pylatex

September 15, 2024

[ ]:
```
from pylatex import Command, Document, Section, Subsection
from pylatex.utils import NoEscape, italic

def fill_document(doc):
    with doc.create(Section("Una Seccion")):
        doc.append("Texto interno de la seccion")
        doc.append(italic("Texto en cursiva"))
        with doc.create(Subsection("Una subseccion")):
            doc.append("Una Subseccion con otros caracteres: %&$&%$&()")


doc = Document("Basico")
doc.preamble.append(Command("title","Cheatsheets ejemplo"))
doc.preamble.append(Command("author","Raul Mamani Cusi"))
doc.preamble.append(Command("date",NoEscape(r"\today")))
doc.append(NoEscape(r"\maketitle"))
fill_document(doc)


doc.generate_pdf("EjemploBasico.pdf")
doc.generate_tex("EjemploBasico.tex")
tex = doc.dumps()
tex
```

[ ]:
```
'\\documentclass{article}%\n\\usepackage[T1]{fontenc}%\n\\usepackage[utf8]{input
enc}%\n\\usepackage{lmodern}%\n\\usepackage{textcomp}%\n\\usepackage{lastpage}%\
n%\n\\title{Cheatsheets ejemplo}%\n\\author{Raul Mamani Cusi}%\n\\date{\\today}%
\n%\n\\begin{document}%\n\\normalsize%\n\\maketitle%\n\\section{Una
Seccion}%\n\\label{sec:UnaSeccion}%\nTexto interno de la
seccion%\n\\textit{Texto en cursiva}%\n\\subsection{Una
subseccion}%\n\\label{subsec:Unasubseccion}%\nUna Subseccion con otros
caracteres: \\%\\&\\$\\&\\%\\$\\&()\n\n%\n\\end{document}'
```

[ ]:
```
# Herencia basica

from pylatex import Command, Document, Section, Subsection
from pylatex.utils import NoEscape, italic, bold
```

```
class MiDocumento(Document):
    def __init__(self):
        super().__init__()

        self.preamble.append(Command("title","Cheatsheets ejemplo 2␣
 ↪(Heredado)"))
        self.preamble.append(Command("author","Raul Mamani Cusi"))
        self.preamble.append(Command("date",NoEscape(r"\today")))
        self.append(NoEscape(r"\maketitle"))
    def fill_document(self):
        with self.create(Section("Una Seccion")):
            self.append("Texto interno de la seccion")
            self.append(italic("Texto en cursiva"))
            self.append(bold("Texto en Negrilla"))
            with self.create(Subsection("Una subseccion")):
                self.append("Una Subseccion con otros caracteres: %&$&%$&()")

doc = MiDocumento()
doc.fill_document()

# Creacion de una seccion en el documento
with doc.create(Section("Una nueva seccion")):
    doc.append("Otro texto añadido despues")

doc.generate_pdf("Basico con herencia.pdf")
doc.generate_tex("Basico con Herencia.tex")
```

[ ]:
```
# COnfig example
import pylatex.config as cf
from pylatex import Document, NoEscape

lorem = """
Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere
cubilia Curae; Phasellus facilisis tortor vel imperdiet vestibulum. Vivamus et
mollis risus. Proin ut enim eu leo volutpat tristique. Vivamus quam enim,
efficitur quis turpis ac, condimentum tincidunt tellus. Praesent non tellus in
quam tempor dignissim. Sed feugiat ante id mauris vehicula, quis elementum nunc
molestie. Pellentesque a vulputate nisi, ut vulputate ex. Morbi erat eros,
aliquam in justo sed, placerat tempor mauris. In vitae velit eu lorem dapibus
consequat. Integer posuere ornare laoreet.

Donec pellentesque libero id tempor aliquam. Maecenas a diam at metus varius
rutrum vel in nisl. Maecenas a est lorem. Vivamus tristique nec eros ac
hendrerit. Vivamus imperdiet justo id lobortis luctus. Sed facilisis ipsum ut
tellus pellentesque tincidunt. Mauris libero lectus, maximus at mattis ut,
venenatis eget diam. Fusce in leo at erat varius laoreet. Mauris non ipsum
pretium, convallis purus vel, pulvinar leo. Aliquam lacinia lorem dapibus
```

```
tortor imperdiet, quis consequat diam mollis.

Praesent accumsan ultrices diam a eleifend. Vestibulum ante ipsum primis in
faucibus orci luctus et ultrices posuere cubilia Curae; Suspendisse accumsan
orci ut sodales ullamcorper. Integer bibendum elementum convallis. Praesent
accumsan at leo eget ullamcorper. Maecenas eget tempor enim. Quisque et nisl
eros.
"""

# Función principal que ejecuta varias configuraciones y genera archivos PDF.
def main():
    # Establece la configuración activa a 'Version1' con la opción de usar␣
 ↪indentación.
    cf.active = cf.Version1()

    # Crea un documento utilizando los datos del texto 'lorem' sin escaparlo,␣
 ↪respetando la indentación.
    doc = Document(data=NoEscape(lorem))

    # Genera un archivo PDF con el nombre "config1_with_indent" y mantiene el␣
 ↪archivo .tex sin limpiarlo.
    doc.generate_pdf("config1_with_indent", clean_tex=False)

    # Establece la configuración activa a 'Version1' pero sin indentación.
    cf.active = cf.Version1(indent=False)

    # Crea otro documento con el texto 'lorem', pero sin aplicar la indentación.
    doc = Document(data=NoEscape(lorem))

    # Genera un archivo PDF con el nombre "config2_without_indent" sin limpiar␣
 ↪el archivo .tex.
    doc.generate_pdf("config2_without_indent", clean_tex=False)

    # Utiliza un contexto de 'Version1' con indentación habilitada␣
 ↪temporalmente.
    with cf.Version1().use():
        # Crea un documento dentro del contexto temporal con indentación.
        doc = Document(data=NoEscape(lorem))

        # Genera un archivo PDF con el nombre "config3_with_indent_again" sin␣
 ↪limpiar el archivo .tex.
        doc.generate_pdf("config3_with_indent_again", clean_tex=False)

    # Crea otro documento fuera del contexto anterior sin especificar la␣
 ↪configuración de indentación.
    doc = Document(data=NoEscape(lorem))
```

```
    # Genera un archivo PDF con el nombre "config4_without_indent_again" sin␣
 ↪limpiar el archivo .tex.
    doc.generate_pdf("config4_without_indent_again", clean_tex=False)


main()
```

Environment example Wrapping existing LaTeX environments with the Environment class.

classAllTT(*, options=None, arguments=None, start_arguments=None, **kwargs)[source] A class to wrap LaTeX's alltt environment.

Parameters: options (str or list or Options) – Options to be added to the \begin command arguments (str or list or Arguments) – Arguments to be added to the \begin command start_arguments (str or list or Arguments) – Arguments to be added before the options

[ ]:
```
from pylatex import Document, Section
from pylatex.base_classes import Environment
from pylatex.package import Package
from pylatex.utils import NoEscape


class AllTT(Environment):
    """A class to wrap LaTeX's alltt environment."""

    packages = [Package("alltt")]
    escape = False
    content_separator = "\n"


# Create a new document
doc = Document()
with doc.create(Section("Wrapping Latex Environments")):
    doc.append(
        NoEscape(
            r"""
        The following is a demonstration of a custom \LaTeX{}
        command with a couple of parameters.
        """
        )
    )

    # Put some data inside the AllTT environment
    with doc.create(AllTT()):
        verbatim = (
            "This is verbatim, alltt, text.\n\n\n"
            "Setting \\underline{escape} to \\underline{False} "
```

```
            "ensures that text in the environment is not\n"
            "subject to escaping...\n\n\n"
            "Setting \\underline{content_separator} "
            "ensures that line endings are broken in\n"
            "the latex just as they are in the input text.\n"
            "alltt supports math: \\(x^2=10\\)"
        )
        doc.append(verbatim)

    doc.append("This is back to normal text...")

# Generate pdf
doc.generate_pdf("environment_ex", clean_tex=False)
```

El siguiente ejemplo demuestra varias caracteristicas de PyLatex Incluye ecuaciones, tablas ecuaciones con objetos numpy, tikz plots, y figuras

[ ]:
```
import os  # Proporciona funciones para interactuar con el sistema operativo,␣
 ↪como manejar rutas de archivos.
import numpy as np  # Biblioteca para manejar arreglos y operaciones␣
 ↪matemáticas.
from pylatex import (  # pylatex permite generar archivos LaTeX mediante Python.
    Alignat,  # Para manejar ecuaciones alineadas.
    Axis,  # Crea un eje dentro de un entorno TikZ para gráficos.
    Document,  # Representa un documento LaTeX.
    Figure,  # Inserta figuras (imágenes) en el documento.
    Math,  # Inserta expresiones matemáticas en el documento.
    Matrix,  # Representa matrices matemáticas en el documento.
    Plot,  # Permite crear gráficos dentro de los ejes de TikZ.
    Section,  # Crea una sección en el documento.
    Subsection,  # Crea una subsección dentro de una sección.
    Tabular,  # Inserta tablas en el documento.
    TikZ,  # Utiliza el entorno TikZ para gráficos y dibujos.
)
from pylatex.utils import italic  # Utilidad para poner texto en cursiva.

# Define el nombre del archivo de imagen basado en la ruta del script.
image_filename = "gatito.jpg" #os.path.join(os.path.dirname(__file__), "kitten.
 ↪jpg")

# Define las opciones de geometría para el documento, como los márgenes␣
 ↪superior y lateral.
geometry_options = {"tmargin": "1cm", "lmargin": "2cm"}  # margenes␣
 ↪personalizados

# Crea un documento LaTeX con las opciones de geometría especificadas.
doc = Document(geometry_options=geometry_options)
```

```
# Crea una sección llamada "The simple stuff" y agrega contenido.
with doc.create(Section("The simple stuff")):
    # Añade texto normal y en cursiva, así como algunos caracteres especiales␣
 ↪de LaTeX.
    doc.append("Some regular text and some")
    doc.append(italic("italic text. "))
    doc.append("\nAlso some crazy characters: $&#{}")

    # Crea una subsección llamada "Math that is incorrect" con una ecuación␣
 ↪incorrecta.
    with doc.create(Subsection("Math that is incorrect")):
        # La ecuación matemática es incorrecta: 2 * 3 = 9 (debería ser 6).
        doc.append(Math(data=["2*3", "=", 9]))

    # Crea una subsección llamada "Table of something" con una tabla␣
 ↪personalizada.
    with doc.create(Subsection("Table of something")):
        with doc.create(Tabular("rc|cl")) as table:
            table.add_hline()  # Añade una línea horizontal.
            table.add_row((1, 2, 3, 4))  # Añade una fila con valores.
            table.add_hline(1, 2)  # Añade una línea horizontal entre las dos␣
 ↪primeras columnas.
            table.add_empty_row()  # Añade una fila vacía.
            table.add_row((4, 5, 6, 7))  # Añade otra fila con valores.

# Define una matriz columna 'a' y una matriz 'M' para operaciones matemáticas.
a = np.array([[100, 10, 20]]).T  # Matriz columna (transpuesta de un vector␣
 ↪fila).
M = np.matrix([[2, 3, 4], [0, 0, 1], [0, 0, 2]])  # Matriz 3x3.

# Crea una sección llamada "The fancy stuff" que contiene ejemplos avanzados.
with doc.create(Section("The fancy stuff")):
    # Crea una subsección con ecuaciones de matrices correctas.
    with doc.create(Subsection("Correct matrix equations")):
        # Inserta las matrices y su multiplicación: M * a.
        doc.append(Math(data=[Matrix(M), Matrix(a), "=", Matrix(M * a)]))

    # Crea una subsección para el entorno Alignat (alineación de ecuaciones).
    with doc.create(Subsection("Alignat math environment")):
        with doc.create(Alignat(numbering=False, escape=False)) as agn:
            # Añade una fracción en LaTeX y la ecuación de la multiplicación de␣
 ↪matrices.
            agn.append(r"\frac{a}{b} &= 0 \\")  # Fracción a/b = 0.
            agn.extend([Matrix(M), Matrix(a), "&=", Matrix(M * a)])  # Matrices␣
 ↪alineadas.
```

```
    # Crea una subsección llamada "Beautiful graphs" para agregar gráficos␣
 ↪generados.
    with doc.create(Subsection("Beautiful graphs")):
        with doc.create(TikZ()):  # Utiliza el entorno TikZ para gráficos.
            plot_options = "height=4cm, width=6cm, grid=major"  # Opciones de␣
 ↪tamaño y cuadrícula.
            with doc.create(Axis(options=plot_options)) as plot:
                # Añade un gráfico basado en la función "-x^5 - 242".
                plot.append(Plot(name="model", func="-x^5 - 242"))

                # Define coordenadas personalizadas para otro gráfico.
                coordinates = [
                    (-4.77778, 2027.60977),
                    (-3.55556, 347.84069),
                    (-2.33333, 22.58953),
                    (-1.11111, -493.50066),
                    (0.11111, 46.66082),
                    (1.33333, -205.56286),
                    (2.55556, -341.40638),
                    (3.77778, -1169.24780),
                    (5.00000, -3269.56775),
                ]
                # Añade un gráfico basado en las coordenadas proporcionadas.
                plot.append(Plot(name="estimate", coordinates=coordinates))

    # Crea una subsección para agregar una imagen de un gatito.
    with doc.create(Subsection("Cute kitten pictures")):
        with doc.create(Figure(position="h!")) as kitten_pic:
            # Inserta una imagen del archivo 'kitten.jpg' con un ancho de 120px.
            kitten_pic.add_image(image_filename, width="180px")
            # Añade un pie de foto para la imagen.
            kitten_pic.add_caption("Look it's on its back")

# Genera el archivo PDF del documento, sin eliminar el archivo .tex generado.
doc.generate_pdf("full", clean_tex=False)
```

[ ]:
```
# Headers & Footers

from pylatex import (  # Importamos las clases necesarias de pylatex para␣
 ↪manejar el documento y estilos.
    Document,  # Representa el documento LaTeX.
    Foot,  # Para crear un pie de página.
    Head,  # Para crear una cabecera.
    LargeText,  # Texto de gran tamaño.
    LineBreak,  # Para agregar saltos de línea.
    MediumText,  # Texto de tamaño medio.
```

```
    MiniPage,  # Un entorno que permite alinear contenido dentro de una página.
    PageStyle,  # Define un estilo de página.
    simple_page_number,  # Inserta un número de página simple.
)
from pylatex.utils import bold  # Utilidad para poner texto en negritas.


# Función principal para generar un encabezado personalizado en el documento.
def generate_header():
    # Define las opciones de geometría del documento, como los márgenes.
    geometry_options = {"margin": "2.5cm"}  # Margen de 0.7 pulgadas.

    # Crea el documento LaTeX con las opciones de geometría especificadas.
    doc = Document(geometry_options=geometry_options)

    # Agrega un estilo de página personalizado para la cabecera y pie de página.
    header = PageStyle("header")  # Define un nuevo estilo de página llamado␣
 ↪"header".

    # Creación de la cabecera izquierda (Head "L").
    with header.create(Head("L")):
        header.append("Page date: ")  # Añade el texto "Page date: ".
        header.append(LineBreak())  # Inserta un salto de línea.
        header.append("R3")  # Añade el texto "R3", representando la revisión o␣
 ↪versión.

    # Creación de la cabecera central (Head "C").
    with header.create(Head("C")):
        header.append("Company")  # Añade el nombre de la compañía en el centro␣
 ↪de la cabecera.

    # Creación de la cabecera derecha (Head "R").
    with header.create(Head("R")):
        header.append(simple_page_number())  # Añade el número de página en el␣
 ↪lado derecho.

    # Creación del pie de página izquierdo (Foot "L").
    with header.create(Foot("L")):
        header.append("Left Footer")  # Añade el texto "Left Footer" en el pie␣
 ↪izquierdo.

    # Creación del pie de página central (Foot "C").
    with header.create(Foot("C")):
        header.append("Center Footer")  # Añade el texto "Center Footer" en el␣
 ↪centro del pie.
```

```
    # Creación del pie de página derecho (Foot "R").
    with header.create(Foot("R")):
        header.append("Right Footer")  # Añade el texto "Right Footer" en el␣
 ↪pie derecho.

    # Agrega el estilo de página personalizado al preámbulo del documento.
    doc.preamble.append(header)

    # Cambia el estilo del documento al estilo de página recién creado.
    doc.change_document_style("header")

    # Agrega un encabezado principal centrado en el cuerpo del documento.
    with doc.create(MiniPage(align="c")):
        doc.append(LargeText(bold("Title")))  # Añade un título grande en␣
 ↪negritas.
        doc.append(LineBreak())  # Inserta un salto de línea.
        doc.append(MediumText(bold("As at:")))  # Añade texto de tamaño medio␣
 ↪en negritas.

    # Genera el archivo PDF con el nombre "header" y mantiene el archivo .tex␣
 ↪sin limpiarlo.
    doc.generate_pdf("header", clean_tex=False)


# Llamada a la función principal para generar el PDF con el encabezado y pie de␣
 ↪página.
generate_header()
```

[ ]:
```
# Test para estructuras de listas en PyLaTeX.
# Más información sobre listas en LaTeX: http://en.wikibooks.org/wiki/LaTeX/
 ↪List_Structures
from pylatex import (
    Command,  # Permite ejecutar comandos LaTeX dentro del documento.
    Description,  # Crea una lista descriptiva, donde cada ítem tiene una␣
 ↪etiqueta personalizada.
    Document,  # Representa el documento LaTeX.
    Enumerate,  # Crea una lista numerada.
    Itemize,  # Crea una lista con viñetas.
    NoEscape,  # Permite incluir contenido que no debe ser "escapado"␣
 ↪(interpretado como texto plano).
    Section,  # Crea una sección en el documento.
)

# Punto de entrada principal del script.
if __name__ == "__main__":
    # Crea un documento LaTeX.
```

```
    doc = Document()

    # Crear una lista con viñetas ("itemize") como el siguiente ejemplo en␣
↪LaTeX:
    # \begin{itemize}
    #   \item The first item
    #   \item The second item
    #   \item The third etc \ldots
    # \end{itemize}

    # Sección para la lista "Itemize".
    with doc.create(Section('"Itemize" list')):
        # Crea un entorno de lista con viñetas.
        with doc.create(Itemize()) as itemize:
            # Añade ítems a la lista.
            itemize.add_item("the first item")  # Primer ítem.
            itemize.add_item("the second item")  # Segundo ítem.
            itemize.add_item("the third etc")  # Tercer ítem.
            # Añade puntos suspensivos con el comando \ldots en LaTeX.
            itemize.append(Command("ldots"))

    # Crear una lista numerada ("enumerate") como el siguiente ejemplo en LaTeX:
    # \begin{enumerate}[label=\alph*),start=20]
    #   \item The first item
    #   \item The second item
    #   \item The third etc \ldots
    # \end{enumerate}

    # Sección para la lista "Enumerate".
    with doc.create(Section('"Enumerate" list')):
        # Crea un entorno de lista numerada, comenzando desde el número 20 y␣
↪con el formato de letra (\alph*)).
        with doc.create(
            Enumerate(enumeration_symbol=r"\alph*)", options={"start": 20})
        ) as enum:
            # Añade ítems a la lista numerada.
            enum.add_item("the first item")  # Primer ítem.
            enum.add_item("the second item")  # Segundo ítem.
            # Tercer ítem con puntos suspensivos, utilizando NoEscape para␣
↪evitar que se interpreten caracteres especiales.
            enum.add_item(NoEscape("the third etc \\ldots"))

    # Crear una lista descriptiva ("description") como el siguiente ejemplo en␣
↪LaTeX:
    # \begin{description}
    #   \item[First] The first item
    #   \item[Second] The second item
```

```
    #   \item[Third] The third etc \ldots
    # \end{description}

    # Sección para la lista "Description".
    with doc.create(Section('"Description" list')):
        # Crea un entorno de lista descriptiva.
        with doc.create(Description()) as desc:
            # Añade ítems a la lista descriptiva con etiquetas personalizadas.
            desc.add_item("First", "The first item")  # Primer ítem con la␣
 ↪etiqueta "First".
            desc.add_item("Second", "The second item")  # Segundo ítem con la␣
 ↪etiqueta "Second".
            # Tercer ítem con puntos suspensivos, usando NoEscape para no␣
 ↪escapar los caracteres especiales de LaTeX.
            desc.add_item("Third", NoEscape("The third etc \\ldots"))

    # Genera el archivo PDF a partir del contenido del documento y conserva el␣
 ↪archivo .tex.
    doc.generate_pdf("lists", clean_tex=False)
```

[ ]:
```
from pylatex import Document, LongTable, MultiColumn  # Importa las clases␣
 ↪necesarias de pylatex.

# Función para generar un documento con una tabla larga que se extiende por␣
 ↪varias páginas.
def genenerate_longtabu():
    # Opciones de geometría del documento, estableciendo márgenes y la␣
 ↪inclusión de encabezado y pie de página.
    geometry_options = {"margin": "2.54cm", "includeheadfoot": True}

    # Crea un documento LaTeX con números de página y opciones de geometría␣
 ↪especificadas.
    doc = Document(page_numbers=True, geometry_options=geometry_options)

    # Generar una tabla larga usando LongTable, que permite dividir la tabla en␣
 ↪múltiples páginas.
    with doc.create(LongTable("l l l")) as data_table:
        # Añade una línea horizontal al inicio de la tabla.
        data_table.add_hline()
        # Añade la fila de encabezado con tres columnas.
        data_table.add_row(["header 1", "header 2", "header 3"])
        # Añade otra línea horizontal después del encabezado.
        data_table.add_hline()
        # Indica que esta es la cabecera que debe repetirse en cada página.
        data_table.end_table_header()
```

```
        # Añade una línea horizontal antes de finalizar el pie de página␣
 ↪intermedio.
        data_table.add_hline()
        # Añade una fila en el pie de página, informando que la tabla continúa␣
 ↪en la próxima página.
        data_table.add_row((MultiColumn(3, align="r", data="Continua en la␣
 ↪siguente pagina"),))
        # Añade otra línea horizontal.
        data_table.add_hline()
        # Finaliza la definición del pie de página para las páginas intermedias.
        data_table.end_table_footer()

        # Añade una línea horizontal antes del pie de página final.
        data_table.add_hline()
        # Añade una fila en el último pie de página, indicando que la tabla no␣
 ↪continúa en la siguiente página.
        data_table.add_row((MultiColumn(3, align="r", data="Fin de Tabla"),))
        # Añade otra línea horizontal.
        data_table.add_hline()
        # Finaliza el pie de página para la última página.
        data_table.end_table_last_footer()

        # Define una fila de contenido que se repetirá.

        # Añade 150 filas a la tabla, cada una con los mismos valores.
        for i in range(150):
            row = [f"Content{i}", "9", "Longer String"]
            data_table.add_row(row)

    # Genera el documento PDF a partir del contenido y conserva el archivo .tex␣
 ↪generado.
    doc.generate_pdf("longtable", clean_tex=False)

# Llamada a la función para generar el PDF con la tabla larga.
genenerate_longtabu()
```

[ ]:
```
import matplotlib  # Importa el módulo matplotlib para la creación de gráficos.

from pylatex import Document, Figure, NoEscape, Section  # Importa las clases␣
 ↪necesarias de PyLaTeX.

# Configuración de Matplotlib para no utilizar el servidor X (útil en entornos␣
 ↪como TravisCI).
matplotlib.use("Agg")  # Esto evita la necesidad de una interfaz gráfica al␣
 ↪generar gráficos.
```

```
import matplotlib.pyplot as plt  # noqa: Importa pyplot de matplotlib para␣
 ↪crear gráficos.

# Función principal para generar un documento LaTeX con una figura creada por␣
 ↪matplotlib.
def main(fname, width, *args, **kwargs):
    # Opciones de geometría para el documento (márgenes derecho e izquierdo).
    geometry_options = {"right": "2cm", "left": "2cm"}

    # Crea un documento LaTeX con el nombre especificado y las opciones de␣
 ↪geometría.
    doc = Document(fname, geometry_options=geometry_options)

    # Añade texto introductorio al documento.
    doc.append("Introduction.")

    # Crea una sección en el documento llamada "I am a section".
    with doc.create(Section("I am a section")):
        # Añade un párrafo dentro de la sección.
        doc.append("Take a look at this beautiful plot:")

        # Crea una figura que contiene un gráfico (plot) de matplotlib.
        with doc.create(Figure(position="htbp")) as plot:
            # Añade el gráfico de matplotlib al documento, especificando el␣
 ↪ancho de la imagen.
            # El ancho se define utilizando LaTeX (ej: "1\textwidth") y se␣
 ↪permite pasar más argumentos.
            plot.add_plot(width=NoEscape(width), *args, **kwargs)
            # Añade un pie de figura.
            plot.add_caption("I am a caption.")

        # Añade una nota al final de la sección indicando que el gráfico fue␣
 ↪creado con matplotlib.
        doc.append("Creacion utilizando Matplolib.")

    # Añade un texto de conclusión al final del documento.
    doc.append("Conclusion.")

    # Genera el archivo PDF del documento, conservando el archivo .tex generado.
    doc.generate_pdf(clean_tex=False)

# Punto de entrada principal del script.
if __name__ == "__main__":
    # Datos para crear un gráfico simple.
    x = [0, 1, 2, 3, 4, 5, 6]  # Valores para el eje x.
    y = [15, 2, 7, 1, 5, 6, 9]  # Valores para el eje y.
```

```
    # Genera un gráfico de línea usando matplotlib.
    plt.plot(x, y)

    # Llama a la función main para crear un documento con el gráfico.
    # El primer argumento es el nombre del archivo de salida, el segundo define el ancho de la imagen (1\textwidth).
    # Se pasa un argumento adicional "dpi=300" para especificar la resolución del gráfico.
    main("matplotlib_ex-dpi", r"1\textwidth", dpi=300)
```

[ ]:
```
from pylatex import Document, LineBreak, MiniPage, VerticalSpace  # Importa las clases necesarias de PyLaTeX.

# Función para generar etiquetas utilizando LaTeX.
def generate_labels():
    # Opciones de geometría para el documento, con márgenes de 0.5 pulgadas.
    geometry_options = {"margin": "2cm"}

    # Crea un documento LaTeX con las opciones de geometría especificadas.
    doc = Document(geometry_options=geometry_options)

    # Cambia el estilo del documento a "empty", lo que significa que no habrá encabezados ni pies de página.
    doc.change_document_style("empty")

    # Genera 10 etiquetas.
    for i in range(10):
        # Crea una "MiniPage" que representa un contenedor con un ancho del 50% del ancho de la página.
        with doc.create(MiniPage(width=r"0.5\textwidth")):
            # Añade el nombre a la MiniPage.
            doc.append("Vladimir Gorovikov")
            doc.append("\n")  # Añade un salto de línea.
            # Añade el nombre de la compañía.
            doc.append("Company Name")
            doc.append("\n")  # Añade otro salto de línea.
            # Añade la ubicación (ciudad).
            doc.append("Somewhere, City")
            doc.append("\n")  # Añade otro salto de línea.
            # Añade el país.
            doc.append("Country")

        # Si el índice es impar (i % 2 == 1), añade un espacio vertical y un salto de línea.
        if (i % 2) == 1:
```

```
            doc.append(VerticalSpace("20pt"))  # Añade un espacio vertical de␣
 ↪20 puntos.
            doc.append(LineBreak())  # Añade un salto de línea para cambiar de␣
 ↪fila.

    # Genera el archivo PDF con las etiquetas y conserva el archivo .tex␣
 ↪generado.
    doc.generate_pdf("minipage", clean_tex=False)

# Llama a la función para generar el documento PDF con las etiquetas.
generate_labels()
```

[ ]:
```
# Tablas con multiples
from pylatex import Document, MultiColumn, MultiRow, Section, Subsection,␣
 ↪Tabular  # Importa las clases necesarias de PyLaTeX.

# Crea un documento LaTeX con el nombre "multirow".
doc = Document("multirow")

# Crea una sección principal llamada "Multirow Test" para el contenido de la␣
 ↪tabla.
section = Section("Multirow Test")

# Crea subsecciones para organizar diferentes ejemplos de tablas en el␣
 ↪documento.
test1 = Subsection("MultiColumn")  # Subsección para ejemplos de MultiColumn.
test2 = Subsection("MultiRow")  # Subsección para ejemplos de MultiRow.
test3 = Subsection("MultiColumn and MultiRow")  # Subsección para ejemplos␣
 ↪combinados de MultiColumn y MultiRow.
test4 = Subsection("Vext01")  # Subsección para un ejemplo más avanzado de␣
 ↪MultiRow.

# Tabla 1: Ejemplo de uso de MultiColumn
table1 = Tabular("|c|c|c|c|")  # Define una tabla con 4 columnas, todas␣
 ↪centradas y separadas por líneas verticales.
table1.add_hline()  # Añade una línea horizontal superior.
# Añade una fila con una celda que se extiende por las 4 columnas utilizando␣
 ↪MultiColumn.
table1.add_row((MultiColumn(4, align="|c|", data="Multicolumn"),))
table1.add_hline()  # Añade una línea horizontal después de la fila.
# Añade una fila de contenido normal (sin MultiColumn).
table1.add_row((1, 2, 3, 4))
table1.add_hline()  # Añade una línea horizontal.
# Otra fila con valores normales.
table1.add_row((5, 6, 7, 8))
table1.add_hline()  # Añade una línea horizontal.
```

```
# Añade una fila donde la primera celda es un valor normal, y las siguientes 3␣
 ↪celdas están fusionadas con MultiColumn.
row_cells = ("9", MultiColumn(3, align="|c|", data="Multicolumn not on left"))
table1.add_row(row_cells)
table1.add_hline()  # Añade una línea horizontal.

# Tabla 2: Ejemplo de uso de MultiRow
table2 = Tabular("|c|c|c|")  # Define una tabla con 3 columnas centradas y␣
 ↪separadas por líneas verticales.
table2.add_hline()  # Añade una línea horizontal superior.
# Añade una fila donde la primera celda ocupa 3 filas verticalmente utilizando␣
 ↪MultiRow, y las demás celdas son normales.
table2.add_row((MultiRow(3, data="Multirow"), 1, 2))
table2.add_hline(2, 3)  # Añade una línea horizontal solo en las columnas 2 y 3.
table2.add_row(("", 3, 4))  # Fila vacía para la primera celda de la segunda␣
 ↪columna.
table2.add_hline(2, 3)  # Añade otra línea horizontal en las columnas 2 y 3.
table2.add_row(("", 5, 6))  # Fila vacía en la primera columna, con contenido␣
 ↪en las demás.
table2.add_hline()  # Añade una línea horizontal completa.
# Añade otra fila con un MultiRow que ocupa 3 filas verticalmente, dejando las␣
 ↪otras celdas vacías.
table2.add_row((MultiRow(3, data="Multirow2"), "", ""))
table2.add_empty_row()  # Añade una fila vacía.
table2.add_empty_row()  # Añade otra fila vacía.
table2.add_hline()  # Añade una línea horizontal al final.

# Tabla 3: Ejemplo combinado de MultiColumn y MultiRow
table3 = Tabular("|c|c|c|")  # Define una tabla con 3 columnas centradas.
table3.add_hline()  # Añade una línea horizontal.
# Añade una fila donde una celda utiliza MultiColumn y MultiRow a la vez, y␣
 ↪otra celda contiene "X".
table3.add_row(
    (MultiColumn(2, align="|c|", data=MultiRow(2, data="multi-col-row")), "X")
)
# Añade una fila adicional donde se mantiene el MultiColumn con celdas vacías y␣
 ↪"X" en la tercera columna.
table3.add_row((MultiColumn(2, align="|c|", data=""), "X"))
table3.add_hline()  # Añade una línea horizontal.
# Añade una fila con "X" en todas las celdas.
table3.add_row(("X", "X", "X"))
table3.add_hline()  # Añade una línea horizontal.

# Tabla 4: Ejemplo avanzado de MultiRow con varias filas fusionadas.
table4 = Tabular("|c|c|c|")  # Define una tabla con 3 columnas centradas.
table4.add_hline()  # Añade una línea horizontal.
```

```
# Crea una celda en la primera columna que ocupa 4 filas, y una en la segunda␣
 ↪columna que ocupa 2 filas.
col1_cell = MultiRow(4, data="span-4")
col2_cell = MultiRow(2, data="span-2")
# Añade una fila donde las celdas están fusionadas verticalmente en las␣
 ↪primeras dos columnas.
table4.add_row((col1_cell, col2_cell, "3a"))
table4.add_hline(start=3)  # Añade una línea horizontal solo en la tercera␣
 ↪columna.
# Añade una fila con contenido solo en la tercera columna.
table4.add_row(("", "", "3b"))
table4.add_hline(start=2)  # Añade una línea horizontal desde la segunda␣
 ↪columna.
# Añade otra fila donde la segunda columna está fusionada verticalmente y la␣
 ↪tercera columna tiene contenido.
table4.add_row(("", col2_cell, "3c"))
table4.add_hline(start=3)  # Añade una línea horizontal solo en la tercera␣
 ↪columna.
# Añade una fila final con contenido en la tercera columna.
table4.add_row(("", "", "3d"))
table4.add_hline()  # Añade una línea horizontal completa al final.

# Añade las tablas a las subsecciones correspondientes.
test1.append(table1)  # Añade la tabla1 al test1 (MultiColumn).
test2.append(table2)  # Añade la tabla2 al test2 (MultiRow).
test3.append(table3)  # Añade la tabla3 al test3 (combinación de MultiColumn y␣
 ↪MultiRow).
test4.append(table4)  # Añade la tabla4 al test4 (ejemplo avanzado).

# Añade las subsecciones a la sección principal.
section.append(test1)
section.append(test2)
section.append(test3)
section.append(test4)

# Añade la sección completa al documento.
doc.append(section)

# Genera el archivo PDF del documento y conserva el archivo .tex generado.
doc.generate_pdf(clean_tex=False)
```

[ ]:
```
import numpy as np  # Importa la biblioteca NumPy, que se utiliza para manejar␣
 ↪arreglos y matrices numéricas.

from pylatex import Document, Math, Matrix, Section, Subsection, VectorName  #␣
 ↪Importa las clases necesarias de PyLaTeX para generar un documento LaTeX.
```

```python
# Bloque principal del script, que se ejecuta cuando se corre el archivo␣
 ↪directamente.
if __name__ == "__main__":
    # Crea un arreglo NumPy de tamaño (3, 1), representando un vector columna.
    a = np.array([[100, 10, 20]]).T  # El vector columna 'a' es la transpuesta␣
 ↪de un arreglo fila.

    # Crea un documento LaTeX vacío.
    doc = Document()

    # Crea una sección principal llamada "Numpy tests".
    section = Section("Numpy tests")

    # Crea una subsección llamada "Array" para contener operaciones␣
 ↪relacionadas con arreglos.
    subsection = Subsection("Array")

    # Convierte el arreglo 'a' en una matriz LaTeX utilizando Matrix().
    vec = Matrix(a)
    # Crea un nombre de vector "a" en formato LaTeX.
    vec_name = VectorName("a")
    # Genera una ecuación LaTeX que muestra 'a' como vector columna.
    math = Math(data=[vec_name, "=", vec])

    # Añade la ecuación a la subsección.
    subsection.append(math)
    # Añade la subsección a la sección principal.
    section.append(subsection)

    # Crea una nueva subsección llamada "Matrix" para trabajar con matrices.
    subsection = Subsection("Matrix")
    # Define una matriz 3x3 utilizando NumPy.
    M = np.matrix([[2, 3, 4], [0, 0, 1], [0, 0, 2]])
    # Convierte la matriz 'M' en una representación LaTeX con delimitadores de␣
 ↪corchetes "b".
    matrix = Matrix(M, mtype="b")
    # Genera una ecuación LaTeX que muestra la matriz 'M'.
    math = Math(data=["M=", matrix])

    # Añade la ecuación a la subsección.
    subsection.append(math)
    # Añade la subsección a la sección principal.
    section.append(subsection)

    # Crea una nueva subsección llamada "Product" para mostrar el producto de␣
 ↪matrices.
```

```
    subsection = Subsection("Product")

    # Genera una ecuación LaTeX que muestra el producto entre la matriz 'M' y el vector 'a'.
    math = Math(data=["M", vec_name, "=", Matrix(M * a)])
    # Añade la ecuación a la subsección.
    subsection.append(math)

    # Añade la subsección a la sección principal.
    section.append(subsection)

    # Añade la sección principal al documento.
    doc.append(section)

    # Genera el archivo PDF del documento y conserva el archivo .tex generado.
    doc.generate_pdf("numpy_ex", clean_tex=False)
```

[ ]:
```
#### Comandos en pylatex
from pylatex import Document, Section, UnsafeCommand  # Importa las clases necesarias para crear un documento, secciones y comandos no seguros.
from pylatex.base_classes import Arguments, CommandBase, Environment  # Importa clases base para crear entornos y comandos personalizados.
from pylatex.package import Package  # Importa la clase para gestionar paquetes en LaTeX.
from pylatex.utils import NoEscape  # Importa NoEscape para evitar que ciertos comandos se interpreten como texto regular.

# Define una clase que representa un entorno personalizado en LaTeX.
class ExampleEnvironment(Environment):
    """
    Clase que representa un entorno LaTeX personalizado.

    Esta clase define un entorno LaTeX llamado 'exampleEnvironment',
    que utiliza el paquete 'mdframed' para crear un marco alrededor del contenido.
    """

    _latex_name = "exampleEnvironment"  # Nombre del entorno en LaTeX.
    packages = [Package("mdframed")]  # Paquete requerido para el entorno, en este caso 'mdframed' que permite crear marcos.

# Define una clase que representa un comando personalizado en LaTeX.
class ExampleCommand(CommandBase):
    """
    Clase que representa un comando LaTeX personalizado.
```

```
    Esta clase define un comando LaTeX llamado 'exampleCommand',
    que utiliza el paquete 'color' para colorear el texto.
    """

    _latex_name = "exampleCommand"  # Nombre del comando en LaTeX.
    packages = [Package("color")]  # Paquete requerido, en este caso 'color' para aplicar colores al texto.

# Crear un nuevo documento LaTeX.
doc = Document()

# Sección en el documento donde se muestra cómo utilizar comandos personalizados.
with doc.create(Section("Custom commands")):
    # Añade una breve explicación al documento utilizando NoEscape para que \LaTeX{} no se escape como texto.
    doc.append(
        NoEscape(
            r"""
        The following is a demonstration of a custom \LaTeX{}
        command with a couple of parameters.
        """
        )
    )

    # Definir el nuevo comando en LaTeX.
    new_comm = UnsafeCommand(
        "newcommand",  # Comando en LaTeX para definir un nuevo comando.
        "\exampleCommand",  # El nombre del nuevo comando que se creará.
        options=3,  # Indica que el comando tiene 3 argumentos.
        extra_arguments=r"\color{#1} #2 #3 \color{black}",  # Define cómo se comporta el comando: aplica color a los argumentos 2 y 3.
    )
    doc.append(new_comm)  # Añade la definición del comando al documento.

    # Uso del nuevo comando con diferentes argumentos.
    doc.append(ExampleCommand(arguments=Arguments("blue", "Hello", "World!"))) # Usa el comando para colorear el texto en azul.
    doc.append(ExampleCommand(arguments=Arguments("green", "Hello", "World!"))) # Usa el comando para colorear el texto en verde.
    doc.append(ExampleCommand(arguments=Arguments("red", "Hello", "World!"))) # Usa el comando para colorear el texto en rojo.

# Sección en el documento donde se muestra cómo utilizar entornos personalizados.
with doc.create(Section("Custom environments")):
```

```
    # Añade una breve explicación sobre el uso de entornos personalizados.
    doc.append(
        NoEscape(
            r"""
        The following is a demonstration of a custom \LaTeX{}
        environment using the mdframed package.
        """
        )
    )

    # Definir un estilo para el marco (box) utilizando mdframed.
    mdf_style_definition = UnsafeCommand(
        "mdfdefinestyle",  # Comando en LaTeX para definir un nuevo estilo.
        arguments=[
            "my_style",  # Nombre del nuevo estilo.
            ("linecolor=#1," "linewidth=#2," "leftmargin=1cm,"␣
 ↪"leftmargin=1cm"),  # Definición de los parámetros del estilo (color de␣
 ↪línea, grosor, etc.).
        ],
    )

    # Definir el nuevo entorno utilizando el estilo definido anteriormente.
    new_env = UnsafeCommand(
        "newenvironment",  # Comando en LaTeX para definir un nuevo entorno.
        "exampleEnvironment",  # Nombre del nuevo entorno.
        options=2,  # Indica que el entorno tiene 2 argumentos.
        extra_arguments=[
            mdf_style_definition.dumps() + r"\begin{mdframed}[style=my_style]",␣
 ↪ # Inicia el entorno con el estilo definido.
            r"\end{mdframed}",  # Finaliza el entorno.
        ],
    )
    doc.append(new_env)  # Añade la definición del nuevo entorno al documento.

    # Uso del nuevo entorno con los argumentos definidos.
    with doc.create(ExampleEnvironment(arguments=Arguments("blue", 3))) as␣
 ↪environment:  # Crea una instancia del entorno con el color rojo y grosor de␣
 ↪línea 3.
        environment.append("This is the actual content")  # Contenido dentro␣
 ↪del entorno.

# Genera el archivo PDF, conservando también el archivo .tex generado.
doc.generate_pdf("own_commands_ex", clean_tex=False)
#
```

[ ]:
```
%pip install quantities
```

```
Collecting quantities
  Downloading quantities-0.16.0-py3-none-any.whl.metadata (8.4 kB)
Requirement already satisfied: numpy>=1.20 in
c:\programdata\miniconda3\lib\site-packages (from quantities) (1.26.4)
Downloading quantities-0.16.0-py3-none-any.whl (102 kB)
   ---------------------------------------- 0.0/102.1 kB ? eta -:--:--
   ---- ----------------------------------- 10.2/102.1 kB ? eta -:--:--
   ---- ----------------------------------- 10.2/102.1 kB ? eta -:--:--
   ---------------------- --------------- 61.4/102.1 kB 465.5 kB/s eta 0:00:01
   ---------------------------------------- 102.1/102.1 kB 589.1 kB/s eta 0:00:00
Installing collected packages: quantities
Successfully installed quantities-0.16.0
Note: you may need to restart the kernel to use updated packages.
```

[ ]:
```python
import quantities as pq  # Importa el paquete 'quantities', que se utiliza para manejar unidades físicas y constantes.

from pylatex import Document, Math, Quantity, Section, Subsection  # Importa clases necesarias de PyLaTeX para generar documentos y manejar matemáticas.

# Bloque principal del script que se ejecuta cuando se ejecuta el archivo directamente.
if __name__ == "__main__":
    # Crea un documento LaTeX.
    doc = Document()

    # Crea una sección llamada "Quantity tests" para organizar las pruebas con cantidades.
    section = Section("Quantity tests")

    # Primera subsección: Trabajo con escalares que tienen unidades físicas.
    subsection = Subsection("Scalars with units")

    # Definición de constantes y variables físicas con unidades.
    G = pq.constants.Newtonian_constant_of_gravitation  # Constante de gravitación universal.
    moon_earth_distance = 384400 * pq.km  # Distancia entre la Tierra y la Luna en kilómetros.
    moon_mass = 7.34767309e22 * pq.kg  # Masa de la Luna en kilogramos.
    earth_mass = 5.972e24 * pq.kg  # Masa de la Tierra en kilogramos.

    # Calcula la fuerza gravitacional entre la Tierra y la Luna utilizando la fórmula de Newton.
    moon_earth_force = G * moon_mass * earth_mass / moon_earth_distance**2

    # Crea un objeto 'Quantity' con la fuerza calculada, redondeando a 4 cifras significativas y expresada en newtons.
```

```
    q1 = Quantity(
        moon_earth_force.rescale(pq.newton),  # Convierte la unidad a newtons.
        options={"round-precision": 4, "round-mode": "figures"},  # Opciones␣
↪para redondear la cantidad.
    )

    # Representa la ecuación de la fuerza gravitacional en formato LaTeX.
    math = Math(data=["F=", q1])
    subsection.append(math)  # Añade la ecuación a la subsección.
    section.append(subsection)  # Añade la subsección a la sección principal.

    # Segunda subsección: Trabajo con escalares que no tienen unidades físicas.
    subsection = Subsection("Scalars without units")

    # Ejemplo de escalar sin unidad: población mundial.
    world_population = 7400219037  # Población mundial (aproximada).

    # Crea un objeto 'Quantity' con la población, redondeando a 2 cifras␣
↪significativas.
    N = Quantity(
        world_population,
        options={"round-precision": 2, "round-mode": "figures"},  # Redondeo a␣
↪2 cifras.
        format_cb="{0:23.17e}".format,  # Formato científico para mostrar la␣
↪cantidad.
    )

    # Representa la población en formato LaTeX.
    subsection.append(Math(data=["N=", N]))
    section.append(subsection)  # Añade la subsección a la sección principal.

    # Tercera subsección: Trabajo con escalares que tienen incertidumbres␣
↪asociadas.
    subsection = Subsection("Scalars with uncertainties")

    # Definición de cantidades con incertidumbres.
    width = pq.UncertainQuantity(7.0, pq.meter, 0.4)  # Anchura de 7 metros con␣
↪una incertidumbre de 0.4 metros.
    length = pq.UncertainQuantity(6.0, pq.meter, 0.3)  # Longitud de 6 metros␣
↪con una incertidumbre de 0.3 metros.

    # Cálculo del área con incertidumbres.
    area = Quantity(
        width * length,  # Calcula el área como el producto de la anchura por␣
↪la longitud.
```

```
        options="separate-uncertainty",  # Muestra la incertidumbre de forma␣
↪separada en LaTeX.
        format_cb=lambda x: "{0:.1f}".format(float(x)),  # Formato con 1 cifra␣
↪decimal.
    )

    # Representa el área con incertidumbre en formato LaTeX.
    subsection.append(Math(data=["A=", area]))
    section.append(subsection)  # Añade la subsección a la sección principal.

    # Añade la sección completa al documento LaTeX.
    doc.append(section)

    # Genera el archivo PDF del documento y conserva el archivo .tex generado.
    doc.generate_pdf("quantities_ex", clean_tex=False)
```

[ ]:
```
import os  # Importa el módulo os para manejar rutas de archivos de manera␣
↪independiente del sistema operativo.

from pylatex import Document, Figure, NoEscape, Section, SubFigure  # Importa␣
↪las clases necesarias de PyLaTeX para generar el documento y manejar figuras␣
↪y subfiguras.

# Bloque principal del script, que se ejecuta cuando se ejecuta el archivo.
if __name__ == "__main__":
    # Crea un nuevo documento LaTeX con el nombre de archivo por defecto␣
↪"subfigures".
    doc = Document(default_filepath="subfigures")

    # Define la ruta de la imagen "kitten.jpg" ubicada en el mismo directorio␣
↪que el archivo Python.
    image_filename = "gatito.jpg" # os.path.join(os.path.dirname(__file__),␣
↪"kitten.jpg")

    # Crea una sección en el documento titulada "Showing subfigures".
    with doc.create(Section("Mostrando varias figuras")):
        # Crea un entorno de figura con la opción "h!" para controlar la␣
↪posición de la figura en el documento.
        with doc.create(Figure(position="h!")) as kittens:
            # Crea una subfigura dentro de la figura principal, ubicada en la␣
↪parte inferior ("b") y con un ancho del 45% de la línea.
            with doc.create(
                SubFigure(position="b", width=NoEscape(r"0.45\linewidth"))
            ) as left_kitten:
                # Añade la imagen del gatito a la subfigura y ajusta su ancho␣
↪al de la línea.
```

```
                left_kitten.add_image(image_filename,␣
↪width=NoEscape(r"\linewidth"))
                # Añade un pie de imagen a la subfigura.
                left_kitten.add_caption("Kitten on the left")

            # Crea otra subfigura, también con el 45% del ancho de la línea.
            with doc.create(
                SubFigure(position="b", width=NoEscape(r"0.45\linewidth"))
            ) as right_kitten:
                # Añade la misma imagen del gatito a la segunda subfigura,␣
↪ajustando su tamaño al ancho disponible.
                right_kitten.add_image(image_filename,␣
↪width=NoEscape(r"\linewidth"))
                # Añade un pie de imagen a la segunda subfigura.
                right_kitten.add_caption("Kitten on the right")

            # Añade un pie de figura a la figura principal que contiene ambas␣
↪subfiguras.
            kittens.add_caption("Two kittens")

    # Genera el archivo PDF y conserva el archivo .tex para posibles␣
↪modificaciones.
    doc.generate_pdf(clean_tex=False)
```

[ ]:
```
from pylatex import (  # Importa las clases necesarias de PyLaTeX para trabajar␣
↪con documentos y layout avanzado.
    Document,
    HorizontalSpace,  # Clase para crear espacios horizontales personalizados.
    HugeText,  # Clase para crear texto de gran tamaño.
    MediumText,  # Clase para crear texto de tamaño mediano.
    MiniPage,  # Clase para crear bloques de contenido dentro de una página.
    SmallText,  # Clase para crear texto pequeño.
    TextBlock,  # Clase para crear bloques de texto que pueden ubicarse en␣
↪posiciones específicas.
    VerticalSpace,  # Clase para crear espacios verticales personalizados.
)
from pylatex.utils import bold  # Función que permite poner el texto en␣
↪negritas.

# Define las opciones de geometría del documento, estableciendo márgenes de 0.5␣
↪pulgadas.
geometry_options = {"margin": "0.5in"}

# Crea un documento LaTeX sin sangría (indent=False) y con las opciones de␣
↪geometría especificadas.
doc = Document(indent=False, geometry_options=geometry_options)
```

```
# Cambia la longitud de las unidades horizontales y verticales para trabajar␣
 ↪con módulos de 1 mm.
doc.change_length("\TPHorizModule", "1mm")  # Ajusta los módulos horizontales a␣
 ↪1mm.
doc.change_length("\TPVertModule", "1mm")  # Ajusta los módulos verticales a␣
 ↪1mm.

# Crea un entorno MiniPage que ocupa el ancho total de la página.
with doc.create(MiniPage(width=r"\textwidth")) as page:

    # Crea un bloque de texto que ocupa un área de 100mm y se ubica en la␣
 ↪posición (0, 0).
    with page.create(TextBlock(100, 0, 0)):
        page.append("**** Ten Thousand Dollars")  # Añade el texto "**** Ten␣
 ↪Thousand Dollars" al bloque.

    # Crea un segundo bloque de texto que ocupa 100mm y se ubica en la posición␣
 ↪(0, 30).
    with page.create(TextBlock(100, 0, 30)):
        # Añade el nombre de la compañía y la dirección al bloque de texto.
        page.append("COMPANY NAME")
        page.append("\nSTREET, ADDRESS")  # Añade un salto de línea.
        page.append("\nCITY, POSTAL CODE")  # Añade otro salto de línea.

    # Crea un bloque de texto que ocupa 100mm y se ubica en la posición (150,␣
 ↪40).
    with page.create(TextBlock(100, 150, 40)):
        page.append(HugeText(bold("VOID")))  # Añade el texto "VOID" en grande␣
 ↪y en negritas.

    # Crea un bloque de texto que ocupa 80mm y se ubica en la posición (150, 0).
    with page.create(TextBlock(80, 150, 0)):
        # Añade la fecha en texto mediano y negritas, seguida de un espacio␣
 ↪horizontal y texto pequeño.
        page.append("DATE")
        page.append(MediumText(bold("2016 06 07\n")))  # Añade la fecha en␣
 ↪formato grande.
        page.append(HorizontalSpace("10mm"))  # Añade un espacio horizontal de␣
 ↪10mm.
        page.append(SmallText("Y/A M/M D/J"))  # Añade la leyenda sobre el␣
 ↪formato de fecha en texto pequeño.

    # Crea un bloque de texto que ocupa 70mm y se ubica en la posición (150,␣
 ↪30).
    with page.create(TextBlock(70, 150, 30)):
```

```
        page.append(MediumText(bold("$***** 10,000.00")))  # Añade la cantidad en negritas y texto mediano.

    # Añade un espacio vertical de 100mm al final del contenido.
    page.append(VerticalSpace("100mm"))

# Genera el archivo PDF del documento y conserva el archivo .tex generado.
doc.generate_pdf("textblock", clean_tex=False)
```

[ ]:
```
from pylatex import (  # Importa las clases necesarias de PyLaTeX para generar dibujos con TikZ.
    Document,
    TikZ,  # Clase para crear un entorno TikZ en el documento.
    TikZCoordinate,  # Clase para definir coordenadas en el entorno TikZ.
    TikZDraw,  # Clase para dibujar líneas y formas.
    TikZNode,  # Clase para crear nodos (elementos etiquetados) en el entorno TikZ.
    TikZOptions,  # Clase para especificar opciones de estilo en TikZ.
    TikZUserPath,  # Clase para dibujar caminos personalizados.
)

# Bloque principal que se ejecuta cuando se ejecuta el archivo directamente.
if __name__ == "__main__":
    # Crear un documento LaTeX.
    doc = Document()

    # Crear un entorno TikZ en el documento para incluir gráficos.
    with doc.create(TikZ()) as pic:
        # Definir opciones para el nodo TikZ (caja de texto).
        node_kwargs = {
            "align": "center",  # Alinear el texto en el centro.
            "minimum size": "100pt",  # Establecer un tamaño mínimo de 100pt.
            "fill": "black!20",  # Relleno de la caja con un gris al 20%.
        }

        # Crear un nodo (rectángulo con texto) en la coordenada (0,0).
        box = TikZNode(
            text="My block",  # El texto dentro del nodo.
            handle="box",  # Identificador del nodo para referencia posterior.
            at=TikZCoordinate(0, 0),  # La posición del nodo en las coordenadas TikZ.
            options=TikZOptions("draw", "rounded corners", **node_kwargs),  # Opciones de estilo: borde redondeado y dibujado.
        )

        # Añadir el nodo al entorno TikZ.
        pic.append(box)
```

```
        # Dibujar un rectángulo en las coordenadas especificadas.
        pic.append(
            TikZDraw(
                [TikZCoordinate(0, -6), "rectangle", TikZCoordinate(2, -8)],  #␣
↪Especifica el dibujo de un rectángulo.
                options=TikZOptions(fill="red"),  # Rellena el rectángulo de␣
↪color rojo.
            )
        )

        # Dibujar una línea desde el borde izquierdo (west) del nodo hacia una␣
↪coordenada relativa.
        pic.append(TikZDraw([box.west, "--", "++(-1,0)"]))  # La coordenada es␣
↪1 unidad a la izquierda del borde del nodo.

        # Ejemplo del uso de 'with' para crear un camino personalizado␣
↪(TikZDraw).
        with pic.create(TikZDraw()) as path:
            # Empezar desde el borde derecho (east) del nodo.
            path.append(box.east)

            # Definir las opciones de la curva de camino, incluyendo los␣
↪ángulos de entrada (in) y salida (out).
            path_options = {"in": 90, "out": 0}  # La curva entra a 90 grados y␣
↪sale a 0 grados.

            # Crear un camino curvo con una flecha al final.
            path.append(TikZUserPath("edge", TikZOptions("-latex",␣
↪**path_options)))

            # Añadir una coordenada relativa (1, 0) para extender el camino.
            path.append(TikZCoordinate(1, 0, relative=True))  # Coordenada␣
↪relativa 1 unidad a la derecha.

    # Generar el archivo PDF con el nombre 'tikzdraw' y conservar el archivo .
↪tex.
    doc.generate_pdf("tikzdraw", clean_tex=False)
```