Scrivere un pacchetto per LaTeX3
Il pacchetto kantlipsum

Enrico Gregorio

Sommario

Il pacchetto kantlipsum viene usato come pretesto per illustrare alcune tecniche di programmazione con il linguaggio LaTeX3.

Abstract

The kantlipsum package is used as pretext to illustrate some programming techniques with the LaTeX3 language.

1 Introduzione

No, LaTeX3 non è ancora uscito. Ma è già disponibile e in uno stato piuttosto avanzato il livello più basso, quello della *programmazione*. Uno dei criteri che guidano gli sviluppatori è infatti la divisione più netta possibile fra i vari livelli: programmazione, produzione di interfacce con l'utente e livello utente. Alla fine i documenti LaTeX3 non saranno tanto diversi dagli attuali, ma tutti i comandi come `\section` saranno ridefiniti in termini di strutture del livello intermedio che a loro volta saranno basate sul livello più basso.

L'esperienza di LaTeX 2_ε permetterà di definire meglio tutti i comandi per gli utenti, evitando le acrobazie che hyperref o altri pacchetti devono eseguire per intervenire su comandi definiti quando, per esempio, degli ipertesti non si aveva la minima idea.

Con il livello intermedio completo non saranno più necessari pacchetti come titlesec, caption o geometry, perché le interfacce di accesso ai titoli, alle didascalie o alle impostazioni di pagina saranno molto più maneggevoli di ora. O forse ci sarà titlesec3 che avrà un compito più facile, potendosi appoggiare a un'interfaccia notevolmente più semplice da gestire.

Il livello più basso, quello della programmazione, è *molto* diverso dall'attuale, ma anche molto più ricco: non si deve inventare l'acqua calda ogni volta; forse a prezzo di scottarsi all'inizio.

Nel pacchetto che descriveremo viene solo sfiorata la superficie della ricca dotazione di funzioni messe a disposizione da LaTeX3. La distribuzione contiene la descrizione commentata di tutte le funzioni nel file `interface3.pdf`.

2 I pacchetti lipsum e kantlipsum

Qualche anno fa il pacchetto lipsum permise di evitare di inventarsi testi fasulli per scrivere esempi o esaminare pagine complete. Il comando `\lipsum` compone sette capoversi di finto latino, il famoso *Lorem ipsum dolor sit amet* che è in uso fra i tipografi da lungo tempo. Con `\lipsum[1-150]` si compongono tutti i capoversi che sono a disposizione, ma naturalmente è possibile anche sceglierne di meno o uno solo con `\lipsum[2]` o quello che si vuole (spesso uso il secondo che è più corto del primo).

Un difetto di lipsum è che il testo pseudolatino non si comporta bene con le regole di sillabazione dell'inglese, producendo spesso divisioni di riga imperfette e relative *overfull box*.

Il libro *Dive into Python* include un programmino scritto in Python che produce capoversi in stile kantiano, imitando un programma per Mac OS dei tempi andati. Per la precisione, il primo sviluppatore aveva cercato di imitare lo stile della *Kritik der reiner Vernunft*, cioè la "Critica della ragion pura", uno dei più famosi trattati di Immanuel Kant.

L'idea di usare questo programmino per produrre testo fittizio circolava già e ne ho colto l'occasione per esercitarmi nella nuova sintassi di LaTeX3. Il primo passo è stato di produrre un po' di capoversi, per ora sono 164; ne vediamo un esempio:

> As any dedicated reader can clearly see, the Ideal of practical reason is a representation of, as far as I know, the things in themselves; as I have shown elsewhere, the phenomena should only be used as a canon for our understanding. The paralogisms of practical reason are what first give rise to the architectonic of practical reason. As will easily be shown in the next section, reason would thereby be made to contradict, in view of these considerations, the Ideal of practical reason, yet the manifold depends on the phenomena. Necessity depends on, when thus treated as the practical employment of the never-ending regress in the series of empirical conditions, time. Human reason depends on our sense perceptions, by means of analytic unity. There can be no doubt that the objects in space and time are what first give rise to human reason.

La sintassi è del tutto simile a quella di lipsum.

- `\kant` produce i primi sette capoversi;
- `\kant[3]` produce il capoverso numero 3;
- `\kant[11-42]` produce i capoversi dal numero 11 al numero 42.

Se si usa la variante asterisco `\kant*` (con l'eventuale argomento facoltativo come prima) non ci saranno interruzioni di capoverso alla fine di ciascun pezzo.

In realtà si può scambiare il significato delle due varianti dando l'opzione `nopar` al caricamento di kantlipsum, esattamente come accade in lipsum. Il nuovo pacchetto ha un'altra opzione, `numbers`, che appiccica all'inizio di un capoverso il numero, in modo da poter controllare meglio l'effetto:

> 2 • Let us suppose that the noumena have nothing to do with necessity, since knowledge of the Categories is a posteriori. Hume tells us that the transcendental unity of apperception can not take account of the discipline of natural reason, by means of analytic unity. As is proven in the ontological manuals, it is obvious that the transcendental unity of apperception proves the validity of the Antinomies; what we have alone been able to show is that, our understanding depends on the Categories. It remains a mystery why the Ideal stands in need of reason. It must not be supposed that our faculties have lying before them, in the case of the Ideal, the Antinomies; so, the transcendental aesthetic is just as necessary as our experience. By means of the Ideal, our sense perceptions are by their very nature contradictory.

sarebbe il risultato di `\kant[2]` con quell'opzione.

3 Cominciamo con LaTeX3

Vediamo la struttura di un pacchetto che usi la sintassi di LaTeX3. L'inizio richiede il caricamento di expl3 e l'annuncio del pacchetto:

```
\RequirePackage{expl3}
\GetIdInfo$Id: kantlipsum.dtx
  0.5 2011-12-05 12:00:00Z Enrico $
  {Dummy text in Kantian style}
\ProvidesExplPackage
  {\ExplFileName}{\ExplFileDate}
  {\ExplFileVersion}{\ExplFileDescription}
```

È simile a `\ProvidesPackage`; il comando `\GetIdInfo` permette di evitare di dover scrivere le informazioni su versione, data e così via in più posti.

Le opzioni possono ancora essere dichiarate come nel modo tradizionale, è probabile che LaTeX3 ne definisca uno leggermente diverso.

```
\DeclareOption { par }
  {
   \cs_set:Nn \kgl_star:
     { \c_space_tl }
   \cs_set:Nn \kgl_nostar:
     { \par }
  }
\DeclareOption{ nopar }
  {
   \cs_set:Nn \kgl_star:
     { \par }
   \cs_set:Nn \kgl_nostar:
     { \c_space_tl }
  }
\DeclareOption{ numbers }
  {
   \cs_set:Nn \kgl_number:n
     {
      #1\nobreakspace
      \textbullet\nobreakspace
     }
  }
\cs_new_eq:NN \kgl_number:n \use_none:n
\ExecuteOptions { par }
\ProcessOptions \scan_stop:
```

Già si vede qualcosa di bizzarro. La prima cosa da sapere è che nell'ambiente di programmazione di LaTeX3 gli spazi sono *ignorati*, quindi possono essere usati con generosità senza rischi. Se si desidera un vero spazio, al suo posto si scrive `~`, come vedremo poi nella sezione sui messaggi.

L'altra cosa strana sono i nomi dei comandi. In LaTeX3 ogni comando dice qualcosa di sé già nel nome. Per esempio, `\c_space_tl` dice di sé che è una costante e che è di tipo 'token list'. Non è altro che la vecchia conoscenza `\space`. Non si può usare `~` là perché comunque TeX ignora gli spazi espliciti dopo i nomi di comandi.

I comandi propri di kantlipsum contengono tutti il prefisso `kgl` (per 'Kant generator lipsum') che è propriamente il nome del *modulo*. Per esempio, `\cs_set:Nn` fa parte del modulo `cs` ('control sequences'). La convenzione non è seguita in modo rigorosissimo: le costanti 'universali' come `\c_space_tl` non hanno nome di modulo. Il suffisso finale è diverso: `_tl` e `:Nn`. Un suffisso come il primo è riservato a variabili e costanti; un suffisso che comincia con i due punti è riservato alle *funzioni*.

LaTeX3 cerca di distinguere rigorosamente tra variabili e funzioni; una variabile (o una costante) contiene un dato, una funzione esegue un certo compito. Il compito di `\cs_set:Nn` è di definire un'altra funzione. Il suffisso dice che tipo di argomenti prende la funzione. Nel caso specifico `N` indica un argomento di tipo 'macro' mentre `n` indica una lista di token tra graffe (che può essere anche, secondo le regole sintattiche di TeX, un solo token). La riga

```
\cs_set:Nn \kgl_star: { \c_space_tl }
```

definisce `\kgl_star:` come una funzione senza argomenti, la cui azione è di inserire uno spazio. In LaTeX tradizionale avremmo scritto

```
\newcommand*{\kgl@star}{\space}
```

Notiamo che in LaTeX3 il carattere `@` non ha l'uso a cui molti sono abituati; come separatore per indicare comandi del livello più basso si usa infatti `_`.

Una sorpresa si ha con

```
\cs_set:Nn \kgl_number:n
  {
   #1\nobreakspace
   \textbullet\nobreakspace
  }
```

La funzione `\kgl_number:n` è quella che produce il numero, se viene scelta l'opzione `numbers`, oppure non fa nulla. La definizione tradizionale sarebbe stata

```
\newcommand*{\kgl@number}[1]{%
  #1~\textbullet~}
```

Un momento! Dov'è indicato il numero di argomenti nella nuova? Lo dice semplicemente il nome della funzione: `\kgl_number:n` è una funzione che prende un argomento e `\cs_set:Nn` prima di tutto esamina il *nome* della funzione da definire e ne imposta la definizione in modo opportuno.

I due punti appaiono solo nelle *funzioni* e ciò che segue è la *firma* (in inglese *signature*) della funzione stessa. Indica il numero e il tipo degli argomenti della funzione: `n` sta per un normale argomento tra graffe, `N` per un unico token simbolico.

Alla fine della serie di opzioni c'è la riga

```
\cs_new_eq:NN \kgl_number:n \use_none:n
```

che sarebbe stata

```
\let\kgl@number\@gobble
```

La differenza fra le famiglie di funzioni `\cs_set` e `\cs_new` è che quelle del secondo tipo controllano che il nome della funzione da definire non abbia già un significato; inoltre le funzioni `\cs_new` agiscono *globalmente*, mentre le `\cs_set` solo localmente. Come si vede dal suffisso, `\cs_new_eq:NN` prende come argomenti due funzioni; `eq` sta per 'equivalent': la prima è quella da definire, che viene resa equivalente alla seconda. Propriamente sostituisce `\global\let`, in questo caso non c'è differenza.

Dopo il classico `\ProcessOptions` c'è `\scan_stop:`, che è la vecchia conoscenza `\relax`, cioè una funzione non espandibile che non fa nulla. Segue `\RequirePackage{xparse}`, niente di strano.

Un'altra caratteristica di LaTeX3 è che rinomina *tutte* le primitive di TeX, minimizzando il rischio di conflitti. È ancora aperta la questione di quali saranno impiegabili ancora al livello utente; probabilmente parecchie, se non si vuole che molti documenti richiedano pesanti interventi per essere adattati alla nuova versione. Ma c'è molto tempo per pensarci.

4 Messaggi

La sezione successiva del pacchetto definisce i messaggi di errore o di avviso.

```
\msg_new:nnn {kantlipsum}
  {how-many}
  {The~package~provides~paragraphs~
   1~to~#1.~Values~outside~this~
   range~will~be~ignored.}
\msg_new:nnnn {kantlipsum}
  {already-defined}
  {Control~sequence~#1~already~defined.}
  {The~control~sequence~#1~is~
   already~defined,~I'll~ignore~it}
```

A ogni messaggio viene associato un nome; vedremo più avanti come vengono usati. Si noti che gli spazi vanno indicati con `~` ed eventuali fine riga con `\\` (non serve più `\MessageBreak`); ma i messaggi vengono mandati a capo in modo automatico e un `\\` serve raramente.

Il primo sarà un messaggio informativo (solo tre argomenti); il secondo di errore, perché ha i due tipici modi di avvisare l'utente: corto e lungo (che si legge premendo `h`). I messaggi possono leggere fino a quattro argomenti, in questo caso ciascuno ne ha uno.

5 Dichiarazioni di variabili e costanti

Una delle raccomandazioni degli sviluppatori è di dichiarare in anticipo le variabili e le costanti da usare in seguito. In questo pacchetto dichiariamo due variabili numeriche e una sequenza:

```
\int_new:N \l_kgl_start_int
\int_new:N \l_kgl_end_int
\seq_new:N \g_kgl_pars_seq
\seq_gput_right:Nx \g_kgl_pars_seq
  {This~is~the~Kant~lipsum~generator.}
```

Si tratta di due variabili intere (contatori, nel gergo usuale) e una sequenza, cioè una lista indicizzabile di elementi. Le sequenze cominciano da zero, perciò la inizializziamo con un elemento fasullo.

6 Comandi per l'utente

Qui faremo tesoro delle funzionalità di xparse che permette facilmente di definire comandi con varianti e con argomenti facoltativi.

```
\NewDocumentCommand{\kant}
  {s >{\SplitArgument{1}{-}}O{1-7} }
```

```
{
 \group_begin:
 \IfBooleanTF{#1}
   {
    \cs_set_eq:NN \kgl_par: \kgl_star:
   }
   {
    \cs_set_eq:NN \kgl_par: \kgl_nostar:
   }
 \kgl_process:nn #2
 \kgl_print:
 \group_end:
}
```

Con `s` dichiariamo che il comando `\kant` ha una variante asterisco e con `O` che ha un argomento facoltativo. La presenza dell'asterisco rende vero il condizionale `\IfBooleanTF{#1}`. Lo scopo è di definire `\kgl_par:` equivalente a `\kgl_star:` oppure a `\kgl_nostar:`. Questi ultimi riceveranno una definizione più avanti.

L'argomento opzionale ha, come valore di default, `1-7`. Nel caso non si voglia dare un valore di default, si può indicare l'argomento con `o` e, in tal caso, l'assenza dell'argomento opzionale rende vero il condizionale `\IfNoValueTF{#2}` (qui, perché l'argomento è il secondo nella lista). In genere un argomento opzionale viene indicato nel secondo argomento di `\NewDocumentCommand` con la sola `o` o con `O{...}`; in questo caso, però, diciamo anche a LaTeX3 di spezzare in corrispondenza del trattino (se c'è) quello che viene passato come argomento: questa è la funzione di `>{\SplitArgument{1}{-}}` che indica come va massaggiato l'argomento che segue. Nei tre casi possibili, ciò che verrà visto realmente come `#2` è:

- `{1}{7}` se l'argomento facoltativo non è espresso e quindi LaTeX usa quello di default;
- `{42}{\NoValue}` se si dà come argomento facoltativo `[42]` (o un altro numero qualsiasi;
- `{11}{42}` se si dà `\kant[11-42]`.

Lo stesso accade con `\kant*`, naturalmente. Il primo argomento di `\SplitArgument` dice quanti gruppi di parentesi graffe vogliamo (uno in più di quanto scritto) e il secondo qual è il token che usiamo come separatore; nel primo gruppo va la parte dell'argomento fino al primo trattino, nel secondo tutto il resto. Se non c'è trattino, il secondo gruppo conterrà `\NoValue`.

Nel primo caso le variabili intere `\l_kgl_start_int` e `\l_kgl_end_int` vengono impostate a 1 e 7; nel secondo caso viene chiamata la funzione `\kgl_process:nn` che ha *due* argomenti. Al termine viene eseguita la funzione `\kgl_print:`. Tutto ciò in un gruppo per mantenere locali i significati assegnati alle funzioni e i valori alle variabili. Naturalmente `\group_begin:` e `\group_end:` sono le vecchie conoscenze `\begingroup` e `\endgroup`.

Il pacchetto definisce anche `\kantdef`; lo lasciamo da parte, per ora.

7 Funzioni interne

Definiamo ora le funzioni che servono per la meccanica delle macro a livello utente. La prima è `\kgl_process:nn`.

```
\cs_new:Nn \kgl_process:nn
  {
   \int_set:Nn \l_kgl_start_int {#1}
   \IfNoValueTF{#2}
     { \int_set:Nn \l_kgl_end_int {#1} }
     { \int_set:Nn \l_kgl_end_int {#2} }
  }
```

Il suo scopo è di esaminare ciò che c'era nell'argomento facoltativo dato a `\kant` che sarà della forma `{42}{\NoValue}` o `{11}{42}`; LaTeX3 si trova davanti uno fra

```
\kpg_process:nn {42}{\NoValue}
\kpg_process:nn {11}{42}
```

In ogni caso la variabile (intera) `\l_kgl_start_int` viene impostata al valore dato dal primo argomento. La variabile `\l_kgl_start_int` viene impostata allo stesso valore nel primo caso, al secondo argomento altrimenti.

Sistemate le variabili, vediamo com'è definita la funzione `\kgl_print:`. Prima definiamo una funzione ausiliaria

```
\cs_new:Nn \kgl_use:n
  {
   \kgl_number:n {#1}
   \seq_item:Nn \g_kgl_pars_seq {#1}
  }
```

Essenzialmente è 'stampa il capoverso kantiano con il numero dato come argomento'. La funzione `\kgl_number:n` stampa il numero (se è stata data l'opzione `numbers`) oppure ingoia l'argomento e poi accede all'elemento con il dato numero della sequenza contenente i capoversi kantiani.

```
\cs_new:Nn \kgl_print:
  {
   \prg_stepwise_function:nnnN
     {\l_kgl_start_int}
     {1}
     {\l_kgl_end_int}
     \kgl_use:n
  }
```

La funzione fa avviare un ciclo. Il primo argomento di `\prg_stepwise_function:nnnN` è il valore iniziale, il secondo è il passo, il terzo è il valore finale, il quarto la funzione da eseguire, che deve

prendere un argomento. Per esempio, se i valori fossero 2 e 5, il risultato sarebbe equivalente alla successione di comandi

```
\kgl_use:n {2} \kgl_use:n {3}
\kgl_use:n {4} \kgl_use:n {5}
```

e quindi verrebbero stampati i capoversi kantiani dal 2 al 5.

Naturalmente `\kgl_print:n` non è strettamente necessaria: il codice avrebbe potuto essere inserito direttamente nella definizione di `\kant`. Ma gli sviluppatori di LaTeX3 insistono nel chiedere ai programmatori di spezzare il codice in moduli più piccoli e facilmente controllabili. I calcolatori moderni non sono molto rallentati da queste espansioni in più e la chiarezza è certamente un traguardo importante.

Ultima funzione: quella che definisce i capoversi.

```
\cs_new:Nn \kgl_newpara:n
  {
   \seq_gput_right:Nn \g_kgl_pars_seq
     { #1 \kgl_par: }
  }
```

Il capoverso, dato come argomento, viene inserito come nuovo elemento della sequenza globale che li conterrà tutti, con l'aggiunta alla fine di `\kgl_par:`. È facilissimo per la funzione `\kgl_use:n` accedere ai vari elementi della sequenza.

Questo non sarebbe stato possibile in LaTeX 2_ε, dove il concetto di sequenza non è presente. Non volendo definirsi tutta l'infrastruttura, si sarebbe potuto ovviare definendo, come fa lipsum, tante macro contenenti i vari capoversi. Per automatizzare la faccenda si sarebbe potuto usare un contatore e definire le macro con nomi come `\kgl@1`, `\kgl@2` e così via. Vediamo come:

```
\@tempcnta=\z@
\def\kgl@newpara#1{%
  \advance\@tempcnta\@ne
  \expandafter\def\csname kgl@%
    \number\@tempcnta\endcsname{%
    #1\kgl@par}%
}
```

Illeggibile.

8 I capoversi

Il codice prosegue con la definizione delle frasi kantiane.

```
\group_begin:
\char_set_catcode_space:n {`\ }

\kgl_newpara:n {As any dedicated reader
can clearly see, the Ideal of practical
reason is a representation of, as far as
I know, the things in themselves; as I
have shown elsewhere, the phenomena
...
Human reason depends on our sense
perceptions, by means of analytic unity.
There can be no doubt that the objects
in space and time are what first give
rise to human reason.}
```

⟨*altri 163 capoversi simili*⟩

```
\group_end:

\msg_info:nnx{kantlipsum}{how-many}
  {
   \int_eval:n
     {\seq_length:N \g_kgl_pars_seq - 1}
  }
```

Apriamo un gruppo e, per comodità, facciamo tornare lo spazio al suo significato usuale, per non dover scrivere `~` al suo posto nel testo. L'effetto di ciascuna delle funzioni `\kgl_newpara:n` è di aggiungere successivamente i capoversi alla sequenza `\g_kgl_pars_seq`. L'elemento numero 1 sarà

```
As any dedicated reader can clearly see,
...
give rise to human reason.\kgl_par:
```

L'azione di `\kgl_par:` è di emettere un comando `\par` se si è con l'opzione `par`, uno spazio altrimenti (ma la logica è rovescia se il tutto è prodotto con `\kant*`).

Alla fine emettiamo un messaggio informativo (solo nel file log): `\msg_info:nnx` prende come argomenti i due indicativi del messaggio definito all'inizio e un terzo argomento (espanso) che verrà inserito al posto di `#1` nel messaggio. Dobbiamo fare un piccolo calcolo perché la lunghezza della sequenza comprende anche l'elemento 0.

9 Il comando mancante

C'è un altro comando, definito più o meno solo per fare un esempio. Un utente del forum chiese come poter definire una macro che si espandesse a uno dei capoversi di lipsum. Perciò in kantlipsum è prevista la funzione.

```
\NewDocumentCommand{\kantdef}{mm}
  {
   \group_begin:
   \cs_set_eq:NN \kgl_number:n \use_none:n
   \cs_set_eq:NN \kgl_par: \prg_do_nothing:
   \cs_if_exist:NTF #1
     {
      \msg_error:nnx
        { kantlipsum }
        { already-defined }
        { \token_to_str:N #1 }
     }
     {
```

```
      \cs_new:Npx #1 { \kgl_use:n {#2} }
    }
  \group_end:
}
```

Usiamo ancora le funzionalità di xparse, definendo il comando con due argomenti obbligatori, indicati con `mm` ('mandatory'). Si apre un gruppo e lì si ridefiniscono `\kgl_number:n` come `\use_none:n` (cioè ingoiare l'argomento) e `\kgl_par:n` come `\prg_do_nothing:`, sinonimo di 'non fare nulla', ma espandibile (è il vecchio `\@empty`). A questo punto entra in azione un condizionale: `\cs_if_exist:NTF` controlla se il token dato come primo argomento è definito e se lo è esegue il secondo argomento, altrimenti il terzo. Perciò, se dicessimo `\kantdef{\kant}{1}`, ovviamente saremmo nel ramo 'vero' e si avrebbe il messaggio di errore che il comando è già definito e il secondo argomento sarebbe ignorato. Se invece diciamo `\kantdef{\pippo}{1}` (e supponiamo che `\pippo` non sia definito), verrà eseguito

```
\cs_new:Npx \pippo { \kgl_use:n {1} }
```

Qui usiamo un tipo ancora diverso di `\cs_new`. Infatti `\pippo` è un comando a livello utente e quindi non ha il suffisso che dice quanti argomenti prende. Dunque dobbiamo specificarlo come tutto ciò che va da `\pippo` alla graffa aperta: niente, quindi non ci sono argomenti. Ricordiamo che `\kgl_use:n {1}` contiene `\kgl_number:n` e `\kgl_par:`, ma siccome ne abbiamo neutralizzato le azioni, queste funzioni spariranno e resterà solo il testo.

Di fatto `\cs_new:Npx` è semplicemente `\long\xdef` con in più il controllo che il comando dato come primo argomento non sia definito.

Una piccola annotazione: abbiamo dovuto costruirci il messaggio di errore perché ancora non ci sono tutte le interfacce utente e se usassimo semplicemente `\cs_new:Npx` avremmo un errore ma il comando sarebbe *ridefinito* comunque.

▷ Enrico Gregorio
Dipartimento di Informatica
Università di Verona
Enrico dot Gregorio at univr dot it