Tablas y listas

Composición de textos científicos

6 de noviembre de 2020

1. Más allá de las matrices en modo matemático

Ya habíamos comentado que había un entorno llamado `array` menos automático que `matrix`. En pocas palabras, en su uso más básico da lugar a una matriz en la que podemos especificar el alineamiento mediante los caracteres `l`, `c` o `r` que indican justificado a la izquierda, al centro o la derecha. Se ha de poner uno por columna en un bloque entre llaves justo después de abrir el entorno. Para ilustrar el uso de `array` y su comparación con `matrix` consideremos

```
\begin{pmatrix}
 1 & 1 & 1 \\ 12 & 12 & 12 \\ 123 & 123 & 123
\end{pmatrix}
\qquad
\left(
\begin{array}{lcr}
 1 & 1 & 1 \\ 12 & 12 & 12 \\ 123 & 123 & 123
\end{array}
\right)
```

que da lugar a

$$\begin{pmatrix}
 1 & 1 & 1 \\ 12 & 12 & 12 \\ 123 & 123 & 123
\end{pmatrix}
\qquad
\left(
\begin{array}{lcr}
 1 & 1 & 1 \\ 12 & 12 & 12 \\ 123 & 123 & 123
\end{array}
\right)$$

La mayor parte de las veces el propósito de `array` no es escribir matrices numéricas con justificaciones exóticas sino colocar fórmulas. Por ejemplo, habíamos visto ya cómo el entorno `cases` nos permitía definir de manera cómoda $f(x) = |x| + |x - 1|$ a trozos, resultando

$$f(x) = \begin{cases} -2x + 1 & \text{si } x \leq 0, \\ 1 & \text{si } 0 < x \leq 1, \\ 2x - 1 & \text{si } x > 1. \end{cases}$$

Pero si no estamos de acuerdo con la justificación no tenemos libertad para cambiarla con este entorno. Utilizando `array` en la forma

```
f(x)=
\left\{
\begin{array}{rl}
 -2x+1 &\text{si }x\le 0,
 \\
 1 &\text{si }0<x\le 1,
 \\
 2x-1 &\text{si }x> 1.
\end{array}
\right.
```

conseguimos un resultado que a algunos les parecerá una mejora estética:

$$f(x)=\left\{\begin{array}{rl} -2x+1 & \text{si } x\le 0,\\ 1 & \text{si } 0<x\le 1,\\ 2x-1 & \text{si } x>1. \end{array}\right.$$

A modo de repaso, si alguien juzga que la llave está demasiado separada puede insertar tras `\left\{` un `\hspace{-5pt}` o alguna otra cantidad, también nos puede parecer que los renglones están muy juntos y preferimos cambiar los `\\` por `\\[2pt]`. Recuérdese no obstante que no hay que abusar de los espacios absolutos. Casi siempre lo más cómodo y lo mejor es dejar hacer a LaTeX.

Otro uso de `array` es crear tablas que tengan mucho contenido en modo matemático. Sobre todo en ellas aparecen a veces ciertas finuras que no hemos considerado con el uso básico anterior, por ejemplo romper la justificación para un elemento en particular, agrupar varios elementos, incluir líneas divisorias, etc. En la práctica estas cuestiones surgen más en tablas en el modo de texto que en fórmulas del modo matemático por eso las posponemos al entorno `tabular` que es el hermano de `array` para texto. Todo o casi todo lo que veremos a continuación para `tabular` se puede usar también con `array` aunque es menos probable que lo necesitemos.

2. Tablas de texto

En ausencia de paquetes especiales, las tablas de texto en LaTeX se crean primordialmente con el entorno `tabular`. Lo más básico es similar a lo mencionado sobre `array` pero casi siempre en texto necesitamos decorar más el resultado. Pongamos por ejemplo que quiero hacer una tabla con fechas de viajes el origen y el destino y con este fin escribo

```
\begin{center}
 \begin{tabular}{ccc}
  \textbf{Fecha}&\textbf{Origen}&\textbf{Destino}
```

```
  \\
  31/01/21&Madrid& Barcelona
  \\
  12/06/21&Helsinki& Pekín
  \\
  18/10/21&Camberra& Manila
 \end{tabular}
\end{center}
```

El resultado es el que cabe esperar:

Fecha	**Origen**	**Destino**
31/01/21	Madrid	Barcelona
12/06/21	Helsinki	Pekín
18/10/21	Camberra	Manila

Casi todos juzgaríamos que una tabla de este estilo, ya sea en un libro o en una *web*, es un poco sosa. Lo primero es que en las tablas que estamos acostumbrados a ver los elementos no están flotando, hay líneas de separación para favorecer la legibilidad. Las líneas verticales que separan las columnas se indican al mismo tiempo que la justificación en la primera línea del entorno. Así `{c|cc}` significa que queremos una línea vertical entre la primera y la segunda columna y `{|c|c|c|}` que las queremos entre todas y al principio y al final. Las líneas horizontales se indican con `\hline` justo antes de empezar la fila correspondiente. Por ejemplo, al cambiar el entorno `tabular` anterior por

```
 \begin{tabular}{|c|c|c|}
  \textbf{Fecha}&\textbf{Origen}&\textbf{Destino}
  \\ \hline
  31/01/21&Madrid& Barcelona
  \\ \hline
  12/06/21&Helsinki& Pekín
  \\ \hline
  18/10/21&Camberra& Manila
  \\ \hline
 \end{tabular}
```

se obtiene:

Fecha	**Origen**	**Destino**
31/01/21	Madrid	Barcelona
12/06/21	Helsinki	Pekín
18/10/21	Camberra	Manila

Nótese que la última línea horizontal no tiene fila que le siga, `\hline` precede a una línea vacía. Si quisiéramos una línea horizontal arriba habría que poner

`\hline` justo después del comienzo del entorno. También se admiten líneas duplicadas tanto verticales como horizontales. Se usan más las horizontales para separar la cabecera de la tabla.

Como ejemplo, si en la fuente anterior cambiamos las primeras líneas por

```
\begin{tabular}{||c|c|c|}
 \hline
 \textbf{Fecha}&\textbf{Origen}&\textbf{Destino}
 \\ \hline\hline
```

el resultado será:

Fecha	**Origen**	**Destino**
31/01/21	Madrid	Barcelona
12/06/21	Helsinki	Pekín
18/10/21	Camberra	Manila

Una variante de `\hline` es `\cline{...}` que tiene como argumento dos números de columna separados por un guión. Lo que indica es que la línea horizontal solo se trazará entre ellas. Por ejemplo, si quisiéramos quitar la línea horizontal entre las dos primeras fechas porque ambas son de temporada alta, lo podríamos conseguir reemplazando en las líneas

```
31/01/21&Madrid& Barcelona
\\ \hline
```

el `\hline` por `\cline{2-3}`. Si por el contrario quisiéramos eliminar la línea entre Madrid y Helsinki porque ambas están en Europa, podríamos `\cline{1-1}` `\cline{3-3}`. Sobre la tabla original con líneas el efecto sería:

Fecha	**Origen**	**Destino**
31/01/21	Madrid	Barcelona
12/06/21	Helsinki	Pekín
18/10/21	Camberra	Manila

Fecha	**Origen**	**Destino**
31/01/21	Madrid	Barcelona
12/06/21	Helsinki	Pekín
18/10/21	Camberra	Manila

A las justificaciones `l`, `c` o `r` de `array` se añade en `tabular` otra[1] que se indica mediante `p{...}` donde los puntos suspensivos representan cierta longitud. Esto permite incluir párrafos del ancho deseado en una tabla.

Por ejemplo, la siguiente tabla se ha obtenido con `|c|p{6.5cm}|`

Tiempo	**Propósito**
Presente	Lo voy a pasar muy bien y voy a sacar las mejores notas posibles.
Futuro	Con mis notazas las mejores empresas se pelearán por mí.

[1] En realidad también funciona en `array` pero tiene una utilidad dudosa porque considera los elementos correspondientes en modo texto.

Seguro que a más de uno no le gusta demasiado que "Presente" y "Futuro" aparezcan en la parte superior de la celda y querrían tener control sobre ello. Esto requiere paquetes especiales o hacer ajustes a ojo con `\raisebox`.

Cualquiera que haya creado una tabla en HTML sabe que a menudo es necesario pegar celdas, sobre todo en horizontal. Esto último se consigue en LaTeX con `\multicolumn{...}{...}{...}`. El primer argumento indica el número de celdas que pegamos, el segundo la justificación que deseamos y el tercer el contenido de las celdas. Por ejemplo, si queremos unir las celdas de "Origen" y "Destino" en una llamada "Itinerario" reemplazaríamos la línea

```
\textbf{Fecha}&\textbf{Origen}&\textbf{Destino}
```

por

```
\textbf{Fecha}&\multicolumn{2}{c|}{\textbf{Itinerario}}
```

Si pusiéramos `c` en vez de `c|` o `|c|` entonces no se cerraría la celda por la derecha. Aunque en principio suene extraño, usar `\multicolumn` con una sola celda tiene sentido porque nos da control sobre la justificación. Por ejemplo, si quisiéramos que "Pekín" apareciera justificado a la derecha bastaría cambiar `Pekín` por

```
\multicolumn{1}{r|}{Pekín}
```

El resultado de estos dos cambios combinados es:

Fecha	**Itinerario**	
31/01/21	Madrid	Barcelona
12/06/21	Helsinki	Pekín
18/10/21	Camberra	Manila

El pegado de celdas en verticales requiere cargar un paquete con

```
\usepackage{multirow}
```

que define un comando `\multirow{...}{...}{...}`. Su primer y tercer argumentos tienen el mismo significado que en `\multicolumn` mientras que el segundo indica el ancho de la columna. Dicho sea de paso, si alguna vez tienes curiosidad o te enfrentas a una tabla imposible, algunos de los paquetes más famosos que definen nuevos comandos y entornos para controlar las tablas son `array`, `tabularx`, `dcolumn` y `booktabs`.

Las líneas verticales que separan las columnas representadas por `|` en la definición de la justificación se pueden cambiar por cualquier expresión. Basta sustituirlas por `@{...}` donde los puntos suspensivos representan el nuevo separador de columnas. La utilidad práctica es reducida aunque es fácil imaginar situaciones en que nos ahorra teclear. Por ejemplo, digamos que en nuestra lista queremos que la ciudad origen y destino estén siempre separadas por una flecha entonces abriríamos el entorno con

```
\begin{tabular}{|c|r@{$\,\to\,$}l|}
```

Se ha utilizado la flecha corta del modo matemático y los espacios pequeños `\,` dejan una separación adicional. La justificación natural de las dos columnas involucradas es `rl` para que el nombre de las ciudades aparezca pegado a la flecha. El resultado es:

Fecha	**Itinerario**
31/01/21	Madrid $\to$ Barcelona
12/06/21	Helsinki $\to$ Pekín
18/10/21	Camberra $\to$ Manila

Como "Itinerario" tiene su propia justificación no se ve afectado por `@{...}`.

Una cosa un poco fastidiosa con el entorno `tabular`, sobre todo al incluir alguna fórmula matemática con un operador grande o incluir un tipo de letra mayor que el habitual, es que a veces no considera espaciamientos verticales adecuados. Por ejemplo, si en la tabla anterior hacemos la"B" de Barcelona mayor mediante `{\Large B}`, el resultado es:

Fecha	**Itinerario**
31/01/21	Madrid $\to$ {\Large B}arcelona
12/06/21	Helsinki $\to$ Pekín
18/10/21	Camberra $\to$ Manila

Una pequeña chapucilla para resolverlo si no queremos cargar paquetes especiales es insertar una línea vertical suficientemente alta de ancho cero (para que sea invisible). Las líneas en TEX se representan con `\rule{...}{...}` donde el primer argumento es el ancho y el segundo el alto. Escribiendo entonces `\rule{0pt}{12pt}{\Large B}` forzamos a que la celda se vuelva más alta para que quepa algo de `12pt` por encima de la línea. El resultado es mucho más satisfactorio:

Fecha	**Itinerario**
31/01/21	Madrid $\to$ {\Large B}arcelona
12/06/21	Helsinki $\to$ Pekín
18/10/21	Camberra $\to$ Manila

Una manera más drástica es redefinir el comando que indica la proporción de los espacios verticales extra. Por ejemplo,

```
\renewcommand{\arraystretch}{1.2}
```

multiplica todos los espacios verticales extra por 1,2 y si en la tabla anterior usamos el entorno con

```
\begin{center}
 \renewcommand{\arraystretch}{1.2}
 \begin{tabular}{|c|r@{$\,\to\,$}l|}
```

obtendremos:

Fecha	**Itinerario**
31/01/21	Madrid → Barcelona
12/06/21	Helsinki → Pekín
18/10/21	Camberra → Manila

Si `\renewcommand{\arraystretch}{1.2}` no estuviera dentro del entorno `center` o de cualquier bloque que separara la definición del resto, tendría un efecto global sobre las tablas siguientes.

Hay además dos longitudes que pueden resultar de utilidad en la presentación de una tabla. En LaTeX las longitudes se asignan por medio de

`\setlength{`nombre de la longitud`}{`valor`}`

La longitud `\arrayrulewidth` mide el grosor de las líneas empleada en la tabla y `\tabcolsep` la separación básica entre columnas, asignándole el valor `0pt` conseguiríamos contenidos de celdas pegados a sus límites verticales. Como ejemplo, para obtener

Fecha	**Itinerario**
31/01/21	Madrid → Barcelona
12/06/21	Helsinki → Pekín
18/10/21	Camberra → Manila

se ha empleado

```
\begin{center}
 \setlength{\arrayrulewidth}{1pt}
 \setlength{\tabcolsep}{30pt}
 \begin{tabular}{|c|r@{$\,\to\,$}l|}
```

El cambio de longitudes necesita estar también de un bloque si no queremos que tenga un efecto global sobre lo que viene después. Si no empleásemos el entorno `center` y quisiéramos que el resultado solo afectase a la tabla en curso, podríamos abrir una llave antes de la asignación de las longitudes y cerrarla tras `\end{tabular}`.

Los ejemplos que hemos visto muestran que una tabla medianamente compleja requieren un código algo elaborado y repetitivo posiblemente con muchos `\multicolumn` en las celdas, lo que motiva la existencia de algunos asistentes para llevar a cabo gráficamente la elaboración de tablas. Una aplicación *web* muy a tener en cuenta es el generador de tablas *online* de `https://www.tablesgenerator.com/`. Mejora los asistentes que he visto en los editores. Puede resultar un poco pesado escribir una tabla grande allí pero nos sacará de más de un apuro si la usamos para recordar cómo se hace algo construyendo algún ejemplo mínimo.

3. Listas

A pesar de que desde el punto de vista del código LaTeX las tablas y las listas no tienen absolutamente nada que ver, es cierto que visualmente las listas recuerdan muchas veces a tablas y esto motiva incluirlas en esta entrega.

Los dos tipos de listas más empleados en LaTeX corresponden a los entornos `itemize` y `enumerate` con la diferencia sugerida por el nombre de que el primero no numera y el segundo sí. En ambos cada ítem debe ir precedido por `\item`.

Por ejemplo,

- Desayunar
- Ir a clase
- Aburrirme

se obtiene con

```
\begin{itemize}
 \item Desayunar
 \item Ir a clase
 \item Aburrirme
\end{itemize}
```

Mientras que si cambiamos `itemize` por `enumerate` obtendremos

1. Desayunar
2. Ir a clase
3. Aburrirme

El contenido de cada punto no está limitado a lo que quepa en una línea, se pueden incluir uno o varios párrafos que quedaran justificados tras el punto o el número que corresponda al entorno. Por ejemplo, con un par de párrafos en el primer punto obtendríamos:

- Desayunar pero desayunar bien fuerte para tener mucha energía, no vale con un par de galletas y un vaso de leche.

  Al menos eso es lo que dice mucha gente aunque pocos lo hagan.

Antes de seguir, no sé si es cosa mía pero a mí siempre me parece que estos entornos dejan más espacio entre un ítem y el siguiente del que a mí me parece agradable. Hay una longitud llamada `\itemsep` que indica cuánto difiere esta separación de cierta cantidad. Cambiando esta longitud

justo a continuación de abrir el entorno se consigue que su ámbito sea local[2]. La separación entre el punto o número y el ítem sí me parece adecuada pero por si alguien quiere modificarla, se llama `\labelsep`. En ejemplo con `enumerate` escribiendo tras `\begin{enumerate}`

```
\setlength{\itemsep}{-5pt}\setlength{\labelsep}{20pt}
```

se obtiene:

1. Desayunar
2. Ir a clase
3. Aburrirme

Si necesitas cosas más sofisticadas sobre márgenes y centrados de listas, el paquete `enumitem` ofrece soluciones a muchos problemas.

Si por alguna razón queremos que no se respete el punto o el número que debiera aparecer, basta indicar la expresión que preferimos entre corchetes como argumento de `\item`. En el caso de `enumerate` cada ítem especial será saltado en la cuenta. Por ejemplo

```
\begin{enumerate}\setlength{\itemsep}{-3pt}
 \item Desayunar
 \item[*.] Ir a clase
 \item Aburrirme
\end{enumerate}
```

daría lugar a

1. Desayunar
*. Ir a clase
2. Aburrirme

Emplear una expresión de unos cuantos caracteres es lícito y puede tener cierto sentido sobre todo en `itemize` pero, si no usamos el paquete antes citado, correremos el riesgo de que haya texto que se escape por el margen. En ese caso quizá el entorno `description` (mucho menos usado) nos parezca más adecuado que `itemize`. Para entender la diferencia, consideremos

```
\begin{itemize}\setlength{\itemsep}{-5pt}
 \item[\bf Uno.] Primer punto.
 \item[\bf Dos.] Segundo punto.
 \item[\bf No lo olvides.] Ojo al margen.
\end{itemize}
```

que genera

[2]Por si te lo estás preguntando, en `tabular` esto no es posible por alguna razón y de ahí lo de aislar las longitudes que hemos visto antes en un bloque.

Uno. Primer punto.
Dos. Segundo punto.
No lo olvides. Ojo al margen.

Cambiando `itemize` por `description` pasaría a ser:

Uno. Primer punto.
Dos. Segundo punto.
No lo olvides. Ojo al margen.

Para terminar veamos las listas múltiples y una suerte de cambio global de los argumentos de `\item`. Antes de ello hay que notar que aunque el paquete `babel` estrictamente no es incompatible con lo que vamos a ver (no da error), sí provoca rarezas a evitar en ciertos idiomas. Con este fin, es conveniente desactivar las decisiones que toma sobre las listas en español incluyendo el parámetro `es-nolists` al cargar `babel`. En el caso de este documento, la cabecera incluye

```
\usepackage[spanish, es-nolists]{babel}
```

Cuando en una lista uno de los ítem es a su vez una lista o la contiene, LaTeX juiciosamente decide utilizar símbolos o numeraciones independientes. Por ejemplo, al compilar el siguiente código[3]

```
\begin{itemize}\setlength{\itemsep}{-3pt}
 \item Voy a la tienda
 \item Cuando llego
      \begin{itemize}\setlength{\itemsep}{-1pt}
       \item Busco la sección de helados
       \item Compro legumbres
      \end{itemize}
 \item Vuelvo a casa
\end{itemize}
```

se obtiene

- Voy a la tienda
- Cuando llego
  - Busco la sección de helados
  - Compro legumbres
- Vuelvo a casa

[3]La separación del margen del segundo bloque `itemize` en la fuente es solo para favorecer la legibilidad y la localización de posibles errores, esos espacios son ignorados. Se podría escribir `\begin{itemize}` justo a continuación de `llego`. De la misma forma en la fuente en una misma línea puede haber varios `\item`.

y para `enumerate` lo mismo produciría

1. Voy a la tienda
2. Cuando llego
   (a) Busco la sección de helados
   (b) Compro legumbres
3. Vuelvo a casa

Ya sea en una lista normal o en una lista de lista uno puede no estar de acuerdo con el símbolo usado por LaTeX. Ya hemos visto cómo modificarlo puntualmente con un argumento de `\item` pero hay algo más potente: Los comandos `\labelitemi`, `\labelitemii`, etc. (hasta cuatro niveles) guardan el tipo de símbolo que introduce `itemize` y lo podemos cambiar a voluntad con

```
\renewcommand{\labelitemi}{...}, etc.
```

Si esta redefinición del comando la hacemos después de abrir el entorno solo tendrá efecto para la lista en curso y sus sublistas. En el último ejemplo de `itemize` cuando utilizamos como primeras líneas

```
\begin{itemize}\setlength{\itemsep}{-3pt}
 \renewcommand{\labelitemi}{$\looparrowright$}
 \renewcommand{\labelitemii}{$\to$}
```

el resultado es:

↬ Voy a la tienda
↬ Cuando llego
   → Busco la sección de helados
   → Compro legumbres
↬ Vuelvo a casa

El análogo de `\labelitemi`, etc. para `enumerate` es `\labelenumi`, etc. En este caso lo que podemos indicar son cinco formas diferentes de numeración:

`\arabic`	Números
`\alph`	Letras minúsculas
`\Alph`	Letras mayúsculas
`\roman`	Números romanos en minúscula
`\Roman`	Números romanos en mayúscula

La forma de numeración escogida se debe aplicar a un contador[4] que según el nivel de la lista se llama `enumi`, `enumii`, etc.

Lo mejor para indicar cómo se hace es dar un ejemplo. Si la lista inicial la quisiéramos numerar con números romanos en mayúscula seguidos de una flecha utilizaríamos

```
\begin{enumerate}\setlength{\itemsep}{-3pt}
 \renewcommand{\labelenumi}{\Roman{enumi}$\to$}
 \item Desayunar
 \item Ir a clase
 \item Aburrirme
\end{enumerate}
```

El resultado es:

I→ Desayunar
II→ Ir a clase
III→ Aburrirme

Los contadores en LaTeX se asignan por medio de `\setcounter`, lo cual permite comenzar la cuenta donde queramos. Por ejemplo, si en una lista anterior hubiéramos terminado en IV quizá quisiéramos comenzar la nueva en V. Para ello usaríamos en la primera línea

```
\begin{enumerate}\setcounter{enumi}{4}
```

obteniendo

V→ Desayunar
VI→ Ir a clase
VII→ Aburrirme

Si empleamos `\setcounter` antes de un `\item` modificaremos la cuenta a nuestra voluntad, por ejemplo, con `\setcounter{enumi}{20}` antes del segundo `\item` el resultado es:

V→ Desayunar
XXI→ Ir a clase
XXII→ Aburrirme

Por supuesto, es bastante dudoso que realmente deseemos incluir saltos de numeración en situaciones típicas.

[4] Un contador es una variable que almacena un número entero.

4. Ejemplos con paquetes especiales

Antes nos hemos referido al paquete `multirow` y habíamos mencionado brevemente cómo utilizar la nueva instrucción `\multirow` que provee. Veamos algunos ejemplos.

La tabla:

Características	
Color de pelo	Rubio
	Moreno
	Castaño
	Pelirrojo

se ha conseguido pegando cuatro celdas de la primera columna en una de anchura `100pt`. El código fuente completo es:

```
\begin{center}
\begin{tabular}{ |c|c| }
\multicolumn{2}{|c|}{\textbf{Características}}
\\ \hline
\multirow{4}{100pt}{Color de pelo} & Rubio  \\ \cline{2-2}
& Moreno \\  \cline{2-2}
& Castaño \\ \cline{2-2}
& Pelirrojo \\
\hline
\end{tabular}
\end{center}
```

Si reducimos el segundo argumento de `\multirow{4}{100pt}{...}` forzaremos a que el texto de la celda ocupe dos líneas. Por ejemplo, con `40pt` el resultado es:

Características	
Color de pelo	Rubio
	Moreno
	Castaño
	Pelirrojo

Por supuesto, no es obligatorio que las celdas pegadas lleguen hasta la última línea de la tabla. Así, añadiendo `\\ \hline A&B` tras `Pelirrojo` en la tabla original el resultado es

Características	
Color de pelo	Rubio
	Moreno
	Castaño
	Pelirrojo
A	B

Si uno quiere jugar con la posición vertical, existe la posibilidad de introducir en tercer lugar una longitud opcional entre corchetes indicando cuánto deseamos subir el texto. Por ejemplo, con

```
\multirow{4}{100pt}[10pt]{Color de pelo}
```

se obtiene:

Características	
Color de pelo	Rubio
	Moreno
	Castaño
	Pelirrojo
A	B

Los valores negativos bajan el texto.

Respecto a las listas, numeradas o no, es un poco incómodo que no se pueda controlar la distancia al margen. El paquete `enumitem` da la libertad para hacerlo así como de asignar otras longitudes de manera unificada. Si deseamos usarlo hay que escribir en la cabecera

```
\usepackage{enumitem}
```

Las longitudes que vamos a modificar en los siguientes ejemplos son `leftmargin` que indica la distancia del texto de los ítems al margen izquierdo, `topsep` que da la separación vertical de la lista con respecto al texto y `labelsep` e `itemsep` que ya habíamos visto. Todas se asignan como *nombre=longitud*, separados por comas si es necesario, dentro de corchetes como argumento del entorno. Por ejemplo

1. Desayunar
2. Ir a clase
3. Aburrirme

Se ha obtenido con

```
\begin{enumerate}[leftmargin=100pt]
\item Desayunar
\item Ir a clase
\item Aburrirme
\end{enumerate}
```

Los mismo funcionaría con `itemize`.

Quizá te hayas percatado de que tanto con las listas numeradas como en las no numeradas, hay un espacio de separación al principio y al final mayor

que un salto de línea. Si asignamos a `topsep` un valor negativo, acercaremos la lista al texto.

En el próximo ejemplo vamos a modificar todas las longitudes mencionadas con:

```
[leftmargin=30pt, topsep=-1pt, labelsep=20pt, itemsep=-2pt]
```

El resultado es:

1. Desayunar
2. Ir a clase
3. Aburrirme

Nota que aunque hemos puesto una distancia al margen de `30pt` los números están casi pegados a él. Esto se debe a que esta distancia se calcula hasta el texto, no hasta las etiquetas cuya separación en el ejemplo casi compensa esta distancia.

En `itemize` podemos incluir también en los argumentos una definición `label=` que cambie el símbolo usado como etiqueta. Por ejemplo, con

```
\begin{itemize}[label=$\square$., itemsep=-2pt]
```

en nuestro ejemplo, el resultado es:

- □. Desayunar
- □. Ir a clase
- □. Aburrirme

En cada sublista podemos definir de nuevo `label` de acuerdo con nuestros gustos.

También con `enumerate` es posible incluir como argumento una definición de `label`. Por ejemplo,

```
[label=\textbf{\Alph*}., leftmargin=2cm, itemsep=-2pt]
```

como argumento de `enumerate` en el ejemplo que venimos manejando daría:

A. Desayunar

B. Ir a clase

C. Aburrirme

El asterisco de `\Alph*` indica el contador en curso. Tendría el mismo efecto emplear en su lugar `\Alph{enumi}`.

Hay otras características de las listas que se pueden ajustar con el paquete `enumitem` y que están descritas en su documentación.