

# Documentation

---

Project documentation

*Author*

*Copyright © 2026*

## Table of Contents

---

1. Bem-vindo	3
1.1 Começando	3
1.2 Estrutura	3
1.3 Exportar para PDF	3
2. Descarregar PDF	4
3. Versionamento	5
3.1 Pré-requisitos	5
3.2 Publicar uma nova versão	5
3.3 Definir a versão predefinida	5
3.4 Aliases	5
3.5 Desenvolvimento local	5
4. Examples	6
4.1 Hello World	6
4.1.1 Exemplo com anotações	6
4.2 Mais exemplos	7

# 1. Bem-vindo

---

Bem-vindo à documentação do projeto.

## 1.1 Começando

---

Adicione secções e páginas conforme necessário. A navegação é configurada em `mkdocs.yml` na secção `nav`.

## 1.2 Estrutura

---

- **Início** — esta página
- **Download PDF** — obter a documentação em PDF em inglês ou português
- **Exemplos** — Hello World e mais exemplos de código em várias linguagens
- Adicione novas páginas em `docs/` e referencie-as em `mkdocs.yml`

## 1.3 Exportar para PDF

---

Os PDFs são gerados em cada build. O PDF do idioma predefinido fica em `site/pdf/documentation.pdf`. Para gerar PDFs em inglês e português, execute:

```
./scripts/build-pdf-all.sh
```

É necessário o WeasyPrint; no macOS instale primeiro as dependências do sistema:

```
brew install cairo pango gdk-pixbuf
```

## 2. Descarregar PDF

---

Descarregue a documentação num único PDF. Escolha o idioma:

[Download PDF — English](#)[Download PDF — Português](#)

### Dois PDFs exigem o script de build

Um `mkdocs build` ou `mkdocs serve` normal executa duas builds de idioma; o PDF é escrito no mesmo caminho em cada uma, por isso fica só **um** ficheiro (português) e a ligação PT pode dar 404. Para obter **ambos** os PDFs (EN e PT) e ligações de download a funcionar, execute primeiro o script e depois sirva o site gerado:

```
./scripts/build-pdf-all.sh
./scripts/serve-with-pdfs.sh
```

Depois abra o URL indicado (ex.: `http://127.0.0.1:8000/repo/` ou `http://127.0.0.1:8000/`). Ambas as ligações de download funcionarão.

### Build único (ambos os PDFs, sem servidor):

```
./scripts/build-pdf-all.sh
```

**Apenas um PDF** (ex.: para CI): execute `mkdocs build` ou `mkdocs serve`; o ficheiro fica em `site/pdf/documentation.pdf` (o conteúdo será da última build, normalmente PT).

## 3. Versionamento

---

A documentação é versionada com [mike](#). O seletor de versão no cabeçalho permite alternar entre as versões publicadas.

### 3.1 Pré-requisitos

---

- Definir `site_url` em `mkdocs.yml` com o URL de publicação (ex.: `https://username.github.io/repo/`).
- Publicar o site numa branch (ex.: `gh-pages`) que o `mike` utilizará.

### 3.2 Publicar uma nova versão

---

Publicar a documentação atual como nova versão e definir o alias `latest`:

```
mike deploy --push --update-aliases 1.0 latest
```

- `1.0` — identificador da versão (pode ser `2.0`, `dev`, etc.).
- `latest` — alias; quem aceder ao URL raiz é redirecionado para esta versão.

Publicar outra versão sem alterar a predefinida:

```
mike deploy --push 0.9
```

### 3.3 Definir a versão predefinida

---

Redirecionar a raiz do site para a versão associada a um alias:

```
mike set-default --push latest
```

Assim, ao abrir `site_url` o utilizador é redirecionado para a versão `latest`.

### 3.4 Aliases

---

Use aliases para versões estáveis, de desenvolvimento ou releases específicos:

```
mike deploy --push --update-aliases 2.0 latest  
mike deploy --push 2.0-beta dev  
mike deploy --push 1.0 stable
```

Depois defina a predefinida, por exemplo:

```
mike set-default --push latest
```

### 3.5 Desenvolvimento local

---

Ao executar `mkdocs serve` o `mike` não é usado; o seletor de versão só é preenchido após publicar com `mike`. Para testar o versionamento, faça build e publique pelo menos uma versão na sua branch.

## 4. Examples

---

### 4.1 Hello World

---

Exemplos clássicos de "Hello, World!" em diferentes linguagens de programação. Alterne entre eles usando as abas abaixo.

Python    JavaScript    Go    Java    C++    Ruby

```
print("Hello, World!")

console.log("Hello, World!");

package main

import "fmt"

func main() {
    fmt.Println("Hello, World!")
}

public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}

#include <iostream>

int main() {
    std::cout << "Hello, World!" << std::endl;
    return 0;
}

puts "Hello, World!"
```

---

#### 4.1.1 Exemplo com anotações

O bloco abaixo usa **números de linha** ( `linenums="1"` ), **linhas destacadas** ( `hl_lines="2 3"` ), um **título** e **anotações** — passe o rato ou toque no marcador para ver a nota.

```
1 def greet():
2     print("Hello, World!") # (1)!
3     return None           # (2)!
```

1. Envia a saudação para stdout. Use `print()` para saída simples na consola.
2. O `return None` explícito é opcional em Python; a função retorna `None` por defeito.

## 4.2 Mais exemplos

---

Exemplos adicionais de "Hello, World!" noutras linguagens. Use as abas para alternar.

**Rust**   **PHP**   **Swift**   **Kotlin**   **C**   **Bash**

```
fn main() {  
    println!("Hello, World!");  
}
```

```
<?php  
echo "Hello, World!";  
  
print("Hello, World!");
```

```
fun main() {  
    println("Hello, World!")  
}
```

```
#include <stdio.h>  
  
int main(void) {  
    printf("Hello, World!\n");  
    return 0;  
}
```

```
echo "Hello, World!"
```