
Documento Equipo 5

Versión 1

Cristina, Pedro, Noah

09 de enero de 2025

1.1 Entorno Virtual

El entorno virtual permite aislar las dependencias de un proyecto en Python, evitando conflictos entre diferentes proyectos que puedan requerir distintas versiones de bibliotecas.

1.1.1 1. Cómo crear el entorno virtual en el proyecto

Para crear un entorno virtual, primero debemos abrir una terminal y navegar al directorio del proyecto. Luego, ejecutamos el siguiente comando:

```
python -m venv nombre_entorno
```

Reemplazamos `nombre_entorno` por el nombre que deseamos dar al entorno virtual. Esto creará una carpeta en el directorio del proyecto que contendrá el entorno virtual.

1.1.2 2. Cómo activar el entorno virtual

Para activar el entorno virtual, utilizamos los siguientes comandos dependiendo del sistema operativo y ubicándonos en el directorio raíz del proyecto:

- **Windows:**

```
.\nombre_entorno\Scripts\activate
```

- **Linux/Mac:**

```
source nombre_entorno/bin/activate
```

Una vez activado, deberías ver el nombre del entorno en el prompt de la terminal, indicando que el entorno virtual está activo.

1.2 Instalación de dependencias del proyecto

Para ejecutar correctamente este proyecto de Python, es necesario instalar las dependencias especificadas en el archivo requirements.txt. A continuación, se detallan los pasos necesarios para realizar esta instalación de manera efectiva.

1.2.1 1. Instalar las dependencias con el archivo requirements.txt

Con el entorno virtual activado, instala las dependencias ejecutando:

```
pip install -r requirements.txt`
```

Este comando utiliza pip para leer el archivo requirements.txt e instala todas las librerías listadas en él.

1.2.2 2. Verificar la instalación

Para asegurarte de que las dependencias se instalaron correctamente, puedes listar los paquetes instalados con:

```
pip list
```

Esto mostrará una lista de todas las librerías instaladas en tu entorno virtual.

1.2.3 3. Desactivar el entorno virtual (opcional)

Cuando hayas terminado de trabajar en el proyecto, puedes desactivar si quieres el entorno virtual con:

```
deactivate
```

Archivo requirements.txt

Nuestro archivo requirements.txt se vería tal que así:

```
alabaster==1.0.0
babel==2.16.0
certifi==2024.8.30
charset-normalizer==3.4.0
colorama==0.4.6
docutils==0.21.2
idna==3.10
imagesize==1.4.1
Jinja2==3.1.4
markdown-it-py==3.0.0
MarkupSafe==3.0.2
mdit-py-plugins==0.4.2
mdurl==0.1.2
myst-parser==4.0.0
packaging==24.2
pandoc==2.4
plumbum==1.9.0
ply==3.11
psycopg==3.2.1
psycopg-binary==3.2.1
psycopg-pool==3.2.2
Pygments==2.18.0
```

(continúe en la próxima página)

(proviene de la página anterior)

```
PySide6==6.7.2
PySide6_Addons==6.7.2
PySide6_Essentials==6.7.2
pywin32==308
PyYAML==6.0.2
requests==2.32.3
setuptools==72.1.0
shiboken6==6.7.2
snowballstemmer==2.2.0
Sphinx==8.1.3
sphinx-rtd-theme==3.0.2
sphinxcontrib-applehelp==2.0.0
sphinxcontrib-devhelp==2.0.0
sphinxcontrib-htmlhelp==2.1.0
sphinxcontrib-jquery==4.1
sphinxcontrib-jsmath==1.0.1
sphinxcontrib-qthelp==2.0.0
sphinxcontrib-serializinghtml==2.0.0
typing_extensions==4.12.2
tzdata==2024.1
urllib3==2.2.3
wheel==0.44.0
```

Detalles sobre cada una de las dependencias

1. alabaster==1.0.0

■ Descripción:

- Alabaster es un tema de apariencia para **Sphinx**.

■ Propósito:

- Proporciona un diseño limpio y sencillo para la documentación generada con Sphinx. Es el tema por defecto de Sphinx.

2. babel==2.16.0

■ Descripción:

- Babel es una librería para internacionalización y localización (**i18n**) de aplicaciones.

■ Propósito:

- Facilita la traducción de textos, así como el formateo de fechas, números y otros datos específicos de cada región.

3. certifi==2024.8.30

■ Descripción:

- Certifi proporciona una lista de certificados de autoridades de certificación (**CA**) actualizados.

■ Propósito:

- Garantiza solicitudes HTTPS seguras verificando la validez de los certificados SSL/TLS.

4. charset-normalizer==3.4.0

- **Descripción:**

- Librería para detectar y manejar codificaciones de texto.

- **Propósito:**

- Facilita decodificar correctamente textos con diversas codificaciones, especialmente en respuestas HTTP.
-

5. **colorama==0.4.6**

- **Descripción:**

- Proporciona soporte para imprimir texto con color en la consola de **Windows**.

- **Propósito:**

- Permite aplicaciones de consola con salida en color en sistemas Windows.
-

6. **docutils==0.21.2**

- **Descripción:**

- Librería para procesar documentos en **reStructuredText**.

- **Propósito:**

- Fundamental para Sphinx en la generación de documentación en diferentes formatos.
-

7. **idna==3.10**

- **Descripción:**

- Implementa el estándar **IDNA** para nombres de dominio internacionalizados.

- **Propósito:**

- Permite trabajar con nombres de dominio que contienen caracteres no ASCII.
-

8. **image==1.4.1**

- **Descripción:**

- Librería para obtener dimensiones de imágenes sin cargarlas completamente.

- **Propósito:**

- Utilizada por Sphinx para determinar tamaños de imágenes en la documentación.
-

9. **Jinja2==3.1.4**

- **Descripción:**

- Motor de plantillas para Python inspirado en **Django**.

- **Propósito:**

- Utilizado por Sphinx para generar HTML dinámico a partir de plantillas.
-

10. **markdown-it-py==3.0.0**

- **Descripción:**

- Analizador de **Markdown** escrito en Python.
-

- **Propósito:**

- Facilita convertir texto Markdown a HTML en aplicaciones de documentación.
-

11. MarkupSafe==3.0.2

- **Descripción:**

- Librería para escapar cadenas y evitar inyecciones de código.

- **Propósito:**

- Utilizada por **Jinja2** para garantizar seguridad en plantillas.
-

12. mdit-py-plugins==0.4.2

- **Descripción:**

- Plugins adicionales para **markdown-it-py**.

- **Propósito:**

- Añade funcionalidades como tablas y matemáticas al procesador Markdown.
-

13. mdurl==0.1.2

- **Descripción:**

- Librería para manejar URLs en documentos Markdown.

- **Propósito:**

- Facilita el parseo y manipulación de enlaces en texto Markdown.
-

14. myst-parser==4.0.0

- **Descripción:**

- Parser de **Markdown** para Sphinx.

- **Propósito:**

- Permite escribir documentación en Markdown para proyectos Sphinx.
-

15. packaging==24.2

- **Descripción:**

- Herramienta para manejar metadatos de paquetes.

- **Propósito:**

- Facilita trabajar con versiones y dependencias de paquetes Python.
-

16. pandoc==2.4

- **Descripción:**

- Herramienta universal de conversión de documentos.

- **Propósito:**

- Convierte documentos entre formatos como Markdown, HTML y LaTeX.
-

17. plumbum==1.9.0**■ Descripción:**

- Librería para ejecutar comandos del sistema desde Python.

■ Propósito:

- Facilita la automatización y ejecución de comandos en scripts Python.
-

18. ply==3.11**■ Descripción:**

- Implementación de **Lex** y **Yacc** en Python.

■ Propósito:

- Se usa para crear analizadores léxicos y sintácticos.
-

19. psycopg==3.2.1**■ Descripción:**

- Cliente para bases de datos **PostgreSQL**.

■ Propósito:

- Facilita ejecutar consultas SQL en bases de datos PostgreSQL.
-

20. psycopg-binary==3.2.1**■ Descripción:**

- Versión binaria de **psycopg**.

■ Propósito:

- Permite instalar psycopg sin necesidad de compilación.
-

21. psycopg-pool==3.2.2**■ Descripción:**

- Librería para **pooling** de conexiones PostgreSQL.

■ Propósito:

- Mejora el rendimiento al reutilizar conexiones de base de datos.
-

22. Pygments==2.18.0**■ Descripción:**

- Librería para resaltar sintaxis de código.

■ Propósito:

- Usada por Sphinx para colorear bloques de código.
-

23. PySide6==6.7.2

- **Descripción:**

- Bindings de **Qt6** para Python.

- **Propósito:**

- Permite desarrollar interfaces gráficas con Qt6 en Python.
-

24. **pywin32==308**

- **Descripción:**

- Extensiones para trabajar con la API de Windows.

- **Propósito:**

- Facilita automatización y manipulación de características del sistema Windows.
-

25. **PyYAML==6.0.2**

- **Descripción:**

- Librería para leer y escribir archivos **YAML**.

- **Propósito:**

- Facilita el manejo de configuraciones en **YAML**.
-

26. **requests==2.32.3**

- **Descripción:**

- Librería para hacer solicitudes **HTTP**.

- **Propósito:**

- Simplifica consumir APIs y descargar recursos web.
-

27. **setuptools==72.1.0**

- **Descripción:**

- Herramienta para empaquetar proyectos Python.

- **Propósito:**

- Facilita la distribución e instalación de paquetes.
-

28. **shiboken6==6.7.2**

- **Descripción:**

- Generador de bindings para **PySide6**.

- **Propósito:**

- Crea enlaces entre C++ y Python.
-

29. **snowballstemmer==2.2.0**

- **Descripción:**

- Librería para **stemming** (reducción de palabras).
-

- **Propósito:**

- Mejora la búsqueda en documentación.
-

30. **Sphinx==8.1.3**

- **Descripción:**

- Generador de documentación automática.

- **Propósito:**

- Crea documentación en HTML, PDF, etc.
-

31. **sphinx-rtd-theme==3.0.2**

- **Descripción:**

- Tema de Sphinx diseñado para **Read the Docs**.

- **Propósito:**

- Proporciona una apariencia moderna y fácil de navegar para documentación alojada en **Read the Docs**.
-

32. **sphinxcontrib-applehelp==2.0.0**

- **Descripción:**

- Extensión de Sphinx para generar documentación en formato **Apple Help**.

- **Propósito:**

- Permite crear archivos de ayuda compatibles con aplicaciones de **macOS**.
-

33. **sphinxcontrib-devhelp==2.0.0**

- **Descripción:**

- Extensión de Sphinx para generar documentación en formato **Devhelp**.

- **Propósito:**

- Facilita crear archivos de ayuda compatibles con el visor **Devhelp** de GNOME.
-

34. **sphinxcontrib-htmlhelp==2.1.0**

- **Descripción:**

- Extensión de Sphinx para generar documentación en formato **HTML Help**.

- **Propósito:**

- Permite crear archivos .chm (HTML compilado) utilizados en sistemas **Windows**.
-

35. **sphinxcontrib-jquery==4.1**

- **Descripción:**

- Extensión que incluye **jQuery** en la documentación generada por Sphinx.

- **Propósito:**

- Asegura la compatibilidad con scripts basados en jQuery en proyectos de documentación.
-

36. sphinxcontrib-jsmath==1.0.1**■ Descripción:**

- Extensión de Sphinx para renderizar fórmulas matemáticas usando **jsMath**.

■ Propósito:

- Permite mostrar ecuaciones matemáticas en la documentación HTML sin necesidad de otras herramientas.
-

37. sphinxcontrib-qthelp==2.0.0**■ Descripción:**

- Extensión de Sphinx para generar documentación en formato **Qt Help**.

■ Propósito:

- Facilita crear archivos de ayuda compatibles con el visor **Qt Assistant**.
-

38. sphinxcontrib-serializinghtml==2.0.0**■ Descripción:**

- Extensión de Sphinx para generar documentación en formato **HTML serializado**.

■ Propósito:

- Permite exportar la documentación a un formato adecuado para aplicaciones que requieren HTML serializado.
-

39. typing_extensions==4.12.2**■ Descripción:**

- Proporciona funcionalidades de anotaciones de tipo avanzadas para Python.

■ Propósito:

- Permite usar nuevas características de tipado en versiones antiguas de Python.
-

40. tzdata==2024.1**■ Descripción:**

- Base de datos de zonas horarias globales.

■ Propósito:

- Permite manejar fechas y horas con información actualizada de zonas horarias.
-

41. urllib3==2.2.3**■ Descripción:**

- Librería para manejar conexiones HTTP con funciones avanzadas.

■ Propósito:

- Facilita realizar solicitudes HTTP seguras y eficientes. Es una dependencia clave de **requests**.
-

42. wheel==0.44.0

- **Descripción:**

- Herramienta para construir y empaquetar distribuciones en formato **Wheel** (.whl).

- **Propósito:**

- Mejora el proceso de instalación de paquetes Python al ofrecer un formato más rápido y eficiente.

1.3 Creación Inicial del Proyecto con Sphinx

A continuación, vamos a detallar todo el proceso para la creación inicial de un proyecto con Sphinx, explicando los comandos, las opciones necesarias y cómo configurarlo para usar **Markdown** con **MyST-Parser**.

1.3.1 1. Instalación de Sphinx y MyST-Parser

Con el entorno virtual activado, procedemos a instalar Sphinx, antes de empezar, hay que asegurarse de tener Sphinx y MyST-Parser instalados en el entorno. Se pueden instalar con pip:

```
pip install sphinx myst-parser
```

- **Sphinx:** Es la herramienta principal para generar documentación.
- **MyST-Parser:** Permite usar archivos Markdown (.md) en lugar de reStructuredText (.rst).

1.3.2 2. Crear un Proyecto con Sphinx

sphinx-quickstart es la herramienta que utilizamos para inicializar un proyecto de documentación con Sphinx. Esta aplicación interactiva genera la estructura base del proyecto, incluyendo los directorios necesarios, el archivo de configuración (conf.py) y un índice inicial (index.rst o index.md). Se ejecuta el comando para iniciar un nuevo proyecto de Sphinx:

```
sphinx-quickstart
```

- Este comando genera una estructura básica dentro de la carpeta docs/.

Opciones Importantes en el Proceso:

Al ejecutar sphinx-quickstart, se deben responder algunas preguntas:

1. **Directorio donde se guardará la documentación:**

- Por defecto: docs/.

2. **Separar archivos fuente y archivos compilados (build):**

- Se responde yes.
- Esto crea una estructura con dos carpetas: source/ (archivos fuente) y build/ (archivos generados).

3. **Nombre del proyecto:** Introduce el nombre del proyecto.

4. **Nombre del autor:** Se escribe el autor del proyecto.

5. **Idioma:** Para trabajar en español se escribe es.

1.3.3 3. Estructura del Proyecto

Tras ejecutar sphinx-quickstart, se creara una estructura básica como esta:

```
docs/
├── build/           # Archivos generados por Sphinx (HTML, PDF, etc.)
├── source/          # Archivos fuente de la documentación
│   ├── _static/     # Archivos estáticos (CSS, imágenes, etc.)
│   ├── _templates/  # Plantillas HTML personalizadas
│   ├── conf.py       # Archivo de configuración principal
│   ├── index.md      # Archivo raíz del proyecto (puedes reemplazarlo por Markdown)
├── Makefile         # Comandos para compilar la documentación (Linux/macOS)
└── make.bat         # Comandos para compilar la documentación (Windows)
```

1.3.4 4. Configuración para Usar Markdown con MyST-Parser

Para habilitar **Markdown** en lugar de reStructuredText, se realizan los siguientes cambios:

4.1. Instalación de MyST-Parser

Hay que asegurarse de que myst-parser está instalado:

```
pip install myst-parser
```

4.2. Modificación del Archivo conf.py

Se edita el archivo docs/source/conf.py y se añaden estas configuraciones:

```
# Se habilitarán las extensiones necesarias
extensions = [
    'myst_parser', # Habilita soporte para archivos Markdown
]
```

4.2.1 Especificación de las extensiones de archivo fuente

```
source_suffix = {
    '.rst': 'restructuredtext', # Opcional, si se usan archivos .rst
    '.md': 'markdown', # Soporte para archivos Markdown
}
```

Con esta configuración, Sphinx será capaz de procesar tanto archivos .rst como .md, facilitando la integración de diferentes formatos de contenido.

4.2.2 Configuración del idioma

```
language = 'es' # Idioma seleccionado
```

4.2.3 Configuración para archivos estáticos (CSS e imágenes)

```
html_static_path = ['_static']`
```

1.3.5 5. Creación de la Documentación en Markdown

Se sustituye el archivo `index.rst` por un archivo `index.md`.

1.3.6 6. Compilación de la Documentación

Para generar los archivos HTML, utiliza el comando:

```
make html
```

- Esto crea la salida HTML en `docs/build/html`.
- Se abre `index.html` en el navegador para ver el resultado con el siguiente comando:

```
start .\build\html\index.html
```

1.3.7 7. Personalización Adicional

7.1. Estilo del Código

Se puede personalizar el estilo de resaltado de sintaxis con CSS en la carpeta `_static/`:

1. Se crea un archivo CSS (`custom.css`) en `docs/source/_static/`.
2. Se añade a `conf.py`:

```
html_css_files = ['custom.css']
```

Ejemplo básico de CSS para el resaltado de código:

```
pre {
    background-color: #2d2d2d;
    color: #f8f8f2;
    padding: 10px;
    border-radius: 5px;
    overflow-x: auto;
    font-family: "Courier New", Courier, monospace;
}
```

1.4 Estructura inicial del proyecto

Cuando creas un proyecto Sphinx utilizando `sphinx-quickstart`, se generan varias carpetas y archivos para estructurar tu documentación. Si se trabajan con **MyST-Parser** para usar Markdown (`.md`) en lugar de ReStructuredText (`.rst`), estas carpetas seguirán siendo útiles, pero la estructura se adaptará ligeramente. A continuación se muestra una descripción detallada del propósito de cada carpeta y archivo:

1.4.1 1. Estructura Generada

```
docs/
├── build/
├── source/
│   ├── _static/
│   ├── _templates/
│   ├── conf.py
│   ├── index.rst
│   ├── make.bat
│   └── Makefile
```

1.1. Carpeta `build/`

- Carpeta donde se almacenan los archivos generados por Sphinx en los distintos formatos de salida, como HTML, PDF o LaTeX, después de ejecutar el proceso de construcción (`make html`, `make pdf`, etc.).
- **Subcarpetas:**
 - `html/`: Archivos HTML generados.
 - `doctrees/`: Archivos intermedios que Sphinx usa para rastrear dependencias entre documentos.
 - `latex/`: Archivos LaTeX generados (si se habilita la salida PDF).

i Nota

Esta carpeta se regenera cada vez que se ejecuta el comando `make html`.

1.2. Carpeta `source/`

Carpeta principal que contiene los archivos fuente de la documentación. Aquí se encuentran los archivos que definimos y editamos, como los documentos en formato reStructuredText (`.rst`) y las configuraciones adicionales.

1.2.1. Subcarpetas Importantes

- `_static/`:
 - Subcarpeta dentro de `source` reservada para incluir recursos estáticos, como imágenes, archivos CSS personalizados, scripts JavaScript, entre otros.
 - Aquí irán el archivo `custom.css` y cualquier imagen que usemos en la documentación.
 - Ejemplo:

```
docs/source/_static/
├── images/
│   └── imagen1.png
└── custom.css
```

- `_templates/`:
 - Carpeta utilizada para personalizar las plantillas HTML de la documentación. Es útil si queremos modificar el diseño o la estructura de las páginas generadas.

- Se pueden modificar las plantillas HTML que Sphinx usa (como el diseño de encabezados y pies de página).

1.2.2. Archivos Importantes en `source/`

■ `conf.py`:

- El archivo de configuración principal de Sphinx.
- Contiene ajustes para extensiones (como `myst-parser`), rutas de búsqueda, tema visual (`html_theme`) etc.
- Al usar MyST-Parser, hay que asegurarse de incluir:

```
extensions = ['myst_parser']
source_suffix = {
    '.md': 'markdown',
}
```

■ `index.md`:

- El archivo raíz de la documentación.
- Define el índice inicial y vincula los archivos fuente usando la directiva `toctree`.
- Al usar Markdown con MyST, el archivo se ve como:

```
# Título Principal de la Documentación

{toctree}
:maxdepth: 2
:caption: Índice:

archivo1
carpeta/archivo2
```

- `#`: Indica un **título de nivel 1** en Markdown. En este caso, es el título principal de la documentación.
- `##`: Indica un **título de nivel 2**. Aquí es un subtítulo para la sección del índice.

• **Directiva** `{toctree}`

La directiva `{toctree}` es una característica especial de **MyST-Parser** y **Sphinx** que permite construir un índice o tabla de contenidos (ToC).

• **Propósito de** `{toctree}`:

- Crea un árbol de navegación que incluye enlaces a otros documentos o secciones.
- Sphinx utiliza este árbol para generar automáticamente la estructura de navegación en los archivos HTML generados.

◦ **Parámetros de** `{toctree}`:

1. `:maxdepth: 6`

- ◊ Especifica el nivel de profundidad para los encabezados que se incluirán en la tabla de contenidos.

- ◇ 6 es el máximo, lo que significa que Sphinx incluirá encabezados desde # (nivel 1) hasta ##### (nivel 6) de los documentos referenciados.

Ejemplo:

- ◇ Si un archivo tiene:

```
# Título de Nivel 1
## Título de Nivel 2
### Título de Nivel 3`
```

Con :maxdepth: 2, solo se incluirían los niveles 1 y 2 en el índice.

2. :glob:

- ◇ Habilita el uso de **comodines** para incluir varios archivos de forma automática.
- ◇ Por ejemplo, configuracion_inicial/* seleccionará todos los archivos dentro de la carpeta configuracion_inicial/.

```
configuracion_inicial/*
```

Este patrón selecciona **todos los archivos** en la carpeta configuracion_inicial/. Sphinx incluirá cada archivo de forma automática en el índice generado.

Ejemplo:

- ◇ Si configuracion_inicial/ contiene los siguientes archivos:

```
configuracion_inicial/001.env.md
configuracion_inicial/002.instalacion_librerias.md
configuracion_inicial/003.Creacion_proyecto_Sphinx
```

Estos serán añadidos al índice y convertidos en enlaces en la navegación HTML generada.

1.3. Archivo Makefile

- **Propósito:** Contiene comandos para construir la documentación en diferentes formatos.
- **Uso:**
 - Se ejecuta make html para construir documentación HTML.
 - Se utiliza con el comando make desde una terminal.
 - Sistema Operativo: Es un archivo diseñado común en sistemas **Unix, Linux y macOS**.
 - Otros comandos incluyen:
 - make latexpdf: Genera un PDF si se tiene LaTeX configurado.
 - make epub: Genera un eBook.

1.4. Archivo make.bat

- **Propósito:** Archivo equivalente al Makefile para sistemas Windows.
- **Uso:**
 - Ejecuta comandos como make.bat html en el terminal de Windows, para construir documentación HTML.

- Sistema operativo: Es un archivo diseñado para el sistema **Windows**.

1.4.2 2. Flujo de Trabajo con MyST-Parser

1. **Configuración de MyST-Parser en conf.py:** Asegúrate de que myst-parser esté habilitado:

```
extensions = ['myst_parser']
source_suffix = {
    '.md': 'markdown',
}
```

2. **Estructura Markdown:**

- Los archivos Markdown (.md) reemplazan los .rst. Escribe tus documentos con encabezados #, listas, bloques de código, etc.
- Usa MyST para integrar características de Sphinx en Markdown. Por ejemplo:

```
# Título del Documento

{toctree}
:maxdepth: 2

archivo1.md
carpeta/archivo2.md
```

3. **Compilación de la documentación:**

- Ejecuta:
make html
- Los archivos HTML generados estarán en docs/build/html.

4. **Personalización:**

- Añade CSS en _static/ para personalizar el diseño.
- Usa _templates/ para modificar las plantillas.

Esta estructura te permite trabajar con Markdown fácilmente, aprovechando la potencia de Sphinx y MyST-Parser para generar documentación de alta calidad.

1.5 Creación de ficheros de código y generación automática de documentación

1.5.1 1. Creación del fichero de código Python

Se crea un fichero de código Python con **docstring en formato Google Style**. Este formato facilita la generación automática de documentación, ya que utiliza una estructura clara y legible tanto para humanos como para herramientas de generación de documentación.

```
1 def add_numbers(a, b):
2     """Add two numbers.
3
4     Args:
```

(continúe en la próxima página)

(proviene de la página anterior)

```

5     a (float): The first number.
6     b (float): The second number.
7
8     Returns:
9         float: The sum of the two numbers.
10    """
11    return a + b
12
13
14    def subtract_numbers(a, b):
15        """
16        Subtract one number from another.
17
18        Args:
19            a (float): The number to subtract from.
20            b (float): The number to subtract.
21
22        Returns:
23            float: The result of the subtraction.
24        """
25        return a - b
26
27
28    def multiply_numbers(a, b):
29        """Multiply two numbers.
30
31        Args:
32            a (float): The first number.
33            b (float): The second number.
34
35        Returns:
36            float: The product of the two numbers.
37        """
38        return a * b
39
40
41    def divide_numbers(a, b):
42        """Divide one number by another.
43
44        Args:
45            a (float): The numerator.
46            b (float): The denominator.
47
48        Returns:
49            float: The result of the division.
50
51        Raises:
52            ValueError: If the denominator is zero.
53        """
54        if b == 0:
55            raise ValueError("The denominator cannot be zero.")
56        return a / b

```

Explicación del formato Google Style:

- Args: Define los argumentos que la función recibe, indicando su nombre, tipo y propósito.
- Returns: Describe lo que devuelve la función, incluyendo el tipo de dato.

- Raises(opcional): Especifica las excepciones que la función podría lanzar.

1.2. Pasos para generar la documentación automáticamente

1. Instalación de Sphinx:

Hay que asegurarse de tener Sphinx instalado en el entorno de trabajo. Si no, hay que instalarlo con:

```
pip install sphinx
```

2. Ejecución de sphinx-apidoc:

sphinx-apidoc es una herramienta de línea de comandos incluida en Sphinx que genera automáticamente archivos de documentación para módulos y paquetes Python. Su propósito principal es agilizar el proceso de creación de documentación basada en docstrings, generando archivos de entrada para Sphinx a partir de la estructura del código fuente. Esta herramienta es especialmente útil en proyectos grandes con múltiples módulos y paquetes, ya que automatiza la creación de archivos de documentación necesarios para cada módulo, en el formato configurado (por ejemplo, .rst, .md o cualquier otro formato compatible).

```
sphinx-apidoc -o docs/source src/ --separate --suffix .md
```

- -o docs/source: Indica la carpeta donde se generarán los archivos de documentación.
- src/: Especifica el directorio donde se encuentran los archivos de código Python.
- --separate: Genera un archivo separado para cada módulo detectado en lugar de consolidar toda la documentación en un único archivo.
- --suffix .md: Configura la extensión de los archivos generados como .md. Si esta opción no se especifica, el valor predeterminado es .rst.

Al ejecutar el comando sphinx-apidoc, en nuestro caso los archivos generados son:

- math_operations
- modules

3. Compilación de la documentación:

Una vez se tengan los archivos generados, hay que compilar la documentación usando:

- Hay que navegar al directorio de documentación (docs/):

```
cd docs
```

- Luego hay que ejecutar el siguiente comando para generar la documentación HTML:

```
sphinx-build -b html source build
```

Esto creará una carpeta docs/build/html/ con los archivos HTML de la documentación generada.

1.5.2 2. Resultados obtenidos

Cuando ejecutas el comando sphinx-apidoc, se generan una serie de archivos en el directorio docs/source que contienen la estructura básica de la documentación para el proyecto. Este comando facilita la generación automática de documentación a partir del código fuente, especialmente de los docstrings de los módulos y clases Python.

2.1. Estructura de Archivos Generada

Al ejecutar sphinx-apidoc -o docs/source src/, se creará una serie de archivos en la carpeta docs/source:

a. index.rst o index.md

Este es el archivo principal de la documentación. Contiene la estructura base, la cual incluirá todos los módulos generados por sphinx-apidoc. En nuestro caso al estar usando Markdown, este archivo es index.md.

Ejemplo de un archivo index.md:

```
# Documentación de T05: Sphinx y Myst-Parser

## Índice

```{toctree}
:maxdepth: 2
:glob:

configuracion_inicial/*
comandos_mas_usados/*
```

Este archivo contiene:

- Un título de la documentación.
- Una sección de índice que lista los módulos o archivos que se incluyen en la documentación generada.

**b. Archivos .rst o .md para cada módulo Python**

Cada módulo de Python dentro de la carpeta src/ tendrá un archivo generado. Nuestro proyecto tiene el módulo math\_operations.py, se genera el archivo:

- math\_operations.rst o math\_operations.md

Este archivo contienen la documentación generada a partir de los docstrings de los módulos. Por ejemplo, si tu módulo math\_operations.py tiene funciones con docstrings, el archivo generado incluirá lo siguiente:

```
math_operations module
=====

.. automodule:: math_operations
 :members:
 :undoc-members:
 :show-inheritance:
```

Este archivo generado tiene varias secciones importantes:

- automodule:: math\_operations: Esto indica que Sphinx debe generar la documentación automáticamente para el módulo math\_operations.
- :members:: Esto genera la documentación para todas las funciones y clases dentro del módulo.
- :undoc-members:: Si hay funciones o clases sin docstrings, esta directiva las incluirá en la documentación.
- :show-inheritance:: Si hay clases, muestra también la jerarquía de herencia.

Este es el formato que Sphinx usa para incluir la documentación del código fuente, asegurándose de que los cambios en los archivos de código fuente se reflejen automáticamente en la documentación.

c. modules.rst o modules.md

Este archivo lista todos los módulos generados, lo cual es útil si hay muchos módulos en el proyecto. Por ejemplo:

```
src
===

.. toctree::
 :maxdepth: 4

 math_operations
```

Este archivo crea un índice de módulos que Sphinx incluirá en la documentación final.

## 2.2. Contenido de los Archivos Generados

### a. Documentación Generada Automáticamente desde el Código

El objetivo principal de sphinx-apidoc es generar documentación automáticamente a partir de los docstrings en el código Python. Para cada archivo .py que contiene docstrings, se generará una representación en reStructuredText o Markdown.

Por ejemplo, nuestro archivo math\_operations.py con el siguiente código:

```
def add_numbers(a, b):
 """Suma dos números."""
 return a + b
```

El archivo math\_operations.md generado tendrá algo similar a esto:

```
math_operations
=====

.. automodule:: math_operations
 :members:
 :undoc-members:
 :show-inheritance:

add_numbers(a, b)

Suma dos números.
```

La directiva automodule incluye el módulo completo, y las funciones o clases del módulo se detallan debajo con sus respectivas descripciones.

### b. Incluir Todo el Código del Proyecto

Los archivos generados también incluyen código fuente si usamos la extensión viewcode en el archivo conf.py de Sphinx. Esto agrega enlaces al código fuente y puede incluir la visualización completa del código si se activa la opción correspondiente.

### c. Dependencias de Módulos

Si el proyecto tiene dependencias de otros módulos dentro de `src/`, el sistema de documentación puede generar automáticamente enlaces a esos módulos y sus funciones.

### 2.3. Uso de los Archivos Generados

Una vez que sphinx-apidoc ha generado estos archivos, puedes:

- **Editar manualmente** el archivo `index.rst` o `index.md` para personalizar la estructura de la documentación.
- **Incluir o excluir módulos** de la documentación generada editando los archivos `.rst` o `.md` generados.
- **Ejecutar** `sphinx-build` para generar la documentación final en formato HTML, PDF, etc.

Por ejemplo, se puede generar la documentación HTML con el siguiente comando:

```
sphinx-build -b html source build
```

Esto generará una carpeta `build` que contendrá la documentación final en formato HTML. Se pueden abrir estos archivos en el navegador para visualizar la documentación generada.

### 2.4. Personalización de la Documentación Generada

Aunque los archivos generados por sphinx-apidoc contienen una buena base de documentación, se pueden personalizar aún más, por ejemplo:

- **Agregar enlaces y referencias:** Se puede usar `:ref:` y `:doc:` para vincular otros documentos generados por Sphinx.
- **Agregar secciones adicionales:** Se puede agregar más texto explicativo o personalizar la apariencia con archivos CSS o HTML.

### 1.5.3 3. Conclusiones

- **Automatización:** El uso de sphinx-apidoc simplifica la generación de documentación a partir de código Python, ahorrando tiempo y esfuerzo.
- **Legibilidad:** El uso de docstring en formato Google Style mejora la claridad y consistencia de la documentación.
- **Compatibilidad:** Los archivos `.rst` generados por Sphinx son fácilmente convertibles a otros formatos, como `.md`, para adaptarse a distintas necesidades de publicación.

## 1.6 Generación de HTML a partir de la documentación

La generación de documentación HTML es uno de los puntos clave en un flujo de trabajo con **Sphinx** y **MyST-Parser**. Este proceso convierte los archivos fuente (Markdown o reStructuredText) en una interfaz web navegable y estilizada.

### 1.6.1 1. Preparar la estructura del proyecto

Asegúrate de que la estructura de tu proyecto tiene la siguiente organización básica:

```
docs/
├── source/
│ ├── conf.py <- Archivo de configuración principal
│ ├── index.md <- Página principal (puede ser index.rst o index.md)
│ ├── _static/ <- Recursos estáticos (imágenes, CSS, JS)
│ ├── _templates/ <- Plantillas personalizadas para HTML
│ └── secciones/ <- Más archivos .md o .rst
├── make.bat <- Script para construir la documentación (Windows)
└── Makefile <- Script para construir la documentación (Linux/macOS)
```

## 1.6.2 2. Crear contenido en Markdown

En la carpeta source, crea o edita el archivo index.md (o index.rst si trabajas con reStructuredText).

### 2.1. Ejemplo de index.md:

```
Documentación del Proyecto

¡Bienvenido a la documentación oficial del proyecto!

Introducción

Esta documentación cubre todas las características clave.

Secciones

- [Instalación](instalacion.md)
- [Uso Básico](uso_basico.md)

Referencias

Más información en [referencia técnica](referencia.md).
```

**Importante:** En caso de usar Markdown, asegúrate de que en conf.py tengas habilitado MyST-Parser:

```
extensions = [
 "myst_parser"
]
```

## 1.6.3 3. Configurar la salida HTML en conf.py

Abre el archivo conf.py y verifica lo siguiente:

```
Tema HTML
html_theme = 'sphinx_rtd_theme'

Directorios de recursos estáticos
html_static_path = ['_static']

Tipo de archivos fuente
```

(continúe en la próxima página)

(proviene de la página anterior)

```
source_suffix = {
 '.rst': 'restructuredtext',
 '.md': 'markdown',
}
```

Estas configuraciones aseguran que Sphinx procese correctamente los archivos .md y genere la salida en HTML.

---

## 1.6.4 4. Generar la documentación HTML

- **Windows:** Abre una terminal en la carpeta raíz de tu proyecto (donde está make.bat) y ejecuta:

```
make.bat html
```

- **Linux/macOS:** En terminal, usa el comando:

```
make html
```

### 4.1. ¿Qué sucede aquí?

- Sphinx lee los archivos .md o .rst en el directorio source.
  - Convierte estos archivos en HTML.
  - El resultado se guarda en el directorio build/html.
- 

## 1.6.5 5. Explorar la salida HTML

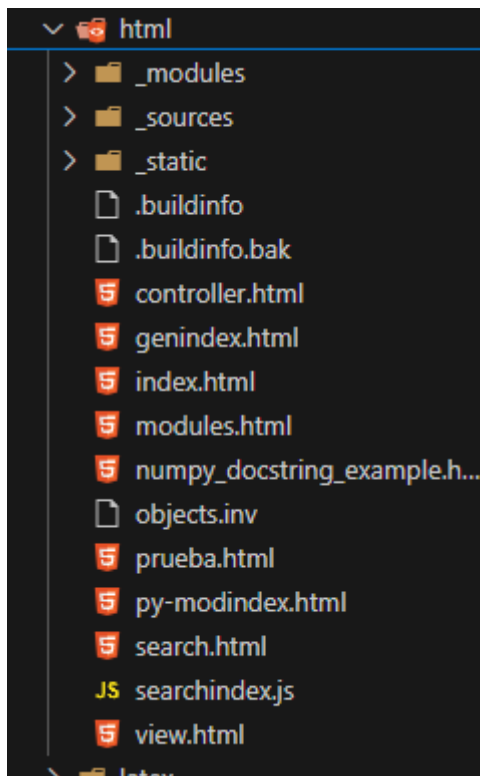
Una vez finalizado el proceso, navega hasta:

```
docs/build/html/index.html
```

- Abre el archivo index.html en tu navegador web preferido.
- Deberías ver una interfaz estilizada con el contenido de tu documentación.



### 5.1. Ejemplo de salida HTML:



## 1.7 Generación de PDF a partir de la documentación

Generar un documento PDF a partir de tu documentación es una funcionalidad poderosa que ofrece **Sphinx** gracias a su integración con **LaTeX**. Sin embargo, este proceso requiere una configuración adecuada y algunas dependencias instaladas previamente.

### 1.7.1 1. Requisitos previos

Antes de generar archivos PDF, asegúrate de cumplir con los siguientes requisitos:

#### 1. Instalar LaTeX:

- En **Windows**: Descarga e instala **MiKTeX** desde <https://miktex.org/download>.
- En **Linux**: Instala TeX Live con:

```
sudo apt install texlive-latex-base texlive-fonts-recommended texlive-latex-extra
```

- En **macOS**: Instala **MacTeX** desde <https://tug.org/mactex/>.

#### 2. Verificar la instalación:

Abre una terminal y verifica que `pdflatex` está disponible:

```
pdflatex --version
```

Deberías ver información sobre la versión instalada.

#### 3. Paquetes adicionales de LaTeX:

Es posible que necesites algunos paquetes adicionales. Puedes instalarlos manualmente o permitir que MiKTeX los descargue automáticamente cuando sea necesario.

## 1.7.2 2. Configurar el archivo conf.py

Abre el archivo conf.py y asegúrate de tener la siguiente configuración para LaTeX:

```
Usar pdflatex como motor de compilación
latex_engine = 'pdflatex'

Configurar opciones para el documento PDF
latex_elements = {
 'papersize': 'a4paper', # Tamaño del papel
 'pointsize': '10pt', # Tamaño de fuente principal
 'preamble': r'\usepackage[utf8]{inputenc}', # Codificación de entrada
 'figure_align': 'htbp' # Alineación de figuras
}

Documentos PDF a generar
latex_documents = [
 ('index', 'documento.tex', 'Título del Documento',
 'Autor del Documento', 'manual'),
]
```

### 2.1 Explicación de las opciones clave:

- `papersize`: Define el tamaño de la página (a4paper o letterpaper).
  - `pointsize`: Tamaño de la fuente en puntos (10pt, 11pt, 12pt).
  - `preamble`: Permite agregar configuraciones adicionales de LaTeX.
  - `figure_align`: Controla la alineación de las figuras.
- 

## 1.7.3 3. Generar archivos LaTeX

En la terminal, dirígete a la carpeta raíz donde se encuentra el archivo make.bat (en Windows) o Makefile (en Linux/macOS).

### ▪ Windows:

```
make.bat latex
```

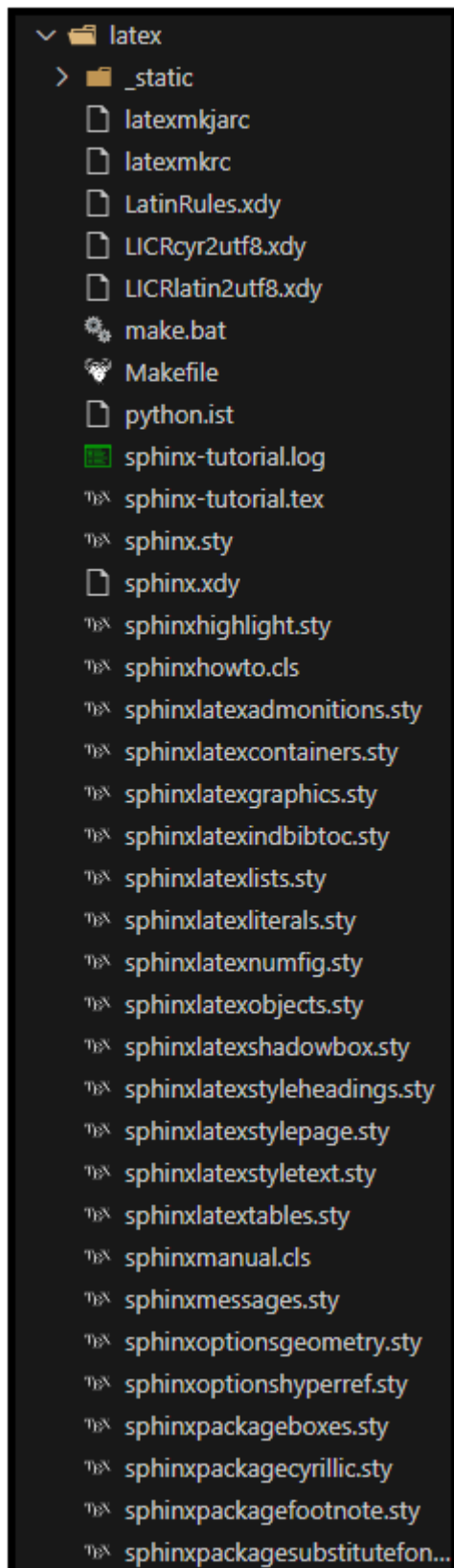
### ▪ Linux/macOS:

```
make latex
```

### 3.1. Resultado esperado:

Esto generará una carpeta build/latex con varios archivos, incluyendo un archivo .tex.

**Ejemplo de resultado:**



### 1.7.4 4. Compilar el archivo .tex a PDF

Dirígete a la carpeta build/latex y compila el archivo .tex:

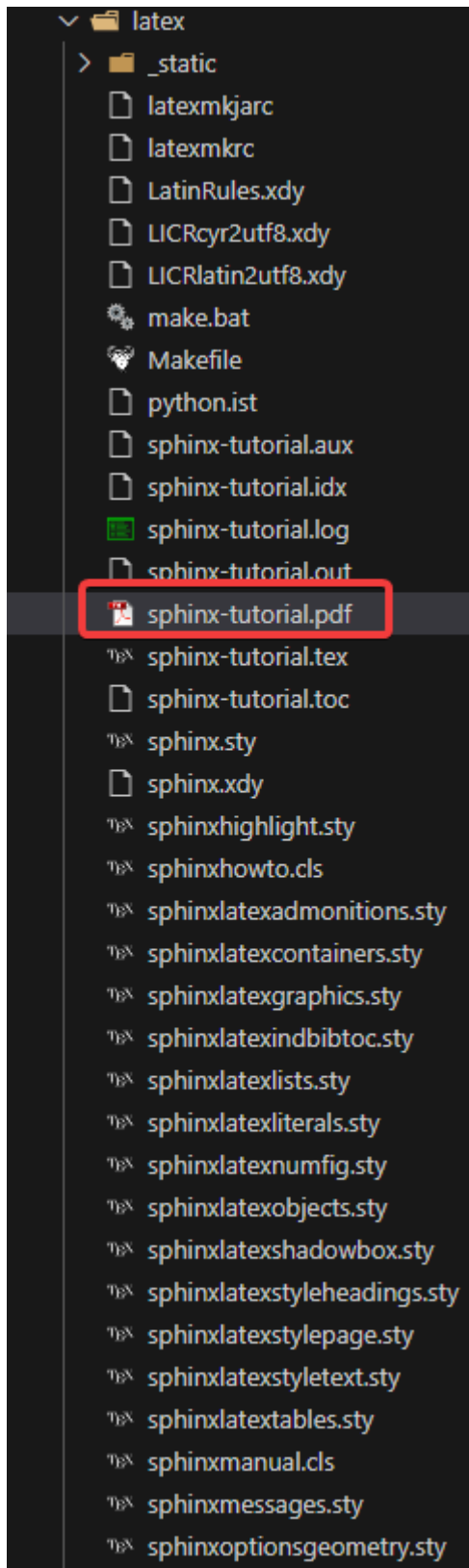
```
pdflatex documento.tex
```

- Si aparece un diálogo de instalación de paquetes (en MiKTeX), selecciona un servidor local (por ejemplo, España) y permite la instalación de los paquetes requeridos.
- Es posible que necesites ejecutar el comando varias veces para generar correctamente el índice y las referencias cruzadas.

#### 4.1. Resultado esperado:

Al finalizar, verás un archivo documento.pdf en la misma carpeta.

**Ejemplo de resultado final:**



## 1.7.5 5. Verificar y revisar el PDF

Abre el archivo PDF con tu lector de PDF preferido y verifica que:

- La tabla de contenidos esté correctamente generada.
- Las imágenes y tablas estén alineadas adecuadamente.
- Las referencias cruzadas y enlaces funcionen correctamente.

## 1.7.6 6. Solución de problemas comunes

### 6.1. Error: Paquete LaTeX faltante

- Si ves mensajes de error sobre paquetes faltantes, instala los paquetes manualmente o permite que MiKTeX los descargue automáticamente.

### 6.2. Error de codificación UTF-8

- Agrega esta línea en `conf.py` dentro de `latex_elements`:

```
'preamble': r'\usepackage[utf8]{inputenc}'
```

### 6.3. Error de compilación en tablas o figuras

- Asegúrate de que las tablas y figuras estén bien formateadas y no excedan los márgenes de la página.

## 1.8 Guía de uso MyST – Parser

### 1.8.1 1. Typography

El soporte de tipografía en MyST-Parser permite a los usuarios crear contenido estilizado y bien formateado en documentos escritos en Markdown. MyST-Parser extiende Markdown estándar con características avanzadas que incluyen símbolos especiales, estilos de texto enriquecidos y más.

#### Características principales de la tipografía en MyST-Parser

##### 1. Énfasis de texto

MyST-Parser utiliza una sintaxis similar a Markdown para resaltar texto con *énfasis* (cursiva) y **fuerte énfasis** (negrita).

- **Cursiva:** Usa un solo asterisco (\*) o guion bajo (\_):

```
Texto en cursiva o _Texto en cursiva_
```

Resultado: *Texto en cursiva*

- **Negrita:** Usa dos asteriscos (\*\*) o guiones bajos dobles (\_\_):

```
Texto en negrita o __Texto en negrita__
```

Resultado: **Texto en negrita**

- **Combinación de estilos:** Puedes combinar cursiva y negrita:

```
Texto combinado
```

Resultado: *Texto combinado*

## 2. Texto tachado

Para tachar texto, usa dos tildes (~~):

```
~~Texto tachado~~
```

Resultado: ~~Texto tachado~~

## 3. Citas tipográficas

Puedes usar > para indicar una cita en bloque:

```
> Este es un bloque de cita.
```

Resultado:

Este es un bloque de cita.

## 4. Subíndice y superíndice

MyST-Parser incluye soporte para subíndices y superíndices utilizando notación inline.

- **Subíndice:** Usa ~ para envolver el texto del subíndice:

```
H~2~O
```

Resultado: H<sub>2</sub>O

- **Superíndice:** Usa ^ para envolver el texto del superíndice:

```
E = mc^2^
```

Resultado: E = mc<sup>2</sup>

## 5. Carácter de escape

Para incluir caracteres literales que podrían ser interpretados como sintaxis (por ejemplo, \*, \_ o ~), precede el carácter con una barra invertida (\):

```
Este es un asterisco literal: *
```

Resultado: Este es un asterisco literal: \*

## 6. Separadores horizontales

Crea líneas horizontales para dividir contenido con tres o más guiones (---), asteriscos (\*\*\*) o guiones bajos (\_\_\_):

```

```

Resultado:

## 7. Letras y símbolos especiales

MyST-Parser soporta caracteres Unicode directamente en los archivos Markdown. Además, puedes utilizar entidades HTML si lo necesitas:

Corazón: o &#x2764;

Resultado: Corazón: o

### 1.8.2 2. Admonitions

Las admonitions son bloques especiales utilizados para resaltar contenido importante o específico, como notas, advertencias o ejemplos. MyST-Parser permite crear una amplia variedad de admonitions mediante una sintaxis intuitiva que mejora la claridad y el formato de los documentos.

#### Características de las admonitions en MyST-Parser

##### 1. Sintaxis básica

Para crear una admonition, usa la siguiente sintaxis:

```
```{admonition} Título opcional :class: type Contenido de la admonition.```
```

Resultado:

Título opcional

Contenido de la admonition.

2. Tipos predefinidos de admonitions

MyST-Parser incluye varios tipos de admonitions predefinidas para diferentes propósitos. La sintaxis es similar a la básica, pero especificando el tipo de admonition.

■ Nota (note):

```
```{admonition} Nota :class: note Esta es una nota informativa.```
```

- Así se vería:

#### Nota

Esta es una nota informativa.

###### ■ Advertencia (warning):

```
```{admonition} Advertencia :class: warning Ten cuidado con esta advertencia.```
```

- Así se vería:

Advertencia

Ten cuidado con esta advertencia.

■ **Consejo** (tip):

```
```{admonition} Consejo :class: tip Aquí tienes un consejo útil. ```
```

- Así se vería:


 **Consejo**

Aquí tienes un consejo útil.

■ **Importante** (important):

```
```{admonition} Importante :class: important Esta información es crucial. ```
```

- Así se vería:


 **Importante**

Esta información es crucial.

■ **Peligro** (danger):

```
```{admonition} Peligroso :class: danger ¡Atención! Esta acción puede ser peligrosa. ```
```

- Así se vería:


 **Peligroso**

¡Atención! Esta acción puede ser peligrosa.

■ **Precaución** (caution):

```
```{admonition} Precaución :class: caution Ten cuidado al realizar esta acción, podría haber ↪ consecuencias inesperadas. ```
```

- Así se vería:


 **Precaución**

Ten cuidado al realizar esta acción, podría haber consecuencias inesperadas.

■ **Atención** (attention):

```
```{admonition} Atención :class: attetion Esta sección requiere tu atención inmediata para ↪ evitar errores críticos. ```
```

- Así se vería:

 **Atención**

Esta sección requiere tu atención inmediata para evitar errores críticos.

■ **Pista** (hint):

```
```{admonition} Pista :class: hint Aquí hay una sugerencia útil para facilitar tu tarea. ```
```

- Así se vería:

i Pista

Aquí hay una sugerencia útil para facilitar tu tarea.

■ **Precaución** (seealso):

```
```{admonition} Véase también :class: seealso Consulta la sección anterior para obtener más contexto sobre este tema. ```
```

- Así se vería:

#### **i Véase también**

Consulta la sección anterior para obtener más contexto sobre este tema.

### 3. Admonitions con títulos personalizados

Puedes añadir un título personalizado a cualquier admonition especificando el título después del tipo:

```
```{admonition} Título personalizado :class: tip Contenido con un título personalizado. ```
```

Resultado:

i Título personalizado

Contenido con un título personalizado.

4. Admonitions anidadas

Puedes anidar admonitions dentro de otras para mayor claridad. Por ejemplo:

```
```{admonition} Nota :class: note Esto es un ejemplo de nota.
  ```{admonition} Tip :class: tip Esto es un ejemplo de tip anidado. ```
```
```

Resultado:

#### **i Nota**

Esto es un ejemplo de nota.

#### **i Tip**

Esto es un ejemplo de tip anidado.

## 5. Admonitions con bloques de código

Puedes incluir bloques de código dentro de una admonition:

```
```{admonition} Ejemplo de código
:class: tip
```

Aquí tienes un bloque de código:

```
```python
print("Hola, mundo!")```
```

Resultado:

```
``` {admonition} Ejemplo de código
:class: tip
```

Aquí tienes un bloque de código:

```
```python
print("Hola, mundo!")```
```

## 6. Personalización de admonitions

### 6.1. Entender la Estructura de Admonitions en Sphinx

En Sphinx, las admonitions son bloques especiales que resaltan contenido específico, como notas, advertencias o ejemplos. Estos bloques se generan en el HTML final con clases CSS específicas que permiten su personalización.

Algunos tipos comunes explicados anteriormente son:

- note
- warning
- important
- hint
- caution
- attention
- seealso

#### 6.1.1. Estructura HTML Generada por una Admonition

Al compilar la documentación, una admonition podría generarse como:

```
<div class="admonition note">
 <p class="admonition-title">Nota</p>
 <p>Este es un contenido dentro de una admonition de tipo "note".</p>
</div>
```

## 6.2. Crear un Archivo de Estilos Personalizado en Sphinx

### 6.2.1. Estructura del Proyecto

Asegúrate de tener la siguiente estructura en tu proyecto Sphinx:

```
docs/
├── _static/
│ ├── custom.css # Archivo CSS personalizado
│ ├── conf.py
│ ├── index.rst
│ └── ...
```

### 6.2.2. Editar el Archivo `custom.css`

Abre (o crea) el archivo `_static/custom.css` y agrega estilos personalizados para las admonitions:

```
/* === Estilos Generales para todas las Admonitions === */
.admonition {
 border-left: 5px solid #444;
 border-radius: 5px;
 margin: 1em 0;
 padding: 1em;
 background-color: #f9f9f9;
}

/* === Título de las Admonitions === */
.admonition .admonition-title {
 font-weight: bold;
 font-size: 1.1em;
 margin-bottom: 0.5em;
}

/* === Personalización por Tipo de Admonition === */

/* NOTE */
.admonition.note {
 border-left-color: #2196F3;
 background-color: #E3F2FD;
}
.admonition.note .admonition-title {
 color: #0D47A1;
}

/* WARNING */
.admonition.warning {
 border-left-color: #FF9800;
 background-color: #FFF3E0;
}
.admonition.warning .admonition-title {
 color: #E65100;
}

/* IMPORTANT */
.admonition.important {
 border-left-color: #4CAF50;
```

(continúe en la próxima página)

(proviene de la página anterior)

```

 background-color: #E8F5E9;
}
.admonition.important .admonition-title {
 color: #1B5E20;
}

/* HINT */
.admonition.hint {
 border-left-color: #9C27B0;
 background-color: #F3E5F5;
}
.admonition.hint .admonition-title {
 color: #6A1B9A;
}

```

### 6.2.3. Explicación de los Estilos:

1. `.admonition`: Define un estilo base para todas las admonitions.
2. `.admonition-title`: Estiliza los títulos dentro de las admonitions.
3. `.admonition.note`, `.admonition.warning`, **etc.**: Estiliza cada tipo de admonition con colores específicos para los bordes y el fondo.

### 6.3. Configurar `conf.py` para Incluir CSS Personalizado

Abre tu archivo `conf.py` y verifica que la configuración sea correcta:

```

Habilitar myst_parser para soporte Markdown
extensions = [
 'myst_parser',
 'sphinx.ext.autodoc',
 'sphinx.ext.napoleon',
 'sphinx.ext.viewcode',
 'sphinx_rtd_theme',
]

Añadir estilos CSS personalizados
html_static_path = ['_static']
html_css_files = [
 'custom.css', # Asegúrate de que custom.css esté en la carpeta _static
]

```

### 6.4. Configurar el tema (opcional)

`html_theme = "sphinx_rtd_theme"`

- `html_static_path`: Especifica dónde buscar archivos estáticos.
- `html_css_files`: Carga el archivo `custom.css` en la salida HTML.

### 1.8.3 3. Images and Figures

El manejo de **imágenes** y **figuras** en MyST-Parser permite enriquecer la documentación con contenido visual, facilitando la comprensión de conceptos complejos y mejorando la presentación general de tus documentos.

En esta guía, aprenderás:

1. Cómo insertar imágenes básicas.
2. Cómo usar figuras con títulos y referencias.
3. Configurar opciones avanzadas para imágenes.
4. Personalizar estilos de imágenes y figuras.

#### 1. Imágenes Básicas en MyST-Parser

##### 1.1. Sintaxis Básica para Insertar una Imagen

En Markdown estándar, puedes insertar imágenes usando la siguiente sintaxis:

```
! [Texto alternativo](ruta/a/la/imagen.png)
```

- **Texto alternativo:** Describe la imagen para accesibilidad y en caso de que no se cargue correctamente.
- **Ruta de la imagen:** Puede ser una ruta relativa o absoluta.

**Ejemplo:**

```
! [Logo de MyST](../_static/images/logo.png)
```

**Resultado:**



#### 2. Figuras con Título y Referencia

##### 2.1. Insertar una Figura con Título

Una **figura** es una imagen con un título opcional y, en algunos casos, con referencias cruzadas para enlazarla desde otras partes del documento.

```

```{figure} ../_static/images/logo.png
:alt: Logo de MyST
:width: 200px
:align: center

```

Este es el logo oficial de MyST-Parser.````

Resultado:



Figura 1: Este es el logo oficial de MyST-Parser.

Explicación:

- :alt: Proporciona texto alternativo para la imagen.
- :width: Ajusta el ancho de la imagen (en píxeles o porcentaje).
- :align: Alinea la imagen (center, left, right).
- El contenido debajo de la directiva actúa como el **título de la figura**.

2.2. Referencia Cruzada a una Figura

Para referenciar la figura en otro punto del documento, añade la etiqueta :name: Referencia:

```

```{figure} ../_static/images/logo.png
:alt: Logo de MyST
:width: 200px
:align: center

```

Este es el logo oficial de MyST-Parser.````

Puedes consultar el logo en la [Figura 1](#).

#### Resultado:



**Figura 1:** Este es el logo oficial de MyST-Parser.

### 3. Opciones Avanzadas para Imágenes y Figuras

MyST-Parser ofrece más opciones para controlar el comportamiento de imágenes y figuras.

#### 3.1. Escalar Imágenes

```
```{figure} ../_static/images/logo.png
:alt: Logo de MyST
:scale: 50%
```

Logo escalado al 50% de su tamaño original.```

- :scale: Ajusta el tamaño de la imagen como un porcentaje de su tamaño original.

3.2. Alinear Imágenes

```
```{figure} ../_static/images/logo.png
:alt: Logo de MyST
:align: right
```

Imagen alineada a la derecha.```

- :align: center (centro)
- :align: left (izquierda)
- :align: right (derecha)

#### 3.3. Enlazar Imágenes

Puedes hacer que una imagen sea un enlace:



```
[![Logo de MyST](../_static/images/logo.png)](https://myst-parser.readthedocs.io)
```

**Resultado:**

### 1.8.4 4. Tables

Las **tablas** son una herramienta esencial en la documentación técnica, ya que permiten organizar y presentar datos de forma clara y estructurada. **MyST-Parser** extiende la funcionalidad de Markdown para ofrecer opciones avanzadas de creación y personalización de tablas en **Sphinx**.

## 1. Tablas Básicas en MyST-Parser

### 1.1. Sintaxis Básica de Tablas Markdown

Puedes crear tablas simples utilizando la sintaxis básica de Markdown:

```
| Encabezado 1 | Encabezado 2 | Encabezado 3 |
|-----|-----|-----|
| Celda 1 | Celda 2 | Celda 3 |
| Celda 4 | Celda 5 | Celda 6 |
```

**Explicación:**

- Las tuberías (|) delimitan columnas.
- La segunda línea (---) define los encabezados.
- Cada fila se define en una nueva línea.

**Resultado:**

Encabezado 1	Encabezado 2	Encabezado 3
Celda 1	Celda 2	Celda 3
Celda 4	Celda 5	Celda 6

## 2. Tablas con Alineación

### 2.1. Alinear el Contenido de las Columnas

Puedes controlar la alineación del texto dentro de las columnas usando los símbolos ::

Izquierda	Centro	Derecha	
:-----	:-----:	-----:	
Celda 1	Celda 2	Celda 3	
Celda 4	Celda 5	Celda 6	

#### Explicación:

- :--- → Alineación a la izquierda.
- :---: → Alineación centrada.
- ---: → Alineación a la derecha.

#### Resultado:

Izquierda	Centro	Derecha
Celda 1	Celda 2	Celda 3
Celda 4	Celda 5	Celda 6

## 3. Tablas Complejas con Formato Avanzado

### 3.1. Crear Tablas con MyST Directivas

MyST-Parser admite el uso de directivas para crear tablas más avanzadas con opciones adicionales:

```

'''{list-table} Título de la Tabla
:header-rows: 1
:widths: 20 40 40

* - Treat
 - Quantity
 - Description
* - Albatross
 - 2.99
 - On a stick!
* - Crunchy Frog
 - 1.49
 - If we took the bones out, it wouldn't be
 crunchy, now would it?
* - Gannet Ripple
 - 1.99
 - On a stick! '''

```

**Explicación:**

- `:header-rows:` → Define cuántas filas son para encabezados (en este caso, 1).
- `:widths:` → Define el ancho de cada columna (en porcentaje).
- Cada fila de la tabla se define con `* -`.

**Resultado:**

Tabla 1: Título de la Tabla

Treat	Quantity	Description
Albatross	2.99	On a stick!
Crunchy Frog	1.49	If we took the bones out, it wouldn't be crunchy, now would it?
Gannet Ripple	1.99	On a stick!

**4. Tablas con Referencias y Nombres****4.1. Agregar una Etiqueta a una Tabla**

Puedes etiquetar una tabla para referenciarla más adelante:

```

```{list-table} Título de la Tabla
:header-rows: 1
:name: tabla-ejemplo

* - Encabezado A
- Encabezado B
* - Valor 1
- Valor 2```

```

Explicación:

- `:name:` Asigna un identificador único a la tabla.
- Puedes referenciar la tabla utilizando `#nombre-de-la-tabla`.

1.8.5 5. Source code and APIs

El uso de código fuente y APIs en la documentación técnica es esencial para proporcionar ejemplos prácticos, fragmentos reutilizables y una referencia clara para desarrolladores. MyST-Parser, combinado con Sphinx, ofrece herramientas avanzadas para formatear, resaltar y documentar código fuente y APIs de forma clara y organizada.

1. Mostrar Código Fuente en Markdown**1.1. Bloques de Código (Code Blocks)**

Para mostrar bloques de código en MyST-Parser, puedes usar la sintaxis estándar de Markdown con tres acentos graves (````):

```
```python
def hola_mundo():
 print("¡Hola, mundo!")```
```

**Resultado:**

```
def hola_mundo():
 print("¡Hola, mundo!")
```

- Puedes usar otros lenguajes como javascript, html, css, etc.

**2. Incrustar Código en Línea**

Puedes incrustar fragmentos de código dentro de una línea de texto utilizando acentos graves simples (```):

El comando ``print("Hola")`` muestra un mensaje en la consola.

**Resultado:**

El comando `print("Hola")` muestra un mensaje en la consola.

**3. Resaltar Fragmentos de Código (Highlighting)**

Puedes resaltar partes específicas de un bloque de código utilizando la directiva `{code-block}` de MyST-Parser:

```
```{code-block} python
:emphasize-lines: 2

def suma(a, b):
    resultado = a + b # Esta línea está resaltada
    return resultado```
```

Resultado:

```
def suma(a, b):
    resultado = a + b # Esta línea está resaltada
    return resultado
```

Explicación:

- `:emphasize-lines:` → Resalta líneas específicas dentro del bloque.
- Puedes especificar varias líneas con 1,3 o rangos con 1-3.

4. Mostrar Archivos de Código Completo

Si quieres incluir un archivo de código fuente completo en tu documentación, puedes usar la directiva `literalinclude`:

```

{literalinclude} ../_static/ejemplo.py
:language: python
:linenos:
:emphasize-lines: 2

```

Parámetros:

- :language: → Define el lenguaje para el resaltado de sintaxis.
- :linenos: → Muestra números de línea.
- :emphasize-lines: → Resalta líneas específicas.

Estructura del Proyecto:

```

docs/
├── source/
│   ├── conf.py
│   ├── codigo/
│   │   └── ejemplo.py

```

ejemplo.py:

```

def saludo():
    print("¡Hola desde un archivo externo!")

```

Resultado en la documentación:

```

1 def saludo():
2     print("¡Hola desde un archivo externo!")

```

5. Documentación de APIs

5.1. Autogenerar Documentación de APIs

Sphinx puede autogenerar documentación de APIs usando la extensión `sphinx.ext.autodoc`.

1. **Asegúrate de tener habilitada la extensión en `conf.py`:**

```

extensions = [
    "sphinx.ext.autodoc",
    "myst_parser"
]

```

2. **Crea un archivo `.rst` o `.md` para tu módulo:**

```

# Documentación de API

## Módulo `mi_modulo`

{autofunction} mi_modulo.saludo

```

3. **Estructura del Proyecto:**

```
docs/
├── source/
│   ├── conf.py
│   └── mi_modulo.py
```

mi_modulo.py:

```
def saludo():
    """Esta función imprime un saludo."""
    print("¡Hola desde la API!")
```

4. Compila la Documentación:

```
make html
```

1.8.6 6. Cross-references

Las **referencias cruzadas (cross-references)** permiten enlazar diferentes secciones, encabezados, archivos, imágenes, tablas, funciones y otros elementos dentro de tu documentación. Esto facilita la navegación y mantiene la coherencia en documentos extensos.

1. Referencias Cruzadas a Secciones

1.1. Sintaxis Básica

En **MyST-Parser**, puedes referenciar encabezados usando etiquetas (#) y referencias directas:

```
(sec-mi-seccion)=
## Mi Sección Importante
```

Puedes referenciar esta sección usando:

```
[Enlace a la sección](#sec-mi-seccion)
```

- (sec-mi-seccion)=: Crea una etiqueta para la sección.
- [Enlace a la sección](#sec-mi-seccion): Enlaza a la etiqueta.

1.2. Referencia desde otro archivo

Si la sección está en otro archivo, usa:

```
[Ir a Mi Sección](otro_archivo.md#sec-mi-seccion)
```

2. Referencias Cruzadas a Archivos

Puedes enlazar directamente a otros archivos Markdown:

```
[Ir a la Guía de Estilo](guia_estilo.md)
```

3. Referencias a Funciones, Clases y Métodos (Autodoc)

Si estás documentando con `sphinx.ext.autodoc`, puedes enlazar elementos de código automáticamente.

Consulte la función `[`suma`](codigo.ejemplo_autodoc.suma)` para más detalles.

También puedes usar referencias automáticas:

```
:func:`codigo.ejemplo_autodoc.suma`
:class:`codigo.ejemplo_autodoc.Calculadora`
```

4. Referencias a Figuras e Imágenes

Al agregar una imagen con una etiqueta, puedes referenciarla más adelante:

```
(fig-ejemplo)=
![:Ejemplo de Imagen](imagen.png)
```

Puedes ver la figura anterior aquí: `[Figura 1](#fig-ejemplo)`

5. Referencias a Tablas

De manera similar, puedes etiquetar tablas para referencias cruzadas:

```
(tab-ejemplo)=
| Encabezado 1 | Encabezado 2 |
|-----|-----|
| Valor 1 | Valor 2 |
```

Referencia a la tabla: `[Tabla 1](#tab-ejemplo)`

6. Referencias con la Directiva `ref`

También puedes utilizar la directiva `ref` de Sphinx:

Consulte la sección `:ref: `sec-mi-seccion`` para más detalles.

- `:ref:` permite una referencia robusta y puede redirigir aunque cambie el nombre del archivo.
- Usa etiquetas únicas para evitar conflictos.

7. Referencias a Admonitions

Si usas etiquetas en admonitions, también puedes enlazarlas:

```
(adm-importante)=
````{admonition} Nota Importante
Este es un mensaje importante.````
Puedes revisar la [Nota Importante](#adm-importante).
```

## 8. Buenas Prácticas para Referencias Cruzadas

- **Etiqueta todo:** Usa etiquetas únicas para secciones, imágenes, tablas y bloques importantes.
- **Prefijos útiles:** Usa prefijos claros, como sec-, fig-, tab-, adm-.
- **Evita enlaces rotos:** Usa el comando `make linkcheck` para verificar enlaces en tu documentación.

## 9. Verificar Referencias Cruzadas

Al compilar tu documentación:

```
make clean
make html
make linkcheck
```

- `make linkcheck`: Verifica que todos los enlaces y referencias funcionen correctamente.

### 1.8.7 7. Organising-Content

Sphinx te permite organizar tu contenido en múltiples documentos e incluir contenido de otros documentos.

Esta sección describe cómo hacerlo con MyST Markdown.

#### 1 Estructura del documento

Los documentos de MyST Markdown individuales están estructurados utilizando *encabezados*.

Todos los encabezados en el nivel raíz del documento se incluyen en la Tabla de Contenidos (ToC) de esa página.

Muchos temas de HTML ya incluyen esta ToC en una barra lateral, pero también puedes incluirla en el contenido principal de la página usando la directiva `contents`:

Opciones:

- **:depth:** Especifica la profundidad de la ToC.
- **:local:** Solo incluye encabezados de la sección actual del documento.
- **:backlinks:** Incluye un enlace a la ToC al final de cada sección.
- **:class:** Permite añadir una clase CSS personalizada a la ToC.

#### Advertencia

Debido a que la estructura del documento se determina mediante los encabezados, es problemático tener encabezados «no consecutivos» en un documento, como:

```
Encabezado 1
Encabezado 3
```

Si deseas incluir un encabezado así, puedes usar la extensión `attrs_block` para envolver el encabezado en un div, de modo que no se incluya en la ToC:

```
Encabezado 1
```

**Encabezado 3**



**i Configurar un título en el front-matter**

Added in version 0.17.0.

La configuración `myst_title_to_header = True` permite que una clave `title` esté presente en el *front matter* del documento.

Esto será utilizado como el encabezado del documento (analizado como Markdown). Por ejemplo:

```

title: Mi Título con *énfasis*

```

sería equivalente a:

```
Mi Título con *énfasis*
```

**2 Insertar otros documentos directamente en el documento actual**

La directiva `include` permite insertar el contenido de otro documento directamente en el flujo del documento actual.

**➡ Ver también**

La directiva `literalinclude` para incluir código fuente desde archivos.

Las siguientes opciones permiten incluir solo una parte de un documento:

- **:start-line:** Solo incluye el contenido a partir de este número de línea (la numeración comienza en 1, y los números negativos cuentan desde el final).
- **:end-line:** Solo incluye el contenido hasta (pero excluyendo) esta línea.
- **:start-after:** Solo incluye el contenido después de la primera aparición del texto especificado.
- **:end-before:** Solo incluye el contenido antes de la primera aparición del texto especificado (pero después de cualquier texto posterior).

Las siguientes opciones permiten modificar el contenido del documento incluido, para hacerlo relativo a la ubicación donde se está insertando:

- **:heading-offset:** Desplaza todos los niveles de encabezado por este número entero positivo, por ejemplo, cambiando `#` a `####`.
- **:relative-docs:** Convierte las referencias a archivos Markdown en relativas al documento actual si comienzan con un cierto prefijo.
- **:relative-images:** Convierte las referencias a imágenes Markdown en relativas al documento actual.

Opciones adicionales:

- **:encoding:** La codificación de texto del archivo externo.

**i Inclusión desde/hacia archivos reStructuredText**

Como se explica en *esta sección*, todas las directivas de MyST analizan su contenido como Markdown. Para incluir rST, primero debemos «envolver» la directiva en la directiva `eval-rst`:

Para incluir un archivo MyST dentro de un archivo reStructuredText, podemos usar la opción `parser` de la directiva `include`:

```
.. include:: include.md
 :parser: myst_parser.sphinx_
```

**Importante**

La opción parser requiere docutils >= 0.17.

### 3 Usar toctree para incluir otros documentos como hijos

Para estructurar un proyecto con múltiples documentos, se utiliza la .

Esto designa documentos como hijos del documento actual, construyendo una jerarquía de documentos anidada que comienza desde un .

Opciones del toctree:

- **:glob:** Coincide con todos los documentos disponibles y los inserta alfabéticamente.

Opciones de visualización dentro del documento:

- **:caption:** Un título para este toctree.
- **:hidden:** No se muestra dentro del documento.
- **:includehidden:** Incluye entradas toctree de hijos ocultos.
- **:maxdepth:** Profundidad de los sub-encabezados del documento mostrados.
- **:titlesonly:** Solo muestra el primer encabezado de nivel superior.
- **:reversed:** Invierte el orden de las entradas en la lista.

Opciones adicionales:

- **:name:** Permite referenciar el toctree.
- **:numbered:** Numera todos los encabezados en los hijos. Si se especifica un entero, este será la profundidad a numerar.

**Truco**

Los sub-árboles (sub-toctrees) se numeran automáticamente, así que no des la bandera :numbered: a estos.

#### 1.8.8 8. Syntax-Extension

El MyST-Parser es altamente configurable, utilizando la «plugabilidad» inherente del analizador . Las siguientes sintaxis son opcionales (deshabilitadas por defecto) y se pueden habilitar *a través de* la configuración conf.py de Sphinx. Su objetivo es generalmente añadir sintaxis más «amigables con Markdown»; a menudo habilitando y renderizando que amplían la [especificación de CommonMark](#).

Para habilitar todas las sintaxis explicadas a continuación:

```
myst_enable_extensions = [
 "amsmath",
 "attrs_inline",
 "colon_fence",
 "deflist",
 "dollarmath",
 "fieldlist",
 "html_admonition",
 "html_image",
 "linkify",
 "replacements",
```

(continúe en la próxima página)

(proviene de la página anterior)

```
"smartquotes",
"strikethrough",
"substitution",
"tasklist",
]
```

Distinto en la versión 0.13.0: `myst_enable_extensions` reemplaza las opciones de configuración anteriores: `admonition_enable`, `figure_enable`, `dmath_enable`, `amsmath_enable`, `deflist_enable`, `html_img_enable`.

## 1 Tipografía

Añadiendo `"smartquotes"` a `myst_enable_extensions` (en el `conf.py`) se convertirán automáticamente las comillas estándar en variantes de apertura/cierre:

- `'comillas simples'`: “comillas simples”
- `"comillas dobles"`: «comillas dobles»

Añadiendo `"replacements"` a `myst_enable_extensions` se convertirán automáticamente algunos textos tipográficos comunes:

Texto	Convertido
(c), (C)	(c)
(tm), (TM)	(tm)
(r), (R)	(r)
(p), (P)	(p)
+−	+−
...	...
'''	'''

## 2 Matemáticas

La matemática se analiza agregando a la lista `myst_enable_extensions` en el archivo `conf.py` una o ambas de las siguientes extensiones:

- `"dollarmath"` para análisis de matemáticas delimitadas por `$` y `$$`.
- `"amsmath"` para análisis directo de entornos LaTeX de [amsmath](#).

Esto permite el análisis de matemáticas como:

- Matemáticas en línea: `$...$`
- Matemáticas en bloque: `$$...$$`

Por ejemplo, `$x_{hey}=it+is^{math}$` se renderiza como  $(x_{hey}=it+is^{math})$ .

El bloque de matemáticas se especifica con `$$` y puede incluir etiquetas:

```
$$
e = mc^2
$$ (eqn:mejor)

Esta es la mejor ecuación {eq}`eqn:mejor`.
```

Opciones adicionales permiten controlar cómo se analiza el contenido de las matemáticas, como `myst_dmath_allow_space` y `myst_dmath_allow_digits`.

### 3 Listas de Definiciones

Añadiendo "deflist" a `myst_enable_extensions`, puedes utilizar listas de definiciones.

```
Término
: Definición

Otro término
: Definición extendida
```

Las definiciones pueden incluir elementos de bloque como párrafos, citas o bloques de código.

### 4 Listas de Tareas

Añadiendo "tasklist" a `myst_enable_extensions`, puedes utilizar listas de tareas:

```
- [] Tarea pendiente
- [x] Tarea completada
```

### 5 Sustituciones

Añadiendo "substitution" a `myst_enable_extensions`, puedes definir sustituciones, que se agregan al archivo `conf.py` o al encabezado del archivo:

```

myst:
 substitutions:
 key1: "Soy una **sustitución**"
 key2: |
      ````{note}
      Sustitución anidada: {{ key1 }}
      ````

```

Estas sustituciones se pueden utilizar en línea o como bloques y admiten anidamiento.

### 6 Anclajes Automáticos

El MyST Parser puede generar automáticamente «slugs» para anclas de encabezados, lo que permite referenciarlos directamente:

```
myst_heading_anchors = 3
```

Esto habilita enlaces como `[(#mi-ancla)]` para referenciar encabezados en Markdown.

## 1.8.9 9. Roles y Directivas

Los roles y las directivas ofrecen una manera de extender la sintaxis de MyST de manera ilimitada, interpretando un fragmento de texto como un tipo específico de marcado, según su nombre.

Casi todos los [roles de docutils](#), [directivas de docutils](#), , o se pueden usar en MyST.

## 1. Sintaxis

### 1.1 Directivas - una extensión a nivel de bloque

La sintaxis de las directivas se define con triples de acentos graves y llaves. Es efectivamente un bloque de código de Markdown con llaves alrededor del lenguaje, y un nombre de directiva en lugar de un nombre de lenguaje. Aquí está la estructura básica:

```
```{directivename} argumentos
:key1: valor1
:key2: valor2

Este es el contenido de la directiva
```

Por ejemplo:

i Esta es mi advertencia

Este es mi contenido

1.2 Configuración de las directivas (opciones)

Muchas directivas pueden tomar pares clave/valor en un bloque de opciones opcional al inicio de la directiva.

```
10 a = 2
11 print('mi primera línea')
12 print(f'mi {a}nda línea')
```

Las opciones también pueden incluir valores multilínea, o estar delimitadas con --- en lugar de :.

1.3 Roles - un punto de extensión en línea

Los roles son similares a las directivas, pero se usan en línea. Para definir un rol en línea, usa la siguiente forma:

```
{nombre-del-rol}`contenido del rol`
```

Por ejemplo:

Sabemos desde Pitágoras que $a^2 + b^2 = c^2$.

2. Roles y Directivas de MyST

2.1 Insertar la fecha y el tiempo de lectura

Added in version 0.14.0: El rol sub-ref y el conteo de palabras.

Puedes insertar la fecha de «última actualización» y el tiempo estimado de lectura en tu documento mediante sustituciones accesibles *a través de* sub-ref.

Por ejemplo:

```
> {sub-ref}`today` | {sub-ref}`wordcount-words` palabras | {sub-ref}`wordcount-minutes` min de ↵
↵lectura
```

today se reemplaza por la fecha en la que se analiza el documento, y el tiempo de lectura se calcula usando la configuración `myst_words_per_minute`.

1.9 Comandos utilizados al trabajar con Markdown, MyST-Parser, Sphinx

1.9.1 1. Crear un entorno virtual

```
python -m venv venv
```

1.9.2 2. Activar el entorno virtual

```
venv\Scripts\activate
```

1.9.3 3. Configuración inicial del proyecto Sphinx

Comando para iniciar un proyecto Sphinx

```
sphinx-quickstart
```

- Este comando inicializa un proyecto Sphinx, creando la estructura básica con carpetas como docs, source, y archivos como conf.py, index.rst, etc.

Comando para actualizar la versión de Sphinx

```
pip install -U sphinx
```

1. pip install: Instala un paquete de Python (en este caso, sphinx) desde el repositorio de PyPI (Python Package Index).
2. -U o --upgrade: Actualiza el paquete a la versión más reciente disponible en PyPI.

1.9.4 4. Instalación de dependencias necesarias

Instalar Sphinx

```
pip install sphinx
```

Instalar MyST-Parser

```
pip install myst-parser
```

- Habilita el soporte para archivos Markdown en Sphinx.

Instalar un tema

```
pip install sphinx_rtd_theme
```

- Instala el tema utilizado en la documentación.

1.9.5 4. Generación de documentación

Para generar documentación en formato HTML

```
make html
```

- Genera la documentación en HTML. El resultado se encuentra en la carpeta docs/build/html.

Para limpiar los archivos generados

```
make clean
```

- Elimina todos los archivos generados previamente (HTML, PDF, etc.).
-

Actualizar dependencias

```
pip install --upgrade sphinx myst-parser
```

1.10 src

```
.. toctree:: :maxdepth: 4
```

```
math_operations
```

1.11 math_operations module

```
.. automodule:: math_operations :members: :undoc-members: :show-inheritance:
```