Documentacion de T05_DOC_PRUEBAS Documentation

version

2025, Grupo 03

enero 12, 2025

Contenido

SphinxProyecto Grupo 03

Índice General

Entorno Virtual

En este apartado explicaremos en detalle cómo crear y activar un entorno virtual para nuestro proyecto.

¿Qué es un entorno virtual?

Un entorno virtual es una herramienta que permite aislar las dependencias del proyecto para evitar conflictos con otras aplicaciones instaladas en el sistema.

Pasos para crear un entorno virtual

1. Asegúrate de tener instalado Python 3.6 o superior.
2. Ejecuta el siguiente comando para crear un entorno virtual:

   ```
   python -m venv nombre_del_entorno
   ```

3. Después, para activar el entorno virtual deberemos ejecutar el siguiente comando:

   ```
   .\(nombre_entorno)\Scripts\activate
   ```

Instalación de las Librerías Necesarias

En este apartado explicamos cómo instalar las librerías requeridas para el proyecto, incluyendo una breve descripción de su propósito.

¿Qué librerías necesitamos y para qué se utilizan?

1. **Sphinx**: Herramienta que permite generar documentación en varios formatos como HTML, PDF o ePub.
2. **sphinx-rtd-theme**: Proporciona un diseño visual moderno y profesional basado en el estilo de Read the Docs.
3. **myst-parser**: Permite escribir documentación utilizando archivos Markdown, además del formato reStructuredText.
4. **sphinx.ext.autodoc**: Extensión de Sphinx que genera automáticamente documentación a partir de los docstrings de los módulos, clases, métodos y funciones de un proyecto Python.
5. **sphinx.ext.napoleon**: Extensión que facilita la interpretación de docstrings en los formatos Google y NumPy. Esto permite que los desarrolladores escriban documentación en un estilo más natural, alineado con estas convenciones populares.
6. **sphinx_markdown_builder**: Extensión que permite a Sphinx generar documentación en formato Markdown como salida, además de los formatos estándar como HTML y PDF.

Pasos para instalar las librerías usando un archivo “requirements.txt”

Paso 1: Crear un archivo “requirements.txt”

Para facilitar la instalación, crea un archivo llamado “requirements.txt” con este contenido:

```
sphinx
sphinx-rtd-theme
myst-parser
sphinx.ext.autodoc
```

```
sphinx.ext.napoleon
sphinx_markdown_builder
sphinx-autodoc-typehints
```

Una vez introducido, guarda el archivo.

Paso 2: Instalar las librerías

Ahora, ejecuta el siguiente comando para instalar las librerías listadas:

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

Pasos para instalar las librerías usando un método manual.

Paso 1: Ejecutar el comando directamente

```
pip install sphinx sphinx-rtd-theme myst-parser sphinx_markdown_builder sphinx-autodoc-typeh
```

Creación inicial del proyecto con Sphinx

Sphinx es una herramienta potente para generar documentación técnica. Aquí, detallaremos los pasos para configurar el proyecto inicial.

Activación del entorno virtual

Para que funcionen los comandos que aparecerán mas adelante, deberemos activar nuestro entorno virtual, para ello, introduciremos el siguiente comando donde se encuentre nuestro entorno virtual:

```
.\(nombre_entorno)\Scripts\activate
```

Creación del proyecto inicial con Sphinx

Paso 1: Inicializar proyecto

Navega a la carpeta raíz de tu proyecto y ejecuta el siguiente comando:

```
sphinx-quickstart
```

Esto lo que hará es iniciar un asistente interactivo que configurará el proyecto inicial. A continuación, te mostraremos las preguntas típicas que aparecerán y cómo responderlas.

Paso 2: Preguntas en la creación

Separate source and build directories (y/n): Responde “y” para mantener una estructura más organizada con carpetas separadas para los archivos fuente (source/) y los archivos generados (_build/).

Project name: Escribe el nombre de tu proyecto, por ejemplo:

```
ProyectoSphinxG03
```

Author name: Escribe tu nombre como autor, por ejemplo:

```
Grupo 03
```

Project release: Indica la versión inicial de tu proyecto, como:

```
1.0
```

Do you want to use a Makefile? (y/n): Responde “y” para facilitar la generación de documentación en sistemas Linux/macOS.

Do you want to use a Windows command file? (y/n): Responde “y” si estás trabajando en Windows.

Language: Especifica el idioma de la documentación, en este caso:

```
es
```

Explicación de la Estructura inicial del proyecto

Una vez hayamos creado el proyecto con Sphinx, veremos que una serie de archivos y ficheros se han creado. Vamos a proceder a explicar cual es el uso de cada uno.

Ramificación inicial

```
ProyectoSphinxG03/
■■■ build/                    # Carpeta de salida con la documentación generada (HTML, PD
■   ■■■ html/                 # Archivos generados en formato HTML
■   ■■■ doctrees/             # Archivos intermedios usados por Sphinx para construir la
■■■ source/                   # Carpeta que contiene los archivos fuente de la documentac
■   ■■■ _static/              # Recursos estáticos personalizados (CSS, JS, imágenes)
■   ■■■ _templates/           # Plantillas personalizadas para la salida de la documentac
■   ■■■ conf.py               # Archivo principal de configuración de Sphinx
■   ■■■ index.rst             # Archivo raíz que define la estructura de la documentación
■   ■■■ otros_archivos.rst    # Otros archivos de contenido reStructuredText
■■■ Makefile                  # Archivo para construir la documentación desde la línea de
■■■ make.bat                  # Archivo para construir la documentación en sistemas Windo
```

Fichero “/source”

Contiene los archivos fuente de la documentación, como configuraciones, contenidos y recursos personalizados. Dentro crearemos los ficheros con la información necesaria para crear el html.

Archivo “conf.py”

Es el archivo principal de configuración de Sphinx. Aquí se definen aspectos como:

-Extensiones a utilizar.
 \ -Idioma de la documentación.
 \ -Temas visuales.
 \ -Configuraciones para soportar formatos como Markdown o reStructuredText.
 \ -Información extra como autor/es, copyright, release…

Archivo “index.rst”

Archivo principal de la documentación. Funciona como el punto de entrada (similar a un índice) y define la estructura de la documentación. Puede incluir enlaces a otros archivos.

Fichero “/_static”

Carpeta para recursos estáticos, como imágenes, hojas de estilo (CSS) personalizadas o scripts JavaScript.

Fichero “/_templates”

Carpeta para plantillas personalizadas que modifican la apariencia o estructura de la documentación generada.

Fichero “/build”

Se utiliza para almacenar los archivos generados por Sphinx, como la documentación en formato HTML, PDF, LaTeX, etc.

Contiene estas subcarpetas comunes:
 \ **html/**: Archivos HTML generados.
 \ **latex/**: Archivos LaTeX para exportar a PDF.

A la hora de crear el HTML, se creará automaticamente la carpeta html/

Archivo “Makefile”

Es un script utilizado en sistemas Linux/macOS que permite automatizar tareas comunes relacionadas con Sphinx, como generar documentación en diferentes formatos.

Ejemplo de uso:

Generar HTML:

```
make html
```

Generar LaTeX:

```
make latex
```

Archivo «make.bat»

Es un archivo por lotes para sistemas Windows que ofrece las mismas funcionalidades que el Makefile.

Ejemplo de uso en Windows:

Generar HTML:

```
make.bat html
```

Explicación del archivo “conf.py”

¿Qué es el archivo “conf.py”?

El archivo conf.py es la configuración principal para un proyecto de documentación generado con Sphinx. Aquí se definen detalles como la información del proyecto, extensiones habilitadas, rutas, idioma, temas de diseño, entre otros. A continuación, explicaremos en detalle los apartados de este archivo, siempre respaldado del propio archivo que tenemos:

Información sobre el proyecto

```
project = 'Documentacion de T05_DOC_PRUEBAS'
copyright = '2025, Grupo 03'
author = 'Grupo 03'
release = '1'
```

project: Define el nombre del proyecto que se documenta. En este caso, “Documentacion de T05_DOC_PRUEBAS”.
 \ **copyright**: Indica el aviso de derechos de autor que aparecerá en la documentación. Incluye el año y el propietario. Aquí es “2025, Grupo 03”.
 \ **author**: Nombre del autor o grupo responsable de la documentación. Aquí es “Grupo 03”.
 \ **release**: Versión o número de lanzamiento del proyecto documentado. Aquí es “1”

Importes y ruta

El archivo cuenta con diversos importes para el correcto funcionamiento del mismo, y para crear un html estilizado y agradable a la vista; además de establecer una ruta para buscar los modulos que va a usar:

```
import os  # Módulo para gestionar rutas del sistema operativo
import sys  # Módulo para modificar el path de búsqueda de Python
import sphinx_rtd_theme  # Tema para mejorar el diseño visual de la documentación

# os.path.abspath('../') obtiene la ruta absoluta del directorio superior al actual
# sys.path.insert(0, ...) añade esta ruta al inicio de sys.path, que es donde Python busca l
sys.path.insert(0, os.path.abspath('../'))
```

Configuración general

Extensiones

La lista “extensions” contiene módulos adicionales que amplían las capacidades de Sphinx:

```
extensions = [
    "sphinx.ext.autodoc",  # Genera documentación automáticamente desde docstrings
    "sphinx.ext.napoleon",  # Interpreta docstrings en formatos Google y NumPy
    "sphinx.ext.viewcode",  # Agrega enlaces al código fuente desde la documentación
    "sphinx_rtd_theme",  # Tema visual basado en Read the Docs
    "myst_parser",  # Habilita soporte para archivos Markdown
    "sphinx_markdown_builder", #Permite a Sphinx generar salida en formato Markdown para doc
    "sphinx_autodoc_typehints" #se usa para mejorar cómo se muestran las anotaciones en los
]
```

sphinx.ext.autodoc: Genera documentación automáticamente a partir de docstrings del código fuente.
 \ **sphinx.ext.napoleon**: Soporta docstrings en los formatos de estilo Google y NumPy, facilitando su interpretación y conversión a HTML.
 \ **sphinx.ext.viewcode**: Incluye enlaces al código fuente desde la documentación.
 \ **sphinx_rtd_theme**: Integra el tema visual basado en Read the Docs.
 \ **myst_parser**: Permite incluir archivos en formato Markdown (.md) en la documentación.
 \ **sphinx_markdown_builder**: Permite a Sphinx generar salida en formato Markdown para documentación.
 \ **sphinx_autodoc_typehints**: Se usa para mejorar cómo se muestran las anotaciones en los docstrings.

Rutas de plantillas

```
templates_path = ['_templates']
```

templates_path = [“_templates”]: Indica el directorio donde se encuentran las plantillas personalizadas para la documentación.

Exclusiones

```
exclude_patterns = []
```

exclude_patterns = []: Lista de patrones (archivos o directorios) que Sphinx ignorará durante la generación de la documentación.

Idioma

```
language = 'es'
```

language = “es”: Configura el idioma de la documentación a español.

Tipos de archivo fuente

```
source_suffix = {
    '.rst': 'restructuredtext',  # Archivos en formato reStructuredText
    '.md': 'markdown',  # Archivos en formato Markdown
}
```

source_suffix: Mapea las extensiones de los archivos fuente soportados por Sphinx a sus formatos respectivos. Aquí se incluyen:
 \ “.rst”: Archivos en formato reStructuredText.
 \ “.md”: Archivos en formato Markdown (habilitados por myst_parser).

Opciones para salida HTML

Tema visual

```
html_theme = 'sphinx_rtd_theme'
```

html_theme = “sphinx_rtd_theme”: Define el tema para la documentación en formato HTML. En este caso, se utiliza sphinx_rtd_theme, un tema visual basado en el estilo de Read the Docs.

Recursos estáticos

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

html_static_path = [“_static”]: Especifica el directorio donde se almacenan archivos estáticos personalizados, como hojas de estilo (CSS), JavaScript o imágenes. Estos recursos se aplican a la documentación generada.

Extensiones de MyST

Esta configuración activa una extensión específica en MyST-Parser para habilitar las cercas de dos puntos en Markdown.

```
myst_enable_extensions = ["colon_fence",
                          "replacements",
                          "strikethrough",
                          "dollarmath"
                          ]
```

colon_fence: Permite el uso de cercas de dos puntos (:::) para crear bloques personalizados como admoniciones, bloques estilizados o contenido enriquecido. **replacements**: Habilita el uso de variables o sustituciones en Markdown para reutilizar texto fácilmente. **strikethrough**: Permite usar sintaxis de tachado en Markdown. **dollarmath**: Habilita la sintaxis matemática con delimitadores de dólar. Útil para escribir expresiones matemáticas

Heading Anchor

Esta línea define el nivel de encabezados que tendrán anclas automáticas generadas. Las anclas permiten crear enlaces directos a encabezados específicos en la documentación. Esto es útil para facilitar la navegación y compartir enlaces a secciones concretas.

```
myst_heading_anchors = 2
```

Explicación de index.md

¿Qué es?

El archivo **index.md** (o index.rst, dependiendo del formato que uses) es el archivo principal de configuración de contenido de un proyecto Sphinx. Actúa como el punto de entrada para la generación de la documentación y define la estructura básica del proyecto, indicando qué archivos y secciones incluir, cómo organizarlos y cómo se mostrarán en el índice de navegación.

Cómo se crean los índices

El índice en forma de árbol (toctree) organiza los contenidos de la documentación en una estructura jerárquica. Cada sección o subsección se puede definir con esta directiva, permitiendo agrupar documentos relacionados.

Las propiedades principales del toctree son: **:maxdepth:** Define la profundidad del árbol, es decir, cuántos niveles de subtítulos se incluyen. **:caption:** Especifica un título para el árbol. **:numbered:** Numera automáticamente las secciones y subsecciones.

Ejemplo sacado de nuestro propio archivo:

```
.. toctree::
   :maxdepth: 2
```

```
:caption: 1. Configuración Inicial del Proyecto
:numbered:

configuracion_inicial/001.env
configuracion_inicial/002.instalacion_librerias
configuracion_inicial/003.creacion_con_sphinx
...
```

Tipos de índices adicionales

Índice alfabético (genindex)

Este es un índice generado automáticamente que organiza los términos o conceptos documentados en orden alfabético.

Cómo se incluye:

Se genera automáticamente si está habilitado en la configuración de Sphinx. Para incluirlo manualmente en un archivo, puedes usar la directiva:

```
.. include:: genindex
```

O su equivalente en Markdown con MyST:

```
```{include} genindex
```

Uso típico: Es útil para búsquedas rápidas de términos específicos. Los términos se definen usando la directiva … index:: en reStructuredText o {index} en MyST.

Índice temático o búsqueda global (search)

Este índice permite realizar búsquedas en toda la documentación.

Cómo se incluye:

Si está habilitada la extensión de búsqueda (que lo está por defecto en Sphinx), se genera automáticamente. Para incluirlo manualmente, puedes usar:

```
.. include:: search
```

O en Markdown:

```
```{include} search
```

Uso típico: Es accesible a través de la barra de búsqueda en la interfaz HTML generada. Muy útil en proyectos con gran cantidad de contenido.

Índice de referencia cruzada (:ref:)

Este índice no es un índice como tal, pero permite crear enlaces cruzados entre diferentes partes de la documentación.

Cómo se incluye:

Definiendo etiquetas (labels) en las secciones que deseas enlazar. En reStructuredText:

```
.. _mi_seccion:


Título de la sección
====================
Contenido aquí.
```

Y para referenciarlo:

```
Ve a la sección :ref:`mi_seccion`.
```

En MyST:

```
```{label} mi_seccion
Título de la sección
Contenido aquí.
```

Para referenciarlo:

```
Ve a la sección {ref}`mi_seccion`.
```

Uso típico: Facilita la navegación interna en la documentación.

Generacion automatica de documentacion.

Para generar la documentacion de un codigo de manera automática a través de sphinx será necesario los siguientes requisitos:

Requisitos

En el archivo de configuración es necesario que en el apartado de **extensions** de nuestro archivo “conf.py” se encuentren las siguientes extensiones (Pueden haber más, pero las siguientes son las obligatorias):

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

También será necesario instalar **sphinx-markdown-builder**. Este lo instalaremos mediante el siguiente comando:

```
pip install sphinx-markdown-builder
```

y lo añadiremos a el apartado de las extensiones:

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

Como generar la documentacion

Para generar la documentación es necesario ejecutar el comando **sphinx-apidoc** con el que generaremos nuestra documentación. Desde la ruta /docs:

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

Después de ejecutar este comando deberemos ejecutar el siguiente comando situandonos en el fichero /docs para generar los archivos con la extension .md

```
sphinx-build -b markdown source/ build/
```

Archivos Generados

Despues de ejecutar los comandos anteriores para construir la documentacion se generaran archivos Markdown que incluiran:

- index.md: Archivo principal que enlaza con el resto de los documentos
- «nombre del archivo».md: Esta es una documentacion especifica del codigo .py proporcionado el cual incluye descripciones de las funciones, clases y sus atributos.

Cambiar el formato de salida

Se puede cambiar el formato de salida añadiendo esta linea al archivo de configuracion
```

```
myst_enable_extensions = ["colon_fence"]
```

Generacion de HTML a partir de la Documentacion

Para generar la documentación en formato HTML a partir de los archivos **.rst** generados por **sphinx-apidoc** hay que hacer lo siguiente pero primero una pequeña explicacion de que archivos se usuaran y que Script se va a ejecutar.

Archivos que se usaran para generar la documentacion

Sphinx tomará los archivos .rst generados anteriormente, que contienen la documentación de tu código Python, y los procesará para generar la salida en HTML. Estos archivos se encuentran en el directorio docs/source/.

Que script se ejecutara

Para generar el archivo HTML a partir de los archivos **.rst** se utiliza el siguiente comando

```
sphinx-build -b html docs/source/ docs/build/
```

Resultado generado

Después de ejecutar el comando anterior, Sphinx generará los archivos HTML en el directorio especificado (docs/build/). Estos archivos incluirán la documentación de el código, y podremos visualizarlos en un navegador web.

Generar PDF

Para generar el PDF a partir de los HTML usaremos la herramienta **rst2pdf**. Este es el método mas sencillo de usar y configurar.

Intalar rst2pdf

Con el siguiente comando instalaremos **rst2pdf**.

```
pip install rst2pdf
```

Configurar rst2pdf para que funcione

Para configurar **rst2pdf** solo tendremos que introducirlo dentro de nuestro archivo **conf.py** añadiremos dentro de las extensiones la siguiente **rst2pdf.pdfbuilder**.

```
extensions = ['rst2pdf.pdfbuilder']
```

Comando para generar PDF

```
sphinx-build -b pdf docs/source docs/build/pdf
```

Y con este comando finalmente generaremos un PDF que tendrá el mismo contenido que los HTML que estará ubicado dentro de la carpeta “build/pdf”.

Estructura final del proyecto

Tras usar y ejecutar ciertos comandos a lo largo de la guia y el proyecto, se puede observar un cambio en la estructura del mismo

Descripcion de los Componentes de la estructura

Docs

Esta es la carpeta principal donde se gestionan los archivos relacionados con la documentación.

Build

Contiene los archivos finales generados por Sphinx. Dependiendo del formato de salida (html, markdown, latex, etc.), los archivos aquí variarán entre:

- HTML: Archivos navegables en un navegador web
- Marckdown: Archivos legibles y procesables por sistemas como GitHub
- PDF: Archivos de texto que solo apareceran si se configuro para latex que en nuestro caso habra archivos.

Source

Contiene los archivos fuente utilizados por Sphinx para construir la documentación.

- conf.py: Archivo de configuracion principal. Define extensiones, rutas, formatos de salida, etc.
- index.rst: Archivo indice inicial que enlaza con el resto de la documentacion.
- modules.rst: Generado automaticamente por sphinx-apidoc. Lista los modulos y funciones del proyecto.
- static: Archivos como imagenes, CSS o JavaScript que se incluyen en la documentacion.
- templates: Plantillas personalizadas para modificar el diseño de la documentacion.
- Makefile y make.bat: Archivos para contruir la documentacion:

— En Linux/Mac make html , make markdown, etc.

— En Windows make.bat html, make.bat markdown, etc.

Explicación de Comandos Usados en Sphinx

Crear un Entorno Virtual

```
python -m venv env
```

Este comando crea un entorno virtual llamado env. Un entorno virtual es un espacio aislado para instalar las dependencias de Python, evitando conflictos con otras versiones de librerías o proyectos

Activar el entorno Virtual

```
.\env\Scripts\activate
```

Este comando activa el entorno virtual creado previamente. Una vez activado, cualquier paquete que instales usando pip se instalará solo dentro de ese entorno.

Comandos para las Dependencias de Sphinx

Usando el archivo **requirements.txt**:

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

Manualmente:

```
pip install -U sphinx
```

```
pip install sphinx_rtd_theme
```

```
pip install myst-parser
```

```
pip install sphinxcontrib-napoleon
```

```
pip install sphinx-autobuild
```

```
pip install sphinx_markdown_builder
```

```
pip install sphinx-autodoc-typehints
```

1. **sphinx**: La herramienta principal para generar documentación.
2. **sphinx-rtd-theme**: Un tema profesional para la documentación, utilizado comúnmente en proyectos de documentación en línea.
3. **sphinx-autobuild**: Permite previsualizar la documentación generada de manera automática a medida que realizas cambios.
4. **sphinxcontrib-napoleon**: Habilita el soporte para los formatos de docstring de Google y NumPy.
5. **myst-parser**: Permite usar archivos Markdown (.md) en lugar de reStructuredText (.rst) en Sphinx.
6. **sphinx_markdown_builder**: Permite a Sphinx generar salida en formato Markdown para documentación
7. **sphinx-autodoc-typehints**: Se usa para mejorar cómo se muestran las anotaciones en los docstrings

Inicializar un Proyecto Sphinx

```
sphinx-quickstart
```

Este comando inicializa un proyecto de documentación Sphinx, generando una estructura básica de archivos y directorios que contiene:

1. **conf.py**: Archivo de configuración.
2. **index.rst o index.md**: El archivo principal de documentación.

Al ejecutarlo, el comando te hará algunas preguntas para configurar el proyecto (por ejemplo, el nombre del proyecto, la versión, etc.).

Generar Documentacion Automatica desde el Codigo

```
sphinx-apidoc -o source/
```

Este comando genera automáticamente la documentación de los módulos Python presentes en el directorio actual. El resultado se guarda en el directorio source/. Los archivos generados incluyen la documentación en formato reStructuredText (.rst) para cada archivo Python.

Generar la Documentacion en HTML

```
.\docs\make.bat html
```

Este comando construye la documentación en formato HTML. Los archivos generados se encuentran en el directorio build/html/. Puedes abrir index.html en tu navegador para ver el resultado.

Generar la Documentacion en PDF

```
sphinx-build -b pdf docs/source docs/build/pdf
```

El comando «sphinx-build -b pdf» genera la documentación en formato PDF a partir de la documentación fuente.

Generar documentacion de manera automatica

```
sphinx-build -b markdown source/ build/
```

Este comando hace que genere el .md . Será generado gracias al sphinx-apidoc debido a la opcion -b, que hará que lo genere como .md en vez de .srt.

Typography

Que es Typography

“Typography” es una funcion de myST que nos permite controlar los aspectos tipográficos del texto. Su principal uso es generar documentos profesionales bien esctructurados.

Ahora pasaremos a explicar como usarlo y las funciones basicas.

Formateo de texto

Para remarcar una palabra en negrita la rodearemos con con dos *

Ejemplo

```
**Negrita**
```

Salida: Negrita

Para remarcar una palabra en cursiva la rodearemos con un solo *

Ejemplo

```
*Cursiva*
```

Salida *Cursiva*

Comillas tipograficas

MyST-Parser convierte automáticamente comillas rectas en comillas tipográficas:

Ejemplo

```
"Comillas Rectas"
```

Salida «Comillas rectas transformadas»

Ejemplo

```
'Comillas simples'
```

Salida

“Comillas simples transformadas”

Este comportamiento puede ser desactivado desde el archivo de configuracion de Sphinx añadiendo la siguiente linea

```
myst_enable_extensions = [
    "smartquotes",
]
```

Guiones y rayas

MyST parser soporta

- → solo un guión se transforma en punto.

– → dos guiones seran un guión largo o mediano.

— → tres guiones seran un guión largo.

Ejemplo

```
El intervalo de tiempo es de 2--4 días --- esto es importante.
```

Salida

El intervalo de tiempo es de 2–4 días — esto es importante.

Control de Espacios o Guionado

MyST parser también tiene funciones para controlar el espacio o los Guionados.

Espacios inseparables

Usaremos \ para generar un espacio Inseparable

Ejemplo

```
Esto es un espacio inseparable\!
```

Salida

Esto es un espacio inseparable!

Admonitions

Extension necesaria

Para que las “admonitions” funcionen es necesario que dentro del archivo de configuracion esté la siguiente línea:

```
myst_enable_extensions = [
    "colon_fence",

]
```

Que son los admonitions

Son bloques especiales que se pueden incluir en la documentación para destacar contenido con distintos propositos- Estos bloques son:

- Notas
- Advertencias
- Consejos
- Errores
- Peligros

Estos nos permiten mejorar la legibilidad de la documentación y atraer la atención hacia un contenido específico.

Como usarlas

En MyST, las admoniciones se crean con la siguiente secuencia

```
:::{admonition-type}:::

siendo un ejemplo

:::{note}:::
```

Tipos de admoniciones

MyST cuenta con diferentes tipos de admoniciones ya predefinidas las que podremos usar directamente sin especificar el tipo, estas son las siguientes:

Sintaxis basica

Para declarar una admonicion tendremos que usar los «:» y escribir entre «{}» el tipo de la admonicion.

Nota

para declarar una admonicion de tipo nota escribiremos «note» entre corchetes

Ejemplo

```
:::{note}

Esto es una nota
:::
```

Salida

> **Note**
>
> Esto es una nota

Advertencia

Para escribir una admonicion de tipo advertencia escribiremos «warning» entre los corchetes

Ejemplo

```
:::{warning}

Esto es una advertencia
:::
```

Salida

> **Warning**
>
> Esto es una advertencia

Peligro

Para escribir una admonicion de tipo peligro escribiremos «danger» entre los corchetes

Ejemplo

```
:::{danger}

Esto es peligroso
:::
```

Salida

> **!DANGER!**
>
> Esto es peligroso

Consejo

Para escribir una admonicion de tipo consejo escribiremos «tip» entre los corchetes

Ejemplo

```
:::{tip}

Esto es un consejo
:::
```

Salida

> **Tip**
>
> Esto es un consejo

Error

Para escribir una admonicion de tipo error escribiremos «error» entre los corchetes

Ejemplo

```
:::{error}

Esto es un error
:::
```

Salida

> **Error**
>
> Esto es un error

Admoniciones personalizadas

Para declarar una admonicion personalizada escribiremos admonition entre corchetes y seguidamente el titulo personalizado fuera de estos:

Ejemplo

```
:::{admonition}Titulo Personalizado

Esto es una admonicion personalizada
:::
```

Salida

> **Titulo Personalizado**
>
> Esto es una admonicion personalizada

Insertar imagen en sphinx

Para insertar una imágen tenemos dos maneras:

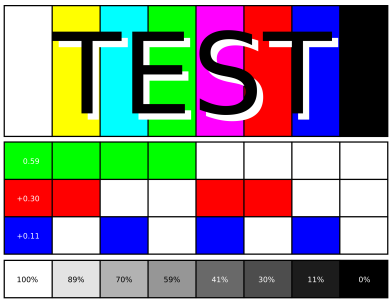
Imágen básica con markdown

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

Imágen avanzada con MyST

```
```{image} ruta\\a\\la\\imagen.png
:alt: Texto alternativo para la imagen
:width: Anchura (en px)
```
```

Imagen insertada



Insertar tablas en sphinx

Para insertar una tabla tenemos dos maneras:

Tabla básica con markdown

Para hacer una tábla básica con markdown podemos usar este código especificando directamente sobre las cajas los encabezados y valores de la tabla:

```
| Encabezado 1 | Encabezado 2 | Encabezado 3 |
|--------------|--------------|--------------|
| Valor 1      | Valor 2      | Valor 3      |
| Otro 1       | Otro 2       | Otro 3       |
```

Tabla avanzada con MyST

Para insertar una tabla avanzada en la que se puedan especificar parámetros como la anchura de cada columna podemos usar MyST de la siguiente forma:

```
    ```{list-table} Ejemplo de tabla con MyST
       :header-rows: 1
       :widths: 30 30 40

       * - Encabezado 1
         - Encabezado 2
         - Encabezado 3
       * - Fila 1, Columna 1
         - Fila 1, Columna 2
         - Fila 1, Columna 3
       * - Fila 2, Columna 1
         - Fila 2, Columna 2
         - Fila 2, Columna 3
```

Tabla creada

Ejemplo de tabla con MyST

Encabezado 1	Encabezado 2	Encabezado 3
Fila 1, Columna 1	Fila 1, Columna 2	Fila 1, Columna 3
Fila 2, Columna 1	Fila 2, Columna 2	Fila 2, Columna 3
```

Insertar codigo fuente y APIs en sphinx

Representar códgio fuente

Representar código de forma general

Si queremos documentar un trozo de código sin resaltar ninguna sintaxis específica del lenguaje utilizado podemos hacerlo de la siguiente forma:

````
```bash
# Ejemplo de código en Python
def saludar(nombre):
    return f"Hola, {nombre}!"
    ```
````

Este ejemplo quedaría representado de esta forma:

```
# Ejemplo de código en Python
def saludar(nombre):
    return f"Hola, {nombre}!"
```

Representar código con una sintaxis específica

Si queremos documentar un trozo de código marcando con colores la sintaxis del lenguaje utilizado podemos especificar después de las comillas el lenguaje al que nos estamos refiriendo:

````
```python
# Ejemplo de código en Python
def saludar(nombre):
    return f"Hola, {nombre}!"
    ```
````

Este ejemplo quedaría representado de esta forma:

```python
# Ejemplo de código en Python
def saludar(nombre):
    return f"Hola, {nombre}!"
```

Documentar una API con sphinx

Instalar Autodoc-Typehints

Primero, debemos instalar la dependecia sphinx-autodoc-typehints en nuestro entorno ejecutando el siguiente comando:

```
pip install sphinx-autodoc-typehints
```

Implementar la dependencia dentro de nuestro proyecto

Ahora, debemos implementar la dependencia que acabamos de instalar dentro de nuestro archivo «conf.py», más concretamente dentro del arreglo «extensions»:

```
# Lista de extensiones activadas en Sphinx
extensions = [
    'sphinx.ext.autodoc',  # Genera documentación automáticamente desde docstrings
    'sphinx.ext.napoleon',  # Interpreta docstrings en formatos Google y NumPy
    'sphinx.ext.viewcode',  # Agrega enlaces al código fuente desde la documentación
    'sphinx_rtd_theme',  # Tema visual basado en Read the Docs
    'myst_parser',  # Habilita soporte para archivos Markdown
    "sphinx_autodoc_typehints"
]
```

Comentar de forma correcta el código

Ahora, comentamos de forma correcta el codigo de nuestro proyecto:

```python
def saludar(nombre: str) -> str:
    """
    Saluda a un usuario.

    Args:
        nombre (str): Nombre del usuario.

    Returns:
        str: Mensaje de saludo personalizado.
    """
    return f"Hola, {nombre}!"
```

Generar el archivo de documentacion

Para generar el archivo de documentacion de nuestro codigo usaremos «sphinx-apidoc»:

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

Esto generará archivos `.rst` en la carpeta `docs/source/`

Referencias cruzadas con MyST Parser

MyST-Parser ofrece caracteristicas de referenciado cruzado, para enlazarle URLs, documentos, cabeceras y más que generan «warnings» cuando se rompen.

Crear objetivos explícitos

Los objetivos son anclas personalizadas a las que puedes referirte en otras partes de tu documentación

Hay tres formas principales de crear objetivos:

```
1. Anotar un bloque de sintaxis con `(target)=`
2. Anotar un bloque de sintaxis con un atributo `{#id}`
3. Añadir la opción `nombre` a una directiva

(heading-target)=
### Cabecera

{#paragraph-target}
Esto es un parrafo, con un atributo `id`.

Esto es un [con un atributo `id`]{#span-target}.

:::{note}
:name: directive-target

Esto es una directiva con la opción `nombre`.
:::

[reference1](#heading-target), [reference2](#paragraph-target),
[reference3](#span-target), [reference4](#directive-target)
```

Objetivos implícitos

También puedes referenciar de forma directa a los objetivos a los que quieres apuntar, para ello hay dos formas:

Que el titulo del enlace sea directamente lo que está referenciando

```
<project:#objetivos-implícitos>
```

Objetivos implícitos

O que el titulo del enlace sea personalizado

```
[Objetivos Implicitos](#objetivos-implícitos)
```

Objetivos Implicitos

Organizar contenido en Sphinx

Sphinx nos permite organizar el contenido en distintos documentos, así como incluir contenido de otros documentos.

Estructura de documento

Los documentos de Sphinx están organizados mediante cabeceras.

Todas las cabeceras al nivel de la raiz del documento se incluyen en la Tabla de Contenidos de esa página.

Muchos temas ya incluyen esta tabla de contenidos en una barra lateral, pero también podemos incluirla en el contenido principal de la página con la directiva «contents».

````
```{contents} Table of Contents
:depth: 3
```
````

Insertar documentos dentro del contenido de otro documento

La directiva «include» nos permite insertar el contenido de otro documento directamente al flujo del documento actual.

````
```{include} nombre_documento.extension
    :start-line: linea en la que quieres empezar a insertar
    :end-line: linea en la que quieres parar de mostrar
```
````

De otra forma, si lo que quieres es insertar un bloque de codigo de otro documento, usamos la directiva «literalinclude».

````
```{literalinclude} nombre_documento.extension
```
````

Organizar documentos con toctree

Si lo que queremos es que algunos documentos sean hijos de otros, podemos usar la directiva «toctree», indicando dentro de esta los documentos que queremos hacer hijos.

````
```{toctree}
examples/content_child1.md
examples/content_child2.md
```
````

Extensiones de Sintaxis

Las extensiones que queramos añadir a nuestra documentacion las debemos añadir a un arreglo de Strings dentro de `conf.py` llamado `myst_enable_extensions`

```
myst_enable_extensions = [
    "replacements",
    "strikethrough",
    "dollarmath"
]
```

Replacements

La extension `Replacements` nos permite reemplazar ciertos caracteres entre paréntesis por ciertos caracteres especiales, como por ejemplo:

```
- (c)
- (tm)
- (r)
```

- ©
- ™
- ®

Strikethrough

La extension `Strikethrough` nos permite tachar un texto dentro de nuestro documento poniendolo entre ~~

```
~~Este texto está tachado~~
```

<s>Este texto está tachado</s>

Dollarmath

La extension `dollarmath` nos permite expresar funciones matemáticas introduciendolas entre $

```
$x_{hey}=it+is^{math}$
```

$x_{hey} = it + is^{math}$

Otras extensiones

En MyST Parser existen muchos tipos de extensiones de sintaxis que nos permiten hacer nuestros documentos más ricos en contenido, algunas de las extensiones que no hemos mencionado son:

```
myst_enable_extensions = [
    "amsmath",
    "attrs_inline",
    "colon_fence",
    "deflist",
    "fieldlist",
    "html_admonition",
    "html_image",
    "linkify",
    "smartquotes",
    "substitution",
    "tasklist",
]
```

Roles y Directivas

Roles

En MyST Parser los roles se usan para indicarle a un bloque de código cómo comportarse, por ejemplo, para introducir una foto dentro de nuestra documentación utilizamos el rol «image» entre llaves:

```
    ```{image} ruta\\a\\la\\imagen.png
```

Directivas

Las directivas son parámetros opcionales que se incluyen dentro de los roles para indicarles como tienen que comportarse, se especifican entre :
```

```
```{image} ruta\\a\\la\\imagen.png
 :alt: Texto alternativo para la imagen
 :width: Anchura (en px)
```
```