Sesión X

LaTeX avanzado I

- Descomponer el documento en ficheros
- Números y contadores
- Comandos y entornos
- Entrega de ejercicios

Sección 1

Descomponer el documento en ficheros

input

Escribir un libro completo en un único archivo no es cómodo.
Por eso LaTeX permite escribir modularmente.
Podemos escribir en diferentes archivos `.tex`, y luego juntarlos en un principal.

Código [modular.tex]

```
\documentclass{standalone}
\begin{document}
        \input{modulo1.tex}
        \input{modulo2.tex}
\end{document}
```

Código [modulo1.tex]

```
Un texto.
```

Código [modulo2.tex]

```
Otro texto.
```

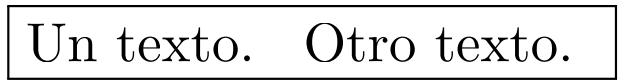


Figura: Resultado de compilar `modular.tex`

include

https://tex.stackexchange.com/questions/246/when-should-i-use-input-vs-include (1405 upvotes)

\input{filename} imports the commands from filename.tex into the target file; it's equivalent to typing all the commands from filename.tex right into the current file where the \input line is.

\include{filename} essentially does a \clearpage before and after \input{filename}, together with some magic to switch to another .aux file, and omits the inclusion at all if you have an \includeonly without the filename in the argument. This is primarily useful when you have a big project on a slow computer; changing one of the include targets won't force you to regenerate the outputs of all the rest.
\include{filename} gets you the speed bonus, but it also can't be nested, can't appear in the preamble, and forces page breaks around the included text.

\input anidados

\input translada copia-y-pega el archivo indicado.
Los path deben darse siempre con respecto al archivo que compila.

- main.tex
- folder1
  - file1.tex
  - folder2
    - file2.tex

Código [main.tex]

```
\documentclass{standalone}
\begin{document}
\input{folder1/file1.tex}
\end{document}
```

Código [file1.tex]

```
Un texto.
\input{folder1/folder2/file2.tex}
```

Código [file2.tex]

```
Otro texto.
```

El paquete `import`

Este paquete nos permite utilizar `paths` relativos

- `main.tex`
- folder1
  - `file1.tex`
  - folder2
    - `file2.tex`

Código [`main.tex`]

```
\documentclass{standalone}
\usepackage{import}
\begin{document}
\subimport{folder1/}{file1.tex}
\end{document}
```

Código [`file1.tex`]

```
Un texto.
\subimport{folder2/}{file2.tex}
```

Código [`file2.tex`]

```
Otro texto.
```

Sección 2

Números y contadores

Operaciones con números

`\numexp` permite operar con enteros.

Código

```
\def\x{3}
\x,
\x*\x,
\the\numexpr\x*\x\relax
```

3, 3*3, 9

Se envuelve en `\the` y `\relax` por razones internas de LaTeX[1].
Se pueden operar también dimensiones, y otros tipos de números.

Ver:
`https://tex.stackexchange.com/questions/245635/formal-syntax-rules-of-dimexpr-numexpr-glueexpr`

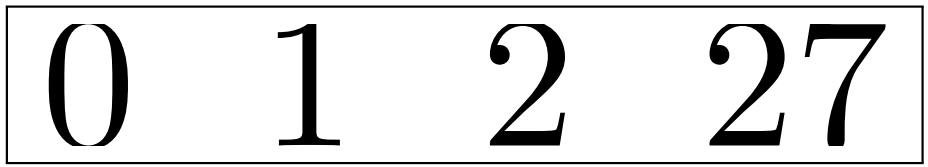
[1] Expansión de tokens que discutiremos más adelante

Contadores

Los contadores son enteros con funcionalidades adicionales

Código

```
\documentclass{standalone}

\begin{document}

\newcounter{micontador}
\arabic{micontador}

\stepcounter{micontador}
\arabic{micontador}

\addtocounter{micontador}{1}
\arabic{micontador}

\setcounter{micontador}{27}
\arabic{micontador}

\end{document}
```



Contadores

Código

```
\documentclass{standalone}

\begin{document}

\newcounter{micontador}
\setcounter{micontador}{1}

\arabic{micontador}
\themicontador

\alph{micontador}
\Alph{micontador}

\roman{micontador}
\Roman{micontador}

\fnsymbol{micontador}
\end{document}
```

1 1a A i I *

numberwithin

Se puede forzar a un contador a depender de otros. Por ejemplo
`\numberwithin{equation}{section}`
hace que las ecuaciones de la sección n se numeren como $(n,1)$, $(n,2)$, ...

numberwithin

Se puede forzar a un contador a depender de otros. Por ejemplo
`\numberwithin{equation}{section}`
hace que las ecuaciones de la sección n se numeren como $(n,1)$, $(n,2)$, ...

Esencialmente lo que se hace es reiniciar el contador de ecuaciones cuando se cambia de sección y hacer
`\renewcommand{\theequation}{\thesection.\arabic{equation}}`

Sección 3

Comandos y entornos

Creando comandos: \newcommand[1]

Código

```
\documentclass{standalone}

\newcommand{\deciralgo}
    {Digo: ``Hola''.}

\begin{document}
    \deciralgo
\end{document}
```

Digo: “Hola”.

[1] Nota histórica: \newcommand es un overlay para LaTeX del comando \def de TeX. Más detalles en link.

\newcommand

Código

```
\documentclass{standalone}

\newcommand{\deciralgo}[1]
    {Digo: ``#1''.}

\begin{document}
    \deciralgo{Hola}
    \deciralgo{Adios}
\end{document}
```

Digo: “Hola”. Digo: “Adios”.

\newcommand

Código

```
\documentclass{standalone}

\newcommand{\deciralgo}[2][Digo]
    {#1: ``#2''.}

\begin{document}
    \deciralgo{Hola}

    \deciralgo[Grito]{Adios}
\end{document}
```

\renewcommand

Si una comando ya está definido, puede eliminarse y volverse a definir con
`\renewcommand{\cmd}{defn}`
Link para más detalles.

\newenvironment

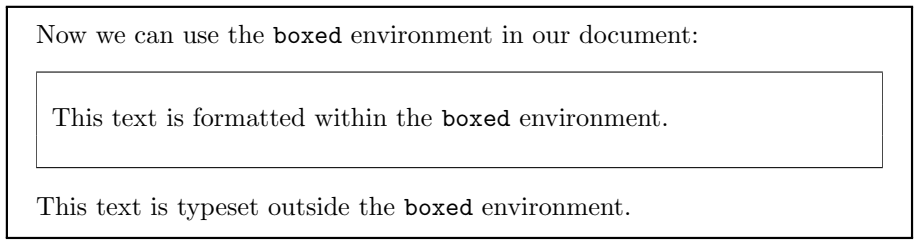
`\newenvironment{name}[numarg][optarg_default]{begin_def}{end_def}`
donde

- `name` is the name of this user-defined argument;
- `numarg` is the number of arguments, from 1 to 9, this environment accepts. If `[numarg]` is omitted then the environment does not accept any arguments—such as the boxed environment defined in the next example;
- `optarg_default` makes the first argument optional and provides a default value—i.e., it is the value used if an optional argument value is not provided;
- `begin_def` is LaTeX code executed when the environment starts (opens), i.e., when you write `\begin{name}`. Within this code you can use arguments accepted by the environment—note that the optional argument is #1 and the remaining arguments are accessed using #2 to `#numarg`;
- `end_def` is LaTeX code executed when the environment ends (closes); i.e., when you write `\end{name}`. You cannot use any of the arguments within this code section.

\newenvironment

Código

```
%https://www.overleaf.com/learn/latex/Environments
\documentclass[varwidth]{standalone}
\newenvironment{boxed}
    {\begin{center}
    \begin{tabular}{|p{0.9\textwidth}|}
    \hline\\
    }
    {
    \\\\\hline
    \end{tabular}
    \end{center}
    }
%
\begin{document}
Now we can use the \texttt{boxed} environment in
↪  our document:

\begin{boxed}
This text is formatted within the \texttt{boxed}
↪  environment.
\end{boxed}

This text is typeset outside the \texttt{boxed}
↪  environment.
\end{document}
```



\newenvironment

Código

```
%https://www.overleaf.com/learn/latex/Environments
\documentclass[varwidth]{standalone}
\newenvironment{boxed}[2][This is a box]
    {\begin{center}
    Argument 1 (\#1)=#1\\[1ex]
    \begin{tabular}{|p{0.9\textwidth}|}
    \hline\\
    Argument 2 (\#2)=#2\\[2ex]
    }
    {
    \\\\\hline
    \end{tabular}
    \end{center}
    }
\begin{document}

\begin{boxed}{Some preliminary text}
This text is \textit{inside} the environment.
\end{boxed}

\begin{boxed}[This is not the default
↪  value]{Some more preliminary text}
    This text is still \textit{inside} the
    ↪  environment.
\end{boxed}

\end{document}
```

Argument 1 (#1)=This is a box

Argument 2 (#2)=Some preliminary text

This text is *inside* the environment.

Argument 1 (#1)=This is not the default value

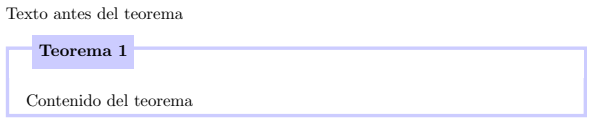
Argument 2 (#2)=Some more preliminary text

This text is still *inside* the environment.

Teorema personalizado con mdframed I

```
% Adaptado de
%https://tex.stackexchange.com/questions/227901/difficult-mdframed-example
\documentclass[]
    {article}
\usepackage{amsmath}
\usepackage[framemethod=TikZ]{mdframed}
%% the following is common for all examples in mdframed manual
\mdfsetup{skipabove=\topskip,skipbelow=\topskip}
%%% upto here
\newcounter{theo}[section]
\newenvironment{theo}[1][]{%
    \stepcounter{theo}%
    \mdfsetup{%
        frametitle={
            %
            \tikz[baseline=(current bounding box.east),outer sep=0pt]
            \node[anchor=east,rectangle,fill=blue!20]{\strut Teorema~\thetheo};
        }
    }%
    \mdfsetup{
        innertopmargin=10pt,linecolor=blue!20,%
        linewidth=2pt,topline=true,
        frametitleaboveskip=\dimexpr-\ht\strutbox\relax,
    }
    \begin{mdframed}[]\relax%
}
{
    \end{mdframed}
}
```

Teorema personalizado con `mdframed` II




```
\begin{document}
    \noindent Texto antes del teorema

    \begin{theo}
        Contenido del teorema
    \end{theo}
\end{document}
```

A este documento le falta la posibilidad de poner nombre a los teoremas. Volveremos sobre esto.

Sección 4

Entrega de ejercicios

Ejercicio a entregar

Entregar un documento `semana10.tex`:

1. Cada uno de los siguientes ejercicios en un archivo .tex distintos, con un `\input` en `semana10.tex`.
2. Un nuevo comando `\fraccion` que reciba dos números y tal que `\fraccion{8}{5}` devuelva
$$\begin{array}{r} 8 \\ \times\ 5 \\ \hline 40 \end{array}$$
3. Crear un entorno personalizado de ejercicio, con un contador que se numere relativo a la sección en la que se encuentra (no utilizar librerías).
4. Un entorno personalizado de teorema, que se numere con su propio contador y sólo utilice números impares.
(Puntos adicionales si los teoremas aparecen enmarcados)