computational group theory with GAP

1. GAP in SageMath
   - the GAP system
   - combinatorics and list manipulations
   - integer matrices and normal forms
2. Computing with Groups
   - permutation groups
   - group libraries
   - algebraic extensions
3. Rubik Magic's Cube
   - building a GUI
   - six rotations as group actions
   - analyzing Rubik's magic cube with GAP

MCS 507 Lecture 36
Mathematical, Statistical and Scientific Software
Jan Verschelde, 13 November 2023

computational group theory with GAP

1. GAP in SageMath
   - the GAP system
   - combinatorics and list manipulations
   - integer matrices and normal forms
2. Computing with Groups
   - permutation groups
   - group libraries
   - algebraic extensions
3. Rubik Magic's Cube
   - building a GUI
   - six rotations as group actions
   - analyzing Rubik's magic cube with GAP

the GAP system

GAP stands for Groups, Algorithms and Programming.
GAP is free software, under the GNU Public License.

Development began at RWTH Aachen (Joachim Neubüser),
version 2.4 was released in 1988, 3.1 in 1992.
Coordination was transferred to St. Andrews (Steve Linton) in 1997,
version 4.1 was released in 1999.
Release 4.4 was coordinated in Fort Collins (Alexander Hulpke).

We run GAP explicitly in SageMath via

1. the class `Gap`, do `help(gap);` or
2. opening a Terminal Session with GAP,
   type `gap_console();` in a SageMath terminal.

using GAP in SageMath

```
SageMath version 8.8, Release Date: 2019-06-26
Using Python 2.7.15. Type "help()" for help.
```

```
[sage: gap_console()
        GAP 4.10.1 of 23-Feb-2019
  GAP   https://www.gap-system.org
        Architecture: x86_64-apple-darwin18.6.0-default64-kv3
 Configuration:  gmp 6.0.0, readline
 Loading the library and packages ...
 Packages:   Alnuth 3.1.0, AtlasRep 1.5.1, AutPGrp 1.10, CRISP 1.4.4,
             CTblLib 1.2.2, FactInt 1.6.2, FGA 1.4.0, GAPDoc 1.6.2,
             IRREDSOL 1.4, LAGUNA 3.9.2, Polycyclic 2.14, PrimGrp 3.3.2,
             SmallGrp 1.3, Sophus 1.24, TomLib 1.2.7, TransGrp 2.0.4
 Try '??help' for help. See also '?copyright', '?cite' and '?authors'
gap>
```

the OSCAR project includes GAP

https://oscar.computeralgebra.de

OSCAR stands for Open Source Computer Algebra Research.

The project builds on and extends four cornerstone systems:

1. GAP: Groups, Algorithms, and Programming,
   a System for Computational Discrete Algebra;
2. Singular: computational algebraic geometry;
3. Polymake: polyhedral geometry;
4. Antic (Hecke, Nemo): number theory;

with many other packages using the Julia programming language.

computational group theory with GAP

1. GAP in SageMath
   - the GAP system
   - combinatorics and list manipulations
   - integer matrices and normal forms
2. Computing with Groups
   - permutation groups
   - group libraries
   - algebraic extensions
3. Rubik Magic's Cube
   - building a GUI
   - six rotations as group actions
   - analyzing Rubik's magic cube with GAP

combinatorics

To get all 3-tuples of elements in $\{0, 1, 2\}$:

```
gap> L := Tuples([0..2],3);
[ [ 0, 0, 0 ], [ 0, 0, 1 ], [ 0, 0, 2 ], [ 0, 1, 0 ],
  [ 0, 1, 1 ], [ 0, 1, 2 ], [ 0, 2, 0 ], [ 0, 2, 1 ],
  [ 0, 2, 2 ], [ 1, 0, 0 ], [ 1, 0, 1 ], [ 1, 0, 2 ],
  [ 1, 1, 0 ], [ 1, 1, 1 ], [ 1, 1, 2 ], [ 1, 2, 0 ],
  [ 1, 2, 1 ], [ 1, 2, 2 ], [ 2, 0, 0 ], [ 2, 0, 1 ],
  [ 2, 0, 2 ], [ 2, 1, 0 ], [ 2, 1, 1 ], [ 2, 1, 2 ],
  [ 2, 2, 0 ], [ 2, 2, 1 ], [ 2, 2, 2 ] ]
gap> Length(L);
27
gap> NrTuples([0..2],3);
27
```

in a SageMath session

```
sage: L = gap("Tuples([0..1],2)");
sage: L
[ [ 0, 0 ], [ 0, 1 ], [ 1, 0 ], [ 1, 1 ] ]
sage: type(L)
<class 'sage.interfaces.gap.GapElement'>
sage: list(L)
[[ 0, 0 ], [ 0, 1 ], [ 1, 0 ], [ 1, 1 ]]
sage: type(_)
<type 'list'>
sage:
```

list manipulations

Continuing after `gap> L := Tuples([0..2],3);`
we filter those 3-tuples that sum to two:

```
gap> a := L[26];
[ 2, 2, 1 ]
gap> Sum(a);
5
gap> F := Filtered(L,n->Sum(n) = 2);
[ [ 0, 0, 2 ], [ 0, 1, 1 ], [ 0, 2, 0 ],
  [ 1, 0, 1 ], [ 1, 1, 0 ], [ 2, 0, 0 ] ]
gap> IsList(F);
true
```

computational group theory with GAP

1. GAP in SageMath
   - the GAP system
   - combinatorics and list manipulations
   - integer matrices and normal forms
2. Computing with Groups
   - permutation groups
   - group libraries
   - algebraic extensions
3. Rubik Magic's Cube
   - building a GUI
   - six rotations as group actions
   - analyzing Rubik's magic cube with GAP

defining matrices

Defining matrices is straightforward:

```
gap> M := [[1,2,3],[4,5,6],[7,8,9]];
[ [ 1, 2, 3 ], [ 4, 5, 6 ], [ 7, 8, 9 ] ]
gap> IsMatrix(M);
true
gap> Determinant(M);
0
gap> v := NullspaceIntMat(M);
[ [ 1, -2, 1 ] ]
gap> IsMatrix(v);
true
```

matrix computations

The nullspace is a matrix,
to verify that M*v equals zero:

```
gap> IsRowVector(v[1]);
true
gap> TransposedMat(v);
[ [ 1 ], [ -2 ], [ 1 ] ]
gap> w := TransposedMat(v);
[ [ 1 ], [ -2 ], [ 1 ] ]
gap> M*w;
[ [ 0 ], [ 0 ], [ 0 ] ]
```

the Hermite normal form

The Hermite normal form of an integer matrix A is
an upper triangular matrix H obtained from A
via multiplication with a unimodular matrix Q: $QA = H$.

```
gap> A := RandomMat(3,3,[-5..5]);
[ [ -4, 1, 5 ], [ -3, -4, 5 ], [ 2, -4, -3 ] ]
gap> H := HermiteNormalFormIntegerMat(A);
[ [ 1, 0, 20 ], [ 0, 1, 4 ], [ 0, 0, 27 ] ]
```

The method `TriangulizeIntegerMat`(A)
changes A to its Hermite normal form.

computing the transforms

```
gap> N := HermiteNormalFormIntegerMatTransform(A);
rec( normal := [ [ 1, 0, 20 ], [ 0, 1, 4 ],
      [ 0, 0, 27 ] ],
  rowC := [ [ 1, 0, 0 ], [ 0, 1, 0 ], [ 0, 0, 1 ] ],
  rowQ := [ [ -16, 11, -15 ], [ -3, 2, -3 ],
      [ -20, 14, -19 ] ], rank := 3,
  signdet := -1,
  rowtrans := [ [ -16, 11, -15 ], [ -3, 2, -3 ],
      [ -20, 14, -19 ] ] )
gap> N.rowtrans*A;
[ [ 1, 0, 20 ], [ 0, 1, 4 ], [ 0, 0, 27 ] ]
gap> N.rowtrans*A = N.normal;
true
gap> Determinant(N.rowtrans);
-1
```

the Smith normal form

The Smith normal form of an integer matrix A
is a diagonal matrix S obtained via multiplications of A
with two unimodular matrices P, Q so that $PAQ = S$.

```
gap> S := SmithNormalFormIntegerMat(A);
[ [ 1, 0, 0 ], [ 0, 1, 0 ], [ 0, 0, 27 ] ]
```

The method `DiagonalizeIntegerMat`(A)
changes A to its Smith normal form.

computing the transforms

```
gap> SNF := SmithNormalFormIntegerMatTransforms(A);
rec( normal := [ [ 1, 0, 0 ], [ 0, 1, 0 ],
      [ 0, 0, 27 ] ],
  rowC := [ [ 1, 0, 0 ], [ 0, 1, 0 ], [ 0, 0, 1 ] ],
  rowQ := [ [ -16, 11, -15 ], [ -3, 2, -3 ],
      [ -20, 14, -19 ] ],
  colC := [ [ 1, 0, 0 ], [ 0, 1, 0 ], [ 0, 0, 1 ] ],
  colQ := [ [ 1, 0, -20 ], [ 0, 1, -4 ],
      [ 0, 0, 1 ] ], rank := 3,
  signdet := -1,
  rowtrans := [ [ -16, 11, -15 ], [ -3, 2, -3 ],
      [ -20, 14, -19 ] ],
  coltrans := [ [ 1, 0, -20 ], [ 0, 1, -4 ],
      [ 0, 0, 1 ] ] )
gap> SNF.rowtrans*A*SNF.coltrans;
[ [ 1, 0, 0 ], [ 0, 1, 0 ], [ 0, 0, 27 ] ]
gap> SNF.rowtrans*A*SNF.coltrans = SNF.normal;
true
```

computational group theory with GAP

1. GAP in SageMath
   - the GAP system
   - combinatorics and list manipulations
   - integer matrices and normal forms
2. Computing with Groups
   - permutation groups
   - group libraries
   - algebraic extensions
3. Rubik Magic's Cube
   - building a GUI
   - six rotations as group actions
   - analyzing Rubik's magic cube with GAP

permutation groups

The full permutation group on n elements has size $n!$
so holding the entire group in memory should be avoided.

```
gap> a := (1,2);
(1,2)
gap> Inverse(a);
(1,2)
gap> b := (1,3);
(1,3)
gap> c := a*b;
(1,2,3)
gap> g := Group(a,b);
Group([ (1,2), (1,3) ])
gap> Size(g);
6
```

using an enumerator

```
gap> e := Enumerator(g);
<enumerator of perm group>
gap> e[1];
()
gap> e[6];
(1,3)
gap> G := List([1..Size(g)], i -> e[i]);
[ (), (1,2,3), (1,3,2), (2,3), (1,2), (1,3) ]
```

We see all elements of the full permutation group on three elements.

the multiplication table

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

The multiplication table shows that $G, \star$ is a group:

- $\forall a, b \in G$: $a \star b \in G$,
- $\forall a, b, c \in G$: $(a \star b) \star c = a \star (b \star c)$,
- $\exists 1 \in G, \forall a \in G$: $1 \star a = a$,
- $\forall a \in G, \exists b \in G$: $a \star b = 1$.

group actions

The orbit of a group on an item is
the result of all group actions on that item.

```
gap> G;
[ (), (1,2,3), (1,3,2), (2,3), (1,2), (1,3) ]
gap> Orbit(g,[1,2,3],OnTuples);
[ [ 1, 2, 3 ], [ 3, 1, 2 ], [ 2, 3, 1 ],
  [ 1, 3, 2 ], [ 2, 1, 3 ], [ 3, 2, 1 ] ]
```

Observe the cycle notation, e.g.: `(1,3,2)` denotes
that 1 is mapped to 3, 3 to 2, and 2 to 1.

computational group theory with GAP

1. GAP in SageMath
   - the GAP system
   - combinatorics and list manipulations
   - integer matrices and normal forms
2. Computing with Groups
   - permutation groups
   - group libraries
   - algebraic extensions
3. Rubik Magic's Cube
   - building a GUI
   - six rotations as group actions
   - analyzing Rubik's magic cube with GAP

cyclic groups

GAP knows several groups, available in libraries.

Cyclic groups are generated by one single element, e.g.:

```
gap> c5 := CyclicGroup(5);
<pc group of size 5 with 1 generators>
gap> g := GeneratorsOfGroup(c5);
[ f1 ]
gap> G := List([1..5],i->g^i);
[ [ f1 ], f1^2, [ f1^3 ], f1^4, [ <identity> of ... ] ]
```

`pc` stands for polycyclic.

groups of prescribed order

We can use `CyclicGroup` to make groups with a prescribed number of elements:

```
gap> c4 := CyclicGroup(4);
<pc group of size 4 with 2 generators>
gap> g := GeneratorsOfGroup(c4);
[ f1, f2 ]
gap> a := g[1]; b := g[2];
f1
f2
gap> L := [ 1, a, b, a*b ];
[ 1, f1, f2, f1*f2 ]
gap> a*b*a;
<identity> of ...
gap> b*b;
<identity> of ...
gap> (a*b)*(a*b);
f2
```

groups generated by one generator

The command CyclicGroup() in GAP generates "pc" or polycyclic groups which may have more than one generator.
To make a group with one generator, one can work in GAP as follows:

```
gap> f := FreeGroup("x");
<free group on the generators [ x ]>
gap> g := GeneratorsOfGroup(f);
[ x ]
gap> x := g[1];
x
gap> relation := [ x^4 ];
[ x^4 ]
gap> G := f/relation;
<fp group on the generators [ x ]>
gap> Size(G);
4
gap> L := List([1..Size(G)], i->e[i]);
[ <identity ...>, x, x^-1, x^2 ]
```

the dihedral group

The dihedral group of order $2n$ consists of the isometries of a regular n-gon: (1) the n rotations through angles $2\pi k/n$; and (2) n reflections about lines through the center and either through a vertex or bisecting an edge.

```
gap> d := DihedralGroup(10);
<pc group of size 10 with 2 generators>
gap> e := Enumerator(d);
<enumerator>
gap> Size(d);
10
gap> L := List([1..Size(d)],i->e[i]);
[ <identity> of ..., f2, f2^2, f2^3, f2^4,
  f1, f1*f2, f1*f2^2, f1*f2^3, f1*f2^4 ]
gap> GeneratorsOfGroup(d);
[ f1, f2 ]
```

subgroups

A subset of the generators of a group *G*
generates a subgroup of *G*:

```
gap> d := DihedralGroup(10);
<pc group of size 10 with 2 generators>
gap> g := GeneratorsOfGroup(d);
[ f1, f2 ]
gap> s := Group(g[1]);
<pc group with 1 generators>
gap> IsSubgroup(d,s);
true
```

computational group theory with GAP

1. GAP in SageMath
   - the GAP system
   - combinatorics and list manipulations
   - integer matrices and normal forms
2. Computing with Groups
   - permutation groups
   - group libraries
   - algebraic extensions
3. Rubik Magic's Cube
   - building a GUI
   - six rotations as group actions
   - analyzing Rubik's magic cube with GAP

algebraic extensions

We consider an irreducible polynomial f with rational coefficients.

```
gap> x := Indeterminate(Rationals,"x");
x
gap> f := x^3 + 2*x^2 + 1;
x^3+2*x^2+1
gap> IsIrreducible(f);
true
gap> F := AlgebraicExtension(Rationals,f);
<algebraic extension over the Rationals of degree 3>
gap> a := RootOfDefiningPolynomial(F);
a
gap> a^3;
-2*a^2-1
```

The extension of the field of rational numbers $\mathbb{Q}$ with a root a of f is denoted by $\mathbb{Q}(a)$. Because a is a root of $f = x^3 + 2x^2 + 1$, we have $a^2 = -2a^2 - 1$ and $\mathbb{Q}(a)$ consists of elements of the form $q_0 + q_1 a + q_2 a^2$, for $q_0, q_1, q_2 \in \mathbb{Q}$.

factoring over $\mathbb{Q}(a)$

We started with $f = x^3 + 2x^2 + 1 \in \mathbb{Q}[x]$
and added to $\mathbb{Q}$ a root a of f to define $\mathbb{Q}(a)$.
The polynomial f is now reducible over $\mathbb{Q}(a)$:

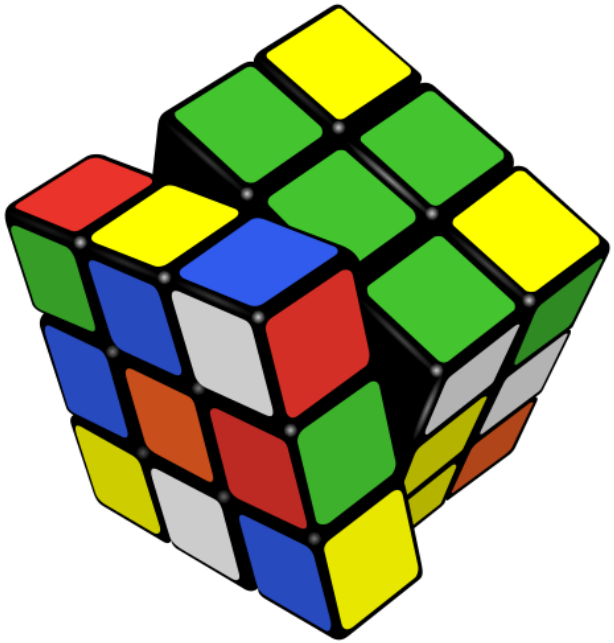
```
gap> y := Indeterminate(F,"y");
y
gap> g := y^3 + 2*y^2 + 1;
y^3+!2*y^2+!1
gap> Factors(g);
[ y+(-a), y^2+(a+2)*y+(a^2+2*a) ]
```

We defined $g = y^3 + 2y^2 + 1 \in \mathbb{Q}(a)[y]$.
Over $\mathbb{Q}(a)$: $g = (y - a)(y^2 + (a + 2)y + (a^2 + 2a))$.

computational group theory with GAP

1. GAP in SageMath
   - the GAP system
   - combinatorics and list manipulations
   - integer matrices and normal forms
2. Computing with Groups
   - permutation groups
   - group libraries
   - algebraic extensions
3. Rubik Magic's Cube
   - building a GUI
   - six rotations as group actions
   - analyzing Rubik's magic cube with GAP

Rubik Magic's Cube



3-D combination puzzle
invented in 1974 by Ernö Rubik.

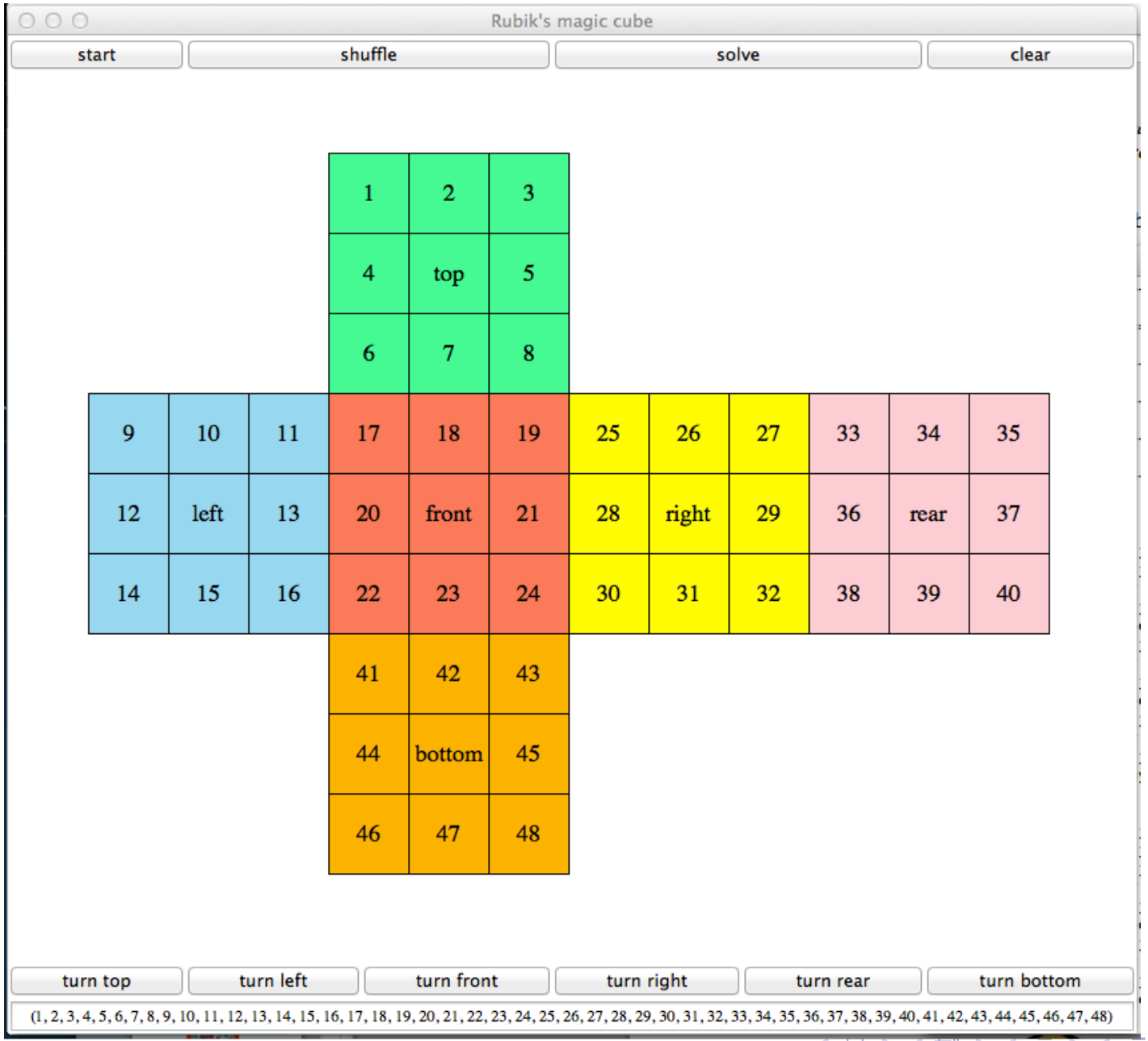
As each face rotates independently,
the colors of the sides mix up.

One of the top-selling puzzle games.

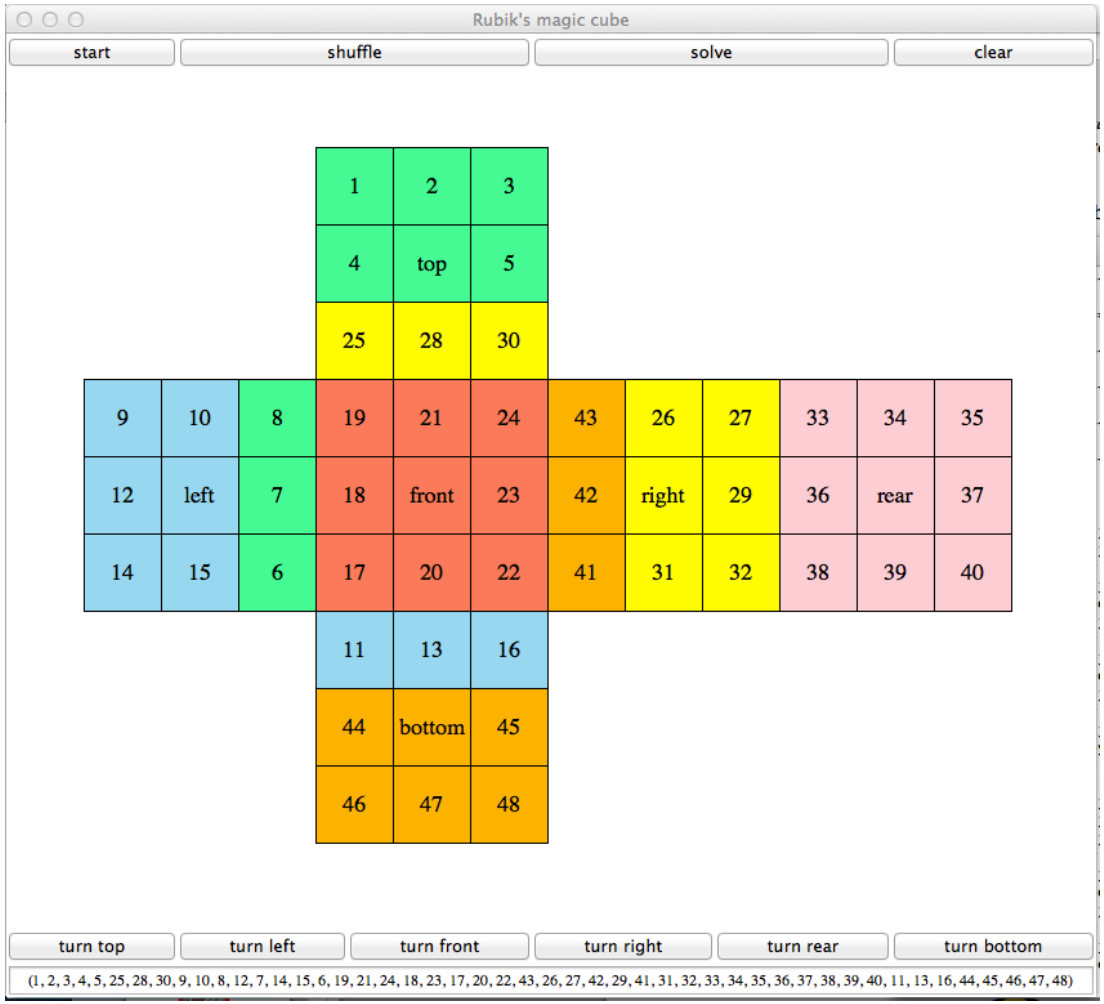
Computational fun:

- Build a GUI to model the puzzle.
- Analyze group actions with GAP.
- Supercomputing: smallest number of moves to solve ≤ 26,
  result of Daniel Kunkle and Gene Cooperman, 2009.

a GUI with Tkinter



one rotation



design of the GUI

Based on GAP example by Martin Schönert (1993) at
`http://www.gap-system.org/Doc/Examples/rubik.html`

Elements of the GUI:

- The GUI exports a canvas to show all six sides of a cube as a box that is cut open.
- The start button displays the initial state.
- With the shuffle button an animation starts, which applies a random sequence of moves.
- The random shuffle is undone with the solve.
- To clear the canvas, press the clear button.
- With the six buttons under the canvas, the user can apply any of the six rotations.
- The state of the cube is shown in the entry widget at the bottom.

computational group theory with GAP

1. GAP in SageMath
   - the GAP system
   - combinatorics and list manipulations
   - integer matrices and normal forms
2. Computing with Groups
   - permutation groups
   - group libraries
   - algebraic extensions
3. Rubik Magic's Cube
   - building a GUI
   - six rotations as group actions
   - analyzing Rubik's magic cube with GAP

six rotations as group actions

The state of the cube is describe by a permutation on 48 numbers.
Turning the front face is described by the permutation
$(17,19,24,22)(18,21,23,20)(6,25,43,16)(7,28,42,13)(8,30,41,11)$
which consist of 5 cycles, with meaning

$$\begin{aligned}
(17,19,24,22) &: 17 \mapsto 19 \mapsto 24 \mapsto 22 \mapsto 17, \\
(18,21,23,20) &: 18 \mapsto 21 \mapsto 23 \mapsto 20 \mapsto 18, \\
(6,25,43,16) &: 6 \mapsto 25 \mapsto 43 \mapsto 16 \mapsto 6, \\
(7,28,42,13) &: 7 \mapsto 28 \mapsto 42 \mapsto 13 \mapsto 7, \\
(8,30,41,11) &: 8 \mapsto 30 \mapsto 41 \mapsto 11 \mapsto 8.
\end{aligned}$$

Turning top, left, right, rear, and bottom are defined by
$(1,3,8,6)(2,5,7,4)(9,33,25,17)(10,34,26,18)(11,35,27,19)$,
$(9,11,16,14)(10,13,15,12)(1,17,41,40)(4,20,44,37)(6,22,46,35)$,
$(25,27,32,30)(26,29,31,28)(3,38,43,19)(5,36,45,21)(8,33,48,24)$,
$(33,35,40,38)(34,37,39,36)(3,9,46,32)(2,12,47,29)(1,14,48,27)$, and
$(41,43,48,46)(42,45,47,44)(14,22,30,38)(15,23,31,39)(16,24,32,40)$.

tuple assignments in Python

The cycle $(1,2,3) : 1 \mapsto 2 \mapsto 3 \mapsto 1$ applied to $(1,2,3)$ gives $(2,3,1)$:

```
>>> (a,b,c) = (1,2,3)
>>> (a,b,c) = (b,c,a)
>>> a,b,c
(2, 3, 1)
>>> (a,b,c) = (1,2,3)
>>> (c,a,b) = (a,b,c)
>>> a,b,c
(2, 3, 1)
```

The inverse is $(3,2,1) : 3 \mapsto 2 \mapsto 1 \mapsto 3$ to get back the identity:

```
>>> (a,b,c) = (c,a,b)
>>> a,b,c
(1, 2, 3)
>>> (a,b,c) = (2,3,1)
>>> (b,c,a) = (a,b,c)
>>> a,b,c
(1, 2, 3)
```

computational group theory with GAP

1. GAP in SageMath
   - the GAP system
   - combinatorics and list manipulations
   - integer matrices and normal forms
2. Computing with Groups
   - permutation groups
   - group libraries
   - algebraic extensions
3. Rubik Magic's Cube
   - building a GUI
   - six rotations as group actions
   - analyzing Rubik's magic cube with GAP

analyzing Rubik's magic cube with GAP

Following the example by Martin Schönert
at `http://www.gap-system.org/Doc/Examples/rubik.html`:

```
gap> cube := Group(
> ( 1, 3, 8, 6)( 2, 5, 7, 4)( 9,33,25,17)
> (10,34,26,18)(11,35,27,19),
> ( 9,11,16,14)(10,13,15,12)( 1,17,41,40)
> ( 4,20,44,37)( 6,22,46,35),
> (17,19,24,22)(18,21,23,20)( 6,25,43,16)
> ( 7,28,42,13)( 8,30,41,11),
> (25,27,32,30)(26,29,31,28)( 3,38,43,19)
> ( 5,36,45,21)( 8,33,48,24),
> (33,35,40,38)(34,37,39,36)( 3, 9,46,32)
> ( 2,12,47,29)( 1,14,48,27),
> (41,43,48,46)(42,45,47,44)(14,22,30,38)
> (15,23,31,39)(16,24,32,40) );
<permutation group with 6 generators>
```

the size of the group

To get the size of the group:

```
gap> Size(cube);
43252003274489856000
gap> Collected(Factors(last));
[ [ 2, 27 ], [ 3, 14 ], [ 5, 3 ], [ 7, 2 ], [ 11, 1 ] ]
```

The last result tells us that the size is $2^{27}3^{14}5^{3}7^{2}11$.

the orbits

```
gap> SizeScreen( [40, ] );;
gap> orbits := Orbits(cube, [1..48]);
[ [ 1, 3, 17, 14, 8, 38, 9, 41, 19,
      48, 22, 6, 30, 33, 43, 11, 46,
      40, 24, 27, 25, 35, 16, 32 ],
  [ 2, 5, 12, 7, 36, 10, 47, 4, 28,
      45, 34, 13, 29, 44, 20, 42, 26,
      21, 37, 15, 31, 18, 23, 39 ] ]
```

The first orbit contains the corner points,
the second orbit contains the points at the edges.

Clearly, the group cannot move a corner point onto a point at an edge.

a homomorphism of the free group into the cube group

```
gap> SizeScreen( [50,] );;
gap> G := FreeGroup("top","left","front","right",
> "rear","bottom");
<free group on the generators
[ top, left, front, right, rear, bottom ]>
gap> hom := GroupHomomorphismByImages( G, cube,
> GeneratorsOfGroup(G), GeneratorsOfGroup(cube) );
[ top, left, front, right, rear, bottom ] ->
[ (1,3,8,6)(2,5,7,4)(9,33,25,17)(10,34,26,
    18)(11,35,27,19),
  (1,17,41,40)(4,20,44,37)(6,22,46,35)(9,11,16,
    14)(10,13,15,12),
  (6,25,43,16)(7,28,42,13)(8,30,41,11)(17,19,24,
    22)(18,21,23,20),
  (3,38,43,19)(5,36,45,21)(8,33,48,24)(25,27,32,
    30)(26,29,31,28),
  (1,14,48,27)(2,12,47,29)(3,9,46,32)(33,35,40,
    38)(34,37,39,36),
  (14,22,30,38)(15,23,31,39)(16,24,32,40)(41,43,
    48,46)(42,45,47,44) ]
```

decomposing a random element

```
gap> r := Random(cube);
(1,40,9,14,35,46)(2,23,10,31,7,29,20,34,42,4,45,
18,36,13)(3,24,11,25,33,30,6,19,27,43,17,8)(5,
28,44)(12,37)(15,26,21)(16,41,22)(32,48,38)
gap> p := PreImagesRepresentative(hom, r);
top*front*right*top*right^-1*top^-1*front^
-1*top^-1*left^-1*top^-1*rear^
-1*top*rear*left*top^-2*left^-1*top*left*front^
-1*left*front*left^-1*top^-1*left^
2*rear*left*rear^-1*front^
-1*left*front*left*top^-2*left*top^2*left^
2*top*left*rear*left*rear^-1*top^
-1*bottom*front*bottom^-1*front^-1*top*left^
-1*top^-2*front^-1*bottom^-1*rear^
-1*bottom*left*rear*bottom^2*rear^-1*left^
-1*right^-1*top*front*right^-1
gap> Length( last );
68
```

verifying that the decomposition is correct

```
gap> Image( hom, p);
(1,40,9,14,35,46)(2,23,10,31,7,29,20,34,42,4,45,
18,36,13)(3,24,11,25,33,30,6,19,27,43,17,8)(5,
28,44)(12,37)(15,26,21)(16,41,22)(32,48,38)
gap> last = r;
true
```

Application to our GUI for Rubik's magic cube puzzle:

- Capture the state after some turns of the faces.
- Decompose the state using the group homomorphism.

Summary + Exercises

The GAP Manual Release 4.4.12 has 905 pages (pdf).

1. Generate random matrices with small numbers (e.g.: one digit), and compute the Hermite normal form.
   Experiment with the growth of the size of the numbers as the dimension of the matrix grows.
2. Extend the GUI `rubik_magic_cube.py`
   with a button `solve with GAP`.
   Pressing the button calls GAP to resolve the current state.