You can start GAP with the ggap icon (in Windows on the desktop, under OSX in the `Applications` folder), under Unix you call the `gap.sh` batch file in the `gap4r4/bin` directory. The program will start up and you will get window into which you can input text at a prompt `>`. We will write this prompt here as `gap>` to indicate that it is input to the program.

```
gap>
```

You now can type in commands (followed by a semicolon) and GAP will print out the result.
To leave GAP you can either call `quit;` or close the window.
You should be able to use the mouse and cursor keys for editing lines. (In the text version under Unix, the standard EMACS keybindings are accepted.) Note that changing prior entries in the worksheet does not change the prior calculation, but just executes the same command again. If you want to repeat a sequence of commands, pasting the lines in is a better approach.
You can use the online help with `?` to get documentation for particular commands.

```
gap> ?gcd
```

A double question mark `??` checks for keywords or parts of command names. If multiple sections apply GAP will list all of them with numbers, one can simply type `?`*`number`* to get a particular section.

```
gap> ??gcd
Help: several entries match this topic (type ?2 to get match [2])
[1] Reference: Gcd
[2] Reference: Gcd and Lcm
[3] Reference: GcdInt
[4] Reference: Gcdex
[5] Reference: GcdOp
[6] Reference: GcdRepresentation
[7] Reference: GcdRepresentationOp
[8] Reference: TryGcdCancelExtRepPolynomials
[9] GUAVA (not loaded): DivisorGCD
gap> ?6
```

You might want to create a transcript of your session in a text file. You can do this by issuing the command `LogTo(`*`filename`*`);` Note that the filename must include the path to a user writable directory. Under Windows

- Paths are always given with a forwards slash (/), even though the operating system uses a backslash.
- Instead of drive letters, the prefix `/cygdrive/`*`letter`* is used, e.g. `/cygdrive/c/` is the main drive.
- It can be convenient to use `DirectoryDesktop()` or `DirectoryHome()` to create a file on the desktop or in the `My Documents` folder.

```
LogTo("mylog")
LogTo("/cygdrive/c/Documents and Settings/Administrator/My Documents/logfile.txt");
LogTo(Filename(DirectoryDesktop(),"logfile.txt"));
```

**Note:** At the moment the directory functionality is in a *separate* file `mydirs.g` which can be found on the course web pages and needs to be read in first. To avoid a bootstrap issue with directory names, it is most convenient to put this file in GAP's library folder ( `/Applications/gap4r4/lib` under OSX, respectively `/Program Files/GAP/gap4r4/lib` under Windows) to enable simple reading with `ReadLib("mydirs.g");` You can end logging with the command

```
LogTo();
```

Some general hints:

- GAP is picky about upper case/lower case. `LogTo` is not the same as `logto`.
- All commands end in a semicolon.
- If you create larger input, it can be helpful to put it in a text file and to read it from GAP. This can be done with the command `Read("filename");` – the same issues about paths apply.
- By terminating a command with a double semicolon `;;` you can avoid GAP displaying the result. (Obviously, this is only useful if assigning it to a variable.)
- everything after a hash mark (`#`) is a comment.

We now do a few easy calculations. If you have not used GAP before, it might be useful to do these on the computer in parallel to reading.

**Integers and Rationals** GAP knows integers of arbitrary length and rational numbers:

```
gap> -3; 17 - 23;
-3
-6
gap> 2^200-1;
160693804425899027554196209234116260252220299378279283530137 5
gap> 123456/7891011+1;
2671489/2630337
```

The 'mod' operator allows you to compute one value modulo another. Note the blanks:

```
gap> 17 mod 3;
2
```

GAP knows a precedence between operators that may be overridden by parentheses and can compare objects:

```
gap> (9 - 7) * 5 = 9 - 7  * 5;
false
gap> 5/3<2;
true
```

You can assign numbers (or more general: every GAP object) to variables, by using the assignment operator `:=`. Once a variable is assigned to, you can refer to it as if it was a number. The special variables `last`, `last2`, and `last3` contain the results of the last three commands.

```
gap> a:=2^16-1; b:=a/(2^4+1);
65535
3855
gap> 5*b-3*a;
-177330
gap> last+5;
-177325
gap> last+2;
-177323
```

The following commands show some useful integer calculations related to quotient and remainder:

```
gap> Int(8/3); # round down
2
gap> QuoInt(76,23); # integral part of quotient
3
gap> QuotientRemainder(76,23);
[ 3, 7 ]
gap> 76 mod 23; # remainder (note the blanks)
7
gap> 1/5 mod 7;
3
```

**Lists** Objects separated by commas and enclosed in square brackets form a list.
Any collections of objects, including sets, are represented by such lists. (A `Set` in GAP is a sorted list.)

```
gap> l:=[5,3,99,17,2]; # create a list
[ 5, 3, 99, 17, 2 ]
gap> l[4]; # access to list entry
17
gap> l[3]:=22; # assignment to list entry
22
gap> l;
[ 5, 3, 22, 17, 2 ]
gap> Length(l);
5
gap> 3 in l; # element test
true
gap> 4 in l;
false
gap> Position(l,2);
5
gap> Add(l,17); # extension of list at end
gap> l;
[ 5, 3, 22, 17, 2, 17 ]
gap> s:=Set(l); # new list, sorted, duplicate free
[ 2, 3, 5, 17, 22 ]
gap> l;
```

```
[ 5, 3, 22, 17, 2, 17 ]
gap> AddSet(s,4); # insert in sorted position
gap> AddSet(s,5); # and avoid duplicates
gap> s;
[ 2, 3, 4, 5, 17, 22 ]
```

Results that consist of several numbers typically are represented as a list.

```
gap> DivisorsInt(96);
[ 1, 2, 3, 4, 6, 8, 12, 16, 24, 32, 48, 96 ]
gap> Factors(2^126-1);
[ 3, 3, 3, 7, 7, 19, 43, 73, 127, 337, 5419, 92737, 649657, 77158673929 ]
```

There are powerful list functions that often can save programming loops: `List`, `Filtered`, `ForAll`, `ForAny`, `First`. They take as first argument a list, and as second argument a function to be applied to the list elements or to test the elements. The notation `i -> xyz` is a shorthand for a one parameter function.

```
gap> l:=[5,3,99,17,2];
[ 5, 3, 99, 17, 2 ]
gap> List(l,IsPrime);
[ true, true, false, true, true ]
gap> List(l,i -> i^2);
[ 25, 9, 9801, 289, 4 ]
gap> Filtered(l,IsPrime);
[ 5, 3, 17, 2 ]
gap> ForAll(l,i -> i>10);
false
gap> ForAny(l,i -> i>10);
true
gap> First(l,i -> i>10);
99
```

A special case of lists are *ranges*, indicated by double dots. They can also be used to create arithmetic progressions:

```
gap> l:=[10..100];
[ 10 .. 100 ]
gap> Length(l);
91
```

**Vectors and Matrices** Lists are also used to form vectors and matrices:
A *vector* is simply a list of numbers. A list of (row) vectors is a matrix. GAP knows matrix arithmetic.

```
gap> vec:=[1,2,3,4];
[ 1, 2, 3, 4 ]
gap> vec[3]+2;
5
gap> 3*vec+1;
[ 4, 7, 10, 13 ]
```

```
gap> mat:=[[1,2,3,4],[5,6,7,8],[9,10,11,12]];
[ [ 1, 2, 3, 4 ], [ 5, 6, 7, 8 ], [ 9, 10, 11, 12 ] ]
gap> mat*vec;
[ 30, 70, 110 ]
gap> mat:=[[1,2,3],[5,6,7],[9,10,12]];;
gap> mat^5;
[ [ 289876, 342744, 416603 ], [ 766848, 906704, 1102091 ],
  [ 1309817, 1548698, 1882429 ] ]
gap> DeterminantMat(mat);
-4
```

The command `Display` can be used to get a nicer output:

```
gap> 3*mat^2-mat;
[ [ 113, 130, 156 ], [ 289, 342, 416 ], [ 492, 584, 711 ] ]
gap> Display(last);
[ [  113,  130,  156 ],
  [  289,  342,  416 ],
  [  492,  584,  711 ] ]
```

**Roots Of Unity** The expression `E(`$n$`)` is used to denote the $n$-th root of unity ($e^{\frac{2\pi i}{n}}$):

```
gap> root5:=E(5)-E(5)^2-E(5)^3+E(5)^4;;
gap> root5^2;
5
```

**Finite Fields** To compute in finite fields, we have to create special objects to represent the residue classes (Internally, GAP uses so-called Zech Logarithms and represents all nonzero elements as power of a generator of the cyclic multiplicative group.)

```
gap> gf:=GF(7);
GF(7)
gap> One(gf);
Z(7)^0
gap> a:=6*One(gf);
Z(7)^3
gap> b:=3*One(gf);
Z(7)
gap> a+b;
Z(7)^2
gap> Int(a+b);
2
```

Non-prime finite fields are created in the same way with prime powers, note that GAP automatically represents elements in the smallest field possible.

```
gap> Elements(GF(16));
[ 0*Z(2), Z(2)^0, Z(2^2), Z(2^2)^2, Z(2^4), Z(2^4)^2, Z(2^4)^3, Z(2^4)^4,
  Z(2^4)^6, Z(2^4)^7, Z(2^4)^8, Z(2^4)^9, Z(2^4)^11, Z(2^4)^12, Z(2^4)^13,
  Z(2^4)^14 ]
```

We can also form matrices over finite fields.

```
gap> mat:=[[1,2,3],[5,6,7],[9,10,12]]*One(im);
[ [ Z(7)^0, Z(7)^2, Z(7) ], [ Z(7)^5, Z(7)^3, 0*Z(7) ],
  [ Z(7)^2, Z(7), Z(7)^5 ] ]
gap> mat^-1+mat;
[ [ Z(7)^4, Z(7)^4, Z(7)^4 ], [ Z(7)^3, Z(7)^0, Z(7)^5 ],
  [ Z(7), Z(7)^0, Z(7)^3 ] ]
```

**Integers modulo** If we want to compute in the integers modulo $n$ (what we called $\mathbb{Z}_n$ in the lecture) without need to always type `mod` we can create obhjects that immediately reduce their arithmetic modulo $n$:

```
gap> im:=Integers mod 6; # represent numbers for ‘‘modulo 6’’ calculations
(Integers mod 6)
```

To convert "ordinary" integers to residue classes, we have to multiply them with the"One" of these residue classes, the command `Int` converts back to ordinary integers:

```
gap> a:=5*One(im);
ZmodnZObj( 5, 6 )
gap> b:=3*One(im);
ZmodnZObj( 3, 6 )
gap> a+b;
ZmodnZObj( 2, 6 )
gap> Int(last);
2
```

(If one wants one can get all residue classes or – for example test which are invertible).

```
gap> Elements(im);
[ ZmodnZObj(0,6),ZmodnZObj(1,6),ZmodnZObj(2,6),ZmodnZObj(3,6),
  ZmodnZObj(4,6),ZmodnZObj(5,6) ]
gap> Filtered(last,x->IsUnit(x));
[ ZmodnZObj( 1, 6 ), ZmodnZObj( 5, 6 ) ]
gap> Length(last);
2
```

If we calculate modulo a *prime* the default output looks a bit different.

```
gap> im:=Integers mod 7;
GF(7)
```

(The name `GF` stands for *Galois Field*, explanation later.) Also elements display differently. Therefore it is convenient to once issue the command

```
gap> TeachingMode(true);
```

which (amongst others) will simplify the display.

**Groups and Homomorphisms** We can write permutations in cycle form and multiply (or invert them):

```
gap> a:=(1,2,3,4)(6,5,7);
(1,2,3,4)(5,7,6)
gap> a^2;
(1,3)(2,4)(5,6,7)
gap> a^-1;
(1,4,3,2)(5,6,7)
gap> b:=(1,3,5,7)(2,6,8);; a*b;
```

**Note:** GAP multiplies permutations *from left to right*, i.e. $(1,2,3)\cdot(2,3) = (1,3)$. (This might differ from what you used in prior courses.)
A group is generated by the command `Group`, applied to generators (permutations or matrices). It is possible to compute things such as `Elements`, group `Order` or `ConjugacyClasses`.

```
gap> g:=Group((1,2,3,4,5),(2,5)(3,4));
Group([ (1,2,3,4,5), (2,5)(3,4) ])
gap> Elements(g);
[ (), (2,5)(3,4), (1,2)(3,5), (1,2,3,4,5), (1,3)(4,5), (1,3,5,2,4),
  (1,4)(2,3), (1,4,2,5,3), (1,5,4,3,2), (1,5)(2,4) ]
gap> Order(g);
10
gap> c:=ConjugacyClasses(g);
[ ()^G, (2,5)(3,4)^G, (1,2,3,4,5)^G, (1,3,5,2,4)^G ]
gap> List(c,Size);
[ 1, 5, 2, 2 ]
gap> List(c,Representative);
[ (), (2,5)(3,4), (1,2,3,4,5), (1,3,5,2,4) ]
```

Homomorphisms can be created by giving group generators and their images under the map. (GAP will check that the map indeed defines a homomorphism first. One can use `GroupHomomorphismByImagesNC` to skip this test.)

```
gap> g:=Group((1,2,3),(3,4,5));;
gap> mat1:=[ [ 0, -E(5)-E(5)^4, E(5)+E(5)^4 ], [ 0, -E(5)-E(5)^4, -1 ],
>   [ 1, 1, E(5)+E(5)^4 ] ];;
gap> mat2:=[ [ 1, 0, E(5)+E(5)^4 ], [ -E(5)-E(5)^4, 0, E(5)+E(5)^4 ],
>  [ -E(5)-E(5)^4, -1, -1 ] ];;
gap> img:=Group(mat1,mat2);;
gap> hom:=GroupHomomorphismByImages(g,img,[(1,2,3),(3,4,5)],[mat1,mat2]);
[ (1,2,3), (3,4,5) ] ->
[ [[0,-E(5)-E(5)^4,E(5)+E(5)^4],[0,-E(5)-E(5)^4,-1],[1,1,E(5)+E(5)^4]],
  [[1,0,E(5)+E(5)^4],[-E(5)-E(5)^4,0,E(5)+E(5)^4],[-E(5)-E(5)^4,-1,-1]] ]
gap> Image(hom,(1,2,3,4,5));
[ [E(5)+E(5)^4,0,1], [E(5)+E(5)^4,1,0], [ -1, 0, 0 ] ]
gap> r:=[ [ 1, E(5)+E(5)^4, 0 ], [ 0, E(5)+E(5)^4, 1 ], [ 0, -1, 0 ] ];;
gap> PreImagesRepresentative(hom,r);
(1,4,2,3,5)
```

**Group Actions** The general setup for group actions is to give:

- The acting group
- The domain (this is optional if one calculates the `Orbit` of a point
- In the case of computing an orbit, the staring point $\omega$
- (There is the option to give group generators and acting images as extra argument. For the moment just forget it.)
- A function $f(\omega, g)$ that is used to compute $\omega^g$. If not given the function `OnPoints`, which returns $\omega \hat{} g$, is used.

Common functions are: `Orbit`, `Orbits`, `Stabilizer`. The function `ActionHomomorphism` can be used to compute a homomorphism into $S_\Omega$ given by the permutation action.

```
gap> Orbit(g,1);
[ 1, 2, 3, 4 ]
gap> Orbit(g,[1,2],OnSets);
[ [ 1, 2 ], [ 2, 3 ], [ 3, 4 ], [ 1, 3 ], [ 1, 4 ], [ 2, 4 ] ]
gap> Orbit(g,[1,2],OnTuples);
[ [ 1, 2 ], [ 2, 3 ], [ 2, 1 ], [ 3, 4 ], [ 1, 3 ], [ 3, 2 ], [ 4, 1 ],
  [ 2, 4 ], [ 4, 3 ], [ 3, 1 ], [ 4, 2 ], [ 1, 4 ] ]
gap> Orbits(Group((1,2,3,4),(1,3)(2,4)),Combinations([1..4],2),OnSets);
[   [ [ 1, 2 ], [ 2, 3 ], [ 3, 4 ], [ 1, 4 ] ],
    [ [ 1, 3 ], [ 2, 4 ] ]   ]
gap> Stabilizer(g,1);
Group([ (2,3,4), (3,4) ])
```