Funzioni e expl3

Enrico Gregorio

31 ottobre 2020

GuITmeeting 2020

Funzioni

Una funzione (matematica) è una regola che a un *input* associa un ben determinato *output*

Funzioni

Una funzione (matematica) è una regola che a un *input* associa un ben determinato *output*

Molti linguaggi di programmazione adoperano quest'idea

Funzioni

Una funzione (matematica) è una regola che a un *input* associa un ben determinato *output*

Molti linguaggi di programmazione adoperano quest'idea

Anche **expl3**

Funzioni

Il concetto non è difficile da digerire:

```
\newcommand{\title}[1]{\gdef\@title{#1}}
```

Funzioni

Il concetto non è difficile da digerire:

```
\newcommand{\title}[1]{\gdef\@title{#1}}
```

Che differenza c'è fra `\title` e `\@title` a parte @?

Funzioni

Il concetto non è difficile da digerire:

```
\newcommand{\title}[1]{\gdef\@title{#1}}
```

Che differenza c'è fra `\title` e `\@title` a parte @?

Ruoli! La macro `\title` esegue un'azione

Funzioni

Il concetto non è difficile da digerire:

```
\newcommand{\title}[1]{\gdef\@title{#1}}
```

Che differenza c'è fra `\title` e `\@title` a parte @?

Ruoli! La macro `\title` esegue un'azione

La macro `\@title` è un semplice contenitore

Funzioni e variabili

In **expl3** si distingue tra *funzioni* e *variabili*

Funzioni e variabili

In **expl3** si distingue tra *funzioni* e *variabili*

Nell'esempio precedente `\title` è una funzione e `\@title` è una variabile

Funzioni e variabili

In **expl3** si distingue tra *funzioni* e *variabili*

Nell'esempio precedente `\title` è una funzione e `\@title` è una variabile

Una funzione non produce necessariamente un *output* "tangibile": ciò che la distingue è *eseguire qualche azione*

Funzioni e variabili

In **expl3** si distingue tra *funzioni* e *variabili*

Nell'esempio precedente `\title` è una funzione e `\@title` è una variabile

Una funzione non produce necessariamente un *output* "tangibile": ciò che la distingue è *eseguire qualche azione*

L'azione di `\title` è di impostare il valore della variabile `\@title`

Nomi

In LaTeX tradizionale è difficile distinguere tra funzioni e variabili solo guardandone il “nome”

Nomi

In LaTeX tradizionale è difficile distinguere tra funzioni e variabili solo guardandone il “nome”

In **expl3** invece è facilissimo, se si seguono le convenzioni raccomandate

Nomi

In LaTeX tradizionale è difficile distinguere tra funzioni e variabili solo guardandone il “nome”

In **expl3** invece è facilissimo, se si seguono le convenzioni raccomandate

- Variabili

  `\l_`⟨*prefisso*⟩`_`⟨*nome proprio*⟩`_`⟨*tipo*⟩
  `\g_`⟨*prefisso*⟩`_`⟨*nome proprio*⟩`_`⟨*tipo*⟩
  `\c_`⟨*prefisso*⟩`_`⟨*nome proprio*⟩`_`⟨*tipo*⟩

- Funzioni

  `\`⟨*prefisso*⟩`_`⟨*nome proprio*⟩`:`⟨*segnatura*⟩

Nomi

Nel caso del titolo avremo qualcosa come

```
\NewDocumentCommand{\title}{m}
 {
  \example_title:n { #1 }
 }

\tl_new:N \g_example_title_tl

\cs_new_protected:Nn \example_title:n
 {
  \tl_gset:Nn \g_example_title_tl { #1 }
 }
```

Nomi

⟨*prefisso*⟩ è una successione di caratteri alfabetici che dovrebbe caratterizzare univocamente il pacchetto o il codice personale

⟨*nome proprio*⟩ è una successione di caratteri alfabetici e *underscore* che identifichi la funzione o la variabile

⟨*tipo*⟩ (per le variabili) è il tipo di variabile (nell'esempio è `tl` per *token list*)

Per le variabili, `g`, `l` e `c` indicano che si tratta di una variabile globale, locale o costante.

Nomi


\g_example_title_tl
globale
prefisso
nome proprio
tipo


\example_title:n
prefisso
nome proprio
segnatura

Segnatura

Per le funzioni, invece di un tipo generico, c'è una *segnatura* che è divisa dal resto con un carattere "due punti"

Segnatura

Per le funzioni, invece di un tipo generico, c'è una *segnatura* che è divisa dal resto con un carattere "due punti"

È una stringa di caratteri tra

N n e f x T F c V v o w

Segnatura

Per le funzioni, invece di un tipo generico, c'è una *segnatura* che è divisa dal resto con un carattere "due punti"

È una stringa di caratteri tra

```
N n   e f x   T F   c   V v   o   w
```

Ciascun carattere (eccetto `w`) indica un argomento della funzione

Segnatura

Alcune funzioni predefinite nel nucleo di **expl3**

`\scan_stop:`	nessun argomento
`\tl_new:N`	un argomento
`\use_none:n`	un argomento
`\seq_item:Nn`	due argomenti
`\seq_case:nnTF`	quattro argomenti

Che differenza c'è fra `N` e `n`? Ottima domanda

Segnatura

- `N` indica che l'argomento è un singolo token
- `n` indica che l'argomento è una lista di token tra graffe

Segnatura

- `N` indica che l'argomento è un singolo token
- `n` indica che l'argomento è una lista di token tra graffe

```
\seq_item:Nn \l_example_pippo_seq { 640 }
```

Segnatura

- `N` indica che l'argomento è un singolo token
- `n` indica che l'argomento è una lista di token tra graffe

```
\seq_item:Nn \l_example_pippo_seq { 640 }
```

Attenzione

Secondo le regole sintattiche di TEX, una chiamata

```
\seq_item:Nn { \l_example_pippo_seq } 1
```

sarebbe equivalente a

```
\seq_item:Nn \l_example_pippo_seq { 1 }
```

ma seguire le convenzioni è meglio: *lo giuro sul mio onore*

Definire funzioni

```
\cs_new:Nn \example_inciso:n { -- #1 -- }

\cs_new_protected:Nn \example_data:nnn
 {
  \tl_set:Nn \l_example_data_iso_tl { #3 - #2 - #1 }
 }
```

Definire funzioni

```
\cs_new:Nn \example_inciso:n { -- #1 -- }

\cs_new_protected:Nn \example_data:nnn
 {
  \tl_set:Nn \l_example_data_iso_tl { #3 - #2 - #1 }
 }
```

Qual è la differenza? Una funzione deve essere `protected` se esegue azioni "non espandibili" come impostare valori di variabili e, quasi sempre, quando richiama altre funzioni `protected` (ma ci sono eccezioni)

Definire funzioni

```
\cs_new:Nn \example_inciso:n { -- #1 -- }

\cs_new_protected:Nn \example_data:nnn
 {
  \tl_set:Nn \l_example_data_iso_tl { #3 - #2 - #1 }
 }
```

Qual è la differenza? Una funzione deve essere `protected` se esegue azioni "non espandibili" come impostare valori di variabili e, quasi sempre, quando richiama altre funzioni `protected` (ma ci sono eccezioni)

Nel dubbio si usi `protected`

Definire funzioni

Nella segnatura della funzione da definire possono comparire solo caratteri `N` e `n`

Definire funzioni

Nella segnatura della funzione da definire possono comparire solo caratteri `N` e `n`

E gli altri?

Definire funzioni

Nella segnatura della funzione da definire possono comparire solo caratteri `N` e `n`

E gli altri?

È ora di *varianti*

Varianti

Supponiamo di voler definire un’interfaccia per mantenere variabili dove l’utente registra qualcosa da impiegare in seguito.

Varianti

Supponiamo di voler definire un'interfaccia per mantenere variabili dove l'utente registra qualcosa da impiegare in seguito.

Modo classico: `\newcommand{\pippo}{Pluto}` e poi si adopera `\pippo` quando serve

Varianti

Supponiamo di voler definire un'interfaccia per mantenere variabili dove l'utente registra qualcosa da impiegare in seguito.

Modo classico: `\newcommand{\pippo}{Pluto}` e poi si adopera `\pippo` quando serve

Ma qui vogliamo usare **expl3**

Varianti

Supponiamo di voler definire un'interfaccia per mantenere variabili dove l'utente registra qualcosa da impiegare in seguito.

Modo classico: `\newcommand{\pippo}{Pluto}` e poi si adopera `\pippo` quando serve

Ma qui vogliamo usare **expl3**

Per allocare una variabile *tl* e impostarla

```
\tl_new:N \l_example_pippo_tl
\tl_set:Nn \l_example_pippo_tl { Pluto }
```

Varianti

Supponiamo di voler definire un'interfaccia per mantenere variabili dove l'utente registra qualcosa da impiegare in seguito.

Modo classico: `\newcommand{\pippo}{Pluto}` e poi si adopera `\pippo` quando serve

Ma qui vogliamo usare **expl3**

Per allocare una variabile *tl* e impostarla

```
\tl_new:N \l_example_pippo_tl
\tl_set:Nn \l_example_pippo_tl { Pluto }
```

Qual è il problema? Che a livello utente questa sintassi non è permessa e saremmo cattivi programmatori se la imponessimo comunque

Varianti

L’interfaccia per l’utente prevederà due comandi

```
\setvar{pippo}{Pluto}
\usevar{pippo}
```

(che risolve anche il problema degli spazi dopo i comandi)

Varianti

L'interfaccia per l'utente prevederà due comandi

```
\setvar{pippo}{Pluto}
\usevar{pippo}
```

(che risolve anche il problema degli spazi dopo i comandi)

```
\NewDocumentCommand{\setvar}{mm}
 {
  \example_setvar:cn { l_example_var_#1_tl } { #2 }
 }
\NewExpandableDocumentCommand{\usevar}{m}
 {
  \example_usevar:c { l_example_var_#1_tl }
 }
```

Varianti

Ora dobbiamo *programmare* in **expl3**

Varianti

Ora dobbiamo *programmare* in **expl3**

```
\cs_new_protected:Nn \example_setvar:Nn
 {
  \tl_clear_new:N #1
  \tl_set:Nn #1 { #2 }
 }
\cs_new:Nn \example_usevar:n
 {
  \tl_use:N #1
 }
```

Varianti

Qui viene il bello! Ora dobbiamo passare un nome costruito con quello che l'utente decide.

```
\cs_generate_variant:Nn \example_setvar:Nn { cn }
\cs_generate_variant:Nn \example_usevar:N { c }
```

Varianti

Qui viene il bello! Ora dobbiamo passare un nome costruito con quello che l'utente decide.

```
\cs_generate_variant:Nn \example_setvar:Nn { cn }
\cs_generate_variant:Nn \example_usevar:N { c }
```

Ora, quando l'utente dice `\setvar{pippo}{Pluto}` viene eseguito

```
\example_setvar:cn { l_example_var_pippo_tl } { Pluto }
```

e la variante c dice a TEX che prima di eseguire la funzione il primo argomento va trasformato in un singolo token, quindi avremo

```
\example_setvar:Nn \l_example_var_pippo_tl { Pluto }
```

Varianti

Una delle varianti più utili è V

Varianti

Una delle varianti più utili è V

Supponiamo di avere una funzione `\example_foo:n` che fa qualcosa con il suo argomento

Varianti

Una delle varianti più utili è V

Supponiamo di avere una funzione `\example_foo:n` che fa qualcosa con il suo argomento

Supponiamo ora che in alcune situazioni ci serva fornire come argomento il contenuto di una *tl*, oppure di una *fp*

Varianti

Una delle varianti più utili è V

Supponiamo di avere una funzione `\example_foo:n` che fa qualcosa con il suo argomento

Supponiamo ora che in alcune situazioni ci serva fornire come argomento il contenuto di una *tl*, oppure di una *fp*

Niente di più facile e non dobbiamo sapere nulla di come sono implementati i diversi tipi di variabili

Varianti

Una delle varianti più utili è V

Supponiamo di avere una funzione `\example_foo:n` che fa qualcosa con il suo argomento

Supponiamo ora che in alcune situazioni ci serva fornire come argomento il contenuto di una *tl*, oppure di una *fp*

Niente di più facile e non dobbiamo sapere nulla di come sono implementati i diversi tipi di variabili

```
\cs_generate_variant:Nn \example_foo:n { V }
```

e ora possiamo dare l'istruzione

```
\example_foo:V \l_example_var_pippo_tl
```

che sarebbe equivalente a `\example_foo:n { Pluto }`

Varianti

Variante della variante

```
\cs_generate_variant:Nn \example_foo:n { v }
\example_foo:V { l_example_var_pippo_tl }
```

Ovviamente questa seconda variante si adopererà per casi in cui il nome da costruire dipenda dal contesto e sarà passato come argomento a qualche altra funzione

Espansione

Ci sarebbe da parlare per ore di espansione

Espansione

Ci sarebbe da parlare per ore di espansione

La variante più interessante e anche la più recente introdotta in **expl3** è e

Espansione

Ci sarebbe da parlare per ore di espansione

La variante più interessante e anche la più recente introdotta in **expl3** è e

Che problemi risolve?

Espansione

Ci sarebbe da parlare per ore di espansione

La variante più interessante e anche la più recente
introdotta in **expl3** è e

Che problemi risolve?

In linguaggio matematico, se f è una funzione di due variabili, mentre g è una funzione di una variabile a valori in $\mathbb{R}^2$, la notazione

$$f(g(x))$$

è perfettamente lecita

Espansione

Il problema è che TEX elabora macro e lo fa nell'ordine opposto

da fuori a dentro

Espansione

Il problema è che TEX elabora macro e lo fa nell'ordine opposto
da fuori a dentro

Supponiamo di avere `\example_a:nn` e `\example_b:n`
dove l'output della seconda è definito con

```
\cs_new:Nn \example_b:n
 {
  {--#1--}{``#1''}
 }
```

e quindi

```
\example_b:n { x } → {--x--}{``x''}
```

Espansione

Il problema è che TEX elabora macro e lo fa nell'ordine opposto
da fuori a dentro

Supponiamo di avere `\example_a:nn` e `\example_b:n`
dove l'output della seconda è definito con

```
\cs_new:Nn \example_b:n
 {
  {--#1--}{``#1''}
 }
```

e quindi

```
\example_b:n { x } → {--x--}{``x''}
```

Come facciamo a dare questo in pasto a `\example_a:nn`?

Espansione

In passato sarebbe stato necessario conoscere il numero di passi di espansione oppure adoperare l'espansione ricorsiva (limitata) con `\romannumeral`

Espansione

In passato sarebbe stato necessario conoscere il numero di passi di espansione oppure adoperare l'espansione ricorsiva (limitata) con `\romannumeral`

Ora è semplice!

Espansione

In passato sarebbe stato necessario conoscere il numero di passi di espansione oppure adoperare l'espansione ricorsiva (limitata) con `\romannumeral`

Ora è semplice!

```
\exp_last_unbraced:Ne \example_a:nn { \example_b:n { x } }
```

Il nome dell'*attivatore* non è forse il meglio che si possa immaginare, ma fa il suo dovere

Espansione

In passato sarebbe stato necessario conoscere il numero di passi di espansione oppure adoperare l'espansione ricorsiva (limitata) con `\romannumeral`

Ora è semplice!

```
\exp_last_unbraced:Ne \example_a:nn { \example_b:n { x } }
```

Il nome dell'*attivatore* non è forse il meglio che si possa immaginare, ma fa il suo dovere

Lo possiamo considerare come un simbolo della composizione di funzioni