Introduzione a GAP

Ilaria Colazzo - Prof. Francesco Catino

Università del Salento

3 Maggio 2017

SOMMARIO

Windows

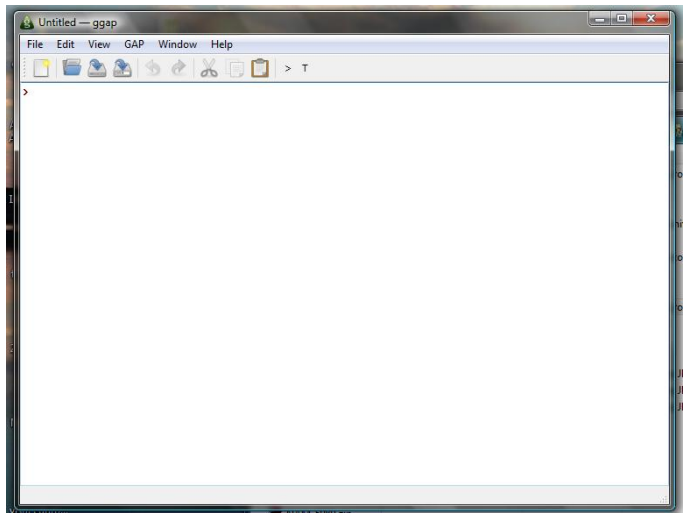
Per installare GAP, andare all'indirizzo `http://www.math.colostate.edu/~hulpke/CGT/education.html` scaricare il file `GAP4412Setup.exe`. Una volta scaricato il file aprirlo e iniziare l'installazione cliccando l'immagine riportata accanto. Verrà installato sia il programma GAP che GGAP un interfaccia grafica che semplifica l'utilizzo del programma.

Macintosh

Per installare GAP, andare all'indirizzo `http://www.math.colostate.edu/~hulpke/CGT/education.html` scaricare il file `GAP4.4.12a.mpkg.zip`. Una volta scaricato il file aprirlo e iniziare l'installazione cliccando l'immagine riportata accanto. Verrà installato sia il programma GAP che GGAP un interfaccia grafica che semplifica l'utilizzo del programma.

L'interfaccia utente GGAP

Aperto GGAP appare il foglio di lavoro. Nella parte alta della finestra troviamo la barra degli strumenti che permette di fare le operazioni sul foglio di lavoro.
Accanto al simbolo > andranno inseriti da tastiera i vari comandi.
ATTENZIONE: Ogni comando in GAP termina con il simbolo ;



Workspace e Worksheet (I)

L'interfaccia di GGAP offre due modi di salvare (e caricare) i fogli di lavoro: **Workspace** (opzione di default) e **Worksheet**. Per salvare (caricare, risp.) andare nel menu `File` (in alto a sinistra) e selezionare `Save` (`Open`, risp.)

Workspace (estensione `.gwp`): salva l'intera sessione di lavoro che comprende sia ciò che è visualizzato sullo schermo che tutte le variabili definite dall'utente.

PRO aprendo un file di questo tipo è subito possibile riprendere il lavoro nel punto in cui ci si è interrotti.

CONTRO bassa compatibilità con differenti computer o con differenti versioni di GAP, file solitamente molto pesanti.

Workspace e Worksheet (II)

Worksheet (estensione `.gws`): salva solo il testo (sia gli input che i rispettivi output) mostrato sullo schermo.

PRO alta compatibilità con differenti computer o con differenti versioni di GAP, file di piccole dimensioni.

CONTRO gli oggetti definiti non sono salvati, una volta riaperto il file è necessario eseguire nuovamente tutti i comandi.

Confronti

Il simbolo di uguaglianza, $=$, verifica se due espressioni sono uguali o no. Ad esempio:

```
> 8 = 9;
 false
> 1^3+12^3=9^3+10^3;
true
```

Esercizio

- Si calcoli 3^{121};
- Si determini se $2^{25} + (45 * 51)$ è più grande di 34 milioni.

Confronti

Il simbolo di uguaglianza, $=$, verifica se due espressioni sono uguali o no. Ad esempio:

```
> 8 = 9;
 false
> 1^3+12^3=9^3+10^3;
true
```

Esercizio

- Si calcoli 3^{121};
- Si determini se $2^{25} + (45 * 51)$ è più grande di 34 milioni.

Variabili

Un oggetto può essere memorizzato in una variabile. Per le variabili può essere usato un qualsiasi nome che non sia una parola riservata.
L'operatore di assegnamento è un due punti seguito dal simbolo uguale, :=. Una volta che un dato è stato registrato all'interno di una variabile questa può essere usata in sua vece. Ad esempio:

```
> variabile1:=126;
  126
> variabile1^2;
  15876
```

Il simbolo = è l'operatore di confronto. Per assegnare un valore ad una variabile usare il simbolo :=.

```
> variabile2=14;
Variable: 'variabile2' must have a value
```

Si osservi che nel linguaggio GAP non è necessario dichiarare il tipo di variabili.

Funzioni (I)

Una funzione prende in input uno o più argomenti e restituisce un output. GAP ha molte funzioni già implementate e pronte per essere utilizzate (alcune le vedremo nel seguito).
In alcuni casi definire nuove funzioni è utile.
Una funzione può essere definita usando la parola `function` seguita dagli argomenti e finisce con la parola `end`. `return` restituisce l'output.
Ad esempio, scriviamo una funzione che prende in input un intero e gli aggiunge 2:

```
>Piu2:=function(x) return x+2; end;
 function( x ) ... end
>Piu2(15);
 17
```

Funzioni (II)

Nel caso di funzioni che in input prendano un solo argomento si può utilizzare la scrittura più rapida in cui si scrive l'input seguito da una freccia, $->$, seguita dall'output.

```
>Piu2:=(x -> x+2);
 function( x ) ... end
>Piu2(15);
 17
```

Esercizio

Si definisca la funzione `SommaDeiPriminInteri` che prenda in input un intero n e restituisca la somma $1+2+3+\cdots+n$. Si utilizzi la funzione per calcolare questa somma per $n = 100$ e $n = 987$.

Funzioni (II)

Nel caso di funzioni che in input prendano un solo argomento si può utilizzare la scrittura più rapida in cui si scrive l'input seguito da una freccia, $->$, seguita dall'output.

```
>Piu2:=(x -> x+2);
 function( x ) ... end
>Piu2(15);
 17
```

Esercizio

Si definisca la funzione `SommaDeiPriminInteri` che prenda in input un intero n e restituisca la somma $1+2+3+\cdots+n$. Si utilizzi la funzione per calcolare questa somma per $n=100$ e $n=987$.

Liste (I)

Una lista inizia con una parentesi quadra aperta, [, e termina con una parentesi quadra chiusa,] gli elementi sono separati da una virgola.
Il comando `Length` restituisce il numero di elementi nella lista.
Per accedere all'i-esimo elemento della lista `L` si usa il comando `L[`i`]`, la stessa notazione può essere utilizzata per assegnare un valore alla lista.
L'appartenenza alla lista può essere verificata utilizzando `x in L`, il comando `Position(L,x)` restituisce la posizione della prima occorrenza di `x` nella lista.
Il comando `Add(L,x)` aggiunge in coda l'oggetto `x` alla lista `L`, `Append(L,L2)` aggiunge tutti gli elementi della lista `L2`.

Liste (II)

```
>L:=[2,3,5,7,9,11,1];
 [ 2, 3, 5, 7, 9, 11, 1 ]
>Length(L);
 7
>L[3];
 5
>L[3]:=1;
 1
>L;
 [ 2, 3, 1, 7, 9, 11, 1 ]
>4 in L;
 false
>1 in L;
 true
>Position(L,1);
 3
```

Liste (III)

Bisogna prestare attenzione che assegnando una lista a due variabili non si fa una copia della lista, così cambiando l'una si cambia anche l'altra.
Per fare una copia di una lista si può usare il comando `ShallowCopy`.

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

Operazioni su Liste

`List(L,`*`funzione`*`)` applica *funzione* a tutti gli elementi della lista `L` e dà la lista dei risultati.
`Filtred(L,`*`funzione`*`)` restituisce una lista di quegli elementi di `L` per i quali la *funzione* è vera.
`Number(L,`*`funzione`*`)` restituisce il numero degli elementi di `L` per i quali *funzione* è vera.

```
>List([1..12],x->x^2);
 [ 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144 ]
>Filtered([1..20],IsPrime);
 [ 2, 3, 5, 7, 11, 13, 17, 19 ]
>Number([1..20],IsPrime);
 8
```

Vettori e Matrici (I)

In GAP i vettori sono semplicemente delle liste e le matrici sono liste di vettori riga. Ad esempio:

```
>vec:=[-1,2,1];
 [ -1, 2, 1 ]
>M:=[[1,2,3],[4,5,6],[7,8,9]];
 [ [ 1, 2, 3 ], [ 4, 5, 6 ], [ 7, 8, 9 ] ]
>vec*M; #vec è visto come vettore riga
 [ 14, 16, 18 ]
>M*vec; #vec è visto come vettore colonna
 [ 6, 12, 18 ]
>vec*vec;
 6
>Display(M); #per visualizzare meglio la matrice
 [ [  1,  2,  3 ],
   [  4,  5,  6 ],
   [  7,  8,  9 ] ]
```

Vettori e Matrici (II)

Le matrici e i vettori spesso non sono modificabili. I comandi `ShallowCopy(`*`vettore`*`)` o `MutableCopyMat(`*`matrice`*`)` permettono di ottenere copie degli oggetti che siano modificabili.

Insiemi

Gli insiemi sono implementati come liste ordinate. Data una lista `L` il comando `Set(L)` restituisce una copia della lista ordinata. Ad esempio:

```
>L:=Set([5,-3,7,2]);
 [ -3, 2, 5, 7 ]
```

Il comando `AddSet(L,x)` inserisce l'elemento `x` in `L` nella posizione appropriata.

```
>AddSet(L,4);L;
 [ -3, 2, 4, 5, 7 ]
```

I comandi `Union(L,L2)`, `Intersection(L,L2)` e `Difference(L,L2)` permettono di implementare l'unione, l'intersezione e la differenza di insiemi.

Programmazione di base (I)

Il *loop* di base utilizzato in GAP è il ciclo `for`. Solitamente il `for` ripete alcune istruzioni per una variabile intera che scorre all'interno di una lista. Le istruzioni iniziano con la parola `do` e terminano con la parola `od`. Il controllo condizionale inizia con `if` seguito da una variabile booleana (che può assumere i valori `true` o `false`), dalla parola `then`, da un insieme di istruzioni che verranno eseguite se la condizione è vera e termina con la parola `fi`. Condizioni addizionali possono essere date con il comando `elseif` (se si vogliono inserire condizioni aggiuntive) o con `else` (se non si vogliono condizioni aggiuntive).

Programmazione di base (II)

Ad esempio costruiamo un programma che stampi i primi 40 numeri primi.

```
>for i in [1..40] do if IsPrime(i) then Print(i, "\n"); fi; od;
 2
 3
 5
 7
 11
 13
 17
 19
 23
 29
 31
 37
```