Introdução ao GAP

Ricardo Franquiz & Osnel Broche
Universidade Federal de Lavras

IX Workshop de Matemática e Matemática Aplicada

4 de dezembro de 2024
Ouro Preto

Um primeiro contacto com GAP

O que é o GAP?

O **GAP** (Groups, Algorithms, Programming) é um sistema de álgebra discreta computacional inicialmente orientado à Teoria dos Grupos, embora hoje em dia este sistema possui muitas outras ferramentas desenvolvidas para o analise das estruturas algébricas que aparecem nas distintas áreas dentro da álgebra como Representações de grupos finitos, Representações de álgebras, Álgebra homológica, etc.

Um primeiro contacto com GAP

O que é o GAP?

O GAP foi originalmente desenvolvido pelo departamento de matemática (LDFM), da Universidade Técnica de Aquisgrán, Alemanha, entre os anos de 1986 e 1997. Posteriormente a manutenção e o desenvolvimento deste passou a ser coordenado por um grupo de universidades na europa e os Estados Unidos entre as quais estão a Universidade Técnica de Aquisgrán, a Universidade de Saint Andrews na Escocia, a Universidade Técnica de Brunswick, a Universidade Estadual de Colorado em Fort Collins, Colorado, a Universidade Técnica de Kaiserslautern.

Um primeiro contacto com GAP

O que é o GAP?

Algumas das características que descrevem o GAP são:

- GAP foi escrito na linguagem de programação C, este possui um intérprete da linguagem GAP e os algoritmos sobre funções básicas.
- GAP é um sistema com licencia GNU (General Public License). Ele é distribuído livremente de forma tal que o usuários podem fazer modificações no sistema.
- GAP possui uma grande biblioteca de funciones (escrita na linguagem GAP) que implementa a grande parte dos algoritmos.
- Possui varias bibliotecas de dados de estruturas algébricas entre a que destaca uma biblioteca de grupos finitos de ordem menor que 2000.
- Grande parte do desenvolvimento e evolução do GAP se deve à contribuição dos usuários com pacotes (extensões autocontidas do núcleo do sistema).

Um primeiro contacto com GAP

O que é o GAP?

- A distribuição do GAP é feita principalmente pelo site do sistema

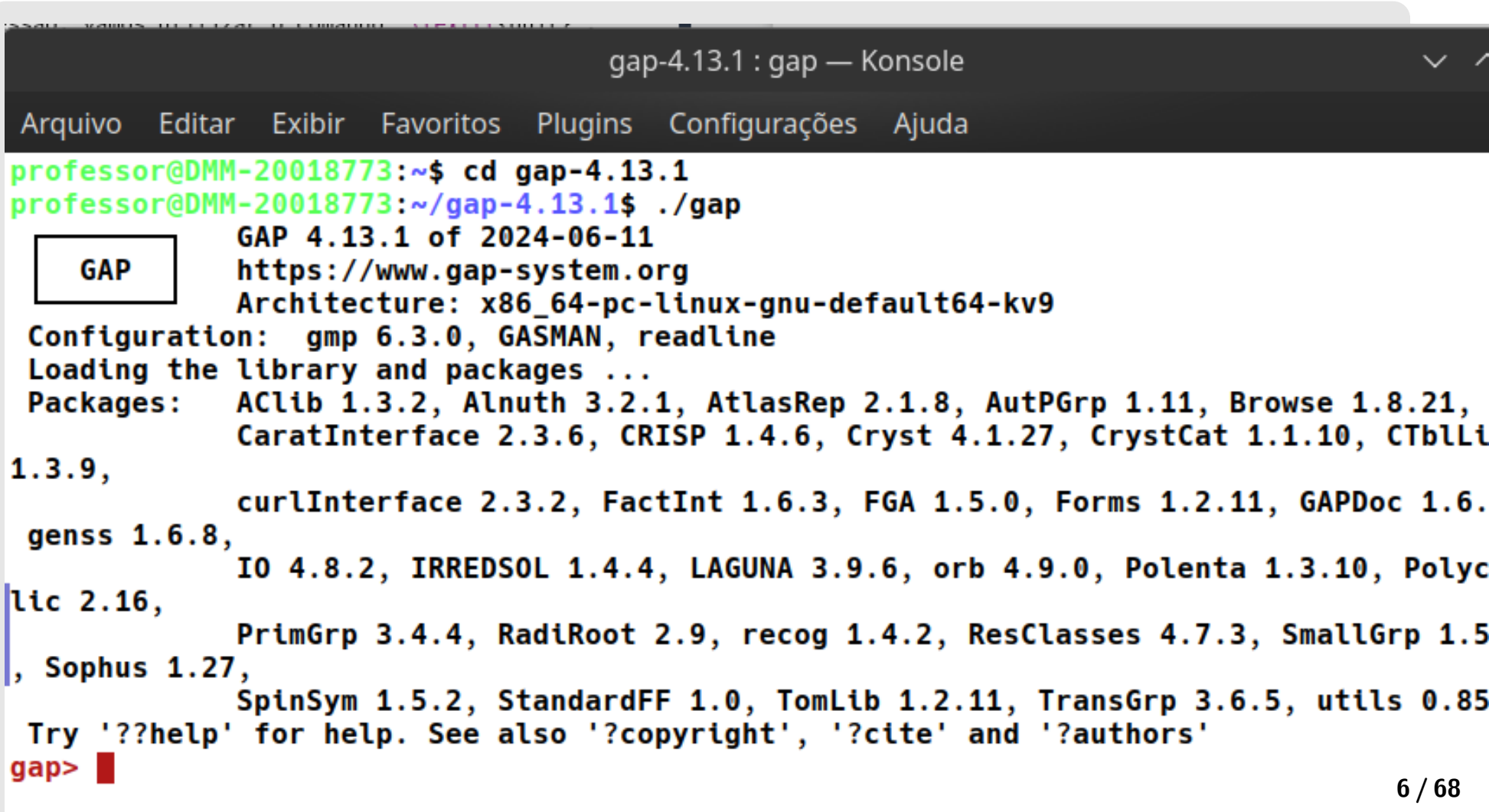
  `https://www.gap-system.org/`

- Atualmente encontra-se na versão 4.13.1, lançada em Julho de 2024.
- No site podem encontrar tanto as instruções de instalação como uma extensa documentação que inclui desde material básico para aprender a usar este até um completo manual sobre o sistema e os pacotes desenvolvidos.
- Atualmente em outros softwares como SINGULAR e SageMath tem incluída uma interface de GAP.

Um primeiro contato com GAP

Funcionamento básico

Uma vez instalado o GAP, ao iniciar aparecerá uma tela da seguinte forma:



Um primeiro contato com GAP

Funcionamento básico

- Qualquer instrução deve ser finalizado com um ponto e vírgula ";" para o GAP mostrar em a resposta de executar tal instrução.
- Se queremos executar uma instrução mas que a mesma não apareça na tela encerramos a ordem com ";;".
- Para escrever comentários nos códigos devemos escrever primeiro "#"e depois o comentário.
- Para encerrar a sessão, vamos utilizar o comando "`quit;`"

Um primeiro contato com GAP

Funcionamento básico

- Qualquer instrução deve ser finalizado com um ponto e vírgula ";" para o GAP mostrar em a resposta de executar tal instrução.
- Se queremos executar uma instrução mas que a mesma não apareça na tela encerramos a ordem com ";;".
- Para escrever comentários nos códigos devemos escrever primeiro "#"e depois o comentário.
- Para encerrar a sessão, vamos utilizar o comando `"quit;"`

Um primeiro contato com GAP

Funcionamento básico

- Qualquer instrução deve ser finalizado com um ponto e vírgula ";" para o GAP mostrar em a resposta de executar tal instrução.
- Se queremos executar uma instrução mas que a mesma não apareça na tela encerramos a ordem com ";;".
- Para escrever comentários nos códigos devemos escrever primeiro "#"e depois o comentário.
- Para encerrar a sessão, vamos utilizar o comando `"quit;"`

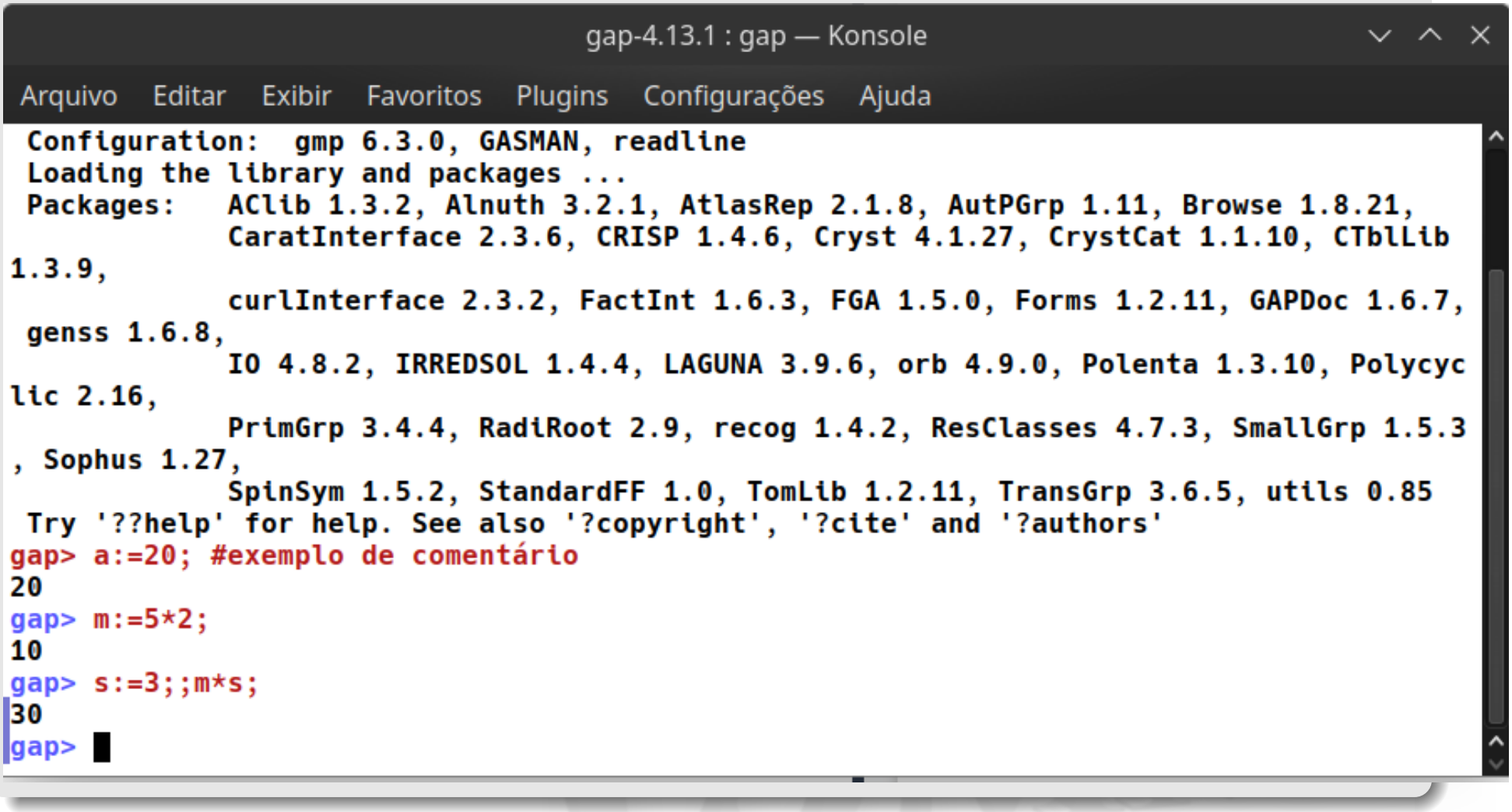
Um primeiro contato com GAP

Funcionamento básico

- Qualquer instrução deve ser finalizado com um ponto e vírgula ";" para o GAP mostrar em a resposta de executar tal instrução.
- Se queremos executar uma instrução mas que a mesma não apareça na tela encerramos a ordem com ";;".
- Para escrever comentários nos códigos devemos escrever primeiro "#"e depois o comentário.
- Para encerrar a sessão, vamos utilizar o comando "`quit;`"

Um primeiro contato com GAP

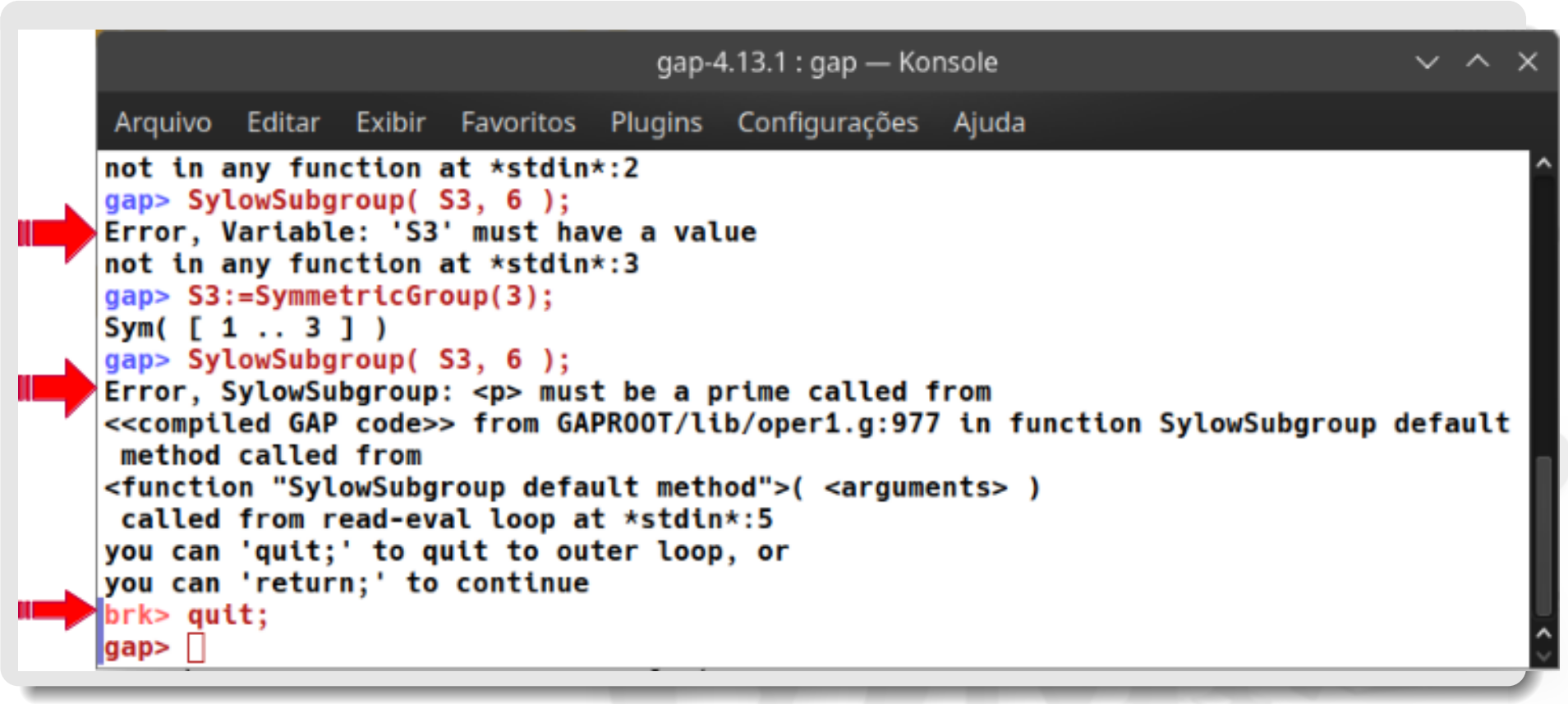
Funcionamento básico



Um primeiro contato com GAP

Funcionamento básico

No caso de executar de forma errada uma instrução no GAP, este mostrará na tela uma mensagens de erro da seguinte forma:



Um primeiro contato com GAP

Funcionamento básico

GAP permite comparar duas expressões entre si. Neste caso, o símbolo "="é usado para igualdade, enquanto o símbolo "<>"é usado para denotar $\neq$.

```
gap> x:=97;;y:=77;;
gap> x=y;
false
gap> x<>y;
true
gap>
```

Um primeiro contato com GAP

Funcionamento básico

Uma cadeia de caracteres é uma expressão delimitada pelo símbolo ".

```
gap> Exemplo:= "IX Workshop em Matemática e Matemática Aplicada- Ouro preto";
"IX Workshop em Matemática e Matemática Aplicada- Ouro preto"
gap>
```

Obs.: Cadeias de caracteres podem ser comparadas:

```
gap> WMMA:="Workshop de Matemática e Matemática Aplicada";; CC:="Workshop de Matemática
 e Matemática Aplicada";;
gap> WMMA=CC;
true
gap>
```

Um primeiro contato com GAP

Funções Print, LogTo e Read

GAP bem equipado com a função `Print` que permite imprimir dados na tela.

```
gap> n:=10;; m:=5;;
gap> Print(n, " es um número maior que ", m);
10 es um número maior que 5
gap>
```

Um primeiro contato com GAP

Funções Print, LogTo e Read

GAP bem equipado com a função `LogTo("NomedoArquivo")` que permite guardar em um arquivo .txt nosso trabalho. Este arquivo ficará guardado na pasta que contém os

```
gap> LogTo("Teste_GAP_9WMMA");
gap> x:=97;;y:=77;;
gap> x=y;
false
gap> WMMA:="Workshop de Matemática e Matemática Aplicada";; CC:="Workshop de Matemática
 e Matemática Aplicada";;
gap> WMMA<>CC;
false
gap> LogTo();
gap>
```

Os arquivos gerados por LogTo() funcionam como um diário, onde ficará registrado absolutamente todo o que foi feito na sesão.

Um primeiro contato com GAP

Funções Print, LogTo e Read

Um outra função muito útil do GAP é `Read(Arquivo.gap)`. Esta função Read() é uma ferramenta que vai permitir organizar seu trabalho no GAP. Se você precisar executar a mesma sequência de instruções várias vezes, pode criar um arquivo .gap e carregá-lo usando Read quando necessário.

Um primeiro contato com GAP

Funções aritméticas básicas e estruturas algébricas

GAP bem equipado com as operações aritméticas básicas que permitiria usar este como calculadora se quiser:

`a + b` denota soma dos elementos `a` e `b`.

`a - b` denota substração dos elementos `a` e `b`.

`a * b` denota multiplicação dos elementos `a` e `b`.

`a / b` denota divisão dos elementos `a` e `b`.

`a^b` denota a operação de potenciação dos elementos `a` e `b`.

Uma operação especial que o GAP possui é o cálculo de inteiros modulares. Assim, se `a` e `b` são inteiros, `a mod b` retorna um número `r` que está entre 0 e $|b|$.

Um primeiro contato com GAP

Funções aritméticas básicas e estruturas algébricas

GAP oferece a possibilidade de trabalhar com algumas estruturas algébricas como corpos, anéis, álgebras e grupos.

- **Corpos:** Para gerar o corpo $\mathbb{F}_q$, em que $q = p^n$ é uma potência de um primo, basta escrever `GF(q)`.

Um primeiro contato com GAP

Funções aritméticas básicas e estruturas algébricas

GAP oferece a possibilidade de trabalhar com algumas estruturas algébricas como corpos, anéis, álgebras e grupos.

- **Anéis:** Existe a possibilidade de gerar um anel de inteiros módulo algum n. Para isso, basta escrever a instrução `Integers mod n`.

Um primeiro contato com GAP

Funções aritméticas básicas e estruturas algébricas

GAP oferece a possibilidade de trabalhar com algumas estruturas algébricas como corpos, anéis, álgebras e grupos.

- **Grupos:** Estas são as estruturas mais desenvolvidas no GAP. De fato, GAP bem equipado com uma biblioteca que contém 423.164.062 grupos de ordem no máximo 2000 com excepção da ordem 1024. Estes grupos encontram-se classificados e listados no GAP mediante uma indexação feita por pares ordenados de números.

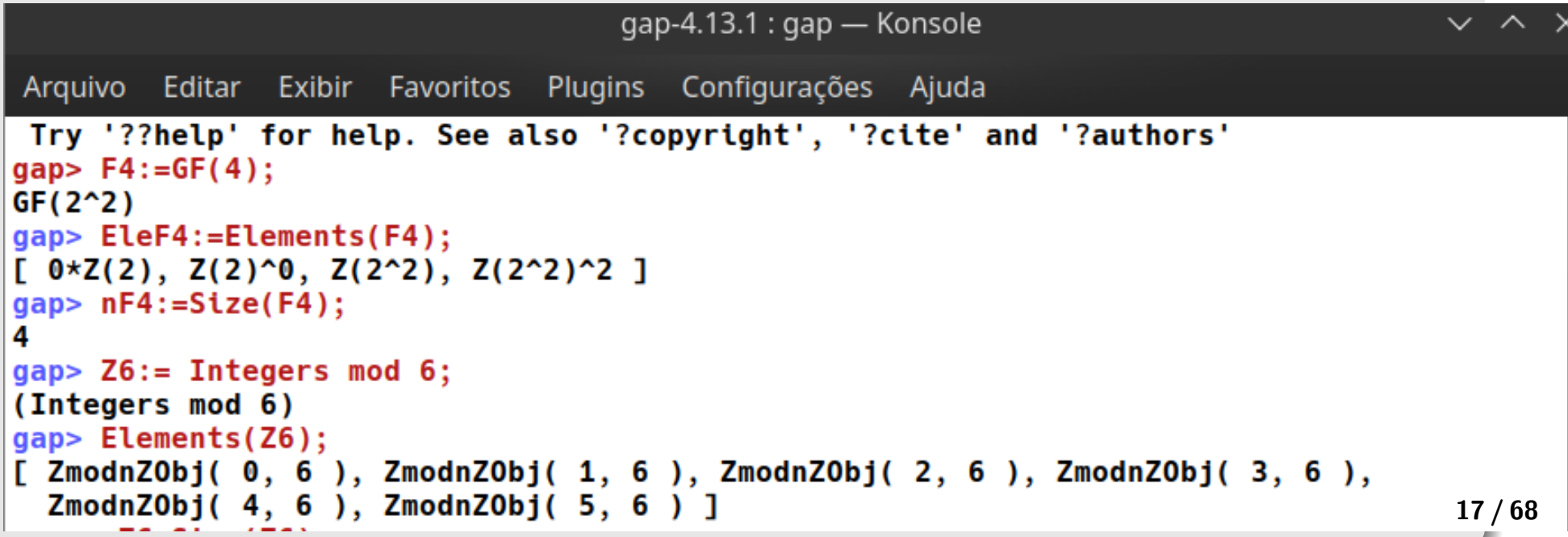
Um primeiro contato com GAP

Funções aritméticas básicas e estruturas algébricas

GAP possui uma série de funções básicas que permitem tanto extrair informações das estruturas algébricas mencionadas anteriormente.

Se queremos saber os elementos de qualquer dessas estruturas, basta usar a função `Elements( )`.

Se queremos saber o tamanho o número de elementos da estrutura que estamos usando, basta usar a função `Size( )`.



Um primeiro contato com GAP

Estruturas de dados: Listas

As listas são estruturas de dados que permitem o armazenamento de sequência de objetos de qualquer tipo.

Uma lista tem sempre os elementos dispostos entre colchetes, separados por vírgulas.

Uma lista deve ser previamente definida, mesmo que seja vazia.

```
gap> L := [1,3,5,8];
[ 1, 3, 5, 8 ]
gap> V:=[];  #Lista vazia!
[  ]
gap>
```

Um primeiro contato com GAP

Estruturas de dados: Listas

Algumas funções para manipular listas no GAP:

- `L[i]` retorna o elemento da lista `L` na posição i.

```
gap> L := [1,3,5,8];
[ 1, 3, 5, 8 ]
gap> L[2];
3
```

- `IsList(Lista)` é uma função booleana que retorna `true` se a estrutura é uma lista ou `false` em caso contrário.

```
gap> L := [1,3,5,8];
[ 1, 3, 5, 8 ]
gap> IsList(L);
true
```

Um primeiro contato com GAP

Estruturas de dados: Listas

Algumas funções para manipular listas no GAP:

- `Add(L, i)` Anexa o elemento *i* no final da lista `L`.

```
gap> L := [1,3,5,8];
[ 1, 3, 5, 8 ]
gap> Add(L,3);
gap> L;
[ 1, 3, 5, 8, 3 ]
```

- `Append(L1, L2)` Concatena a lista `L1` com a segunda lista `L2`, aparecendo os elementos da lista `L2` na lista `L1`.

```
gap> L1 := [1,3,5,8];
[ 1, 3, 5, 8 ]
gap> L2 := [2,4,6,0];
[ 2, 4, 6, 0 ]
gap> Append(L1,L2);
gap> L1;
[ 1, 3, 5, 8, 2, 4, 6, 0 ]
```

Um primeiro contato com GAP

Estruturas de dados: Listas

- `Length(L)` Retorna a quantidade de elementos qa lista `L`.

```
gap> L:=[1,3,5,8];
[ 1, 3, 5, 8 ]
gap> Length(L);
4
```

- `[n..m]` Denota uma lista definida a partir de um intervalo, começando no n e acabando no m, com saltos unitários.
- `x in L` Confirma se um objeto `x` está na lista `L`. Caso o objeto faça parte da lista, GAP retorna `true`. Em outro caso retorna `false`.

```
gap> L:=[1,3,5,8];
[ 1, 3, 5, 8 ]
gap> 3 in L;
true
gap> 0 in L;
false
```

Programando com GAP

Criando programas com poucas ferramentas

Até agora exploramos algumas funções predefinidas dentro do GAP. No entanto, o GAP nos oferece ainda mais flexibilidade, permitindo que definamos nossas próprias funções personalizadas.

Vamos ilustrar como criar tais funções com um simples exemplo, elevar um número ao quadrado, feito de duas formas diferentes:

Programando com GAP

Criando programas com poucas ferramentas

- Uma forma de criar uma função é atribuir uma regra a uma variável. Para atribuir uma regra usamos o símbolo ->".

Exemplo:

```
quad := x->x^2;
```

Programando com GAP

Criando programas com poucas ferramentas

- Uma outra forma é usar o comando `function( )`. Basicamente, este comando cria pequenos programas que podem definir funções.

  Neste caso devemos indicar ao GAP o que ele deve retornar. Para isso usamos o comando `return` *"instrução"*. Finalmente devemos terminar a função com o comando `end;`

Exemplo:

```
quad2 := function(x)
return x^2;
end;
```

Programando com GAP

Criando programas com poucas ferramentas

Todas estas instruções no GAP retornam o mesmo objeto, ou seja, o quadrado de um número

```
gap> quad := x -> x^2;
function( x ) ... end
gap> quad(5);
25
gap> quad2 := function(x)
>                 return x^2;
>                 end;
function( x ) ... end
gap> quad2(5);
25
gap> 5^2;
25
```

Exercício!: A função raiz quadrada de um número está predefinida no GAP como o comando `Sqrt( )`. Crie está função usando as formas descritas anteriormente.

Programando com GAP

Criando programas com poucas ferramentas

Todas estas instruções no GAP retornam o mesmo objeto, ou seja, o quadrado de um número

```
gap> quad := x -> x^2;
function( x ) ... end
gap> quad(5);
25
gap> quad2 := function(x)
>                 return x^2;
>                 end;
function( x ) ... end
gap> quad2(5);
25
gap> 5^2;
25
```

Exercício!: A função raiz quadrada de um número está predefinida no GAP como o comando `Sqrt( )`. Crie está função usando as formas descritas anteriormente.

Programando com GAP

Criando programas com poucas ferramentas

Podemos criar pequenos programas usando listas e atribuições.

Para isso vamos usar a função `List( )`. Esta função do GAP cria listas com dados.

Exemplo:

```
L:=[1,3,5,8];
List(L, y -> y^2);
```

Usando funções já criadas:

Example:

```
quad:=x -> x^2;
L:=[1,3,5,8];
List(L, y -> quad(y));
```

Programando com GAP

Criando programas com poucas ferramentas

Podemos criar pequenos programas usando listas e atribuições.

Para isso vamos usar a função `List( )`. Esta função do GAP cria listas com dados.

Exemplo:

```
L:=[1,3,5,8];
List(L, y -> y^2);
```

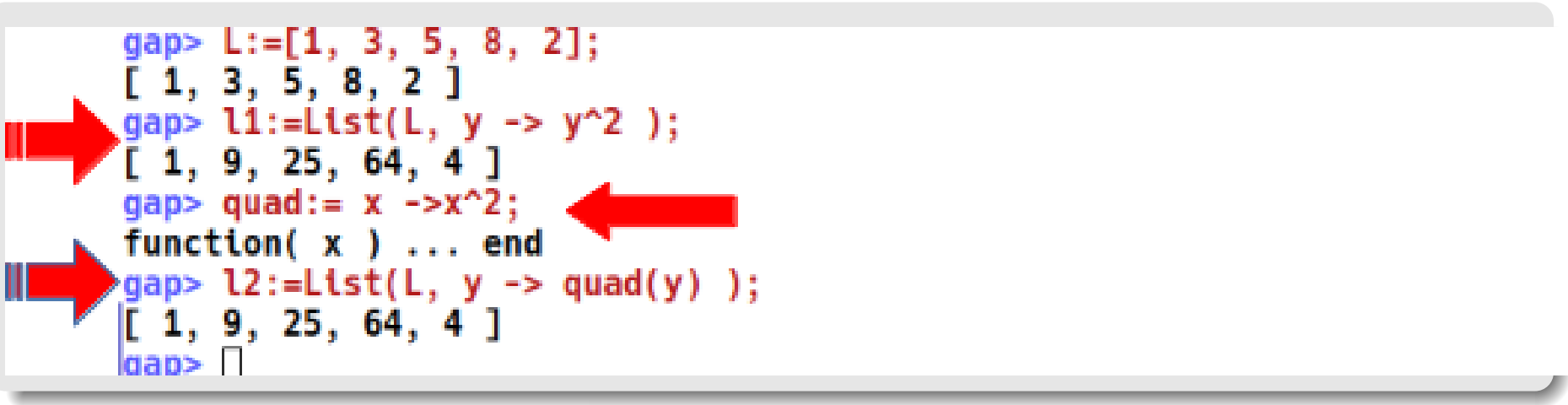
Usando funções já criadas:

Example:

```
quad:=x -> x^2;
L:=[1,3,5,8];
List(L, y -> quad(y));
```

Programamando com GAP

Criando programas com poucas ferramentas



Programando com GAP

Criando programas com poucas ferramentas: Usando Listas

Vamos criar uma função que selecione os números pares em uma lista de números inteiros.

```
FiltrarPares := function(lista)
  return Filtered(lista, x -> IsInt(x/2));
end;
```

A função `FiltrarPares` usa as seguintes duas funções do GAP:

- `Filtered( )` é uma função que filtra os elementos de uma lista baseado em alguma condição dada.
- `IsInt( )` é uma função booleana que verifica se um elemento é um inteiro ou não. Se o elemento for inteiro, esta função retorna `true`, caso contrário retorna `false`.

Programando com GAP

Criando programas com poucas ferramentas: Usando Listas

Vamos criar uma função que selecione os números pares em uma lista de números inteiros.

```
FiltrarPares := function(lista)
  return Filtered(lista, x -> IsInt(x/2));
end;
```

A função `FiltrarPares` usa as seguintes duas funções do GAP:

- `Filtered( )` é uma função que filtra os elementos de uma lista baseado em alguma condição dada.
- `IsInt( )` é uma função booleana que verifica se um elemento é um inteiro ou não. Se o elemento for inteiro, esta função retorna `true`, caso contrário retorna `false`.

Programando com GAP

Criando programas com poucas ferramentas

```
gap> L:=[1, 3, 5, 8, 2];
[ 1, 3, 5, 8, 2 ]
gap> FiltrarPares := function(lista)
> return Filtered(lista, x -> IsInt(x/2));
> end;
function( lista ) ... end
gap> FiltrarPares(L);
[ 8, 2 ]
```

Alternativamente poderíamos criar a mesma função usando a função booleana predefinida `IsEvenInt( )` que verifica se um elemento é um número par ou não.

```
FiltrarPares := function(lista)
  return Filtered(lista, x -> IsEvenInt(x));
end;
```

Programando com GAP

Criando programas com poucas ferramentas

```
gap> FiltrarPares := function(lista)
> return Filtered(lista, x -> IsEvenInt(x));
> end;
function( lista ) ... end
gap> FiltrarPares(L);
[ 8, 2 ]
```

Exercício: Crie a mesma função para filtrar os números impares de uma lista

Programando com GAP

Criando programas com poucas ferramentas

GAP possui a função `Reversed( )` que permite inverter a ordem das listas. Usando `Reversed` podemos criar uma função que inverte a ordem de qualquer lista.

```
InverterLista := function(lista)
  return Reversed(lista);
end;
```

Podemos criar também funções que usem outras funções de listas. Por exemplo, usar `InverterLista` e criar uma função que retorne os números pares da lista invertida.

```
FiltrarParesInver := function(lista)
   return Filtered(InverterLista(lista),x
   ->IsEvenInt(x));
end;
```

Programando com GAP

Criando programas com poucas ferramentas

GAP possui a função `Reversed( )` que permite inverter a ordem das listas. Usando `Reversed` podemos criar uma função que inverte a ordem de qualquer lista.

```
InverterLista := function(lista)
  return Reversed(lista);
end;
```

Podemos criar também funções que usem outras funções de listas. Por exemplo, usar `InverterLista` e criar uma função que retorne os números pares da lista invertida.

```
FiltrarParesInver := function(lista)
   return Filtered(InverterLista(lista),x
   ->IsEvenInt(x));
end;
```

Programando com GAP

Criando programas com poucas ferramentas

```
gap> L:=[1, 3, 5, 8, 2];
[ 1, 3, 5, 8, 2 ]
gap> InverterLista := function(lista)
>     return Reversed(lista);
> end;
function( lista ) ... end
gap>
gap> FiltrarParesInver := function(lista)
>     return Filtered(InverterLista(lista),x ->IsEvenInt(x));
> end;
function( lista ) ... end
gap> InverterLista(L);
[ 2, 8, 5, 3, 1 ]
gap> FiltrarParesInver(L);
[ 2, 8 ]
gap> FiltrarPares(L);
[ 8, 2 ]
```

Programando com GAP

Criando programas com ferramentas de recursividade

Existem três tipos de formas de fazer operações recursivas no GAP: `if...else`, `for...do` e `while...do`.

Além disso, existem três importantes operadores lógicos com os quais o GAP trabalha: `not`, `and` e `or`.

Programando com GAP

Criando programas com ferramentas de recursividade

Funções criadas com algoritmos que envolvem uma sequência de instruções com varias variáveis envolvidas, precisam de um certo cuidado.

Para o GAP cada variável que usamos está vinculada a algum valor. Quando as funções envolvem algoritmos com uma sequência de instruções com varias variáveis, é preciso dizer ao GAP quais variáveis serão usadas localmente na função que estamos criando.

```
func:= function(parâmetros)
    local (Variáveis a serem usadas);
       Sequência de instruções;
end;
```

Programando com GAP

Criando programas com ferramentas de recursividade

A ferramenta `for...do` funciona com a mesma lógica como em outras linguagens de programação.

```
for "Condição" do
   "Instrução";
od;
```

Programando com GAP

Criando programas com ferramentas de recursividade- Exemplos

Somar todos os números de 1 até n.

```
SomaNat := function(n)
    local s, i;
        s:=0;
        for i in [1..n] do
            s:=s+i;
        od;
    return s;
end;
```

Programando com GAP

Criando programas com ferramentas de recursividade- Exemplos

```
gap> SomaNat := function(n)
>     local s, i;
>         s:=0;
>         for i in [1..n] do
>             s:=s+i;
>         od;
>     return s;
> end;
function( n ) ... end
gap> SomaNat(1);
1
gap> SomaNat(10);
55
gap> SomaNat(25);
325
gap> SomaNat(100);
5050
```

Programando com GAP

Criando programas com ferramentas de recursividade- Exemplos

Fatorial de um número:

```
Fat:=function(n)
    local c,i;
    c:=1;
    for i in [n,n-1..1] do
        c:=c*i;
    od;
return c;
end;
```

GAP já possui uma função predefinida que calcula o fatorial de um número n, `Factorial(n)`.

Programando com GAP

Criando programas com ferramentas de recursividade- Exemplos

```
gap> Fat:=function(n)
>          local c,i;
>          c:=1;
>          for i in [n,n-1..1] do
>              c:=c*i;
>          od;
>      return c;
>      end;
function( n ) ... end
gap> Fat(3);
6
gap> Fat(5);
120
gap> Fat(10);
3628800
gap> Fat(0);
1
gap> Factorial(3);
6
gap> Factorial(5);
120
```

Programando com GAP

Criando programas com ferramentas de recursividade- Exemplos

Colocar antes de factorial

Função usando `Print` e `for`

```
Elev3:= function(n)
    local i;
    for i in [1..n] do
      Print( i, " elevado ao cubo é ", i^3, "\n");
    od;
end;
```

Programando com GAP

Criando programas com ferramentas de recursividade- Exemplos

Exercício!: Criar uma função que retorne uma lista dos fatoriais dos números desde 1 até n

```
gap> Elev3:= function(n)
>     local i;
>     for i in [1..n] do
>         Print( i, " elevado ao cubo é ", i^3, "\n");
>     od;
> end;
function( n ) ... end
gap> Elev3(4);
1 elevado ao cubo é 1
2 elevado ao cubo é 8
3 elevado ao cubo é 27
4 elevado ao cubo é 64
```

Programando com GAP

Criando programas com ferramentas de recursividade

Analogamente, a ferramenta `if...else` também funciona com a mesma lógica como em outras linguagens de programação.

```
if "Condição" then
"Instrução";
fi;
```

```
if "Condição1 then
    "Instrução 1";
else "Instrução 2";
fi;
```

```
if "Condição1" then
   "Instrução 1";
elif "Condição 2" then
    "Instrução 2";
fi;
```

Programando com GAP

Criando programas com ferramentas de recursividade-Exemplos

Decidir se um número é par ou ímpar:

```
EPar := function(n)
   if n mod 2 = 0 then
      return "Par";
   else
      return "Impar";
   fi;
end;
```

Programando com GAP

Criando programas com ferramentas de recursividade- Exemplos

```
gap> EPar := function(n)
>     if n mod 2 = 0 then
>         return "Par";
>     else
>         return "Impar";
>     fi;
> end;
function( n ) ... end
gap> EPar(2);
"Par"
gap> EPar(12);
"Par"
gap> EPar(121);
"Impar"
```

Programando com GAP

Criando programas com ferramentas de recursividade- Exemplos

$$f : n \longmapsto \begin{cases} n^3 & \text{se } n \equiv 0 \mod 3 \\ n^5 & \text{se } n \equiv 1 \mod 3 \\ 0 & \text{se } n \equiv 2 \mod 3 \end{cases}$$

Função por partes

```
fpart := 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;
```

Programando com GAP

Criando programas com ferramentas de recursividade- Exemplos

```
gap> fpart := 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> fpart(2);
0
gap> fpart(6);
216
gap> 6^3;
216
gap> fpart(4);
1024
gap> 4^5;
1024
gap>
```

Programando com GAP

Criando programas com ferramentas de recursividade- Exemplos

Elementos inversos em estruturas algébricas

```
InvS:= function(G)
    local x,y,e,L,elems;
      L:=[];
      e:= MultiplicativeNeutralElement(G);
      elms:=AsList(G);
    for x in elms do
        for y in elms do
            if x*y = e and y*x = e then
                Add(L, y);
            break;
            fi;
        od;
    od;
        return L;
end;
```

Programando com GAP

Criando programas com ferramentas de recursividade- Exemplos

Usando funções predefinidas que permitem a manipulação de listas podemos obter informações das estruturas que estamos analisando.

Elementos inversos em estruturas algébricas

```
S:=Estrtura algébrica;
e:=MultiplicativeNeutralElement( S );
elms:=AsList( S );
List( elms, x -> First( elms, y -> x*y = e and y*x = e ) );
```

- `AsList()` é uma função que transforma a estrutura algébrica com a qual estamos trabalhando em uma lista.
- `First()` é uma função que encontra o primeiro elemento da lista com a propriedade desejada.

Programando com GAP

Criando programas com ferramentas de recursividade- Exemplos

Usando funções predefinidas que permitem a manipulação de listas podemos obter informações das estruturas que estamos analisando.

Elementos inversos em estruturas algébricas

```
S:=Estrtura algébrica;
e:=MultiplicativeNeutralElement( S );
elms:=AsList( S );
List( elms, x -> First( elms, y -> x*y = e and y*x = e ) );
```

- `AsList()` é uma função que transforma a estrutura algébrica com a qual estamos trabalhando em uma lista.
- `First()` é uma função que encontra o primeiro elemento da lista com a propriedade desejada.

Programando com GAP

Criando programas com ferramentas de recursividade- Exemplos

```
gap> InvS:= function(G)
>     local x,y,e,L,elems;
>       L:=[];
>       e:= MultiplicativeNeutralElement(G);
>       elms:=AsList(G);
>     for x in elms do
>         for y in elms do
>             if x*y = e and y*x = e then
>                 Add(L, y);
>             break;
>             fi;
>         od;
>     od;
>         return L;
> end;
function( G ) ... end
gap> S:= Integers mod 5;
GF(5)
gap> InvS(S);
[ Z(5)^0, Z(5)^3, Z(5)^2, Z(5) ]
gap> e:=MultiplicativeNeutralElement( S );
Z(5)^0
gap> elms:=AsList( S );
[ 0*Z(5), Z(5)^0, Z(5), Z(5)^2, Z(5)^3 ]
gap> List( elms, x -> First( elms, y -> x*y = e and y*x = e ) );
[ fail, Z(5)^0, Z(5)^3, Z(5)^2, Z(5) ]
```

Programando com GAP

Criando programas com ferramentas de recursividade- Exemplos

```
gap> R:= Integers mod 6;
(Integers mod 6)
gap> InvS(R);
[ ZmodnZObj( 1, 6 ), ZmodnZObj( 5, 6 ) ]
gap> eR:=MultiplicativeNeutralElement( R );
ZmodnZObj( 1, 6 )
gap> elmsR:=AsList( R );
[ ZmodnZObj( 0, 6 ), ZmodnZObj( 1, 6 ), ZmodnZObj( 2, 6 ), ZmodnZObj( 3, 6 ), ZmodnZObj( 4
  ZmodnZObj( 5, 6 ) ]
gap> List( elmsR, x -> First( elmsR, y -> x*y = eR and y*x = eR ) );
[ fail, ZmodnZObj( 1, 6 ), fail, fail, fail, ZmodnZObj( 5, 6 ) ]
```

Programando com GAP

Criando programas com ferramentas de recursividade- Exemplos

Podemos criar funções com ferramentas de recursividade cujos algoritmos são mais avançados:

Sequência de Fibonacci:

```
fib := function ( n )
   if n = 1 or n = 2 then
       return 1;
   else
        return fib (n -1)+ fib(n -2);
    fi ;
end ;
```

Programando com GAP

Criando programas com ferramentas de recursividade- Exemplos

```
gap> fib := function ( n )
>                   if n = 1 or n = 2 then
>                       return 1;
>                   else
>                        return fib (n -1)+ fib(n -2);
>                    fi ;
>  end;
function( n ) ... end
gap> fib(1);
1
gap> fib(2);
1
gap> fib(3);
2
gap> fib(15);
610
```

Exercício: criar a função fatorial usando ferramenta de recursividade e a estrategia de chamar a própria função dentro do seu algoritmo como foi feito na função fib

Programando com GAP

Criando programas com ferramentas de recursividade- Exemplos

```
gap> fib := function ( n )
>                   if n = 1 or n = 2 then
>                       return 1;
>                   else
>                        return fib (n -1)+ fib(n -2);
>                    fi ;
>  end;
function( n ) ... end
gap> fib(1);
1
gap> fib(2);
1
gap> fib(3);
2
gap> fib(15);
610
```

Exercício: criar a função fatorial usando ferramenta de recursividade e a estrategia de chamar a própria função dentro do seu algoritmo como foi feito na função fib

Programação Básica com Estruturas Algébricas dentro do GAP!!!

Explorando estruturas algébricas

Grupos finitos

O GAP possui varias bibliotecas de dados de estruturas algébricas. A estrutura algébrica mais desenvolvida são os grupos. Em particular GAP possui uma livraria com 423.164.062 grupos finitos de ordem menor que 2000, com excepção da ordem 1024.

Além disso, GAP possui um vasto conjunto de funções que permitem fazer um analise minucioso dos grupos finitos.

Explorando estruturas algébricas

Grupos finitos- Funções predefinidas

O GAP bem equipado com uma serie de funções que permitem gerar grupos usando poucas informações tal como a ordem do grupo ou outra propriedade especifica do grupo.

- `TrivialGroup`: é uma função que retorna o grupo trivial. Esta função não precisa de informação alguma para GAP retornar o grupo trivial.

- `CycliGroup`: é uma função do GAP que gera um grupo cíclico dada o ordem do grupo desejado, GAP retornará um grupo cíclico dessa mesma ordem.

  Se desejamos saber se o grupo G gerado é de fato um grupo cíclico, basta usar a função `IsCyclic(G)`. Se o grupo for cíclico, GAP retorna `true`. Caso contrário, GAP retorna `false`.

Explorando estruturas algébricas

Grupos finitos- Funções predefinidas

O GAP bem equipado com uma serie de funções que permitem gerar grupos usando poucas informações tal como a ordem do grupo ou outra propriedade especifica do grupo.

- `TrivialGroup`: é uma função que retorna o grupo trivial. Esta função não precisa de informação alguma para GAP retornar o grupo trivial.

- `CycliGroup`: é uma função do GAP que gera um grupo cíclico dada o ordem do grupo desejado, GAP retornará um grupo cíclico dessa mesma ordem.

  Se desejamos saber se o grupo G gerado é de fato um grupo cíclico, basta usar a função `IsCyclic(G)`. Se o grupo for cíclico, GAP retorna `true`. Caso contrário, GAP retorna `false`.

Explorando estruturas algébricas

Grupos finitos- Funções predefinidas

O GAP bem equipado com uma serie de funções que permitem gerar grupos usando poucas informações tal como a ordem do grupo ou outra propriedade especifica do grupo.

- `TrivialGroup`: é uma função que retorna o grupo trivial. Esta função não precisa de informação alguma para GAP retornar o grupo trivial.

- `CycliGroup`: é uma função do GAP que gera um grupo cíclico dada o ordem do grupo desejado, GAP retornará um grupo cíclico dessa mesma ordem.

  Se desejamos saber se o grupo G gerado é de fato um grupo cíclico, basta usar a função `IsCyclic(G)`. Se o grupo for cíclico, GAP retorna `true`. Caso contrário, GAP retorna `false`.

Explorando estruturas algébricas

Grupos finitos- Funções predefinidas

- `DihedralGroup`: é uma função do GAP que gera um grupo diedral dada o ordem do grupo desejado. A função `DihedralGroup(n)` retorna um grupo diedral de ordem n.

  Se desejamos saber se o grupo G gerado é de fato um grupo cíclico, basta usar a função `IsDihedralGroup(G)`. Se o grupo for diedral, GAP retorna `true`. Caso contrário, GAP retorna `false`.

Explorando estruturas algébricas

Grupos finitos- Funções predefinidas

- `DihedralGroup`: é uma função do GAP que gera um grupo diedral dada o ordem do grupo desejado. A função `DihedralGroup(n)` retorna um grupo diedral de ordem n.

  Se desejamos saber se o grupo G gerado é de fato um grupo cíclico, basta usar a função `IsDihedralGroup(G)`. Se o grupo for diedral, GAP retorna `true`. Caso contrário, GAP retorna `false`.

Explorando estruturas algébricas

Grupos finitos- Funções predefinidas

- `GeneralLinearGroup`: é uma função do GAP que gera um grupo general linear de matrizes de uma certa ordem dada no corpo ou anel desejado. A função `GeneralLinearGroup(n,q)` ou `GL(n,q` retorna o grupo general linear das matrizes $n \times n$ sob o corpo de $\mathbb{F}_q$.

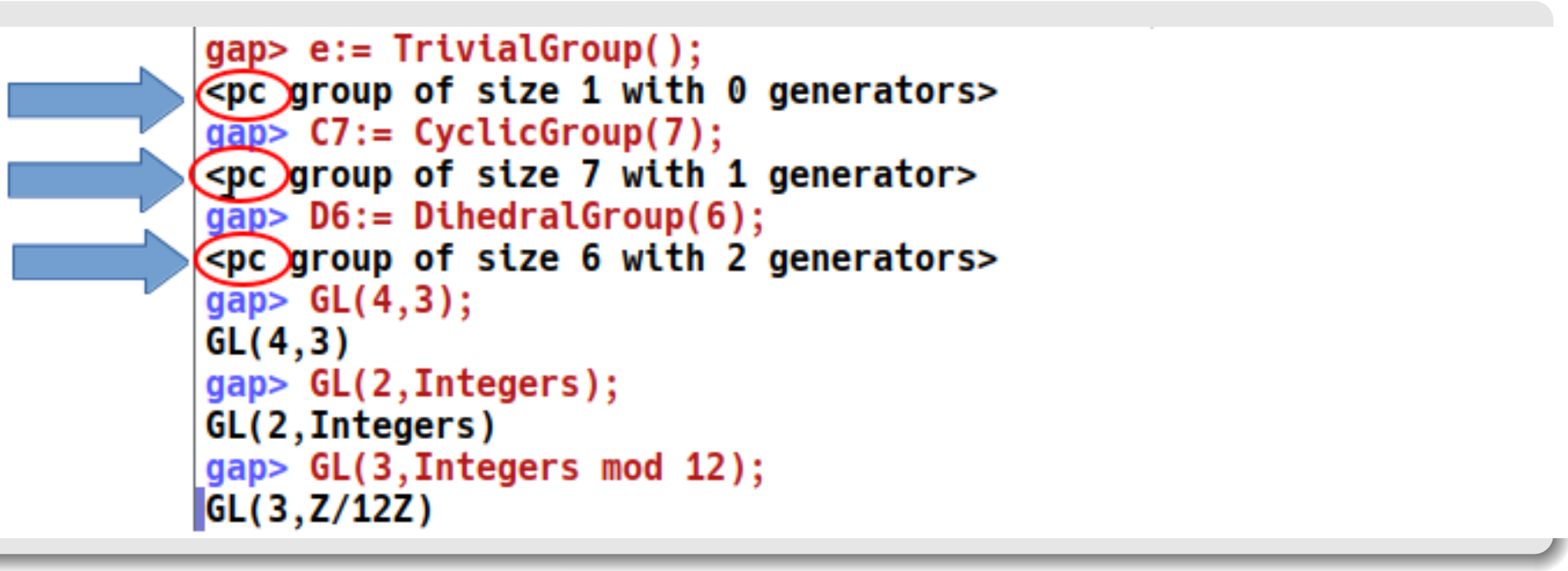
Explorando estruturas algébricas

Grupos finitos- Funções predefinidas

- `GeneralLinearGroup`: é uma função do GAP que gera um grupo general linear de matrizes de uma certa ordem dada no corpo ou anel desejado. A função `GeneralLinearGroup(n,q)` ou `GL(n,q` retorna o grupo general linear das matrizes $n \times n$ sob o corpo de $\mathbb{F}_q$.

Explorando estruturas algébricas

Grupos finitos



Ao usar as funções predefinidas para alguns grupos, GAP pode retornar uma descrição do grupo onde menciona o tipo de grupo e a ordem do mesmo e o número de geradores.

Explorando estruturas algébricas

Grupos finitos

Em algumas ocações o tipo de grupo pode ser **"pc group of..."**, isso significa que o grupo é um grupo policíclico (Um grupo *policíclico* é um grupo solúvel com a propriedade de que todo subgrupo é finitamente gerado).

Uma observação importante é que atualmente GAP somente gera estes grupos para corpos finitos, o anel de inteiros $\mathbb{Z}$ e anéis de inteiros modulares $\mathbb{Z}/n\mathbb{Z}$.

Explorando estruturas algébricas

Grupos finitos

Além disso, GAP possui uma série de funções que permitem gerar grupos dada a lista de geradores.

- `Group([`*gens*, *id*`])` é uma função que gera um grupo dada uma lista de geradores que contém o elemento identidade.
- `AsGroup([`*gens*`])` é uma função que gera um grupo de uma lista de objetos.
- `AllSmallGroups(`*n*`)` é uma função que retorna todos os grupos de ordem *n* que existem na livraria de GAP.
- `SmallGroup(`*n*, *m*`)` é uma função que retorna um grupo especifico que se encontra na livraria do GAP, em que `n` denota a ordem do grupo desejado e `m` é um identificador que o GAP usa para encontrar o grupo em questão dentro da livraria.

Explorando estruturas algébricas

Grupos finitos

- `StrctureDescription(G)` é uma função que retorna a escritura de um grupo G com um pequeno string de caracteres.

```
gap> G:=Group([(1,2,3,4),(1,2)]);
Group([ (1,2,3,4), (1,2) ])
gap> AsGroup([(1,2,3,4),(1,2)]);
fail
gap>
gap> G:=Group([(1,2)]);
Group([ (1,2) ])
gap> AsGroup([(),(1,2)]);
Group([ (1,2) ])
gap> AsGroup([(1,2)]);
fail
gap> StructureDescription(G);
"C2"
```

Explorando estruturas algébricas

Grupos finitos

GAP também possui uma série de funções predefinidas com características básicas dos grupos.

- `IsGroup(G)` é uma função booleana que retorna `true` se o objeto G é um grupo. Caso contrário retorna `false`.
- `Order(G)` é uma função que retorna a ordem do grupo G.
- `GeneratorsOfGroup(G)` é uma função que retorna uma lista dos geradores do grupo G.

Explorando estruturas algébricas

Grupos finitos

```
gap> ListGrupos:=AllSmallGroups(16);
[ <pc group of size 16 with 4 generators>, <pc group of size 16 with 4 generators>,
  <pc group of size 16 with 4 generators>, <pc group of size 16 with 4 generators>,
  <pc group of size 16 with 4 generators>, <pc group of size 16 with 4 generators>,
  <pc group of size 16 with 4 generators>, <pc group of size 16 with 4 generators>,
  <pc group of size 16 with 4 generators>, <pc group of size 16 with 4 generators>,
  <pc group of size 16 with 4 generators>, <pc group of size 16 with 4 generators>,
  <pc group of size 16 with 4 generators>, <pc group of size 16 with 4 generators> ]
gap> StructureDescription(ListGrupos[1]);
"C16"
gap> StructureDescription(ListGrupos[5]);
"C8 x C2"
gap> StructureDescription(ListGrupos[8]);
"QD16"
gap> SmallGroup(10,1);
<pc group of size 10 with 2 generators>
gap> StructureDescription(SmallGroup(10,1));
"D10"
gap> H:=SmallGroup(10,1);
<pc group of size 10 with 2 generators>
gap> Order(H);
10
gap> GeneratorsOfGroup(H);
[ f1, f2 ]
```

Explorando estruturas algébricas

Grupos finitos

Observação: Nesse link podem encontrar informações detalhadas relacionadas aos grupos que o GAP tem na livraria:

```
https://people.maths.bris.ac.uk/~matyd/GroupNames/index.html
```

Explorando estruturas algébricas

Grupos finitos- Programando

Vamos agora criar alguns programas para explorar as informações sobre grupos que o GAP oferece:

Elementos inversos em um grupo

```
G:=SmallGroup(n,m);
e:=MultiplicativeNeutralElement( G );
elms:=AsList( G );
List( elms, x -> First( elms, y -> x*y = e and y*x = e ) );
```

Lista de subgrupos de Sylows de um grupo

```
G:=SmallGroup(n,m);
SylowSubgroups := function ( G, p )
    return AsList(SylowSubgroup(G,p)^G);
end;
```

Explorando estruturas algébricas

Grupos finitos- Programando

Vamos agora criar alguns programas para explorar as informações sobre grupos que o GAP oferece:

Elementos inversos em um grupo

```
G:=SmallGroup(n,m);
e:=MultiplicativeNeutralElement( G );
elms:=AsList( G );
List( elms, x -> First( elms, y -> x*y = e and y*x = e ) );
```

Lista de subgrupos de Sylows de um grupo

```
G:=SmallGroup(n,m);
SylowSubgroups := function ( G, p )
    return AsList(SylowSubgroup(G,p)^G);
end;
```

Explorando estruturas algébricas

Grupos finitos- Programando

Função que procura na biblioteca de grupo do GAP aqueles grupos que sejam nilpotentes, no abelianos e com índice de nilpotência maior que 2.

```
EleGs:=function(n)
   local g,PNnA,GNnA,g1;
    g:=AllSmallGroups(n);
    PNnA:=PositionsProperty( g, x-> IsNilpotent(x) and not IsAbelian(x)
                        and NilpotencyClassOfGroup(x)>2);
    GNnA:=g{PNnA};
    g1:=List(GNnA,StructureDescription);
           return GNnA;
end;
```

Explorando estruturas algébricas

Grupos finitos- Programando

Função que mostra a lista de subgrupos normais de um grupo G.

```
NormalSubgrpList:= function(G)
    local L, NList, cc, i, N;
L:=[];
cc:=ConjugacyClassesSubgroups(G);
NList:=List(cc,x->Representative(x));
    for i in [1..Length(NList)] do
        N:=NList[i];
        if IsNormal(G,N) then
        Add(L,N);
        fi;
    od;
return L;
end;
```

Exercício: Crie uma função que procure os subgrupos normais de um grupo G que sejam também p-subgrupos de Sylow de G para algum primo p

Explorando estruturas algébricas

Grupos finitos- Programando

Função que mostra a lista de subgrupos normais de um grupo G.

```
NormalSubgrpList:= function(G)
    local L, NList, cc, i, N;
L:=[];
cc:=ConjugacyClassesSubgroups(G);
NList:=List(cc,x->Representative(x));
    for i in [1..Length(NList)] do
        N:=NList[i];
        if IsNormal(G,N) then
        Add(L,N);
        fi;
    od;
return L;
end;
```

Exercício: Crie uma função que procure os subgrupos normais de um grupo G que sejam também p-subgrupos de Sylow de G para algum primo p

Explorando estruturas algébricas

Referências

- D. J. de Souza, *ÁLGEBRA COM ENFOQUE COMPUTACIONAL - O SISTEMA GAP*, Trabalho de Conclusão de Curso, Universidade Federal de Lavras. 2018.
- The GAP Group, *GAP – Groups, Algorithms, and Programming, Version 4.13.1*; 2024, `https://www.gap-system.org`.
- L. Vendramin, *Mini-course on GAP- Lecture 1*, Dalhousie University, Halifax. 2020.
- Material didático para aprender GAP `https://www.gap-system.org/doc/teach/`