Mini-course on GAP – Lecture 2

Leandro Vendramin

Vrije Universiteit Brussel

March 2023

Functions

Let us play with Collatz conjecture. For $n \in \mathbb{N}$ let

$$f(n) = \begin{cases} n/2 & \text{if } n \text{ es even,} \\ 3n+1 & \text{if } n \text{ is odd.} \end{cases}$$

The conjecture is that no matter what number n you start with, there is $m \in \mathbb{N}$ such that $f^m(n) = 1$, where $f^m = f \circ \cdots \circ f$ (m-times). Let us test the conjecture for $n = 5$.

```
gap> f := function(n)
> if n mod 2 = 0 then
> return n/2;
> else
> return 3*n+1;
> fi;
> end;
function( n ) ... end
gap> f(f(f(f(f(5)))));
1
```

An exercise with functions

Write a function that for each n returns the smallest integer m such that $f^m(n) = 1$.

Strings

A string is an expression delimited by the symbol " (Quotation mark):

```
gap> string := "hello world";
hello world
```

To extract one character one uses the expression `string[position]`; to extract substrings `string{positions}`.

```
gap> string[1];
'h'
gap> string[3];
'l'
gap> string{[1,2,3,4,5]};
"hello"
gap> string{[7,8,9,10,11]};
"world"
gap> string{[11,10,9,8,7,6,5,4,3,2,1]};
"dlrow olleh"
```

Strings

There are several functions that allow us to work with strings.
`String` converts anything into a string of characters.

```
gap> String(1234);
"1234"
gap> String(01234);
"1234"
gap> String([1,2,3]);
"[ 1, 2, 3 ]"
gap> String(true);
"true"
```

The function `ReplacedString` replace substrings:

```
gap> ReplacedString("Hello world", "world", "all");
"Hello all"
```

Strings

`Print` allows us to print data in the screen.

```
gap> string := "Hello world";;
gap> Print(string);
Hello world
```

Let us see another example:

```
gap> n := 100;;
gap> m := 5;;
gap> Print(n, " times ", m, " is ", n*m);
100 times 5 is 500
```

Strings

The function `Print` can be used with some special characters. For example, `\n` means "new line".

```
gap> Print("Hello\nworld");
Hello
world
gap> Print("To write \\...");
To write \...
```

The functions `PrintTo` and `AppendTo` work as `Print` but the output goes to a file. It is important to remark that `PrintTo` will overwrite an existing file!

Strings

Some exercises:

1. Write a function that given a list `lst` of words and a letter `x`, returns a sublist of `lst` where every word starts with `x`.
2. Use the function `Permuted` to write a function that shows all the anagrams of a given word.
3. Write a function that given a list of words returns the longest one.

Another exercise:
Play with the functions `JoinStringsWithSeparator`, `SplitString`, `LowercaseString` and `UppercaseString`.

Lists

A list is an ordered sequence of objects (maybe of different type), including empty places.

```
gap> IsList([1, 2, 3]);
true
gap> IsList([1, 2, 3, "abc"]);
true
gap> IsList([1, 2,, "abc"]);
true
gap> 2 in [1, 2, 5, 4, 10];
true
gap> 3 in [0,10,"abc"];
false
```

Lists are written using square brackets!

Lists

Let us create a list with the first six prime numbers. `Size` or `Length` return the number of non-empty elements of the list.

```
gap> primes := [2, 3, 5, 7, 11, 13];
[ 2, 3, 5, 7, 11, 13 ]
gap> Size(primes);
6
```

To access to an element inside a list one should refer to the position.

```
gap> primes[1];
2
gap> primes[2];
3
```

Let us obtain the sublist consisting of the elements in the second, third and fifth position:

```
gap> primes{[2,3,5]};
[ 3, 5, 11 ]
```

Another example (to avoid confusion):

```
gap> list := ["a", "b", "c", "d", "e", "f"];
[ "a", "b", "c", "d", "e", "f" ]
gap> list{[1,3,5]};
[ "a", "c", "e" ]
```

To find elements inside a list one uses `Position`. If the element we are looking for does not belong to the list, `Position` will return `fail`; otherwise it will return the first place where the element appears.

```
gap> Position([5, 4, 6, 3, 7, 3, 7], 5);
1
gap> Position([5, 4, 6, 3, 7, 3, 7], 1);
fail
gap> Position([5, 4, 6, 3, 7, 3, 7], 7);
5
```

Lists

`Add` and `Append` are used to add elements at the end of a list.

```
gap> primes;
[ 2, 3, 5, 7, 11, 13 ]
gap> # Add 19 at the end of the list
gap> Add(primes, 19);
gap> primes;
[ 2, 3, 5, 7, 11, 13, 19 ]
gap> # Add the prime 17 at position 7
gap> Add(primes, 17, 7);
gap> primes;
[ 2, 3, 5, 7, 11, 13, 17, 19 ]
gap> # Add 23 and 29 at the end
gap> Append(primes, [23, 29]);
gap> primes;
[ 2, 3, 5, 7, 11, 13, 17, 19, 23, 29 ]
```

Lists

To remove elements from a list one uses `Remove`.

```
gap> # Remove the first element of the list
gap> l := [10, 20, 30];;
gap> Remove(l, 1);
10
gap> l;
[ 20, 30 ]
```

Lists

`Concatenation` concatenates two or more lists. This function returns a new list consisting of the lists used in the argument.

```
gap> Concatenation([1,2,3],[4,5,6]);
[ 1, 2, 3, 4, 5, 6 ]
```

Is there any difference between `Append` and `Concatenation`? Yes! The function `Concatenation` does not modify the lists used in the argument. `Append` does.

Lists

`Collected` returns a new list where each element of the original list appears with multiplicity. Example:

```
gap> Factors(720);
[ 2, 2, 2, 2, 3, 3, 5 ]
gap> Collected(last);
[ [ 2, 4 ], [ 3, 2 ], [ 5, 1 ] ]
```

Lists

To make a copy of a list one should use the function `ShallowCopy`. The following example shows the difference between `ShallowCopy` and the assignment operator.

```
gap> a := [1, 2, 3, 4];;
gap> b := a;;
gap> c := ShallowCopy(a);;
gap> Add(a, 5);
gap> a;
[ 1, 2, 3, 4, 5 ]
gap> b;
[ 1, 2, 3, 4, 5 ]
gap> c;
[ 1, 2, 3, 4 ]
gap> Add(b, 10);
gap> a;
[ 1, 2, 3, 4, 5, 10 ]
gap> b;
[ 1, 2, 3, 4, 5, 10 ]
```

Lists

The function `Reversed` returns a list containing the elements of our list in reversed order. In the following example the variable `list` will not be modified by the function `Reversed`:

```
gap> list := [2, 4, 7, 3];;
gap> Reversed(list);
[ 3, 7, 4, 2 ]
gap> list;
[ 2, 4, 7, 3 ]
```

SortedList returns a new list where the elements are sorted with respect to the operator <=. In the following example one sees that SortedList will not modify the value of the variable list:

```
gap> list := [2, 4, 7, 3];;
gap> SortedList(list);
[ 2, 3, 4, 7 ]
gap> list;
[ 2, 4, 7, 3 ]
```

Sort sorts a list in increasing order.

```
gap> list := [2, 4, 7, 3];;
gap> Sort(list);
gap> list;
[ 2, 3, 4, 7 ]
```

Can you recognize the difference between Sort and SortedList?

Lists

Say that we want to apply `SortedList` or `Sort` to a given list. In this case, all the elements of the list must be of the same type and comparable with respect to the operator <=.

Lists

`Filtered` allows us to obtain the elements of a list that satisfy a particular given property. The function `Number` returns the number of elements of a list that satisfy a given property. `First` returns the first element of a list that satisfy a given property.

```
gap> list := [1, 2, 3, 4, 5];;
gap> Filtered(list, x->x mod 2 = 0);
[ 2, 4 ]
gap> Number(list, x->x mod 2 = 0);
2
gap> Filtered(list, x->x mod 2 = 1);
[ 1, 3, 5 ]
gap> First(list, x->x mod 2 = 0);
2
```

Lists

Let us compute how many powers of 2 divide 18000. This number is four, as the following code shows:

```
gap> Factors(18000);
[ 2, 2, 2, 2, 3, 3, 5, 5, 5 ]
gap> Number(Factors(18000), x->x=2);
4
```

We get the same results as follows:

```
gap> Collected(Factors(18000));
[ [ 2, 4 ], [ 3, 2 ], [ 5, 3 ] ]
```

Lists

There are very nice ways to create lists. The following examples need no further explanations.

```
gap> List([1, 2, 3, 4, 5], x->x^2);
[ 1, 4, 9, 16, 25 ]
gap> List([1, 2, 3, 4, 5], IsPrime);
[ false, true, true, false, true ]
```

Lists

We want to prove that

$$1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\cdots+\frac{1}{9999}-\frac{1}{10000}=\frac{1}{5001}+\frac{1}{5002}+\cdots+\frac{1}{10000}.$$

The equality is a particular case of a general formula. However, here is the code to solve this particular case:

```
gap> n := 5000;;
gap> Sum(List([1..2*n], j->(-1)^(j+1)*1/j))=\
> Sum(List([n+1..2*n], j->1/j));
true
```

Ranges

Ranges are lists where the difference between two consecutive integers is a constant.

```
gap> Elements([1,3..11]);
[ 1, 3, 5, 7, 9, 11 ]
gap> Elements([1..5]);
[ 1, 2, 3, 4, 5 ]
gap> Elements([0,-2..-8]);
[ -8, -6, -4, -2, 0 ]
gap> AsList([0,-2..-8]);
[ 0, -2 .. -8 ]
gap> IsRange([1..100]);
true
gap> IsRange([1,3,5,6]);
false
```

Ranges

We can use `Elements` to list all the elements in a given range. Conversely, `ConvertToRangeRep` converts (if possible) a list into a range.

```
gap> list := [ 1, 2, 3, 4, 5 ];;
gap> ConvertToRangeRep(list);;
gap> list;
[ 1 .. 5 ]
gap> list := [ 7, 11, 15, 19, 23 ];
gap> IsRange(list);
true
gap> ConvertToRangeRep(list);
gap> list;
[ 7, 11 .. 23 ]
```

Sets

A set is a particular type of ordered list that contains no gaps with no repetitions. To convert a list to a set one uses `Set`.

```
gap> list := [1, 2, 3, 1, 5, 6, 2];;
gap> IsSet(list);
false
gap> Set(list);
[ 1, 2, 3, 5, 6 ]
```

Sets

To add elements use `AddSet` and `UniteSet`.
To remove them, `RemoveSet`.

```
gap> set := Set([1, 2, 4, 5]);;
gap> # Let us add the number 10
gap> AddSet(set, 10);
gap> set;
[ 1, 2, 4, 5, 10 ]
gap> # Let us remove the number 4
gap> RemoveSet(set, 4);
gap> set;
[ 1, 2, 5, 10 ]
gap> UniteSet(set, [1, 1, 5, 6]);
gap> set;
[ 1, 2, 5, 6, 10 ]
```

Sets

To perform basic set operations one uses `Union`, `Intersection`, `Difference` and `Cartesian`.

```
gap> S := Set([1, 2, 8, 11]);;
gap> T := Set([2, 5, 7, 8]);;
gap> Intersection(S, T);
[ 2, 8 ]
gap> Union(S, T);
[ 1, 2, 5, 7, 8, 11 ]
gap> Difference(S, T);
[ 1, 11 ]
gap> Difference(S, S);
[  ]
gap> Cartesian(S, T);
[ [ 1, 2 ], [ 1, 5 ], [ 1, 7 ], [ 1, 8 ], [ 2, 2 ],
  [ 2, 5 ], [ 2, 7 ], [ 2, 8 ], [ 8, 2 ], [ 8, 5 ],
  [ 8, 7 ], [ 8, 8 ], [ 11, 2 ], [ 11, 5 ],
  [ 11, 7 ], [ 11, 8 ] ]
```

Loops

We continue with basic GAP programming. Now we are about to learn about loops. Our presentation will be based on the following very simple problem. We want to check that

$$1 + 2 + 3 + \cdots + 100 = 5050.$$

Of course we can use `Sum`, which sums all the elements of a list:

```
gap> Sum([1..100]);
5050
```

Loops

An equivalent way of doing this uses **for ... do ... od**:

```
gap> s := 0;;
gap> for k in [1..100] do
> s := s+k;
> od;
gap> s;
5050
```

Loops

Yet another equivalent way of doing this uses **while** ... **do** ... **od**:

```
gap> s := 0;;
gap> k := 1;;
gap> while k<=100 do
> s := s+k;
> k := k+1;
> od;
gap> s;
5050
```

Loops

Yet another equivalent way of doing this uses **repeat ... until**:

```
gap> s := 0;;
gap> k := 1;;
gap> repeat
> s := s+k;
> k := k+1;
> until k>100;
gap> s;
5050
```

Loops

Now let us compute (again) Fibonacci numbers. This is better than the method we used before. Let us write a non-recursive function to compute Fibonacci numbers.

```
gap> fibonacci := function(n)
> local k, x, y, tmp;
> x := 1;
> y := 1;
> for k in [3..n] do
> tmp := y;
> y := x+y;
> x := tmp;
> od;
> return y;
> end;
function( n ) ... end
```

Is it really better than the previous function for computing Fibonacci numbers? Of course it is!

```
gap> fibonacci(100);
354224848179261915075
gap> fibonacci(1000);
434665576869374564356885276750406258025646605173\
717804024817290895365554179490518904038798400792\
551692959225930803226347752096896232398733224711\
616429964409065331879382989696499285160037044761\
37795166849228875
```

Loops

For computing Fibonacci numbers there an ever better solution! An easy induction exercise shows that (f_n) can be computed using

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n = \begin{pmatrix} f_{n-1} & f_n \\ f_n & f_{n+1} \end{pmatrix}, \quad n \geq 1.$$

Loops

We use this clever trick to compute (very efficiently) Fibonacci numbers:

```
gap> fibonacci := function(n)
> local m;
> m := [[0,1],[1,1]]^n;;
> return m[1][2];
> end;
function( n ) ... end
gap> fibonacci(10);
55
gap> fibonacci(100000);
<integer 259...875 (20899 digits)>
```

Loops

Divisors of a given integer can be obtained with `DivisorsInt`. In this example we run over the divisors of 100 and print only those numbers that are odd.

```
gap> Filtered(DivisorsInt(100), x->x mod 2 = 1);
[ 1, 5, 25 ]
```

Similarly

```
gap> for d in DivisorsInt(100) do
> if d mod 2 = 1 then
> Display(d);
> fi;
> od;
1
5
25
```

Loops

With **continue** one can skip iterations. An equivalent (but less elegant) approach to the previous problem is the following:

```
gap> for d in DivisorsInt(100) do
> if d mod 2 = 0 then
> continue;
> fi;
> Display(d);
> od;
1
5
25
```

With **break** one breaks a loop. In the following example we run over the numbers $1, 2, \ldots, 100$ and stop when a number whose square is divisible by 20 appears.

```
gap> First([1..100], x->x^2 mod 20 = 0);
10
```

Similarly:

```
gap> for k in [1..100] do
> if k^2 mod 20 = 0 then
> Display(k);
> break;
> fi;
> od;
10
```

`ForAny` returns **`true`** if there is an element in the list satisfying the required condition and **`false`** otherwise. Similarly `ForAll` returns **`true`** if all the elements of the list satisfy the required condition and **`false`** otherwise.

```
gap> ForAny([2,4,6,8,10], x->x mod 2 = 0);
true
gap> ForAll([2,4,6,8,10], x->(x > 0));
true
gap> ForAny([2,3,4,5], IsPrime);
true
gap> ForAll([2,3,4,5], IsPrime);
false
```

Now it is time to work with groups. We start with some elementary constructions.

Groups

One constructs groups with the function `Group`. We compute the order of the following groups:

- The group generated by the transposition (12)
- The group generated by the 5-cycle (12345)
- The group generated by the permutations $\{(12), (12345)\}$:

```
gap> Order(Group([(1,2)]));
2
gap> Order(Group([(1,2,3,4,5)]));
5
gap> Order(Group([(1,2),(1,2,3,4,5)]));
120
```

For $n \in \mathbb{N}$ let C_n be the (multiplicative) cyclic group of order n. One construct cyclic groups with `CyclicGroup`. With no extra arguments, this function returns an abstract presentation of a cyclic group.

Let us construct the cyclic group C_2 of size two as an abstract group, as a matrix group and as a permutation group.

```
gap> CyclicGroup(2);
<pc group of size 2 with 1 generators>
gap> CyclicGroup(IsMatrixGroup, 2);
Group([ [ [ 0, 1 ], [ 1, 0 ] ] ])
gap> CyclicGroup(IsPermGroup, 2);
Group([ (1,2) ])
```

Recall that a matrix group is a subgroup of $\mathbf{GL}(n, K)$ for some $n \in \mathbb{N}$ and some field K. A permutation group is a subgroup of some Sym_n.

For $n \in \mathbb{N}$ the dihedral group of order $2n$ is the group

$$\mathbb{D}_{2n} = \langle r, s : srs = r^{-1},\, s^2 = r^n = 1 \rangle.$$

To construct dihedral groups we use `DihedralGroup`. With no extra arguments, the function returns an abstract presentation of a dihedral group. As we did before for cyclic groups, we can construct dihedral groups as permutation groups.

Let us construct $\mathbb{D}_6$, compute its order and check that this is an abelian group.

```
gap> D6 := DihedralGroup(6);;
gap> Order(D6);
6
gap> IsAbelian(D6);
false
```

To display the elements of the group we use `Elements`:

```
gap> Elements(DihedralGroup(6));
[ <identity> of ..., f1, f2, f1*f2, f2^2, f1*f2^2 ]
gap> Elements(DihedralGroup(IsPermGroup, 6));
[ (), (2,3), (1,2), (1,2,3), (1,3,2), (1,3) ]
```

Groups

One constructs the symmetric group Sym_n with `SymmetricGroup`.
To construct the alternating group Alt_n one uses `AlternatingGroup`.
The elements of Sym_n are permutations of the set $\{1, \ldots, n\}$.

Let us construct Alt_4 and Sym_4 and display their elements.

```
gap> S4 := SymmetricGroup(4);;
gap> A4 := AlternatingGroup(4);;
gap> Elements(A4);
[ (), (2,3,4), (2,4,3), (1,2)(3,4), (1,2,3), (1,2,4),
  (1,3,2), (1,3,4), (1,3)(2,4), (1,4,2), (1,4,3),
  (1,4)(2,3) ]
```

Now let us check that

```
gap> (1,2,3) in A4;
true
gap> (1,2) in A4;
false
gap> (1,2,3)(4,5) in S4;
false
```

Let us check that Sym_3 has two elements of order three and three elements of order two. One computes order of elements with `Order`.

```
gap> S3 := SymmetricGroup(3);;
gap> List(S3, Order);
[ 1, 2, 3, 2, 3, 2 ]
gap> Collected(List(S3, Order));
[ [ 1, 1 ], [ 2, 3 ], [ 3, 2 ] ]
```

Let us show that

$$G = \left\langle \begin{pmatrix} 0 & i \\ i & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \right\rangle$$

is a non-abelian group of order eight not isomorphic to a dihedral group. Recall that the imaginary unit $i = \sqrt{-1}$ is `E(4)`.

```
gap> a := [[0,E(4)],[E(4),0]];;
gap> b := [[0,1],[-1,0]];;
gap> G := Group([a,b]);;
gap> Order(G);
8
gap> IsAbelian(G);
false
```

To check that $G \not\simeq \mathbb{D}_8$ we see that G contains a unique element of order two and $\mathbb{D}_8$ has five elements of order two:

```
gap> Number(G, x->Order(x)=2);
1
gap> Number(DihedralGroup(8), x->Order(x)=2);
5
```

The Mathieu group M_{11} is a simple group of order 7920. It can be defined as the subgroup of Sym_{11} generated by

$$(1\,2\,3\,4\,5\,6\,7\,8\,9\,10\,11), \quad (3\,7\,11\,8)(4\,10\,5\,6).$$

Let us construct M_{11} and check with `IsSimple` that M_{11} is simple:

```
gap> a := (1,2,3,4,5,6,7,8,9,10,11);;
gap> b := (3,7,11,8)(4,10,5,6);;
gap> M11 := Group([a,b]);;
gap> Order(M11);
7920
gap> IsSimple(M11);
true
```

Groups

The function `Group` can also be used to construct infinite groups. Let us consider two matrices with finite order and such that their product has infinite order.

```
gap> a := [[0,-1],[1,0]];;
gap> b:= [[0,1],[-1,-1]];;
gap> Order(a);
4
gap> Order(b);
3
gap> Order(a*b);
infinity
gap> Order(Group([a,b]));
infinity
```

Not always we will be able to determine whether an element has finite order or not!

Groups

With `Subgroup` we construct the subgroup of a group generated by a list of elements. The function `AllSubgroups` returns the list of subgroups of a given group. The index of a subgroup can be computed with `Index`.

The subgroup of Sym_3 generated by (12) is $\{\mathrm{id}, (12)\}$ and has index three. The subgroup of Sym_3 generated by (123) is $\{\mathrm{id}, (123), (132)\}$ and has index two:

```
gap> S3 := SymmetricGroup(3);;
gap> Elements(Subgroup(S3, [(1,2)]));
[ (), (1,2) ]
gap> Index(S3, Subgroup(S3, [(1,2)]));
3
gap> Elements(Subgroup(S3, [(1,2,3)]));
[ (), (1,2,3), (1,3,2) ]
gap> Index(S3, Subgroup(S3, [(1,2,3)]));
2
```

A subgroup K of G is said to be normal if $gKg^{-1} \subseteq K$ for all $g \in G$. If K is normal in G, then G/K is a group. With `IsSubgroup` we check that Alt_4 is a subgroup of Sym_4. With `IsNormal` we see that Alt_4 is a subset of Sym_4 under conjugation:

```
gap> S4 := SymmetricGroup(4);;
gap> A4 := AlternatingGroup(4);;
gap> IsSubgroup(S4,A4);
true
gap> IsNormal(S4,A4);
true
gap> Order(S4/A4);
2
```

The subgroup of Sym_4 generated by (123) is not normal in Sym_4:

```
gap> IsNormal(S4, Subgroup(S4, [(1,2,3)]));
false
```

Let us show that in $\mathbb{D}_8$ there are subgroups H and K such that K is normal in H, H is normal in G and K is not normal in G.

```
gap> D8 := DihedralGroup(IsPermGroup, 8);;
gap> K := Subgroup(D8, [(2,4)]);;
gap> Elements(K);
[ (), (2,4) ]
gap> H := Subgroup(D8, [(1,2,3,4)^2,(2,4)]);;
gap> Elements(H);
[ (), (2,4), (1,3), (1,3)(2,4) ]
gap> IsNormal(D8, K);
false
gap> IsNormal(D8, H);
true
gap> IsNormal(H, K);
true
```

Let us compute the quotients of the cyclic group C_4. Since every subgroup of C_4 is normal, we can use `AllSubgroups` to check that C_4 contains a unique non-trivial proper subgroup K. The quotient C_4/K has two elements:

```
gap> C4 := CyclicGroup(IsPermGroup, 4);;
gap> AllSubgroups(C4);
[ Group(()), Group([ (1,3)(2,4) ]),
  Group([ (1,2,3,4) ]) ]
gap> K := last[2];;
gap> Order(C4/K);
2
```

For $n \in \mathbb{N}$ the generalized quaternion group is the group

$$Q_{4n} = \langle x, y \mid x^{2n} = y^4 = 1, x^n = y^2, y^{-1}xy = x^{-1} \rangle.$$

We use `QuaternionGroup` to construct generalized quaternion groups. We can use the filters `IsPermGroup` (resp. `IsMatrixGroup`) to obtain generalized quaternion groups as permutation (resp. matrix) groups.

Let us check that each subgroup of the quaternion group Q_8 of order eight is normal and that Q_8 is non-abelian:

```
gap> Q8 := QuaternionGroup(IsMatrixGroup, 8);;
gap> IsAbelian(Q8);
false
gap> ForAll(AllSubgroups(Q8), x->IsNormal(Q8,x));
true
```

If G is a group, its center is the subgroup

$$Z(G) = \{x \in G : xy = yx \text{ for all } y \in G\}.$$

The commutator of two elements $x, y \in G$ is defined as

$$[x, y] = x^{-1}y^{-1}xy.$$

The commutator subgroup, or derived subgroup of G, is the subgroup $[G, G]$ generated by all the commutators of G.

Let us check that Alt_4 has trivial center and that its commutator is the group $\{\mathrm{id}, (12)(34), (13)(24), (14)(23)\}$:

```
gap> A4 := AlternatingGroup(4);;
gap> IsTrivial(Center(A4));
true
gap> Elements(DerivedSubgroup(A4));
[ (), (1,2)(3,4), (1,3)(2,4), (1,4)(2,3) ]
```

Direct products of groups are constructed with `DirectProduct`. Example: the groups $C_4 \times C_4$ and $C_2 \times Q_8$ have both order 16, have both three elements of order two and twelve elements of order four.

```
gap> C4 := CyclicGroup(IsPermGroup, 4);;
gap> C2 := CyclicGroup(IsPermGroup, 2);;
gap> Q8 := QuaternionGroup(8);;
gap> C4xC4 := DirectProduct(C4, C4);;
gap> C2xQ8 := DirectProduct(C2, Q8);;
gap> Collected(List(C4xC4, Order));
[ [ 1, 1 ], [ 2, 3 ], [ 4, 12 ] ]
gap> Collected(List(C2xQ8, Order));
[ [ 1, 1 ], [ 2, 3 ], [ 4, 12 ] ]
```

Are these two groups isomorphic? No. An easy way to see this is the following: $C_4 \times C_4$ is abelian and $C_2 \times Q_8$ is not:

```
gap> IsAbelian(C4xC4);
true
gap> IsAbelian(C2xQ8);
false
```

Alternatively:

```
gap> IsomorphismGroups(C4xC4,C2xQ8);
fail
```

Recall that if G is a group and $g \in G$, the conjugacy class of g in G is the subset $g^G = \{x^{-1}gx : x \in G\}$. The centralizer of g in G is the subgroup

$$C_G(g) = \{x \in G : xg = gx\}.$$

`ConjugacyClass` computes a conjugacy class The centralizer can be computed with `Centralizer`.

Let us check that Sym_3 contains three conjugacy classes with representatives id, (12) and (123), so that

$$(12)^{\mathrm{Sym}_3} = \{(12),(13),(23)\}, \qquad (123)^{\mathrm{Sym}_3} = \{(123),(132)\}.$$

```
gap> S3 := SymmetricGroup(3);;
gap> ConjugacyClasses(S3);
[ ()^G, (1,2)^G, (1,2,3)^G ]
gap> Elements(ConjugacyClass(S3, (1,2)));
[ (2,3), (1,2), (1,3) ]
gap> Elements(ConjugacyClass(S3, (1,2,3)));
[ (1,2,3), (1,3,2) ]
```

Let us check that $C_{\mathrm{Sym}_3}((123)) = \{\mathrm{id},(123),(132)\}$:

```
gap> Elements(Centralizer(S3, (1,2,3)));
[ (), (1,2,3), (1,3,2) ]
```

In this example we use the function `Representative` to construct a list of representatives of conjugacy classes of Alt_4:

```
gap> A4 := AlternatingGroup(4);;
gap> List(ConjugacyClasses(A4), Representative);
[ (), (1,2)(3,4), (1,2,3), (1,2,4) ]
```

With the function `IsConjugate` we can check whether two elements are conjugate. If two elements g and h are conjugate, we want to find an element x such that $g = x^{-1}hx$. For that purpose we use `RepresentativeAction`.

Let us check that (123) and $(132) = (123)^2$ are not conjugate in Alt_4:

```
gap> A4 := AlternatingGroup(4);;
gap> g := (1,2,3);;
gap> IsConjugate(A4, g, g^2);
false
```

Now we check that (123) and (134) are conjugate in Alt_4. We also find an element $x = (234)$ such that $(134) = x^{-1}(123)x$:

```
gap> h := (1,3,4);;
gap> IsConjugate(A4, g, h);
true
gap> x := RepresentativeAction(A4, g, h);
(2,3,4)
gap> x^(-1)*g*x=h;
true
```

It is well-known that the converse of Lagrange theorem does not hold. Let us show that Alt_4 has no subgroups of order six.

A naive idea to prove that Alt_4 has no subgroups of order six is to study all the $\binom{12}{6} = 924$ subsets of Alt_4 of size six and check that none of these subsets is a group:

```
gap> A4 := AlternatingGroup(4);;
gap> k := 0;;
gap> for x in Combinations(Elements(A4), 6) do
> if Size(Subgroup(A4, x))=Size(x) then
> k := k+1;
> fi;
> od;
gap> k;
0
```

This is an equivalent way of doing the same thing:

```
gap> ForAny(Combinations(Elements(A4), 6),\
> x->Size(Subgroup(A4, x))=Size(x));
false
```

Here we have another idea: if Alt_4 has a subgroup of order six, then the index of this subgroup in Alt_4 is two. With `SubgroupsOfIndexTwo` we check that Alt_4 has no subgroups of index two:

```
gap> SubgroupsOfIndexTwo(A4);
[  ]
```

Of course we can construct all subgroups and check that there are no subgroups of order six:

```
gap> List(AllSubgroups(A4), Order);
[ 1, 2, 2, 2, 3, 3, 3, 3, 4, 12 ]
gap> 6 in last;
false
```

It is enough to construct all conjugacy classes of subgroups!

An exercise on commutators

It is known that the commutator of a finite group is not always equal to the set of commutators. Carmichael's book[1] shows the following example: Let G be the subgroup of Sym_{16} generated by the permutations

$$
\begin{aligned}
a &= (13)(24), & b &= (57)(6,8),\\
c &= (911)(10,12), & d &= (13,15)(14,16),\\
e &= (13)(5,7)(9,11), & f &= (12)(3,4)(13,15),\\
g &= (56)(7,8)(13,14)(15,16), & h &= (9\ 10)(11\ 12).
\end{aligned}
$$

Show that $[G,G]$ has order 16 and that the set of commutators has 15 elements. In particular, one can show that $cd \in [G,G]$ and that cd is not a commutator.

[1] Introduction to the theory of groups of finite order. Dover Publications, Inc., New York, 1956

An exercise on commutators

Here is the solution:

```
gap> a := (1,3)(2,4);;
gap> b := (5,7)(6,8);;
gap> c := (9,11)(10,12);;
gap> d := (13,15)(14,16);;
gap> e := (1,3)(5,7)(9,11);;
gap> f := (1,2)(3,4)(13,15);;
gap> g := (5,6)(7,8)(13,14)(15,16);;
gap> h := (9,10)(11,12);;
gap> G := Group([a,b,c,d,e,f,g,h]);;
gap> D := DerivedSubgroup(G);;
gap> Size(D);
16
gap> Size(Set(List(Cartesian(G,G), Comm)));
15
gap> c*d in Difference(D,\
> Set(List(Cartesian(G,G), Comm)));
true
```

Group homomorphisms

Now we work with group homomorphisms. There are several ways to construct group homomorphisms.

The function `GroupHomomorphismByImages` returns the group homomorphism constructed from a list of generators of the domain and the value of the image at each generator. Properties of group homomorphisms can be studied with `Image`, `IsInjective`, `IsSurjective`, `Kernel`, `PreImage`, `PreImages`, etc.

Group homomorphisms

The map $\mathrm{Sym}_4 \to \mathrm{Sym}_3$ that maps each transposition of Sym_4 into (12) extends to a group homomorphism f. This homomorphism f is not injective and it is not surjective.

```
gap> S4 := SymmetricGroup(4);;
gap> S3 := SymmetricGroup(3);;
gap> f := GroupHomomorphismByImages(S4, S3,\
> [(1,2),(1,3),(1,4),(2,3),(2,4),(3,4)],\
> [(1,2),(1,2),(1,2),(1,2),(1,2),(1,2)]);;
gap> Size(Kernel(f));
12
gap> IsInjective(f);
false
gap> Size(Image(f));
2
gap> (1,2,3) in Image(f);
false
```

Group homomorphisms

To construct the canonical canonical map $G \to G/K$ one uses the function `NaturalHomomorphismByNormalSubgroup`. Let us construct $C_{12} = \langle g : g^{12} = 1 \rangle$ as a group of permutations, the subgroup $K = \langle g^6 \rangle$ and the quotient C_{12}/K. We also construct the canonical (surjective) map $C_{12} \to C_{12}/K$:

```
gap> g := (1,2,3,4,5,6,7,8,9,10,11,12);;
gap> C12 := Group(g);;
gap> K := Subgroup(C12, [g^6]);;
gap> f := NaturalHomomorphism\
> ByNormalSubgroup(C12, K);
[ (1,2,3,4,5,6,7,8,9,10,11,12) ] -> [ f1 ]
gap> Image(f, g^6);
<identity> of ...
```

An exercise on group homomorphisms

Verify the correspondence theorem for the groups G and G/K defined in the previous slide: subgroups of G containing K are in bijective correspondence with subgroups of G/K.

Group homomorphisms

The function `AutomorphismGroup` computes the automorphism group of a finite group. If G is a group, the automorphisms of G of the form $x \mapsto g^{-1}xg$, where $g \in G$, are the inner automorphisms of G. The function `IsInnerAutomorphism` checks whether a given automorphism is inner.

Let us check that $\mathrm{Aut}(\mathrm{Sym}_3)$ is a non-abelian group of six elements:

```
gap> aut := AutomorphismGroup(SymmetricGroup(3));
<group of size 6 with 2 generators>
gap> IsAbelian(aut);
false
```

For $n \in \{2,3,4,5\}$ each automorphism of Sym_n is inner. Here is the code:

```
gap> for n in [2..5] do
> G := SymmetricGroup(n);;
> if ForAll(AutomorphismGroup(G),\
> x->IsInnerAutomorphism(x)) then
> Print("Each automorfism of S",\
> n, " is inner.\n");
> fi;
> od;
Each automorphism of S2 is inner.
Each automorphism of S3 is inner.
Each automorphism of S4 is inner.
Each automorphism of S5 is inner.
```

It is known that in Sym_6 there are non-inner automorphisms:

```
gap> S6 := SymmetricGroup(6);;
gap> Number(AutomorphismGroup(S6),\
> x->IsInnerAutomorphism(x)=false);;
720
```

The automorphism of Sym_6 given by $(123456) \mapsto (162)(35)$ and $(12) \mapsto (12)(34)(56)$ is not inner.

```
gap> f := First(AutomorphismGroup(S6),\
> x->IsInnerAutomorphism(x)=false);
[ (1,2,3,4,5,6), (1,2) ] ->
[ (1,6,2)(3,5), (1,2)(3,4)(5,6) ]
```

Group homomorphisms

Let us compute the image of this homomorphism in some transpositions:

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

Alternatively:

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

Group homomorphisms

With `AllHomomorphisms` one constructs the set of group homomorphisms between two given groups. `AllEndomorphisms` computes all endomorphisms.

There are ten endomorphisms of Sym_3.

```
gap> S3 := SymmetricGroup(3);;
gap> Size(AllEndomorphisms(S3));
10
```

The center of $C_2 \times \mathrm{Sym}_3$ is not stable under endomorphisms of $C_2 \times \mathrm{Sym}_3$. We see that $Z(C_2 \times \mathrm{Sym}_3) = \{\mathrm{id}, (12)\}$ and that there exists at least one endomorphism of $C_2 \times \mathrm{Sym}_3$ that permutes the non-trivial element of the center:

```
gap> C2 := CyclicGroup(IsPermGroup, 2);;
gap> S3 := SymmetricGroup(3);;
gap> C2xS3 := DirectProduct(C2, S3);;
gap> Center(C2xS3);
Group([ (1,2) ])
gap> ForAll(AllEndomorphisms(C2xS3),\
> f->Image(f,(1,2)) in [(), (1,2)]);
false
```

Group homomorphisms

To prove that $\mathsf{Aut}(\mathrm{Sym}_6)/\mathsf{Inn}(\mathrm{Sym}_6) \simeq C_2$ we use the function `InnerAutomorphismsAutomorphismGroup`, which returns the inner automorphism group of a given group.

```
gap> S6 := SymmetricGroup(6);;
gap> A := AutomorphismGroup(S6);;
gap> Size(A);
1440
gap> I := InnerAutomorphismsAutomorphismGroup(A);;
gap> Order(A/I);
2
```

A particular type of group homomorphism is given by actions.

Let us see how the alternating group Alt_4 acts on a coset space by right multiplication. First we define Alt_5 and we compute the list of conjugacy classes of subgroups: there are nine conjugacy classes of subgroups!

```
gap> A5 := AlternatingGroup(5);;
gap> l := ConjugacyClassesSubgroups(A5);;
gap> Size(l);
9
```

Actions

We can learn some information on these groups:

```
gap> List(l, x->Order(Representative(x)));
[ 1, 2, 3, 4, 5, 6, 10, 12, 60 ]
gap> List(l, x->Index(A5, Representative(x)));
[ 60, 30, 20, 15, 12, 10, 6, 5, 1 ]
gap> List(l, \
> x->StructureDescription(Representative(x)));
[ "1", "C2", "C3", "C2 x C2", "C5",
  "S3", "D10", "A4", "A5" ]
```

Actions

Let H be the subgroup of Alt_5 isomorphic to the cyclic group C_5 of order five. We now construct the action of Alt_5 on Alt_5/H by right multiplication:

```
gap> H := Representative(l[5]);;
gap> Elements(H);
[ (), (1,2,3,4,5), (1,3,5,2,4),
  (1,4,2,5,3), (1,5,4,3,2) ]
gap> f := ActionHomomorphism(A5,\
> RightCosets(A5,H), OnRight);;
gap> Kernel(f);
1
gap> IsInjective(f);
true
gap> IsSurjective(f);
false
```