, x] x$,

that is $c : n \mapsto n^c \cdot [C, x^{-1}]$. This is an affine (linear+translation) action. Orbit/Stab. yield classes within $Nx$ and centralizers.

$N$-orbits correspond to translation by $[m, x^{-1}]$, translation vectors generate a subspace of $N$. The $N$-orbit blocks are cosets of this subspace.
Conjugacy Classes Lifting

In step, $N \triangleleft G$ elementary abelian. Know classes of $G/N$.

If $g \sim h$ in $G$ then $Ng \sim Nh$ in $G/N$. So each $G/N$-class is union of (images of) $G$-classes.

Let $Nx$ be class representative with centralizer $C/N$. Then $c \in C$ acts on elements $nx \in Nx$ by $(nx) \leq n^c \cdot x^c = n^c [c, x^{-1}]x$, that is $c : n \mapsto n^c \cdot [c, x^{-1}]$. This is an affine (linear + translation) action. Orbit/Stab. yield classes within $Nx$ and centralizers. $N$-orbits correspond to translation by $[m, x^{-1}]$, translation vectors generate a subspace of $N$. The $N$-orbit blocks are cosets of this subspace.
SetInfoLevel(InfoHomClass,2);
g:=TransitiveGroup(12,270);
\[2^6\]L(6):2=2wrL(6):2 # L(6) is A5 on 6 points
gap> cl:=ConjugacyClasses(g);
#I 3:60; 1
#I top factor gives conjugating representation, deg 12
#I so far 7 classes computed # for L(6):2
#I abelian factor 2: 64->1 central:false
#I Trying subspace 1 in 6
#I 7 new classes (7 total)
#I 8 new classes (15 total)
#I 9 new classes (24 total)
#I 4 new classes (28 total)
#I 4 new classes (32 total)
#I 3 new classes (35 total)
#I 2 new classes (37 total)
#I Now: 37 classes (37 total)
#I forming classes
[ ()^G, (8,9)^G, (6,7)(8,9)^G, (1,12)(6,7)(8,9)^G, (1,12)
(6,7)(8,9)(10,11)^G, ...]
```
gap> Length(cl);
37

gap> PositionProperty(cl,x-> g.1 in x);
2

gap> PositionProperty(cl,x-> g.2 in x);
25

gap> RepresentativeAction(g,g.2,Representative(cl[25]));
(1,2,5,10)(3,4,11,12)(6,7)(8,9)

gap> RepresentativeAction(g,g.2,Representative(cl[24]));
fail

gap> Collected(List(cl,Size));
[ [1,2], [6,2], [15,2], [20,1], [60,4], [80,2], [120,4], [240,11], [320,2],
  [384,4], [640,3] ]

gap> List(cl,x->PositionProperty(cl,y->Representative(x)^-1 in y));
[1,2,3,4,5,6,7,8,9,12,13,10,11,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37]

gap> PermList(last);
(10,12)(11,13)
```
Example: Subgroups

Assume $N \triangleleft G$ is elementary abelian.

Arbitrary subgroup $U \leq G$. Then $A = \langle N, U \rangle \supseteq N$ and $B = N \cap U \leq N$.

May assume that $A$ is known from prior step. $B$ is subgroup of vector space (and thus easily found).

Also $B \triangleleft U$ and $B \triangleleft N$, so $B \triangleleft A$ and $U/B$ is complement to $N/B$ in $A/B$.

Conjugacy: $A$ up to $G/N$, $B$ up to $N_G(A)$.
Lifting Subgroups

For fixed $A,B$, find all possible $U$:

- $U/B$ is complement to $N/B$ in $A/B$.
- Determine classes of complements: As $N/B$ is vector space this is the one-cohomology group (posh variant of linear algebra).
- Fuse under further conjugation by $N_G(A) \cap N_G(B)$.
gap> a := SL(2, 5);;
gap> a := Image(IsomorphismPermGroup(a));; Size(a);
120

gap> w := WreathProduct(a, Group((1, 2, 3), (1, 2)));;
<permutation group of size 10368000 with 8 generators>
gap> SetInfoLevel(InfoLattice, 1);
gap> u := ConjugacyClassesSubgroups(w);
#I  Found isomorphism A5
#I  Fetching subgroups of simple A5 from table of marks
#I  socle index 6 has 3 factors from 1 types
#I  subdirect level 1 got 9, subdirect level 2 got 113
#I  subdirect level 3 got 1995
#I  extending 1 [...]
#I  extending 1995 [...]
#I  Overall 1212 subgroups
#I  Step 2 : 8 # now run through possible B, test A's
#I  Normal subgroup 1, 1211 subgroups to consider
#I  Normal subgroup 2, 794 subgroups to consider [...]
gap> Length(u);
5356

gap> Size(u[4000]); Size(Representative(u[4000]));
10800 480
Hodge Podge
Input / Output / External Programs

➤ Know already Read, PrintTo

➤ There is ReadCSV and PrintCSV to input/output spreadsheet format — list of records.

➤ Read in a text file — InputTextFile, ReadByte/ReadLine, CloseStream — as strings and thus parse arbitrary data.

➤ Call external programs with Exec(string) — string is passed to sh (Unix, Mac) or command.com (Windows).
Method Selection, Investigation

GAP tries to select the most specific method that can still apply.

```
gap> g:=SymmetricGroup(10);;
gap> TraceMethods(SylowSubgroupOp);
gap> SylowSubgroup(g,2);
#I  SylowSubgroupOp: symmetric at gpprmsya.gi:1567
Group([ (1,2), [...]
```

Code, decision steps: `ApplicableMethod(Operation,args,printlevel)`: returns the GAP function that implements the code.

```
gap> me:=ApplicableMethod(SylowSubgroupOp,[g,2],2);
#I 1: Direct construction for natural GL: 416
#I 2: handled by nice monomorphism: 389
#I 3: pcgs computable groups using special pcgs: 171
#I 4: symmetric: 81
function( G, p ) ... end
```

Higher print level indicates why methods were not used.
Skipping Methods

Can force a ranking change by installing methods with rank offsets.

Method selection will not run tests, but a method can be installed without required prerequisite, do the test, and exit with TryNextMethod(); if it turns out not to apply.

An extra parameter \( k \) to ApplicableMethod gives the \( k \)-th method choice (after \( k-1 \) skips)

```gap
gap> me:=ApplicableMethod(SylowSubgroupOp,[g,2],1,3);
#I 4: symmetric: 81, Skipped
[There would be a method for alternating groups here]
#I 6: permutation groups: 77
#I 8: generic method for group and prime: 54
function( G, p ) ... end
```
Installing Methods

InstallMethod(Op, idstring, famattr, proplist, offset, func)

Examples by looking into the library files (.gi: Installations), though there are some unusual situations for special cases.

There also is InstallOtherMethod to override requirements that were set in the declaration of the operation with DeclareMethod (in .gd file)
Installing Methods

InstallMethod(Op, idstring, famattr, proplist, offset, func)

The identifier for the operation that this method is for – e.g. Order.

Examples by looking into the library files (.gi: Installations), though there are some unusual situations for special cases.

There also is InstallOtherMethod to override requirements that were set in the declaration of the operation with DeclareMethod (in .gd file)
Installing Methods

InstallMethod(Op, idstring, famattr, proplist, offset, func)

A string that will be used by TraceMethods and ApplicableMethod to identify this method.

Examples by looking into the library files (.gi: Installations), though there are some unusual situations for special cases.

There also is InstallOtherMethod to override requirements that were set in the declaration of the operation with DeclareMethod (in .gd file)
Installing Methods

InstallMethod(Op, idstring, famattr, proplist, offset, func)

A list of filters (properties, categories), one for each argument, that are required. For example
(IsPermGroup, IsPosInt].
These are used to calculate the basic method value.

Examples by looking into the library files (.gi: Installations), though there are some unusual situations for special cases.

There also is InstallOtherMethod to override requirements that were set in the declaration of the operation with DeclareMethod (in .gd file)
Installing Methods

InstallMethod(Op, idstring, famattr, proplist, offset, func)

A function that takes the *families* (*how elements fit together, say polynomials over GF(3) vs. GF(5)*) of the arguments and need to return true. Can be simply true, or predefined such as IsCollsElms.

Examples by looking into the library files (.gi: Installations), though there are some unusual situations for special cases.

There also is InstallOtherMethod to override requirements that were set in the declaration of the operation with DeclareMethod (in .gd file)
Installing Methods

InstallMethod(Op, idstring, famattr, proplist, offset, func)

An integer that will change the method rank.

Examples by looking into the library files (.gi: Installations), though there are some unusual situations for special cases.

There also is InstallOtherMethod to override requirements that were set in the declaration of the operation with DeclareMethod (in .gd file)
Installing Methods

InstallMethod(Op, idstring, famattr, proplist, offset, func)

An ordinary GAP function, of as many arguments as were specified by the filters, that will be called to execute the method and will return the result.

Examples by looking into the library files (.gi: Installations), though there are some unusual situations for special cases.

There also is InstallOtherMethod to override requirements that were set in the declaration of the operation with DeclareMethod (in .gd file)
Own Objects

Create your own objects with their own arithmetic:

```plaintext
DeclareCategory("IsMyCategory", IsMultiplicativeElementWithInverse);
MyFam:=NewFamily("MyFam", IsMultiplicativeElementWithInverse,
    IsMultiplicativeElementWithInverse, IsObject);
DeclareRepresentation("IsMyRep", IsPositionalObjectRep and IsMyCategory,
    []);
MyObj:=function(r,i)
    if r=0 and i=0 then Error("zero");fi;
    return Objectify(NewType(MyFam, IsMyCategory and IsMyRep), [r,i]);
end;
InstallMethod(PrintObj, "Identifier String", true, [IsMyRep], 0,
    function(o)
        Print("MyObj", o!1, ",", o!2, ")
    end);
```
InstallMethod(OneOp,"mine",true,[IsMyRep],0,
function (a) return MyObj(1,0); end);

InstallMethod(\ *,"mine",true,[IsMyRep,IsMyRep],0,
function (a,b)
  return MyObj(a![1]*b![1]-a![2]*b![2],a![1]*b![2]+a![2]*b![1]);
end);

InstallMethod(InverseOp,"mine",true,[IsMyRep],0,
function (a)
  local n;
  n:=a![1]^2+a![2]^2; return MyObj(a![1]/n,-a![2]/n);
end);

InstallMethod(\=,"mine",true,[IsMyRep,IsMyRep],0,
function (a,b) return a![1]=b![1] and a![2]=b![2]; end);

InstallMethod(\<,"mine",true,[IsMyRep,IsMyRep],0,
function (a,b) return a![1]<b![1] or (a![1]=b![1] and a![2]<b![2]); end);
gap> Read("myobj.g");
MyObj(0,1)
gap> b:=MyObj(3,2);
MyObj(3,2)
gap> a/b;
MyObj(2/13,3/13)
gap> a^17=a;
true
gap> b^40;
MyObj(-794950134656302081199,-18988330689345532174800)
gap> g:=Group(a);
<group with 1 generators>
gap> Size(g);
4

gap> IsomorphismPcGroup(g);
[ MyObj(0,1) ] -> [ f1 ]
gap> Elements(g);
[ MyObj(-1,0), MyObj(0,-1), MyObj(0,1), MyObj(1,0) ]
Running Through Tuples

Suppose we want to enumerate not objects, but pairs (or tuples) of objects \((A,B,\ldots)\) up to conjugacy.

1. Enumerate \(A\) up to \(G\)-conjugacy and let \(C=C_G(A)\).

2. Then enumerate \(B\) up to \(C\)-conjugacy.

If we know \(B\) up to \(G\)-conjugacy, say \(B\) is a representative with centralizer \(D\), then

- The \(G\)-class of \(B\) corresponds to cosets \(D\backslash G\).
- The \(C\)-classes within this \(G\)-class correspond to double cosets \(D\backslash G/C\).

(This is the underlying tool for \(G\)Quotients and others.)
Subgroups of $S_7$ generated by a $2^3$, and a 4-cycle.

gap> g := SymmetricGroup(7);;


gap> a := (1,2)(3,4)(5,6);;

gap> b := (1,2,3,4);;

gap> c := Centralizer(g, a);
Group([ (1,2), (1,5)(2,6), (3,4), (3,5)(4,6), (5,6) ])

gap> d := Centralizer(g, b);;

gap> dc := DoubleCosets(g, c, d);; Length(dc);
8


gap> bs := List(dc, x -> b ^ Representative(x));;

gap> pairs := List(bs, x -> [ a, x ]);;

gap> subs := List(pairs, x -> Subgroup(g, x));;

gap> List(subs, Size);
[ 8, 120, 8, 24, 40, 5040, 240, 240 ]

gap> RepresentativeAction(g, subs[1], subs[3]);
fail

gap> RepresentativeAction(g, subs[7], subs[8]);
()

gap> sr := List(SubgroupsOrbitsAndNormalizers(g, subs, false),
             x -> x.representative);; # finds reps of the 7 classes
Construction In Parts

Construct all objects combined from two parts, $\Omega$ and $\Delta$, automorphism groups are respectively $G$ and $H$.

Select the joining parts $\omega \in \Omega$, $\delta \in \Delta$ up to action of $G$, let $A = \text{Stab}_G(\omega)$. Ditto $B = \text{Stab}_H(\delta)$.

If $K$ is the group of symmetries of the possible combinations, both $A$ and $B$ induce subgroups.

The $G \times H$ orbits of combinations again correspond to double cosets $A \backslash K / B$. 
Intransitive Groups

Let $U$ be intransitive on $\Omega \cup \Delta$ with projections $\alpha : U \rightarrow A \leq S_\Omega$, $\beta : U \rightarrow B \leq S_\Delta$. Images chosen up to $S_\Omega$, $S_\Delta$ conjugacy.
Intransitive Groups

Let $U$ be intransitive on $\Omega \cup \Delta$ with projections $\alpha: U \to A \leq S_\Omega$, $\beta: U \to B \leq S_\Delta$. Images chosen up to $S_\Omega$, $S_\Delta$ conjugacy.

$D = (\ker \beta) \alpha \triangleleft A$ and $E = (\ker \alpha) \beta \triangleleft B$. 
Intransitive Groups

Let $U$ be intransitive on $\Omega \cup \Delta$ with projections $\alpha: U \to A \leq S_\Omega$, $\beta: U \to B \leq S_\Delta$. Images chosen up to $S_\Omega$, $S_\Delta$ conjugacy.

$D = (\ker \beta)\alpha \triangleleft A$ and $E = (\ker \alpha)\beta \triangleleft B$.

$\chi: A/D \to B/E$ isom. of factor groups
Intransitive Groups

Let $U$ be intransitive on $\Omega_\Delta$ with projections $\alpha: U \to A \leq S_\Omega$, $\beta: U \to B \leq S_\Delta$. Images chosen up to $S_\Omega, S_\Delta$ conjugacy.

$D = (\ker \beta)\alpha < A$ and $E = (\ker \alpha)\beta < B$.

$\chi: A/D \to B/E$ isom. of factor groups

Possible isomorphisms $A/D \to B/E$ correspond to $\text{Aut}(A/D)$, if any exist.
Intransitive Groups

To construct all such groups, select

- $A \leq S_\Omega, B \leq S_\Delta$ up to conjugacy.

- Select $D \triangleleft A$ up to $N_{S_\Omega}(A)$-conjugacy.

- Ditto $E \triangleleft B$ such that $A/D \cong B/E$.

- $\chi$ from representatives of $N(D) \setminus \text{Aut}(A/D) / N(E)$.

- Not covered: Swap $\Omega$ and $\Delta$. 
IntransitiveGroups:=function(A,B)
local gps,SOm,NA,SDe,NB,P,emb,D,qd,f,E,qe,iso,au,gens,auperm,dc,chi,new; 
gps:=[ ]; 
SOm:=SymmetricGroup(MovedPoints(A)); NA:=Normalizer(SOm,A); 
SDe:=SymmetricGroup(MovedPoints(B)); NB:=Normalizer(SDe,B); 
P:=DirectProduct(SOm,SDe); emb:=List([1,2],x->Embedding(P,x)); 
for D in 
SubgroupsOrbitsAndNormalizers(NA,NormalSubgroups(A),false) do 
qd:=NaturalHomomorphismByNormalSubgroup(A,D.representative); 
f:=Image(qd); gens:=GeneratorsOfGroup(f); 
for E in 
SubgroupsOrbitsAndNormalizers(NB,NormalSubgroups(B),false) do 
qe:=NaturalHomomorphismByNormalSubgroup(B,E.representative); 
iso:=IsomorphismGroups(Image(qd),Image(qe)); 
if iso<>fail then 
au:=AutomorphismGroup(f); auperm:=IsomorphismPermGroup(au); 
end if; 
end for; 
end for; 
end for;
for dc in DoubleCosetRepsAndSizes(Image(auperm),
    Subgroup(Image(auperm),
        List(GeneratorsOfGroup(D.normalizer),
            g->ImagesRepresentative(auperm,
                GroupHomomorphismByImages(f,f,gens,
                    List(gens,x->Image(qd,
                        PreImagesRepresentative(qd,x)^g)))))),
    Subgroup(Image(auperm),
        List(GeneratorsOfGroup(E.normalizer),
            g->ImagesRepresentative(auperm,
                GroupHomomorphismByImages(f,f,gens,
                    List(gens,x->PreImagesRepresentative(iso,Image(qe,
                        PreImagesRepresentative(qe,
                            Image(iso,x))^g))))))))) do
    chi:=PreImagesRepresentative(auperm,dc[1])*iso;
# construct one product, embedded into P:
# Generators for A, fused for A/D, together
# with generators for E

```plaintext
new:=List(GeneratorsOfGroup(A),x->Image(emb[1],x)*
    Image(emb[2],PreImagesRepresentative(qe,
    Image(chi,Image(qd,x)))));

Append(new,List(GeneratorsOfGroup(E.representative),
    x->Image(emb[2],x)));

Add(gps,Subgroup(P,new));
```
gap> IntransitiveGroups(SymmetricGroup(4), SymmetricGroup(3));
[ Group([ (1,2,3,4)(5,6), (1,2)(6,7) ]),
  Group([ (1,2,3,4)(6,7), (1,2)(6,7), (5,6,7) ]),
  Group([ (1,2,3,4), (1,2), (5,6,7), (5,6) ]) ]

gap> List(last,Size);
[ 24, 72, 144 ]

gap> as:=AllTransitiveGroups(NrMovedPoints,4);;

gap> bs:=AllTransitiveGroups(NrMovedPoints,3);;

gap> r:=Concatenation(List(Cartesian(as,bs),
  x->IntransitiveGroups(x[1],x[2])));

gap> Collected(List(r,Size));
[ [12,5], [24,7], [36,1], [48,1], [72,3], [144,1] ]

gap> u:=ConjugacyClassesSubgroups(SymmetricGroup(7));;

gap> u:=List(u,Representative);;

gap> s:=Filtered(u,
  x->Set(List(Orbits(x,[1..7]),Length))=[3,4]);

gap> Collected(List(s,Size));
[ [12,5], [24,7], [36,1], [48,1], [72,3], [144,1] ]