

docuLaTeX_{v1.0.0}

José Alberto López López

Contenido

1	Introducción	1
2	Instalación	1
3	Inicio rápido	1
4	Características principales	4
4.1	Niveles de anidación	5
5	Entornos	5
5.1	Entornos de código	6
5.2	Entornos de lista	9
5.3	Entornos de descripción especiales	10
5.4	Otros entornos	11
6	Delimitadores	12
6.1	Consideraciones	14
7	Secciones	15
7.1	Secciones de librerías	16
8	Términos reservados	17
9	Acciones específicas, errores y buenas prácticas	18
10	Entornos y elementos de lista	18
10.1	Descripciones	19
11	Tablas	21
12	Comentarios	21
13	Referencias informales	22

14 Configuraciones JSON	23
15 CLI	27
16 Archivos de definición de lenguajes	29
17 Logs	31
17.1 Estado de línea	31
17.2 Comparación de líneas	31
18 Problemas conocidos	31
19 TODO	32
20 Licencia	33

Introducción

En este proyecto se define un formato práctico de texto plano para escribir documentación de código de programación, junto con un parser que convierte dicho formato en archivos de salida $\text{\LaTeX}$, mediante los cuales se puede producir un archivo PDF final mediante los motores $\text{\LuaTeX}$ ó $\text{\XeTeX}$, con el mismo contenido de la documentación en **TXT** pero con un estilo agradable y profesional, el cuál está inspirado en el estilo del manual TikZ & PGF.

El objetivo principal es proporcionar un formato simple para la escritura práctica de documentación de código, con un formato final de calidad, generado automáticamente mediante $\text{\LaTeX}$, pero a partir de una sintaxis más corta y accesible.

Instalación

El programa está disponible como un paquete Python, por lo cual se puede instalar fácilmente mediante `pip` o `pip3`, como cualquier otro paquete o librería, mediante un comando como el siguiente:

Comando 2.1

```
pip3 install doculate
```

Inicio rápido

Una vez instalado el paquete, este se puede utilizar mediante el comando `doculate`, indicando el nombre de un archivo **TXT** de entrada como argumento, el cual debe estar en la ruta donde se ejecute el comando, o bien indicar su ruta y nombre. Además el contenido del archivo debe tener la sintaxis que se describe en esta documentación o se podrían generar errores.

Comando 3.1

```
doculate archivo_entrada.txt
```

A continuación se da un ejemplo del contenido que puede tener el archivo de entrada, con varios de los componentes sintácticos definidos por defecto. Solo se debe asegurar que las líneas indentadas utilicen tabuladores y no espacios (se recomienda hacer una búsqueda y reemplazo de cuatro espacios `UUUU` por tabulador `\t`).

Ejemplo 3.1

Python

El lenguaje de programación Python fue lanzado en el año 1991.
Es un lenguaje interpretado con una sintaxis amigable.

A continuación se presenta un ejemplo de código escrito en Python:

Ejemplo:

```
import pandas as pd
from numpy import array

# Comentario de línea
x = array([1, 2, 3])

s = "Esto es una cadena"
print(s)

def my_func(a: int, b: float):
    return a + b
```

Algunas de las desventajas principales que presenta frente a otros lenguajes son :

- Velocidad: es más lento que la mayoría de los lenguajes compilados.
- Alto nivel: es una desventaja si se requiere mayor control de aspectos como el uso de memoria.

Es b"importante" mencionar que con la ayuda de algunas librerías se puede aumentar considerablemente la eficiencia.

Términos reservados

En esta sección se presentan algunas de las s"palabras clave" de Python.

*for

Instrucción para ejecutar un bucle o i"loop".

Sintaxis:

```
for num in range(num_inicial, num_final):
    # Realizar acciones...
```

Se puede utilizar la sentencia ``break`` para q"abortar" la ejecución del bucle.

`*if`
`*elif`
`*else`

Estructuras de control para sentencias condicionales.
Sus características son:

- Si la condición de ``if`` se evalúa a ``True``, se ejecutará lo que haya en su bloque.
- Si la condición de ``if`` se evalúa a ``False`` y la de ``elif`` a ``True``, se ejecutará el contenido de ``elif``.
- Si las condiciones de ``if`` y ``elif`` se evalúan a ``False`` o ``None``, se ejecutará el bloque ``else``.

En las condiciones se pueden usar operadores booleanos como ``and``, ``or`` y ``not``.

`*print()`

Función para imprimir información.

Opciones:

- ``sep``: separador de cada argumento a imprimir.
- ``end``: elemento concatenado al final de la cadena impresa.
- ``file``: dirige la salida hacia el archivo, en vez de la terminal.

Sus argumentos se pueden separar con comas, por ejemplo: ``print("Hola", "mundo!")``.

Con el archivo anterior, se puede especificar la opción `--programming-languages` para aplicar resaltado de sintaxis predefinido para Python.

Comando 3.2

```
doculate archivo_entrada.txt --programming-languages Python
--title "Ejemplo Python" --author Anónimo
```

Después de ejecutar el comando anterior, se deberían de generar los archivos `.tex` necesarios para generar el PDF final.

Características principales

El procesamiento del parser consiste en analizar línea a línea el documento `TXT` para convertir o mapear cada una de ellas en el formato `LATEX` correspondiente.

Una línea puede ser identificada como alguno de los siguientes elementos:

- Título de sección.
- Línea de texto.
- Línea en blanco.
- Inicio de un entorno.
 - Contenido de entorno de código.
 - Inicio de elemento de lista, numeración o descripción.
 - Contenido de elemento de lista, numeración o descripción.
 - Contenido de entorno matemático.
 - Fila de tabla.
- Término reservado.
- Acción específica.
- Buena práctica.
- Error.
- Contenido de término reservado, acción específica, buena práctica o error.
- Título de párrafo.
- Comentario de una sola línea.
- Inicio de comentario multilínea.
 - Contenido de comentario multilínea.
- Final de comentario multilínea.
- Referencia informal.

Niveles de anidación

Cada vez que se inicie un entorno mediante su identificador correspondiente, su contenido deberá estar indentado un nivel adicional respecto al identificador.

Las indentaciones se deben realizar mediante tabuladores, uno para cada nivel adicional.

Cada tabulación que se realice al inicio de una línea corresponderá a un nivel de anidación adicional, a menos que dicha línea corresponda a un entorno cuyas líneas son ignoradas, por ejemplo, entornos de código; o bien, que la línea corresponda a un patrón ignorable, por ejemplo, líneas vacías o de comentario.

Un elemento que tenga menos indentaciones que el que lo precede representa el fin de uno o más entornos o elementos previos.

Entornos

Existen 2 tipos principales de entornos: de lista y de código. Los entornos de lista pueden ser listas normales con viñeta, enumeraciones o descripciones. Los entornos de código pueden ser de diferentes tipos. A continuación se muestran ejemplos de cómo generar un entorno de código, uno de lista, uno de enumeración y uno de descripción.

Ejemplo 5.1

Texto previo.

Ejemplo:

 contenido de entorno de código

Más texto...

 -Elemento de lista 1.

 Continuación o contenido de elemento 1.

 -Elemento de lista 2.

 -Elemento de lista 3.

Más texto...

 1. Elemento de enumeración 1.

 Continuación o contenido de elemento 1.

 2. Elemento de enumeración 2.

```

3. Elemento de enumeración 3.

Más texto...

-Elemento 1: descripción de elemento 1.
  Continuación o contenido de elemento 1.
-Elemento 2: descripción de elemento 2.
-Elemento 3: descripción de elemento 3.

```

Como se puede observar, el identificador que da inicio a los entornos de lista (como **Ejemplo:** en entornos de código) es opcional.

Los tipos de entornos disponibles, así como los identificadores que los inician se listan a continuación.

Entornos de código

Los identificadores de entornos de código o de lista se pueden redefinir o extender mediante las opciones de configuración JSON: `environments.<id_env>.values` y `environments.<id_env>.add_values` (ver *Configuraciones JSON*).

Ejemplo Identificadores: **Ejemplo**, **Ej..**

Ejemplos Identificadores: **Ejemplos**, **Ejs..**

Antiejemplo Identificadores:

```

Antiejemplo
Anti-ejemplo
Ant. ej.
A. ej.

```

Antiejemplos Identificadores:

```

Antiejemplos
Anti-ejemplos
Ant. ejs.
A. ejs.

```

Sintaxis Identificadores: **Sintaxis**.

Sintaxis código Identificadores: Sintaxis código.

Código Identificadores: Código, Cod..

Comando Identificadores: Comando, Comandos código.

Configuración Identificadores:

Configuraciones

Configuración

Confs.

Conf.

Definición Identificadores: Definición, Def..

Definiciones Identificadores: Definiciones, Defs..

Código ejemplo Identificadores:

Código ejemplo

Cod. ej.

Códigos ejemplo

Cods. ej.

Código antiejemplo Identificadores:

Código antiejemplo

Códigos antiejemplo

Cod. antiej.

Cods. antiej.

Cod. aej.

Cods. aej.

Función Identificadores:

Función	Cód. función
Func.	Cód. func.
Funciones código	Código funciones
Funciones cód.	Cód. funciones
Funcs. cód.	Cód. funcs.
Código función	

Procedimiento Identificadores:

Procedimiento	Cód. procedimiento
Proc.	Cód. proc.
Procedimientos código	Código procedimientos
Procedimientos cód.	Cód. procedimientos
Procs. cód.	Cód. procs.
Código procedimiento	

Clase Identificadores:

Clase	Cls. cód.
Clase código	Código clase
Clases código	Código clases
Clase cód.	Cód. clases
Clases cód.	Cód. clas.
Clas. cód.	Cód. cls.

Los IDs de configuración JSON de estos entornos son:

<code>syntax</code>	<code>antiexample</code>	<code>codeantiexample</code>	<code>function</code>
<code>codesyntax</code>	<code>antiexamples</code>	<code>commandscodesyntax</code>	<code>procedure</code>
<code>example</code>	<code>code</code>	<code>configuration</code>	<code>definition</code>
<code>examples</code>	<code>codeexample</code>	<code>class</code>	<code>definitions</code>

El color de las barras de título de los entornos de código se puede reestablecer mediante la configuración `code_environments_title_colors`. Si solo se quiere cambiar el color de la barra para algunos entornos se puede usar la configuración `add_code_environments_title_colors`, ya que con `code_environments_title_colors` el color de la barra para los entornos que no se hayan establecido será el color por defecto indicado en la configuración `code_title_color`. También se puede cambiar el color de fondo del contenido de entornos de código mediante la configuración `code_background_color`, y el de las líneas del marco que contiene el cuadro de código mediante `code_frame_color`. Los colores se pueden indicar en RGB, mediante 3 números enteros separados por comas del 0 al 255 o mediante el formato de nombres mostrado en sus valores por defecto en la sección *Configuraciones JSON*.

El nombre del entorno mostrado en la barra de título es el primer valor de la lista dada en la configuración `environments.<id_env>.values`. Actualmente no se pueden definir y agregar nuevos entornos, pero un truco que se puede hacer para emular esto es cambiar el primer valor de la lista `environments.<id_env>.values` de un entorno que no se vaya a utilizar, por el nombre del entorno deseado, y ya que `<id_env>` solo actúa como identificador a nivel de programación, el nombre que aparecerá en el documento será el que se haya establecido.

Entornos de lista

Cómo se mencionó anteriormente, los identificadores de los siguientes entornos de lista son opcionales.

Lista Identificadores: `Lista`.

Enumeración Identificadores: `Enumeración`, `Enum..`

Descripción Identificadores: `Descripción`.

El contenido de estos entornos debe estar indentado un nivel adicional en cualquier caso. Asimismo, el contenido de un elemento de lista, después de la primera línea que contiene su viñeta, también deberá estar indentado un nivel.

Los IDs de configuración JSON de estos entornos son: `list`, `enumerate`, y `description`.

Entornos de descripción especiales

La mayoría de los siguientes entornos de descripción están pensados para describir componentes de código, como opciones, argumentos de funciones, miembros de clases, etc.

Opciones Identificadores: `Opciones`.

Campos Identificadores: `Campos`, `Llaves`, `Claves`.

Argumentos Identificadores:

`Argumentos`

`Args`.

`Argumentos requeridos`

`Args. req.`

`Argumentos opcionales`

`Args. opt.`

Parámetros Identificadores:

`Parámetros`

`Params`.

`Parámetros requeridos`

`Params. req.`

`Parámetros opcionales`

`Params. opt.`

Clases Identificadores: `Clases`.

Miembros Identificadores: `Miembros`.

Métodos Identificadores: `Métodos`, `Méts..`

Atributos Identificadores: `Atributos`, `Atribs..`

Valores Identificadores:

`Valores`

`Vals`.

`Posibles valores`

Pos. vals.

Constantes

Opciones de configuración Identificadores:

Opciones de configuración

Opc. conf..

Comandos Identificadores: Comandos.

Subcomandos Identificadores: Subcomandos, Subcom..

Funciones Identificadores: Funciones, Funcs..

Procedimientos Identificadores: Procedimientos, Procs..

Los IDs de configuración JSON de estos entornos son:

options	functions
values	members
arguments	methods
attributes	fields
parameters	commands
classes	subcommands
configurationoptions	procedures

Otros entornos

Los siguientes son entornos alternativos para insertar expresiones matemáticas, tablas o insertar código $\text{\LaTeX}$ directamente. Al igual que los identificadores de entornos de lista, el identificador de tabla también es opcional.

Ecuación Identificadores:

Texto matemático

Mat.

Ecuación

Ec.

Expresiones matemáticas Identificadores:

Expresiones matemáticas

Ecuaciones

Mats.

Ecs.

Expresiones matemáticas alineadas Identificadores:

Expresiones matemáticas alin.

Ecuaciones alin.

Alin. Mats.

Alin. Ecs.

A. Ecs.

Tabla Identificadores: Tabla.

LaTeX Identificadores: LaTeX, Latex, latex.

Comentario Identificadores: Comentario, Comentarios.

Los IDs de configuración JSON de estos entornos son:

```
math    mathexpressions    comment
table   alignedmathexpressions  latex
```

Delimitadores

Una línea a de texto puede contener uno o más delimitadores de modo línea, incluyendo anidaciones de estos. Los delimitadores permiten dar distintos formatos o estilos a porciones de texto.

Existen 2 clases principales de delimitadores: normales y de no anidación. Los normales permiten anidar otros delimitadores dentro de estos, son útiles, por ejemplo, para resaltar texto en formato de código dentro de texto en itálicas o cosas por el estilo. Los de no anidación toman su contenido tal cual, hasta encontrar el delimitador de cierre, por lo que son adecuados para insertar código entre texto.

Varios de los delimitadores están definidos mediante un caracter o cadena constructora diferente para cada uno, a partir de la cual se establecen tres variantes de un mismo tipo de delimitador: normal, de no anidación y normal sin ambigüedad, mediante los caracteres base por defecto ' , " y { , } , respectivamente. Por ejemplo, el caracter constructor de las variantes para el delimitador de negritas (*bold*) es **b**, generando las variantes **b' . . . '**, **b" . . . "** y **b{ . . . }**.

Los delimitadores normales son más prácticos, pero también más propensos a errores. Cuando el caracter o cadena previa a un delimitador normal de cierre coincide con el caracter constructor de otro delimitador, el parser lo identificará erróneamente como delimitador de apertura. Para estos casos es que existen los delimitadores normales no ambiguos, que tienen mucha menos probabilidad de caer en dichas situaciones.

A continuación se listan los diferentes tipos de delimitadores predefinidos. A menos que se indique otra cosa, estos están definidos de la forma descrita, mediante el caracter constructor que se indique.

Texto normal sin formato caracter constructor: **t**.

Negritas caracter constructor: **b**.

Itálicas normales: **i' . . . '**, **i" . . . "**, **i{ . . . }**; no anidación: **" . . . "**.

Inclinado caracter constructor: **s**.

Mayúsculas (*small caps*) caracter constructor: **sc**.

Comillas caracter constructor: **q**.

Subrayado caracter constructor: **u**.

Código no anidación: **` . . . `**.

Código referenciado no anidación: **r` . . . `**. Aún no implementado.

Código especial no anidación: **e" . . . "**. Código con el caracter **`**.

Código especial referenciado no anidación: **re" . . . "**. Aún no implementado.

Matemáticas no anidación: **\$. . . \$**.

Nombre de archivo no anidación: **f" . . . "**.

Ruta de directorio no anidación: **d" . . . "**..

Etiqueta referenciable no anidación: **l" . . . "**.

Referencia a etiqueta caracter constructor: **r**. Aún no implementado.

Cita no anidación: **c" . . . "**. Aún no implementado.

Nota al pie caracter constructor: **fn**.

Vínculo web no anidación: `h"..."`.

Tecla no anidación: `k"..."`.

Botón genérico no anidación: `bu"..."`.

Comentario entre línea no anidación: `cc"..."`.

LaTeX no anidación: `L"..."`, `LaT"..."`, `LaTeX"..."`. Inyección de código $\LaTeX$.

Los identificadores de configuración JSON (donde se indique `<delim_id>`) de los delimitadores son: `italics`, `normal_text`, `bold`, `math`, `slanted`, `small_caps`, `quotes`, `underline`, `code`, `referenced_code`, `special_code`, `referenced_special_code`, `comment`, `file`, `directory`, `label`, `reference`, `cite`, `footnote`, `hyperlink`, `keyboard`, `button` y `LaTeX`.

Tanto los caracteres constructores como los caracteres base por defecto se pueden cambiar mediante las siguientes configuraciones:

- `delimiters.<delim_id>.signature_constructor`
- `default_inline_normal_delimiter`
- `default_inline_no_nesting_delimiter`
- `default_unambiguous_normal_delimiters`

También se pueden añadir nuevos delimitadores a los existentes mediante un caracter constructor o lista de estos en la configuración `delimiters.<delim_id>.add_signature_constructor`.

O bien se pueden redefinir o agregar delimitadores explícitamente como pares de cadenas arbitrarias, mediante las configuraciones:

- `delimiters.<delim_id>.normal`
- `delimiters.<delim_id>.add_normal`
- `delimiters.<delim_id>.no_nesting`
- `delimiters.<delim_id>.add_no_nesting`

En la sección *Configuraciones JSON* se pueden consultar ejemplos de estas configuraciones.

Consideraciones

Se debe tener cuidado de abrir y cerrar los delimitadores de forma correcta, sobre todo los anidados; así como verificar que no haya ambigüedad en los normales de cierre. Por lo cual

se recomienda preferir los delimitadores de no anidación (*no nesting*) y usar los normales solo cuando sea realmente necesario, prefiriendo también los no ambiguos.

Si un caracter o conjunto de caracteres, ingresado para que sea interpretado como texto coincide con algún delimitador, dicha cadena de caracteres se debe escapar mediante `\`. Si no se escapa, es posible que el parser la interprete como un delimitador, causando un error o comportamiento no deseado. Por ejemplo, el caracter `}` por defecto es considerado un delimitador de cierre y además como un caracter con reemplazo especial, que debe traducirse al archivo `.tex` como `\}`. Este es un caso especial donde el caracter se debe escapar cuando se requiera que se lea como texto (a menos que se redefinan los delimitadores), ya que de lo contrario pueden haber situaciones donde previamente se utilice el delimitador de apertura correspondiente y el caracter se interprete como su cierre de manera errónea.

El parser de antemano ignora el reemplazo de las cadenas especiales que coinciden con algún delimitador, dando prioridad a que sean tratadas como delimitadores y haciendo que el usuario deba escaparlas manualmente cuando requiera que sean tratadas como texto.

Secciones

Se permiten hasta seis niveles de secciones y dos niveles finales de título de párrafo y título de subpárrafo. Por ejemplo, un documento se podría dividir en: libro, parte, capítulo, sección, subsección, subsubsección; utilizando todos sus niveles de secciones y de párrafos, o bien comenzar desde un nivel más bajo, por ejemplo, capítulo, que es como está definido por defecto.

El nivel raíz de secciones se puede establecer mediante la configuración `root_section`. Esta configuración puede ser útil para establecer el formato y espaciado de secciones de forma práctica. Por ejemplo, en documentos pequeños, se puede establecer el valor a `section` o `subsection` para un cambio continuo de secciones, sin cambiar de página, ya que el valor por defecto `chapter` hace que se cambie de página para iniciar cada capítulo, lo cual hacer ver secciones con poco contenido muy aisladas.

De forma análoga a Markdown, el primer nivel de sección se indica mediante un `#`, el nivel dos mediante `##` y así sucesivamente hasta llegar al nivel cuatro (o seis si se usan todas). El título de un párrafo se indica mediante `#####` y el de un subpárrafo mediante `#####`, independientemente del nivel de sección actual en el documento.

Se puede cambiar el caracter que identifica el título de una sección (por defecto `#`) mediante la configuración `header_mark`.

Secciones de librerías

Se define un símbolo `+` (configuración `library_mark`) para diferenciar la apertura de una sección de librería, framework, paquete, etc., de la de una sección normal, indicadas mediante el símbolo `#`. Lo cual significa que la sección llevará el nombre de la librería, posiblemente con un formato especial y, opcionalmente, algún prefijo o sufijo, como la palabra “Librería”, “Framework”, etc., antes del nombre de esta. Por lo que la información que se presente en la sección estará relacionada a dicha librería, hasta que esta termine.

El uso de `+` es análogo a `#`. Por defecto, el mismo número de `+` y `#` indican el mismo nivel de sección. Por lo cual, patrones de subsecciones como `++` ó `+++` se puede usar para abrir secciones de sublibrerías, submódulos, etc., definidas dentro de una librería más grande; o bien para comenzar secciones de librerías dentro de otras secciones normales.

Otra característica de esta sintaxis para secciones de librerías es que, si dos o más del mismo nivel se colocan en líneas consecutivas, estas se colocaran en el mismo comando de sección, y aparecerán en el PDF con un formato determinado; por ejemplo, separadas por comas o apiladas una debajo de otra. Esto puede ser útil, por ejemplo, para documentar librerías que tengan alias populares y se quiera incluir tanto este como el nombre oficial en el título de sección; o bien, incluir los identificadores principales con los que se importan, cuando exista más de uno. De esta forma no se rompe con la sintaxis homóloga si se quiere hacer una búsqueda rápida en el TXT de cualquiera de los identificadores de librerías.

Si no se desea utilizar este diferenciador de secciones para librerías, se puede seguir utilizando `#` sin problema, incluso con delimitadores de formato.

El siguiente es un ejemplo de un posible uso de las secciones de librerías.

Ejemplo 7.1

```
+ numpy
+ np

Texto...

*Acción específica numpy.
  Texto...

Texto...

*funcionDeNumpy()
  Texto...

Texto...

# Gráficos
```

```
Texto...

++ matplotlib

Texto...
```

Términos reservados

La documentación de un término reservado se identifica cuando una línea inicia con el caracter `*` (por defecto) y esta no termina con un punto, en cuyo caso se trataría de una acción específica, buena práctica o error.

Lo que siga después del caracter `*` es colocado en el PDF con un formato de código y resaltado de sintaxis si aplica, sin necesidad de usar delimitadores de código. El contenido subsecuente relacionado al término deberá estar indentado un nivel.

Los términos reservados son elementos principales de una documentación de código, por lo cual se considera que estos, al igual que las acciones específicas, buenas prácticas y errores, prácticamente siempre estarán en el nivel más superior del documento, es decir, sin anidarse en otros entornos. Es por ello que la forma de establecer cuándo se inicia, cuándo se está dentro del contenido de un término y cuándo este finaliza, es distinta a la de un entorno de lista o de código.

La forma de detectar el inicio de documentación de un término es cuando se detecta su patrón en una línea. A partir de ahí se debe indentar un nivel el contenido del término, y la forma de finalizar el pseudoentorno del término solo se dará mediante alguna de las siguientes situaciones:

- Mediante el inicio de otro término, acción específica, buena práctica o error.
- Mediante el inicio de una nueva sección, subsección, etc.
- Al continuar con texto u otro contenido que ya no pertenece a la documentación del término, removiendo el tabulador inicial del contenido del término.

Todo el contenido entre el inicio de un término y alguna de las situaciones anteriores será considerado como parte de su documentación.

No se espera ni se incentiva la situación en que se tenga que anidar la documentación de un término dentro de otro entorno, ya que para ello están los entornos como `Opciones`, donde se pueden documentar subtérminos relacionados a un término principal.

Acciones específicas, errores y buenas prácticas

Las acciones específicas, errores y buenas prácticas son elementos de documentación alternativos a los términos reservados, con contenido de temas concretos, como sus nombres lo indican. Siguen una sintaxis similar a la de los términos: por defecto inician con el caracter `*` seguido del título en cuestión, el cual debe finalizar con un punto.

Ejemplos de títulos de acciones específicas, errores o buenas prácticas pueden ser:

- *Medir el tiempo de ejecución de un bloque de código.*
- *Error de conversión de enteros a cadenas.*
- *Usar sentencias cortas para mejorar legibilidad.*

en los cuales también se pueden usar delimitadores de formato.

Se puede establecer un caracter para identificar el título de una acción, error o buena práctica mediante la configuración `action_error_best_practice_mark`. Actualmente estos 3 elementos son indistinguibles entre sí en el documento final, solo pudiendo diferenciarlos por el título de la sección en la que se encuentren y su título mismo.

Al igual que los términos reservados, su contenido debe estar indentado un nivel.

Entornos y elementos de lista

El contenido (i.e. líneas subsecuentes) de un elemento de lista debe anidarse una indentación adicional a la línea que contiene la viñeta del elemento.

En ausencia de identificador de entorno o diferentes viñetas establecidas en la configuración, el patrón del primer elemento, en conjunto con los valores de configuración para listas, determinan qué entorno de lista (normal o de descripción) se utilizará para elementos de lista consecutivos posteriores.

Debido a que la línea inicial de un elemento de lista o elemento de descripción pueden coincidir en su patrón, tanto si cualquiera -en su primera línea- incluye o no dos puntos o punto final, se establecen tres formas de diferenciar dichos elementos para evitar ambigüedades:

1. Establecer viñetas `TXT` distintas para el elemento de lista y el elemento de descripción.
2. Iniciar los entornos explícitamente con los identificadores `Lista` o `Descripción`, ó sus variantes.

3. Mantener el valor de la configuración `colon_rule_when_no_identifiers` en verdadero, la cual señala que, si no se indica el identificador de entorno de lista o descripción explícitamente, las listas nunca tendrán el signo de dos puntos en la primera línea de su primer elemento y las descripciones sí lo tendrán para evitar ambigüedades, lo cual implica que el usuario se haga responsable de cumplir con la regla, si no quiere obtener comportamientos no deseados.

Se debe optar por usar al menos una de las tres alternativas, de lo contrario podría producirse un error por ambigüedad, o bien, si la variable `default_list_elem` está activa, se usará el entorno `Lista` por defecto.

Descripciones

Existen dos tipos de formato para descripciones: en la misma línea o en la siguiente línea, que consisten simplemente en colocar el elemento a describir en la misma línea que su descripción o colocar la descripción debajo de este.

A continuación se dan ejemplos de los formatos aceptados para los dos tipos de descripciones, recordando que el formato para todos los elementos lo establece el primero:

Ejemplo 10.1

Texto previo...

```
%% Descripción en línea aparte
-Elemento 1:
    Texto de descripción...
-Elemento 2:
    Texto de descripción...
```

Texto para finalizar entorno implícito e iniciar el siguiente...

```
%% Descripción en línea aparte
-Elemento 1
    Texto de descripción...
-Elemento 2: Texto de descripción...
```

Texto previo seguido de identificador (opcional) de entorno...

Descripción:

```
%% Descripción en la misma línea
-Elemento 1: Texto de descripción...
-Elemento 2:
    Texto de descripción larga...
-Elemento 3: Texto de descripción...

Texto posterior...
```

Las descripciones en misma línea siempre deben llevar :_□ como separador entre el término y su descripción.

Si se tiene una lista de descripciones sin especificar el entorno e incompleta, a la que aún no se han agregado las descripciones a sus elementos y no se quiere que se trate como una lista normal con viñetas, solo se debe agregar dos espacios al final del primer elemento. Eso será suficiente para que se interprete como entorno de descripción y que los elementos siguientes sean tratados como elementos de descripción. Por ejemplo:

Ejemplo 10.2

```
-`visible`□□
-`hidden`
```

O bien se le puede agregar una descripción solo al primer elemento para no usar este truco.

Los elementos del entorno en el que se use esta sintaxis serán tratados con un formato de descripción sobre la misma línea, debido a que es más adecuada para descripciones cortas. Los elementos de descripción de línea aparte son más adecuados para descripciones más largas o formales, por lo cual en estos se incentiva a que se incluyan tanto el nombre del entorno como las descripciones de los elementos desde un inicio. Aunque las descripciones de línea aparte también podrían nacer como descripciones prácticas sobre la misma línea, sin nombre de entorno ni contenido de elementos, y posteriormente cambiar muy fácilmente a una descripción de línea aparte.

Si por alguna razón se quiere que un entorno de descripción en formato de línea aparte se interprete como formato sobre la misma línea, se puede hacer lo mismo, es decir, agregar dos espacios al final del primer elemento.

Ejemplo 10.3

Opciones:

```
-`visible`□□
    descripción del elemento.
-`hidden`
```

Por las razones mencionadas no existe, y tampoco se incentiva, una forma de interpretar un entorno de descripción en la misma línea en el **TXT** como de línea aparte en el documento de salida.

Por defecto el texto de los elementos a describir aparecerá en negritas, pero se pueden utilizar delimitadores para aplicar formato tanto en el elemento como en su descripción. Por ejemplo, delimitadores de código (``...``) en elementos de entornos como **Opciones**, como se muestra en el ejemplo anterior, ya que aún no se aplica este formato por defecto en tales entornos.

Tablas

El distintivo del patrón de una fila de tabla es que el contenido de sus celdas debe estar separado por uno o más tabuladores.

Al igual que los entornos de lista, una tabla puede ser insertada con o sin su identificador **Tabla**.

Para tablas de una sola columna, estas deben estar en un entorno explícito de tabla (que inicie con identificador), o bien, en entornos implícitos, cada fila deberá terminar con un tabulador. Por lo que, en entornos implícitos de tablas de una sola columna, no es opcional que el elemento termine con uno o más tabuladores, en este caso siempre se debe finalizar con un tabulador cada línea que corresponda a una fila de la tabla.

No se puede anidar nada dentro de una fila de tabla. En otras palabras, no puede haber nada en un entorno de tabla que esté más anidado que su primera fila.

Si se requiere una celda vacía, simplemente se tiene que colocar un espacio en blanco en la celda correspondiente.

A partir de la primera fila de la tabla se infiere el número total de columnas de esta, por lo cual, si tiene celdas vacías, se deben indicar mediante un espacio en blanco. Las celdas vacías finales y consecutivas de una fila (excepto la primera) se pueden omitir.

Comentarios

Se provee un identificador para comentarios de línea (por defecto **%**).

También se permiten comentarios de bloque, así como comentarios entre línea, mediante los delimitadores de apertura y cierre de comentario de bloque (por defecto **(%, %)**). Asimismo, los delimitadores **cc"**, **"** permiten insertar comentarios entre línea exclusivamente; mientras que los identificadores de entorno **Comentario** y **Comentarios**, permiten insertar comentarios de bloque o multilínea exclusivamente.

A menos que los delimitadores de comentarios de bloque (`(%%, %%)`) inicien y terminen sobre la misma línea, no se permite iniciar un bloque de comentario en una línea donde el iniciador de bloque (`(%%)`) sea precedido de caracteres que no sean espacios o tabuladores, en cuyo caso se debería usar el identificador para comentarios de línea. Tampoco se permite cerrar un comentario de bloque y sobre la misma línea insertar caracteres que no sean espacios en blanco.

Por defecto el contenido de comentarios de bloque no es trasladado al archivo `.tex` como comentario. Los comentarios que inicien y terminen en la misma línea sí se mantendrán en el archivo `.tex` de salida igualmente como comentarios. Para modificar este comportamiento se proporcionan las opciones de configuración JSON `parse_block_comments`, `parse_line_comments` y `parse_inline_line_comments`.

Referencias informales

Se define un formato simple de referencia informal, el cual puede o no ser mostrado en el documento final, mediante la opción `parse_informal_references`. Este formato está pensado para anotar referencias rápidas en el archivo `TXT`, ya que una referencia `BibTeX` (para la cual también se planea definir un formato `TXT` lo más práctico posible) es más elaborada.

Una referencia rápida consiste en una línea que comienza con un par de corchetes sin contenido, seguidos de texto arbitrario correspondiente al contenido de la referencia.

Si se indica mostrar las referencias en el documento final, por defecto se muestran tal cual como en el `TXT`.

Ejemplo 13.1

```
[ ] El Universo LaTeX, Rodrigo De Castro Korgi.  
[ ] LaTeX in 24 Hours, Dilip Datta.  
[ ] https://tex.stackexchange.com/
```

Las referencias rápidas pueden ser anotadas en el `TXT` como forma de recordatorio, para luego transformarlas a un formato estándar. Como trabajo futuro, además de implementar un formato completo en `TXT` que se pueda traducir a un documento `BibTeX` de salida, también se planea generar otro formato intermedio, de una o pocas líneas, donde se definan de manera práctica los campos `BibTeX` mínimos para que puedan igualmente ser traducidos automáticamente a un archivo `BibTeX` de salida.

Configuraciones JSON

Se pueden establecer diversas opciones de configuración para la sintaxis del archivo TXT de entrada, el formato del PDF de salida, entre otras, a partir de un archivo JSON. Si no se indica ningún archivo de configuración y existe un archivo JSON con alguno de los siguientes nombres en el directorio de trabajo, este archivo será utilizado por defecto como archivo de configuración:

- `conf.json`
- `config.json`
- `configuration.json`
- `doculate.json`
- `docuLaTeX.json`

A continuación se muestra un ejemplo de un archivo de configuración con todas las configuraciones y sus valores por defecto, excepto para las configuraciones de entornos y delimitadores, donde solo se muestra un ejemplo de configuración para un entorno y un delimitador. La mayoría de estas opciones no están disponibles mediante CLI, por lo que si se quiere una configuración detallada, se tendrá que usar un archivo JSON. Los valores por defecto mostrados se utilizarán en las configuraciones que tengan valor `null` o que no se encuentren en el archivo.

Para las configuraciones donde no se muestre un valor por defecto, este se obtiene a partir de otras configuraciones o como entrada del usuario. Al lado se coloca un comentario con una breve explicación para estos casos.

Código 14.1

```
{
  // Utilizar el archivo de configuración indicado, ignorando el actual.
  "config_file": null,
  // Entrada principal, debe indicarse.
  "txt_input_file": ,
  "output_dir": "./",
  // Por defecto, mismo valor que "output_dir".
  "output_line_state_log_dir": ,

  // Por defecto, mismo valor que nombre de archivo "txt_input_file".
  "title": ,
  "author": "",
```

```

"date": "\\today",

"main_font_size": "12pt",

"page_left_margin": "25mm",
"page_right_margin": "25mm",
"page_top_margin": "40mm",
"page_bottom_margin": "30mm",

"front_page_left_margin": "50pt",
"front_page_right_margin": "50pt",
"front_page_top_margin": "1ex",
"front_page_bottom_margin": "1ex",

"title_font_size": "200",

"tableofcontents_name": "Contents",

"root_section": "chapter",

"header_mark": "#",
"library_mark": "+",
// Por defecto, mismo valor que "header_mark".
"paragraph_mark": ,
"section_marks_unit": "-",
"term_mark": "\\*",
"action_error_best_practice_mark": "\\*",
"list_mark": "-",
"description_mark": "-",
"informal_reference_mark": "\\[\\]",
"block_comment_delimiters": ["(%", "%)"],
"line_comment_symbol": "%%",

"LaTeX_special_replacements": {
  "\\": "\\textbackslash{}",
  "^": "\\textasciicircum{}",
  "_": "\\_",
  "&": "\\&",
  "%": "\\%",
  "$": "\\$",
  "#": "\\#",
  "{": "\\{",
  "}": "\\}",

```

```

    "~": "\\textasciitilde{}",
    "TeX": "\\TeX{}",
    "LaTeX": "\\LaTeX{}",
    "BibTeX": "\\textsc{Bib}\\TeX{}"
  },
  "LaTeX_code_special_replacements": {
    "\\": "\\\\",
    "%": "\\%",
    "{": "\\{",
    "}": "\\}",
    "#": "\\#"
  },
  "LaTeX_url_special_replacements": {
    "\\": "\\\\",
    "%": "\\%",
    "#": "\\#"
  },
  // Diccionario para agregar valores a "LaTeX_special_replacements".
  "add_LaTeX_special_replacements": null,
  // Diccionario para agregar valores a "LaTeX_code_special_replacements".
  "add_LaTeX_code_special_replacements": null,
  // Diccionario para agregar valores a "LaTeX_url_special_replacements"
  "add_LaTeX_url_special_replacements": null,

  "colon_rule_when_no_identifiers": true,
  "default_list_element_when_no_identifiers": true,

  "parse_informal_references": true,
  "parse_block_comments": false,
  "parse_line_comments": true,
  "parse_inline_line_comments": true,

  "header_pre_chapter_mark": "Chapter ",
  "header_pos_chapter_mark": ". ",

  "front_page_file_name": "front_page.tex",
  "preamble_file_name": "preamble.tex",

  // Por defecto texto plano, sin resaltado de sintaxis.
  "main_programming_language": ,
  // Lista de cadenas con nombres de lenguajes soportados.
  "programming_languages": [],

```

```
// Lista de cadenas con rutas/nombres de archivos.
"programming_languages_definition_files": [],
"use_file_name_if_no_main_language": false,

"code_background_color": "gray!10!white",
"code_title_color": "gray!70!black",
"code_frame_color": "gray!70!white",
"code_environments_title_colors": {
  "Sintax": "violet!30!gray",
  "CodeSintax": "violet!30!gray",
  "Example": "green!30!gray",
  "Examples": "green!30!gray!85!black",
  "AntiExample": "red!30!gray",
  "AntiExamples": "red!30!gray!85!black",
  "Code": "blue!30!gray",
  "CodeExample": "green!30!gray",
  "CodeAntiExample": "red!30!gray",
  "CommandsCode": "black!30!gray",
  "Configuration": "cyan!30!gray",
  "Class": "blue!30!gray",
  "Function": "blue!30!gray",
  "Procedure": "blue!30!gray",
  "Definition": "yellow!30!gray",
  "Definitions": "yellow!30!gray!85!black"
},
// Mismos campos que "code_environments_title_colors".
"add_code_environments_title_colors": null,

"code_environments_font_size": "small",

"environments": {
  "example": {
    "values": ["Ejemplo", "Ej."],
    "add_values": []
  }
},

"default_inline_normal_delimiter": "'",
"default_inline_no_nesting_delimiter": "\"",
"default_unambiguous_normal_delimiters": ["{", "}"],

"delimiters": {
  "bold": {
```

```
// Genera los valores mostrados en "normal" y "no_nesting", sin necesidad
// de indicarlos.
"signature_constructor": "b",
"add_signature_constructor": null,
"normal": [["b'", "'"], ["b{", "}"]],
"add_normal": null,
"no_nesting": ["b\\", "\\\""],
"add_no_nesting": null
}
}
}
```

Las configuraciones JSON aún no cuentan con verificaciones y pruebas completas para mitigar errores y efectos no deseados, por lo que se recomienda no utilizar valores muy excéntricos.

CLI

Al instalar el paquete, el usuario podrá utilizarlo a través de línea de comandos. La forma más rápida de hacer esto es ejecutar el comando `doculate`, seguido del nombre del archivo de entrada que se encuentre en el mismo directorio donde se ejecute el comando:

Comando 15.1

```
doculate archivo_entrada.txt
```

A continuación se describen los comandos y opciones disponibles al instalar el paquete.

`doculate`

Comando principal del paquete. Sin ningún argumento, buscará un archivo de configuración en el directorio donde se ejecute, con alguno de los nombres descritos en la sección *Configuraciones JSON*, el cual deberá contar con el campo `txt_input_file`. De no encontrar el archivo o la configuración `txt_input_file`, se mostrará un mensaje informándolo y no se realizará ninguna acción. Como argumento principal recibe el nombre del archivo a procesar, equivalente a la configuración `txt_input_file`. Al recibir el nombre del archivo sin ninguna configuración adicional, generará el archivo `.tex` correspondiente al archivo de entrada, además de los archivos correspondientes al preámbulo, mediante los cuales se podrá generar el PDF final, al ejecutar el archivo `preamble.tex` con el motor `LuaATEX` o `XeATEX`.

A continuación se listan las opciones exclusivas del comando `doculate`:

- `--only-parse-file` bandera para no generar los archivos del preámbulo.
- `--only-create-preamble-files` bandera para no generar el archivo `.tex` correspondiente al archivo de entrada.
- `--only-analyze-file` bandera para solo verificar la correcta escritura del archivo `.txt` y generar un archivo `.log` para consultar los estados de cada línea.
- `--parsed-file-name` establece el nombre del archivo `.tex` de salida (por defecto el mismo que el archivo de entrada, cambiando la extensión).
- `--preamble-file-name` establece el nombre del archivo de salida principal del preámbulo (por defecto `preamble.tex`).
- `--line-state-file-name, -s` establece el nombre del archivo *log* de salida con los estados de cada línea del archivo de entrada.

Las configuraciones opcionales disponibles tanto para el comando `doculate`, como para sus subcomandos son las siguientes:

- `--help, -h` muestra información de los comandos disponibles y su uso.
- `--config-file, -c` nombre del archivo de configuración a utilizar. Las configuraciones dadas en línea de comandos reemplazarán a sus equivalentes en el archivo de configuración.
- `--output-dir, -d` directorio de salida para los archivos del preámbulo y/o archivo `.tex` generado.
- `--title, -t` título que se mostrará en la portada del documento (por defecto el nombre del archivo de entrada). Títulos con espacios deben colocarse entre comillas (").
- `--author, -a` autor que se mostrará en la portada del documento. Nombres con espacios deben colocarse entre comillas (").
- `--programming-languages, -p` lenguajes de programación utilizados en el documento separados por espacios, para los cuales se aplicará resaltado de sintaxis. Actualmente se cuenta con soporte por defecto para Python, C++, bash, $\text{\LaTeX}$ y Git, si se requiere algún otro lenguaje, o redefinir los actuales, se tendrán que indicar sus archivos de definición de lenguaje y estilo mediante la opción `--programming-languages-definition-files` (ver sección *Archivos de definición de lenguajes*).
- `--programming-languages-definition-files, -f` archivos de definición de lenguaje y estilo para aplicar resaltado de sintaxis de un lenguaje particular (ver sección *Archivos de definición de lenguajes*).

A continuación se presentan los subcomandos disponibles del comando `doculate`:

`parse-file, p`

Recibe el nombre del archivo `.txt` de entrada y lo convierte en su equivalente `.tex`, sin generar archivos de preámbulo. Es equivalente a la bandera `--only-parse-file` del comando principal. Sus opciones exclusivas son:

`--output-file, -o` nombre del archivo `.tex` de salida.

`--output-log` nombre del archivo *log* que contiene la comparación de las líneas del archivo de entrada y el de salida. El archivo solo se generará si se indica esta opción.

`create-preamble, pr`

Recibe el nombre del archivo `.tex` con el contenido del documento que se ejecutará en el preámbulo (por ejemplo, el archivo de salida de `parse-file`) y con ello genera los archivos del preámbulo. Es equivalente a la bandera `--only-create-preamble-files` del comando principal. Sus opciones exclusivas son:

`--output-file, -o` nombre del archivo principal `.tex` del preámbulo.

`analyze-file, a`

Recibe el nombre del archivo `.txt` de entrada y realiza un análisis de su sintaxis, pudiéndose generar errores si hay alguna incongruencia. Genera un archivo *log* con el estado de cada línea del archivo de entrada. Es equivalente a la bandera `--only-analyze-file` del comando principal. Sus opciones exclusivas son:

`--output-file, -o` nombre del archivo *log* de salida.

Algunos ejemplos del uso de los subcomandos se muestran a continuación:

Comando 15.2

```
doculate parse-file archivo_entrada.txt
doculate pr preambulo.tex
doculate analyze-file archivo_entrada.txt -o analisis.log
```

Archivos de definición de lenguajes

Si se requiere el resaltado de sintaxis de un lenguaje específico, o redefinir el de los lenguajes preestablecidos, se pueden incluir archivos de definición donde se indique cómo realizar el resaltado para cada lenguaje. Para cada lenguaje que se vaya a definir se requieren dos archivos: uno con la definición de las palabras o términos reservados y otro de estilo.

Por el momento no se cuenta con opciones de configuración en el archivo JSON para establecer la definición de un lenguaje, por lo cual esto se tiene que hacer en un formato L^AT_EX específico en los archivos mencionados.

A continuación se muestra un ejemplo simple de cada uno de los archivos de definición para el lenguaje Python.

`python_language.tex`

Ejemplo 16.1

```
\lstdefinlanguage{Python}{
  % Core keywords
  morekeywords = {as, and, break, class, continue, def, del, elif, else,
    for, from, if, import, in, is, not, or, pass, print, return, try, while},
  % Built-ins
  morekeywords = [2]{abs, all, any, basestring, bin, bool, bytearray,
    callable, chr, classmethod, cmp, compile, complex, delattr, dict, dir},
  sensitive = true,
  morecomment = [1]\#,
  morestring = [b]',
  morestring = [b]",
  morestring = [s]{''}{''},
  morestring = [s]{""}{""},
  style = Python
}
```

`python_style.tex`

Ejemplo 16.2

```
\definecolor{purple-python-keywords}{RGB}{153, 0, 153}
\definecolor{green-python-strings}{RGB}{0, 153, 0}

\lstdefinestyle{Python}{
  keywordstyle = \color{orange!80!black}\bfseries, % Core keywords
  keywordstyle = {[2]\color{purple-python-keywords}}, % Built-ins
  stringstyle = \color{green-python-strings},
  commentstyle = \color{gray}\itshape
}
```

Los nombres de los archivos de definición se deberán indicar en la opción `--programming-languages-definition-files`, así como el nombre del lenguaje (indicado en el comando `\lstdefinlanguage` ó `\lstdefinestyle`) en la opción `--programming-languages`, como se muestra en el siguiente ejemplo:

Comando 16.1

```
doculate archivo_entrada.txt --programming-languages-definition-files  
python_language.tex python_style.tex --programming-languages Python
```

Si se requiere una definición más avanzada, se puede consultar la documentación del paquete `listings` (<https://ctan.math.washington.edu/tex-archive/macros/latex/contrib/listings/listings.pdf>).

Logs

Estado de línea

En el log de salida del subcomando `analyze-file` u opción `--only-analyze-file`, se escribe, para cada línea (además de la línea misma), las siguientes características de su estado:

1. Número de línea (comenzando desde 1).
2. Pila de anidación de entornos.
3. Identificador (texto, línea en blanco, contenido de, entorno de línea, etc.).
4. Nivel de anidación (número de tabuladores iniciales).

Comparación de líneas

El log de salida de la opción `--output-log` del subcomando `parse-file` contiene, para cada línea del TXT de entrada, dicha línea original y su correspondiente traducción a $\text{\LaTeX}$.

Problemas conocidos

- Espacios al final de términos o al final de código entre texto podrían generar errores o que se bloquee el proceso de generación del PDF.

- Las comillas dobles (") en valores de configuración del archivo JSON deben escaparse (e.g. "configuracion": "\"").
- Caracteres no alfanuméricos en valores de configuración JSON, como %, *, ?, etc., pueden generar errores o comportamientos no deseados si no se escapan mediante \\. Por ejemplo, si se genera un error, en vez de colocar "configuracion": "*", intentar con "configuracion": "*".
- Los entornos de lista o tabla implícitos en el TXT (sin identificador que indique su inicio) deben estar indentados. Por ejemplo, si se inicia un entorno de lista al mismo nivel del texto o contenido que lo precede, se generará un error o comportamiento no deseado.
- Por el momento todo el contenido de términos, acciones específicas, errores y buenas prácticas, también debe estar indentado un nivel, ya que podrían generarse efectos no deseados de no hacerlo.
- El caracter backtick (`) aún sin parsear en el TXT, esto es, pasado directo al .tex como texto plano, sin usarlo como delimitador (lo cual es por defecto), tendrá también un efecto de delimitador de código entre texto en el documento final, ya que L^AT_EX lo usa como delimitador corto por defecto.

TODO

A continuación se listan las 5 principales características por agregar en futuras versiones.

- Glosario de términos reservados.
- Listas (índices) de acciones específicas, errores y buenas prácticas que incluyan la sección que las contiene.
- Referencias cruzadas a secciones, términos reservados, acciones específicas, entornos, etc.
- Argumentos de entornos para modificar cosas como el lenguaje de programación utilizado, estilo, etc.
- Panel opcional en entornos de código para mostrar su salida.

La lista completa se puede consultar en el archivo `TODO.md`.

Licencia

Este proyecto está licenciado únicamente para uso personal y no comercial. Consultar el archivo `LICENSE` para conocer los términos completos.