Mini-course on GAP – Lecture 2

Leandro Vendramin

Universidad de Buenos Aires

Dalhousie University, Halifax
January 2020

Let us start with basic GAP programming.
Outline:

- Objects and variables
- Conditionals
- Functions
- Strings
- Lists
- Ranges
- Sets
- Loops

Objects and variables

An object is something that we can assign to a variable. So an object could be either a number, a string, a group, a field, an element of a group, a group homomorphism, a ring, a matrix, a vector space...

Objects and variables

To assign an object to a variable one uses the operator := as the following example shows:

```
gap> p := 32;;
gap> p;
32
gap> p = 32;
true
gap> p := p+1;;
gap> p;
33
gap> p = 32;
false
```

WARNING:
Don't forget that the symbols = (conditional) and := (assignment operator) are different!

Objects and variables

What if I forgot to assign the result of a calculation for further use? We can do the following:

```
gap> 2*(5+1)-6;
6
gap> n := last;
6
```

One also has `last2` and `last3`.

Conditionals

There are three very important operators: `not`, `and`, `or`. We also have comparison operators; for example the expression `x<>y` returns `true` if `x` and `y` are different, and `false` otherwise.

```
gap> x := 20;; y := 10;;
gap> x <> y;
true
gap> x > y;
true
gap> (x > 0) or (x < y);
true
gap> (x > 0) and (x < y);
false
gap> (2*y < x);
false
gap> (2*y <= x);
true
gap> not (x < y);
true
```

Conditionals

The `if` statement allows one to execute statements depending on the value of some boolean expression.

```
gap> n := 10;;
gap> if n mod 2 = 0 then
> n := n/2;
> else
> n := (n+1)/2;
> fi;
gap> n;
5
```

Better examples will appear soon, we need to use functions!

Functions

There are two ways of constructing functions. For example, to construct the map $x \mapsto x^2$ either we use the one-line definition

```
gap> square := x->x^2;
function( x ) ... end
```

or the classical

```
gap> square := function(x)
> return x^2;
> end;
function( x ) ... end
```

In both cases we will obtain the same result!

Functions

One can also define functions with no arguments.

```
gap> hi := function()
> Display("Hello world");
> end;
function( ) ... end
gap> hi();
Hello world
```

Functions

Let us write a function to compute the map

$$f : n \mapsto \begin{cases} n^3 & \text{si } n \equiv 0 \bmod 3, \\ n^5 & \text{si } n \equiv 1 \bmod 3, \\ 0 & \text{otherwise .} \end{cases}$$

Functions

Here is the code and some experiments:

```
gap> f := function(n)
> if n mod 3 = 0 then
> return n^3;
> elif n mod 3 = 1 then
> return n^5;
> else
> return 0;
> fi;
> end;
function( n ) ... end
gap> f(10);
100000
gap> f(5);
0
gap> f(4);
1024
```

Functions

The Fibonacci sequence f_n is defined as $f_1 = f_2 = 1$ and

$$f_{n+1} = f_n + f_{n-1}$$

for $n \geq 2$. The following function computes Fibonacci numbers:

```
gap> fibonacci := function(n)
> if n = 1 or n = 2 then
> return 1;
> else
> return fibonacci(n-1)+fibonacci(n-2);
> fi;
> end;
function( n ) ... end
gap> fibonacci(10);
55
```

Question: Can you compute f_{100} with this method?

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 ]
```

Lists

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 ]
```

Lists

`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 ] ]
```