MEDEX 1

LaTeX medio

con ejercicios resueltos

David Pacios Izquierdo
Overleaf Advisor
Aficionado a LaTeX
dpacios@ucm.es

MEDEX LATEX medio con ejercicios resueltos

David Pacios Izquierdo

ISBN: 978-10-9610-599-2
Sello: Independently published

Edición: 1ª
Impresión: 1ª
Nº de páginas: 221
Formato: 8x10 pulgadas
Versión: Noir

MEDEX 1 Noir

Esta versión del libro de MEDEX1 está creada para ser visualizada en escala de grises, hay capítulos que usan colores y no se verán, pero el contenido sigue cumpliendo el objetivo.

"No nos ocurre nada
que no estemos preparados para soportar."
Máximo Décimo Meridio

Niveles de LaTeX

Se han creado distintos niveles de conocimiento de LaTeX para catalogar los conocimientos adquiridos por niveles de dicho lenguaje. Con el objetivo de poder estudiar mejor los conceptos.

BaSiX

Tener este nivel acredita el manejo básico de LaTeX su uso sin problemas para resolver problemas cotidianos, poder hacer tablas, insertar imágenes y manejo del modo matemático.

MeDiEX

Este nivel acredita el uso de plantillas básicas, creación y uso de gráficos, creación de entornos personalizados y de comandos avanzados.

AdViSOR

Nivel avanzado de LaTeX, acredita el manejo avanzado de paquetes profesionales como tikz, manejo de plantillas avanzadas y creación de las mismas, creación de documentos modificados por parámetros con soltura.

Índice general

Agradecimientos

Por supuesto a Sara, mi mujer. Por aguantarme durante la escritura de todos los libros del nivel medio de LaTeX, por continuar aquí un día más. Quiero también hacer especial mención a todas aquellas personas, por las cuales mi teléfono suena en todo momento, yo protejo aquello que importa. Por todos esos momentos que no puedo recordar y por esa persona que nunca podré recordar.

A la asociación Socio-Cultural de Ingenierías en Informática, que tantos enemigos ha combatido con éxito como el aburrimiento y la ignorancia, demostrando que se puede tener un equilibrio entre el ocio y la cultura. Por la junta y la familia, aquello que realmente importa y que se debe proteger a toda costa. A todos aquellos profesores, que han realizado con todo el cariño del mundo, todas esas actividades con nosotros. Es un buen momento para recordar a aquellos que luchan y trabajan sin esperar nada a cambio. Que estos agradecimientos sean vuestra justicia.

A vuestro trabajo, por la Oficina de Software Libre y Tecnologías Abiertas, que en el año 2018–2019 ha realizado de los mejores trabajos, siempre llenando, en toda la Universidad Complutense de Madrid. Contra viento y marea se ha demostrado que el trabajo es lo que más cuenta y al final es lo que hace huella en todos.

-Ante la crueldad, conocimiento. Mi Gnosis.-
David Pacios Izquierdo, de título **Pascal**. Tercero con ese nombre.

Prólogo

Lo reconozco, yo también soy culpable de saltarme los prólogos.

No obstante, te pediría que dedicaras unos segundos a leer la primera parte, aquella en la que hablo del genial autor de este libro, el cual he tenido el inmenso honor de prologar.

Sobre el autor (y lo que él representa)

Si estás leyendo MEDEX es porque en algún momento has leído BASIX y por tanto conoces a David (también conocido como Pascal), presidente de la Asociación Socio Cultural de Ingenierías en Informática (ASCII) en la Facultad de Informática de la Universidad Complutense de Madrid.

Si en el prólogo del anterior libro me centré en el autor como persona, esta vez me gustaría dedicar unas líneas a su figura y lo que representa.

David es un claro ejemplo del conjunto de estudiantes que colaboran de forma activa con la Oficina de Software Libre y Tecnologías Abiertas de la Universidad Complutense de Madrid.

Cada uno de ellos es maravillosamente único, pero todos comparten al menos una característica común: un increíble sentido de servicio a los demás, gracias al cual, existe la Oficina, que ha mantenido su esencia hasta el momento de escribir estas líneas.

He perdido la cuenta de las veces que digo diariamente que los estudiantes son el principal activo de nuestra sociedad, pero es que el trabajo realizado por ellos en las actividades de la Oficina no deja de darme la razón.

Dicho trabajo no se ha limitado a *estar en las actividades*. Los estudiantes han invertido un sinfín de horas en su preparación, y otro tanto en el análisis de su desarrollo para mejorarlas. Porque es precisamente el inconformismo otra de sus características comunes, un inconformismo que les permite detectar carencias en la sociedad y destinar grandes esfuerzos a aportar soluciones.

Sobre este libro (y lo también representa)

En tus manos tienes el siguiente peldaño en tu ascensión hacia la acrópolis del LaTeX, la cual sin duda se inició con BASIC.

Los aspectos que cubren este libro son de gran interés en diferentes ámbitos. A lo que el científico-tecnológico respecta, que es el más relacionado con mi actividad, considero que los capítulos dedicados a los entornos matemático y gráfico son fundamentales (lo dice un fervoroso usuario de *gnuplot*).

Por otro lado, Beamer es uno de esos grandes desconocidos que sin duda hará que ames más esta tecnología.

Pero es que además tienes en tus manos uno de los últimos hitos del curso académico 2018/2019 para la Oficina de Software Libre y Tecnologías Abiertas de la Universidad Complutense de Madrid. Y es que ha estado repleto de grandes momentos en forma de talleres sobre tecnologías libres en diferentes lugares del campus (los *Talleres que Molan*), sesiones de instalación de GNU/linux (el *Penguin on Tour*) o incluso un evento de ciberseguridad para todos los niveles abierto y gratuito (la ya famosa *CryptoParty Madrid*).

Este libro es una muestra más de las cosas increíbles que pueden realizar los estudiantes que colaboran con la Oficina y sus asociaciones, no perteneciendo todas a la Facultad de Informática (claro ejemplo de la transversalidad del Software Libre). Es así como, sin más preámbulos, te animo a que disfrutes del mismo y de lo que él representa.

Madrid, primavera de 2019

José Luis Vázquez Poletti
Profesor de la Facultad de Informática
Director de la Oficina de Software Libre y Tecnologías Abiertas
Universidad Complutense de Madrid

1

Introducción a MEDEX

1. Disclaimer (Notas del autor).
2. Sobre los conceptos de nivel básico de LaTeX.
3. Sobre los conceptos de nivel medio de LaTeX.

1.1. Disclaimer (Notas del autor)

Largo es el camino que he tomado para poder realizar este libro. En el libro de LaTeX básico, comenté la historia de como empecé a escribir en LaTeX. Ahora, en este libro de nivel medio, me gustaría continuar la historia de como adquirí un nivel un poco más avanzado de lo común de este lenguaje.
En el curso 2016 – 2017, en la Facultad de Informática de la Universidad Complutense de Madrid, comencé a colaborar activamente en una asociación de estudiantes, posiblemente la más activa y que tocaba todos los campos del ocio y conocimiento, la Asociación Socio-Cultural de Ingenierías en Informática (ASCII). Para su entonces yo comenzaba realizando actividades para la asociación, muchas de ellas de ocio y otras de conocimiento y cultura. Para entonces yo necesitaba de forma imperiosa aprender a escribir de forma correcta, ya que, muchas de las tareas requerían de documentación para que nos cedieran espacios o para que nos concedieran dinero, para ello me negaba a entregar documentos totalmente ilógicos sin sentido alguno y sin formato estándar. Cada documento, dossier o propuesta que entregábamos era distinto, con un formato distinto y creado con herramientas de ofimática horribles que destrozaban el documento cuando pasaba de 20 páginas. Para ello retomé mis conocimientos de LaTeX y pude en poco tiempo generar plantillas básicas para tener los documentos estandarizados.

Primero comencé a generar un documento para justificantes oficiales de participación en la asociación. Lo probé haciendo un justificante de lepra, actualmente varios profesores tienen justificantes de esos. Después realicé plantillas más complejas adquiriendo más conocimientos. Actualmente las plantillas avanzadas de TFG–TFM cumplen varios estándares internacionales y la ISO 690:2010. Durante estos 3 años he aprendido cada día LaTeX para poder realizar los documentos más avanzados, entre los conocimientos adquiridos aprendí a usar bien los compiladores avanzados de XeLaTeX y LuaLaTeX que difieren un poco de los usados por defecto (PdfLaTeX). En la primera edición básica de LaTeX no quise incluir nada sobre idiomas y compiladores ya que lo vi muy extenso como para incorporarlo, en este libro de nivel medio tenemos una gran mención a ambos compiladores. Si has llegado hasta aquí y quieres aprender el nivel medio de LaTeX necesitas saber varias cosas antes.

En este libro recopilo el material adecuado para aprender un nivel medio de LaTeX, esto no significa que no haya mucho más material. El nivel avanzado es prácticamente hablar del uso de TikZ, uno de los paquetes más avanzados de LaTeX y de otros paquetes y lenguajes más avanzados como LuaLaTeX.

Para proceder al uso de este libro, es necesario el conocimiento de los recursos básicos de LaTeX relatados en el libro de conocimientos BASIX También voy a hacer una especial mención a la condición del escritor de este libro en el momento de crearlo. Es posible que haya erratas que se irán corrigiendo con el tiempo. Para este libro no hay editor, ni revisor ni voluntarios para una edición beta con objeto de corregir los fallos. El autor ha escrito

este libro solo y con la salud en contra. Sed indulgentes a la hora de encontrar fallos, el código fuente estará disponible para que podáis hacer cualquier cambio de forma gratuita siempre.

1.2. Sobre los conceptos de nivel básico de LATEX

Estos conceptos se han aprendido en el libro de BASIX Son los justos

Conceptos del nivel básico

1. Introducción básica de documentos en LATEX.
2. Sintaxis básica con modificadores.
3. Estructuras de contenido de enumerados, itemizes y separaciones.
4. Estructuración compleja y modificadores.
5. Símbolos matemáticos básicos.
6. Estructuras matemáticas.
7. Bibliografía y referencias.

Al terminar el curso, el alumno tendrá los conocimientos básicos del LATEX

1.3. Sobre los conceptos de nivel medio de LaTeX

Es importante recalcar que el nivel medio de LaTeX se compone de **3 tomos de conocimientos medios.** MEDeX 1, MEDeX 2, MEDeX 3. Juntos forman el conocimiento completo de nivel medio de LaTeX.

NOTAS DE MEDeX1

Para continuar con el curso de MEDeX2 serán necesario todos los conocimientos impartidos en este tomo.

Conceptos de MEDeX1

1. Distinción de compiladores, fuentes y multi-idioma.
2. Lua básico.
3. LuaLaTeX básico.
4. Inserciones, inputs e incorporaciones externas.
5. Powerdot y Beamer.
6. Creación de comandos y ambientes básicos.
7. Creación de macros con argumentos.
8. Condicionales básicos.
9. Entorno matemático avanzado.
10. Primeros pasos en entorno gráfico y Tikz básico.

Al terminar el curso, el alumno podrá continuar con los conocimientos de MEDeX2.

2

Compiladores, fuentes e idiomas

En este capítulo vamos a ver los distintos compiladores de LaTeX con sus usos, vamos a explicar el compilador XeLaTeX, usaremos varios idiomas en un mismo documento, pondremos ejemplos de idiomas en distintos documentos, aprenderemos a escribir en idiomas que empiezan a escribir por la derecha, aprenderemos a insertar nuestras propias fuentes en nuestro documento, aprenderemos a usarlas y finalmente, realizaremos ejercicios para practicar los conceptos anteriores.

1. Compiladores de LaTeX y distintos usos.
2. XeLaTeX.
3. Uso de varios idiomas en el mismo documento.
4. Ejemplos de idiomas.
5. Uso de texto con comienzo en borde derecho.
6. Inserción de una fuente personalizada.
7. Uso de las fuentes.
8. Ejercicios resueltos.

2.1. Compiladores de LaTeX y distintos usos.

El lenguaje de LaTeX se escribe a través de unas macros escritas para TeX, por lo que presentamos dos lenguajes distintos que se pueden compilar por distintos compiladores. Entre estos compiladores podemos encontrar los siguientes:

- LaTeX: Compila los dos lenguajes vistos anteriormente a dvi.
- pdfLaTeX: Compila los dos lenguajes a pdf.
- XeLaTeX: Compila a pdf pero con gestión Unicode, y nos permite usar las fuentes del sistema sin necesidad de configurar nada. En este capítulo se explicará XeLaTeX.
- LuaLaTeX: Compila a pdf los dos lenguajes y está escrito en Lua, que es un tipo de lenguaje de programación.

2.1.1. LaTeX

Utiliza la compilación a través de dvi (device independent file), que consiste en una serie de datos binarios describiendo de una manera visual el documento, lo dispone de un formato de impresión.
Soporta formatos de imagen como .eps y .ps con el comando `\includegraphics`.

2.1.2. pdfLaTeX

El pdf (Portable Document Format) es un tipo de documento, basado en el PostScript, usado para representar los documentos de una manera independiente al software, hardware y sistema operativo.
Con este compilador, todos nuestros documentos serán compilados a pdf.
Utiliza como formato de imágenes el .png y el .jpg para compilarlos a pdf durante la compilación.

2.1.3. LuaLaTeX

Es el sucesor de pdfLaTeX que incluye características como: generación directa de los pdf que soporta de una manera avanzada las mejoras micro-tipográficas del algoritmo tipográfico Tex. Sus características principales son:

- Soporte nativo del Unicode, soporta todos los caracteres del mundo, desde el inglés hasta el chino tradicional incluyendo el árabe y además de todos los símbolos matemáticos.
- Inclusión de lua como un lenguaje de script incrustado.
- Muchas librerías que incluyen:

- fontloader: Soporta modernos formatos de fuentes como TrueType y OpenType.
- font: Permite modificar las fuentes de manera avanzada sin depender del documento.
- mplib: Una versión integrada del programa gráfico MetaPost.
- callback: Provee enlaces en partes del motor de Tex que antes eran inaccesibles para el programador.
- Útiles librerías para manipular imágenes, documentos pdf, etc.

Cambiando de LaTeX a LuaLaTeX

En algunos paquetes y herramientas es similar al LaTeX, pero tiene tres diferencias como:

- No carga el inputenc, sino que lo codifica en UTF-8.
- No carga el fontenc o el textcomp, sino que carga el fontspec.
- El paquete de idiomas es cargado a través de polyglossia.
- No se utilizan paquetes que cambian las fuentes, sino que se utilizan comandos fontspec.

Ahora nos vamos a familiariciar con el comando fontspec, para ello, seleccionamos nuestra fuente principal con el comando `\setmainfont`, la fuente serif con el comando `\setsansfont` y la fuente mono-espaciada con el comando `\setmonofont`.
El paquete luainputenc provee varias ventajas y desventajas que le permiten recuperar estas dos posibilidades a expensas de perder el real soporte del Unicode.
Para saber como utilizar las fuentes en LuaLaTeX recomendamos utilizar manuales y compilar pequeñas fuentes en distintos documentos.

Paquetes esenciales

Podemos encontrar los distintos paquetes fontspec en el siguiente enlace: `https://github.com/wspr/fontspec/`.
Y también podemos encontrar los distintos paquetes de idiomas polyglossia en el siguiente enlace: `https://github.com/reutenauer/polyglossia/`.

2.1.4. Cambiando de compiladores en Overleaf

En nuestro documento de Overleaf podemos cambiar en los distintos compiladores. Para ello lo haremos de la siguiente manera:

1. Seleccionaremos menú en la parte izquierda de nuestro documento.
2. Buscamos el compilador en la parte izquierda del menú.

3. Seleccionamos nuestro compilador de preferencia.

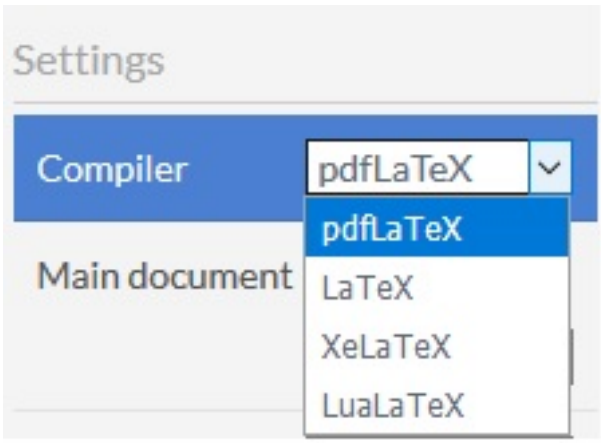


Figura 2.1: Seleccionar nuevo compilador

2.2. XeLaTeX

Con este compilador podremos cambiar entre múltiples idiomas, para ello, cargaremos el paquete polyglossia. Este paquete es exactamente igual que el babel en LaTeX, pero el paquete de XeLaTeX maneja directamente el UTF8. Para ver cómo manejamos los distintos paquetes de idiomas en este compilador, vamos a poner un ejemplo:

Figura 2.2: Ejemplo de XeLaTeX

Además de cambiar el idioma o poner varios idiomas, podremos cambiar la fuente en el preámbulo con el paquete fontspec y con el comando `\setmainfont{}`, y entre corchetes pondremos la fuente que queramos utilizar. Para tenerlo más claro vamos a poner un ejemplo:

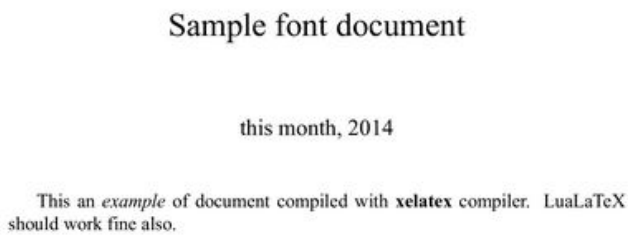


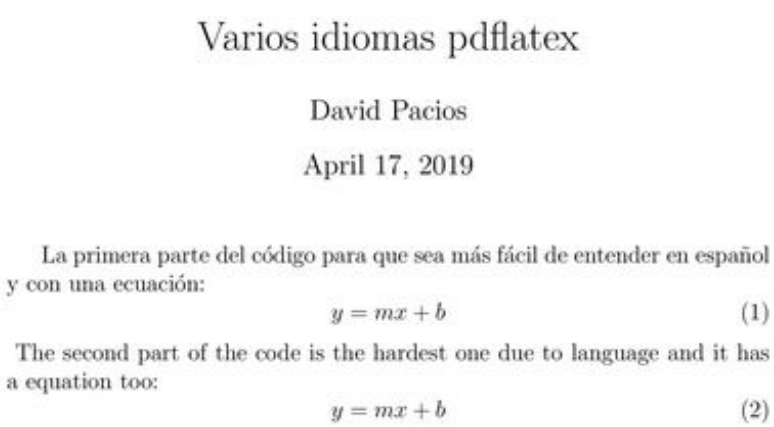
Figura 2.3: Ejemplo de fuente de XeLaTeX

También, como en el caso de los idiomas, podemos utilizar más de un tipo de letra en nuestro documento, para ello, utilizaremos los siguientes comandos:

- `\setmainfont{Times New Roman}`: La fuente Times New Roman.
- `\setsansfont{Arial}`: La fuente Arial.
- `\setmonofont{Courier New}`: La fuente Courier New.

2.3. Uso de varios idiomas en el mismo documento

Escribir varios idiomas en un documento en LaTeX puede ser complicado en un principio, pero con los distintos comandos y compiladores que vamos a ver esta tarea se nos puede hacer más sencilla.
Primero vamos a empezar con el compilador pdfLaTeX, que es el más básico. Para añadir más de un idioma en este caso, sólo tendremos que añadir un idioma más en el preámbulo del paquete babel y seleccionar el idioma que queramos en el documento con el comando `\setlanguage{Idioma}`. Para tenerlo más claro vamos a poner un ejemplo:


Varios idiomas pdflatex

David Pacios

April 17, 2019

La primera parte del código para que sea más fácil de entender en español y con una ecuación:

$$y = mx + b \quad (1)$$

The second part of the code is the hardest one due to language and it has a equation too:

$$y = mx + b \quad (2)$$

Figura 2.4: Dos idiomas en pdfLaTeX

Código 2.1: Idiomas en pdflatex

```
\documentclass[a4paper,openright,12pt]{article}
\usepackage[spanish,USenglish]{babel} %Espa ol, ingl s
\usepackage[utf8]{inputenc}

\title{Varios idiomas pdflatex}
\author{David Pacios}


\begin{document}
\maketitle
\selectlanguage{spanish}
La primera parte del código para que sea más fácil de entender en
   español y con una ecuación:
\begin{equation}
    y=mx+b
\end{equation}
\selectlanguage{USenglish}
The second part of the code is the hardest one due to language and
   it has an equation too:
\begin{equation}
    y=mx+b
\end{equation}
\end{document}
```

Finalmente, por otro lado, tenemos el compilador XeLaTeX que tiene su propio paquete de idiomas, que es el polylossia, y seleccionaremos los idiomas con los comandos `\setmainlanguage{}` y `\setotherlanguage{}`. Para tenerlo más claro vamos a poner un ejemplo:

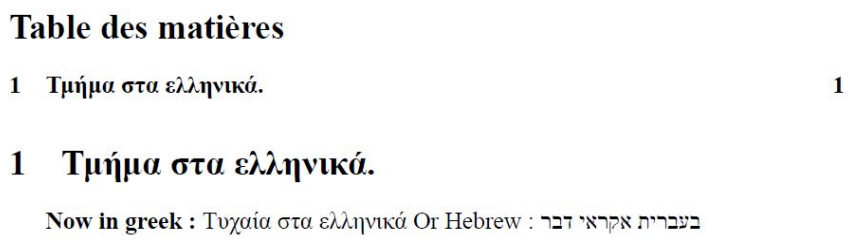


Figura 2.5: Varios idiomas con XeLaTeX

2.4. Ejemplos de idiomas

Al igual que en los casos anteriores de los idiomas, el Braile necesita tener instalado su paquete en el preámbulo. Para ello usamos `\usepackage[puttinydots, useemptybox]{braille}` en el preámbulo. Y en el documento, escribiremos el comando `\braille{}` con lo que queramos escribir en braile entre corchetes. Para tenerlo más claro vamos a poner un ejemplo:

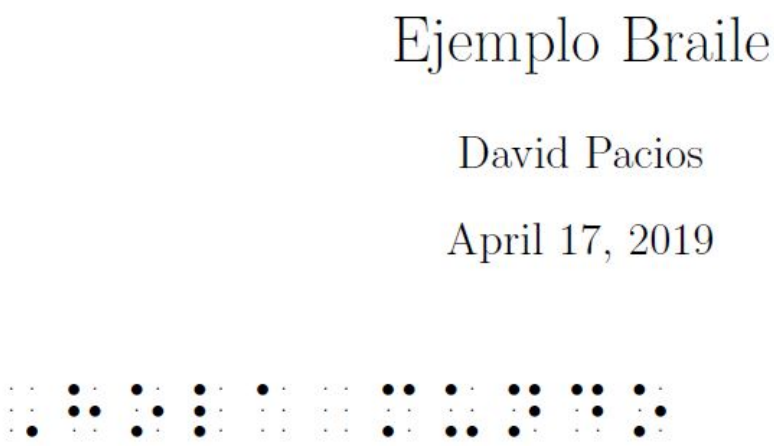


Figura 2.6: Hola mundo en Braile

Código 2.2: Hola mundo en Braile

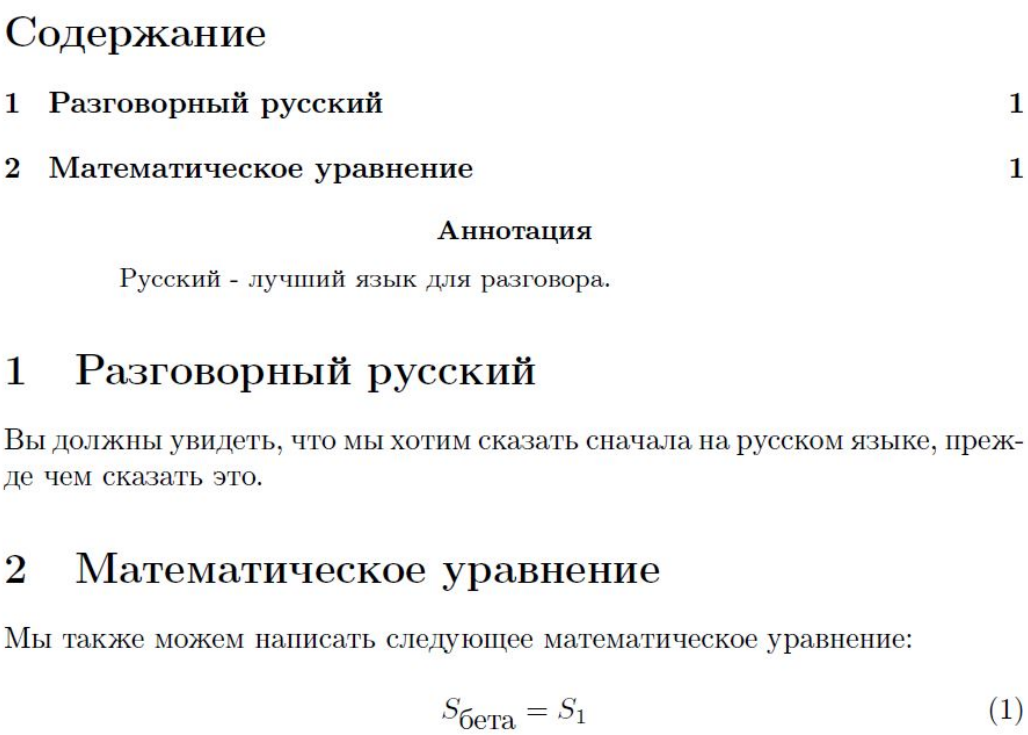
```
1  \documentclass{article}
2  \usepackage[utf8]{inputenc}
3  \usepackage[puttinydots, useemptybox]{braille}
4
5  \title{Ejemplo Braile}
6  \author{David Pacios}
7
8
9  \begin{document}
10 \maketitle
11 \braille{Hola mundo}
12 \end{document}
```

Además de poder escribir en Braile, también podremos escribir en cirílico. Para ello podremos utilizar varias vías, la primera sería con el compilador pdfLaTeX y la segunda sería con

XeLaTeX.
Si queremos utilizar el compilador pdfLaTeX requerirá que utilicemos el paquete babel y el tipo de fuente de escritura con el paquete fontenc. En nuestro preámbulo escribiremos lo siguiente:

- `\usepackage[T2A]{fontenc}`.
- `\usepackage[utf8]{inputenc}`.
- `\usepackage[russian]{babel}`.
- `\usepackage{hyphenat}`.

Una vez hemos visto qué tenemos que escribir en nuestro preámbulo vamos a ver un ejemplo de un texto en cirílico:



Содержание

1 Разговорный русский 1

2 Математическое уравнение 1

Аннотация

Русский - лучший язык для разговора.

1 Разговорный русский

Вы должны увидеть, что мы хотим сказать сначала на русском языке, прежде чем сказать это.

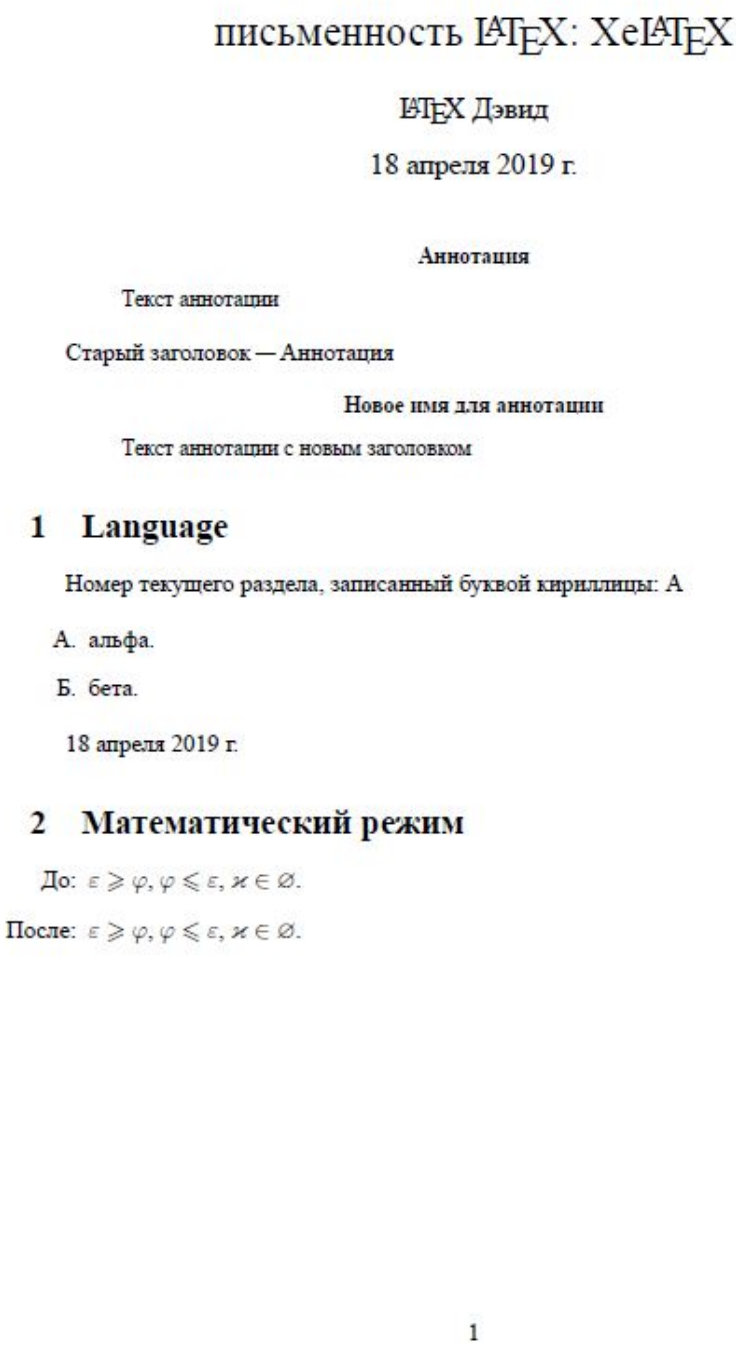
2 Математическое уравнение

Мы также можем написать следующее математическое уравнение:

$$S_{\text{бета}} = S_1 \tag{1}$$

Figura 2.7: Ejemplo de texto en cirílico en pdfLaTeX

Por otro lado, si utilizamos el compilador XeLaTeX, sólo tendremos que poner en nuestro preámbulo el comando `\usepackage[russian]{polyglossia}`. Y vamos a ver otro ejemplo de un texto en cirílico:



письменность LaTeX: XeLaTeX

LaTeX Дэвид

18 апреля 2019 г.

Аннотация

Текст аннотации

Старый заголовок — Аннотация

Новое имя для аннотации

Текст аннотации с новым заголовком

1 Language

Номер текущего раздела, записанный буквой кириллицы: А

А. альфа.

Б. бета.

18 апреля 2019 г.

2 Математический режим

До: $\varepsilon \geqslant \varphi, \varphi \leqslant \varepsilon, \varkappa \in \varnothing.$

После: $\varepsilon \geqslant \varphi, \varphi \leqslant \varepsilon, \varkappa \in \varnothing.$

1

Figura 2.8: Ejemplo de texto en cirílico en XeLaTeX

Otro idioma que podemos escribir en LaTeX es el coreano. Lo podemos compilar tanto en pdfLaTeX como en XeLaTeX. Primero vamos a ver qué tenemos que poner en nuestro preámbulo en XeLaTeX:

- `\usepackage{xeCJK}`: El paquete xeCJK para compilar el coreano.
- `\setCJKmainfont{UnGungseo.ttf}`: El paquete de la fuente de texto principal.
- `\setCJKsansfont{Ungungseo.ttf}`: El paquete de la fuente sans.
- `\setCJKmonofont`: El paquete de la fuente para el entorno verbatim.

Seguidamente, vamos a ver cómo compilar en coreano en el pdfLaTeX. Para ello tenemos que escribir lo siguiente en el preámbulo:

- El paquete CJKuft8 codifica el coreano.
- El entorno del coreano utilizado sería el CJK.

Para tenerlo más claro vamos a ver un ejemplo:

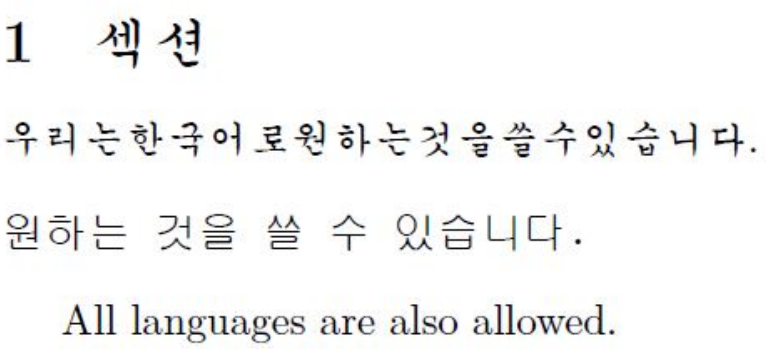


Figura 2.9: Coreano en pdfLaTeX

2.5. Uso de texto con comienzo en borde derecho

LaTeX es capaz de compilar lenguajes a lo largo del mundo, como ya hemos visto en capítulos anteriores, si somos capaces de compilar todos los paquetes de idiomas. En este caso, vamos a utilizar el compilador pdfLaTeX.
Uno de los ejemplos de los idiomas que comienza a la derecha es el árabe. Para ello, vamos a escribir los siguientes paquetes en el preámbulo:

- `\usepackage{arabtex}`: El paquete del idioma árabe.
- `\usepackage[LFE,LAE]{fontenc}`: El paquete para poder escribir de derecha a izquierda y de izquierda a derecha. También nos permite usar la escritura en árabe.
- `\usepackage[arabic]{babel}`: El paquete babel del árabe, además también nos permite introducir textos en latín.

Además, si queremos escribir alguna lengua derivada del latín utilizaremos el comando `\textLR{}`, y entre corchetes, podremos el texto de una lengua derivada del latín. Para tenerlo más claro, vamos a poner un ejemplo de un documento en árabe:

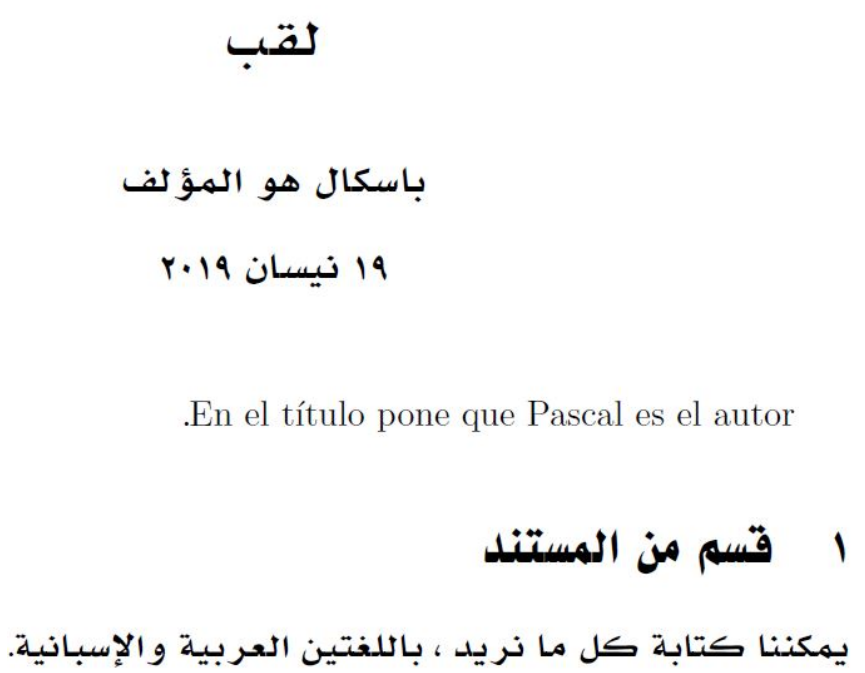


Figura 2.10: Ejemplo del texto en árabe

También podemos escribir en otros idiomas que comienzan en el borde derecho, como por ejemplo, el griego, el chino simplificado y el japonés. Vamos a empezar explicando lo qué tenemos que poner en el preámbulo en el caso que queramos escribir un texto en griego:

- `\usepackage[greek]{babel}`: El paquete babel para el griego.
- `\usepackage{alphabeta}`: Permite introducir los caracteres griegos en las ecuaciones matemáticas.
- `\usepackage[LGR, L1]{fontenc}`: Permite codificar las fuentes en griego a través del comando LGR.
- `\textlatin{}`: Nos permite poner un texto en latín en pequeñas anotaciones.

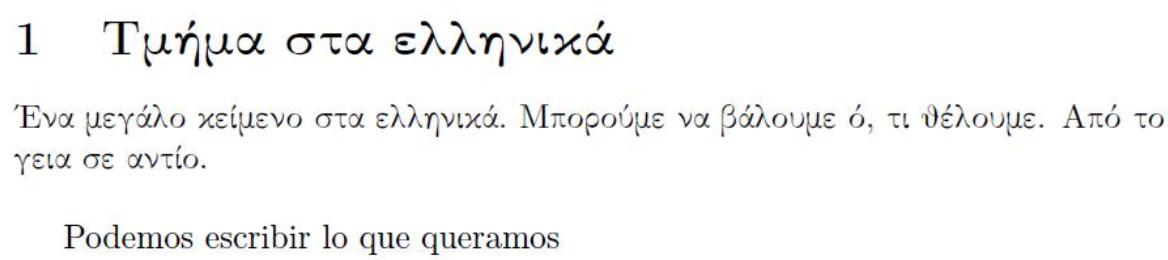


Figura 2.11: Ejemplo del texto en griego

Seguidamente, vamos a proceder a explicar los textos en chino. Para el chino se recomienda utilizar los compiladores XeLaTeX o LuaLaTeX, ya que soportan el UTF-8 directamente. Primero, para realizarlo en XeLaTeX sólo tendremos que poner en nuestro preámbulo el paquete xeCJK, para ello, sólo tendremos que utilizar el comando `\usepackage{xeCJK}`. Y por último, vamos a ver qué tenemos que poner en el preámbulo en caso de utilizar el compilador pdfLaTeX. En nuestro preámbulo sólo tendremos que utilizar el comando `\usepackage{CJKutf8}`, y para comenzar el entorno en chino empezaremos con el comando `\begin{CJK*}` y lo cerraremos con el entorno `\end{CJK*}`. Una vez que utilicemos cualquiera de los dos compiladores nos quedará un texto de la siguiente forma:

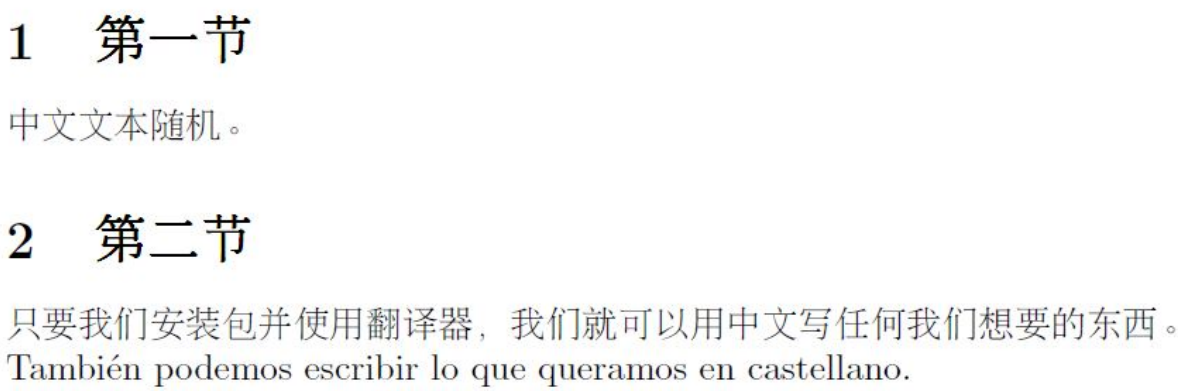


Figura 2.12: Ejemplo del texto en chino

Y finalmente, el japonés y el chino utilizan el mismo paquete de idiomas tanto para pdfLaTeX como XeLaTeX. Por lo que es importante que utilicemos el paquete xeCJK en nuestro documento. Vamos a ver un ejemplo de un texto en japonés:

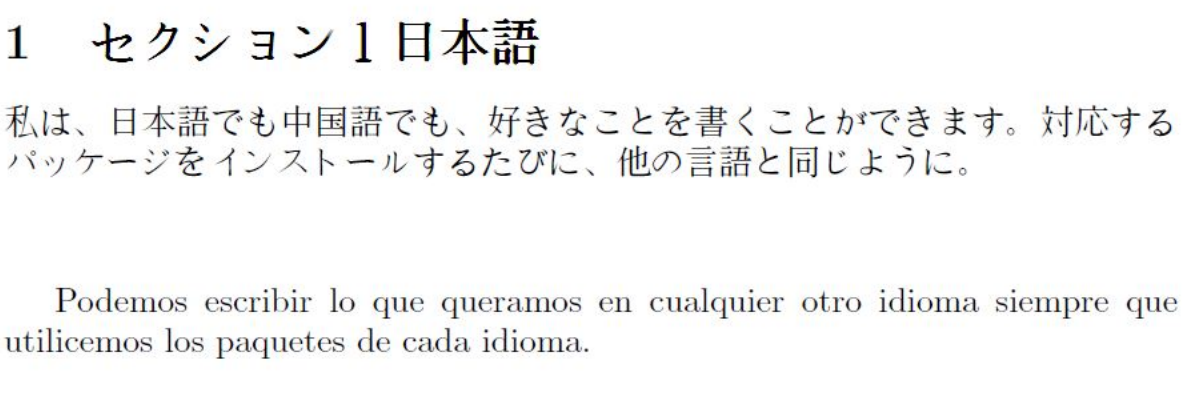


Figura 2.13: Ejemplo del texto en japonés

2.6. Inserción de una fuente personalizada

Como hemos visto en capítulos anteriores, podemos personalizar la fuente utilizada en nuestro documento, pero siempre que utilicemos el compilador XeLaTeX. Para ello, en el preámbulo de nuestro documento el comando el paquete fontspec para indicar que queremos hacer un cambio de fuente, y una vez lo hemos hecho, en el mismo preámbulo indicaremos que queremos cambiar la fuente principal con el comando `\setromanfont[]{}` y entre corchetes, colocaremos la fuente que nos hemos descargado previamente de Google Fonts. Y si queremos poner las siguientes fuentes utilizaremos los comandos `\setsansfont[]{}` y `\setmonofont[]{}`.
Ahora vamos a ver un código de ejemplo de un texto con las fuentes cambiadas:

Código 2.3: Código texto cambiado

```
\documentclass[12pt]{article}
\usepackage{fontspec}
%Es importante el compilador XeLatex
%Se pueden poner las fuentes desde el ordenador directamente o
   agregar al paquete de textos de la siguiente forma:
% Roboto
\setromanfont[
BoldFont=Roboto-Black.ttf,
ItalicFont=Roboto-Bold.ttf,
BoldItalicFont=Roboto-BoldItalic.ttf,
]{Roboto-Black.ttf}
% Sans Pro
\setsansfont[
```

```
BoldFont=SourceSansPro-Black.ttf,
ItalicFont=SourceSansPro-Italic.ttf,
BoldItalicFont=SourceSansPro-BoldItalic.ttf
]{SourceSansPro-Black.ttf}
% Courgette
\setmonofont[Scale=0.90,
BoldFont=Courgette-Regular.ttf,
ItalicFont=Courgette-Regular.ttf,
BoldItalicFont=Courgette-Regular.ttf,
Color={0019D4}
]{Courgette-Regular.ttf}
%-------------------------------------------------------

\title{Ejemplo de documento con fuente cambiada}
\author{El autor del libro}
\date{\today}

\begin{document}

\maketitle
Aquí ponemos un texto de ejemplo con el texto cambiado.


{\sffamily Esto es un ejemplo en \textbf{Source Sans Pro}}
\end{document}
```

Y con este código veremos el siguiente texto:

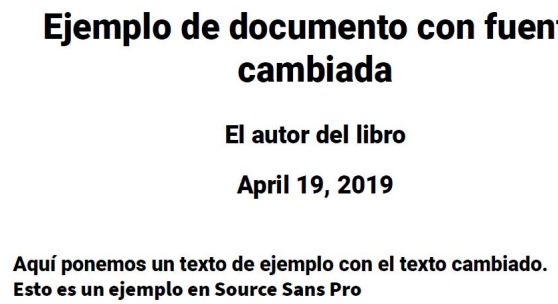


Figura 2.14: Resultado fuente cambiada

Es muy importante antes de cambiar la fuente, haberla instalado antes en el ordenador o haberla subido a nuestro documento en Overleaf.

Además se puede tener en una carpeta distinta en nuestro documento a compilar.

2.7. Uso de las fuentes

Como hemos visto anteriormente, LaTeX utiliza una serie de comandos en el preámbulo para determinar la fuente del texto que estamos escribiendo y sus acentos. Por defecto, LaTeX utilizará en el preámbulo el paquete fontenc, y lo instalaremos como paquete mediante el comando `\usepackage[T1]{fontenc}`. Una vez instalado, si queremos añadir acentos instalaremos el comando `\hyphenation{}` y entre corchetes, podremos señalar los símbolos acentuados.
Por otro lado, en estos paquetes no están incluidos símbolos como £ ó ð, los cuales tendremos que cargar mediante los paquetes `\usepackage{ae}` ó `\usepackage{aecompl}` en el preámbulo de nuestro documento. Si queremos realizarlo en una presentación con diapositivas instalaremos el comando `\usepackage[slides]{ae}` en nuestro preámbulo.
Seguidamente, vamos a ver qué tipo de letra tiene cada paquete, por ejemplo, el paquete bookman carga las fuentes Bookman, Avant Garde y Courier como tipografía. Dentro de LaTeX podemos encontrar estos tipos de fuentes:

- Romano: Palatino, New Century Schoolbook, Bookman y Times.
- Sans serif: Helvetica y Avant Garde.
- Mono-espaciada: Courier.
- Itálica: Zapf Chancery.
- Símbolos: Zapf Dingbats.

Podremos seleccionar cualquiera de este tipo de fuentes mediante el comando `\fontfamily{Tipo}` `\selectfont` en nuestro preámbulo. Entre los tipos de fuentes con sus respectivas variantes podemos encontrar las siguientes:

Cuadro 2.1: Tipo de fuente con sus variantes

Tipo	Series	Variantes	Nombre de la fuente
ptm	m,b	n,sl,it,sc	Adobe Times
ppl	m,b	n,sl,it,sc	Adobe Palatino
pnc	m,b	n,sl,it,sc	Adobe New Century Schoolbook
pbk	m,b	n,sl,it,sc	Adobe Bookman
phv	m,b,mc,bc	n,sl,sc	Adobe Helvetica
pag	m,b	n,sl,sc	Adobe Avant Garde
pcr	m,b	n,sl,sc	Adobe Courier
pzc	m	it	Zapf Chancery
pzd	m	n	Zapf Dingbats

Para cambiar sólo una única parte de la fuente de un documento utilizaremos el comando `\DeclareFixedFont`.
Como hemos visto anteriormente, hay un paquete exclusivo para instalar los símbolos, ese paquete es el pifont. Para instalarlo, escribiremos el comando `\usepackage{pifont}` en nuestro preámbulo, y lo nombraremos en nuestro documento mediante el comando `\ding{}`, y entre corchetes, indicaremos el número de nuestro símbolo a utilizar. Entre los símbolos que podremos colocar son los siguientes:

Cuadro 2.2: Simbolos con su número

	0	1	2	3	4	5	6	7
32		✁	✂	✃	✄	☎	✆	✇
40	✈	✉	☛	☞	✌	✍	✎	✏
48	✐	✑	✒	✓	✔	✕	✖	✗
56	✘	✙	✚	✛	✜	✝	✞	✟
64	✠	✡	✢	✣	✤	✥	✦	✧
72	★	☆	✪	✫	✬	✭	✮	✯
80	✰	✱	✲	✳	✴	✵	✶	✷
88	✸	✹	✺	✻	✼	✽	✾	✿
96	❀	❁	❂	❃	❄	❅	❆	❇
104	❈	❉	❊	❋	●	❍	■	❑
112	❒	❑	❒	▲	▼	◆	❖	◗
120	❘	❙	❚	❛	❜	❝	❞	
160		❡	❢	❣	❤	❥	❦	❧
168	♣	♦	♥	♠	①	②	③	④
176	⑤	⑥	⑦	⑧	⑨	⑩	❶	❷
184	❸	❹	❺	❻	❼	❽	❾	❿
192	①	②	③	④	⑤	⑥	⑦	⑧
200	⑨	⑩	❶	❷	❸	❹	❺	❻
208	❼	❽	❾	❿	➔	→	↔	↕
216	➘	➙	➚	➛	➜	➝	➞	➟
224	➠	➡	➢	➢	➤	➥	➦	➧
232	➨	➩	➪	➫	➬	➭	➮	➯
240		➱	➲	➳	➴	➵	➶	➷
248	➸	➹	➺	➻	➼	➽	➾	

Para poner cada símbolo solo tendremos que sumar la fila al valor de cada columna, por ejemplo, si quisiéramos poner el número uno con el círculo negro tendríamos que poner en nuestro comando `\ding{84}` y si quisiéramos poner el tick, tendríamos que poner el comando `\ding{51}`. Estos símbolos se pueden utilizar tanto en entornos itemize como en entornos enumerate.
Como hemos mencionado antes, hay muchos tipos de letras, y como tal, se pueden nombrar

por comandos, tanto en el preámbulo como en el cuerpo del texto:

- Principal: Se controla con `\rmfamily` en el preámbulo y el cuerpo con el comando `\textrm{}`.
- Sans serif: Se controla con `\sffamily` en el preámbulo y el cuerpo con el comando `\textsf{}`.
- Mono-espaciada: Se controla con `\ttfamily` en el preámbulo y el cuerpo con el comando `\texttt{}`.

2.8. Ejercicios resueltos

Ejercicio 1. Realiza un artículo con un texto en Braile.

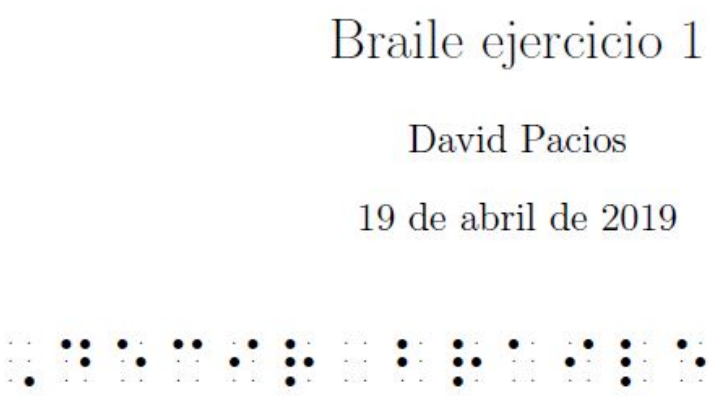


Figura 2.15: Ejercicio 1

Código 2.4: Ejercicio 1

```
\documentclass{article}
\usepackage[utf8]{inputenc}
\usepackage[spanish]{babel}
\usepackage[puttinydots, useemptybox]{braille}

\title{Braile ejercicio 1}
\author{David Pacios}
```

```
\date{\today}

\begin{document}
\maketitle
\braille{Decir braile}
\end{document}
```

Ejercicio 2. Busca una plantilla de texto en chino, japonés o en árabe. Mira su código e intenta modificar algo de la plantilla.


日本語のタイトル

パスカル

April 19, 2019

Abstract

ランダムテキスト

1 Inicio

他のランダムなテキスト

Figura 2.16: Ejemplo de plantilla japonés modificado

IMPORTANTE: Se pueden encontrar plantillas en la dirección `https://www.overleaf.com/latex/templates/japanese-latex-template/jjrfzbwkyyjs`, en el caso de que queramoos modificar la plantilla en japonés.

Ejercicio 3. Cambia las tres fuentes principales de un documento e indica cuáles son por comentarios.

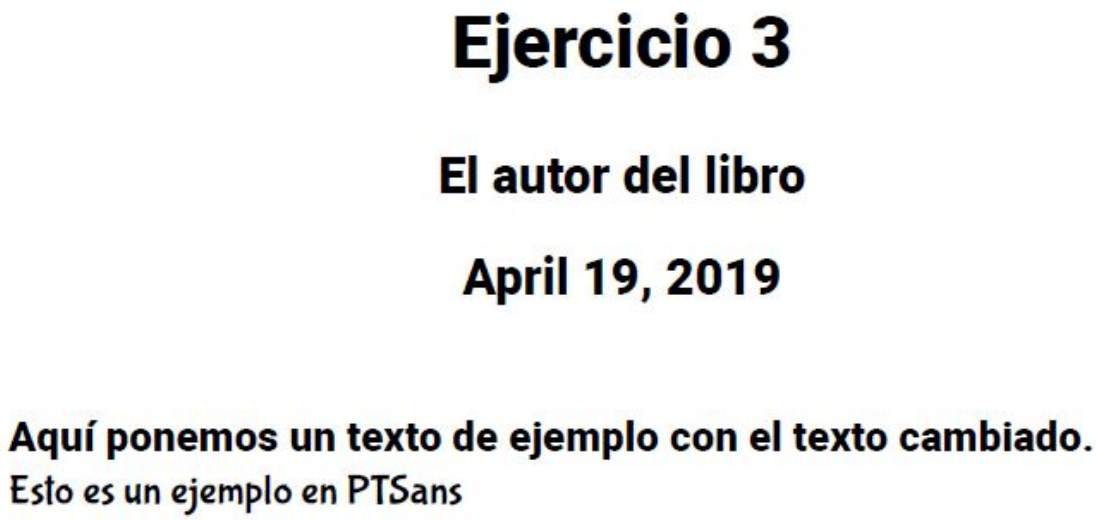


Figura 2.17: Ejercicio 3

Código 2.5: Ejercicio 3

```
\documentclass[12pt]{article}
\usepackage{fontspec}
%Se pueden poner las fuentes desde el ordenador directamente o
   agregar al paquete de textos de la siguiente forma:
% Roboto
\setromanfont[
BoldFont=Roboto-Black.ttf,
ItalicFont=Roboto-Bold.ttf,
BoldItalicFont=Roboto-BoldItalic.ttf,
]{Roboto-Black.ttf}
% Bubble Gum s lo tiene un tipo de fuente que se aplica a todas
\setsansfont[
BoldFont=BubblegumSans-Regular.ttf,
ItalicFont=BubblegumSans-Regular.ttf,
BoldItalicFont=BubblegumSans-Regular.ttf
]{BubblegumSans-Regular.ttf}
% Esta sans tiene dos tipos
```

```
\setmonofont[Scale=0.90,
BoldFont=PTSans-Bold.ttf,
ItalicFont=PTSansRegular.ttf,
BoldItalicFont=PTSansRegular,
Color={0019D4}
]{PTSansRegular.ttf}
%-----------------------------------------------------------------

\title{Ejercicio 3}
\author{El autor del libro}
\date{\today}

\begin{document}

\maketitle
Aquí ponemos un texto de ejemplo con el texto cambiado.


{\sffamily Esto es un ejemplo en \textbf{PTSans}}
```

3

LuaLaTeX: Introducción a Lua

En este capítulo vamos a aprender que es Lua, las variables, los arrays, las funciones y las operaciones que utiliza. Y finalmente, también aprenderemos las estructuras de control y algoritmos.

1. ¿Por qué Lua?
2. Lua: Variables.
3. Lua: Arrays.
4. Lua: Funciones.
5. Lua: Operaciones.
6. Lua: Estructuras de Control y Algoritmos.

3.1. ¿Por qué Lua?

Lua es un lenguaje de programación de tipo scripting, con una interfaz mucho más sencilla que C y que es más sencilla a la hora de programar por lo siguiente:

- Sintaxis agrupada en módulos.
- Todo viene inicializado.
- Sólo requiere texto plano para inicializarlo.
- Si hay un error de sintaxis se muestra.

Para programar en Lua necesitaremos:

- Un script en texto plano tipo .lua.
- Un procesador de texto plano tipo LaTeX.

3.2. Lua: Variables

Una variable es un espacio de memoria al que se accede mediante un nombre que se le asignado previamente en dicho espacio. Vamos a ver dos ejemplos de variables:

Código 3.1: Definición de variables

```
1 nombre= ''David''
2 nota= 8
```

Con una de estas dos variables hago referencia a la nota, que me da un valor de 8, y la variable nombre que contiene el valor de "David" almacenado previamente.
Es importante destacar algunos puntos respecto a las variables:

- Los nombres de las variables deben comenzar por alguna letra o subguión `\_`.
- Los nombres de las variables pueden contener solo letras, números y subguiones.
- Se diferencia entre mayúsculas y minúsculas.

3.2.1. Tipo de variables

Las variables pueden variar según el tipo de dato que contengan y son las siguientes:

- Número.
- String (cadena de texto).
- Boolean (Booleanas).
- Nil (Nulo).

Número

Variable de contenido numérico, no importa si la base es decimal, hexadecimal, binario, etc. Vamos a poner un ejemplo de este tipo de variable:

Código 3.2: Variables numéricas

```
anio= 2012
nota= 8
rojo= 0xFF0000
```

Las variables año y nota están en base 10 y la roja en hexadecimal.

String (cadena de texto)

Es cualquier tipo de variable que contenga una cadena de texto. Vamos a poner ejemplos de variables tipo string:

Código 3.3: Variables string

```
nombre= ''David''
cifra1= '1'
letra= 'a'
anio= ''2012''
```

Las cuatro variables son tipo string porque las comillas indican cadenas.

Boolean (booleanas)

Este tipo de variables sólo pueden tener dos valores, true (verdadero) y false (falso). Esto da lugar a una operación lógica con variables booleanas. Vamos a poner un ejemplo:

Código 3.4: Variables booleanas

```
var1= true
var2= false
var3= false or false
```

Nil (nulo)

Este tipo de variables no tienen ningún valor o no han sido declaradas.

3.3. Lua: Arrays

Estas variables son especiales, son un conjunto de variables o de datos de distintos tipos. Vamos a poner un ejemplo de este tipo de variable:

Código 3.5: Variable array

```
var= {8, ''Texto'', true}
```

Esta variable tiene una sola dimensión y tiene 3 elementos.
También podemos crear otros arrays que tengan elementos con más dimensiones, como por ejemplo:

Código 3.6: Variable array con más elementos

```
var = {{8, ''Texto'', true},
       {''Hola'', 5, true, false},
       {10, 8, 9, 15}}
```

Seguidamente, vamos a ver cómo encontrar el valor de una variable. Como, por ejemplo la del valor "Texto" del primer ejemplo:

Código 3.7: Encontrar valor variable

```
var[2]
```

Otra característica que presentan estos arrays es que hay varias formas de acceder a las variables. Una de las formas de hacerlo es declarando variables dentro de un arreglo, como por ejemplo:

Código 3.8: Nuevo array

```
alumno= {nombre= ''David'', x= 8, y= 14}
```

3.4. Lua: Funciones

Las funciones pueden ser declaradas de varias formas:

Código 3.9: Funciones

```
function function1 ([argumentos]) BLOQUE end
funcion2= function ([argumentos]) BLOQUE end
variable= {false, 8, function ([argumentos]) BLOQUE end}
```

Dentro de las funciones podemos encontrar lo siguiente:

- variable en el caso de la funcion1.
- Los argumentos opcionales se escriben entre paréntesis o entre corchetes.
- El final del bloque se declara con end.

También tenemos la tabla modulo a la que podemos denominar de la siguiente forma:

Código 3.10: Tabla modulo

```
1 modulo.funcion ([argumentos])
```

Si ejecutamos la función con un punto "." la ejecutará con los argumentos y si utilizamos los dos puntos ":" se ejecutará la tabla modulo.

Código 3.11: Dos módulos de ejecución

```
1 datos= {x=8, y=10}
2 datos.edad= 8
3 datos.nota= 10
4
5 function datos.suma{datos, notas}
6             datos.x=datos.x+notas
7 end
```

Otra forma de usar los puntos es mediante la palabra self. Y el ejemplo anterior nos quedaría así:

Código 3.12: Self

```
1 function datos:suma{datos, notas}
2             self.x=self.x+notas
3 end
```

También podemos realizar una función muy simple, como la de Hola mundo. La cual sería la siguiente:

Código 3.13: Hola mundo

```
1 function hello{}
2     print{''Hola mundo''}
3 end
```

3.5. Lua: Operaciones

Las operaciones se realizan escribiendo dos variables y colocando entre ellas un operando, que puede ser una suma, una resta, una multiplicación, una división o una operación lógica. Ahora vamos a ver una serie de operaciones lógicas:

Código 3.14: Operaciones lógicas

```
X == Y -- será verdadero si X es igual a Y
X ~= Y -- será verdadero si X es diferente de Y
X > Y -- verdadero sólo si X es mayor que Y
X < Y -- verdadero sólo si X es menor que Y
X >= Y -- verdadero si X es mayor o igual que Y
X <= Y -- verdadero si X es menor o igual que Y
X -- verdadero si X existe, es decir, si a es distinto de null (
   vacio o nulo) y distinto de false (falso)
```

También podremos realizar ejemplos con el conjunto vacío:

Código 3.15: Otro ejemplo de operaciones

```
7 and 20 -> 30
20 and mostrar() -> mostrar()
nil and "b" -> nil
"nil" and 20 -> 20
false and 20 -> false
false and true -> false
nil and false -> nil
```

3.6. Lua: Estructuras de Control y Algoritmos

3.6.1. Estructuras de Control

Las estructuras de control son ciertas sentencias que usan palabras reservadas para realizar distintas acciones, o realizar cierto código si cumple una condición, y si no se cumple, realizar otro código.
if...then...else: Permite ejecutar un condicional. Vamos a poner un ejemplo:

Código 3.16: Ejemplo condicional

```
if CONDICIÓN then
    BLOQUE1
```

```
else
    BLOQUE2
end
```

Otra variante que tenemos es la variante elseif:

Código 3.17: Código elseif

```
if CONDICION1 then
    BLOQUE1
elseif CONDICIÓN2 then
    BLOQUE2
elseif CONDICIÓN 3 then
    BLOQUE3
end
```

Seguidamente, vamos a realizar un bucle con while...do...end:

Código 3.18: Bucle

```
while CONDICION do
    BLOQUE
end
```

Otro bucle que tenemos es el repeat-until:

Código 3.19: Bucle

```
repeat
    BLOQUE
until CONDICION
```

Y el último bucle que tenemos es el for:

Código 3.20: Bucle

```
for variable= inicio, fin[,step]do
        BLOQUE
end
```

3.6.2. Algoritmos

Los algoritmos son una serie de pasos que nos ayudan a resolver un problema. Vamos a poner un ejemplo de algoritmo con abrir la ventana:

Código 3.21: Algoritmo de cerrar la ventana

```
1.- Caminar hacia la ventana
2.- Verificar si la ventana está abierta (esto representa un if)
3.- si lo está, procedemos a cerrarla (bloque)
4.- FIN
```

Finalmente, vamos a añadirle estructuras de control:

Código 3.22: Algoritmo con estructuras de control

```
Si estamos en la mesa (if condicion then)
Nos vamos de la mesa (bloque1)
Sino (else), nos quedamos en la mesa (bloque2). Ahora seguimos
   avanzando.
Mientras no estemos comiendo: (while condicion do).
Verificar si todos han terminado (otro if).
Si han terminado nos vamos de la mesa (bloque1).
Si no han terminado (else), nos quedamos en la mesa (bloque2).
FIN.
```

4

LuaLaTeX II: Primeros pasos de LuaLaTeX

En este capítulo vamos a definir que es LuaLaTeX, escribiremos en Lua en TeX, aprenderemos modulación básica, veremos los tipos de print y terminaremos con el código de funciones matemáticas de Lua para LuaLaTeX.

1. ¿Qué es LuaLaTeX?
2. Escribir Lua en TeX.
3. Modulación básica.
4. Tipos de print.
5. Código de funciones matemáticas de Lua para LuaLaTeX.

4.1. ¿Qué es LuaLaTeX?

LuaLaTeX es un compilador tipo scripting que trabaja con lenguajes como Lua y Tex. En sus comienzos empezó como una versión de pdfLaTeX con scripting en Lua. Y en 2007 se lanzó como compilador con nombre propio.
Puede manejar tanto las fuentes y los strings Unicode. Estos strings son codificados en UTF-8. Usaremos Lua en lugar de otros lenguajes como Python (Lenguaje que también se puede usar) para realizar pseudo-programas con el objetivo de facilitarnos tareas. Bien programado, todo lo aprendido de Lua se podrá aplicar en LuaLaTeX por lo que aprender esto requiere un conocimiento sobre este lenguaje, una gran abstracción y práctica.

4.2. Escribir Lua en TeX

Para procesar un documento en LuaLaTeX realizaremos lo siguiente:

- Seleccionamos el compilador LuaLaTeX en overleaf o en nuestro procesador de LuaLaTeX. Es importante recordar que este compilador es más lento que pdfLaTeX y por lo tanto tardará más en emitir un resultado.

Aunque es bastante más cómodo el colocar código Lua en varios archivos Lua y luego insertarlos, muchas veces vamos a querer realizar el código en el mismo documento. Para poder realizar esto, LuaLaTeX tiene un par de comandos esenciales `\directlua` y `\latelua`. Ambos comandos funcionan de la misma forma, compilarán aparentemente igual, pero `\latelua` realmente se muestra cuando se compila la página citada. De preferencia escribiremos todo nuestro código Lua usando `\directlua`. Es importante recalcar que existen otros métodos para escritura de código Lua, se enseña uno básico para poder practicar en casa.

`\directlua` será usado de esta forma:

Código 4.1: Código de directlua

```
1 \directlua{Codigo Escrito en Lua}
```

Esta es la forma más básica de escritura de código Lua en LaTeX, dentro de este pequeño ambiente se podrán declarar funciones y comandos que podremos invocar sin problemas y con orden.

Código 4.2: Código de generación de números aleatorios

```
1 \directlua{tex.print(math.random())}
```

Este comando generará un número aleatorio cada vez que compilemos. Esto puede ser usado perfectamente para la generación de documentos únicos mediante Id o Serial para evitar que copien o distribuyan ciertos archivos.

Al escribir ese código en LuaLATEX nos sale lo siguiente: 0.50100888242061. Si volvemos a compilar: 0.40061755869566.
Observamos la alteración en el número a mostrar, cada vez que se compile nos va a mostrar otro número. Vamos a probar a aplicar operaciones básicas al comando.

Código 4.3: Código de generación de números aleatorios con operaciones

```
\directlua{tex.print((math.random()*20) - math.floor(math.random()
   /2)*2)}
```

Este código simula una multiplicación por 20 y una división por su módulo generado. Esto imprime en su primera compilación: 3.1194346598906
Este es el código usado para generar un módulo:

Código 4.4: Código de módulo

```
a - math.floor(a/b)*b
```

También se puede usar:

Código 4.5: Código de módulo básico

```
math.mod(numero,2)
```

Esto se suele usar en bucles para poder realizar operaciones basándonos en el módulo:

Código 4.6: Código de bucle de módulo

```
\directlua{
for i = 1, 100 do
    if (math.mod(i,2) == 0) then
        tex.print( i .. " es divisible")
    end
end
}
```

Este bucle genera todos los números del 1 al 100:

Código 4.7: for de Lua, parte 1

```
for i = 1, 100 do
```

Esto significa que comenzamos en el 1 y terminamos en el 100 y que realizamos lo siguiente:

Código 4.8: for de Lua, parte 2

```
    if (math.mod(i,2) == 0) then
        tex.print( i .. " es divisible")
    end
end
```

Aquí creamos un condicional que realizará la impresión del número que cumpla la condición. Esta condición es que sea par. Esto imprime la siguiente cadena: 2 es divisible 4 es divisible 6 es divisible 8 es divisible 10 es di-visible 12 es divisible 14 es divisible 16 es divisible 18 es divisible 20 es divisible22 es divisible 24 es divisible 26 es divisible 28 es divisible 30 es divisible 32 es divisible 34 es divisible 36 es divisible 38 es divisible 40 es divisible 42 es divisible 44 es divisible 46 es divisible 48 es divisible 50 es divisible 52 es divisible54 es divisible 56 es divisible 58 es divisible 60 es divisible 62 es divisible 64 es divisible 66 es divisible 68 es divisible 70 es divisible 72 es divisible 74 es divisible 76 es divisible 78 es divisible 80 es divisible 82 es divisible 84 es divisible86 es divisible 88 es divisible 90 es divisible 92 es divisible 94 es divisible 96 es divisible 98 es divisible 100 es divisible.

4.3. Modularización básica

Como hemos podido ver, una modificación básica de Lua en LuaLaTeX es agregar el prefijo "tex" a los print. `tex.print` para poder imprimir en el documento. Esto lo vamos a usar también cuando modularicemos.
Modularizar es crear pequeñas funciones que posteriormente serán invocadas en otras partes del texto para poder ser usadas más adelante sin necesidad de que sea de forma instantánea y para que pueda ser invocada con distintos datos.
Vamos a realizar una función para generar un factorial de un número:

Código 4.9: Código función de factorial

```
\directlua{
    function fact (n)
      if n == 0 then
        return 1
      else
        return n * fact(n-1)
      end
    end
    }
```

Ahora para invocar el código del factorial escribiremos lo siguiente en cualquier momento.

Código 4.10: Código de invocación

```
\directlua{tex.print(fact(4))}
```

Esto realizará la operación del factorial de 4 y lo mostrará por pantalla: 24
Vamos a poner otro ejemplo de modularización pero esta vez con texto y mostrando los tipos de cada variable con otro comando

Código 4.11: Código de tipo

```
\directlua{
  uno = 1
  local dos = 2
}
\directlua{
  tex.print(type(uno))
  tex.print(type(dos))
}
```

De esta forma imprimirá por pantalla el tipo de dato que es, si es un tipo nil, number, string, variable... Lo mostrará cuando se imprima. Este código genera: number nil

4.4. Tipos de print

Como ya hemos visto antes, tenemos varias formas de poder imprimir variables, cadenas y entornos dentro de un documento. El comando principal es `tex.print` este imrpime las variables y las cadenas de forma literal.

Código 4.12: Código print

```
tex.print("coso1", "coso2")
```

Este comando es igual que `tex.print` pero esta función puede recibir más de una cadena o matriz de cadenas a la vez, con un número opcional como argumento. También tiene particularidades tiene que lo imprime todo en una línea y los espacios no se omiten y no agregando caracteres al final de la línea.

Código 4.13: Código sprint

```
tex.sprint("coso1", "coso2")
```

Esto imprime en el documento: coso1coso2 Como podemos observar, sin espacio.
Como último dato de printado, vamos a ver el tprint. Esta función toma un número sin ningún límite de tablas como argumentos. Cada tabla es una matriz básica de cadenas con la primera cadena con un argumento que indica la posición. El resto se imprime como el sprint.

Código 4.14: Código tprint

```
tex.tprint({1, "a", "b"}, {"c", "d"})
```

Esto imprime por pantalla: abcd

4.5. Código de funciones matemáticas de Lua para LuaLaTeX

Código 4.15: Código valor absoluto

```
math.abs (x)
```

Devuelve el valor absoluto de x.

Código 4.16: Código valor arco coseno

```
math.acos (x)
```

Devuelve el arco coseno de x (en radianes).

Código 4.17: Código valor arco seno

```
math.asin (x)
```

Devuelve el arco seno de x (en radianes).

Código 4.18: Código valor arco tangente

```
math.atan (x)
```

Devuelve el arco tangente de x (en radianes).

Código 4.19: Código valor arco tangente de y/x

```
math.atan2 (y, x)
```

Devuelve el arco tangente de y/x (en radianes)

Código 4.20: Código valor menor entero

```
math.ceil (x)
```

Devuelve el menor entero mayor o igual que x.

Código 4.21: Código valor coseno de x

```
math.cos (x)
```

Devuelve el coseno de x (se asume que está en radianes).

Código 4.22: Código valor coseno hiperbólico de x

```
math.cosh (x)
```

Devuelve el coseno hiperbólico de x.

Código 4.23: Código valor sexagesimales el valor de x

```
math.deg (x)
```

Devuelve en grados sexagesimales el valor de x (dado en radianes).

Código 4.24: Código valor mayor entero menor o igual que x

```
math.floor (x)
```

Devuelve el mayor entero menor o igual que x.

Código 4.25: Código valor logaritmo natural de x.

```
math.log (x)
```

Devuelve el logaritmo natural de x.

Código 4.26: Código valor de pi.

```
math.pi
```

El valor de pi.

Código 4.27: Código valor del seno

```
math.sin (x)
```

Devuelve el seno de x (se asume que está en radianes).

Código 4.28: Código valor de la raíz cuadrada de x

```
math.sqrt (x)
```

Devuelve la raíz cuadrada de x.

Código 4.29: Código valor del ángulo x

```
math.rad (x)
```

Devuelve en radianes el valor del ángulo x.

Código 4.30: Código valor de x elevado a y

```
math.pow (x, y)
```

Devuelve x elevado a y.

Código 4.31: Código valor de seno hiperbólico de x

```
math.sinh (x)
```

Devuelve el seno hiperbólico de x.

Código 4.32: Código valor de logaritmo decimal de x

```
math.log10 (x)
```

Devuelve el logaritmo decimal de x.

Código 4.33: Código valor máximo de entre sus argumentos

```
math.max (x, ...)
```

Devuelve el mayor valor de entre sus argumentos.

Código 4.34: Código valor de menor entre sus argumentos

```
math.min (x, ...)
```

Devuelve el menor valor de entre sus argumentos.

Código 4.35: Código generador de semillas

```
1 math.randomseed (x)
```

Este código es el último, pero no el menos importante. Este código indica la generación de una semilla (como en C++ los que han usado generación de pseudo-aleatorios) para poder crear números aún más aleatorios. Es importante recalcar que similares semillas producirán los mismos números aleatorios.

5

Inserciones, inputs e incorporación externa

En este capítulo aprenderemos las partes que tienen un documento, las distintas inserciones que podemos realizar en él, cómo utilizar el comando input, las diferencias entre inclue e input, y terminaremos aprendiendo cómo se inserta código en nuestro documento.

1. Partes de un documento.
2. Inserciones.
3. El comando input.
4. Diferencias entre include e input.
5. Insertar código en nuestro documento.

5.1. Partes de un documento

Según el tipo de documentos podemos encontrar las siguientes partes de un documento:

- **report** y **book**: `\part{Titulo}`, `\chapter{Titulo}`.
- **article** y **book**: `\section{Titulo}`, `\subsection{Titulo}`.
- Para todas las clases de documento: `\paragraph{Titulo}`, `\subparagraph{}`.

Además de estas partes, podremos encontrar las siguientes partes en un **book** o en un **report**:
`\frontmatter`: Se utiliza para crear el prefacio, el prológo, los agredicimientos y las distintas tablas de contenido. Aunque para este libro se han utilizado estilos personalizados para crear los prológos y las tablas de contenido.
`\mainmatter`: Contiene las páginas numeradas y los capítulos que contienen el comando `\appendix`.
`\backmatter`: Contiene el índice, la bibliografía y el material adicional.
Estos tres comandos son opcionales y se deben escribir después de `\begin{document}`.

5.1.1. Numeración de un documento

Una vez hemos visto en qué partes se divide un documento, vamos a ver la importancia que tienen cada uno con su tipo de documento. Primero vamos a ver de mayor a menor el número asignado que tiene cada parte en su documentación:

Cuadro 5.1: Número asignado en cada parte del documento

Tipo de documento	subsubsection	subsection	section	chapter	part
article	3	2	1	No existe	0
book o report	3	2	1	0	-1

En los tipos de documentos **book** y **report** no se numeran las sub-secciones y en los **article** no se numeran las sub-subsecciones.
Finalmente, estas numeraciones se pueden ignorar si en algunas de estas partes se añade el asterisco, que ignora la numeración automática.

5.2. Inserciones

Si manejamos documentos muy grandes necesitaremos contar con un documento de raíz e ir añadiendo todos los capítulos. Este problema nos lo encontramos sobretodo en libros, ya que, contiene una gran cantidad de capítulos y un error en el código puede echar a perder el trabajo.

Para evitar que se dé la situación anterior utilizaremos el comando `\include{}` en el preámbulo y entre los corchetes colocaremos la ruta de donde está nuestro capítulo a añadir. Por otro lado, tenemos el comando `\includeonly{}` que realiza las mismas funciones que el `\include{}` pero se escribe en el preámbulo del documento. Ahora vamos a proceder a realizar un ejemplo con el comando `\include{}` y con el comando `\includeonly{}`:

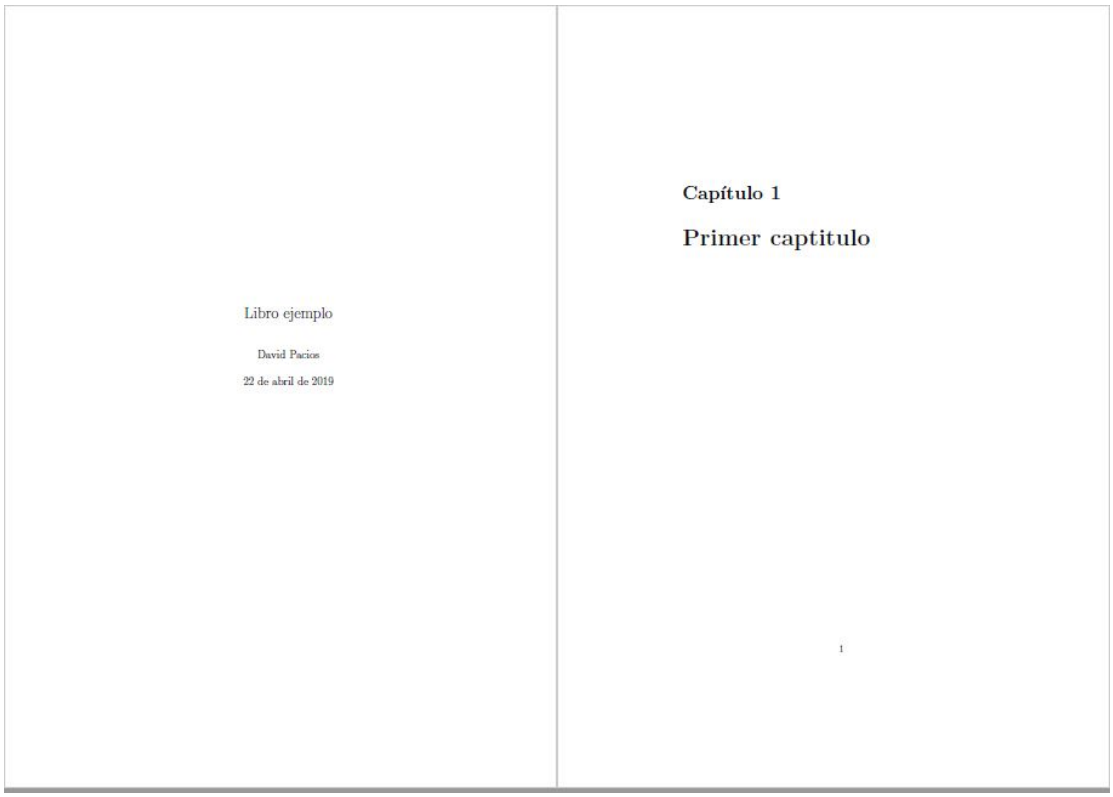


Figura 5.1: Libro con include

Código 5.1: Código del libro con include

```
1 \documentclass[oneside]{book}
2 \usepackage[utf8]{inputenc}
3 \usepackage[spanish]{babel}
4 \title{Libro ejemplo}
5 \author{David Pacios}
6 \date{\today}
7
8 \begin{document}
```

```
\maketitle
\include{Chapter1}
\end{document}
```

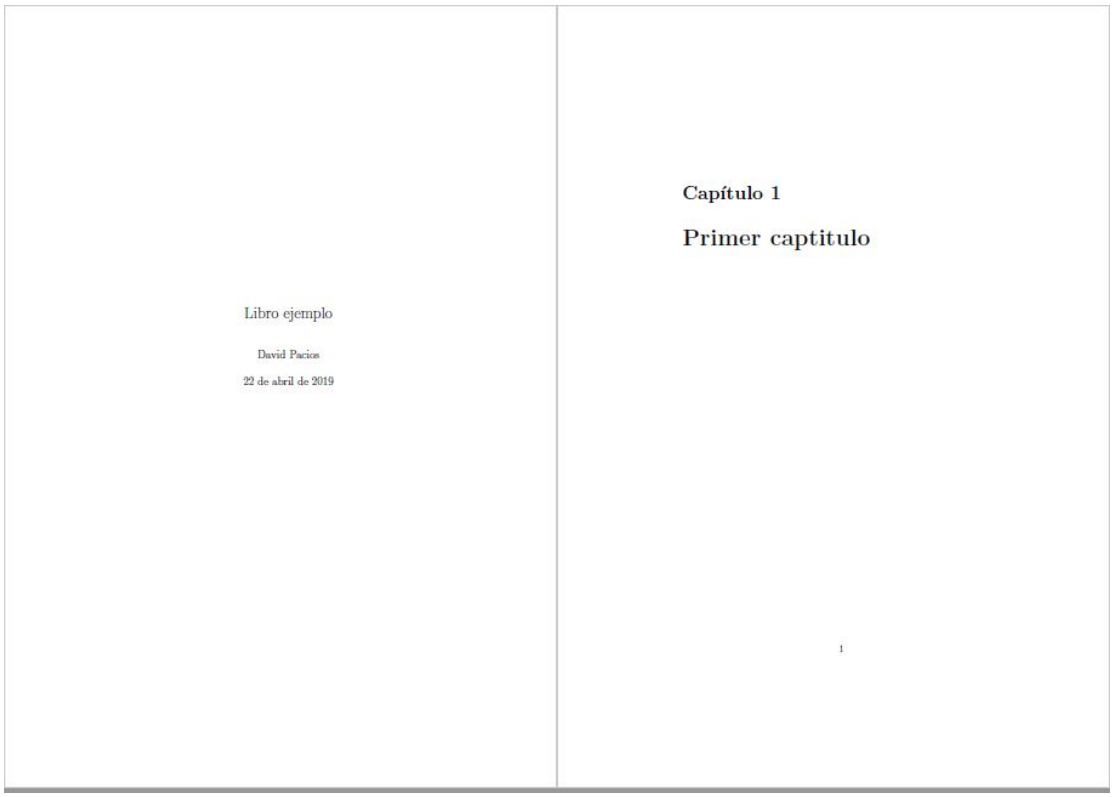


Figura 5.2: Libro con includeonly

Código 5.2: Código del libro con includeonly

```
\documentclass[oneside]{book}
\usepackage[utf8]{inputenc}
\usepackage[spanish]{babel}
\includeonly{Chapter1}
\title{Libro ejemplo}
\author{David Pacios}
\date{\today}
\begin{document}
\maketitle
\end{document}
```

Finalmente vamos a mostrar las diferencias entre `\include{}` e `\includeonly{}`:

- `\includeonly{}` acelera el procesado de los capítulos.
- `\include{}` se coloca en el cuerpo del documento.
- `\includeonly{}` se coloca en el preámbulo.

5.3. El comando input

Con este comando podemos insertar el capitulo directamente sin recurrir al salto de página. Los documentos son insertados directamente desde la raíz.
Para utilizarlo como comando sólo tendremos que poner el comando `\input{}` en el cuerpo del texto y entre corchetes seleccionaremos la ruta al documento o el mismo documento. Finalmente vamos a mostrar un ejemplo con este comando:

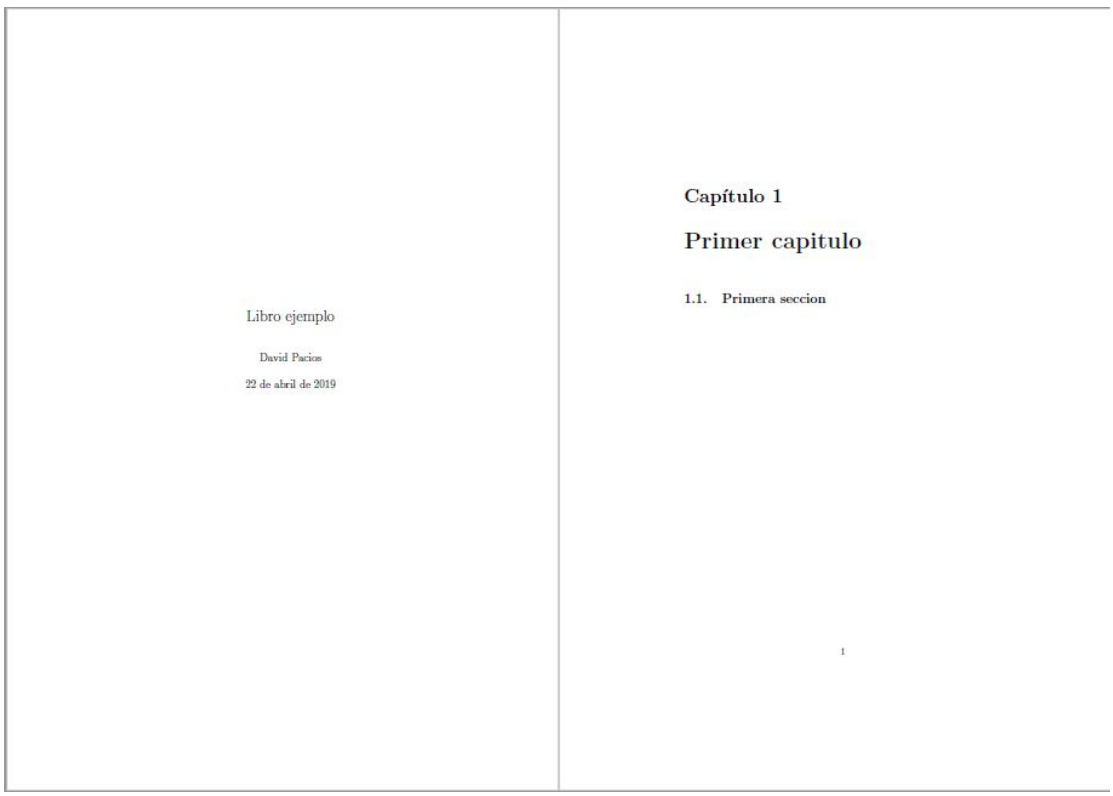


Figura 5.3: Libro con input

Código 5.3: Código del libro con input

```
\documentclass[oneside]{book}
\usepackage[utf8]{inputenc}
\usepackage[spanish]{babel}

\title{Libro ejemplo}
\author{David Pacios}
\date{\today}

\begin{document}
\maketitle
\input{Chapter1.tex}
\end{document}
```

5.4. Diferencias entre include e input

Las propiedades más destacadas del comando `\input{}` son las siguientes:

- Se puede utilizar en todo el documento, ya sea en el preámbulo, en el cuerpo e incluso junto a algunos paquetes.
- Se puede insertar documentos dentro del mismo comando `\input{}`.
- Lo único que hace este comando es insertar el documento.

Las propiedades más destacadas del comando `\include{}` son las siguientes:

- Deja una página en blanco entre la portada y el documento insertado.
- La nueva página insertada tiene unos valores distintos a los del documento.

Para tenerlo más claro vamos a hacer una tabla con sus diferencias más marcadas:

Cuadro 5.2: Diferencias entre input e include

	include	**input**
Nueva página	SI	NO
Comando en preámbulo	SI	SI
Comando en cuerpo	NO	SI
Valores del nuevo documento	SI	NO

5.5. Insertar código en nuestro documento

5.5.1. El entorno verbatim

El entorno que por defecto utiliza LaTeX para mostrar el código es el verbatim. Genera una salida de código monoespaciada. Para comenzar este entorno comenzaremos con el comando `\begin{verbatim}` y terminaremos con el comando `\end{verbatim}`, y entre medias de ambos colocaremos nuestra linea de código. Para tenerlo más claro vamos a poner un ejemplo:

```
Este texto mantiene los comandos \textbf{variados}
y se ignoran \LaTeX{}.
```

Código 5.4: Código verbatim

```
\begin{verbatim}
Este texto mantiene los comandos \textbf{variados}
y se ignoran \LaTeX{}.
\end{verbatim}
```

Además, si queremos colocar una línea de código únicamente o destacar cualquier comando, utilizaremos el comando `\verb` y entre barras como esta `|` colocaremos nuestro comando. Para tenerlo más claro vamos a poner un ejemplo:
Este comando `D: Trabajo\Carpeta` indica la ubicación.

Código 5.5: Código verbatim texto

```
Este comando \verb|D: Trabajo\Carpeta| indica la ubicación.
```

5.5.2. Usando listings para resaltar el código

Otro entorno que tenemos para presentar código es el lstings, es muy importante definir en el preámbulo el paquete listings y una vez hecho, se comenzará con el comando `\begin{lstlisting}[lenguaje, número de línea]`, entre corchetes colocaremos el lenguaje que estamos utilizando, y si nos dividen el código indicaremos la línea, y terminaremos con el comando `\end{lstlisting}`. Para tenerlo más claro vamos a poner un ejemplo:

```
\begin{frame}{Cryptoparty}
    \begin{alertblock}{Evento Internacional}
    Organizado por la UCM en España, este 6 de Abril en el Círculo
       de Bellas artes. \url{https://cryptoparty.ucm.es/}
```

```
    \end{alertblock}
    \begin{exampleblock}{ Qu é es?}
    Las CryptoParties son un evento gratuito y abierto para todo el
        mundo, especialmente para aquellos sin conocimientos
       previos que no hayan asistido previamente.
    \end{exampleblock}
    \begin{center}
     \includegraphics[width=0.2\textwidth]{Figures/Cryptoparty
        _2019.jpg}
    \end{center}
\end{frame}
```

Entre los lenguajes que podemos utilizar se encuentran los siguientes:

- C++.
- Cobol.
- Gnupot.
- HTML.
- Octave.
- Pascal.
- Python.
- Scilab.
- VHDL.
- XML.

5.5.3. Importando el código

Además de poder pegar nuestro código en el listing directamente, también podremos importar nuestro código con el comando `\lstinputlisting[lenguaje]{Programa}`. Para verlo más claro vamos a poner un ejemplo:

```
X=[365.015,404.64,407.55,435.91,546.01,576.93,579.13,696.50,706]; %
   valores de lamba experimental
Y=[365.015, 404.656, 407.783, 435.833, 546.074, 576.960,
   579.066,696.543,]; %valores de lamba te rica
N=19; %n mero de puntos del ajuste


%Calculo de la penmdiente y la ordenada en el origen con su error
xm=mean(X);ym=mean(Y);
sumxx=sum((X-xm).^2);sumxy=sum((X-xm).*(Y-ym));sumyy=sum((Y-ym).^2)
   ;
a=sumxy/sumxx
b=ym-a*xm
d=Y-a*X-b;
```

Código 5.6: Código de importar el código

```
\lstinputlisting[language=Octave, firstline=2, lastline=12]{
   regresionTFG.m}
```

5.5.4. Dando estilo al código

Este entorno se puede modificar en el preámbulo los siguientes parámetros:

- **backgroundcolor**: Indica el color de fondo. Necesita el paquete color o xcolor.
- **commenstyle**: Estilo de los comentarios en el lenguaje.
- **basicstyle**: Fuente, tamaño de la letra en el código.
- **keywordstyle**: Estilo de las palabras clave.
- **numberstyle**: Estilo de las numeraciones.
- **numbersep**: Distancia entre los números del código.
- **stringstyle**: Estilo de las cadenas en el lenguaje.
- **showspaces**(true/false): Enfatiza en los espacios de las cadenas.
- **showstringspaces**(true/false): Enfatiza en los espacios en las cadenas.
- **showtabs**(true/false): Enfatiza en las tabulaciones en el código.

- **numbers**(left/right/none): Posición de los números.
- **prebreak**: Indica una marca al terminar una linea.
- **captionpos** (t/b): Posición del caption.
- **frame** (none/leftline/topline/bottomline/lines/single/shadowbox): Muesta el marco fuera del código.
- **breakwhitespace**: Muestra los espacios que ocurren cuando hay espacios en blanco.
- **breaklines**: Saltos de línea automáticos.
- **keepspaces**: Mantiene los espacios en el código, es útil para la indentación.
- **tabsize**: Tamaño de tabla por defecto.
- **escapeinside**: Especifica algunos caracteres en el código.
- **rulecolor**: Especifica el color en el marco de la caja.

Código 5.7: Código ejemplo personalizado

```
\documentclass{article}
\usepackage[utf8]{inputenc}

\usepackage{listings}
\usepackage{color}
%Colores definidos
\definecolor{codegreen}{rgb}{0,0.6,0}
\definecolor{codegray}{rgb}{0.5,0.5,0.5}
\definecolor{codepurple}{rgb}{0.58,0,0.82}
\definecolor{backcolour}{rgb}{0.95,0.95,0.92}

%Dandole estilo al codigo
\lstdefinestyle{mystyle}{
  backgroundcolor=\color{backcolour},   commentstyle=\color{
     codegreen},
  keywordstyle=\color{magenta},
  numberstyle=\tiny\color{codegray},
  stringstyle=\color{codepurple},
  basicstyle=\footnotesize,
  breakatwhitespace=false,
  breaklines=true,
```

```
  captionpos=b,
  keepspaces=true,
  numbers=left,
  numbersep=5pt,
  showspaces=false,
  showstringspaces=false,
  showtabs=false,
  tabsize=2
}
%Set de mi c digo
\lstset{style=mystyle}
```

Además de lo explicado anteriormente, podemos darle estilo con los siguientes comandos:

- `\lstdefinestyle{mystyle}{}`: Definimos el código llamandolo mystyle y le damos las características que queramos.
- `\lstset{stylemystyle`: Estilo ya definido.

5.5.5. Listado y nombre del código

Como en las imágenes y en las tablas, podremos darle nombre y numerar nuestro código con el comando caption al lado del lenguaje. Para tenerlo más claro vamos a poner un ejemplo:

Código 5.8: Ejemplo con nombre

```
import numpy as np

def incmatrix(genl1,genl2):
    m = len(genl1)
    n = len(genl2)
    M = None #to become the incidence matrix
    VT = np.zeros((n*m,1), int)  #dummy variable
    #compute the bitwise xor matrix
    M1 = bitxormatrix(genl1)
    M2 = np.triu(bitxormatrix(genl2),1)

    for i in range(m-1):
        for j in range(i+1, m):
            [r,c] = np.where(M2 == M1[i,j])
            for k in range(len(r)):
```

```
                VT[(i)*n + r[k]] = 1;
                VT[(i)*n + c[k]] = 1;
                VT[(j)*n + r[k]] = 1;
                VT[(j)*n + c[k]] = 1;

                if M is None:
                    M = np.copy(VT)
                else:
                    M = np.concatenate((M, VT), 1)

                VT = np.zeros((n*m,1), int)
    return M
```

Código 5.9: Ejemplo con nombre código

```
\begin{lstlisting}[language=Python, caption=Ejemplo con nombre]
import numpy as np

def incmatrix(genl1,genl2):
    m = len(genl1)
    n = len(genl2)
    M = None #to become the incidence matrix
    VT = np.zeros((n*m,1), int)  #dummy variable

    #compute the bitwise xor matrix
    M1 = bitxormatrix(genl1)
    M2 = np.triu(bitxormatrix(genl2),1)

    for i in range(m-1):
        for j in range(i+1, m):
            [r,c] = np.where(M2 == M1[i,j])
            for k in range(len(r)):
                VT[(i)*n + r[k]] = 1;
                VT[(i)*n + c[k]] = 1;
VT[(j)*n + r[k]] = 1;
                VT[(j)*n + c[k]] = 1;

                if M is None:
                    M = np.copy(VT)
                else:
```

```
                    M = np.concatenate((M, VT), 1)

                VT = np.zeros((n*m,1), int)

    return M
```

Y si queremos que nos aparezca en una lista utilizaremos el comando `\lstlistoflistings`.

5.5.6. Poner palabras clave

Para poner palabras clave utilizaremos el comando `\lstdefinelanguage{lenguaje}{Características}`. Para tenerlo más claro vamos a poner un código de ejemplo:

Código 5.10: Código palabras clave

```
\lstset{language=LaTeX,
    keywordstyle=\color{rojo},
    texcsstyle=*\color{myblue},
    basicstyle=\textbf\normalfont\ttfamily,
    commentstyle=\color{comments}\ttfamily,
    stringstyle=\rmfamily,
    numbers=left,
    numberstyle=\scriptsize,
    stepnumber=1,
    numbersep=8pt,
    captionpos=top,
    showstringspaces=false,
    breaklines=true,
    frameround=ftff,
    morekeywords={RequirePackage,ProvidesPackage,NeedsTeXFormat},
    backgroundcolor=\color{background},
    literate=
            *{\{}{{\textcolor{myblue}{\{}}}{1}
            {\}}{{\textcolor{myblue}{\}}}}{1}
            {\\}{{\textcolor{myblue}{\textbackslash}}}{1}
            {\$}{{\textcolor{rojo}{\$}}}{1}
            {\&}{{\textcolor{rojo}{\&}}}{1}
           {\documentclass}{{\textcolor{rojo}{\textbackslash
              \documentclass}}}{12}
```

6 Powerdot y Beamer

En este capítulo aprenderemos a realizar presentaciones con Powerdot, Beamer y realizaremos ejercicios para aprender el contenido de este capítulo.

1. Presentación en entorno Powerdot.
2. Presentación en entorno Beamer.
3. Ejercicios resueltos.

6.1. Presentación en entorno Powerdot

6.1.1. ¿Qué es Powerdot?

Powerdot es una clase de documento que nos permite crear unas presentaciones muy vistosas en poco tiempo. En este capítulo vamos a explicar cómo realizar unas buenas presentaciones, añadirles anotaciones, añadir animaciones, añadir código y mostrar sus diferencias respecto del entorno Beamer.
Lo primero es explicar el entorno básico, para ello en nuestro preámbulo tendremos que poner el comando `\documentclass{powerdot}`, con este comando ya instalado tendremos nuestra clase de documento ya instalado. Es importante utilizar para este tipo de presentaciones el compilador de LaTeX, ya que, los compiladores pdfLaTeX o xeLaTeX no compilarán nuestra presentación del todo y nos dará problemas. Una vez hecho, podremos colocarle un título, autor y fecha, dando como resultado lo siguiente:

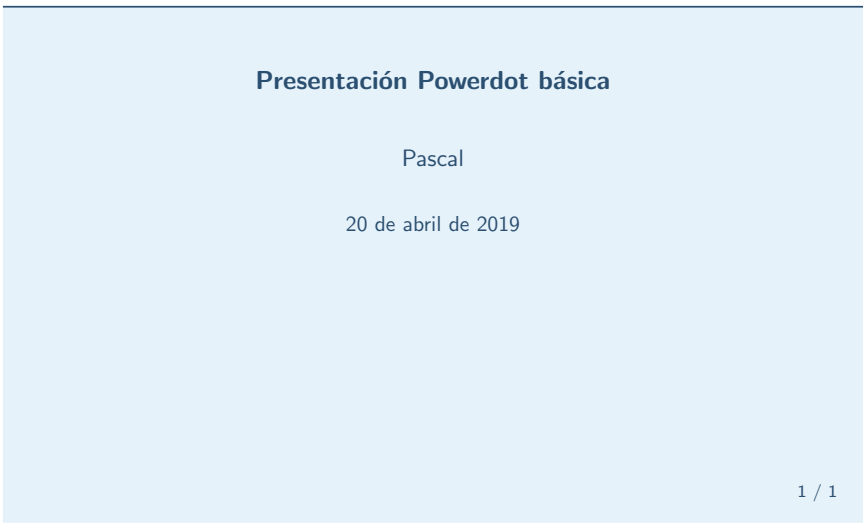


Figura 6.1: Powerdot presentación básica

Código 6.1: Código de la presentación básica en Powerdot

```
1 \documentclass{powerdot}
2 \usepackage[utf8]{inputenc}
3 \usepackage[spanish]{babel}
```

```
% Titulo de la presentacion
\title{Presentación Powerdot básica}
\author{Pascal}
\date{\today}

\begin{document}
  %Titulo
  \maketitle
\end{document}
```

Con este código sólo podremos realizar la portada de la presentación, con su título, pero si queremos realizar un apartado utilizaremos el comando `\section{}` y entre corchetes le daremos el nombre a nuestro apartado, y por otro lado, tenemos el entorno de la diapositiva, que comienza con el comando `\begin{slide}` y termina con el comando `\end{slide}`. Una vez que hayamos escrito esto en nuestro código quedaría algo parecido a lo siguiente en nuestra presentación:

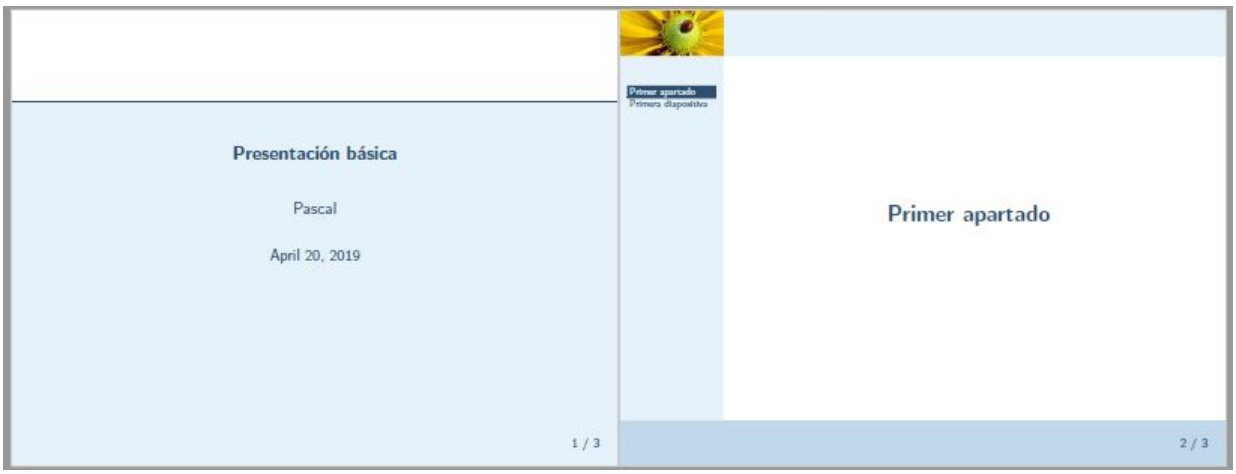


Figura 6.2: Powerdot con la presentación básica

Figura 6.3: Primera diapositiva

Código 6.2: Código Powerdot presentación básica

```
\documentclass{powerdot}
\usepackage[utf8]{inputenc}




%Titulo presentacion y autoria
\title{Presentación básica}
\author{Pascal}
\date{\today}

\begin{document}
  %Titulo
  \maketitle

%Primera secci n que ocupa toda la p gina
\section{Primer apartado}

%Primera diapositiva
\begin{slide}{Primera diapositiva}
\end{slide}
\end{document}
```

6.1.2. Características básicas de la diapositiva

Como hemos visto anteriormente, se le puede dar características a nuestra diapositiva. Para ello, vamos a utilizar la parte opcional de la clase del documento quedando el comando de clase de documento `\documentclass[Parte opcional]{powerdot}`, y en la parte opcional podremos colocar:

- mode: Es el modo de la presentación de la diapositiva entre los cuales podemos encontrar:
  - print: Modo impresión por defecto, borra las capas y las animaciones.
  - present: Modo por defecto para la presentación.
  - handout: Produce un espacio en blanco y nos permite imprimir nuestro documento a dos capas.
- paper: Es el tamaño del papel en la presentación, entre los cuales podemos encontrar:
  - smartboard: La diapositiva se adapta al tamaño del papel.
  - wide screen: El papel se extiende a lo largo de la pantalla.
  - screen: La presentación ocupa tres cuartas partes del papel.
  - a4paper: La presentación se imprime a A4.
  - letterpaper: La presentación se realiza en un papel tipo carta.
- orient: Indica la orientación de la diapositiva, entre las cuales podemos encontrar:
  - landscape: Orientación en horizontal.
  - portrait: Oritentación en vertical.

Vamos a darle unas características básicas a la presentación, utilizando el mode=print, el paper=smartboard y la orientación es orient=landscape. Y vamos a ver cómo quedaría:

Figura 6.4: Presentación de características básicas

Código 6.3: Código presentación de características básicas

```
\documentclass[
%Tama o de la letra
    13pt,
    paper=smartboard, %Diapositiva personalizada
    orient=landscape, %Orientacion del papel
    ]{powerdot}

\usepackage[utf8]{inputenc}
\usepackage[spanish]{babel}



%Datos presentacion
\title{Características básicas de la diapositiva}
\author{Pascal}
\date{\today}

\begin{document}
  %Titulo
  \maketitle
\end{document}
```

Además de poder darle unas características a la diapositiva, vamos a poder añadir notas. Para ello, en la parte opcional de clase documento tendremos que indicar display=note para que nos deje anotar en nuestro documento, y cuando queramos añadir anotación tendremos que trabajar con el entorno note, para ello, comenzaremos nuestro entorno con `\begin{note}` y terminando nuestro entorno con `\end{note}`, y entre medias podremos escribir un texto para nuestra nota y ponerla un título si al principio añadimos unos corchetes. Vamos a realizar una diapositiva con su respectiva anotación:

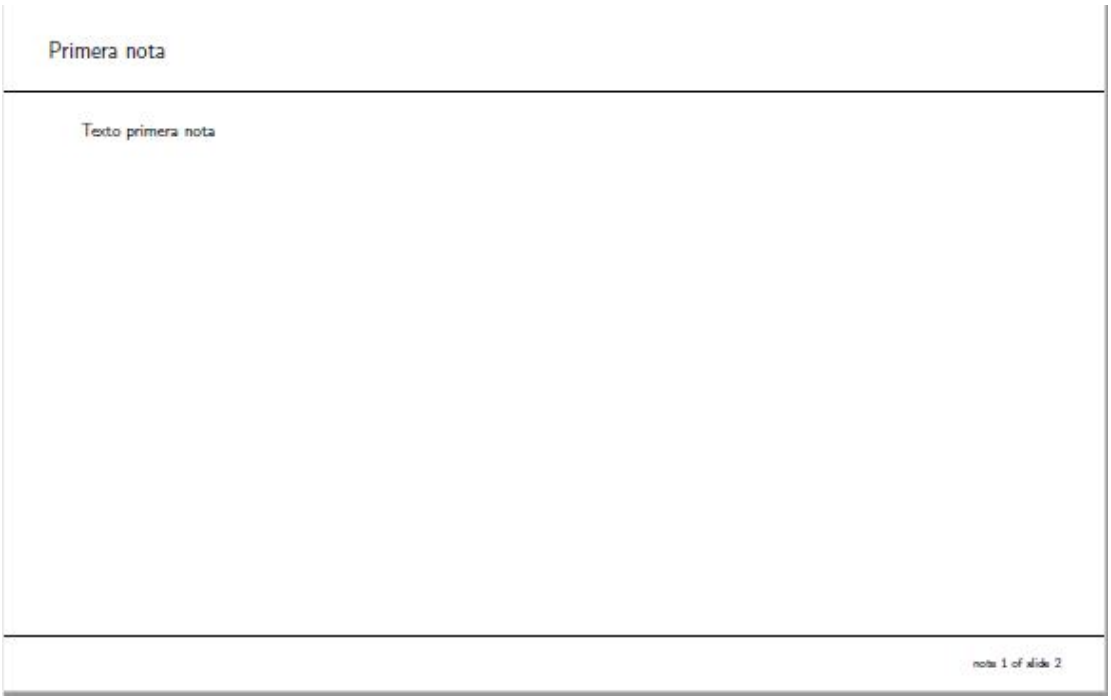


Figura 6.5: Diapositiva anotación

Código 6.4: Código para realizar anotaciones

```
\documentclass[
    13pt,
    display=slidesnotes,
    paper=smartboard,
    orient=landscape,
    ]{powerdot}
\usepackage[utf8]{inputenc}
\usepackage[spanish]{babel}
```

```
9
10 \title{Presentacion con notas}
11 \author{Pascal}
12 \date{\today}
13
14 \begin{document}
15   \maketitle
16
17
18 \begin{slide}{Primera diapositiva}
19 Podemos añadir anotaciones
20 \end{slide}
21 \begin{note}{Primera nota}
22 Texto primera nota
23 \end{note}
24 \end{document}
```

6.1.3. Dando estilo y color a las presentaciones

Podemos cambiar el estilo de nuestra presentación en la parte opcional de la clase del documento, al principio de nuestro preámbulo, para ello sólo tenemos que añadir el comando style= y detrás del igual, seleccionaremos nuestro estilo. Por otro lado, podremos darle color también en el preámbulo con el comando `\pdsetup{palette}`, es importante decir que no todos los estilos se les puede cambiar de color. Ahora vamos a ver un ejemplo de presentación con un estilo y color cambiados:

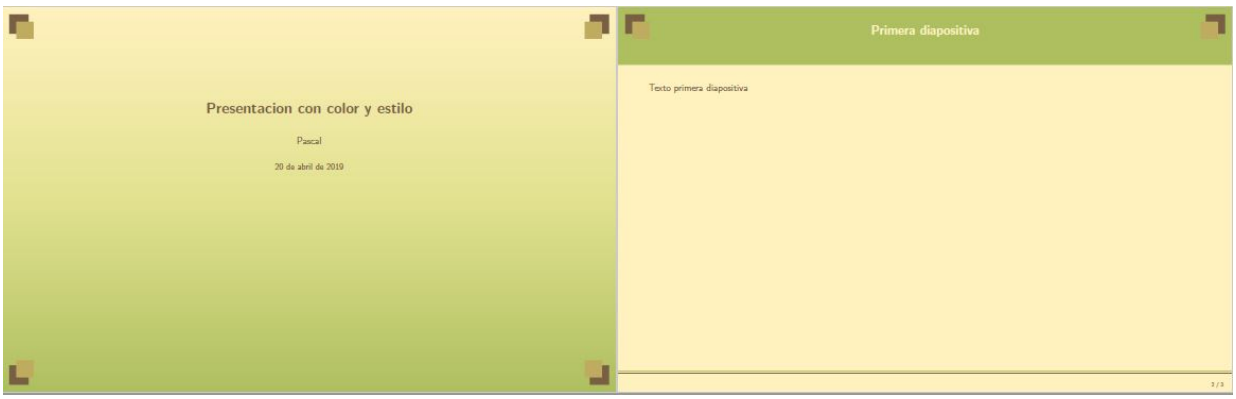


Figura 6.6: Presentación con color y estilo

Código 6.5: Código presentación con color y estilo

```
\documentclass[
    13pt,
    style=klope, %Estilo presentacion
    display=slidesnotes,
    paper=smartboard,
    orient=landscape,
    ]{powerdot}

\pdsetup{
    palette=Spring,  %Color primavera
}

\usepackage[utf8]{inputenc}
\usepackage[spanish]{babel}
\title{Presentacion con color y estilo}
\author{Pascal}
\date{\today}
\begin{document}
  \maketitle
\begin{slide}{Primera diapositiva}
Texto primera diapositiva
\end{slide}
\begin{slide}{Segunda diapositiva}
\end{slide}
\end{document}
```

Finalmente, vamos a mostrar sus estilos con sus respectivas paletas:

Cuadro 6.1: Estilo con paleta de colores

Estilos	Colores
simple	Sin paleta de colores
tycja	Sin paleta de colores
ikeda	Sin paleta de colores
fyma	blue, green, gray, brown, orange
ciment	Sin paleta de colores
elcolors	Sin paleta de colores
aggie	Sin paleta de colores
husky	Sin paleta de colores
sailor	River, Wine, Chocolate, Cocktail
upen	Sin paleta de colores
bframe	Sin paleta de colores
horatio	Sin paleta de colores
paintings	Syndics, Skater, GoldenGate, Moitessier, PearlEarring, Lamentation, HolyWood, Europa, MayThird, Charon
klope	Spring, PastelFlower, BlueWater, BlackWhite
jefka	brown, seagreen, blue, white
pazik	red, brown

6.1.4. Animaciones

Las animaciones en el caso de la clase de documento Powerdot se determinan al principio, en el preámbulo, mediante el comando `\pdsetup{trans}`, y detrás del igual, como en el caso de los estilos podremos determinar las animaciones. De las cuales podemos añadir las siguientes:

- Split.
- Blinds.
- Box.
- Wipe.
- Dissolve.
- Glitter.
- Replace.
- Fly.
- Push.

- Cover.
- Uncover.
- Fade.

Para ver cómo quedaría el código con las animaciones y como quedaría la presentación vamos a poner un ejemplo:

Figura 6.7: Diapositivas con animación

Código 6.6: Código diapositivas con animación

```
\documentclass[
    13pt,
    style=jefka,
    display=slidesnotes,
    paper=smartboard,
    orient=landscape,
    ]{powerdot}

\pdsetup{
    trans=Split,      %Animacion
    palette=blue,
}

\usepackage[utf8]{inputenc}
\usepackage[spanish]{babel}
\title{Animacion}
\author{Pascal}
\date{\today}
```

```
\begin{document}
  \maketitle
\begin{slide}{Primera diapositiva}
\end{slide}
\begin{slide}{Segunda diapositiva}
\end{slide}
\end{document}
```

Además de añadir animaciones entre diapositivas, podemos añadir animaciones a nuestro texto en las animaciones o a los entornos itemize o enumerate, entre las cuales podemos destacar:

- `\pause`: No muestra la parte del texto hasta la siguiente diapositiva.
- `\begin{itemize}[type 1]`: Muestra el siguiente itemizado en la siguiente diapositiva.

No sólo podemos colocar este tipo de animaciones, sino que podemos decidir dónde aparecen las animaciones en el caso de los itemizes o enumerados podemos realizar lo siguiente:

- `\item<-2>`: El elemento itemizado se mostrará en todas las diapositivas menos en la segunda.
- `\item<2->`: El elemento itemizado aparecerá desde la segunda diapositiva.
- `\item<2-5>`: El elemento itemizado aparecerá desde la diapositiva 2 a la 5.

Para tenerlo más claro vamos a poner un ejemplo donde tengamos animaciones en el comando itemize y una animación entre diapositivas:

Figura 6.8: Transición uno

Figura 6.9: Transición dos

Código 6.7: Código de presentación con animaciones

```
\documentclass[
    13pt,
    style=klope,
    display=slidesnotes,
    paper=smartboard,
    orient=landscape,
    ]{powerdot}

\pdsetup{
    trans=Split,
    palette=Spring,
}
\usepackage[utf8]{inputenc}

\title{Presentación con animaciones}
\author{Pascal}
\date{\today}

\begin{document}
  \maketitle
  \section{Primera parte}

\begin{slide}{Primera diapositiva}
Primer texto. \pause \\
Se muestran en secuencia:
    \begin{itemize}[type=1]
```

```
27        \item<2> Uno
28        \item<3> Dos
29        \item<4> Tres
30    \end{itemize}
31 \end{slide}
32 \begin{slide}[method=direct]{Segunda diapositiva}
33 \end{slide}
34 \begin{slide}{Tercera diapositiva}
35 \end{slide}
36 \end{document}
```

6.1.5. Añadir código a la presentación

Como ventaja respecto al Beamer es que podemos añadir cualquier código de cualquier programa para mostrarlo en la presentación. Para poder realizarlo, ponemos en el preámbulo el paquete listings para que nos permita añadir el código. Una vez lo hemos añadido, sólo tenemos que utilizar el entorno code para meter nuestro código, para ello empezaremos con el comando `\begin{code}` y terminaremos con el comando `\end{code}`. Para ver cómo quedaría la presentación con el código vamos a poner un ejemplo:

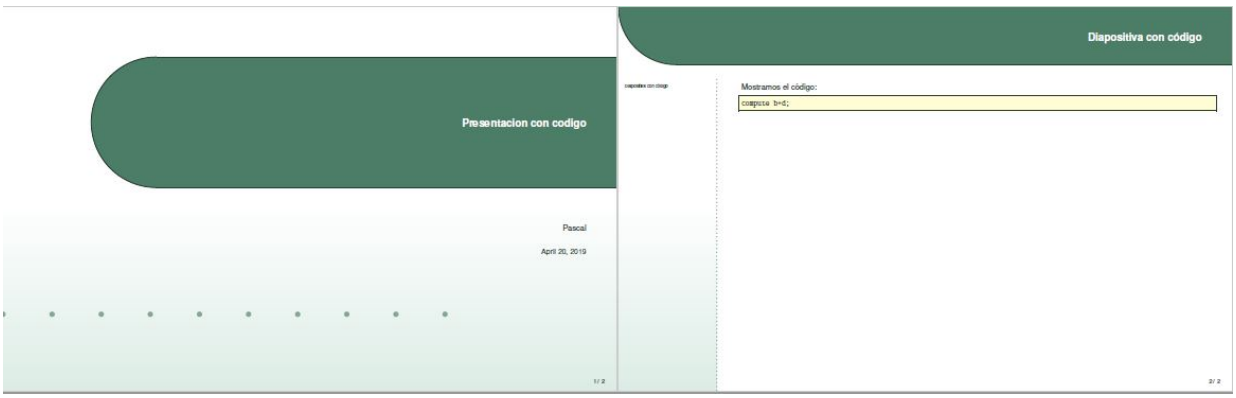


Figura 6.10: Presentación con código

Código 6.8: Código de la presentación con código

```
1 \documentclass[
2     13pt,
3     style=sailor,
4     display=slidesnotes,
```

```
    paper=smartboard,
    orient=landscape,
    ]{powerdot}

\pdsetup{
    trans=Split,
    palette=River,
}

\usepackage[utf8]{inputenc}

%C digo listing
%-----------------------------------------
\usepackage{listings}
\lstnewenvironment{code}{%
\lstset{frame=single,escapeinside='',
backgroundcolor=\color{yellow!20},
basicstyle=\footnotesize \ttfamily}
}{}
\title{Presentacion con codigo}
\author{Pascal}
\date{\today}

\begin{document}
  \maketitle
\begin{slide}{Diapositiva con código}
Mostramos el código:
\begin{code}
compute a+b;

compute b+d;
\end{code}
\end{slide}
\end{document}
```

6.1.6. Diferencias respecto de Beamer

Como hemos visto anteriormente, Powerdot nos permite diseñar una presentación con gran variedad de animaciones y estilos pero, ¿cuál de las dos clases de documento merece más la pena utilizar?¿Beamer o Powerdot? Vamos a ver las diferencias del Powerdot respecto

de Beamer:

Cuadro 6.2: Comparación de Powerdot y Beamer

	Powerdot	**Beamer**
Compilador	Latex	Pdflatex
Diapositivas	Entorno slide	Entorno frame
Animaciones	Se realizan en el preámbulo	Se realizan en el cuerpo del documento
Código	Se puede insertar	No se puede inser
Secciones	Las utiliza	No las utiliza

Como podemos ver en el cuadro 6.2 cada uno tiene un compilador distinto, tiene unas características diferentes, por lo que depende de cada usuario utilizar una clase de documento u otro.

Figura 6.11: Presentación básica Beamer

Figura 6.12: Powerdot presentación básica

Como podemos ver a partir de las imágenes, ni en su estilo más básico se parecen los entornos Powerdot ni Beamer. Por lo que tendremos de libertad de elegir entre un estilo y otro.

6.2. Presentación en entorno Beamer

6.2.1. ¿Qué es Beamer?

Beamer es una clase de documento que está diseñado para realizar presentaciones.
Con este sistema, podemos colocar fórmulas y cajas con comodidad.
Para poder utilizarlo, definiremos en el préambulo la clase beamer.Y utilizaremos el entorno frame para dar el nombre a cada diapositiva.

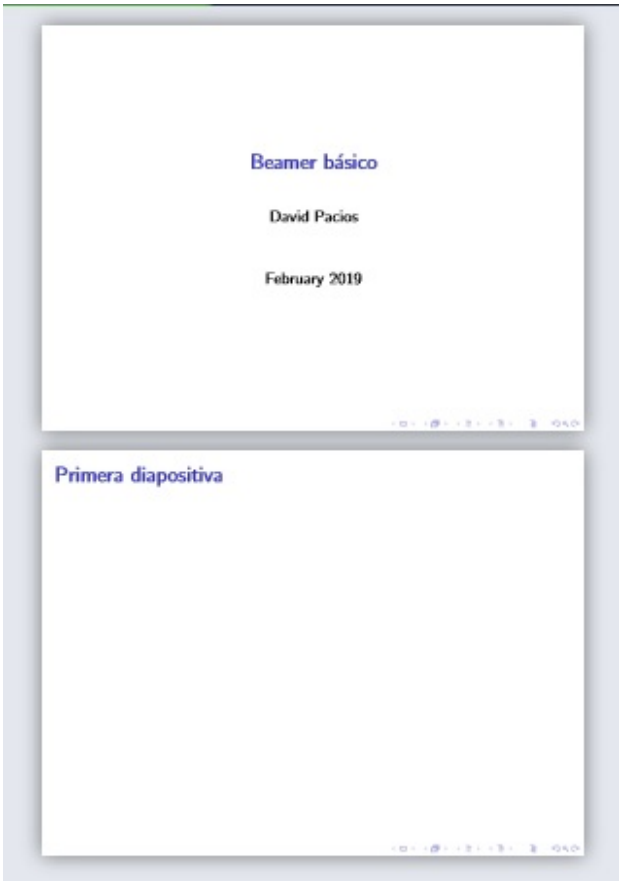


Figura 6.13: Beamer básico

Código 6.9: Código Beamer básico

```
1  \documentclass{beamer}
2  \usepackage[utf8]{inputenc}
3  \usepackage{spanish}[babel]
4
5  \title{Beamer básico}
6  \author{David Pacios}
7  \date{February 2019}
8
9  \begin{document}
10 \frame{\titlepage}
11 \input{Frame1.tex}
12 \end{document}
```

En este código podemos ver un nuevo comando, el comando \input{}, y entre corchetes, colocaremos la situación de lo que queramos insertar en el documento.
Y finalmente, tenemos otro comando para insertar las diapositivas, que es el comando \include{} que funciona igual que el anteriormente descrito.

Código 6.10: Código Beamer básico con include

```
\documentclass{beamer}
\usepackage[utf8]{inputenc}
\usepackage{spanish}[babel]

\title{Beamer básico}
\author{David Pacios}
\date{February 2019}

\begin{document}
%incluir los Frames con \include
\include{Frame/Frame1} %Titulo
\include{Frame/Frame2}

\end{document}
```

6.2.2. Creación de bloques

Para crear bloques dentro del documento utilizaremos el entorno block, que comienza con \begin{block}{} y entre corchetes colocamos el título que le queramos dar el bloque, y finalizamos el entorno con \end{block}.

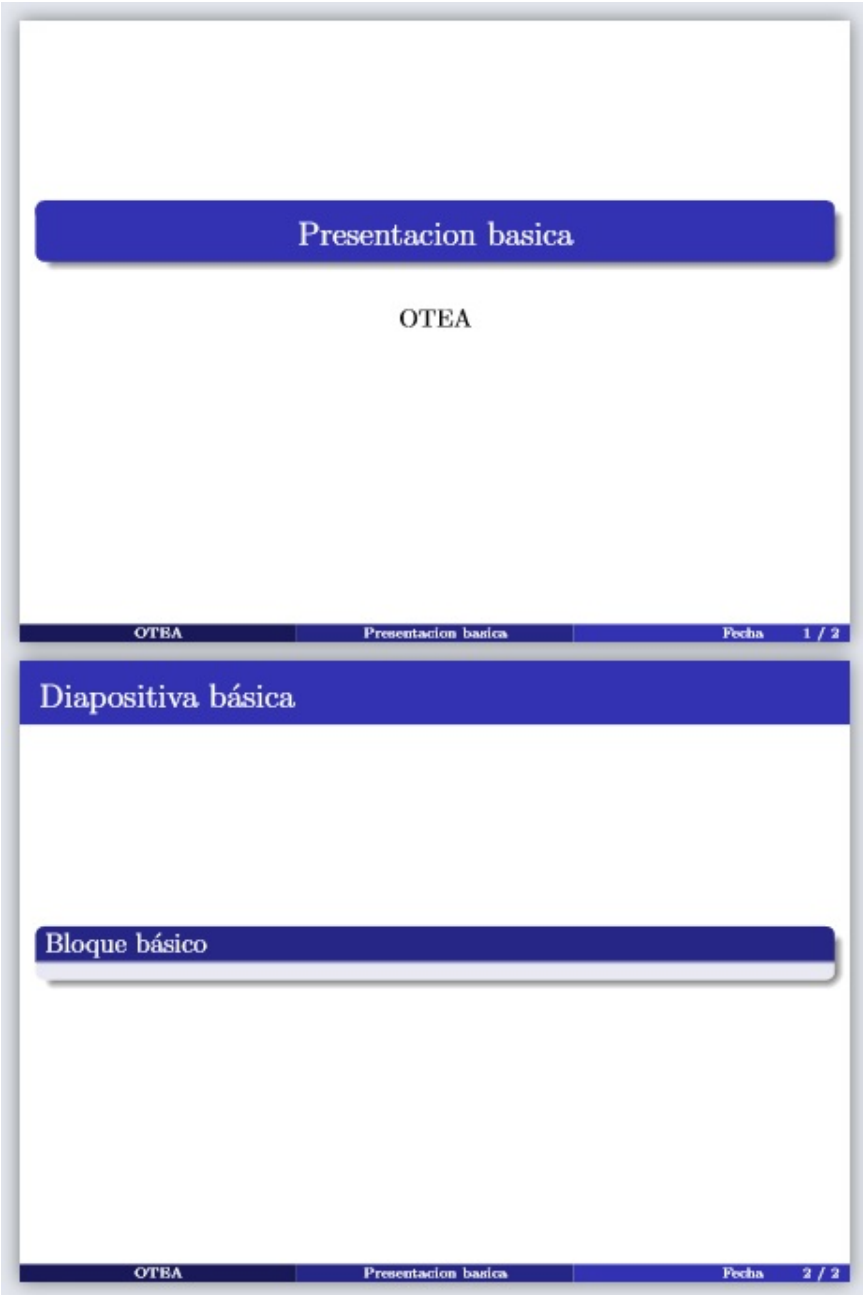


Figura 6.14: Bloque básico

Código 6.11: Código Beamer bloque básico

```
\begin{frame}{Diapositiva básica}
\begin{block}{Bloque básico}
\end{block}
\end{frame}
```

Además de poder colocar bloques normales, podemos utilizar bloques de alerta con el comando alertblock.

Figura 6.15: Bloque alerta básico

Código 6.12: Código Beamer bloque alerta básico

```
\begin{frame}{Diapositiva básica}
\begin{alertblock}{Bloque alerta básico}
Texto bloque
\end{alertblock}
\end{frame}
```

También tenemos otro tipo de bloque básico, que es el comando exampleblock para poner recuadros de ejemplo.

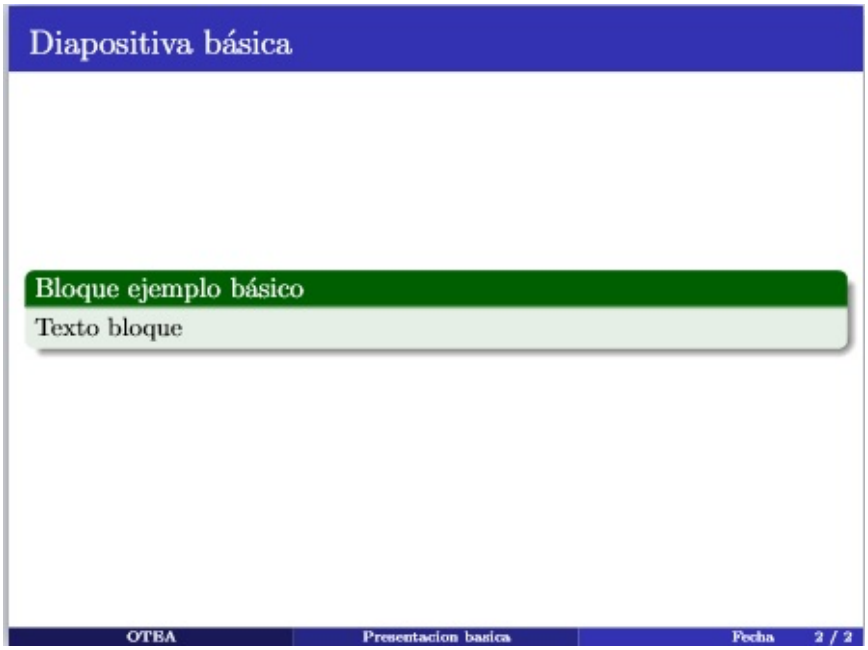


Figura 6.16: Bloque ejemplo básico

Código 6.13: Código Beamer bloque ejemplo básico

```
\begin{frame}{Diapositiva básica}
\begin{exampleblock}{Bloque ejemplo básico}
Texto bloque
\end{exampleblock}
\end{frame}
```

Cajas con color

Para crear cajas con color vamos a utilizar el entorno beamercolorbox, y para definir el color utilizaremos el comando `\setbeamercolor{nombre}{definición color}` en el préambulo.

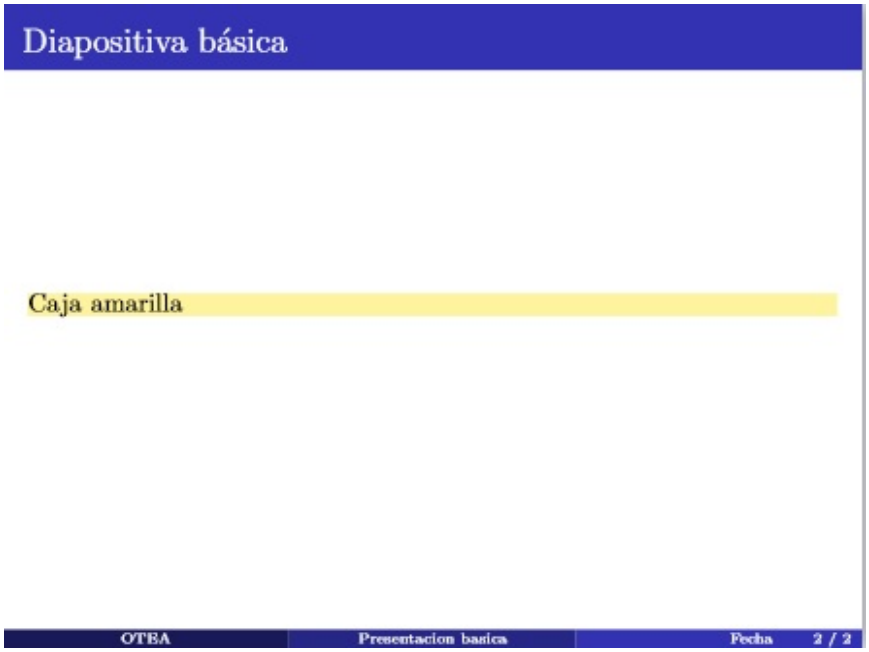


Figura 6.17: Bloque color amarillo

Código 6.14: Código Beamer bloque amarillo

```
%Preambulo main
\documentclass{beamer}
\setbeamercolor{postit}{bg=yellow!50!white}
\begin{document}
\include{Frame/Frame1} %Titulo
\include{Frame/Frame2}
\end{document}
% En la diapositiva
\begin{frame}{Diapositiva básica}
\begin{beamercolorbox}{postit}
Caja amarilla
\end{beamercolorbox}
\end{frame}
```

También tenemos una serie de opciones de alineamiento para las cajas coloreadas que son las siguientes:

- Dimensiones de la caja:
  - **Anchura**: wd.

- **Profundidad**:dp.
- **Altura**:ht.

- **Alineación del contenido en la caja**:
  - left.
  - right.
  - center.

- **Separación entre texto y marco**: \sepDistancia.

6.2.3. Animaciones

Finalmente, en nuestras presentaciones podemos introducir unas animaciones, para que nos vayan saliendo las distintas opciones de distintas formas en la diapositivas. Son más complicadas de programar y se van a explicar detenidamente cada una de ellas.

Generalidad

El primer comando que vamos a explicar es el comando \pause para colocar una pausa.

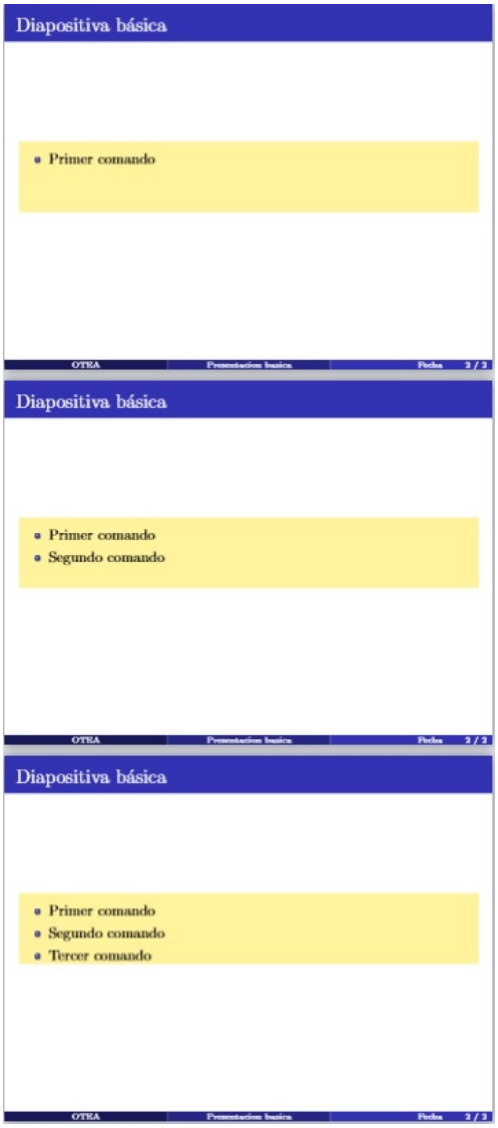


Figura 6.18: Comando pause

Código 6.15: Código comando pause

```
\begin{frame}{Diapositiva básica}
\begin{beamercolorbox}{postit}
\begin{itemize}
    \item Primer comando \pause
    \item Segundo comando \pause
    \item Tercer comando
\end{itemize}
```

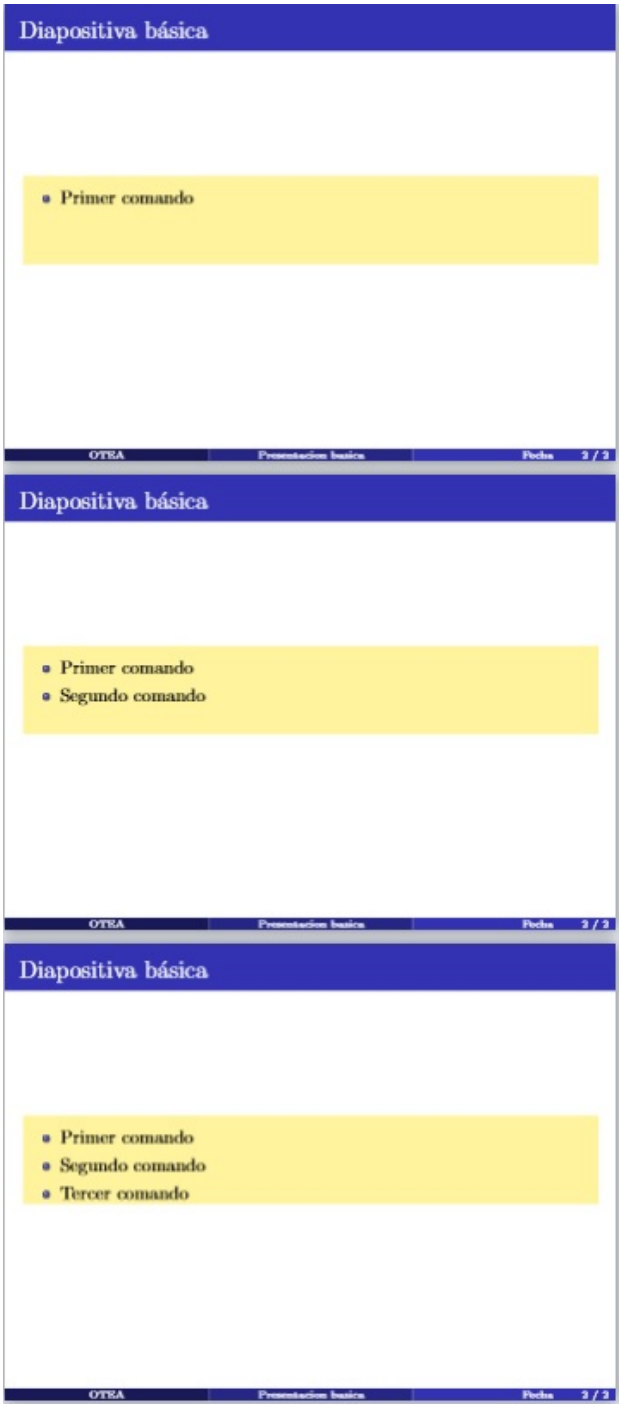


Figura 6.19: Comando pause

Código 6.16: Código comando pause

```
\begin{frame}{Diapositiva básica}
\begin{beamercolorbox}{postit}
\begin{itemize}
    \item Primer comando \pause
    \item Segundo comando \pause
    \item Tercer comando
\end{itemize}
\end{beamercolorbox}
\end{frame}
```

Especificaciones

Hay algunos otros comandos que admiten unas especificaciones que están en el comando `<Rango>` en el que ponemos colocar lo siguiente:

- 1- → Del primero en adelante.
- -3 → Hasta el tercer paso.
- 2-5 → Del segundo al quinto.
- 1-3, 5 → Del primero al tercero, incluyendo el quinto.

El comando que colocaremos será `\Comando<Rango>{Contenido}`.

Comando onslide

Con este comando, seleccionaremos el orden en que queramos que aparezca nuestro contenido. Y lo colocaremos `\onslide<Rango>{Texto}` así.

Figura 6.20: Comando onslide

Código 6.17: Código comando onslide

```
\begin{frame}{Diapositiva básica}
\onslide<2->{Esto se muestra desde el segundo paso}\\
\onslide<1-2>{Esto se muetra entre el primer y segundo paso}\\
Esto siempre.
\end{frame}
```

Comando only

Este comando es similar al anterior, pero las sustituciones a la siguiente diapositiva se realizan de forma animada.

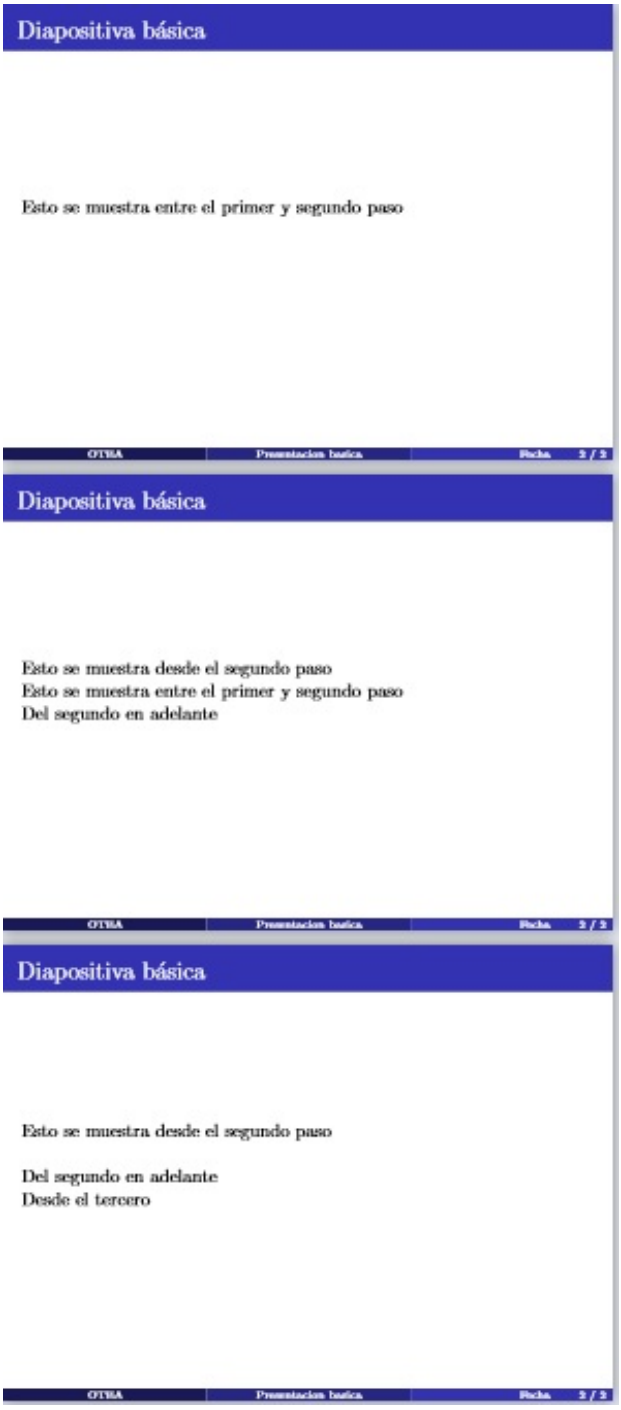


Figura 6.21: Comando only

Código 6.18: Código comando only

```
\begin{frame}{Diapositiva básica}
\only<2->{Esto se muestra desde el segundo paso}\\
\onslide<1-2>{Esto se muestra entre el primer y segundo paso}\\
\onslide<2->{Del segundo en adelante}\\
\only<3>{Desde el tercero}
\end{frame}
```

6.3. Ejercicios resueltos

Ejercicio 1. Realiza un Beamer y un Powerdot básico. Compara sus diferencias en el código.

Figura 6.22: Powerdot básico

Código 6.19: Código ejercicio 1 de Powerdot básico

```
\documentclass{powerdot}
\usepackage[utf8]{inputenc}
\usepackage[spanish]{babel}
\title{Presentación básica}
\author{Pascal}
\date{\today}
\begin{document}
  \maketitle
\begin{slide}{Primera diapositiva}
```

```
\end{slide}
\end{document}
```

Figura 6.23: Beamer básico

Código 6.20: Código ejercicio 1 de Beamer básico

```
\documentclass{beamer}
\usepackage[utf8]{inputenc}
\usepackage[spanish]{babel}
\title{Beamer básico}
\author{Pascal}
\date{\today}

\begin{document}
\frame{\titlepage}
\begin{frame}
\frametitle{Primera diapositiva}
\end{frame}
\end{document}
```

Ejercicio 2. Realiza un bloque básico y un bloque de alerta en una presentación tipo Beamer.

Figura 6.24: Ejercicio 2

Código 6.21: Solución ejercicio 2

```
\documentclass{beamer}
\usepackage[utf8]{inputenc}
\usepackage[spanish]{babel}
\usepackage{block}
\usetheme{Warsaw}
\title{Beamer básico}
\author{Pascal}
\date{\today}

\begin{document}
\frame{\titlepage}
\begin{frame}
\frametitle{Primera diapositiva}
\begin{block}{Primer bloque}
Bloque con texto.
\end{block}
\begin{alertblock}{Bloque aviso}
Bloque aviso con texto.
\end{alertblock}
\end{frame}
\end{document}
```

Ejercicio 3. Cambia el estilo a una presentación de Powerdot, que además se le pueda modificar el color. Y adjúntale algún tipo de animación o de código.

Figura 6.25: Ejericicio 3- Dos primeras diapositivas

Figura 6.26: Ejericicio 3-Segunda diapositiva

Código 6.22: Ejercicio 3

```
\documentclass[
    13pt,
    style=fyma,
    paper=smartboard,
```

```
    orient=landscape,
    ]{powerdot}

\pdsetup{
    trans=Fly,
    palette=blue,
}

\usepackage[utf8]{inputenc}
\usepackage[spanish]{babel}

\usepackage{listings}
\lstnewenvironment{code}{%
\lstset{frame=single,escapeinside='',
backgroundcolor=\color{yellow!20},
basicstyle=\footnotesize \ttfamily}
}{}
\title{Ejercicio 3}
\author{Pascal}
\date{\today}

\begin{document}
  \maketitle
\begin{slide}{Primera diapositiva}
Un texto cualquiera. \pause \\
Texto seguido.
    \begin{itemize}[type=1]
      \item<2> Uno.
      \item<3> Dos.
    \end{itemize}
\end{slide}
\end{document}
```

Ejercicio 4. Realiza el ejercicio anterior, pero añadiéndole un código. Cambia el estilo de la presentación para que sea distinto al del ejercicio anterior.

Figura 6.27: Ejercicio 4

Código 6.23: Ejercicio 4

```
\documentclass[
    13pt,
    style=klope,
    display=slidesnotes,
    paper=smartboard,
    orient=landscape,
    ]{powerdot}
\pdsetup{
    trans=Split,
    palette=Spring,
}

\usepackage[utf8]{inputenc}
\usepackage{listings}
\lstnewenvironment{code}{%
\lstset{frame=single,escapeinside='',
backgroundcolor=\color{yellow!20},
basicstyle=\footnotesize \ttfamily}
}{}
\title{Ejercicio 4}
\author{Pascal}
\date{\today}

\begin{document}
  \maketitle
```

```
\begin{slide}[method=direct]{Primera diapositiva}
Ejemplo de código:
\begin{code}

compute x;

compute x+y;
\end{code}
\end{slide}
\end{document}
```

Ejercicio 5. Realiza un Beamer que contenga alguna animación en un texto o en un itemize.

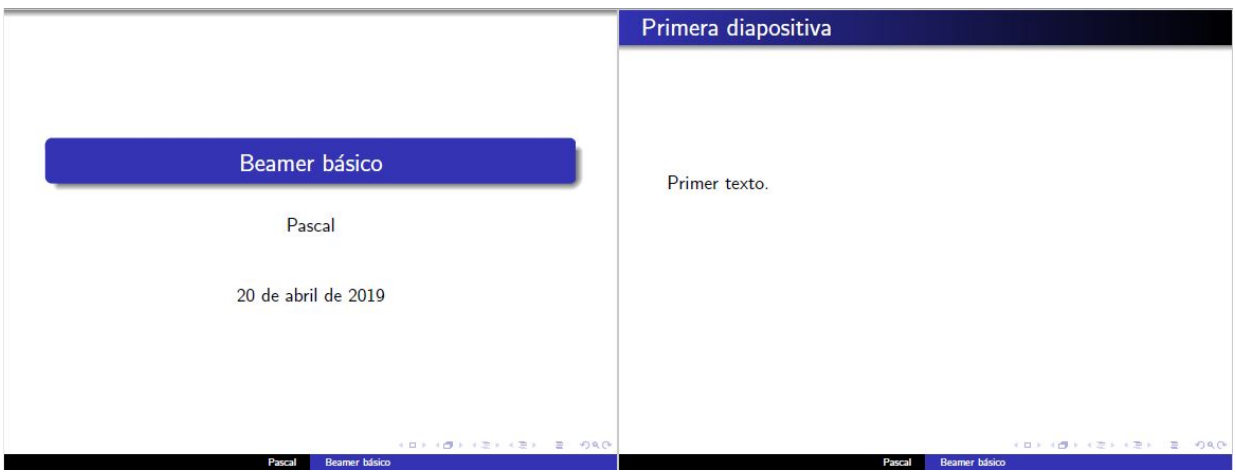


Figura 6.28: Ejercicio 5- Dos primeras diapositivas

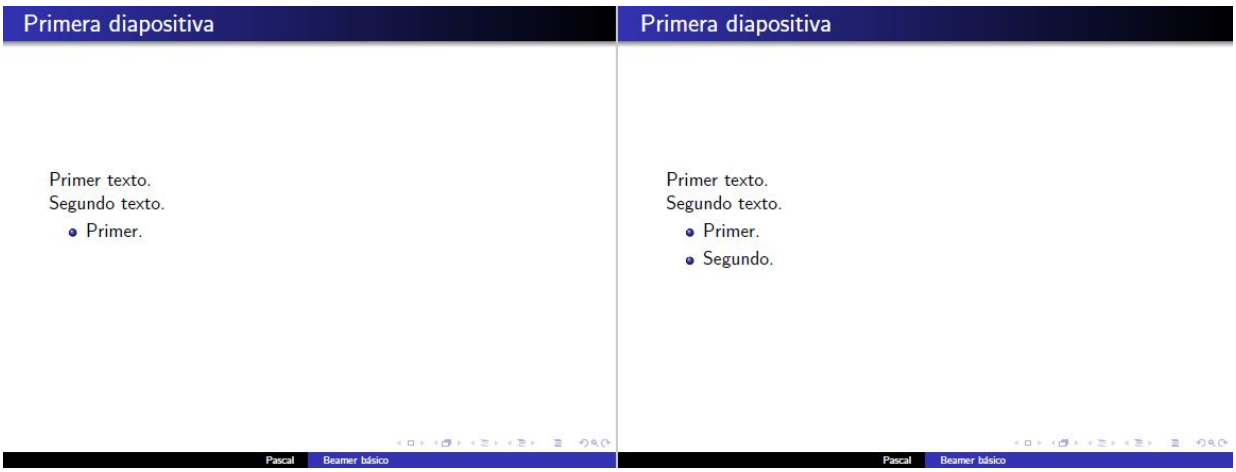


Figura 6.29: Ejercicio 5- Dos últimas diapositivas

Código 6.24: Ejercicio 5

```
\documentclass{beamer}
\usepackage[utf8]{inputenc}
\usepackage[spanish]{babel}

\usetheme{Warsaw}
\title{Beamer básico}
\author{Pascal}
\date{\today}

\begin{document}
\frame{\titlepage}
\begin{frame}
\frametitle{Primera diapositiva}
Primer texto.\\ \pause
Segundo texto.
\begin{itemize}
    \item<2-3> Primer.
    \item <3> Segundo.
\end{itemize}
\end{frame}
\begin{frame}{Segunda diapositiva}
\end{frame}
\end{document}
```

Ejercicio 6. Realiza el mismo ejercicio anterior, pero en un entorno Powerdot. Vuelve a cambiar el estilo del Powerdot junto con su color incluido. Y añádele una anotación.

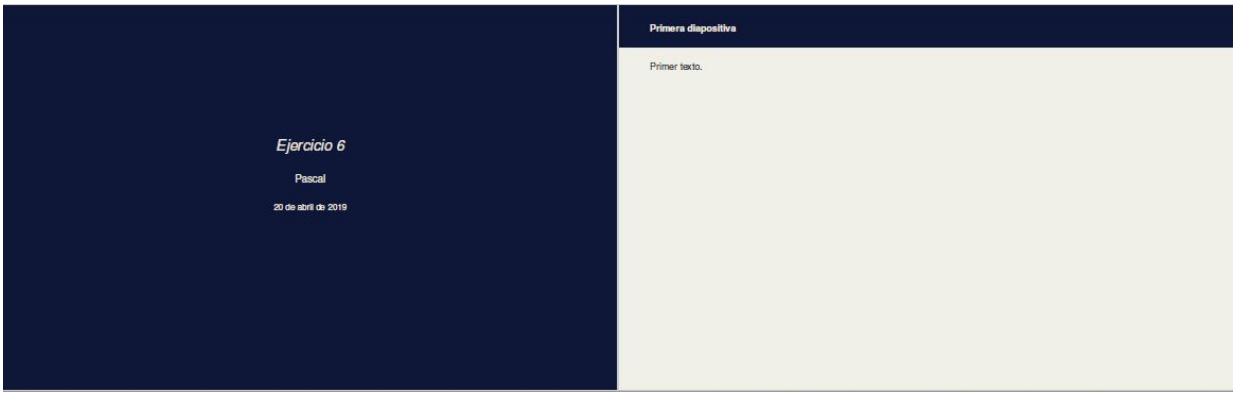


Figura 6.30: Ejercicio 6- Dos primeras diapositivas

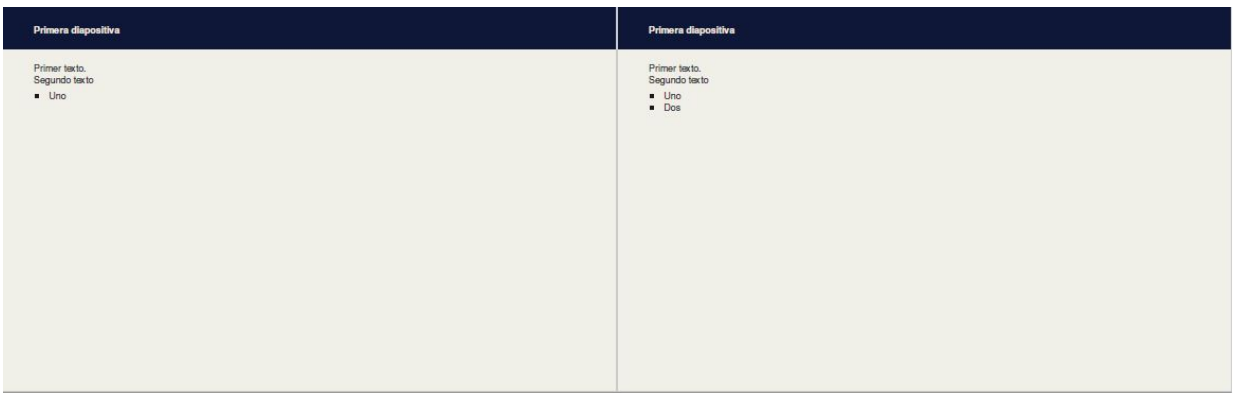


Figura 6.31: Ejercicio 6- Dos últimas diapositivas

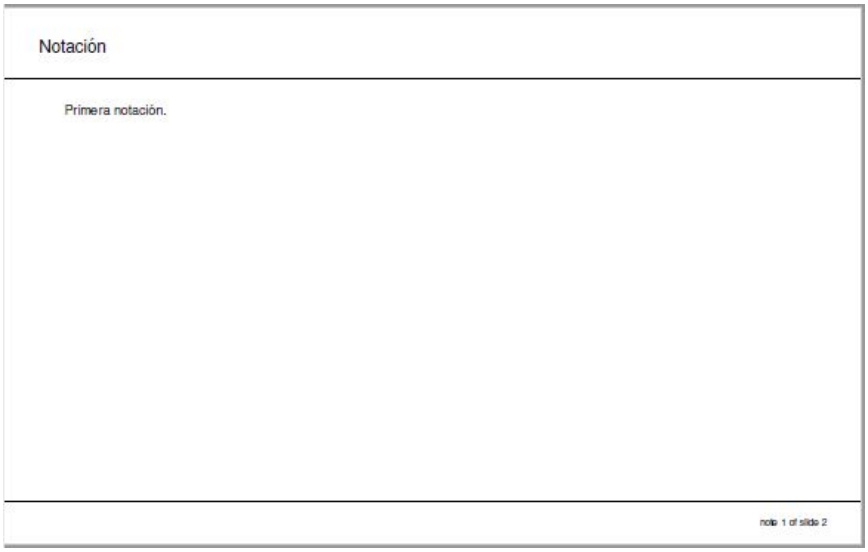


Figura 6.32: Ejercicio 6- Notación

Código 6.25: Ejercicio 6

```
\documentclass[
    13pt,
    style=paintings,
    display=slidesnotes,
    paper=smartboard,
    orient=landscape,
    ]{powerdot}

\pdsetup{
    trans=Box,
    palette=GoldenGate,
}
\usepackage[utf8]{inputenc}
\usepackage[spanish]{babel}
\usepackage{listings}
\lstnewenvironment{code}{
\lstset{frame=single,escapeinside='',
backgroundcolor=\color{yellow!20},
basicstyle=\footnotesize \ttfamily}
}{}

\title{Ejercicio 6}
```

```
\author{Pascal}
\date{\today}

\begin{document}
  \maketitle
\begin{slide}{Primera diapositiva}
Primer texto. \pause \\
Segundo texto
    \begin{itemize}
      \item Uno \pause
    \item Dos
    \end{itemize}
\end{slide}
\begin{note}{Notación}
Primera notación.
\end{note}
\end{document}
```

7

Creación de comandos, entornos y ambientes

En este capítulo aprenderemos a crear contadores, a realizar operaciones con ellos, definiremos el contador declaración, nombraremos y renombramos comandos, y por ultimo, nombraremos y renombraremos entornos.

1. Crear contadores.
2. Operaciones con contadores.
3. Contador declaración.
4. Comandos newcommand y renewcommand.
5. Comandos newenvieroment y renewenvieroment.

7.1. Crear contadores

Antes de entrar a saber qué comandos se utilizan en LaTeX vamos a definir el concepto de contador. El contador es una variable cuyo valor se incrementa o decrementa en un valor fijo. Una vez hemos definido el concepto de contador vamos a ver los comandos que son contadores en LaTeX:

- `\newcounter{variable}`: Crea un contador que en este caso se llama variable y empieza por 0. Este valor se define en el preámbulo.
- `\setcounter{variable}{1}`: Da a la variable valor 1. Este valor se define en el cuerpo del documento.
- `\addtocounter{variable}{1}`: Suma 1 a la variable. Este valor se define en el cuerpo del documento.

A este contador se le puede dar un valor numérico, ya sea en letras o en números. Los comandos que lo pueden definir son los siguientes:

- `\arabic{variable}`: Valor en números.
- `\roman{variable}`, `\Roman{variable}`: Números romanos en minúsculas y mayúsculas.
- `\alph{variable}`, `\Alph{variable}`: Letras en minúsculas y mayúsculas.

Vamos a poner un ejemplo con cada uno de este tipo de comandos con los contadores:
Te lo digo 3 veces.
Te lo digo 4 veces.

Código 7.1: Código de contador simple

```
\newcounter{x}
\addtocounter{x}{3} % sumo 3
Te lo digo \arabic{x} veces.\\
\addtocounter{x}{1} % sumo 1
Te lo digo \arabic{x} veces.
```

Así ponemos MD el número en números romanos.
Así ponemos otro año en números romanos como MMC.

Código 7.2: Código de contador en números romanos

```
\newcounter{y}
\setcounter{y}{1000}
\addtocounter{y}{500}
```

```
Así ponemos \Roman{y} el número en números romanos.\\
\addtocounter{y}{600}
Así ponemos otro año en números romanos como \Roman{y}.
```

También podemos colocar la letra f como contador.
Por otro lado podemos colocar la letra g como contador.

Código 7.3: Código de contador en letras

```
\newcounter{a}
\newcounter{b}
\setcounter{a}{1}
\setcounter{b}{2}
\addtocounter{a}{5}
También podemos colocar la letra \alph{a} como contador.\\
\addtocounter{b}{5}
Por otro lado podemos colocar la letra \alph{b} como contador.
```

7.2. Operaciones con contadores

Como hemos visto en los ejemplos previos, los contadores se pueden sumar y dan lugar a un nuevo valor de variables. Pero no sólo se puede sumar los contadores, sino también se pueden restar, multiplicar e incluso dividir. A continuación vamos a ver los comandos utilizados para cada una de las operaciones:

- `\addtocounter{variable}{n}`: Donde la variable es el contador y n es un entero positivo. Este comando es la suma.
- `\addtocounter{variable}{-n}`: Donde la variable es el contador y n es un entero positivo. Este comando es la resta.
- `\multiply\value{variable} by n`: Donde la variable es el contador y n es un entero positivo. Este comando es la multiplicación.
- `\divide\value{variable} by n`: Donde la variable es el contador y n es un entero positivo. Este comando es la división.

Ahora mostramos el número de la suma con el primer contador: 14.
Ahora mostramos el número de la resta con el segundo contador: 10.
Ahora mostramos la multiplicación con el primer contador: 42.
Ahora mostramos la división con el segundo contador: 5.

Código 7.4: Ejemplo de todas las operaciones con contadores

```
\newcounter{u}
\newcounter{v}
\setcounter{u}{4}
\addtocounter{u}{10}
Ahora mostramos el número de la suma con el primer contador: \
   arabic{u}.\\
\setcounter{v}{20}
\addtocounter{v}{-10}
Ahora mostramos el número de la resta con el segundo contador: \
   arabic{v}.\\
\multiply\value{u} by 3
Ahora mostramos la multiplicación con el primer contador: \arabic{u
   }.\\
\divide\value{v} by 2
Ahora mostramos la división con el segundo contador: \arabic{v}.
```

Ahora vamos a mostrar el resultado en letras: G.
Ahora mostramos el resultado en números: 2000.
Ahora mostramos el resultado en números romanos: XX.
Ahora mostramos el resultado en números otra vez: 56.
Ahora mostramos el resultado en números: 2000.
Ahora mostramos el resultado en números romanos: X.

Código 7.5: Otro ejemplo con operaciones

```
\newcounter{t}
\newcounter{c}
\newcounter{d}
\setcounter{t}{1000}
\setcounter{c}{5}
\setcounter{d}{80}
\addtocounter{c}{2}
Ahora vamos a mostrar el resultado en letras: \Alph{c}.\\
\multiply\value{t} by 2
Ahora mostramos el resultado en números: \arabic{t}.\\
\divide\value{d} by 4
Ahora mostramos el resultado en números romanos: \Roman{d}.\\
\multiply\value{c} by 8
Ahora mostramos el resultado en números otra vez: \arabic{c}.\\
\divide\value{d} by 2
```

```
Ahora mostramos el resultado en números: \arabic{t}.\\
\addtocounter{c}{-8}
Ahora mostramos el resultado en números romanos: \Roman{d}.
```

7.3. Contador declaración

En este capítulo vamos a repasar los dos capítulos anteriores. Primero vamos a definir nuestro contador con `\newcounter{variable}`, donde variable es el nombre del contador. Es importante que los corchetes sólo contengan letras para definir el contador. Una vez que la hemos definido la podremos mostrar con el comando `\thevariable`. El valor inicial de la variable es 0. Le podremos dar un valor a la variable con el comando `\setcounter{variable}{n}`, donde n es un número entero. Con el comando `\value{variable}` es una función que nos muestra el valor inicial de la variable.
0
5
Con LaTeX podemos utilizar todo tipo de contadores:
10
J
j
x
X

Código 7.6: Código contador declaración

```
\newcounter{variable}
\thevariable\\
\setcounter{variable}{5}
\thevariable\\
\addtocounter{variable}{5}
Con \LaTeX{} podemos utilizar todo tipo de contadores:\\
\arabic{variable}\\
\Alph{variable}\\
\alph{variable}\\
\roman{variable}\\
\Roman{variable}
```

10
Ahora vemos la suma 20.
Ahora lo multiplicamos, así 40.

Código 7.7: Código contador declaración

```
\newcounter{r}
\setcounter{r}{10}
\ther\\
\addtocounter{r}{10}
Ahora vemos la suma \ther.\\
\multiply\value{r} by 2
Ahora lo multiplicamos, así \ther.
```

7.4. Comandos newcommand y renewcommand

LaTeX tiene una gran cantidad de comandos, por lo que muchas veces repetiremos ese comando una y otra vez. Para poder acortar esos comandos podemos definir nuevos comandos en el preámbulo con el comando `\newcommand{nombre de comando}{expresión del comando}`. A continuación vamos a mostrar varios ejemplos de este comando:

En un documento se pueden ver los distintos elementos: $\alpha\beta\Gamma$

Lista de ejemplo:

- Primero

§ Segundo

Podemos colocar la letra de los números reales en negrita: $\mathbb{R}$.

Números complejos $\mathbb{C}$, racionales $\mathbb{Q}$ e integrales $\mathbb{Z}$.

Si tenemos que realizar muchos exponenciales podremos realizar lo siguiente:

$$(x+y)^2$$

Y cuando cambiamos el exponente:

$$(y+y)^4$$

El ∞)es a veces representado como $\mathbb{S}$

Figura 7.1: Ejemplo newcommand

Código 7.8: Ejemplo newcommand

```
\documentclass{article}
\usepackage{amssymb}
\usepackage[utf8]{inputenc}
\usepackage[spanish]{babel}

\newcommand{\R}{\mathbb{R}}
\newcommand{\bb}[1]{\mathbb{#1}}
\newcommand{\plusbinomial}[3][2]{(#2 + #3)^#1}
```

```
\begin{document}

En un documento se pueden ver los distintos elementos: $\alpha \
   beta \Gamma$


\vspace{1cm}

Lista de ejemplo:
\begin{itemize}
\item Primero
\item[\S] Segundo
\end{itemize}


\vspace{1cm}

Podemos colocar la letra de los números reales en negrita: \( \R \)
   .


\vspace{1cm}

Números complejos \( \bb{C} \), racionales \( \bb{Q} \) e
   integrales \( \bb{Z} \).


\vspace{1cm}

Si tenemos que realizar muchos exponenciales podremos realizar lo
   siguiente:
\[ \plusbinomial{x}{y} \]
Y cuando cambiamos el exponente:
\[ \plusbinomial[4]{y}{y} \]
\vspace{1cm}
\renewcommand{\S}{\mathbb{S}}
El $\infty$)es a veces representado como \( \S \)
\end{document}
```

Como podemos ver dentro del código hay un comando con un valor opcional donde ese valor indica el valor del nuevo comando. Pero nosotros sólo nos quedaremos con el comando

ya simple, ya que, los comandos opciones son muy complicados. Seguidamente, vamos a mostrar el otro ejemplo:

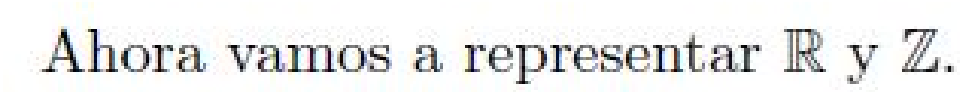


Figura 7.2: Ejemplo newcommand

Código 7.9: Código ejemplo newcommand

```
\documentclass{article}
\usepackage{amssymb}
\usepackage[utf8]{inputenc}
\usepackage[spanish]{babel}

\newcommand{\R}{\mathbb{R}}
\newcommand{\Z}{\mathbb{Z}}

\begin{document}

Ahora vamos a representar $\R$ y $\Z$.

\end{document}
```

Por otro lado, si vemos que nos da problemas algun comando y nos da un mensaje de error necesitaremos volver a definir ese comando con el comando `\renewcommand{nuevo comando}{lo que hace}`, para tenerlo más claro ponemos un ejemplo:

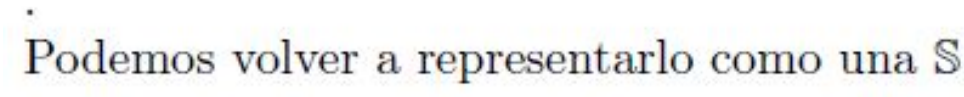


Figura 7.3: Código renewcommand

Código 7.10: Código renewcommand

```
\renewcommand{\S}{\mathbb{S}}
Podemos volver a representarlo como una  \( \S \)
```

7.5. Comandos newenvieronment y renewenvieroment

Al igual que con los comandos, podemos definir nuevos entornos. Para ello utilizaremos el comando `\newenvironment{nuevo nombre}{entorno anterior}`. Vamos a ver un ejemplo nombrando un nuevo entorno:

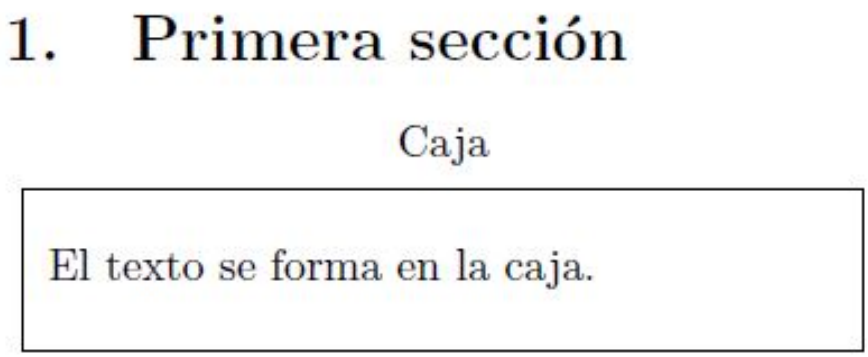


Figura 7.4: Comando newenvieronment

Código 7.11: Código newenvieronment

```
\documentclass{article}
\usepackage[utf8]{inputenc}
\usepackage[spanish]{babel}
\usepackage{geometry}
\usepackage{comment}
\geometry{textwidth = 7cm}
\usepackage{microtype}
%Nuevo entorno
\newenvironment{boxed}[1]
    {\begin{center}
    #1\\[1ex]
    \begin{tabular}{|p{0.9\textwidth}|}
    \hline\\
    }
    {
    \\\\\hline
    \end{tabular}
    \end{center}
    }
\begin{document}
```

```
\section{Primera sección}
\begin{boxed}{Caja}
El texto se forma en la caja.
\end{boxed}
\end{document}
```

Finalmente, podremos sobreescribir los entornos con el comando `\renewenvironment{entorno}{entorno a reescribir}`. Vamos a ver cómo se sobreescriben los entornos ya existentes con un ejemplo:

Entorno con el texto centrado.

Figura 7.5: Comando renewenvironment

Código 7.12: Código renewenvironment

```
\documentclass{article}
\usepackage[utf8]{inputenc}
\usepackage[spanish]{babel}
\usepackage{geometry}
\usepackage{comment}
\geometry{textwidth = 7cm}
\usepackage{microtype}


\begin{document}


\renewenvironment{itemize}
{\begin{center}\em}
{\end{center}}

\begin{itemize}
Entorno con el texto centrado.
\end{itemize}
\end{document}
```

8

Condicionales

En este capítulo aprenderemos macros avanzados, condicionales y bucles generales.

1. Macros avanzados: Argumentos.
2. Condicionales ifthen.
3. Bucles generales.

8.1. Macros avanzadas: Argumentos

Una vez visto el tema de la creación de comandos, entornos y ambientes básicos vamos a pasar al sistema avanzado. En muchas ocasiones vamos a requerir de un sistema experto que recopile variables de entorno para poder modificar ciertas funciones. Aquí entran los argumentos, una opción muy poderosa de LaTeX cuando creamos entornos avanzados. Si bien, hemos definido en el libro de conocimientos básicos de LaTeX que este lenguaje es un conjunto de macros, ahora vamos a realizar las opciones avanzadas.

Para crear un entorno avanzado podemos realizar 3 tareas:

1. Manipulación de LaTeX2 para crear tus propios ambientes y usar ese "nucleo" como entorno controlado con tus macros.

2. Creación de entorno, comando o ambiente tal y como hemos visto en el capítulo anterior.

3. Destruir y renombrar un comando ya existente. De esta forma, por ejemplo, se han creado varias plantillas para adaptarlas a personas que no sabían cierto lenguaje, renombrando del inglés al castellano.

Vamos a centrarnos sobre todo en el punto 2, es la mejor forma de crear macros avanzadas sin destruir medio documento intentándolo. Recordemos que para realizar una creación de comando, tenemos que usar el siguiente código `\newcommand{COMANDO}{DEFINICIÓN}` pero ahora vamos a usar lo siguiente `\newcommand{COMANDO}[ARGUMENTOS]{DEFINICIÓN}` teniendo en cuenta que:

- **Comando:** es el nombre del comando, para ser llamado deberá tener `\` delante del nombre citado.

- **Argumentos:** esto es lo nuevo, argumentos desde el 0 hasta el 9. Si queremos hacer monstruosidades de 10 argumentos tendremos que encadenar creación de comandos o usar paquetes que lo permitan.

- **Definición:** Aquí se pondrá lo que realiza el comando.

Vamos a realizar un ejemplo sencillo de prueba, un comando que genere un texto cuando escribimos su comando. Para ello escribimos el siguiente código:

Código 8.1: Código simple de creación de comandos

```
1 \newcommand{\coso}{Ejemplo de texto, saludos lector}
```

Como podemos observar, el nuevo comando generado se llama `\coso` para invocar el texto resultante, solo tendremos que escribir `\coso`:

Ejemplo de texto, saludos lector

Pero esto sigue siendo macro básica, vista en el capítulo anterior. Si en lugar de esto, queremos que salude a cada persona independientemente de cómo se llame, solo recopilando un dato, necesitamos argumentos:

Código 8.2: Código simple de creación de comandos con argumentos

```
1 \newcommand{\cosoArgumento}[1]{Ejemplo de texto, saludos #1}
```

Observamos que tenemos un comando con 1 argumento, para poder invocarlo de forma correcta, tal y como hemos programado, necesitamos escribir el comando así: `\cosoArgumento{Nombre_que_queramos}` por ejemplo si escribimos `\cosoArgumento{Pascal}` invocamos esto:

Ejemplo de texto, saludos Pascal

Esto es interesante a la hora de realizar plantillas avanzadas, para ello se suele usar entornos de condicionales y bucles.

Una cosa interesante cuando creamos argumentos es la definición por defecto, que raras veces se usa `\newcommand{COMANDO}[ARGUMENTOS][VALOR_DEFECTO]{DEFINICIÓN}` este valor por defecto es el que tomará el valor del argumento si no se especifica.

Código 8.3: Código de creación con valor por defecto

```
1 \newcommand{\cosoDefecto}[1][Tercero]{Ejemplo de texto, saludos #1}
```

Si invocamos `\cosoDefecto[Ejemplo]` con un valor aleatorio este escribirá:

Ejemplo de texto, saludos Ejemplo

Si en lugar de eso, escribimos `\cosoDefecto` tendremos:

Ejemplo de texto, saludos Tercero

Como podemos observar, este comando necesita `[ ]` en lugar de `{ }` si escribimos este comando con `{ }` destruiremos el comando entero y saldrá el valor por defecto junto con el valor dado. Observemos `\cosoDefecto{Ejemplo}`

Ejemplo de texto, saludos TerceroEjemplo

Intentaremos recordar esto, es importante a la hora de realizar valores por defecto de forma correcta. De la misma forma no podemos usar de la forma que queramos los valores de argumentos sin defecto `\cosoArgumento[PascalTercero]` generará:

Ejemplo de texto, saludos [PascalTercero]

Esto indica que recoge todo el parámetro como valor. Hay que tener cuidado cuando lo invoquemos.

Todo esto se puede aplicar a creación de ambientes de forma avanzada. Hasta ahora usábamos la creación de ambientes de esta forma `\newenvironment{ENTORNO}{ANTES}{DESPUÉS}` donde tenemos:

1. **Entorno:** Es el nombre de comando para invocar el nuevo ambiente.
2. **Antes:** Aquí se escribe todos los comandos que se invocarán antes de comenzar el entorno o ambiente.
3. **Después:** Todo lo que se activará después del texto.

Ahora que sabemos como realizar estos ambientes básicos, vamos a realizar un ejemplo básico como los vistos en el tema anterior.

Código 8.4: Código de creación de ambiente básico

```
\newenvironment{ambiente}
  {\vspace{1ex}\hrule\textbf{EJEMPLO DE AMBIENTE}}
  {\vspace{1ex}\hrule}
```

Cuando lo invocamos como ambiente da como resultado:

EJEMPLO DE AMBIENTE Texto de ejemplo.

Código 8.5: Código de ambiente

```
\begin{ambiente}
    Texto de ejemplo.
\end{ambiente}
```

Como podemos observar este ambiente crea un par de lineas horizontales con un texto estándar como parámetro **“Antes”**, posteriormente se pone el texto o los comandos que hayamos puesto dentro del ambiente y para finalizar se pone todo lo escrito en el campo **“Después”**.

Pero esto es muy básico, queremos realizar ambientes avanzados con argumentos. Para esto usaremos la siguiente sitaxis `\newenvironment{ENTORNO}[ARGUMENTOS]{ANTES}{DESPUÉS}`

Código 8.6: Código de creación de ambiente con argumentos

```
\newenvironment{ambienteDos}[1]
  {\vspace{1ex}\hrule\textbf{#1}}
  {\vspace{1ex}\hrule}
```

De esta forma, ahora podemos invocar el ambiente con un argumento, dejando el siguiente texto:

Nota destacada Texto de ejemplo.

Código 8.7: Código de ambiente

```
\begin{ambienteDos}{Nota destacada}
    Texto de ejemplo.
\end{ambienteDos}
```

De esta forma, podemos reescribir ambientes y comandos, crearlos y ahora con argumentos. Los argumentos más avanzados normalmente nos harán realizar toma de decisiones, por ejemplo, si queremos que un documento se imprima el comando correcto es la creación de un documento de dos páginas, mientras que si queremos leerlo en PDF con que se muestre todo en una página. Para ello, puede que nosotros con conocimientos medios de LaTeX sepamos los comandos para cambiar un documento de uno a otro. Pero una persona que no está acostumbrada o que tiene un uso básico no podrá. Por eso se crean las plantillas, normalmente comandos y ambientes creados para facilitar el uso a personas con menos conocimientos o para hacer sencillas las tareas repetitivas. En este libro sembraremos los conocimientos básicos para crear plantillas profesionales en un futuro.

Uno de los mecanismos de creación de plantillas profesionales es hacer uso de esas decisiones en condicionales. Para ello vamos a aprender un par de mecanismos de creación de condicionales y bucles.

8.2. Condicionales ifthen

Antes de comenzar con los condicionales tenemos que agregar el siguiente paquete en el preámbulo `\usepackage{ifthen}`. Y ahora vamos a explicar el recorrido básico de un condicional.

IF
Código si se cumple la condición del IF
ELSE
Código si no se cumple la condición del IF

De esta forma vamos a poner un ejemplo:

IF (Llueve):Meter la ropa
ELSE: Saca la ropa al patio

De esta forma sencilla, solo meteremos la ropa en casa si llueve. En caso contrario la sacaremos al patio. Ahora vamos a poner un ejemplo con código real. Para ello usaremos la creación de un comando nuevo con argumentos.

Código 8.8: Código de ifthen básico

```
\newcommand{\printDiaNoche}[1]{
  \ifthenelse{\equal{#1}{true}}{dia}{}
  \ifthenelse{\equal{#1}{false}}{noche}{}
}
\printDiaNoche{true}
```

Cuando lo invocamos a **true** el valor arrojado es:

dia

Cuando lo invocamos a **false** el valor arrojado es:

noche

Aunque si lo comprobamos, este comando se puede abreviar muchísimo, ya que la sentencia ifthenelse se compone de los siguientes apartados
`\ifthenelse{PRUEBA CONDICIONAL}{EFECTO TRUE}{EFECTO FALSE}` Teniendo como componentes:

- **Prueba Condicional:** Un estamento que se evalúa para saber si se cumple o no. Si se cumple, se realizará el efecto de **TRUE**, si no se cumple se realizará el efecto de **FALSE**.
- **Efecto TRUE:** Se realizará este efecto si se cumple la prueba condicional.
- **Efecto FALSE:** Se realizará si no se cumple la condición de **TRUE**.

De esta forma, vamos a re-implementar el código para reducirlo:

Código 8.9: Código de ifthen reducido

```
\newcommand{\printRedux}[1]{
  \ifthenelse{\equal{#1}{true}}{dia}{noche}}
```

Ahora para poder invocar esto usaremos directamente `\printRedux{}` con la condición que queramos. Cuando lo invocamos el redux a **true** el valor arrojado es:

dia

Cuando lo invocamos de cualquier otra forma, arroja:

noche

8.2.1. Proposiciones atómicas

Es importante conocer estas proposiciones para poder generar condicionales de forma correcta. Bien usados estos comandos sirven para realizar comparaciones de forma eficiente.

- `{numero 1}<{numero 2}` se vuelve true si el número 1 es menor que el 2.
- `{numero 1}>{numero 2}` se vuelve true si el número 1 es mayor que el 2.
- `\isodd{numero}` se valora a true si el número es impar.
- `\isundefined{nombre de comando}` este comando se valora true si el comando que se evalua no existe.
- `\equal{valor 1}{valor 2}` este es el que más se usa. En el ejemplo anterior lo hemos usado. Se vuelve true si el valor 1 es igual que el valor 2. Se puede usar p̄ara los números pero siempre es mejor coger la costumbre de usar el equal.

Hay muchos comandos avanzados para la realización de bucles. Consultar `https://osl.ugr.es/CTAN/macros/latex/base/ifthen.pdf` de la página del CTAN que contiene todos los paquetes para conseguir conceptos avanzados. Pero para la realización de condicionales para plantillas básicas, estos conceptos son suficientes.

8.2.2. Ejemplo de condicionales en plantillas

Un gran ejemplo de explicación de configuración de condicionales para plantillas es el uso de TeFloN:2.0[2019][CC-0] que se va a explicar a continuación.

Código 8.10: Comienzo de TeFloN 2.0

```
1 \documentclass{TeFlon}
2 \begin{document}
3 \incorporarDatos{si}{UCM}{Informática}{Ingeniería Informática}{
    Nombre del tutor}{2018--2019}{Madrid}{Pascal, tercero}{TeFloN:
    Plantilla para TFGs}
4 \tnr{no}
```

Podemos observar 2 comandos creados:

- `\incorporarDatos` con 9 campos (los máximos).
- `\tnr` con 1 campo.

Dentro del campo `\incorporarDatos` muchos de sus argumentos, son condicionales que se pueden observar en la plantilla profesional. En este caso vamos a explicar el primer campo, el cual expresa con un **si** o un **no** si se quiere el documento para imprimir.

Código 8.11: ifthen de TeFloN

```
\newcommand*{\incorporarDatos}[9]{
\ifthenelse{\equal{#1}{si}}{\imprimir{si}}{\imprimir{no}}
\ifthenelse{\equal{#2}{UCM}}{
\newcommand*{\print}{Images/logo_UCM.jpg}
\newcommand*{\facultad}{Facultad de #3}
\newcommand*{\grado}{Grado en #4}
\newcommand*{\tutor}{#5}
\newcommand*{\curso}{#6}
\newcommand*{\ciudad}{#7}
\newcommand*{\npersona}{#8}
\newcommand*{\ntrabajo}{#9}
}{}

}
```

Código 8.12: Condicional de impresión

```
\newcommand*{\imprimir}[1]{
\ifthenelse{\equal{#1}{si}}{\newgeometry{margin=1in, twoside}}{}
  }
```

Este es un buen ejemplo para la creación de condicionales. Esta plantilla se usa de forma profesional y observamos lo sencillo que es su código una vez sabemos qué hace cada comando. Como nota importante, existen muchísimos medios de creación de bucles en LaTeX este es uno de los más avanzados y que su sintaxis no es tan ofuscada, pero se puede usar cualquiera.

8.3. Bucles generales

Se pueden realizar bucles en LaTeX pero jamás serán tan avanzados como los que se pueden realizar en otros lenguajes de programación. Para ello, tenemos que entender antes de comenzar, que todos los comandos que incluyan un símbolo "@" deben estar comprendidos entre `\makeatletter` y `\makeatother`

8.3.1. Bucle for

El comando para realizar un bucle for es el siguiente

Código 8.13: Bucle for

```
\@for\command:={list}\do{commands}
```

Este bucle, primero crea una variable en un comando especial, ordenando estos comandos tal y como se han escrito. También se pueden hacer operaciones con contadores e incrementos. Posteriormente todo lo listado se despliega en la segunda parte. Veamos un ejemplo:

Código 8.14: Bucle for, ejemplo

```
\makeatletter
\@for\basura:={Lunes,Miércoles}\do{El \basura{} es de los peores dí
   as de la semana. }
\makeatother
```

El Lunes es de los peores días de la semana. El Miércoles es de los peores días de la semana. Como truco, agregamos {} al final de los comandos para agregar un espacio en blanco para poder continuar con la frase.

8.3.2. Bucle while-num

Uno de los bucles más útiles para realizar cálculos e iteraciones en base numérica. Vamos a ver un ejemplo para profundizar en esto.

Código 8.15: Bucle while-num

```
\makeatletter
\newcounter{int}
\@whilenum\value{int}<27\do
{\stepcounter{int}\theint, }
\makeatother
```

Como podemos observar, al comienzo, creamos una variable de tipo entero (contador) y después seguimos la secuencia de while. La condición de continuidad del bucle es que el valor del entero sea menor de 27. Y lo que hacemos es incrementar el contador y lo mostramos por pantalla antes. Cuando ejecutamos este bucle sale lo siguiente: 1, 2, 3, 4, 5, 6, 7, 8, 9,10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27,
Gracias a estos bucles y concatenándolos con condicionales, podríamos mostrar solo los números impares por ejemplo.

Código 8.16: Bucle while-num, avanzado

```
\makeatletter
\newcounter{int}
\@whilenum\value{int}<27\do
{\stepcounter{int}\ifthenelse{\isodd{\value{int}}}{\theint, }{}}
\makeatother
```

Generando solo la serie impar: 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27,
En volúmenes superiores, se podrá ver como estos bucles son usados en Tikz para realizar dibujos y gráficos avanzados.

9

Entorno matemático avanzado

En este capítulo repasaremos las matemáticas básicas, matrices y determinantes, aprenderemos a realizar coeficientes en negrita, aprenderemos a destacar símbolos en negrita, a poner símbolos encima o debajo de símbolos, definiremos nuevos comandos, realizaremos división de fórmulas, forzaremos el cambio de numeración, aprenderemos a manejar el entorno subequations, realizaremos diagramas conmutativos y realizaremos ejercicios para repasar los conceptos anteriores.

1. Repaso de matemáticas básicas.
2. Matrices y determinantes.
3. Coeficientes binomiales.
4. Símbolos en negrita.
5. Símbolos encima de símbolos.
6. Definición de nuevos comandos.
7. División de fórmulas.
8. Forzar cambio de numeración.
9. Entorno subequations y nombrar operaciones.
10. Diagramas conmutativos.
11. Ejercicios resueltos.

9.1. Repaso de matemáticas básicas

9.1.1. Modo matemático

Para entrar en el modo matemático es muy importante definir en el preámbulo el paquete `\usepackage{amsmath}`, ya que, sin este paquete no nos funcionará el modo matemático y nos dará error.
Una vez lo hayamos definido, nos permitirá utilizar todos los comandos para poder escribir nuestras fórmulas matemáticas. Seguidamente, vamos a listar todos los comandos que se pueden utilizar:

- `$`: Entrar y salir en el modo matemático en el modo texto. Es decir, que las fórmulas matemáticas están escritas dentro de un texto. Además de este comando, hay otro comando que se puede utilizar en el modo texto, que es el comando `\(\)`.
- `$$`: Entrar y salir del modo matemático resaltado, ya que, las funciones matemáticas están fuera del texto con un tamaño mayor. Además de este comando, hay otro comando que se puede utilizar para el modo de texto resaltado que es `\[\]`.
- Otro entorno que se puede utilizar para escribir las ecuaciones que es el equation. Cuyos comandos son `\begin{equation}` y `\end{equation}`. De esta forma, también escribiremos nuestra fórmula de un modo resaltado, es decir, fuera del texto. Este modo nos permite enumerar nuestras ecuaciones, y en el caso de que no queramos siempre podemos añadirle el símbolo `*` al principio del comando.

Posteriormente, después de describir los comandos vamos a ver una serie de ejemplos para ver cómo se escriben las fórmulas:
La función de la recta pendiente es $y = mx + b$ tangente a la recta.

Código 9.1: Ejemplo de función matemática con el texto

```
1 La funcion de la recta pendiente es $y=mx+b$ tangente a la recta.
```

La función de la recta pendiente es $y = mx + b$ tangente a la recta.

Código 9.2: Ejemplo de función matemática con el texto resaltado

```
1 La funcion de la recta pendiente es \(y=mx+b\) tangente a la recta.
```

La función de la recta pendiente es

$$y = mx + b$$

tangente a la recta.

Código 9.3: Ejemplo de función matemática con el texto resaltado

```
La funcion de la recta pendiente es $$y=mx+b$$ tangente a la recta.
```

La función de la recta tangente es

$$y = mx + b$$

es tangente a la recta.

Código 9.4: Ejemplo de función matemática con el texto resaltado

```
La funcion de la recta tangente es
\[
y=mx+b
\]
es tangente a la recta.
```

La función de la recta tangente es:

$$y = mx + b$$

es tangente a la recta.

Código 9.5: Ejemplo de función matemática con el entorno de la ecuación sin numerar

```
La funcion de la recta tangente es:
\begin{equation*}
y=mx+b
\end{equation*}
es tangente a la recta.
```

La función de la recta tangente es

$$y = mx + b \tag{9.1}$$

es tangente a la recta.

Código 9.6: Ejemplo de función matemática con el entorno de la ecuación numerado

```
La funcion de la recta tangente es
\begin{equation}
y=mx+b
\end{equation}
es tangente a la recta.
```

Como podemos ver, con este último entorno, se puede numerar la ecuación y si hacemos un índice de ecuaciones, nos aparecerá en este índice.

Tablas de símbolos básicos

Para poder usar los símbolos en LaTeX es necesario escribir en el preámbulo el paquete `\usepackage{amssymb}`.
Una vez escrito, podemos escribir los siguientes símbolos:

α:\alpha	θ:\theta	$\o$:\o	τ:\tau
β:\beta	ϑ:\vartheta	π:\pi	υ:\upsilon
γ:\gamma	ι:\iota	ϖ:\varpi	ϕ:\phi
δ:\delta	κ:\kappa	ρ:\rho	φ:\varphi
ϵ:\epsilon	λ:\lambda	ϱ:\varrho	χ:\chi
ε:\varepsilon	μ:\mu	σ:\sigma	ψ:\psi
ζ:\zeta	ν:\nu	ς:\varsigma	ω:\omega
η:\eta	ξ:\xi	Σ:\Sigma	Ψ:\Psi
Γ:\Gamma	Λ:\Lambda	Υ:\Upsilon	Ω:\Omega
Δ:\Delta	Ξ:\Xi	Φ:\Phi	
Θ:\Theta	Π:\Pi		

Cuadro 9.1: Letras griegas

$\digamma$:\digamma	$\varkappa$:\varkappa

Cuadro 9.2: Letras griegas AMS

$\leftarrow$:\leftarrow	$\longleftarrow$:\longleftarrow	$\uparrow$:\uparrow
$\Leftarrow$:\Leftarrow	$\Longleftarrow$:\Longleftarrow	$\Uparrow$:\Uparrow
$\rightarrow$:\rightarrow	$\longrightarrow$:\longrightarrow	$\downarrow$:\downarrow
$\Rightarrow$:\Rightarrow	$\Longrightarrow$:\Longrightarrow	$\Downarrow$:\Downarrow
$\leftrightarrow$:\leftrightarrow	$\longleftrightarrow$:\longleftrightarrow	$\updownarrow$:\updownarrow
$\Leftrightarrow$:\Leftrightarrow	$\Longleftrightarrow$:\Longleftrightarrow	$\Updownarrow$:\Updownarrow
$\mapsto$:\mapsto	$\longmapsto$:\longmapsto	$\nearrow$:\nearrow
$\hookleftarrow$:\hookleftarrow	$\hookrightarrow$:\hookrightarrow	$\searrow$:\searrow
$\leftharpoonup$:\leftharpoonup	$\rightharpoonup$:\rightharpoonup	$\swarrow$:\swarrow
$\leftharpoondown$:\leftharpoondown	$\rightharpoondown$:\rightharpoondown	:\nwarrrow
$\rightleftharpoons$:\rightleftharpoons	$\leadsto$:\leadsto	$\to$:\to

Cuadro 9.3: Símbolos flechas

⇢:\dashrightarrow	⇠:\dashleftarrow
⇇:\leftleftarrows	⇆:\leftrightarrows
⇚:\Lleftarrow	↞:\twoheadleftarrow
↢:\leftarrowtail	↫:\looparrowleft
⇋:\leftrightharpoons	↶:\curvearrowleft
↺:\circlearrowleft	↰:\Lsh
⇈:\upuparrows	↿:\upharpoonleft
⇃:\downharpoonleft	⊸:\multimap
↭:\leftrightsquigarrow	⇉:\rightrightarrows
⇄:\rightleftarrows	:\twoheadrightarrows
↣:\rightarrowtail	↬:\looparrowright
⇌:\rightleftharpoons	↷:\curvearrowright
↻:\circlearrowright	↱:\Rsh
:\downdownnarrows	↾:\upharpoonright
⇂:\downharpoonright	⇝:\rightsquigarrow

Cuadro 9.4: Flechas AMS

↚:\nleftarrow	↛:\nrightarrow	⇍:\nLeftarrow
⇏:\nRightarrow	↮:\nleftrightarrow	⇎:\nleftrightarrow

Cuadro 9.5: Flechas negación AMS

±:\pm	∩:\cap	⋄:\diamond	⊕:\oplus
∓:\mp	∪:\cup	△:\bigtriangleup	⊖:\ominus
×:\times	⊎:\uplus	▽:\bigtriangledown	⊗:\otimes
÷:\div	⊓:\sqcap	◁:\triangleleft	⊘:\oslash
∗:\ast	⊔:\sqcup	▷:\triangleright	⊙:\odot
⋆:\star	∨:\vee	⊲:\lhd	◯:\bigcirc
∘:\circ	∧:\wedge	⊳:\rhd	†:\dagger
•:\bullet	∖:\setminus	⊴:\unlhd	‡:\ddagger
·:\cdot	≀:\wr	⊵:\unrhd	⨿:\amalg

Cuadro 9.6: Operadores binarios

$\sum$:\sum	$\bigcap$:\bigcap	$\bigodot$:\bigodot
$\prod$:\prod	$\bigcup$:\bigcup	$\bigotimes$:\bigotimes
$\coprod$:\coprod	$\bigsqcup$:\bigsqcup	$\bigoplus$:\bigoplus
$\int$:\int	$\bigvee$:\bigvee	$\biguplus$:\biguplus
$\oint$:\oint	$\bigwedge$:\bigwedge	∞:\infty

Cuadro 9.7: Operadores de tamaño variable

$\leq$:\leq	$\geq$:\geq	$\equiv$:\equiv	$\models$:\models
$\prec$:\prec	$\succ$:\succ	$\sim$:\sim	$\perp$:\perp
$\preceq$:\preceq	$\succeq$:\succeq	$\simeq$:\simeq	$\mid$:\mid
$\ll$:\ll	$\gg$:\gg	$\asymp$:\asymp	$\parallel$:\parallel
$\subset$:\subset	$\supset$:\supset	$\approx$:\approx	$\bowtie$:\bowtie
$\subseteq$:\subseteq	$\supseteq$:\supseteq	$\cong$:\cong	$\Join$:\Join
$\sqsubset$:\sqsubset	$\sqsupset$:\sqsupset	$\neq$:\neq	$\smile$:\smile
$\sqsubseteq$:\sqsubseteq	:\sqsupteq	$\doteq$:\doteq	$\frown$:\frown
$\in$:\in	$\ni$:\ni	$\propto$:\propto	=:=
$\vdash$:\vdash	$\dashv$:\dashv	<:<	>:>
::	.\cdot		

Cuadro 9.8: Operadores de relación

,,	;;	: :\colon	.:\ldotp	$\cdot$:\cdotp

Cuadro 9.9: Signos de puntuación

$\sqrt{x}$:\sqrt{x}	$\sqrt[y]{x}$:\sqrt[y]{x}

Cuadro 9.10: Otros símbolos

Hay más símbolos de los que se han explicado arriba, si queréis buscar más sólo tenéis que buscarlos en cualquier página web especializada en LaTeX como ShareLaTeX y Overleaf.

Fórmulas simples

Ahora vamos a explicar cómo poner unas fórmulas simples, que son aquellas que tienen multiplicaciones y divisiones. Además de estas fórmulas simples, vamos a ver cómo elevar algún número y cómo poner los subíndices.
Primero, vamos a poner una fórmula simple, como por ejemplo:

$$3 \times 2 = 6$$

$$10 \div 2 = 5$$

Código 9.7: Ejemplo del código de fórmulas simples

```
\usepackage{amsmath}, definir antes de utilizar cualquier formula
\[
3\times 2=6
\]
\[
10\div 2=5
\]
```

Con esta sencilla estructura, podemos crear unas fórmulas matemáticas que incluyan: sumas, restas, multiplicaciones y divisiones.
Seguidamente, vamos a explicar cómo se elevan los números y cómo se ponen los subíndices. Para elevar los números vamos a utilizar el comando `^{}`, y entre los corchetes ponemos el número que queremos elevar. Para verlo más claro, vamos a poner varios ejemplos:

$$5^2 = 25$$

Código 9.8: Ejemplo del código de elevación de números

```
\usepackage{amsmath}, definir antes de utilizar cualquier formula
\[
5^2=25
\]
```

$$x^x y = x^{xy}$$

Código 9.9: Ejemplo del código de elevación de números

```
\usepackage{amsmath}, definir antes de utilizar cualquier formula
\[
x^xy=x^{xy}
\]
```

Como podemos ver en este segundo ejemplo, es muy importante poner entre corchetes los números que tienen estar elevados, sino, si hay dos números que se quieren elevar, sólo se elevará el primero y no el segundo.
Y por último, vamos a ver cómo poner los subíndices, para realizarlo, vamos a utilizar el comando `_{}`, y colocando entre los corchetes el número que queremos colocar el subíndice. Para verlo en acción vamos a poner varios ejemplos:

$$x_1 + x_2 = x_3$$

Código 9.10: Ejemplo del código de subíndice de números

```
\usepackage{amsmath}, definir antes de utilizar cualquier formula
\[
x_1+x_2=x_3
\]
```

$$y_{xyz} + x_{xyz} + z_{xyz} = 2xyz_xyz$$

Código 9.11: Ejemplo del código de subíndice de números

```
\usepackage{amsmath}, definir antes de utilizar cualquier formula
\[
y_{xyz}+x_{xyz}+z_{xyz}=2xyz_xyz
\]
```

Como podemos ver, al igual que al hacer las elevaciones, si hay más de un número que queremos llevar al subíndice los tendremos que colocar entre los corchetes, sino nos pasará como en el ejemplo, que no todos los números se coloquen en el subíndice.

Estructuras matemáticas: límites, sumatorios

En este capítulo vamos a explicar dos tipos de estructuras matemáticas: los límites y los sumatorios.
Primero vamos a empezar con los límites cuyo comando es `\lim_{x\to}`. En el subíndice vamos a colocar hacia donde tiene el límite y arriba colocaremos nuestro límite. Y su comportamiento es diferente dependiendo del modo matemático donde estemos. Para visualizarlo, vamos a poner diferentes ejemplos:

$$\lim_{x\to 0} \frac{1}{x^2} = \infty$$

Código 9.12: Ejemplo del código de límites

```
\usepackage{amsmath}, definir antes de utilizar cualquier formula
\[
\lim_{x\to 0}\dfrac{1}{x^2}=\infty
\]
```

$$\lim_{x\to 0} \frac{1}{x^2} = \infty$$

Ejemplo del código de límites

```
\usepackage{amsmath}, definir antes de utilizar cualquier formula
\begin{center}
$\lim_{x\to 0}\dfrac{1}{x^2}=\infty$
\end{center}
```

$$\lim_{x\to 0}\frac{1}{x^2}=\infty$$

Código 9.13: Ejemplo del código de límites

```
\usepackage{amsmath}, definir antes de utilizar cualquier formula
\begin{equation*}
\lim_{x\to0}\dfrac{1}{x^2}=\infty
\end{equation*}
```

Como podemos ver, si estamos en distintos modos matemáticos, los límites no se van a comportar de la misma manera, ya que, cambia la posición del subíndice y el ancho del límite. Además, podemos ver que siguen una estructura muy sencilla que es muy fácil de seguir.
Seguidamente, vamos a explicar cómo se realizan los sumatorios. Para realizarlo, utilizamos el comando `\sum_{}^{}`, y su tamaño depende del modo matemático que estemos utilizando. Para tenerlo más claro, vamos a poner varios ejemplos con todos los modos matemáticos para ver cómo se comporta:

$$\sum_{n=1}^{\infty} 2^{-n} = 1$$

Código 9.14: Ejemplo del código de sumatorio

```
\usepackage{amsmath}, definir antes de utilizar cualquier formula
\begin{center}
$\sum_{n=1}^{\infty}2^{-n}=1$
\end{center}
```

$$\sum_{n=1}^{\infty} 2^{-n} = 1$$

Código 9.15: Ejemplo del código de sumatorio

```
\usepackage{amsmath}, definir antes de utilizar cualquier formula
\[
```

```
\sum_{n=1}^{\infty}2^{-n}=1
\]
```

$$\sum_{n=1}^{\infty}2^{-n}=1$$

Código 9.16: Ejemplo del código de sumatorio

```
\usepackage{amsmath}, definir antes de utilizar cualquier formula
\begin{equation*}
\sum_{n=1}^{\infty}2^{-n}=1
\end{equation*}
```

Como podemos observar, el sumatorio no se comporta igual en todos los modos matemáticos.
Por ejemplo, si queremos cambiar la posición de los subíndices o los superíndices en el modo texto, vamos a utilizar el comando `\limits` a continuación de la operación. Vamos a aplicar este comando al ejemplo anterior:

$$\sum\limits_{n=1}^{\infty}2^{-n}=1$$

Código 9.17: Ejemplo del código de sumatorio con los subíndices cambiados

```
\usepackage{amsmath}, definir antes de utilizar cualquier formula
\begin{center}
$\sum\limits_{n=1}^{\infty}2^{-n}=1$
\end{center}
```

Por el contrario, si queremos que los subíndices se hagan a un lado en el modo texto resaltado vamos a utilizar el comando `\nolimits`. Vamos a aplicar este comando nuevo al ejemplo:

$$\sum\nolimits_{n=1}^{\infty}2^{-n}=1$$

Código 9.18: Ejemplo del código de sumatorio con los subíndices a un lado

```
\usepackage{amsmath}, definir antes de utilizar cualquier formula
\[
\sum\nolimits_{n=1}^{\infty}2^{-n}=1
\]
```

Como podemos ver, los sumatorios no se comportan igual en todos los modos matemáticos, esto se puede aplicar igual a los operadores de tamaño variable y a funciones como son los límites.
Los operadores de tamaño variable se pueden encontrar en el capítulo de símbolos matemáticos.

Estructuras matemáticas: fracciones y raíces

En este capítulo vamos a explicar las fracciones y las raíces, tienen estructuras sencillas, pero si queremos añadirle algún elemento tipo paréntesis su estructura cambia y además, también cambian según el modo matemático en el que estemos.
Primero vamos a explicar las fracciones, las cuales podemos expresar por dos comandos, el comando `\frac{numerador}{denominador}` y `\dfrac{numerador}{denominador}`. La diferencia que hay entre estos dos comandos es que en el comando `\frac{numerador}{denominador}`, la fracción va estar expresada de una manera estándar y el comando `\dfrac{numerador}{denominador}` expresa la fracción en el modo displaystyle, este modo disminuye un poco el tamaño de la fracción respecto al anterior. Para tener más clara esta diferencia vamos a verlo con varios ejemplos:

$$\frac{1}{2}x = \frac{1}{2}x$$

Código 9.19: Ejemplo del código de los dos comandos de fracciones

```
\usepackage{amsmath}, definir antes de utilizar cualquier formula
\[
\dfrac{1}{2}x=\frac{1}{2}x
\]
```

Además de lo anterior, la fracción cambia de tamaño según el modo matemático en el que estemos, ya que, no se va a representar en el modo texto que en el resaltado. Para verlo de una manera más clara vamos a poner un ejemplo que contenga una fracción en estos dos modos:
La fracción $\frac{1}{2}x$ es igual a:

$$\frac{1}{2}x$$

Y esta última es de una envergadura similar a:

$$\frac{1}{2}x$$

Código 9.20: Ejemplo del código de la fracción en los distintos entornos matemáticos

```
\usepackage{amsmath}, definir antes de utilizar cualquier formula
La fraccion $\frac{1}{2}x$ es igual a:
\[
\frac{1}{2}x
\]
Y esta ultima es de un envergadura similar a:
\begin{equation*}
\frac{1}{2}x
\end{equation*}
```

Como podemos observar, el tamaño de la fracción no es el mismo en los distintos modos. Este cambio de tamaño en los distintos modos es más fácil de ver con el comando `\frac{numerador}{denominador}` que con el comando `\dfrac{numerador}{denominador}`. Además de tener estos problemas de tamaño según el modo en el que estemos, también nos encontramos con otro problema, el tamaño de los paréntesis o corchetes no se corresponde con nuestra fracción. Para verlo vamos a poner un ejemplo:

$$(\frac{1}{2}x)^2 = \frac{1}{4}x^2$$

Código 9.21: Ejemplo del código del tamaño desigual de los delimitadores

```
\usepackage{amsmath}, definir antes de utilizar cualquier formula
\[
(\dfrac{1}{2}x)^2=\dfrac{1}{4}x^2
\]
```

Cuando nos encontremos con este problema, tenemos dos opciones:

- Escribir al inicio del delimitador el comando `\left` y al final del delimitador el comando `\right` para que se ajuste automáticamente al tamaño de la fracción.
- Por otro lado, podemos poner el comando `\Big` para hacer más grande el delimitador.

Para ver como actúan los dos comandos vamos a corregir el ejemplo anterior:

$$\left(\frac{1}{2}x\right)^2 = \frac{1}{4}x^2$$

Código 9.22: Ejemplo del código corrigiendo el tamaño de los delimitadores con left y right

```
\usepackage{amsmath}, definir antes de utilizar cualquier formula
\[
\left(\dfrac{1}{2}x\right)^2=\dfrac{1}{4}x^2
\]
```

$$\left(\frac{1}{2}x\right)^2 = \frac{1}{4}x^2$$

Código 9.23: Ejemplo del código corrigiendo el tamaño de los delimitadores con Big

```
\usepackage{amsmath}, definir antes de utilizar cualquier formula
\[
\Big(\dfrac{1}{2}x\Big)^2=\dfrac{1}{4}x^2
\]
```

Lo que podemos observar de los ejemplos anteriores es que el comando `\Big` no se ajusta igual que los comandos `\left` y `\right` a la fracción, ya que con el comando no es capaz de ajustar su tamaño al de la fracción. El usuario tendrá que decir que tipo de comando `\Big` quiere ajustar a su fracción.
Por otro lado, tenemos las raíces que pueden ser expresadas por los comandos `\sqrt{}` o `\sqrt[]{}`. Depende de qué tipo de raíz estemos haciendo elegir un comando u otro, ya que, si vamos a hacer una raíz cuadrada elegiremos el comando `\sqrt{}` o si vamos a hacer una raíz cúbica o cuarta vamos a escoger el comando `\sqrt[]{}`. Para ver más clara la diferencia entre estos dos comandos vamos a realizar dos ejemplos:

$\sqrt{2}$ es la raíz cuadrada de 2.
$\sqrt[4]{2}$ es la raíz cuarta de 2.

Código 9.24: Ejemplo del código de la diferencia entre los comandos de las raíces

```
\usepackage{amsmath}, definir antes de utilizar cualquier formula
\begin{center}
$\sqrt{2}$ es la raiz cuadrada de 2.\\
$\sqrt[4]{2}$ es la raiz cuarta de 2.
\end{center}
```

Como podemos ver, una vez sabemos el código de las fracciones y las raíces podemos expresarla de una manera muy sencilla.

Estructuras matemáticas: integrales

En este capítulo vamos a explicar cómo realizar todo tipo de integrales, desde las simples pasando por las integrales en dos puntos y terminando con las dobles.
Lo primero que tenemos que tener en cuenta es el comando de la integral que es `\int`, ya sólo con esto seremos capaces de realizar una integral inmediata como la del siguiente ejemplo:

$$\int x^6 dx = \frac{x^7}{7} + C$$

Código 9.25: Ejemplo del código de una integral inmediata

```
\usepackage{amsmath}, definir antes de utilizar cualquier formula
\[
\int x^6 dx=\dfrac{x^7}{7}+C
\]
```

Una vez tengamos claro como hacer una integral inmediata, podemos pasar a realizar una integral en dos puntos. Para ello vamos a utilizar el siguiente comando `\int_{P1}^{P2}` para delimitarla entre dos puntos. Para tenerlo más claro vamos a poner un ejemplo:

$$\int_0^2 (2x + 2 - x^2 - 2)dx = \left[x^2 - \frac{x^3}{3}\right]_0^2 = 4 - \frac{8}{3} = \frac{4}{3}$$

Código 9.26: Ejemplo del código de una integral en dos puntos

```
\usepackage{amsmath}, definir antes de utilizar cualquier formula
\[
\int_{0}^{2}(2x+2-x^2-2)dx=\left[x^2-\dfrac{x^3}{3}\right]_{0}^{2}
   =4-\dfrac{8}{3}=\dfrac{4}{3}
\]
```

Seguidamente, si queremos realizar integrales dobles y triples utilizaremos comandos como `\iint` y `\iiint`. Para ver cómo serían estas integrales vamos a poner dos ejemplos:

$$\iint f(x) = \iiint g(x)$$

Código 9.27: Ejemplo del código de las integrales dobles y triples

```
\usepackage{amsmath}, definir antes de utilizar cualquier formula
\[
\iint f(x)=\iiint g(x)
\]
```

Además, si queremos juntar una integral junto con algún elemento más como una fracción, no quedará de todo bien. Como lo podemos ver en el siguiente ejemplo:

$$x = \frac{\int_x^y f(x)dx}{g(x)}$$

Código 9.28: Ejemplo del código de la integral junto con una fracción

```
\usepackage{amsmath}, definir antes de utilizar cualquier formula
\[
x=\dfrac{\int_{x}^{y}f(x)dx}{g(x)}
\]
```

Si queremos solucionar este problema, utilizaremos el comando `\displaystyle` para que la integral esté bien dispuesta en la fracción. Con esto, vamos a solucionar este ejemplo:

$$x = \frac{\displaystyle\int_x^y f(x)dx}{g(x)}$$

Código 9.29: Ejemplo del código de la integral junto con una fracción con la integral bien dispuesta

```
\usepackage{amsmath}, definir antes de utilizar cualquier formula
\[
x=\dfrac{\displaystyle\int_{x}^{y}f(x)dx}{g(x)}
\]
```

Y por último, si queremos realizar integrales cerradas simples. Y para representarlas utilizaremos el comando `\oint`. Para verlo más claro vamos a hacer un ejemplo:

$$\oint_x = \int x$$

Código 9.30: Ejemplo del código de la integral cerrada simple

```
\usepackage{amsmath}, definir antes de utilizar cualquier formula
\[
\oint_x=\int x
\]
```

Estructuras matemáticas: textificación

En este capítulo vamos a ver cómo poner texto en nuestras ecuaciones y otros tipos de construcciones que podemos realizar en el modo matemático.
Primero, para colocar texto en nuestro modo matemático usaremos el comando `\text{}`, si no ponemos este comando, el texto saldrá mal. Para verlo más claro vamos a poner un ejemplo:

$$y = mx + b \quad \text{es la ecuación de la recta}$$

Código 9.31: Ejemplo del código de un texto dentro del modo matemático

```
\usepackage{amsmath}, definir antes de utilizar cualquier formula
\[
y=mx+b\quad\text{es la ecuacion de la recta}
\]
```

Como podemos ver en el ejemplo, con el comando nuestro texto se coloca perfectamente y con su espaciado, como si estuviera fuera del modo matemático.
Por otro lado, vamos a ver ciertas construcciones que nos pueden ser útiles para el modo matemático. Entre estas construcciones destacan el `\overbrace{Texto o función}` que pone un brazo por encima del texto y `\underbrace{Texto o función}` colocará un brazo por debajo del texto o de la función. Además, si queremos poner algo por debajo del brazo sólo tendremos que añadirle el comando `_{}` o si queremos ponerle algo por arriba del brazo superior tendremos que añadirle el comando `^{}`. Para verlo más claro vamos a poner varios ejemplos:

$$\overbrace{x^2+x}^{x(x+1)} = \underbrace{x^2+x}_{x(x+1)}$$

Código 9.32: Ejemplo del código de brazo superior y brazo inferior

```
\usepackage{amsmath}, definir antes de utilizar cualquier formula
\[
\overbrace{x^2+x}^{x(x+1)}=\underbrace{x^2+x}_{x(x+1)}
\]
```

Como podemos ver, con estos brazos podemos añadir lo que queramos abajo o arriba de la función o el texto. Además, podemos combinar estos brazos con textos. Vamos a realizar un ejemplo con esta combinación:

$$\overbrace{x^2+x}^{x(x+1)} = \underbrace{x^2+x}_{\text{Lo mismo que en el anterior}}$$

Código 9.33: Ejemplo del código de brazo superior y brazo inferior con texto

```
\usepackage{amsmath}, definir antes de utilizar cualquier formula
\[
\overbrace{x^2+x}^{x(x+1)}=\underbrace{x^2+x}_{\text{Lo mismo que
   en el anterior}}
\]
```

Estas expresiones se pueden concatenar de la siguiente manera:`\underbrace{Texto}_{\underbrace{Texto}}`. Para tenerlo más claro vamos a poner un ejemplo en el que se concatenen varias de estas expresiones:

$$\overbrace{\underbrace{x^2+x}_{x(x+1)}}^{1}$$

$$\overbrace{\overbrace{\text{Texto cualquiera}}^{\text{Texto cualquiera}}}$$

Código 9.34: Ejemplo del código de brazo superior y brazo inferior con texto y concatenados

```
\usepackage{amsmath}, definir antes de utilizar cualquier formula
\[
\underbrace{x^2+x}_{\underbrace{x(x+1)}_{1}}
\]
\[
\overbrace{\text{Texto cualquiera}}^{\overbrace{\text{Texto
   cualquiera}}}
\]
```

Otra construcción que podemos ver es el vector que utilizará el comando `\Vec{}`, y entre los corchetes pondremos nuestro vector. Para verlo mejor vamos a poner un ejemplo:

$$\vec{x_1} = \vec{x_2} - \vec{x_3}$$

Estructuras matemáticas: Sistemas de ecuaciones

Hay varias formas de representar los sistemas de ecuaciones, una es mediante el entorno array y otra es mediante el entorno eqnarray donde las ecuaciones se nos representan respecto al =.
Primero vamos a ver cómo se representan las ecuaciones mediante el entorno array, y para ello, vamos a utilizar los comandos `\begin{array}{ll}` donde las ll representan el número de

ecuaciones y terminando con el comando `\end{array}`. Separaremos las ecuaciones mediante el comando `\\`.Además, intepondremos una llave mediante el comando `\left\{` si la llave está a la izquierda o mediante el comando `\right\}` si está a la derecha, en el caso que esté la llave a uno de los lados, en el otro lado se usará el comando `\left.` o `\right.`. Para tenerlo más claro, vamos a poner un ejemplo, uno con la llave a la izquierda y otra con la llave a la derecha:

$$\left\{\begin{array}{lll} x+y+2z=45 \\ 2x+4y+18z=145 \\ 47x+23y+89z=1089 \end{array}\right.$$

Código 9.35: Ejemplo del código de sistemas de ecuaciones con llave a la izquierda

```
\usepackage{amsmath}, definir antes de utilizar cualquier formula
\[\left\{\begin{array}{lll}
x+y+2z=45\\
2x+4y+18z=145\\
47x+23y+89z=1089
\end{array}\right.
\]
```

$$\left.\begin{array}{lll} 10x+20y+80z=2000 \\ 2x+8y+50z=45 \\ 11x+44y+63z=78 \end{array}\right\}$$

Código 9.36: Ejemplo del código de sistemas de ecuaciones con llave a la derecha

```
\usepackage{amsmath}, definir antes de utilizar cualquier formula
\[\left.\begin{array}{lll}
10x+20y+80z=2000\\
2x+8y+50z=45\\
11x+44y+63z=78
\end{array}\right\}
\]
```

Además de poder darle estos formatos, también podemos realizar el sistema de ecuaciones sin llaves. Para ello, utilizaremos los comandos `\left.` y `\right.`. Y vamos a demostrarlo con un ejemplo:

$$\begin{array}{l} x+y+z=10 \\ 2x+45y+14z=89 \\ 588x+789y+123z=1023 \\ 89x+74y+45z=630 \end{array}$$

Código 9.37: Ejemplo del código de sistemas de ecuaciones sin llave

```
\usepackage{amsmath}, definir antes de utilizar cualquier formula
\[\left.\begin{array}{llll}
x+y+z=10\\
2x+45y+14z=89\\
588x+789y+123z=1023\\
89x+74y+45z=630
\end{array}\right.
\]
```

Por otro lado, tenemos el entorno eqnarray donde se nos colocará el sistema de ecuaciones con un = en medio y con la misma separación en todas las operaciones. Aquí no nos hace falta indicar el número de ecuaciones que vamos a poner, eso sí, para separar las ecuaciones vamos a utilizar el comando \\ y para colocar las ecuaciones utilizaremos &=& para separar las ecuaciones. Para ilustrar todo lo anterior vamos a poner un ejemplo:

$$\begin{aligned} y &= x && (9.2)\\ x^2+2y &= 3x^2+2y && (9.3)\\ (x+y)(2x-y) &= 3x-y && (9.4) \end{aligned}$$

Código 9.38: Ejemplo del código del entorno eqnarray

```
\usepackage{amsmath}, definir antes de utilizar cualquier formula
\begin{eqnarray}
y&=&x\\
x^2+2y&=&3x^2+2y\\
(x+y)(2x-y)&=& 3x-y
\end{eqnarray}
```

Y por último, a este entorno eqnarray se le puede añadir cajas sin marco como \mbox{} para colocar un texto en nuestro sistema de ecuaciones. Para verlo lo podemos colocar en el siguiente ejemplo:

$$\begin{aligned} y &= x && (9.5)\\ x^2+2y &= 3x^2+2y\text{(Ecuación a destacar)} && (9.6)\\ (x+y)(2x-y) &= 3x-y && (9.7) \end{aligned}$$

Código 9.39: Ejemplo del código del entorno eqnarray con caja mbox

```
\usepackage{amsmath}, definir antes de utilizar cualquier formula
\begin{eqnarray}
y&=&x\\
x^2+2y&=&3x^2+2y\mbox{(Ecuacion a destacar)}\\
(x+y)(2x-y)&=& 3x-y
\end{eqnarray}
```

9.2. Matrices y determinantes

Para escribir tanto matrices como determinantes utilizaremos el entorno array, que empieza con `\begin{array}{cc}` y termina con `\end{array}`. Donde en el segundo corchete del inicio especificaremos el número de columnas que queremos que tenga la matriz.

$$A = \left(\begin{array}{cc} 1 & 2 \\ 3 & 4 \end{array} \right)$$

$$B = \left[\begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 5 & 6 & 7 \end{array} \right]$$

Código 9.40: Código de matriz y determinante

```
 \[
   A=
  \left( {\begin{array}{cc}
   1 & 2 \\
   3 & 4 \\
  \end{array} } \right)
\]
\[
  B=
  \left[ {\begin{array}{ccccc}
   1 & 2 & 3 & 4 & 5\\
   3 & 4 & 5 & 6 & 7\\
  \end{array} } \right]
\]
```

Es muy importante haber puesto en el preámbulo anteriormente el modo matemático y si queremos colocar poner un paréntesis o un corchete ponerlo al principio del comando `\left` y al final del comando `\right`. Las `{ccc}` representan las columnas que tiene la matriz, si

tiene por ejemplo, dos columnas se representará como {cc}. Por tanto, las c representan el número de columnas.
Otra forma de realizar matrices es mediante el comando \bmatrix. Este comando nos permite poner matrices entre corchetes. De esta manera tenemos el siguiente ejemplo:

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Código 9.41: Ejemplo del código de matriz entre corchetes

```
\usepackage{amsmath}, definir antes de utilizar cualquier formula
\[I=\begin{bmatrix}
1 & 0 & 0\\
0 & 1 &0 \\
0 & 0 & 1
\end{bmatrix}
\]
```

Por otro lado, tenemos el comando \pmatrix, que nos permite poner a la matriz entre paréntesis. Para ver cómo se usa vamos a realizar un ejemplo:

$$M = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 6 \end{pmatrix}$$

Código 9.42: Ejemplo del código de matriz entre paréntesis

```
\usepackage{amsmath}, definir antes de utilizar cualquier formula
\[M=\begin{pmatrix}
1 & 2 & 3 & 4 & 5\\
6 & 7 & 8 & 9 & 6
\end{pmatrix}
\]
```

Dentro del entorno de la matriz encontramos dos elementos a destacar:

- &: Para separar las columnas de la matriz.
- \\: Para separar filas.

Además de poder realizar matrices, podemos realizar operaciones con matrices como multiplicaciones, divisiones··· Además, podemos combinar distintos entornos de matrices como podemos ver en el siguiente ejemplo:

$$A \cdot B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$$

Código 9.43: Ejemplo de código de operaciones entre matrices

```
\usepackage{amsmath}, definir antes de utilizar cualquier formula
\[A\cdot B=\begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix}\cdot\left(\begin{array}{cc}
  2   & 0 \\
 0    & 2
\end{array}\right)=\begin{pmatrix}
2 & 0\\
0 & 2
\end{pmatrix}
\]
```

Por otro lado, para representar los determinantes de estas matrices cambiaremos los delimitadores a `\left|` y `\right|` para cambiar el paréntesis a este delimitador. Para ver el cambio de la matriz al determinante vamos a poner un ejemplo:

$$|A| = \begin{vmatrix} 1 & 2 \\ 2 & 1 \end{vmatrix}$$

Código 9.44: Ejemplo del código de un determinante

```
\usepackage{amsmath}, definir antes de utilizar cualquier formula
\[|A|=\left|\begin{array}{cc}
  1   & 2 \\
  2   & 1
\end{array}\right|
\]
```

9.3. Coeficientes binomiales

Como hemos explicado en capítulos anteriores, podemos colocar fracciones con el comando `\frac{}{}` pero los coeficientes binomiales son un poco más complicados de poner con este comando. Para ello, utilizaremos el comando `\binom{}{}`, y entre los dos corchetes colocaremos nuestros binomios. Para tenerlo más claro vamos a poner un ejemplo:
$\binom{x}{y} = \binom{x+y}{1}$

$$\binom{x+y+z}{1} = \binom{z+y}{x+y}$$

Código 9.45: Ejemplo del código del binomio

```
$\binom{x}{y}=\binom{x+y}{1}$
\[
\binom{x+y+z}{1}=\binom{z+y}{x+y}
\]
```

9.4. Símbolos en negrita

Es muy importante disponer de los paquetes amsmath, amsymb y euscript en el preámbulo. Una vez que lo tengamos, sólo tenemos que utilizar el comando `\boldsymbol{}` en el modo matemático con sus respectivos tipos de letra que son `\mathsf{}`, `\mathcal{}`, `\mathscr{}` y `\mathfrak{}`. Vamos a ver en una tabla cada tipo de letra con su destacado en negrita:

Cuadro 9.11: Letras en negrita

$\boldsymbol{XYZTEFN}$
$\boldsymbol{\mathsf{XYZTEFN}}$
$\boldsymbol{\mathcal{XYZTEFN}}$
$\boldsymbol{\mathscr{XYZTEFN}}$
$\boldsymbol{\mathfrak{XYZTEFN}}$

Código 9.46: Código de la tabla

```
\begin{table}[H]
\centering
\caption{Letras en negrita}
\begin{tabular}{|l|}
\hline
$\boldsymbol{XYZTEFN}$ \\ \hline
$\boldsymbol{\mathsf{XYZTEFN}}$ \\ \hline
$\boldsymbol{\mathcal{XYZTEFN}}$  \\ \hline
$\boldsymbol{\mathscr{XYZTEFN}}$  \\ \hline
$\boldsymbol{\mathfrak{XYZTEFN}}$  \\ \hline
\end{tabular}
\end{table}m{x+y+z}{1}=\binom{z+y}{x+y}
\]
```

Por otro lado, si queremos colocar alguna letra en negrita, pero el símbolo es demasiado grande por lo que utilizaremos el comando `\pmb{}` para poner estos símbolos más grande

en negrita. Vamos a poner un ejemplo:

$$\boldsymbol{\prod}\boldsymbol{;}\boldsymbol{\bigcup}$$

Código 9.47: Ejemplo de simbolos en negrita

```
\[
\pmb{\prod}; \pmb{\bigcup}
\]
```

9.5. Símbolos encima de símbolos

Si queremos hacer alguna relación entre varias variables, vamos a necesitar utilizar poner una constante sobre otra. Para ello, podremos utilizar más de un comando y son los siguientes:

- `\overset{}{}`: Entre los corchetes podremos lo que queremos encima y debajo.
- `\underset{}{}`: Entre los corchetes podremos lo que queremos encima y debajo.
- `\xrightarrow[]{}`: Lo mismo que los anteriores, pero con flechas.
- `\xleftarrow[]{}`: Lo mismo que los anteriores, pero con flechas.

Vamos a poner un ejemplo de cada uno de estos comandos y vamos a mostrar cómo quedan:

$$\overset{a}{\underset{z}{X}}$$

$$0 \to 1^x \xrightarrow[d-1]{y} z$$

$$\mathsf{R}/N \xleftarrow[z]{x} \mathcal{Z}$$

Código 9.48: Código de los ejemplos anteriores

```
\[
\overset{a}{\underset{z}{X}}
\]
$$0\fd 1^x\xrightarrow[d-1]{y} z$$
\[
\mathsf{R}/N\xleftarrow[z]{x}\mathcal{Z}
\]
```

9.6. Definición de nuevos comandos

Si queremos acortar algún comando o llamarlo de otra manera, para ello vamos a utilizar el comando `\newcommand{Comando renombrado}{Comando a renombrar}`. Es muy importante escribirlo en el preámbulo para poder definirlo. Para tenerlo más claro vamos a definir un nuevo comando, que es la flecha derecha con el comando `\fd`:

$$1 \rightarrow x + y$$

Código 9.49: Código para renombrar

```
%Preambulo
\newcommand{\fd}{\rightarrow}
%Una vez definido
\[
1\fd x+y
\]
```

9.7. División de fórmulas

9.7.1. Entorno multiline

Este entorno se utiliza para dividir las distintas fórmulas, sin tener una alineación predeterminada y lo realiza en dos renglones. Para comenzar el entorno se comenzará con el comando `\begin{multline}` y se cerrará con el comando `\end{multiline}`. Además, los renglones se pueden separar con `\\`. Para tenerlo más claro vamos a poner un ejemplo:

$$\frac{x-y+z}{h} = \\ = \frac{2x+z}{3x-z} - \frac{v(x+z)-z(x+y)}{v-z} = \\ = \frac{x+8}{z-3}$$

Código 9.50: Código del entorno multiline

```
\begin{multline*}
    \frac{x-y+z}{h}=\\
=\frac{2x+z}{3x-z}-\dfrac{v(x+z)-z(x+y)}{v-z}=\\
=\frac{x+8}{z-3}
\end{multline*}
```

$$x + y + z =$$
$$= 2x + y + z = 3x + y \quad (9.8)$$

Código 9.51: Código del entorno multiline

```
\begin{multline}
x+y+z=\\
=2x+y+z=3x+y
\end{multline}
```

9.7.2. Entorno gather

Es similar al entorno anterior, pero aparecerán cada función centrada y numerada. Para ello comenzaremos el entorno con el comando `\begin{gather}` y cerraremos con el comando `\end{gather}`. Y si queremos eliminar la numeración en alguna de las numeraciones utilizaremos el comando `\notag`. Para tenerlo más claro vamos a poner otro ejemplo:

$$x + y = (2x + 8x + z) \quad (9.9)$$
$$z - t = \frac{x}{y} \quad (9.10)$$

Código 9.52: Código del entorno gather

```
\begin{gather}
    x+y=(2x+8x+z)\\
    z-t=\frac{x}{y}
\end{gather}
```

$$2x + y + z = 3x + 2y + z$$
$$z - v = (x + y + z)^2$$

Código 9.53: Código del entorno gather

```
\begin{gather*}
    2x+y+z=3x+2y+z\\
    z-v=(x+y+z)^2
\end{gather*}
```

9.7.3. Entorno align

Este entorno también nos permite alinear las fórmulas, se separan entre los renglones con el comando \\ y se separa cada reglón con el comando &. Comenzará con el comando \begin{align} y terminará con el comando \end{align}. Para tenerlo más claro vamos a poner un ejemplo:

$$X + Z := \{a + b \mid x \in a,\ y \in b\}, \tag{9.11}$$
$$XZ := \{ab \mid xy \in ab,\ xy \in ab\}, \tag{9.12}$$
$$-Z := \{-Z \mid z \in ab\}, \tag{9.13}$$
$$XY - -2 := \{z - -2 \mid ab \in AB,\ ab \neq O\} \tag{9.14}$$

Código 9.54: Código del entorno align

```
\begin{align}
X+Z &:= \{a+b \mid x\in a,\ y\in b\},\\
XZ &:= \{ab \mid xy\in ab,\ xy\in ab\},\\
-Z&:= \{-Z \mid z\in ab\},\\
XY-{-2} &:= \{z-{-2} \mid ab\in AB,\ ab\ne O\}
\end{align}
```

$$\begin{aligned} y &= ay + bz \\ xy' &= axy' + bt \\ yz &= (y - ab)ty \\ yx' &= (zx - bt)yv' \end{aligned} \qquad \begin{aligned} Y &= xY + t \\ XY' &= utX' + v \\ YX &= (z - uT)YX \\ YX' &= (xt - vu)YX' \end{aligned} \qquad \begin{aligned} C &= aX + BT \\ AB' &= atA' + Bt' \\ Ba &= (x - ab)BT \\ AB' &= (xt - ba)BT' \end{aligned}$$

Código 9.55: Código del entorno align

```
\begin{align*}
y &= ay+bz & Y&= xY+t & C&= aX+BT\\
xy' &= axy'+bt & XY' &= utX'+v & AB' &= atA'+Bt'\\
yz&= (y-ab)ty & YX&= (z-uT)YX & Ba &= (x-ab)BT\\
yx' &= (zx-bt)yv' & YX' &= (xt-vu)YX' & AB' &= (xt-ba)BT'
\end{align*}
```

Si queremos colocar texto entre las alineaciones sólo tenemos que utilizar el comando \intertext{}. Para tenerlo más claro vamos a poner varios ejemplos:

$$(x + y)'x'y + xy'$$

Se puede escribir de otra manera:

$$xy' = (xy)' - x'y \tag{9.15}$$

Y las integrales que dan son las siguientes:

$$\int xy' = \int (xy)' - \int x'y \tag{9.16}$$

Código 9.56: Código del entorno align con texto

```
\begin{align}
(x+y)' & x'y+xy'\notag\\
\intertext{Se puede escribir de otra manera:}
xy'&=(xy)'-x'y\\
\intertext{Y las integrales que dan son las siguientes:}
\int xy' &=\int(xy)'-\int x'y
\end{align}
```

$$x + y = \int x - \int z$$

Ahora ponemos otro ejemplo de integral:

$$\int (x + y + z) = 2 + 2x$$

Ponemos otro ejemplo con un sumatorio:

$$\sum_{x=0}^{x=100} 2x + 200 = 1000$$

Código 9.57: Código del entorno align con texto

```
\begin{align*}
    x+y=\int x-\int z\\
\intertext{Ahora ponemos otro ejemplo de integral:}
\int (x+y+z)= 2+2x\\
\intertext{Ponemos otro ejemplo con un sumatorio:}
\sum_{x=0}^{x=100} 2x+200= 1000
\end{align*}
```

9.7.4. Entorno flalign

Este entorno se utilizará para realizar las alineaciones en columnas. Comenzará con el comando `\begin{flalign}` y terminará con el comando `\end{flalign}`. Se separarán las columnas con el comando `&` y se separarán las filas con el comando `\\`. Para tenerlo más claro vamos a poner un ejemplo:

$$
\begin{aligned}
zy &= a+b+z & Z &= tx+v & D &= ax+by\\
xz' &= azx'+tb & XZ' &= uTX'+vT & AT' &= aTA'+TXB'\\
yze &= (1-abt)ty & YT &= (1-ut)YB & Ba &= (1-ab)BA\\
yx' &= (1-ba)yx' & YX' &= (1-vt)YX' & aB' &= (1-ba)BT'
\end{aligned}
$$

Código 9.58: Código del entorno flalign

```
\begin{flalign*}
zy &= a+b+z & Z&= tx+v & D&= ax+by\\
xz' &= azx'+tb & XZ' &= uTX'+vT & AT' &= aTA'+TXB'\\
yze&= (1-abt)ty & YT&= (1-ut)YB & Ba &= (1-ab)BA\\
yx' &= (1-ba)yx' & YX' &= (1-vt)YX' & aB' &= (1-ba)BT'
\end{flalign*}
```

$$
\begin{aligned}
X &= zX+t & Y &= aY+b & B &= xB+Bt && (9.17)\\
xy' &= ay'+bv & XY' &= utXY'+vt & B' &= xB'+Y' && (9.18)\\
z &= (x-a)t & X &= (z-t)X & C &= (x-t)C && (9.19)\\
z' &= (t-a)z' & X' &= (t-x)X' & C' &= (x-t)C' && (9.20)
\end{aligned}
$$

Código 9.59: Código del entorno flalign

```
\begin{flalign}
X &= zX+t & Y&= aY+b & B&= xB+Bt &\\
xy' &= ay'+bv & XY' &= utXY'+vt & B' &= xB'+Y' &\\
z&= (x-a)t & X&= (z-t)X & C &= (x-t)C &\\
z' &= (t-a)z' & X' &= (t-x)X' & C' &= (x-t)C' &
\end{flalign}
```

9.8. Forzar cambio de numeración

Como hemos visto en todas las alineaciones anteriores, la numeración de las fórmulas se hace de manera automática pero en el caso de que no queramos hacer esa numeración y

queramos cambiarla por cualquier símbolo vamos a utilizar el comando \tag y lo podremos numerar o no numerar como en los modos matemáticos anteriores. Vamos a poner un ejemplo con asterisco y uno con letras:

$$\begin{aligned} Y &= xy + zt & X &= tx + zt & C &= XT + ZT & (*)\\ xy &= cd + tz & Y &= xu + yt & D &= aX + bY & (**) \end{aligned}$$

Código 9.60: Código del entorno flalign con asteriscos de numeración

```
\begin{flalign*}
Y &= xy+zt & X &= tx+zt & C &= XT+ZT & \tag{$\ast$}\\
xy &= cd+tz & Y &= xu+yt & D &= aX+bY & \tag{$\ast\ast$}
\end{flalign*}
```

$$\begin{aligned} x + y &= 2x + t & (A)\\ (x+y)^2 &= x^2 + y^2 + 2xy & (B) \end{aligned}$$

Código 9.61: Código del entorno align con letras de numeración

```
\begin{align}
    x+y= 2x+t \tag{A}\\
    (x+y)^2= x^2+y^2+2xy\tag{B}
\end{align}
```

9.9. Entorno subequations y nombrar operaciones

Si queremos utilizar una numeración con subapartados vamos a utilizar el entorno subequations. Este entorno comienza con el comando \begin{subequations} y termina con el comando \end{subequations}, entre medias utilizaremos el entorno de las fórmulas que queremos numerar y haremos referencia a cada uno de estos subapartados con el comando \label{}. Para tenerlo más claro vamos a poner dos ejemplos con dos tipos de entornos distintos:

$$\begin{aligned} x + y &= z + t & (9.21a)\\ Y &= x^2 + y^2 + z^2 & (9.21b) \end{aligned}$$

Podemos nombrar muchas funciones, ya sean 9.21 todas, o cada una por separado: 9.21a o 9.21b.

Código 9.62: Código del entorno subequations y nombrar operaciones

```
\begin{subequations}\label{formulas}
\begin{align}
    x+y= z+t \label{formula suma}\\
    Y= x^2+y^2+z^2 \label{formula cuadrados}
\end{align}
\end{subequations}
Podemos nombrar muchas funciones, ya sean \ref{formulas} todas, o
   cada una por separado: \ref{formula suma} o \ref{formula
   cuadrados}.
```

$$x+y=zx+t=v \qquad t+v=y \quad (9.22a)$$
$$x^2+y^2=zx+y=z^2 \qquad x^3+t^3+z^3=A \quad (9.22b)$$

Podemos nombrar cada una, que es 9.22a o 9.22b. O podemos nombrar todas las ecuaciones 9.22.

Código 9.63: Código del entorno subequations y nombrar ecuaciones

```
\begin{subequations}\label{formulas en columnas}
\begin{flalign}
x+y=z & x+t=v & t+v=y\label{sumandos}\\
x^2+y^2=z & x+y=z^2 & x^3+t^3+z^3=A \label{cuadrados}
\end{flalign}
\end{subequations}
Podemos nombrar cada una, que es \ref{sumandos} o \ref{cuadrados}.
   O podemos nombrar todas las ecuaciones \ref{formulas en columnas
   }.
```

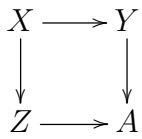
9.10. Diagramas conmutativos

Para poder realizar un diagrama conmutativo sencillo vamos a necesitar instalar el paquete xy en el preámbulo. Para ello necesitaremos colocar el paquete `\usepackage[all]{xy}`. Una vez lo tenemos escrito en el preámbulo, vamos a entrar en el modo matemático y escribir el comando `\xymatrix{}`, donde dentro de los corchetes podremos escribir las flechas para indicar la relación entre objetos con el comando `\ar[]` y dentro de este comando podremos colocar la posición de las flechas:

- r: A la derecha.

- l: A la izquierda.
- u: Hacia arriba.
- d: Hacia abajo.

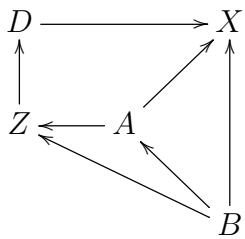
El espacio entre las flechas se señalará con el comando `&`. Para tenerlo más claro vamos a poner un ejemplo:



Código 9.64: Código del diagrama conmutativo simple

```
\usepackage[all]{xy}

\xymatrix{
X \ar[d] \ar[r] & Y \ar[d] \\
Z \ar[r] & A
}
\]
```

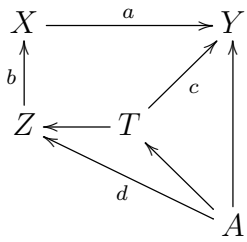
Además, podremos separar entre las distintas filas con el comando `\\`. Vamos a ver un ejemplo donde está la separación:



Código 9.65: Código diagrama conmutativo más complicado

```
\[
\xymatrix{
D \ar[rr]  &  &  X \\
Z \ar[u] & A \ar[l] \ar[ru]  \\
& & B \ar[uu] \ar[lu] \ar[llu]
}
\]
```

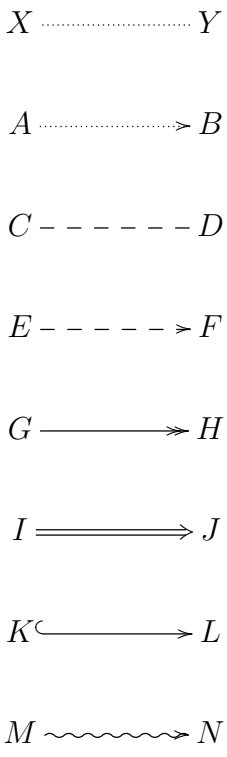
A estas flechas se le pueden colocar elementos tanto arriba como abajo, para poner elementos arriba utilizaremos el comando `\ar[]^{}` y para poner elementos abajo utilizaremos el comando `\ar[]_{}`. Para tenerlo más claro vamos a poner un ejemplo:



Código 9.66: Código diagrama conmutativo con elementos arriba y abajo

```
\[
\xymatrix{
X \ar[rr]^{a}  &  &  Y \\
Z \ar[u]^{b} & T \ar[l] \ar[ru]_{c}  \\
& & A \ar[uu] \ar[lu] \ar[llu]^{d}
}
\]
```

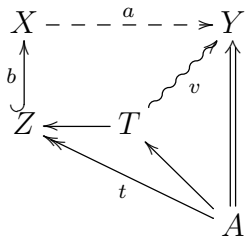
Además, estas flechas podrían ser modificadas con el comando `\ar@{Modificador}[]`. Entre los modificadores podremos utilizar son los siguientes:



Código 9.67: Código de modificadores de flechas

```
\[
\xymatrix{
X \ar@{.}[rr] & & Y \\
A \ar@{.>}[rr] & & B \\
C \ar@{--}[rr] & & D \\
E \ar@{-->}[rr] & & F \\
G \ar@{>>}[rr] & & H \\
I \ar@{=>}[rr] & & J \\
K \ar@{^(->}[rr] & & L \\
M \ar@{~>}[rr] & & N \\
}
\]
```

También vamos a poner un ejemplo con un diagrama con las flechas modificadas:



Código 9.68: Código del diagrama con flechas modificadas

```
\[
\xymatrix{
X \ar@{-->}[rr]^{a}  &  &  Y \\
Z \ar@{^(->}[u]^{b} & T \ar[l] \ar@{~>}[ru]_{v}  \\
& & A \ar@{=>}[uu] \ar[lu] \ar@{>>}[llu]^{t}
}
\]
```

9.11. Ejercicios resueltos

Ejercicio 1. Realiza dos coeficientes binomiales con los dos tipos de modos matemáticos.

$\binom{a+b+c+d}{x+y}=\binom{x+y+z}{a+b}$

$$\binom{x+y+z}{a+b}=\binom{3}{x+y}$$

Código 9.69: Solución ejercicio 1

```
$\binom{a+b+c+d}{x+y}=\binom{x+y+z}{a+b}$
\[
\binom{x+y+z}{a+b}=\binom{3}{x+y}
\]
```

Ejercicio 2. Realiza una letra en negrita en modo matemático y realiza un sumatorio con texto en negrita.

$$\boldsymbol{\mathcal{R}}$$

$$\pmb{\sum_{x}^{y}}=x+y+z$$

Código 9.70: Solución ejercicio 2

```
\[
\boldsymbol{\mathcal{R}}
\]
\[
\pmb{\sum_{x}^{y}}=x+y+z
\]
```

Ejercicio 3. Realiza dos fórmulas que contengan un símbolo encima de símbolos y que contenga algún sumatorio o producto en negrita.

$$\prod=\overset{x}{\underset{a}{b}}$$

$$\sum=x\overset{a}{\leftarrow}y\overset{b}{\rightarrow}z$$

Código 9.71: Solución ejercicio 3

```
\[
\pmb{\prod}=\overset{x}{\underset{a}{b}}
\]
```

```
\[
\pmb{\sum}=x\xleftarrow{a}y\xrightarrow{b}z
\]
```

Ejercicio 4. Realiza dos fórmulas una con el entorno multiline y el entorno gather. Observa la diferencia de alineación entre ambas.

$$x+y+z=$$
$$2x+z+2t \quad (9.23)$$

$$x+z=x+y+z \quad (9.24)$$
$$\frac{x}{y}+z=x+yz \quad (9.25)$$

Código 9.72: Solución ejercicio 4

```
\begin{multline}
    x+y+z=\\
    2x+z+2t
\end{multline}
\begin{gather}
    x+z=x+y+z\\
    \frac{x}{y}+z=x+yz
\end{gather}
```

Ejercicio 5. Con las fórmulas del ejercicio anterior, quítale la enumeración a las del multiline y cambia la enumeración del gather por asteriscos.

$$x+y+z=$$
$$2x+z+2t$$

$$x+z=x+y+z \quad (*)$$
$$\frac{x}{y}+z=x+yz \quad (**)$$

Código 9.73: Solución ejercicio 5

```
\begin{multline*}
    x+y+z=\\
    2x+z+2t
\end{multline*}
\begin{gather}
    x+z=x+y+z\tag{$\ast$}\\
    \frac{x}{y}+z=x+yz\tag{$\ast\ast$}
\end{gather}
```

Ejercicio 6. Realiza tres fórmulas, una con el entorno gather, otra con el entorno align y otra con el entorno flamign. Inserta el texto a la del entorno align y cambia la numeración de la del entorno flamign.

$$x+y+z=\frac{x+y-z}{2x}-2 \tag{9.26}$$
$$2x+6z=3z \tag{9.27}$$

$$x+y+z=2x+z \tag{9.28}$$

En este modo podemos realizar todo tipo de fórmulas:

(9.29)

$$\int x+y=z \tag{9.30}$$

Desde sumatorios a productos:

(9.31)

$$\sum x+y=\prod x+z \tag{9.32}$$

$$x=Y+ZX=Z+T \qquad x=t+b \text{ (A)}$$
$$Y=X+ZZ=X+T \qquad z=x+t \text{ (B)}$$

Código 9.74: Solución ejercicio 6

```
\begin{gather}
```

```
    x+y+z=\frac{x+y-z}{2x}-2\\
    2x+6z=3z
\end{gather}
\begin{align}
    x+y+z=2x+z\\
\intertext{En este modo podemos realizar todo tipo de fórmulas:}\\
\int x+y=z\\
\intertext{Desde sumatorios a productos:}\\
\sum x+y= \prod x+z
\end{align}
\begin{flalign}
x=Y+Z & X=Z+T & x=t+b\tag{A}\\
Y=X+Z & Z=X+T & z=x+t\tag{B}
\end{flalign}
```

Ejercicio 7. Realiza dos fórmulas con el entorno subequations, nómbralas a cada una de ellas en un texto y también de una manera general.

$$x+y+z=\frac{x+y+z}{xy} \tag{9.33a}$$

$$2x+z=\sum x+y \tag{9.33b}$$

$$x+z=2x+z \tag{9.33c}$$

$$xz=\prod xtz \tag{9.33d}$$

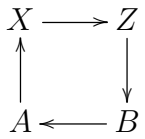
Podemos nombrar la primera fórmula con 9.33a o la penúltima fórmula con 9.33c. Podemos nombrar esta con 9.33b o esta con 9.33d.

Código 9.75: Solución ejercicio 7

```
\begin{subequations}
\begin{gather}
    x+y+z=\frac{x+y+z}{xy}\label{ejercicio1}\\
    2x+z=\sum x+y\label{ejercicio2}
\end{gather}
\begin{flalign}
x+z=2x+z\label{ejericicio3}\\
xz=\prod xtz\label{ejercicio4}
```

```
\end{flalign}
\end{subequations}
Podemos nombrar la primera fórmula con \ref{ejercicio1} o la penú
   ltima fórmula con \ref{ejericicio3}. Podemos nombrar esta con \
   ref{ejercicio2} o esta con \ref{ejercicio4}.
```

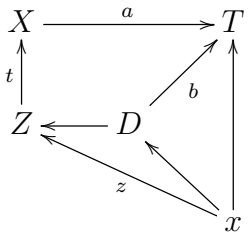
Ejercicio 8. Realiza un diagrama conmutativo simple.



Código 9.76: Solución ejercicio 8

```
\[
\xymatrix{
X \ar[r]  & Z \ar[d] \\
A \ar[u] & B \ar[l]
}
\]
```

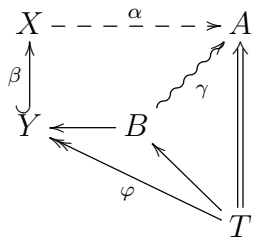
Ejercicio 9. Realiza un diagrama complicando un poco más la relación de las flechas.



Código 9.77: Solución ejercicio 9

```
\[
\xymatrix{
X \ar[rr]^{a}  &  &  T \\
Z \ar[u]^{t} & D \ar[l] \ar[ru]_{b}  \\
& & x \ar[uu] \ar[lu] \ar[llu]^{z}
}
\]
```

Ejercicio 10. Realiza un diagrama conmutativo con las flechas modificadas.



Código 9.78: Solución ejercicio 10

```
\[
\xymatrix{
X \ar@{-->}[rr]^{\alpha}  &  &  A \\
Y \ar@{^(->}[u]^{\beta} & B \ar[l] \ar@{~>}[ru]_{\gamma}  \\
& & T \ar@{=>}[uu] \ar[lu] \ar@{>>}[llu]^{\varphi}
}
\]
```

10

Entorno gráfico y Tikz básico

En este capítulo aprenderemos a utilizar los entornos graphic y graphicx, realizaremos gráficos y gráficos avanzados, y finalmente, realizaremos ejercicios para repasar los conceptos anteriores.

1. Graphic, graphicx.
2. Gráficos.
3. Gráficos avanzados.
4. Ejercicios resueltos.

10.1. Graphic, graphicx

Los paquetes graphic y graphicx se encargan de realizar transformaciones geométricas de objetos, podremos aumentar la escala, rotarla o reflexionarla. Para instalarlos en nuestro preámbulo utilizaremos el comando `\usepackage{graphics}` para el paquete graphic o el comando `\usepackage{graphicx}` para el paquete graphicx. En este capítulo vamos a poder ver las siguientes modificaciones:

- Aumento de la escala.
- Reflexión.
- Rotación.

Además de ver este tipo de modificaciones vamos a ver cómo podemos variar la altura, la anchura y el ángulo de las imágenes.

10.1.1. Aumento de la escala

Si queremos aumentar la escala, ya sea, de una tabla o de un texto vamos a utilizar el comando `\scalebox{Aumento}{Lo que vamos a aumentar}`. Vamos a ver dos ejemplos, uno donde se aumenta el tamaño y otro donde se disminuye:

Este es el teorema de Pitagóras: $a^2+b^2=c^2$
Este es el teorema de Pitagóras: $a^2+b^2=c^2$

Código 10.1: Código del aumento o la disminución

```
\begin{center}
\scalebox{0.7}{Este es el teorema de Pitagóras: $a^2+b^2=c^2$}\\
\scalebox{1.2}{Este es el teorema de Pitagóras: $a^2+b^2=c^2$}
\end{center}
```

También podremos añadir una escala vertical como comando opcional mediante el comando `\scalebox{Aumento}[vertical]{objeto}` o mediante otro comando similar que es `\resizebox{Horizontal}{Vertical}{objeto}`, si queremos mantener la proporción del objeto original utilizaremos el símbolo `!` en los argumentos. Vamos a poner dos ejemplos, uno con el primer comando y otro con el segundo:

Este es el teorema de Pitagóras: $a^2+b^2=c^2$
Este es el teorema de Pitagóras: $a^2+b^2=c^2$

Código 10.2: Código de aumento de escala en vertical

```
\begin{center}
    \scalebox{0.8}[2]{Este es el teorema de Pitagóras: $a^2+b^2=c^2
       $}\\
    \resizebox{7cm}{!}{Este es el teorema de Pitagóras: $a^2+b^2=c
       ^2$}
\end{center}
```

Además, podremos modificar su altura con el comando `\height` o su anchura con el comando `\width`. Podemos realizar un ejemplo cambiando la anchura y la altura de la misma frase:

Código 10.3: Cambiando la altura y la anchura

```
\begin{center}
    \resizebox{0.5\width}{3\height}{Este es el teorema de Pitagóras
       : $a^2+b^2=c^2$}
\end{center}
```

10.1.2. Reflexión

Si queremos dar un efecto visual o resaltar alguna palabra dándola la vuelta, utilizaremos el comando `\reflectbox{objeto}`. Dentro de este objeto podremos colocar lo que queramos, desde textos a tablas, incluyendo imágenes. Para tenerlo más claro vamos a poner un ejemplo:

Palabra vs ɹɐqɐlɐd

Código 10.4: Código de la palabra derecha vs espejo

```
\begin{center}
    Palabra vs \reflectbox{Palabra}
\end{center}
```

10.1.3. Rotación

Finalmente, también podremos rotar el objeto que queramos, desde palabras a cuadros o imágenes. Para ello, utilizaremos el comando `\rotatebox{Angulo}{objeto}`. Para tenerlo más claro vamos a poner un ejemplo:

El teorema de Pitagóras es $a^2+b^2=c^2$

Código 10.5: Código de rotación

```
\begin{center}
    El teorema de Pitagóras es \rotatebox{90}{$a^2+b^2=c^2$}
\end{center}
```

10.1.4. Cambiar la anchura, la altura y el ángulo de la imagen

Como hemos visto en la introducción vamos a poder variar la anchura con el comando width, la altura con el comando height y el ángulo con el comando angle. Estos comandos los podremos en la parte opcional del comando `\includegraphics[Opcional]{ruta de la imagen}`. Para aclarar las ideas vamos a poner un ejemplo modificando la altura y la anchura, y otro ejemplo cambiando el ángulo:

Figura 10.1: Gato modificado

Código 10.6: Código gato modificado

```
\begin{figure}[H]
    \centering
    \includegraphics[width=5cm, height=2cm]{Images/GATOEJ.jpg}
    \caption{Gato modificado}
\end{figure}
```

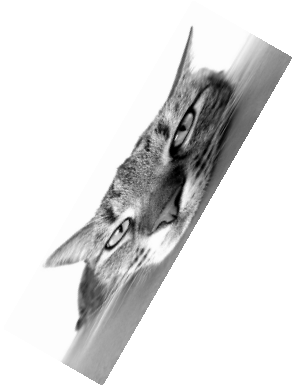


Figura 10.2: Gato modificado y girado

Código 10.7: Código gato modificado y girado

```
\begin{figure}[H]
    \centering
    \includegraphics[width=5cm,height=1.5cm,angle=60]{Images/GATOEJ
        .jpg}
    \caption{Gato modificado y girado}
\end{figure}
```

10.2. Gráficos

10.2.1. Diagrama de barras, círculos

Diagrama de barras

Para realizar un diagrama de barras vamos a instalar los paquetes pgfplots para habilitar la gráfica y textcomp, para añadirle un texto.
Una vez se haya instalado, comenzaremos con el entorno tikzpicture, para ello empezaremos con el comando `\begin{tikzpicture}` y terminaremos con el comando `\end{tikzpicture}`, pero no sólo estos comandos nos serán útiles para realizar la gráfica de barras, sino que deberemos añadir el entorno axis, este entorno comenzará con el comando `\begin{axis}` y terminará con el comando `\end{axis}`. Con este último entorno seremos capaces de añadir los datos de nuestra gráfica de barras y la leyenda. En este tipo de gráficos encontraremos los siguientes nuevos comandos:

- `x tick label style{/pgf/number format/1000 sep=}=`: Este comando definde el estilo del diagrama. Además, podremos incluir los ejes con el comando `\addplot` y si queremos añadir los valores del eje y con el comando `\ybar`.

- enlargelimits0.05=: Alarga los límites de la barra tanto se indique. En este caso, se le daría un límite de altura a la barra de 0.05.
- legend sytle{at={(0.5,-0.2)}, anchor=north, lengend columns=-1=: Indica la posición y el ancho de la leyenda, en este caso, sería muy cercana al eje x.
- ybar interval0.7=: Indica la delgadez de cada barra, si se indica 1 significa que las barras estarán una al lado de la otra y 0 significa que no habrá barras, sino lineas verticales.

A continuación, se mostrará un diagrama de barras con su código de ejemplo:

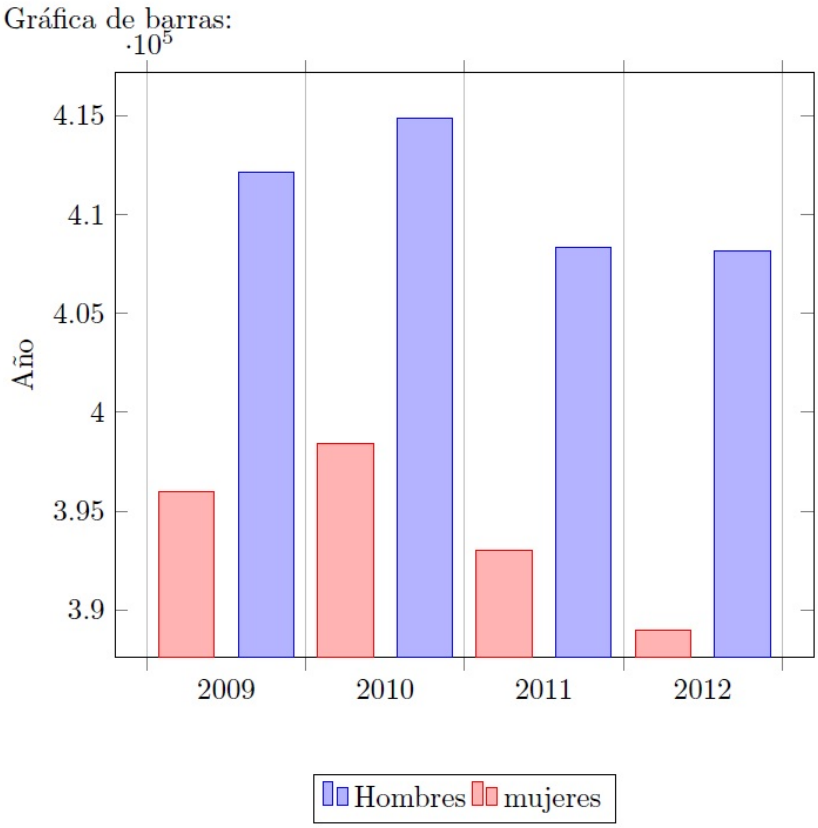


Figura 10.3: Gráfica de barras

Código 10.8: Código diagrama básico

```
\documentclass{article}
\usepackage[margin=0.5in]{geometry}
\usepackage[utf8]{inputenc}
```

```
\usepackage{textcomp}
\usepackage{pgfplots}
\pgfplotsset{width=10cm,compat=1.9}


\begin{document}

Gráfica de barras:
\begin{tikzpicture}
\begin{axis}[
    x tick label style={
                /pgf/number format/1000 sep=},
        ylabel=Año,
        enlargelimits=0.05,
        legend style={at={(0.5,-0.2)},
        anchor=north,legend columns=-1},
        ybar interval=.7,
]
\addplot
        coordinates {(2012,408184) (2011,408348)
                 (2010,414870) (2009,412156) (2008,415 838)};
\addplot
        coordinates {(2012,388950) (2011,393007)
                (2010,398449) (2009,395972) (2008,398866)};
\legend{Hombres, mujeres}
\end{axis}
\end{tikzpicture}
\end{document}
```

Diagrama de círculos

Para realizar un diagrama de círculos vamos a utilizar el paquete pgf-pie.
El entorno del diagrama de círculos comienza con `\begin{tikzpicture}` y termina con `\end{tickzpicture}`, y dentro del entorno con el comando
`\pie[Dimensiones del círculo]{Contenido del círculo}`. Para realizar cada porcentaje utilizaremos `/` y separemos cada porcentaje por comas. Para tenerlo más claro podremos un ejemplo:

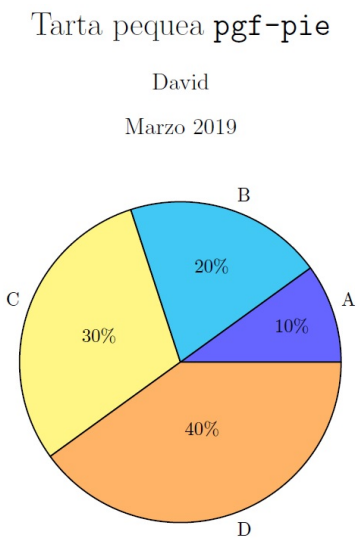


Figura 10.4: Diagrama de círculos

Código 10.9: Código diagrama círculos

```
\usepackage{pgf-pie}

\title{Tarta pequeña \texttt{pgf-pie}}
\author{David}
\date{Marzo 2019}

\begin{document}
\maketitle

\centering

\begin{tikzpicture}
\pie{10/A, 20/B, 30/C, 40/D}
\end{tikzpicture}
\end{document}
```

10.2.2. Esquema con llaves

Para realizar un esquema con llaves necesitaremos instalar en el preámbulo el paquete schemata.
Si queremos realizar una llave simple utilizaremos el comando `\schema{\schemabox{Contenido`

`a la izquierda de la llave}}{\schemabox{Contenido a la derecha}}`. Para tenerlo más claro vamos a poner un ejemplo:

Contenido 1 { Contenido 2

Código 10.10: Código esquema simple

```
\usepackage{schemata}
\schema{\schemabox{Contenido 1}}{\schemabox{Contenido 2}}
```

Con este comando algo enrevesado se puede crear un diagrama simple, pero renombrando con el comando `\newcommand\diagram[2]{\schema{\schemabox{#1}}{\schemabox{#2}}}` hace que podamos realizar un diagrama más sencillo. Y podremos realizar el siguiente esquema:

Criterios de exigencia {
- Constancia: **Trabajar mucho**
- Tiempo: **El tiempo empleado**
- Otros {
  - Buena presentación
  - ...

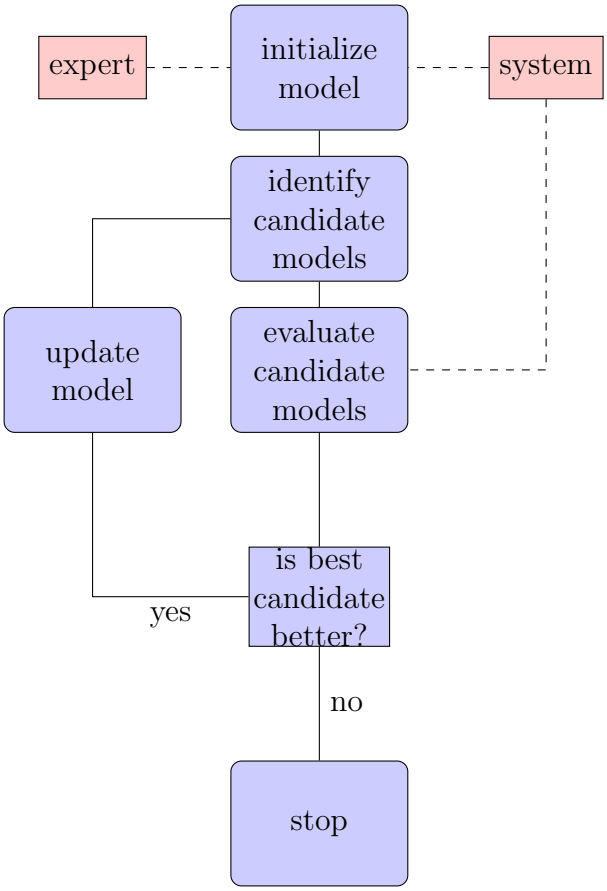
Código 10.11: Código esquema de llaves complejo

```
\usepackage{schemata}
\newcommand\diagram[2]{\schema{\schemabox{#1}}{\schemabox{#2}}}
\diagram{\textbf{Criterios de exigencia}}
{- Constancia: \textbf{Trabajar mucho} \\
- Tiempo: \textbf{El tiempo empleado} \\
- \diagram{Otros}{- Buena presentación \\ - \ldots}}
```

10.2.3. Diagramas de flujo

Para realizar un diagrama de flujo tendremos que instalar el paquete tikz y la librería `\usetikzlibrary{shapes, arrows}` para que nos pueda crear los nodos con sus formas y flechas.

Una vez dentro del documento, utilizaremos el entorno tikzstyle, en el que determinaremos la forma de cada uno de los nodos y su distancia entre ellos. Dentro del documento, empezaremos con el entorno tikzpicture para dar características a los nodos, para ello utilizaremos el comando `\begin{tikzpicture}` para comenzarlo y para describir los nodos utilizaremos el comando `\node[tipo de bloque]{nombre}`, uniremos cada nodo con el comando `\path` y terminaremos nuestro entorno con el comando `\end{tikzpicture}`. Para tenerlo más claro vamos a poner un ejemplo:



Código 10.12: Código diagrama de flujo

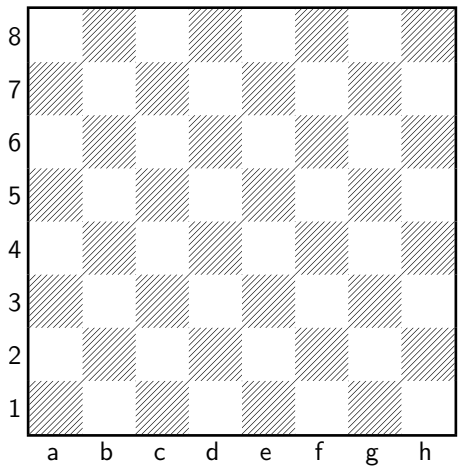
```
\documentclass{article}
\usepackage[latin1]{inputenc}
\usepackage{tikz}
\usetikzlibrary{shapes,arrows}
\usepackage{verbatim}
\usepackage[active,tightpage]{preview}
\PreviewEnvironment{tikzpicture}
\setlength\PreviewBorder{5pt}%
%%%>
\begin{document}
\pagestyle{empty}


% Estilo bloques
```

```
\tikzstyle{decision} = [diamond, draw, fill=blue!20,
    text width=4.5em, text badly centered, node distance=3cm, inner sep=0pt]
\tikzstyle{block} = [rectangle, draw, fill=blue!20,
    text width=5em, text centered, rounded corners, minimum height=4em]
\tikzstyle{line} = [draw, -latex']
\tikzstyle{cloud} = [draw, ellipse,fill=red!20, node distance=3cm,
    minimum height=2em]

\begin{tikzpicture}[node distance = 2cm, auto]
    % Nodos
    \node [block] (init) {initialize model};
    \node [cloud, left of=init] (expert) {expert};
    \node [cloud, right of=init] (system) {system};
    \node [block, below of=init] (identify) {identify candidate models};
    \node [block, below of=identify] (evaluate) {evaluate candidate models};
    \node [block, left of=evaluate, node distance=3cm] (update) {update model};
    \node [decision, below of=evaluate] (decide) {is best candidate better?};
    \node [block, below of=decide, node distance=3cm] (stop) {stop};
    % Nexos uni n
    \path [line] (init) -- (identify);
    \path [line] (identify) -- (evaluate);
    \path [line] (evaluate) -- (decide);
    \path [line] (decide) -| node [near start] {yes} (update);
    \path [line] (update) |- (identify);
    \path [line] (decide) -- node {no}(stop);
    \path [line,dashed] (expert) -- (init);
    \path [line,dashed] (system) -- (init);
    \path [line,dashed] (system) |- (evaluate);
\end{tikzpicture}
\end{document}
```

10.2.4. Ajedrez

Tablero

Las piezas de ajedrez tienen su propio y particular campo de batalla: un tablero de 64 casillas, 32 blancas y 32 negras (a veces se pueden usar otros colores, pero éstos son los más habituales). Lo primero que tenemos que conocer es cómo se coloca dicho tablero. Siempre lo haremos de la misma forma, de tal manera que a nuestra derecha haya una casilla blanca:

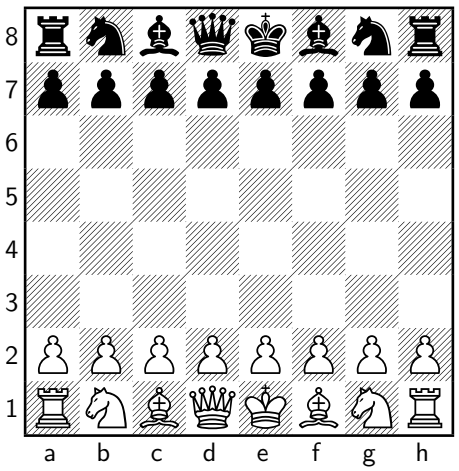


Código 10.13: Código tablero ajedrez

```
\documentclass{article}
\usepackage[spanish]{babel}
\usepackage[utf8]{inputenc}
\usepackage{skak}  % Vamos al jugar al ajedrez

\title{Ajedrez}
\author{David Pacios}

\begin{document}
\showboard
\end{document}
```

El tablero está dividido en filas, columnas y diagonales. Es importante familiarizarse con estos conceptos, ya que serán muy utilizados a lo largo del manual.
El tablero consta de ocho filas, numeradas del 1 al 8, y ocho columnas, definidas por otras tantas letras, de la 'A' a la 'H'. De este modo podemos identificar cada casilla de manera sencilla con sólo dar un número y una letra. Así, por ejemplo, las casillas de las esquinas del tablero serán: a1, h1, a8 y h8.

A continuación vamos a conocer cómo se colocan inicialmente las piezas, el momento en que ambos ejércitos esperan impacientes el comienzo de la batalla. Su distribución es armónica y tiene bastante lógica. Por delante se sitúan los peones, que actúan como soldados de infantería y protegen al resto de las piezas. El rey y la dama se colocan en las posiciones centrales, ya que son las piezas más importantes y de ese modo están más resguardadas. A izquierda y derecha de los monarcas se colocan el resto de piezas, con las torres situadas en las esquinas del tablero, como si de una fortaleza se tratase. Veamos con nuestros propios ojos donde debe situarse cada pieza:



Código 10.14: Código empezar partida

```
\documentclass{article}
\usepackage[spanish]{babel}
\usepackage[utf8]{inputenc}
\usepackage{skak}  % Vamos al jugar al ajedre

\title{Ajedrez}
\author{David Pacios}

\begin{document}
\newgame %Comenzar partida
\showboard
\end{document}
```

10.2.5. Las piezas

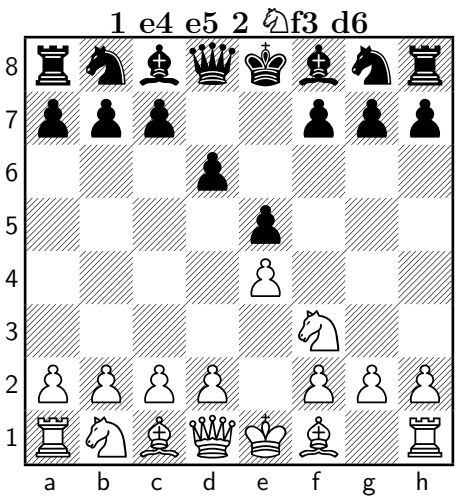
Es importante tener en cuenta que las piezas deben ser indicadas en terminología inglesa. Esto es:

- K – Rey (King)
- Q – Dama (Queen)
- R – Torre (Rook)
- N – Caballo (Knight)
- B – Alfil (Bishop)

Más adelante veremos cómo personalizar la entrada para nuestro idioma.

`\mainline{1. e4 e5 2. Nf3 d6}` Asegúrate de que tras el número de cada jugada haya siempre un punto (aunque luego no aparecerá en la salida), de lo contrario no te compilará el código.

Cierra las llaves en el momento que quieras para introducir comentarios o para representar un diagrama con la posición actual empleando `\showboard`

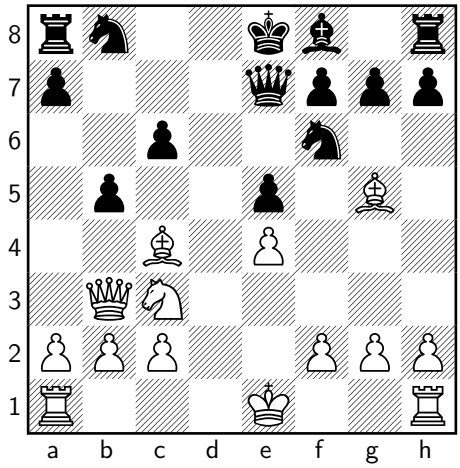


Código 10.15: Código de la jugada

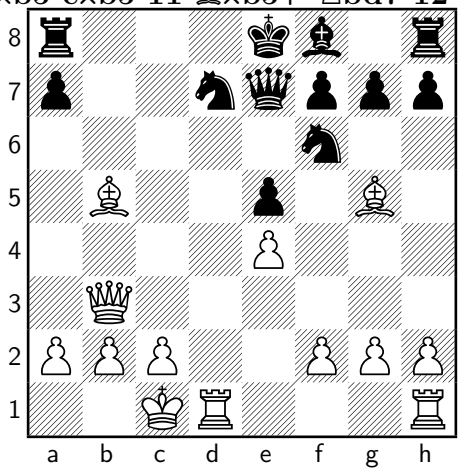
```
\begin{center}
 \newgame
\mainline{1. e4 e5 2.  Nf3 d6} \\
\showboard
\end{center}
```

Este es el código completo, incluido el preámbulo, de la transcripción de la partida. Observa como, tras cada interrupción, continuamos la línea principal con sucesivos comandos `\mainline`.

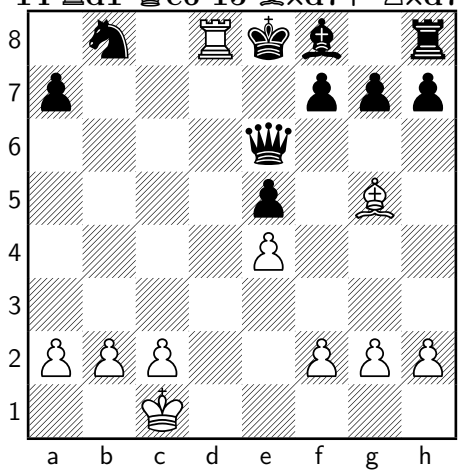
3 d4 ♗g4 4 d×e5 ♗×f3 5 ♕×f3 d×e5 6 ♗c4 ♘f6 7 ♕b3 ♕e7 8 ♘c3 c6 9 ♗g5 b5



10 ♘×b5 c×b5 11 ♗×b5+ ♘bd7 12 O-O-O



12...♖d8 13 ♖×d7 ♖×d7 14 ♖d1 ♕e6 15 ♗×d7+ ♘×d7 16 ♕b8+ ♘×b8 17 ♖d8#



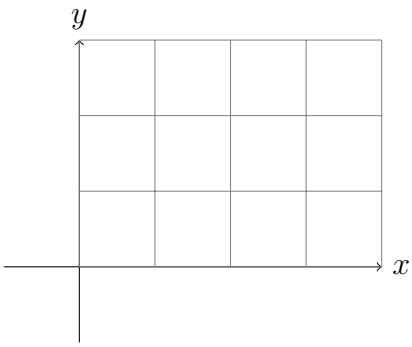
Código 10.16: Código partida de ajedrez

```
\begin{center}
\mainline{3. d4 Bg4 4.dxe5 Bxf3 5.Qxf3 dxe5 6.Bc4 Nf6 7.Qb3 Qe7 8.
   Nc3 c6 9.Bg5 b5}
\showboard
\mainline{10.Nxb5 cxb5 11.Bxb5+ Nbd7 12. O-O-O}
\showboard
\mainline{12... Rd8 13. Rxd7  Rxd7 14. Rd1 Qe6 15. Bxd7+ Nxd7 16.
   Qb8+ Nxb8 17. Rd8#}
\showboard
\end{center}
```

Presta atención a cómo hemos continuado la partida tras la jugada 12 del blanco. Observa también el uso de la O mayúscula en vez del cero para el enroque, así como el símbolo # para el jaque mate.

10.3. Gráficos avanzados

En este capítulo vamos a explicar cómo realizar unos gráficos sencillos con el paquete tikz. Para ello, vamos a instalar el paquete `\usepackage{tikz}`, el paquete `\usepackage{graphicx}` y el paquete `\usepackage{pgfplots}` en nuestro preámbulo.
Todo el entorno de las figuras se va a desarrollar en el tikzpicture, este entorno comienza con el comando `\begin{tikzpicture}` y termina con el comando `\end{tikzpicture}`. Una vez aclarado el entorno, vamos a explicar cómo realizar una rejilla para presentar el entorno gráfico.

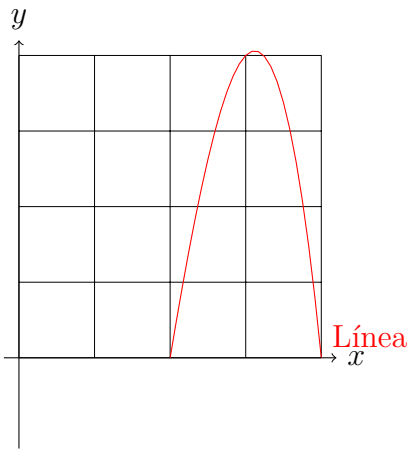


Código 10.17: Código de la rejilla

```
\begin{center}\begin{tikzpicture}
\draw[->] (-1,0) -- (4,0);
\draw (4,0) node[right] {$x$};
\draw [->] (0,-1) -- (0,3);
```

```
\draw (0,3) node[above] {$y$};
\draw [very thin, gray] (0,0) grid (4,3);
\end{tikzpicture}
\end{center}
```

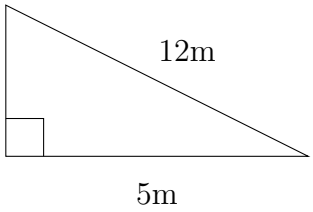
Cómo podemos ver, podemos dibujar cada línea con el comando \draw[] y con node podremos unir esas rectas. A continuación vamos a ver cómo dibujar una línea dentro de la rejilla:



Código 10.18: Código de la rejilla con línea

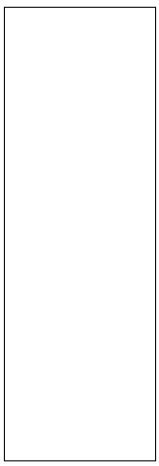
```
\begin{center}
\begin{tikzpicture}
\draw[very thin,color=black] (-0,-0) grid (4,4);

\draw[->] (-0.2,0) -- (4.2,0) node[right] {$x$};
\draw[->] (0,-1.2) -- (0,4.2) node[above] {$y$};

\draw[red] (0,0) plot[domain=2:4] (\x,{-(\x+1)*(\x-2)*(\x-4)}) node
   [anchor=south west] {Línea};
\end{tikzpicture}
\end{center}
```

Finalmente, una vez hemos visto cómo realizar una línea, vamos a ver cómo se realizan los polígonos básicos en tikzpicture. Primero vamos a representar un triángulo rectángulo:



Código 10.19: Código del triángulo

```
\begin{center}
        \begin{tikzpicture}
        \draw (1,1) -- (5,1) -- (1,3) -- cycle;
        \draw (1,1.5) -- (1.5,1.5) -- (1.5,1);
        \node at (3,0.5) {5m};
        \node at (3.4,2.4) {12m};
        \end{tikzpicture}
\end{center}
```

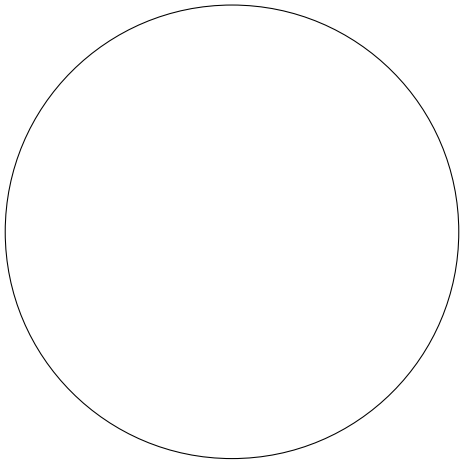


Código 10.20: Código del rectángulo

```
\begin{center}
\begin{tikzpicture}
\draw(1,2) rectangle (3,8);
\end{tikzpicture}
\end{center}
```

Como podemos ver, podemos atajar a la hora de realizar la figura con el comando rectangle dentro del entorno tikzpicture.

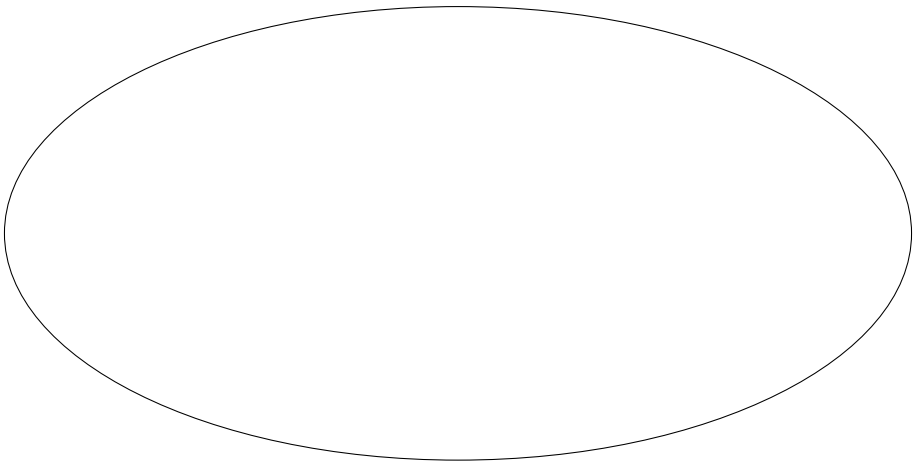
Seguidamente, vamos a realizar un círculo. Para ello, utilizaremos el comando circle, con el primer paréntesis de la función especificaremos la posición del círculo y con el segundo su radio.



Código 10.21: Código del círculo

```
\begin{center}
\begin{tikzpicture}
\draw (0,3) circle (3cm);
\end{tikzpicture}
\end{center}
```

Finalmente, la última figura básica que vamos a explicar es la elipse. Pero la única diferencia que tiene con el círculo, es que contiene dos radios que están separados entre sí por el comando and.



Código 10.22: Código de la elipse

```
\begin{center}
\begin{tikzpicture}
\draw(0,0) ellipse (6 cm and 3 cm);
\end{tikzpicture}
\end{center}
```

10.4. Ejercicios resueltos

Ejercicio 1. Realiza un aumento y una disminución de escala de un texto cualquiera. Seguidamente, realiza lo mismo con una tabla, pero en este caso realiza una tabla con los suficientes datos como para disminuirla de escala.

Este es un texto disminuido en escala.
Este es un texto aumentado en escala.

	Primera columna	**Segunda columna**	**Tercera columna**	**Cuarta columna**	**Quinta columna**
Primera fila	1	2	4	3	5
Segunda fila	10	11	12	13	14
Tercera fila	20	21	22	25	27
Cuarta fila	51	20	23	25	96
Quinta fila	46	33	36	39	56
Sexta fila	23	25	69	96	36
Septima fila	69	55	45	69	89
Octava fila	203	36	87	98	46

Código 10.23: Solución ejercicio 1

```
\begin{center}
\scalebox{0.7}{Este es un texto disminuido en escala.}\\
\scalebox{1.2}{Este es un texto aumentado en escala.}
\end{center}
\begin{table}[H]
\centering
\scalebox{0.7}{\begin{tabular}{|c|c|c|c|c|}
\hline
            & \textbf{Primera columna} & \textbf{Segunda columna}
              & \textbf{Tercera columna} & \textbf{Cuarta columna
              } & \textbf{Quinta columna} \\ \hline
```

```
Primera fila & 1 & 2 & 4 & 3 & 5 \\ \hline
Segunda fila & 10 & 11 & 12 & 13 & 14 \\ \hline

Tercera fila & 20 & 21 & 22 & 25 & 27 \\ \hline
Cuarta fila  & 51 & 20 & 23 & 25 & 96 \\ \hline
Quinta fila  & 46 & 33 & 36 & 39 & 56 \\ \hline
Sexta fila   & 23 & 25 & 69 & 96 & 36 \\ \hline
Septima fila & 69 & 55 & 45 & 69 & 89 \\ \hline
Octava fila  & 203 & 36 & 87 & 98 & 46 \\ \hline
\end{tabular}}
\end{table}
```

Ejercicio 2. Realiza un aumento del texto, utilizando los comandos de altura y anchura. Utiliza todos los comandos que conozcas.

Este es un texto con la anchura aumentada.

Este es un texto con la anchura y la altura modificada.

Código 10.24: Solución ejercicio 2

```
\begin{center}
    \scalebox{2}{Este es un texto con la anchura aumentada.}\\
```

```
    \resizebox{0.8\width}{3\height}{Este es un texto con la anchura
        y la altura modificada.}
\end{center}
```

Ejercicio 3. Realiza tres textos: Uno con el comando scalebox, otro con el resizebox y otro con el rotatebox.

Este es un texto normal que doblamos de tamaño.

Le cambiamos un poco y ya se ve la diferencia. Y le damos una vuelta, y otra.

Código 10.25: Solución ejercicio 3

```
\begin{center}
    Este es un texto normal \scalebox{2}{que doblamos de tamaño}.\\
    \resizebox{0.7\width}{2\height}{Le cambiamos un poco} y ya se
       ve la diferencia.
    Y le damos una vuelta, \rotatebox{30}{y otra}.
\end{center}
```

Ejercicio 4. Modifica una imagen en anchura, altura y gírala unos grados.

Figura 10.5: Imagen del gato modificada en anchura, altura y girada

Código 10.26: Solución ejercicio 4

```
\begin{figure}[H]
    \centering
    \includegraphics[width=3cm, height=3cm, angle=90]{Images/GATOEJ
       .jpg}
```

```
    \caption{Imagen del gato modificada en anchura, altura y girada
        }
\end{figure}
```

Ejercicio 5. Realiza un diagrama de barras sencillos con 2 datos en la leyenda.

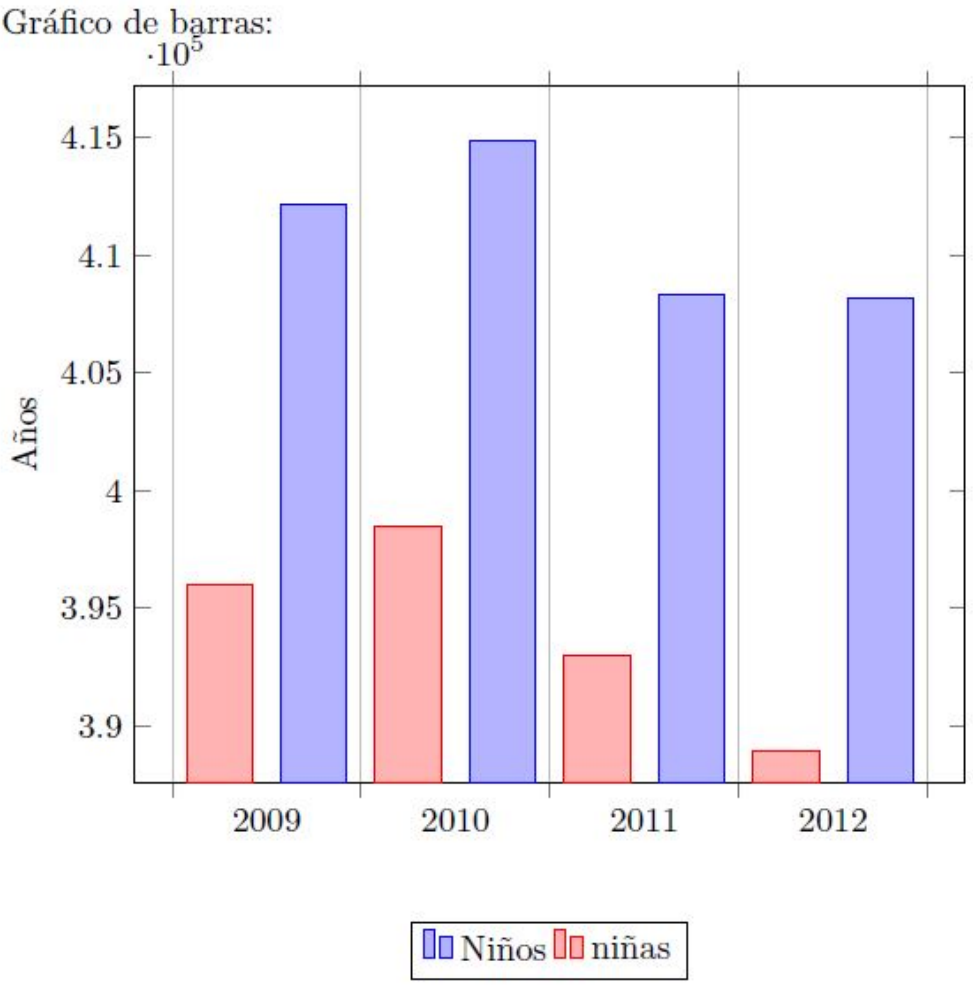


Figura 10.6: Gráficos de barras

Código 10.27: Solución ejercicio 5

```
\documentclass{article}
\usepackage[margin=0.5in]{geometry}
\usepackage[utf8]{inputenc}
\usepackage{textcomp}
\usepackage{pgfplots}

\pgfplotsset{width=10cm,compat=1.9}


\begin{document}

Gráfica de barras:

\begin{tikzpicture}
\begin{axis}[
    x tick label style={
                /pgf/number format/1000 sep=},
        ylabel=Año,
        enlargelimits=0.05,
        legend style={at={(0.5,-0.2)},
        anchor=north,legend columns=-1},
        ybar interval=.7,
]
\addplot
        coordinates {(2012,408184) (2011,408348)
                 (2010,414870) (2009,412156) (2008,415 838)};
\addplot
        coordinates {(2012,388950) (2011,393007)
                (2010,398449) (2009,395972) (2008,398866)};
\legend{Niños,niñas}
\end{axis}
\end{tikzpicture}
\end{document}
```

Ejercicio 6. Realiza un sencillo diagrama de círculo.

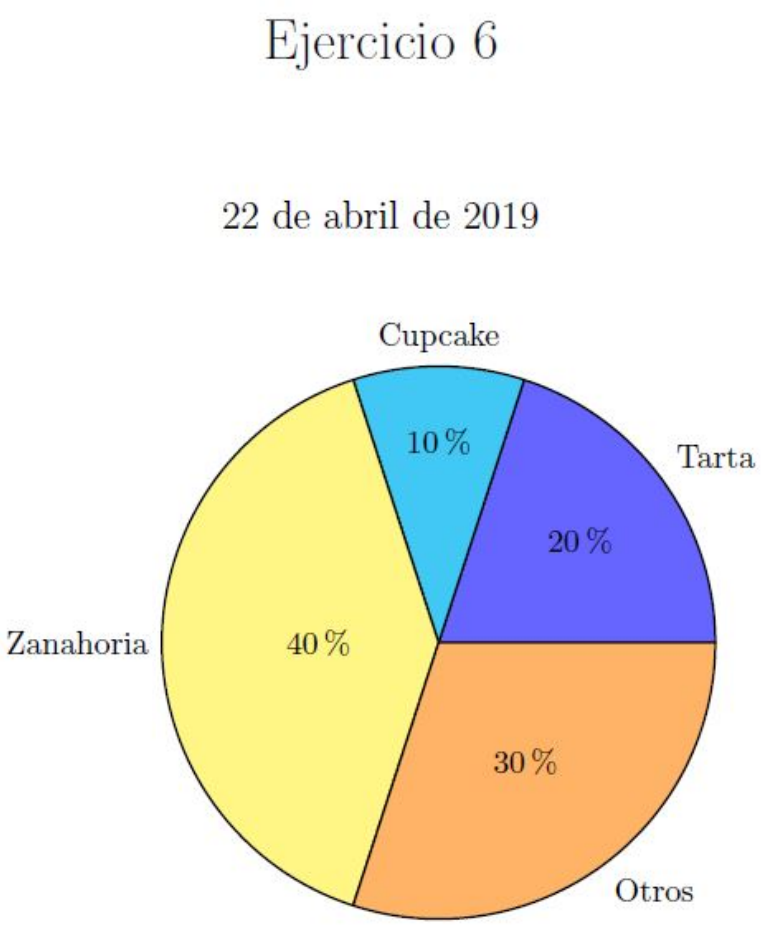


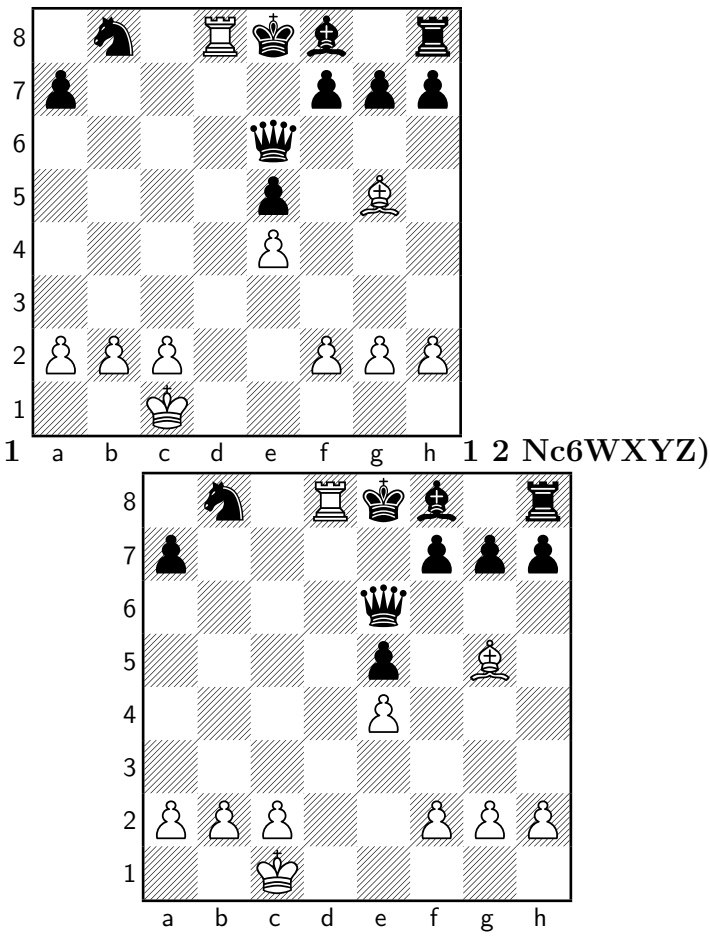
Figura 10.7: Diagrama de círculo sencillo

Código 10.28: Solución ejercicio 6

```
1 \documentclass{article}
2 \usepackage[spanish]{babel}
3 \usepackage{fullpage}
4 \usepackage{pgf-pie}
5 \title{Ejercicio 6}
6
7 \begin{document}
8 \maketitle
9 \centering
```

```
\begin{tikzpicture}
\pie{20/Tarta, 10/Cupcake, 40/Zanahoria, 30/Otros}
\end{tikzpicture}
\end{document}
```

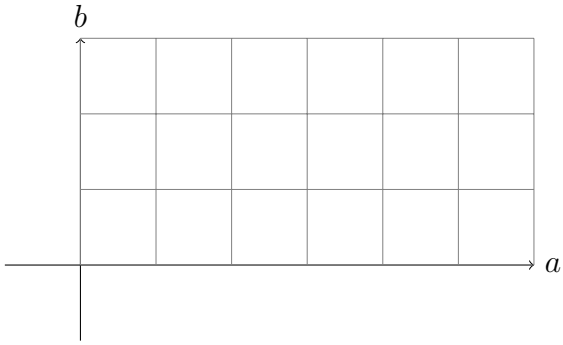
Ejercicio 7. Realiza una pequeña partida de ajedrez sencilla.



Código 10.29: Solución ejercicio 7

```
\begin{center}
\mainline{1.e4}
\showboard
\mainline{1...e5 2.Nf3 Nc6 3.d4}
\showboard
\end{center}
```

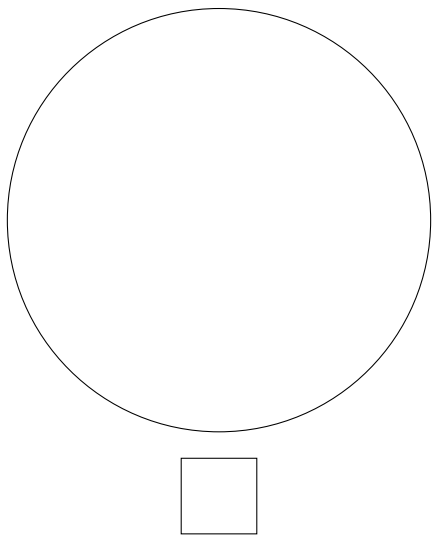
Ejercicio 8. Intenta realizar una rejilla sencilla de proporciones distintas a las del ejemplo mostrado en este libro.

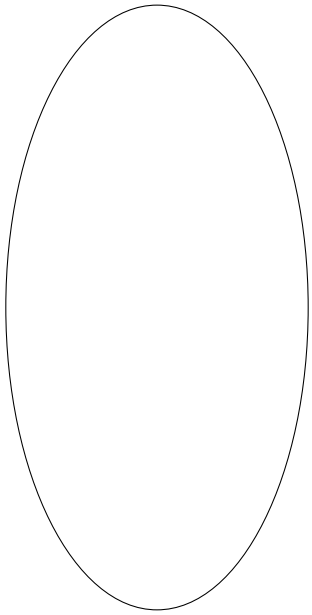


Código 10.30: Solución ejercicio 8

```
\begin{center}
\begin{tikzpicture}
\draw[->] (-1,0) -- (6,0);
\draw (6,0) node[right] {$a$};
\draw [->] (0,-1) -- (0,3);
\draw (0,3) node[above] {$b$};
\draw [very thin, gray] (0,0) grid (6,3);
\end{tikzpicture}
\end{center}
```

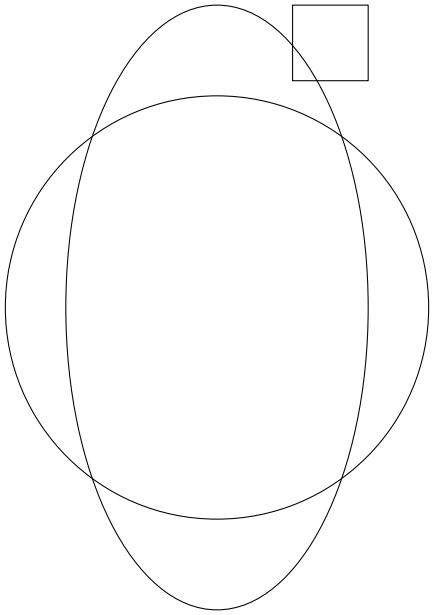
Ejercicio 9. Dibuja un círculo, un rectángulo y una elipse.





Código 10.31: Solución ejercicio 9, primera alternativa

```
\begin{center}
\begin{tikzpicture}
\draw (0,0) circle (2.8cm);
\end{tikzpicture}
\end{center}
\begin{center}
\begin{tikzpicture}
\draw (1,3) rectangle (2,4);
\end{tikzpicture}
\end{center}
\begin{center}
\begin{tikzpicture}
\draw(0,0) ellipse (2 cm and 4 cm);
\end{tikzpicture}
\end{center}
```



Código 10.32: Solución ejercicio 9, segunda alternativa

```
\begin{center}
\begin{tikzpicture}
\draw (0,0) circle (2.8cm);
\draw (1,3) rectangle (2,4);
\draw(0,0) ellipse (2 cm and 4 cm);
\end{tikzpicture}
\end{center}
```

Ejercicio 10. Realiza un texto que contenga al menos una variación en el aumento de la fuente y un giro, añádele alguna imagen que esté modificada y girada. Y por último, añádele una figura geométrica.

Con esto pretendemos, que el lector suba de nivel en LaTeX, no sólo eso sino que además conozca todo lo que pueda hacer con esta magnífica herramienta .

	Primera columna	**Segunda columna**	**Tercera columna**	**Cuarta columna**	**Quinta columna**
Primera fila	1	2	4	3	5
Segunda fila	10	11	12	13	14
Tercera fila	20	21	22	25	27
Cuarta fila	51	20	23	25	96
Quinta fila	46	33	36	39	56
Sexta fila	23	25	69	96	36
Septima fila	69	55	45	69	89
Octava fila	203	36	87	98	46

También se permite utilizar el escalar en una tabla cualquiera. Ahora vamos a mostrar una imagen modificada y rotada .

Figura 10.8: Gato modificado y girado

Código 10.33: Solución ejercicio 10

```
\begin{center}
    Con esto pretendemos, que el lector \scalebox{1.2}{suba de
       nivel en \LaTeX{}}, no sólo eso sino que además conozca \
       resizebox{0.7\width}{1.2\height}{todo lo que pueda hacer}
       con \rotatebox{60}{esta magnífica herramienta}.
\end{center}
\begin{table}[H]
\centering
\scalebox{0.7}{\begin{tabular}{|c|c|c|c|c|c|}
\hline
            & \textbf{Primera columna} & \textbf{Segunda columna}
               & \textbf{Tercera columna} & \textbf{Cuarta columna
               } & \textbf{Quinta columna} \\ \hline
Primera fila & 1                        & 2
   & 4                        & 3                        & 5
                        \\ \hline
Segunda fila & 10                       & 11
   & 12                       & 13                       & 14
                       \\ \hline
Tercera fila & 20                       & 21
   & 22                       & 25                       & 27
                       \\ \hline
Cuarta fila  & 51                       & 20
   & 23                       & 25                       & 96
                       \\ \hline
Quinta fila  & 46                       & 33
   & 36                       & 39                       & 56
                       \\ \hline
Sexta fila   & 23                       & 25
   & 69                       & 96                       & 36
                       \\ \hline
Septima fila & 69                       & 55
   & 45                       & 69                       & 89
                       \\ \hline
Octava fila  & 203                      & 36
   & 87                       & 98                       & 46
                       \\ \hline
\end{tabular}}
\end{table}
```

```
\noindent
También se permite utilizar el escalar en una tabla cualquiera.
   Ahora vamos a mostrar una imagen {\resizebox{1.2\width}{1.5\
   height}{\rotatebox{45}{modificada y rotada}}}.
\begin{figure}[H]
    \centering
    \includegraphics[width=6 cm, height=7 cm, angle=175]{Images/
       GATOEJ.jpg}
    \caption{Gato modificado y girado}
\end{figure}
```

Índice de figuras

Índice de cuadros

Lista de Códigos