

Struttura di Mathematica

Introduzione

Mathematica è costituito da due componenti: il Kernel e il Front End. Il Kernel (nucleo) è il motore di calcolo del programma, mentre il Front End è l'interfaccia utente, naturalmente grafica. Durante una sessione di lavoro l'utente interagisce con il Kernel attraverso il Front End. A loro volta, Kernel e Front End scambiano dati mediante un protocollo di comunicazione denominato MathLink. *Mathematica* dà la possibilità di chiudere il Kernel separatamente, in modo da riprendere un nuovo ciclo di calcolo durante la medesima sessione.

Per default i file di *Mathematica* vengono salvati come Notebook aventi estensione .nb.

L'interazione tra il Kernel e il Front End avviene tramite le cosiddette "celle" di input. Al termine della computazione, il Kernel restituisce il risultato attraverso celle di output.

Per quanto detto, il Kernel può essere disattivato durante la sessione di lavoro direttamente dalla Toolbar (Evaluation->Quit Kernel). Si osservi che la chiusura del Kernel causa la perdita di tutti i dati presenti nel notebook. Al contrario, se si vuole interrompere una sola operazione, si può interrompere il processo, "abortendolo". Per fare ciò, basta andare sul pulsante "Evaluation" sulla Toolbar e selezionare la voce "AbortEvaluation". Si suggerisce la chiusura del Kernel quando c'è il rischio di un crash di sistema dovuti a cicli infiniti.

Per quanto riguarda l'aspetto grafico del Front End, osserviamo che esso può essere personalizzato attraverso i "fogli di stile". Il percorso è: Format->StyleSheet. Qui è possibile scegliere tra vari Templates.

Sintassi

Mathematica è case-sensitive, cioè distingue le maiuscole dalle minuscole. Ad esempio, i nomi delle funzioni built-in sono maiuscoli, come pure i comandi. In generale, le istruzioni hanno la seguente sintassi: `Command[expr1, expr2, ..., exprn]`

Qui `Command` è un comando generico, mentre `expr1,...` sono espressioni che verranno valutate. È buona norma scrivere le funzioni definite dall'utente con lettere minuscole, in modo da distinguerle dalle funzioni built-in. Per quanto riguarda le

funzioni, gli argomenti sono racchiuse tra parentesi quadre, anziché tonde. Per quanto detto, un notebook è un set di celle di input e output. Per non mostrare l'output si utilizza il terminatore `;`

Se in una data fase di sessione caratterizzata dalla k -esima cella di output, si desidera avere in output il contenuto della cella n -esima con $n < k$, dobbiamo utilizzare il comando `Out[n]`, che restituisce l' n -esima cella. Alternativamente si può utilizzare la forma `%%...%`

Esempio:

```
a = 1.2;
```

```
b = 0.8;
```

```
%%
```

```
1.2
```

La sintassi per le righe di commento è: `(* commento *)`

Operatori aritmetici

Le operazioni aritmetiche ordinarie sono `+`, `-`, `*`, `/`, `^`. Il simbolo `*` utilizzato nella moltiplicazione può essere omissso a patto di lasciare uno spazio. Ad esempio:

```
ab
```

```
ab
```

è equivalente a

```
a * b
```

```
a b
```

In alcuni casi particolari, lo spazio non è necessario. Ad esempio:

```
4 Cosh[x]
```

```
4 Cosh[x]
```

è equivalente a:

```
4 * Cosh[x]
```

```
4 Cosh[x]
```

però $xy \neq x*y$

Quindi è preferibile lasciare uno spazio o utilizzare il simbolo `*`.

Costanti built-in

Le principali costanti built-in sono:

```
Infinity
```

```
∞
```

```
(*digitando escape inf escape*)
```

```
∞
```

```
∞
```

```
Pi
```

```
π
```

```
(*digitando escape pi escape*)
```

```
π
```

```
π
```

Base dei logaritmi naturali:

```
E
```

```
e
```

Unità immaginaria:

```
I
```

```
i
```

Per passare dal valore simbolico al valore numerico è necessario utilizzare la funzione `N[]`. Ad esempio

```
N[π]
```

```
3.14159
```

Può anche essere utilizzata la forma abbreviata

```
 $\pi$  // N  
3.14159
```

Operatori Booleani

Gli operatori booleani (connettivi logici) sono: `&&` (**And**), `||` (**Or**), `!` (**Not**), `Xor`

Funzioni Built-in

Le principali funzioni built-in sono:

Funzione logaritmo in base e

```
Log[x] ;
```

Funzione $\log_a x$

```
Log[a, x] ;
```

La funzione valore assoluto

```
Abs[x] ;
```

Le usuali funzioni trigonometriche:

```
Sin[x] ;
```

```
Cos[x] ;
```

```
Tan[x] ;
```

```
ArcCos[x] ;
```

```
ArcSin[x] ;
```

```
ArcTan[x] ;
```

Funzioni definite dall'utente

Assegnazione immediata

Esistono diversi modi per definire una funzione. L'approccio più immediato ha il seguente costrutto: $f[x_]=expr[x]$

Analizziamo questo codice. L' "underscore" (cioè il trattino "_") è il cosiddetto pattern ("modello"). Il pattern comunica al Kernel che la variabile x rappresenta "qualunque cosa". Il secondo membro è invece l'espressione analitica della funzione f .

Ad esempio, se scriviamo:

```
f[x] = x ^ 2
x^2
```

e proviamo a calcolare il valore assunto dalla funzione $f(x) = x^2$ nel punto $x = 2$, *Mathematica* restituisce $f(2)$.

```
f[2]
f[2]
```

Invece, se usiamo la sintassi con l'underscore:

```
f[x_] = x ^ 2
x^2

f[2]
4
```

cioè il risultato corretto. Il costrutto: $f[x_]=expr[x]$ definisce ciò che si chiama *assegnazione immediata* della funzione $f(x)$.

Assegnazione ritardata

Nell'assegnazione ritardata la funzione viene valutata solo quando viene richiamata dall'utente. La sintassi è: $f[x_] := expr[x]$

In output non viene restituito nulla. Per visualizzare l'espressione della funzione, dobbiamo richiamare $f[x]$; l'output sarà $expr[x]$

Per comprendere la differenza tra i due tipi di assegnazione - immediata e ritardata - consideriamo la funzione:

```
f[x_] := Simplify[x^2 + 1 - 3 + x + 2 * x]
```

essendo `Simplify[]` un comando built-in utilizzato per semplificare algebricamente un' espressione. Utilizziamo il comando `?` per chiedere a *Mathematica* cos' è la funzione `f[x]` appena definita. Abbiamo :

```
? f
```

```
Global`f
```

```
f[x_] := Simplify[x^2 + 1 - 3 + x + 2 x]
```

Adesso definiamo:

```
g[x_] = Simplify[x^2 + 1 - 3 + x + 2 x]
-2 + 3 x + x^2
```

che restituisce il risultato dell' applicazione di `Simplify`

Un modo veloce per visualizzare il valore assunto da una funzione f in un punto, consiste nell' usare il comando `//f`. Consideriamo ad esempio, la funzione

```
f[x_] := Cosh[x^2 + 1]
```

Se scriviamo:

```
2 // f
Cosh[5]
```

Volendo il valore numerico:

```
N[f[2]]
74.2099
```

Ma `//Command` agisce anche sulle funzioni built-in, quindi:

```
f[2] // N
74.2099
```

Evidentemente lo stesso risultato può essere raggiunto più velocemente scrivendo:

```
2 // f // N
74.2099
```

Osservazione. Durante una sessione di lavoro con *Mathematica*, il Kernel enumera progressivamente i comandi inseriti, nonché le funzioni definite dall' utente. Gli output corrispondenti vengono poi caricati dal Kernel, anche se una cella di output viene cancellata manualmente. Da ciò segue che se definiamo una funzione f , il Kernel non perde memoria di tale oggetto, quindi volendo definire una nuova funzione, sarà necessario un nuovo simbolo. Per evitare conflitti tra simboli o una

proliferazione degli stessi è conveniente rimuovere dal Kernel tutte quelle informazioni divenute inutili a quel punto della sessione. Una funzione built-in adatta allo scopo è **clear**, la cui sintassi è:

```
Clear[symbol1, symbol2, ...]
```

Dove i vari **symbol** sono oggetti definiti dall'utente. Nell'esempio precedente, volendo definire una nuova funzione con lo stesso simbolo, scriviamo:

```
Clear[f]
```

Un altro comando utilizzato per rimuovere informazione è **Remove**, che è più potente di **clear**, in quanto rimuove completamente un simbolo da un'interazione.

Ad esempio, se con l'input precedente scriviamo:

```
? f
```

```
Global`f
```

Adesso utilizziamo **Remove**:

```
Remove[f]
```

```
? f
```

```
Information::notfound: Symbol f not found. >>
```