*Unidad 5: Documentación de Aplicaciones*

*Profesora Ofelia De Lima Linhares*

---

**Índice de contenidos**



**Objetivos de aprendizaje**

**1.** Documentar la estructura de la información persistente.

**2.** Confeccionar manuales de usuario, guías de referencia y tutoriales.

**3**. Crear y mantener documentación de proyectos de manera efectiva.

## 1. Markdown

Es de suma importancia documentar durante todo el ciclo de vida del proyecto. La falta de documentación es un problema recurrente que provoca desamparo ante situaciones imprevistas. En los últimos tiempos está habiendo un cambio de tendencia, puesto que van apareciendo tecnologías que ayudan a elaborar toda la documentación asociada a los proyectos.

Esta documentación debe incluir tanto aspectos básicos para el usuario final como aspectos más técnicos relacionados con su implementación o puesta en producción, de modo que se transfiera fluidamente el conocimiento necesario del proyecto desde el equipo de desarrollo a otros miembros del equipo, a otros equipos o al cliente final.

### 1.1 Introducción

Cuando se trabaja con diferentes procesadores de texto, aunque cada vez son menos, pueden aparecer problemas de compatibilidad o de presentación entre los formatos.

Los procesadores de texto actuales, como LibreOffice o MS Office, siguen un paradigma para la creación de documentos llamado **WYSIWYG "What You See Is What You Get" (lo que ves es lo que obtienes)**. Este paradigma se aplica tanto a procesadores de texto como a otros tipos de editores, como pueden ser algunos editores web, y se basa en la posibilidad de poder escribir el documento viendo directamente el resultado final pero ocultando la representación interna de los distintos elementos. Por el contrario, este hecho hace que se genere mucha información de la necesaria a la hora de representar internamente los documentos, y estos acaban ocupando mucho más de lo que se necesitaría.

Otro paradigma aunque menos conocido es el **WYSIWYM** "What **You See Is What You Mean" (lo que ves es lo que quieres decir)**, según el cual, el usuario se encarga de introducir los contenidos de forma estructurada, según su significado, en lugar de indicar el formato de representación final. La idea es indicar si lo que se está escribiendo es un título, una sección, el nombre del autor, etc. en lugar de indicar su formato. El propio procesador u otras herramientas externas tendrán que ser quienes generen el documento en su formato final, coherente con el texto y la estructura especificadas.

La ventaja de estos sistemas es que se hacen manifiesta la separación entre contenido y presentación, por lo que solo debemos preocuparnos de la estructura y los contenidos, dejando aparte los aspectos visuales para que se encargue de ellos el propio sistema de exportación.

**Markdown** es un lenguaje que sigue este paradigma, y servirá para generar documentos que permitan centrarnos más en el contenido y no estar pendientes del aspecto final.

### 1.2 Lenguaje Markdown

**Markdown** se define como un lenguaje de marcas ligero con el fin de escribir utilizando un formato de texto plano, fácil de escribir y leer, y convertible a otros formatos como HTML EPUB o PDF.

> **Lenguaje de marcas**
>
> Un lenguaje de marcas o lenguaje de marcado se entiende como una forma de codificación de documentos, que, junto al texto, incorpora etiquetas o marcas con información adicional sobre la estructura del texto o el formato de presentación. Los lenguajes de marcas más conocidos son HTML (HyperText Markup Language) y XML (Extensible Markup Language).
>
> Cuando hablamos de un lenguaje de marcas ligero, hacemos referencia a lenguajes de marcas que utilizan un formato de texto más o menos estandarizado, que ocupa poco espacio y que es fácil de leer y escribir.

El lenguaje **Markdown** fue creado por John Gruber con el propósito de conseguir la máxima legibilidad y facilidad de publicación, y se inspiró en muchas convenciones existentes para el marcado de mensajes de correo electrónico utilizando texto plano. Según John Gruber un documento con formato **Markdown** debería ser publicable tal cual, como texto plano, sin que parezca que se ha marcado con etiquetas o instrucciones de formato.

Actualmente **Markdown** se utiliza en muchos ámbitos especialmente en la generación de documentación de proyectos y de contenido web en sitios como **Github o Read the Doc**s, o en sistemas de gestión de contenidos como Jekyll, Hugo, **Gitbook**, la plataforma Moodle o el procesador de textos **Google Docs**. Para la generación de documentación en PDF, existen herramientas como **Pandoc,** que hacen uso del potente sistema de composición de textos **LaTeX** para su formato.

Con **Markdown** se puede formatear el texto con letras cursivas y negritas con cabeceras o con enlaces utilizando únicamente texto plano. Esto hace que la escritura sea más simple y eficiente, ya que, evita tener que pensar en el formato y permite centrarnos solo en el contenido.

### 1.3. Sintaxis del Lenguaje

**Markdown** se basa en archivos de tipo texto, que no contienen ninguna información interna sobre el formato punto. Esta información se especificará de forma explícita mediante etiquetas, que se verán visibles en todo momento y que facilitarán por un lado su interpretación a la hora de exportarlos a otro formato, y, por otro lado, su lectura.

En este apartado se verán cuáles son los distintos elementos que se pueden utilizar en un texto en formato **Markdown,** así como las principales marcas de formato.

**Sintaxis básica:**

- Encabezados
- Énfasis (negrita y cursiva)
- Listas (ordenadas y desordenadas)
- Enlaces e imágenes
- Tablas
- Bloques de código y citas

**Ejemplos prácticos:**

- Crear un archivo README.md para un proyecto
- Formatear un artículo de blog

**La extensión de los archivos Markdown es .md**

## A.PÁRRAFOS

En Markdown un párrafo es un bloque de texto definido entre dos saltos de línea. Se crea un nuevo párrafo cuando hay una línea en blanco entre dos líneas de texto. Si utilizamos solo un salto de línea, se sobrentiende que es el mismo párrafo y se presentará como tal. En cambio, si queremos añadir solo un salto de línea a mitad de un párrafo, añadiremos al menos dos espacios al final de la línea y un solo salto de línea.

Ejemplo:

```
Párrafo 1

Párrafo 2

Párrafo 3
Párrafo 3
```

## B. ENCABEZADOS O TÍTULOS

Utilizamos el símbolo # antes del texto para indicar el nivel de la cabecera. Se admiten hasta seis niveles de profundidad (######), lo que vendría a ser del h1 hasta el h6 de HTML.

Aunque el lenguaje Markdown no exige un espacio entre # y el título, ni un salto de línea antes de este, se recomienda ponerlo tanto para facilitar la lectura como por compatibilidad con el resto de variantes de Markdown que sí que lo exigen.

```
# Encabezado de primer nivel

## Encabezado de segundo nivel

### Encabezado de tercer nivel

...

###### Encabezado de nivel 6
```

## C. FORMATO DE TEXTO

Markdown nos permite utilizar el símbolo del asterisco como marca de formato para indicar textos en cursiva, en negrita o ambos. Para ello escribiremos uno, dos o tres asteriscos al principio y al final del texto que queramos marcar. Además, debemos tener en cuenta que no se añade espacio entre los asteriscos del principio y la primera palabra ni entre los asteriscos del final y la última palabra.

```
Con un asterisco marcamos un *texto en cursiva*
Con dos asteriscos marcamos un **texto en negrita**
Con tres asteriscos marcamos un ***texto en cursiva y en negrita***
```

## D.LÍNEAS HORIZONTALES

Una línea horizontal se define con al menos tres símbolos: *, - o _. Pueden estar separados por espacios si se quiere.

```
---

* * *

_ _ _
```

## E. LISTAS

Markdown permite el uso tanto de listas ordenadas como de listas no ordenadas.

### Listas no ordenadas

Las listas no ordenadas se marcan utilizando los símbolos *, + o - al principio de cada elemento, y escriben cada ítem en una línea diferente (no se necesita el salto de línea esta vez).

- Elemento 1
- Elemento 2

Cuando queramos incluir varios párrafos en un ítem de la lista, utilizaremos un nuevo párrafo y lo alinearemos con el contenido que empieza después del espacio del marcador de lista.

```
- Elemento 1

  Párrafo dentro de elemento 1

- Elemento 2
```

Una lista puede contener otras listas. Para utilizar una lista dentro de otra, solo deberemos sangrar la lista que está dentro de la otra y alinearla con el primer carácter de texto del elemento que la contiene. Podemos tener tantas listas dentro de otras como queramos.

```
* Elemento 1
  * subelemento 1.1
    * subelemento 1.1.1
    * subelemento 1.1.2
  * subelemento 1.2
  * subelemento 1.3
* Elemento 2
```

En estos casos, dado que podemos utilizar varios símbolos para indicar listas, se suele utilizar un elemento por cada nivel de la lista, con el fin de facilitar la lectura del texto plano.

```
* Elemento 1
  + subelemento 1.1
    - subelemento 1.1.1
    - subelemento 1.1.2
  + subelemento 1.2
  + subelemento 1.3
* Elemento 2
```

### Listas ordenadas

El funcionamiento de las listas ordenadas es el mismo que el de las no ordenadas, salvo que cada elemento de la lista lleva un número.

```
1. Elemento 1
  1. Elemento 1.1
    1. Elemento 1.1.1
    2. Elemento 1.1.2
  2. Elemento 1.2
  3. Elemento 1.3
2. Elemento 2
```

## F. TABLAS

Las tablas sirven para presentar información de forma organizada.

La sintaxis para crear tablas del Markdown de GitHub es una de las más extendidas, y hace uso de barras verticales (|) y guiones (-) para crear las tablas. Los guiones se utilizan para crear el encabezamiento de cada columna, y las barras verticales sirven de separador de cada columna.

En este formato las tablas deben tener necesariamente una cabecera y un cuerpo, y seguirán la sintaxis siguiente:

```
| Cabecera 1 | Cabecera 2 |
| ---------- | ---------- |
| Valor 1    | Valor 2    |
| Valor 3    | Valor 4    |
```

La línea que separa la cabecera del cuerpo |---|---| es obligatoria, pero no es necesario que tenga tantos caracteres como tengan las cabeceras, bastará con tres.

### Formato del contenido de una tabla

Dentro de una tabla podemos utilizar también ciertas marcas de formato, como negritas, cursivas, enlaces, imágenes...

Asimismo, podemos alinear el texto a la izquierda, a la derecha o centrarlo en la columna añadiendo la marca dos puntos al lado izquierdo, al derecho o a los dos lados de los guiones del encabezamiento.

```
| Texto a la izquierda | Texto centrado | Texto a la derecha |
| :------------------- | :------------: | ------------------: |
| xxx                  |      xxx       |                 xxx |
| xxxxx                |     xxxxx      |               xxxxx |
```

## G.FRAGMENTOS DE CÓDIGO

Markdown tiene un amplio uso en la documentación técnica de proyectos informáticos, en los que es habitual incluir fragmentos del código fuente de los programas. Para resaltar este tipo de fragmentos, Markdown utiliza una sintaxis especial con el acento abierto como marca.

Cuando se trata de fragmentos de código que deben ir en la misma línea que el texto (inline), por ejemplo, si queremos indicar una etiqueta HTML, lo hacemos `de este modo`, haciendo uso de un único carácter de acento, mientras que si queremos escribir un bloque de código utilizaremos tres símbolos de acento abierto. Además, detrás de los primeros símbolos podemos especificar de qué lenguaje se trata; así se puede presentar de forma más clara resaltando la sintaxis propia del lenguaje indicado. Por ejemplo, para indicar un bloque de código Python, escribiríamos:

```py
from PySide6.QtWidgets import QApplication, QWidget
import sys
app = QApplication(sys.argv)
window = QWidget()
window.show()
app.exec_()
```

### H.CITAS

En Markdown un bloque de texto en forma de cita consiste en uno o más párrafos u otros elementos, como listas, donde cada línea se encuentra precedida del carácter mayor que (>) y opcionalmente de un espacio. Se pueden anidar citas.

```
> Párrafo de la cita
>
> > Cita anidada
```

### I. ENLACES

Markdown permite tanto generar enlaces a direcciones de Internet como hacer referencia a archivos locales, mediante su ruta relativa o incluso dentro del propio documento.

El formato general para añadir un enlace es el siguiente:

```
[Texto del enlace](URL_o_dirección_relativa)
```

Por ejemplo, para añadir un enlace a un sitio web, escribiremos:

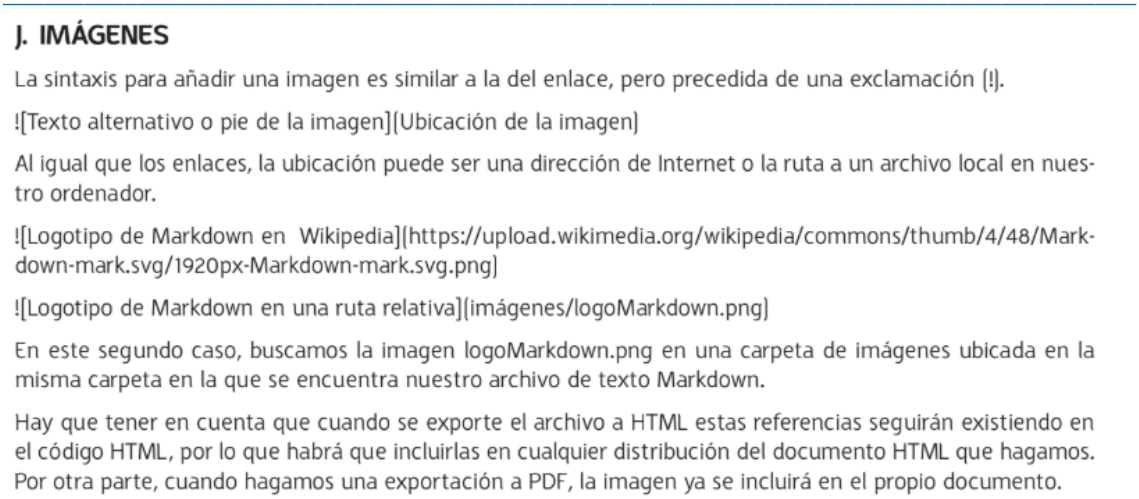
```
[Python](https://www.python.org/)
```

Para añadir un enlace a una sección de nuestro documento, haremos uso del identificador que se asigna automáticamente coincidiendo con el título de la cabecera, o que le hemos asignado nosotros. Por ejemplo, si para el apartado introductorio añadimos un identificador de la forma siguiente:

```
# Introducción
```

Podemos hacer referencia a él como sigue:

```
[Volver a la introducción](#introducción)
```



J. IMÁGENES

La sintaxis para añadir una imagen es similar a la del enlace, pero precedida de una exclamación (!).

![Texto alternativo o pie de la imagen](Ubicación de la imagen)

Al igual que los enlaces, la ubicación puede ser una dirección de Internet o la ruta a un archivo local en nuestro ordenador.

![Logotipo de Markdown en Wikipedia](https://upload.wikimedia.org/wikipedia/commons/thumb/4/48/Markdown-mark.svg/1920px-Markdown-mark.svg.png)

![Logotipo de Markdown en una ruta relativa](imágenes/logoMarkdown.png)

En este segundo caso, buscamos la imagen logoMarkdown.png en una carpeta de imágenes ubicada en la misma carpeta en la que se encuentra nuestro archivo de texto Markdown.

Hay que tener en cuenta que cuando se exporte el archivo a HTML estas referencias seguirán existiendo en el código HTML, por lo que habrá que incluirlas en cualquier distribución del documento HTML que hagamos. Por otra parte, cuando hagamos una exportación a PDF, la imagen ya se incluirá en el propio documento.

Markdown Guide

## 2. Lenguajes para ficheros de documentación

La informática es una rama en constante evolución. Cada cierto tiempo aparecen nuevas soluciones a problemas que van surgiendo o que optimizan de alguna forma las soluciones anteriormente utilizadas. Es el caso de los lenguajes utilizados para la serialización de datos y los archivos de configuración que se estudiaran en este apartado. Estos son ampliamente utilizados en el desarrollo de proyectos, ya sea para la configuración de los sistemas, la configuración de procesos de integración o de despliegue continuos, o la documentación de renderizado de documentos.

### 2.1 Introducción XML y JSON

En Informática, la serialización de datos (serialization o marshaling) se utiliza para replicar estructuras de datos o estado de objetos entre diferentes sistemas, de forma que se puedan almacenar en un fichero o un buffer de datos o transmitir entre ellos por la red, por ejemplo, y, posteriormente, poder ser recompuestos para que se obtenga una copia exacta del original, un clon.

Los usos más comunes de esta técnica son la transmisión de mensajes, la transmisión de datos entre base de datos, la transmisión de archivos y la ejecución de aplicaciones distribuidas entre diferentes sistemas.

El formato más usado para la serialización de datos en los 90 fue el **XML**. Este fue ampliamente utilizado para la comunicación entre cliente y servidor en aplicaciones web **Ajax (Asynchronous JavaScript and XML).**

```
<mensaje>
   <de>Linus</de>
   <para>Bill</para>
   <asunto>Recordatorio</asunto>
   <cuerpo>No olvides nuestra reunión!</cuerpo>
</mensaje>
```

A principios de siglo apareció el formato **JSON JavaScript object notation** que ha ido sustituyendo paulatinamente a **XML**, y hoy en día es el formato más habitual en las aplicaciones cliente - servidor y soportado por la mayoría de los lenguajes de programación.

```
{
   "mensaje": {
       "de": "Linus",
       "para": "Bill",
       "asunto": "Recordatorio",
       "cuerpo": "No olvides nuestra reunión!"
   }
}
```

### 2.2. El lenguaje YAML

Posteriormente a **JSON** apareció **YAML**, que en realidad es un superconjunto de **JSON** con características adicionales, como el soporte para estructuras de datos no jerárquicas. Puede utilizarse con el estilo Python indentado o de una forma más compacta utilizando [...] para las listas y {...} para mapas. Así pues, un JSON no deja de ser un YAML valido.

**YAML** significa **Yet Another Markup Language (‘otro lenguaje de marcas’),** porque se lanzó en una era en la que proliferaba los lenguajes de marcas. Pero luego se reutilizó como **YAML Ain’t Markup Language (YAML no es un lenguaje de marcado’),** un acrónimo recursivo para distinguir su propósito como orientado a datos en lugar de marcado de documentos.

Su principal uso actualmente es la definición de configuraciones en ficheros de configuración. Por ejemplo, las acciones de **GitHub Actions** o la configuración de red en **Ubuntu** a través de **Netplan**.

```yaml
# This workflow will install Python dependencies, run tests and lint with
a single version of Python
# For more information see: https://help.github.com/actions/language-and-
framework-guides/using-python-with-github-actions

name: Integración continua sobre el módulo es_primo

on:
 push:
   branches: [ main ]
 pull_request:
   branches: [ main ]

permissions:
 contents: read

jobs:
 build:

   runs-on: ubuntu-latest
```

```
    steps:
    - uses: actions/checkout@v3
    - name: Configurar Python 3.10
      uses: actions/setup-python@v3
      with:
        python-version: "3.10"
    - name: Instalar dependencias
      run: |
        python -m pip install --upgrade pip
        pip install pylint pytest
        # if [ -f requirements.txt ]; then pip install -r requirements.txt;
fi
    - name: Comprobación de código con pylint
      run: |
        # falla la integración si la puntuación (score) del linting es
menor al umbral indicado
        pylint --fail-under=10 src

    - name: Pruebas con pytest
      run: |
        pytest
```

### 2.3. El lenguaje TOML

**TOML (acrónimo de Tom's Obvious Minimal Language)**, es un lenguaje para archivos de configuración similar a **YAML**. Diseñado para ser fácilmente leído, tiene una semántica mínima. Está inspirado en la sintaxis de los archivos .INI, una extensión para archivos de configuración de aplicaciones sobre todo en el sistema operativo Windows.

TOML

```toml
title = "TOML Example"
[owner]
name = "Tom Preston-Werner"
dob = 1979-05-27T07:32:00-08:00
[database]
server = "192.168.1.1"
ports = [ 8001, 8001, 8002 ]
connection_max = 5000
enabled = true
[servers]
 [servers.alpha]
 ip = "10.0.0.1"
 dc = "eqdc10"

 [servers.beta]
 ip = "10.0.0.2"
 dc = "eqdc10"

[clients]
data = [ ["gamma", "delta"], [1, 2] ]
hosts = [
 "alpha",
 "omega"
]
```

YAML

```yaml
title: YAML Example
owner:
  name: Tom Preston-Werner
  dob: 1979-05-27T07:32:00-08:00
database:
  server: 192.168.1.1
  ports: [ 8001, 8001, 8002 ]
  connection_max: 5000
  enabled: true
servers:
  alpha:
    ip: 10.0.0.1
    dc: eqdc10

  beta:
    ip: 10.0.0.2
    dc: eqdc10
clients:
  data: [ [gamma, delta], [1, 2] ]

  hosts:
    - alpha
    - omega
```

## 3. Documentación de proyectos

Las metodologías ágiles se han impuesto en el desarrollo de productos y la gestión de proyectos, tanto en la industria del software como en el resto de los sectores. Esta metodología impregna todo el proceso de construcción de soluciones, incluso la elaboración de documentación. En este apartado estudiaremos cómo sus principios han influido en el proceso de documentación, integrando este proceso con el resto de los procesos, de forma que ahora se tiende a su automatización y así se consigue reducir los tiempos y los errores.

**Importancia de la documentación**

La documentación es esencial para la comprensión y el mantenimiento de cualquier proyecto. Facilita la colaboración y asegura que el conocimiento se comparta de manera efectiva.

**Tipos de documentación:**

- **Documentación técnica:** Detalles sobre la arquitectura, el diseño y el código del proyecto. Incluye diagramas de arquitectura, descripciones de módulos y componentes, y explicaciones detalladas del código fuente.
- **Manual de usuario:** Instrucciones para los usuarios finales sobre cómo utilizar el software. Incluye guías paso a paso, capturas de pantalla, y soluciones a problemas comunes.
- **Guías de referencia:** Información detallada sobre las funciones y características del software. Incluye descripciones de comandos, parámetros, y ejemplos de uso.
- **Tutoriales:** Pasos guiados para realizar tareas específicas. Incluye ejemplos prácticos, ejercicios, y proyectos de ejemplo para ayudar a los usuarios a aprender a utilizar el software de manera efectiva.

**Buenas prácticas:**

- Mantener la documentación actualizada.
- Utilizar un lenguaje claro y conciso.
- Incluir ejemplos y capturas de pantalla.
- Organizar la documentación de manera lógica y accesible.

### 3.1. DocOps y principios agiles aplicados a la documentación

El propósito último de la documentación es ayudar a desplegar y usar la aplicación, sin tener que descifrar esta información del código. Se convierte, por tanto, en un recurso apreciado tanto por los usuarios finales como por el equipo de desarrollo e implantación. Tanto es así que un buen proyecto sin documentación se puede convertir en un proyecto infrautilizado.

Del mismo modo que aparecieron soluciones **DevOps** para paliar los problemas encontrados en el desarrollo de proyectos, en los últimos años ha surgido el término **DocOps** que es un conjunto de prácticas que funciona para automatizar e integrar el proceso de desarrollo de documentación en los equipos de ingeniería, producto, soporte y, por supuesto, redacción técnica.

**DocOPs** se basa de algún modo en principios ágiles adaptados a la creación de documentación:

1. Principio **KISS (Keep it simple, stupid):** la documentación exhaustiva requerirá un gran esfuerzo en un contexto en que los requisitos cambian constantemente, y por ello se escribirá documentación poco compleja.
2. Principio **YAGNI (You are not gonna read it)**: este principio aplicado a la documentación se puede interpretar como que no se deben documentar cosas que sean posiblemente necesarias en un futuro, sino cosas que realmente se necesiten.
3. Principio **TAGRI (They ain't gonna read it)**: viene a indicar que la documentación muchas veces no va a ser leída por los destinatarios, así que es mejor que mantengamos una documentación ágil que refleje los intereses del destinatario.

De esta forma se consigue documentación ágil, eficaz, ligera y de alto valor.

### 3.2 Documentos de proyectos Python

Aunque según la naturaleza del proyecto la documentación puede variar, a continuación se muestra una tabla con los documentos que suelen formar parte de ella junto con su localización habitual:

| Documento | Localización | Descripción |
|---|---|---|
| README | Raíz | Información para el usuario final sobre qué es el proyecto, cómo instalarlo y cómo usarlo. |
| Licencia | Raíz | Referencia a quién tiene los derechos y como se puede usar y compartir. |
| Guía para contribuir | Raíz | Cómo contribuir al proyecto. |
| Código de conducta | Raíz | Define estándares de cómo participar y contribuir adecuadamente al proyecto y su comunidad. |
| Historial de cambios | Raíz | Lista cronológica de cambios importantes. |
| Docstrings | Archivos .py | Texto que aparece junto a una función, clase o módulo en Python que describe qué hace el código y cómo usarlo. Accesible para los usuarios a través del comando help(). |
| Ejemplos | docs/ | Guías paso a paso sobre cómo funciona alguna parte con más detalle. |
| Referencia a la API | docs/ | Lista organizada de la funcionalidad destinada al usuario (es decir, funciones, clases, etc.) junto con una breve descripción de lo que hacen y de cómo usarlos. Se suele crear automáticamente a partir de los Docstrings. |
| Otra documentación | docs/ | FAQS ('frequently asked questions'), resolución de problemas conocidos... |

### 3.3. Produciendo documentación

La producción de documentación suele consistir en 3 pasos:

1. Escribir la documentación: ya sea cualquiera de los documentos puramente de documentación o los **docstring** que acompañan al desarrollo de software o a los test.
2. Construir la documentación: renderizar la documentación en un formato organizado de forma coherente y que se pueda compartir. Puede ser en HTML o en PDF.
3. Alojar la documentación online: compartir la documentación para que ser accedida fácilmente por cualquier interesado con conexión a internet, por ejemplo, a través de GitHub Pages.

Algunos **hubs de repositorios**, como es el caso de **GitHub**, construyen automáticamente la documentación a partir de las fuentes en formatos **Markdown**.

### 3.4 Generación automática de documentación

El proceso de generación de la documentación también se puede automatizar utilizando herramientas de documentación automatizada, las cuales facilitan su creación y mantenimiento. Hacerlo de forma manual suele llevar a errores y lentifica el proceso de desarrollo, por ello es penalizado gravemente en procesos ágiles.

Además, en un entorno colaborativo, la automatización de la documentación hará que sea más fácil trabajar con diferentes equipos de forma sincronizada; este proceso se añadirá en el de integración y entrega continua de forma que la documentación para el usuario final esté bien estructurada y sea comprensible incluso para alguien con pocos conocimientos técnicos.

### 3.5 Contenidos por Tipo de Documentación:

Cada tipo de documentación tiene un propósito y audiencia, y juntos forman un conjunto completo y cohesivo que cubre todas las necesidades de información del proyecto. Los índices que se proporcionan a continuación para cada tipo de documentación no son del todo estrictos, pero sirven como una guía útil para su elaboración y organización.

#### 3.5.1 Documentación técnica

1. **Introducción**
    - Propósito de la documentación técnica
    - Audiencia objetivo
2. **Arquitectura del sistema**
    - Diagrama de arquitectura
    - Descripción de componentes
3. **Diseño del sistema**
    - Diagramas de diseño
    - Patrones de diseño utilizados
4. **Detalles del código**
    - Estructura del código
    - Comentarios y anotaciones
5. **Configuración y despliegue**

- Requisitos del sistema
- Instrucciones de instalación
- Procedimientos de despliegue

6. **Mantenimiento y soporte**
   - Procedimientos de mantenimiento
   - Solución de problemas comunes

**3.5.2 Manual de usuario**

1. **Introducción**
   - Propósito del manual
   - Audiencia objetivo
2. **Instalación del software**
   - Requisitos del sistema
   - Instrucciones de instalación
3. **Guía de inicio rápido**
   - Primeros pasos
   - Configuración inicial
4. **Uso del software**
   - Descripción de la interfaz
   - Funcionalidades principales
   - Ejemplos de uso
5. **Solución de problemas**
   - Problemas comunes y soluciones
   - Preguntas frecuentes (FAQ)
6. **Apéndices**
   - Glosario de términos
   - Recursos adicionales

**3.5.3. Guías de referencia**

1. **Introducción**
   - Propósito de la guía de referencia
   - Audiencia objetivo
2. **Descripción de comandos**

- Lista de comandos
- Sintaxis y parámetros

3. **Funciones y características**
   - Descripción detallada de funciones
   - Ejemplos de uso
4. **Configuración avanzada**
   - Opciones de configuración
   - Personalización del software
5. **Ejemplos prácticos**
   - Casos de uso
   - Ejemplos de scripts
6. **Apéndices**
   - Referencias cruzadas
   - Recursos adicionales

#### 3.5.4 Tutoriales

1. **Introducción**
   - Propósito del tutorial
   - Audiencia objetivo
2. **Primeros pasos**
   - Instalación y configuración
   - Configuración inicial
3. **Tareas básicas**
   - Tutoriales paso a paso
   - Ejemplos prácticos
4. **Tareas avanzadas**
   - Funcionalidades avanzadas
   - Ejemplos prácticos
5. **Proyectos de ejemplo**
   - Proyectos completos
   - Ejercicios prácticos
6. **Conclusión**
   - Resumen de lo aprendido

- Recursos adicionales

### 3.6 Tipo de documentación, propósito y audiencia

#### 3.6.1 Documentación técnica

**Propósito:** Proporcionar detalles técnicos sobre la arquitectura, el diseño y el código del proyecto. Facilita la comprensión y el mantenimiento del sistema por parte del equipo técnico.

**Audiencia:**

- Desarrolladores
- Ingenieros de software
- Arquitectos de sistemas
- Administradores de sistemas

#### 3.6.2. Manual de usuario

**Propósito:** Ofrecer instrucciones claras y detalladas sobre cómo utilizar el software. Ayuda a los usuarios finales a comprender y utilizar el software de manera efectiva.

**Audiencia:**

- Usuarios finales
- Clientes
- Personal de soporte técnico

#### 3.6.3. Guías de referencia

**Propósito:** Proporcionar información detallada sobre las funciones y características del software. Sirve como una referencia rápida para comandos, parámetros y ejemplos de uso

**Audiencia:**

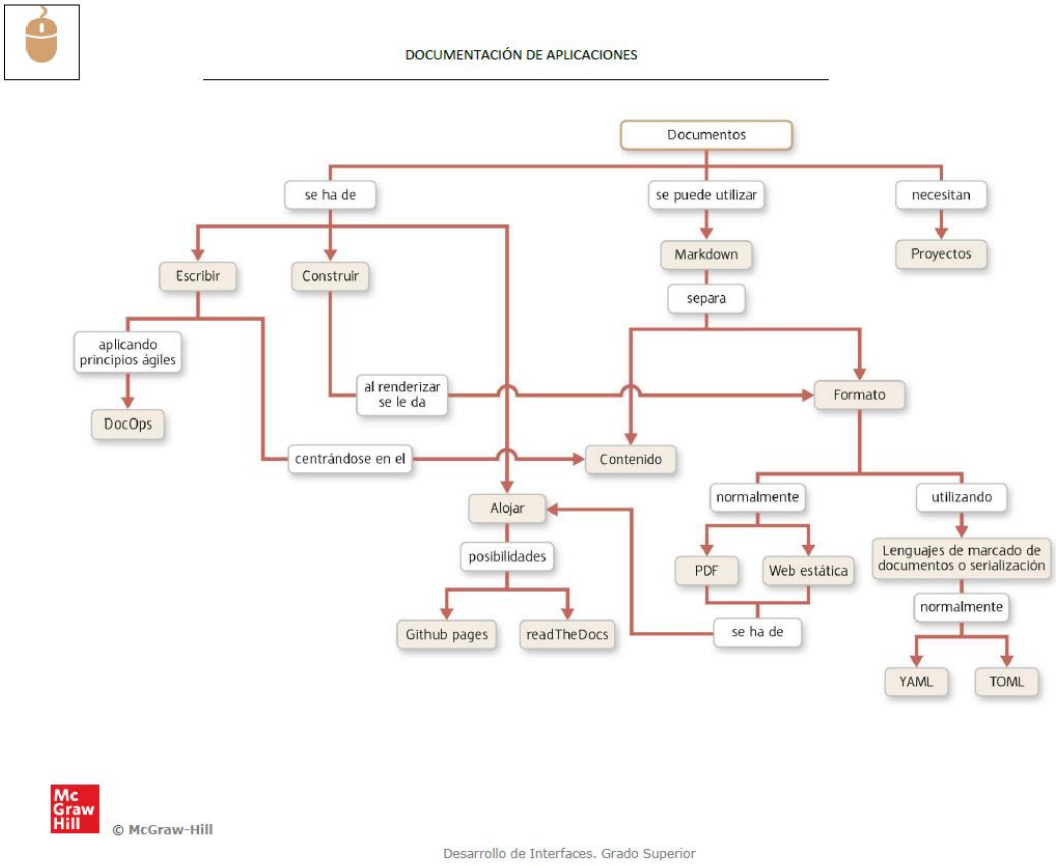
- Desarrolladores
- Ingenieros de software
- Usuarios avanzados

### 3.6.4. Tutoriales

**Propósito:** Ofrecer pasos guiados para realizar tareas específicas. Ayuda a los usuarios a aprender a utilizar el software de manera práctica y efectiva a través de ejemplos y ejercicios.

**Audiencia:**

- Nuevos usuarios
- Desarrolladores en formación
- Cualquier persona interesada en aprender a utilizar el software






Desarrollo de Interfaces. Grado Superior

---

**Bibliografía**:

- **Cunyat Pellicer, F., Catalá Jiménez, J. (2022). *Desarrollo de Interfaces Gráficas*. McGraw Hill.**