

Proyecto Tema 5 Boletín 1

Juan Molero Marín, Julio Rodriguez Lopez, Julian Martinez Lorda

12 de enero de 2025

Bienvenido a la documentación de nuestro proyecto realizado en el tema 5 para la asignatura de Desarrollo de Interfaces.

En esta guía se detallará toda la información necesaria para poder utilizar el framework Sphinx y poder documentar todo tu proyecto de una forma clara y concisa. Además, también generaremos documentación de código con el formato docstring de Google:

1. Configuración Inicial del Proyecto

1.1 Creación del Entorno Virtual:

Nuestro primer paso dentro de este proyecto será la creación de un Entorno Virtual en Python. Para comenzar, ejecutaremos el siguiente comando:

```
python -m venv .env
```

Mediante **python -m venv**, crearemos un *entorno virtual* en el directorio donde estemos situados, en este caso, suele ser la raíz del proyecto, y posteriormente le indicamos el nombre que deseamos que tenga el mismo, en nuestro caso concreto, le hemos asignado **.env**.

A continuación, nuestro siguiente paso será el de instalar las **Librerías** que vamos a necesitar para poder realizar en el proyecto. Para ello, vamos a dirigirnos al siguiente apartado.

1.2 Instalación y Funcionamiento de las Librerías necesarias:

Para la realización de este proyecto de documentación, hemos necesitado las siguientes librerías:

Librería	Descripción
Sphinx	Herramienta para generar documentación en formatos como HTML o PDF. Soporta archivos en reStructuredText o Markdown, y es ideal para la documentación de proyectos de software.
Sphinx-rtd-theme	Permite aplicar el tema Read the Docs en Sphinx, proporcionando un diseño limpio y profesional. Es utilizado como estándar en la documentación.
MyST-Parser	Extiende a Sphinx para procesar archivos Markdown, además de reStructuredText, ofreciendo mayor flexibilidad a los usuarios.
Sphinxcontrib-napoleon	Extensión que facilita la integración de docstrings escritos en los estilos de Google o Numpy. Convierte estos formatos automáticamente al adecuado para el framework.

Para la instalación, se ha efectuado el siguiente comando:

```
pip install sphinx sphinx-rtd-theme myst-parser sphinxcontrib-napoleon
```

Cabe destacar que debemos de tener el **Entorno Virtual** activado en el momento de instalar dichas Librerías. Una vez se hayan instalado, el siguiente paso será plasmarlo dentro de un archivo de requisitos, **requirements.txt**, con el objetivo de que cualquiera pueda emplearlo en el futuro para poder realizar el mismo proyecto. Para eso se crea el archivo de texto plano en la raíz del proyecto, y posteriormente ejecutamos:

```
pip freeze > requirements.txt
```

1.3. Creación inicial del proyecto con Sphinx:

Para inicializar un proyecto con Sphinx, una vez tengamos nuestro entorno virtual creado y las librerías necesarias instaladas, es momento de efectuar el siguiente comando por la terminal:

```
sphinx-quickstart docs
```

El parámetro **docs** hace referencia al directorio donde se quiere ejecutar, en nuestro caso hemos creado y escogido este debido a que es una forma de separar la **documentación** que vamos a generar del directorio donde posteriormente se desarrollará código python, **src**. Tras ejecutarlo, comenzará un proceso donde se tendrá que responder a las siguientes cuestiones:

1. Lo primero será elegir entre separar o no los directorios entre el código fuente (source) y los archivos que se van a generar en base al mismo (build). En nuestro caso concreto, hemos seleccionado la opción “yes”

2. Nombre de nuestro proyecto

3. Autor/es del proyecto

4. Versión del proyecto

5. Idioma del proyecto

Al finalizar con las opciones escogidas, se nos formará la **estructura inicial del proyecto**, que explicaremos en el siguiente apartado.

1.4 Explicación de la Estructura inicial del proyecto:

Al ejecutar el comando explicado dentro del apartado anterior, y seguir el proceso de construcción en función de las especificaciones explicadas, se nos formará la siguiente estructura de proyecto (**en nuestro caso justo en el directorio docs al ejecutar el comando del apartado anterior desde allí**):

1. /build: Directorio donde se irán almacenando los archivos que genere posteriormente **Sphinx**, que se distinguirán en función del formato de salida dentro de distintos subdirectorios: **HMTL,PDF,...**

2. /source: Directorio principal donde se almacenarán los archivos fuente de la documentación, en este caso, serán en formato **Markdown**, así como el archivo de configuración principal del proyecto, en **Código Python**, respectivamente.

3. /_static: Subcarpeta dentro de **source** donde se incluirán recursos estáticos, tales como imágenes o archivos de CSS, entre otros.

4. `/_templates`: Carpeta destinada a archivos que personalicen nuestras plantillas HTML para generar la documentación.

5. `conf.py`: Es el archivo de configuración principal de nuestro proyecto: recoge parámetros como el nombre del mismo o las extensiones que tiene habilitada, así como el tema que se va a emplear, entre otros (**Será explicado más detalladamente en el siguiente apartado**).

6. `index.rst`: Archivo donde se inicializará la documentación. Será el punto de partida para generar un índice y reconducir al usuario a los distintos apartados que desee consultar, los cuales serán desarrollados en otros archivos **Markdown** aparte. Cabe destacar que debe ser renombrado de **.rst** a **.md**.

7. `make.bat`: Script de **Windows** que nos permitirá ejecutar ciertos comandos de construcción de forma posterior, con los que generaremos la documentación en los distintos formatos.

8. `Makefile`: Archivo empleado dentro de los **sistemas Unix (Linux/Mac)** para construir la documentación. Cumple la misma función que el **make.bat** en otros sistemas operativos

1.5. Explicación de `conf.py`

Nuestro archivo de configuración principal sería el siguiente:

```

9  project = 'Proyecto Tema 5 Equipo 01'
10 copyright = '2025, Juan Molero Marin'
11 author = 'Juan Molero Marin'
12 release = '1'
13
14
15 # Importamos módulos para configurar rutas y extensiones
16 import os # Módulo para gestionar rutas del sistema operativo
17 import sys # Módulo para modificar el path de búsqueda de Python
18 import sphinx_rtd_theme # Tema para mejorar el diseño visual de la documentación
19
20 # Configuramos la ruta al directorio raíz del proyecto
21 # os.path.abspath('.') obtiene la ruta absoluta del directorio superior al actual
22 # sys.path.insert(0, ...) añade esta ruta al inicio de sys.path, que es donde Python busca los módulos
23 sys.path.insert(0, os.path.abspath('../..'))
24
25
26 # -- General configuration -----
27 # https://www.sphinx-doc.org/en/master/usage/configuration.html#general-configuration
28
29 extensions = [
30     'sphinx.ext.autodoc', # Genera documentación automáticamente desde docstrings
31     'sphinx.ext.napoleon', # Interpreta docstrings en formatos Google y NumPy
32     'sphinx.ext.viewcode', # Agrega enlaces al código fuente en la documentación generada
33     'sphinx_rtd_theme', # Tema visual basado en Read the Docs
34     'myst_parser', # Habilita soporte para archivos Markdown
35 ]
36
37 # Configuración para soportar múltiples tipos de archivos fuente
38 source_suffix = {
39     '.rst': 'restructuredtext', # Archivos en formato reStructuredText
40     '.md': 'markdown', # Archivos en formato Markdown
41 }
42
43
44 templates_path = ['_templates']
45 exclude_patterns = ['_backup', 'deployment']
46
47 language = 'es'
48
49 # -- Options for HTML output -----
50 # https://www.sphinx-doc.org/en/master/usage/configuration.html#options-for-html-output
51
52 html_theme = 'sphinx_rtd_theme'
53 html_static_path = ['_static']
54

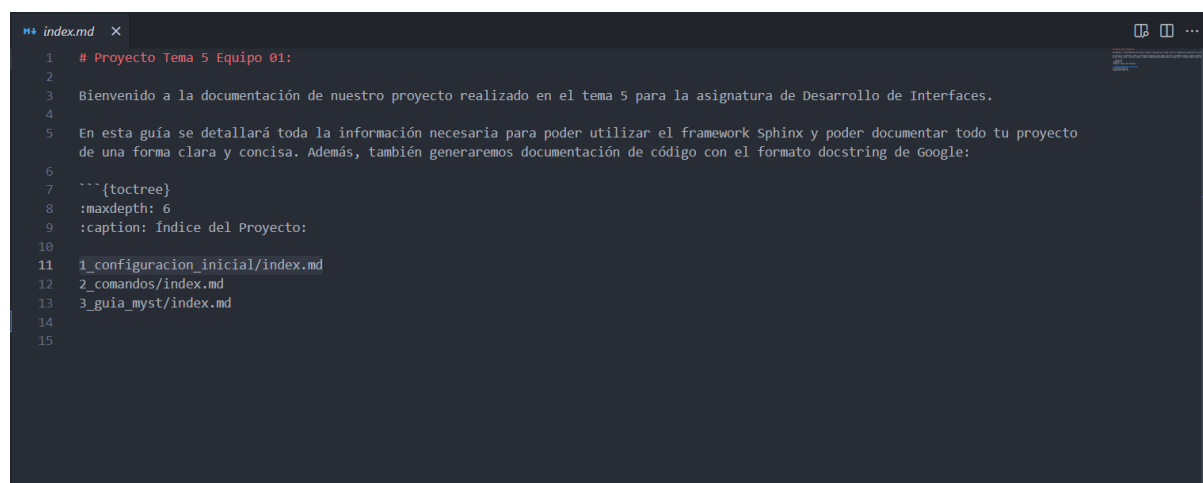
```

Parámetros del archivo de configuración:

Parámetro	Descripción
1. project	Indica el nombre del proyecto.
2. copyright	Se indicará una licencia en caso de querer establecerla.
3. author	Autor(es) que han realizado el proyecto.
4. release	Versión del proyecto. En este caso, es la 1 porque se estableció anteriormente.
5. Importaciones necesarias	Módulos os y sys , necesarios para el manejo de rutas, y el tema de Sphinx que se va a emplear.
6. Extensiones	Se utilizarán: autodoc , napoleon para docstring, viewcode para enlaces al código fuente, sphinx_rtd_theme y myst_parser para soportar Markdown.
7. Otras configuraciones importantes	Uso de source_suffix para soportar archivos fuente (rst y md). Configuración del idioma y las rutas hacia los directorios _static y _templates .

1.6. Explicacion del archivo index.md

Nuestro archivo index.md va a representar el índice principal dentro de la Aplicación. En él, hemos incluido una breve descripción del proyecto también. Lo podemos observar a continuación:



```
1 # Proyecto Tema 5 Equipo 01:
2
3 Bienvenido a la documentación de nuestro proyecto realizado en el tema 5 para la asignatura de Desarrollo de Interfaces.
4
5 En esta guía se detallará toda la información necesaria para poder utilizar el framework Sphinx y poder documentar todo tu proyecto
6 de una forma clara y concisa. Además, también generaremos documentación de código con el formato docstring de Google:
7
8 ```{toctree}
9 :maxdepth: 6
10 :caption: Índice del Proyecto:
11
12 1_configuracion_inicial/index.md
13 2_comandos/index.md
14 3_guia_myst/index.md
15
```

En este archivo podemos observar los siguientes elementos:

1. Título Principal del Proyecto

2. Índice que nos irá redirigiendo a cada uno de los apartados que se deben de explicar: cabe destacar que dichos apartados/directorios poseen sus respectivos índices, que nos volverán a redirigir a nuevos archivos **Markdown** donde ya se explicará el contenido.

3.Elemento Caption: nos ofrece una cabecera descriptiva para el proyecto. Se ha empleado justo antes de la generación del índice.

4.Elemento Maxdepth: elemento que nos define el nivel de profundidad que podrá adquirir nuestro árbol en cuanto a directorios y subdirectorios. En nuestro caso, hemos definido el nivel 6 o nivel 2 en otros índices posteriores.

El resto de elementos que podemos observar dentro de la página es **texto plano**, en el cual podemos plasmar la información deseada.

1.7. Creación de ficheros de código y generación automática de documentación

Para generar documentación en base a un archivo de código Python, tendremos que realizar los siguientes pasos:

1. Lo primero es la creación de un archivo de código que deseemos tranformar posteriormente en documentación mediante Sphinx. Para ello, nosotros en nuestro caso ubicamos un directorio /src a la altura de la raíz, donde nos encontramos con el archivo operaciones.py, respectivamente, que es donde se almacena nuestro código. Cabe destacar que posee el formato docstring con estilo Google:

```
1 def suma(a, b):
2     """Suma dos números.
3
4     Args:
5         a (float): Primer número.
6         b (float): Segundo número.
7
8     Returns:
9         float: Resultado de la suma.
10    """
11    return a + b
12
13
14 def resta(a, b):
15     """Resta dos números.
16
17     Args:
18         a (float): Primer número.
19         b (float): Segundo número.
20
21     Returns:
22         float: Resultado de la resta.
23    """
24    return a - b
25
26
27 def multiplicacion(a, b):
28     """Multiplica dos números.
29
30     Args:
31         a (float): Primer número.
32         b (float): Segundo número.
33
34     Returns:
35         float: Resultado de la multiplicación.
36    """
37    return a * b
38
39
40 def division(a, b):
41     """Divide dos números.
42
43     Args:
44         a (float): Dividendo.
45         b (float): Divisor. No puede ser cero.
```

2.Una vez tengamos nuestro archivo listo, debemos de ejecutar el siguiente comando por la terminal. Cabe destacar que en nuestro caso concreto fue desde el directorio raíz:

```
sphinx-apidoc -o ./docs src
```

En el cual:

Componente	Descripción
sphinx-apidoc	Genera documentación automáticamente a partir de módulos de Python.
o ./docs	Especifica el directorio donde queremos almacenar la información generada.
src	Indica el directorio de donde se debe obtener el código y los módulos que queremos documentar.

3.Tras la ejecución de dicho comando, se nos van a generar dos archivos rst, que hemos pasado a Markdown (operaciones y src). En nuestro caso concreto hemos eliminado el de operaciones y hemos dejado solo el de src, ya que simplemente con el podemos redigir el flujo desde el índice a la nueva documentación genera.

Para ello en este mismo archivo Markdown hacemos referencia a `src.md`, y dentro del mismo, ya enlazamos con la información que nos encontramos dentro de operaciones:

```
30
31 **3.Tras la ejecución de dicho comando, se nos van a generar dos archivos rst, que hemos pasado a Markdown (operaciones y src). En nuestro caso concreto hemos eliminado
32 el de operaciones y hemos dejado solo el de src, ya que simplemente con el podemos redigir el flujo desde el índice a la nueva documentación genera. Para ello en este
33 mismo archivo Markdown hacemos referencia a src.md, y dentro del mismo, ya enlazamos con la información que nos encontramos dentro de operaciones**
34
35 ```{toctree}
36 :maxdepth: 2
37
38 src.md
39
```

```
srcmd
1 ```{eval-rst}
2 .. automodule:: src.operaciones
3    :members:
4    :undoc-members:
5    :show-inheritance:
6
7
8
9
10
11
```

De esta forma, desde este apartado podremos acceder a la documentación que acabamos de generar directamente, la cual vamos a mostrar a continuación

Documentación Generada:

`src.operaciones.division(a, b)`

Divide dos números.

Parámetros

- **a** (*float*) – Dividendo.
- **b** (*float*) – Divisor. No puede ser cero.

Devuelve

Resultado de la división.

Tipo del valor devuelto

float

Muestra

ValueError – Si el divisor es cero.

`src.operaciones.multiplicacion(a, b)`

Multiplica dos números.

Parámetros

- **a** (*float*) – Primer número.

- **b** (*float*) – Segundo número.

Devuelve

Resultado de la multiplicación.

Tipo del valor devuelto

float

```
src.operaciones.operaciones_basicas(a, b)
```

Realiza todas las operaciones básicas entre dos números.

Parámetros

- **a** (*float*) – Primer número.
- **b** (*float*) – Segundo número.

Devuelve

Diccionario con los resultados de suma, resta, multiplicación y división.

Tipo del valor devuelto

dict

Muestra

ValueError – Si el divisor es cero al intentar dividir.

```
src.operaciones.resta(a, b)
```

Resta dos números.

Parámetros

- **a** (*float*) – Primer número.
- **b** (*float*) – Segundo número.

Devuelve

Resultado de la resta.

Tipo del valor devuelto

float

```
src.operaciones.suma(a, b)
```

Suma dos números.

Parámetros

- **a** (*float*) – Primer número.
- **b** (*float*) – Segundo número.

Devuelve

Resultado de la suma.

Tipo del valor devuelto

float

1.8. Generación de HTML a partir de la documentación

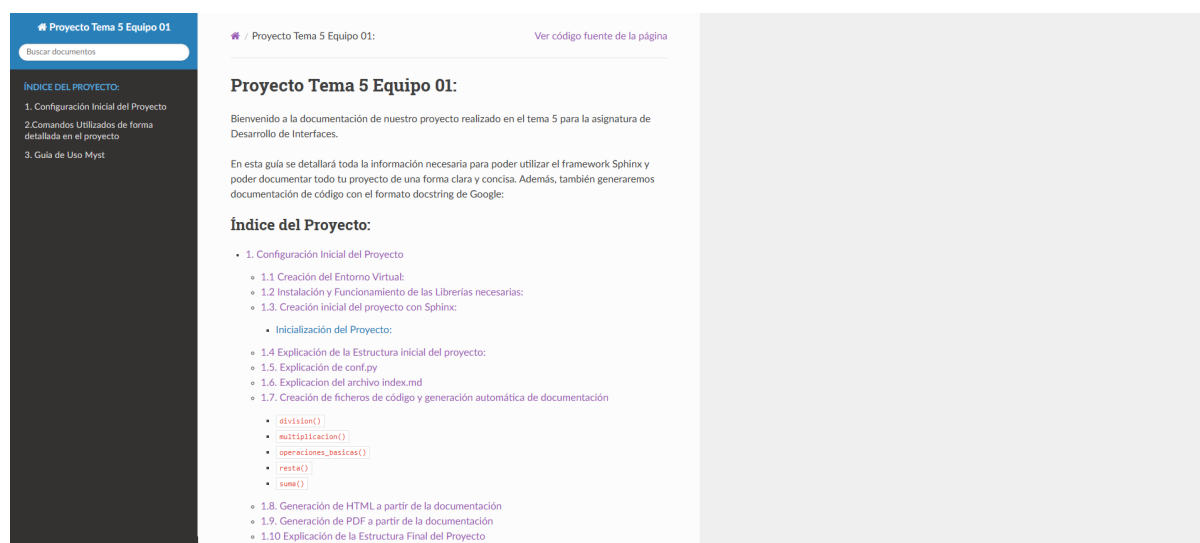
Una vez tengamos todo listo para generar nuestra documentación, será momento de efectuar el siguiente comando por consola:

```
sphinx-build -b html . ../build/html
```

Donde:

Parámetro/Parte	Descripción
sphinx-build	Herramienta de Sphinx con la cual generaremos la documentación.
-b	Indicador para especificar el formato en el cual se generará la documentación.
html	Formato respectivo en el que vamos a generar la documentación.
.	Directorio actual donde nos encontramos. En este caso, se ejecuta desde la carpeta source.
../build/html	Ruta hacia el directorio donde queremos que se guarden los archivos resultantes.

Una vez efectuado el comando, se nos generará nuestra documentación en HTML respectivamente, dentro del subdirectorio **html** ubicado en **/build**. Aunque se nos generarán varios archivos, realmente nos centraremos en el **index.html**, el cual si lo visualizamos podemos observar como ya contendría el índice principal con el cual podemos acceder a nuestra documentación:



1.9. Generación de PDF a partir de la documentación:

Para la generación de archivos PDF, en nuestro caso hemos escogido el sistema que emplea la tipografía LaTeX, mediante la cual gracias a un intérprete vamos a poder generar archivos en formato **.tex**, que posteriormente se compilarán en documentos **PDF** con una apariencia profesional. Para ello, el primer paso será tener una distribución de TeX propiamente instalada dentro del equipo donde se quiera realizar esta tarea. Es por eso que en nuestro caso hemos decidido instalar **MiKTeX**. El proceso de instalación será el siguiente:

1. Buscamos MiKTeX en nuestro navegador e indicamos en que parte del sistema deseamos instalarlo

2. Al instalarlo, abriremos posteriormente su consola para actualizarlo y agregaremos los paquetes de latexpdf necesarios para generar nuestros documentos

3. Finalmente, indicaremos al Sistema Operativo mediante la modificación de la Variable del Sistema Path la ruta de instalación de MiKTeX, con la finalidad de que se reconozca posteriormente el comando que tenemos que emplear

Tras realizar todos estos pasos, es momento de dirigirnos a nuestro proyecto y comenzar a generar la documentación. Para eso ejecutaremos el siguiente comando:

```
.\make.bat clean; .\make.bat latexpdf
```

Comando	Explicación
.\make.bat clean	Ejecuta el script <code>make.bat</code> con la opción <code>clean</code> , lo que significa que limpia cualquier archivo o compilación previa en el proyecto, eliminando archivos generados anteriormente.
.\make.bat latexpdf	Ejecuta el script <code>make.bat</code> con la opción <code>latexpdf</code> , que compila la documentación en formato LaTeX y la convierte en un archivo PDF.

Al ejecutar el comando, la carpeta `build` se nos reestructurará y se nos generará el archivo **.pdf** correspondiente dentro del subdirectorio **/latex**.

IMPORTANTE

- Para que el formato **PDF** salga con el formato deseado y con el fin de evitar errores, como duplicación de páginas en blanco o de índices en la numeración, se ha tenido que agregar la siguiente configuración dentro del archivo **conf.py**:

```
38
39 latex_elements = {
40     'papersize': 'a4paper',
41     'pointsize': '10pt',
42     'fontpkg': r'\usepackage{times}',
43     'preamble': r'\usepackage{fancyhdr}', # Puedes añadir o quitar paquetes aquí
44     'figure_align': 'htbp', # Esto ayuda a que las imágenes se alineen correctamente
45     'maketitle': r'\maketitle\nepage', # Esto puede ayudar a evitar la página en blanco
46     'secnumdepth': 0, # Desactiva la numeración de secciones y subsecciones
47     'tableofcontents': '', # Deja el índice sin numeración
48     'preamble': r'''
49         \setcounter{secnumdepth}{0} % Esto elimina la numeración de las secciones
50     ''', # Reconfigura la numeración de secciones para desactivarla
51 }
52
53
54 latex_documents = [
55     ('index', 'proyecto_tema_5_boletin_1.tex', 'Proyecto Tema 5 Boletín 1', 'Autor', 'manual'),
56 ]
57
```

1.10 Explicación de la Estructura Final del Proyecto:

RESUMEN ESTRUCTURA INICIAL:

Componen- te	Explicación
/build	Directorio donde se irán almacenando los archivos que genere posteriormente Sphinx , que se distinguirán en función del formato de salida dentro de distintos subdirectorios: HTML , PDF , ...
/source	Directorio principal donde se almacenarán los archivos fuente de la documentación, en este caso, serán en formato Markdown , así como el archivo de configuración principal del proyecto, en Código Python , respectivamente.
/static	Subcarpeta dentro de source donde se incluirán recursos estáticos, tales como imágenes o archivos de CSS, entre otros.
/templates	Carpeta destinada a archivos que personalicen nuestras plantillas HTML para generar la documentación.
conf.py	Es el archivo de configuración principal de nuestro proyecto: recoge parámetros como el nombre del mismo o las extensiones que tiene habilitada, así como el tema que se va a emplear, entre otros. (Será explicado más detalladamente en el siguiente apartado).
index.rst	Archivo donde se inicializará la documentación. Será el punto de partida para generar un índice y reconducir al usuario a los distintos apartados que desee consultar, los cuales serán desarrollados en otros archivos Markdown aparte. Cabe destacar que debe ser renombrado de .rst a .md .
make.bat	Script de Windows que nos permitirá ejecutar ciertos comandos de construcción de forma posterior, con los que generaremos la documentación en los distintos formatos.
Makefile	Archivo empleado dentro de los sistemas Unix (Linux/Mac) para construir la documentación. Cumple la misma función que el make.bat en otros sistemas operativos.

A la estructura inicial vista en el apartado anterior, con todos los cambios que se han ido realizando punto, deberíamos incluirle los siguientes directorios/archivos:

1. Subdirectorios /1_configuracion_inicial, 2_comandos, 3_guia_myst:

Estos subdirectorios son los que almacenarán todo el contenido en formato **Markdown**, con el cual hemos ido explicando y documentando todos los apartados y lo vamos a seguir haciendo hasta la finalización. Cabe destacar que cada uno de ellos posee su propio archivo **index.md**, con el objetivo de tener un índice y un direccionamiento más detallado dentro de la documentación.

2. Subdirectorios dentro del directorio /build:

Al generar nuestra documentación, se nos generarán dos directorios dentro del mismo:

- En el caso de generar **HTML**, serán **doctrees** y **html** respectivamente.
- Si generamos **PDF**, se sustituirá el de **html** por **latex**.

3. Directorio src:

Directorio donde durante nuestro proyecto hemos tenido que almacenar el código **Python**, con el que hemos generado la documentación con el formato **docstring** de Google posteriormente, lo cual ya se ha explicado de forma previa.

2. Comandos Utilizados de forma detallada en el proyecto

Estos serían los comandos que hemos utilizado dentro del proyecto:

Comando	Descripción
<code>python -m venv .env</code>	Este comando se ha utilizado en el proyecto para construir el entorno virtual desde la raíz del proyecto. <code>.env</code> indica el nombre que se asignará al entorno.
<code>..env\Scripts\activate</code>	Este comando se ha usado para activar el entorno de trabajo.
<code>pip install sphinx sphinx-rtd-theme myst-parser sphinxcontrib-napoleon</code>	Este comando instala las librerías necesarias para el proyecto: Sphinx, sphinx-rtd-theme, MyST-Parser y sphinxcontrib-napoleon.
<code>pip freeze > requirements.txt</code>	«pip freeze» enumera todas las dependencias instaladas en el entorno virtual o global de Python. El símbolo <code>></code> redirecciona la salida al archivo <code>requirements.txt</code> . Así, este comando guarda las dependencias del proyecto para que otros puedan instalar las mismas y ejecutar el proyecto sin problemas.
<code>sphinx-quickstart docs</code>	Este comando inicializa el proyecto con Sphinx. El parámetro <code>docs</code> indica el directorio donde se quiere ejecutar, en este caso, la carpeta <code>docs</code> . Durante la ejecución, el comando solicita separar los archivos fuente (<code>source</code>) de los generados (<code>build</code>). Se eligió la opción <code>yes</code> . También solicita el nombre del proyecto (indicado como «Proyecto Tema 5 Equipo 01»), los autores, la versión (1), y el idioma (español).
<code>cd .\docs</code>	Cambia el directorio actual a la carpeta <code>docs</code> , donde se encuentra el archivo <code>make.bat</code> . Este paso es necesario antes de ejecutar los siguientes comandos.
<code>.\make.bat clean</code>	Limpia todos los archivos temporales y resultados previos de la generación de la documentación, eliminando los contenidos de la carpeta <code>_build</code> .
<code>.\make.bat html</code>	Compila los archivos fuente de la documentación para generar la versión HTML, cuyo resultado se guarda en la carpeta <code>_build/html</code> .
<code>sphinx-apidoc -o ./docs src</code>	Genera archivos <code>.rst</code> en la carpeta <code>docs</code> a partir del código fuente ubicado en la carpeta <code>src</code> . Esto automatiza la creación de documentación para módulos y paquetes de Python.
<code>sphinx-build -b html . ./build/html</code>	Genera la documentación en formato HTML desde los archivos fuente (<code>.</code>) y guarda los resultados en el directorio <code>./build/html</code> .
<code>.\make.bat latexpdf</code>	Ejecuta el script <code>make.bat</code> con la opción <code>latexpdf</code> , que compila la documentación en formato LaTeX y la convierte en un archivo PDF.

3. Guia de Uso Myst

3.1 Tipografia MYST:

3.1.1 Encabezados

Los encabezados funcionan de la misma manera que en markdown, usando almohadillas consecutivas.

Ejemplo Encabezados

Encabezado de nivel 1
Encabezado de nivel 2
Encabezado de nivel 3
Encabezado de nivel 4

Lo que sería el resultado de:

Markdown

```
# Encabezado de nivel 1  
## Encabezado de nivel 2  
### Encabezado de nivel 3  
#### Encabezado de nivel 4
```

3.1.2 Párrafos

Un párrafo es una línea de texto separada por una línea en blanco.

Ejemplo Parrafo

Ejemplo Parrafo 1.

Ejemplo Parrafo 2.

3.1.3 Cambios de tema

Un cambio de tema separa dos secciones de temas distintos.

Lo que es el resultado de:

Markdown

```
---
```

3.1.4 Estilos de Texto

Estilos de Texto

Este es un texto **en negrita**.

Este es un texto *en cursiva*.

Este texto está en _{subíndice}.

Este texto está en ^{superíndice}.

Lo cual es el resultado de:

Markdown

```
Este es un texto en negrita.
```

```
Este es un texto en cursiva.
```

```
Este texto está en {sub}`subíndice`.
```

```
Este texto está en {sup}`superíndice`.
```


3.1.5 Salto de linea.

Un salto de linea separa dos lineas sin necesidad de una linea en blanco.

i Ejemplo Salto de Linea

Pulgas
tenía
el perro.

Lo cual es el resultado de:

i Markdown

```
Pulgas \  
tenía \  
el perro.
```

3.1.6 Listas numeradas y no numeradas

i Ejemplo Listas Numeradas y No Numeradas

- Lista no numerada empezada por -
- Con dos puntos
 - Y una lista anidada empezada por *
- Tambien puede hacerse a la inversa
- Con una lista no numerada empezada por *
 - Y listas anidadas empezadas por -
- E incluso
 1. Listas numeradas
 2. Anidadas
- 1. Las listas numeradas
- 2. Por supuesto
 1. Pueden tener
 2. Listas anidadas
- 3. Y estar solas

Lo cual es el resultado de:

i Markdown

```
- Lista no numerada empezada por \-  
- Con dos puntos  
  * Y una lista anidada empezada por \*
```

```
* Tambien puede hacerse a la inversa
* Con una lista no numerada empezada por \*
  - Y listas anidadas empezadas por \-
* E incluso
  1. Listas numeradas
  2. Anidadas

1. Las listas numeradas
2. Por supuesto
  1. Pueden tener
  2. Listas anidadas
3. Y estar solas
```

3.1.7 Citas

Aunque la mona se vista de seda, mona se queda.

Esto es el resultado de:

Markdown

```
> Aunque la mona se vista de seda, mona se queda.
```

3.1.8 Notas de Pie de Página

Ejemplo Notas de Pie de Página

- Esta referencia a nota de pie de página esta numerada manualmente.³
- Mientras que esta está numerada automaticamente.¹

Esto es el resultado de:

Markdown

```
- Esta referencia a nota de pie de página esta numerada manualmente.[^3]
- Mientras que esta está numerada automaticamente.[^myref]
[^myref]: This is an auto-numbered footnote definition.
[^3]: This is a manually-numbered footnote definition.
```

³ This is a manually-numbered footnote definition.

¹ This is an auto-numbered footnote definition.

3.2 Admonitions en MyST

3.2.1 ¿Qué son los Admonitions?

Los admonitions en MyST son bloques de contenido destacados que puedes usar para resaltar información importante, advertencias, ejemplos, o cualquier tipo de contenido que quieras destacar. MyST soporta varios tipos de admonitions predefinidos y personalizables.

Ejemplo de Admonition Básico

Este es un admonition básico sin tipo específico.

Esto se logra con:

Markdown

```
```{admonition} Ejemplo de Admonition Básico
Este es un admonition básico sin tipo específico.
```

### 3.2.2 Admonitions Predefinidos

MyST incluye admonitions predefinidos como `note`, `warning`, `tip`, y más.

#### Ejemplo de Admonitions Predefinidos

##### Nota

Este es un admonition de tipo «note».

##### Advertencia

Este es un admonition de tipo «warning».

##### Truco

Este es un admonition de tipo «tip».

Esto se logra con:

#### Markdown

```
```{note}
Este es un admonition de tipo "note".
```

```{warning}
Este es un admonition de tipo "warning".
```
```

```
Este es un admonition de tipo "warning".
...

...{tip}
Este es un admonition de tipo "tip".
...
```

### 3.2.3 Admonitions con Títulos Personalizados

Puedes agregar un título personalizado a un admonition para especificar su propósito.

#### Ejemplo con Título Personalizado

##### Recuerda Esto

Este es un admonition con un título personalizado.

Esto se logra con:

#### Markdown

```
...{admonition} Recuerda Esto
Este es un admonition con un título personalizado.
```

### 3.2.4 Admonitions con Código Formateado

Puedes incluir bloques de código dentro de un admonition para combinar explicaciones con ejemplos.

#### Ejemplo Admonition con Código

##### Advertencia

##### **Importante**

Cuidado con los índices fuera de rango en Python:

```
lista = [1, 2, 3]
print(lista[3]) # Esto genera un IndexError
```

Esto se logra con:

#### Markdown

```

````{warning}
### Importante
Cuidado con los índices fuera de rango en Python:

```python
lista = [1, 2, 3]
print(lista[3]) # Esto genera un IndexError

```

### 3.2.5 Personalización Avanzada

Puedes personalizar la apariencia de los admonitions con estilos adicionales usando etiquetas HTML o CSS dentro de ellos.

#### Ejemplo de Personalización

##### Truco

Este es un admonition personalizado con texto verde y negrita.

Esto se logra con:

#### Markdown

```

````{tip}
<span style="color: green; font-weight: bold;">Este es un admonition_
personalizado con texto verde y negrita.</span>

```

3.3 Imágenes y Figuras en MyST

3.3.1 Imágenes Básicas

Para insertar imágenes básicas en MyST, utiliza la sintaxis de Markdown estándar con `! [] ()`.

Ejemplo de Imagen Básica



Esto se logra con:

i Markdown

```
![Descripción de la imagen](../_static/fun-fish.png "Fun Fish is Fun")
```

3.3.2 Figuras con MyST

Las figuras en MyST son bloques enriquecidos que permiten agregar leyendas y configuraciones adicionales a las imágenes.

i Ejemplo de Figura con Leyenda



Figura 1: Este es un ejemplo de figura con una leyenda centrada.

Esto se logra con:

Markdown

```
```{figure} ../_static/fun-fish.png

align: center
name: pescado-divertido

Este es un ejemplo de figura con una leyenda centrada.
```

### 3.3.3 Ajustes de Alineación

Puedes controlar la alineación de las imágenes o figuras usando el parámetro `align`.

#### **Ejemplo de Alineación**





Figura 4: Figura centrada.

Esto se logra con:

#### Markdown

```
```{figure} ../_static/fun-fish.png
---
align: left
---
Figura alineada a la izquierda.
```
```{figure} ../_static/fun-fish.png
---
align: right
---
Figura alineada a la derecha.
```
```{figure} ../_static/fun-fish.png
---
align: center
---
Figura centrada.
```
```



### 3.3.4 Escalado de Imágenes

Puedes ajustar el tamaño de las imágenes con el parámetro `width`.

#### **i** Ejemplo de Escalado



Figura 5: Figura con un ancho del 50 %.



Figura 6: Figura con un ancho de 200px.

Esto se logra con:

#### **i** Markdown

```
```{figure} ../_static/fun-fish.png
---
width: 50%
---
Figura con un ancho del 50%.
```

```{figure} ../_static/fun-fish.png
---
width: 200px
---
Figura con un ancho de 200px.
```

3.3.5 Referencias a Figuras

Puedes asignar un nombre a las figuras con el parámetro `name` y luego referenciarlas en el texto.

Ejemplo de Referencias a Figuras



Figura 7: Figura con nombre para referencia.

Como se muestra en *Figura con nombre para referencia.*, puedes referenciar figuras en tu documento.

Esto se logra con:

Markdown

```
```{figure} ../_static/fun-fish.png

name: fun-fish

Figura con nombre para referencia.
```

Como se muestra en {ref}`fun-fish`, puedes referenciar figuras en tu
↪ documento.
```

3.4 Tablas en MyST

3.4.1 Tablas Básicas

Las tablas en MyST se pueden crear usando la sintaxis básica de Markdown.

i Ejemplo de Tabla Básica

| Columna 1 | Columna 2 | Columna 3 |
|-----------|-----------|-----------|
| Fila 1 | Valor 1 | Valor 2 |
| Fila 2 | Valor 3 | Valor 4 |

Esto se logra con:

i Markdown

```
| Columna 1 | Columna 2 | Columna 3 |
|-----|-----|-----|
| Fila 1 | Valor 1 | Valor 2 |
| Fila 2 | Valor 3 | Valor 4 |
```

3.4.2 Tablas Mejoradas con MyST

Puedes mejorar tus tablas utilizando directivas de MyST para agregar alineaciones, anchos y estilos avanzados.

i Ejemplo de Tabla Mejorada

| Cabecera 1 | Cabecera 2 | Cabecera 3 |
|---------------|---------------|---------------|
| Fila 1, Col 1 | Fila 1, Col 2 | Fila 1, Col 3 |
| Fila 2, Col 1 | Fila 2, Col 2 | Fila 2, Col 3 |

Esto se logra con:

i Markdown

```
```{list-table}

header-rows: 1

* - Cabecera 1
 - Cabecera 2
 - Cabecera 3
* - Fila 1, Col 1
 - Fila 1, Col 2
 - Fila 1, Col 3
* - Fila 2, Col 1
 - Fila 2, Col 2
 - Fila 2, Col 3
```

### 3.4.3 Tablas con Ancho Personalizado

Puedes ajustar el ancho de las columnas para que se adapten mejor al contenido.

#### Ejemplo de Ancho Personalizado

| Columna Estre-<br>cha | Columna Mediana         | Columna Ancha                                                 |
|-----------------------|-------------------------|---------------------------------------------------------------|
| Corto                 | Texto de longitud media | Este texto es más largo para demostrar el ancho personalizado |

Esto se logra con:

#### Markdown

```
```{list-table}
---
header-rows: 1
widths: 20 30 50
---
* - Columna Estrecha
  - Columna Mediana
  - Columna Ancha
* - Corto
  - Texto de longitud media
  - Este texto es más largo para demostrar el ancho personalizado
```

3.4.4 Combinar Celdas en Tablas

No es posible combinar celdas directamente con la sintaxis de Markdown o MyST, pero puedes usar HTML dentro de MyST si necesitas esta funcionalidad.

Ejemplo de Tabla con Celdas Combinadas

Esto se logra con:

Markdown

```
```{raw} html
<table>
 <tr>
 <th>Cabecera 1</th>
 <th>Cabecera 2</th>
 <th>Cabecera 3</th>
 </tr>
 <tr>
 <td rowspan="2">Celda combinada</td>
 <td>Fila 1, Col 2</td>
 <td>Fila 1, Col 3</td>
 </tr>
</table>
```

```

</tr>
<tr>
 <td>Fila 2, Col 2</td>
 <td>Fila 2, Col 3</td>
</tr>
</table>

```

## 3.5 Códigos Fuente y APIs en MyST

### 3.5.1 Bloques de Código

Para mostrar bloques de código en MyST, puedes usar la sintaxis estándar de Markdown con tres acentos graves ````` o usar directivas enriquecidas para mayor control.

#### Ejemplo de Bloque de Código Básico

```

Este es un bloque de código básico en Python
print("Hola, mundo!")

```

Esto se logra con:

#### Markdown

```

```python
# Este es un bloque de código básico en Python
print("Hola, mundo!")

```

3.5.2 Bloques de Código con Directivas

Las directivas de MyST permiten personalizar aún más los bloques de código, añadiendo títulos, líneas resaltadas, etc.

Ejemplo de Bloque de Código con Directivas

```

1  # Código Python con configuración personalizada
2  x = 10
3  print(f"El valor de x es {x}")

```

Esto se logra con:

Markdown

```

```{code-block} python

name: Ejemplo de Código Python
linenos: true

```

```
emphasize-lines: 2

Código Python con configuración personalizada
x = 10
print(f"El valor de x es {x}")
```

---

### 3.5.3 Incrustar APIs en Documentos

Puedes incluir documentaciones de APIs directamente en tus documentos usando la directiva `automodule` o `autoclass` de Sphinx.

#### Markdown

```
```{automodule} mi_modulo.ejemplo
---
members: true
undoc-members: true
show-inheritance: true
---
```

3.5.4 Resaltar Fragmentos de Código

Puedes resaltar fragmentos específicos dentro del texto utilizando bloques en línea.

Ejemplo de Código en Línea

Este es un ejemplo de `código en línea` dentro de un párrafo.

Esto se logra con:

Markdown

```
Este es un ejemplo de `código en línea` dentro de un párrafo.
```

3.5.5 Adjuntar Archivos de Código

Para incluir archivos de código externos en tu documentación, utiliza la directiva `literalinclude`.

Ejemplo de Inclusión de Archivo

```
1 def suma(a, b):
2     """Suma dos números.
3
4     Args:
5         a (float): Primer número.
6         b (float): Segundo número.
7
8     Returns:
9         float: Resultado de la suma.
10    """
11    return a + b
12
13
14 def resta(a, b):
15     """Resta dos números.
16
17     Args:
18         a (float): Primer número.
19         b (float): Segundo número.
20
21     Returns:
22         float: Resultado de la resta.
23    """
24    return a - b
25
26
27 def multiplicacion(a, b):
28     """Multiplica dos números.
29
30     Args:
31         a (float): Primer número.
32         b (float): Segundo número.
33
34     Returns:
35         float: Resultado de la multiplicación.
36    """
37    return a * b
38
39
40 def division(a, b):
41     """Divide dos números.
42
43     Args:
44         a (float): Dividendo.
45         b (float): Divisor. No puede ser cero.
46
47     Returns:
48         float: Resultado de la división.
49
50     Raises:
51         ValueError: Si el divisor es cero.
52    """
53    if b == 0:
54        raise ValueError("El divisor no puede ser cero.")
55    return a / b
56
57
```

```
58 def operaciones_basicas(a, b):
59     """Realiza todas las operaciones básicas entre dos números.
60
61     Args:
62         a (float): Primer número.
63         b (float): Segundo número.
64
65     Returns:
66         dict: Diccionario con los resultados de suma, resta,
67         ↪ multiplicación y división.
68
69     Raises:
70         ValueError: Si el divisor es cero al intentar dividir.
71     """
72     try:
73         return {
74             "suma": suma(a, b),
75             "resta": resta(a, b),
76             "multiplicacion": multiplicacion(a, b),
77             "division": division(a, b),
78         }
79     except ValueError as e:
80         return {"error": str(e)}
```

Esto se logra con:

Markdown

```
```{literalinclude} ../../../../src/operaciones.py

language: python
linenos: true
emphasize-lines: 3

```

### 3.5.6 Notas sobre la Configuración

Al usar directivas como `automodule` o `literalinclude`, asegúrate de que las rutas de los archivos sean correctas y de que la configuración de Sphinx permita el uso de estas directivas. Esto puede requerir ajustes en `conf.py` como:

```
extensions = [
 'sphinx.ext.autodoc',
 'sphinx.ext.napoleon',
 'sphinx.ext.viewcode',
]
```



## 3.6 Cross-References en MyST

### 3.6.1 Enlaces Internos Básicos

Puedes crear enlaces internos en tu documento MyST utilizando la sintaxis estándar de Markdown o referencias explícitas con etiquetas.

#### Ejemplo de Enlace Interno

Puedes saltar a la sección [3.6.2 Enlaces con Títulos](#3.6.2 Enlaces-con-titulos).

Esto se logra con:

#### Markdown

```
Puedes saltar a la sección [3.6.2 Enlaces con Títulos] (#enlaces-con-
↪titulos).
```

### 3.6.2 Enlaces con Títulos

Los encabezados en MyST automáticamente generan identificadores que pueden ser referenciados mediante # seguido del identificador. Si el encabezado tiene caracteres especiales, estos se reemplazan automáticamente (por ejemplo, espacios por guiones).

#### Ejemplo de Referencia a un Encabezado

Consulta la sección [3.6.1 Enlaces Internos Básicos](#3.6.1 Enlaces-internos-basicos) para más detalles.

Esto se logra con:

#### Markdown

```
Consulta la sección [3.6.1 Enlaces Internos Básicos] (#enlaces-internos-
↪basicos) para más detalles.
```

### 3.6.3 Etiquetas Personalizadas

Puedes agregar etiquetas personalizadas a las secciones o elementos usando la directiva {ref}. Esto permite crear nombres amigables para las referencias.

#### Ejemplo de Etiqueta Personalizada

```
(seccion-personalizada)=
Título de la Sección
```

Contenido de la sección con una etiqueta personalizada.

Y puedes referenciarla con:

```
:ref:`Consulta esta sección personalizada <seccion-personalizada>` para
←obtener más información.
```

### 3.6.4 Referencias Cruzadas a Figuras, Tablas y Código

Puedes referenciar figuras, tablas y bloques de código usando etiquetas y la directiva `:ref:`.

#### **i** Ejemplo de Referencia Cruzada



Figura 8: Este es un pez divertido.

Luego, puedes referenciar la figura con:

```
Mira la figura en {ref}`figura-divertida` para más detalles.
```

Esto se logra asignando una etiqueta en la figura y referenciándola.

### 3.6.5 Referencias Externas

Para enlazar a recursos externos, simplemente usa la sintaxis estándar de Markdown.

#### **i** Ejemplo de Enlace Externo

Visita la documentación de [MyST](#) para más información.

Esto se logra con:

#### **i** Markdown

Visita [la documentación de MyST] (<https://myst-parser.readthedocs.io/>) para más información.

Con estas herramientas, puedes estructurar tus documentos MyST con enlaces internos, referencias cruzadas y etiquetas personalizadas, mejorando la navegación y claridad de tus contenidos.

## 3.7 Organización del Contenido en MyST

### 3.7.1 Secciones y Encabezados

La organización básica del contenido en MyST se realiza mediante encabezados. Estos se generan con el uso de almohadillas #, con un número creciente para indicar la jerarquía.

#### **i** Ejemplo de Encabezados Jerárquicos

```
Título Principal
Subtítulo 1
Subtítulo 1.1
Subtítulo 1.1.1
```

Esto permite estructurar claramente el contenido en niveles.

### 3.7.2 Separadores de Temas

Los separadores --- ayudan a dividir secciones visualmente, indicando cambios de tema o contenido.

#### **i** Ejemplo de Separador

Este es el contenido antes del separador.

Este es el contenido después del separador.

Esto se escribe como:

#### **i** Markdown

```
Este es el contenido antes del separador.

Este es el contenido después del separador.
```

### 3.7.3 Archivos Incluidos

Para incluir contenido de otros archivos dentro de un documento, utiliza la directiva `include`.

#### Ejemplo de Inclusión de Archivo

```
```{include} ../path/to/otro_documento.md
:relative-docs: docs/
:relative-images:
```
```

Esto inserta el contenido del archivo especificado directamente en la posición del documento actual.

---

### 3.7.4 Índices y Tablas de Contenido

Puedes generar automáticamente un índice o tabla de contenido utilizando la directiva `toctree`.

#### Ejemplo de Tabla de Contenidos

```
```{toctree}
:maxdepth: 2
:hidden:

seccion1
seccion2
subdirectorio/seccion3
```
```

Esto genera una tabla de contenido basada en los archivos especificados.

---

### 3.7.5 Etiquetas y Referencias

Organiza contenido utilizando etiquetas personalizadas para referenciar secciones específicas. Usa la directiva `(identificador)=` para asignar una etiqueta.

#### Ejemplo de Etiqueta y Referencia

```
(etiqueta-seccion)=
Título de la Sección

Contenido de la sección etiquetada.

Consulta la sección {ref}`etiqueta-seccion` para más información.
```

## 3.8 Extensiones de Sintaxis en MyST

### 3.8.1 Extensiones Opcionales

El parser de MyST incluye varias extensiones opcionales que permiten agregar características avanzadas al contenido. Estas extensiones se pueden habilitar en la configuración de Sphinx mediante la clave `myst_enable_extensions`.

Ejemplo de configuración en `conf.py`:

```
myst_enable_extensions = [
 "dollarmath",
 "tasklist",
 "fieldlist"
]
```

A continuación, se explican tres extensiones con ejemplos:

### 3.8.2 Matemáticas con `dollarmath`

Permite escribir expresiones matemáticas en línea usando `$` para ecuaciones simples o `$$` para ecuaciones en bloques.

#### Ejemplo de Matemáticas con Dollarmath

```
El área de un círculo es $A = \pi r^2$.

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

Se renderizará como:

- En línea: (  $A = \pi r^2$  )
- Bloque:  
[  $E = mc^2$  ]

### 3.8.3 Listas de Tareas con `tasklist`

Esta extensión permite agregar listas de tareas con casillas marcables en Markdown.

#### Ejemplo de Lista de Tareas

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

Se renderizará como:

- [ ] Tarea pendiente

- [x] Tarea completada
  - [ ] Otra tarea
- 

### 3.8.4 Listas de Campos con fieldlist

Esta extensión se utiliza para estructurar definiciones con pares clave-valor en formato limpio.

#### Ejemplo de Lista de Campos

```
:autor: John Doe
:versión: 1.0
:fecha: 12-01-2025
```

Renderiza el contenido como una tabla de definiciones:

- **autor:** John Doe
  - **versión:** 1.0
  - **fecha:** 12-01-2025
- 

Habilitar estas extensiones puede enriquecer la presentación y funcionalidad del contenido de tu guía.

## 3.9 Roles y Directivas en MyST

### 3.9.1 ¿Qué son los Roles y las Directivas?

Los roles son marcadores en línea que agregan formato o funcionalidad, mientras que las directivas son bloques más grandes para incluir contenido dinámico o personalizado.

---

### 3.9.2 Ejemplo de Roles

#### **Rol: doc**

Permite crear enlaces a otros documentos en tu proyecto.

```
Consulta la documentación en {doc}`seccion1`.
```

Renderiza como:

Consulta la documentación en [Sección 1](#).

---

#### **Rol: ref**

Crea referencias a etiquetas definidas previamente.

```
Consulta más detalles en {ref}`etiqueta-seccion`.
```

Renderiza como:

Consulta más detalles en [Título de la Sección](#).

### Rol: numref

Permite referencias numeradas a figuras, tablas o secciones.

```
Consulta la figura {numref}`figura-1`.
```

## 3.9.3 Ejemplo de Directivas

### Directiva: figure

Incluye una figura con su título y descripción.

```
``{figure} ../images/imagen.png
:alt: Una descripción de la imagen
:width: 50%
```

Título de la Figura

```

Directiva: ``note``
Destaca contenido importante como una nota.
```

#### Nota

Recuerda actualizar la guía periódicamente.

Renderiza como:

```
> **Nota**: Recuerda actualizar la guía periódicamente.
```

```

Directiva: ``warning``
Muestra un mensaje de advertencia.
```

#### Advertencia

Este proceso puede sobrescribir archivos existentes.

Renderiza como:

```
> \textwarning **Advertencia**: Este proceso puede sobrescribir archivos_
↳existentes.
```

---

```
El uso de roles y directivas en MyST mejora la claridad y navegación del_
↳contenido.
```



Figura 3:  
Figura ali-  
neada a la  
derecha.



### S

`src.operaciones`, 8



## D

`division()` (en el módulo `src.operaciones`), 8

## M

`module`

`src.operaciones`, 8

`multiplicacion()` (en el módulo `src.operaciones`), 8

## O

`operaciones_basicas()` (en el módulo `src.operaciones`), 9

## R

`resta()` (en el módulo `src.operaciones`), 9

## S

`src.operaciones`

`module`, 8

`suma()` (en el módulo `src.operaciones`), 9