Welcome to your Jupyter Book

This is a small sample book to give you a feel for how book content is structured.

> **Note**
>
> Here is a note!

And here is a code block:

```
e = mc^2
```

Check out the content pages bundled with this sample book to see more.

1. Markdown Files

Whether you write your book's content in Jupyter Notebooks (`.ipynb`) or in regular markdown files (`.md`), you'll write in the same flavor of markdown called **MyST Markdown**.

1.1. What is MyST?

MyST stands for "Markedly Structured Text". It is a slight variation on a flavor of markdown called "CommonMark" markdown, with small syntax extensions to allow you to write **roles** and **directives** in the Sphinx ecosystem.

1.2. What are roles and directives?

Roles and directives are two of the most powerful tools in Jupyter Book. They are kind of like functions, but written in a markup language. They both serve a similar purpose, but **roles are written in one line**, whereas **directives span many lines**. They both accept different kinds of inputs, and what they do with those inputs depends on the specific role or directive that is being called.

1.2.1. Using a directive

At its simplest, you can insert a directive into your book's content like so:

````
```{mydirectivename}
My directive content
```
````

This will only work if a directive with name `mydirectivename` already exists (which it doesn't). There are many pre-defined directives associated with Jupyter Book. For example, to insert a note box into your content, you can use the following directive:

````
```{note}
Here is a note
```
````

This results in:

> **Note**
>
> Here is a note

In your built book.

For more information on writing directives, see the MyST documentation.

1.2.2. Using a role

Roles are very similar to directives, but they are less-complex and written entirely on one line. You can insert a role into your book's content with this pattern:

```
Some content {rolename}`and here is my role's content!`
```

Again, roles will only work if `rolename` is a valid role's name. For example, the `doc` role can be used to refer to another page in your book. You can refer directly to another page by its relative path. For example, the role syntax `{doc}`../intro`` will result in: Welcome to your Jupyter Book.

For more information on writing roles, see the MyST documentation.

1.2.3. Adding a citation

You can also cite references that are stored in a `bibtex` file. For example, the following syntax: `{cite}`holdgraf_evidence_2014`` will render like this: [HdHPK14].

Moreover, you can insert a bibliography into your page with this syntax: The `{bibliography}` directive must be used for all the `{cite}` roles to render properly. For example, if the references for your book are stored in `references.bib`, then the bibliography is inserted with:

````
```{bibliography}
```
````

Resulting in a rendered bibliography that looks like:

[[HdHPK14]] Christopher Ramsay Holdgraf, Wendy de Heer, Brian N. Pasley, and Robert T. Knight. Evidence for Predictive Coding in Human Auditory Cortex. In *International Conference on Cognitive Neuroscience*. Brisbane, Australia, Australia, 2014. Frontiers in Neuroscience.

1.2.4. Executing code in your markdown files

If you'd like to include computational content inside these markdown files, you can use MyST Markdown to define cells that will be executed when your book is built. Jupyter Book uses *jupytext* to do this.

First, add Jupytext metadata to the file. For example, to add Jupytext metadata to this markdown page, run this command:

```
jupyter-book myst init markdown.md
```

Once a markdown file has Jupytext metadata in it, you can add the following directive to run the code at build time:

````
```{code-cell}
print("Here is some code to execute")
```
````

When your book is built, the contents of any `{code-cell}` blocks will be executed with your default Jupyter kernel, and their outputs will be displayed in-line with the rest of your content.

For more information about executing computational content with Jupyter Book, see The MyST-NB documentation.

2. Content with notebooks

You can also create content with Jupyter Notebooks. This means that you can include code blocks and their outputs in your book.

2.1. Markdown + notebooks

As it is markdown, you can embed images, HTML, etc into your posts!

You can also \(add_{math}\) and
\[math^{blocks} \]
or
\[\begin{split} \begin{aligned} \mbox{mean} la_{tex} \\ \\ math blocks \end{aligned} \end{split}\]
But make sure you $Escape $your $dollar signs $you want to keep!

2.2. MyST markdown

MyST markdown works in Jupyter Notebooks as well. For more information about MyST markdown, check out the MyST guide in Jupyter Book, or see the MyST markdown documentation.

2.3. Code blocks and outputs

Jupyter Book will also embed your code blocks and output in your book. For example, here's some sample Matplotlib code:

```
from matplotlib import rcParams, cycler
import matplotlib.pyplot as plt
import numpy as np
plt.ion()
```

```
# Fixing random state for reproducibility
np.random.seed(19680801)

N = 10
data = [np.logspace(0, 1, 100) + np.random.randn(100) + ii for ii in range(N)]
data = np.array(data).T
cmap = plt.cm.coolwarm
rcParams['axes.prop_cycle'] = cycler(color=cmap(np.linspace(0, 1, N)))


from matplotlib.lines import Line2D
custom_lines = [Line2D([0], [0], color=cmap(0.), lw=4),
                Line2D([0], [0], color=cmap(.5), lw=4),
                Line2D([0], [0], color=cmap(1.), lw=4)]

fig, ax = plt.subplots(figsize=(10, 5))
lines = ax.plot(data)
ax.legend(custom_lines, ['Cold', 'Medium', 'Hot']);
```

There is a lot more that you can do with outputs (such as including interactive outputs) with your book. For more information about this, see the Jupyter Book documentation

3. Flask 소개 (Intro)

3.1. Flask 초간단 소개

플라스크 둘러보기를 통해 개략적으로 Flask를 이해하는 시간을 갖도록 하겠습니다.

플라스크는 기본적으로 Python 언어를 사용한 웹 프레임워크입니다. Python 웹 프레임워크는 Django(장고)와 Flask(플라스크)가 있습니다. Flask는 마이크로 프레임워크를 지향하고 있습니다.

> 마아크로 프레임워크란?
>
> 마이크로 프레임워크 (Micro Framework)는 소규모 웹 애플리케이션 프레임워크를 가리키기 위한 용어입니다. 개발에 필요한 대부분의 기능을 미리 제공하는 풀 스택 프레임워크 (Full-stack Framework)와 대조되는 용어입니다. 마이크로 프레임워크는 개발자에게 핵심적으로 필요한 기능만 제공하고 나머지 부분들은 개발자의 몫으로 남겨둡니다. 다음과 같은 내용들은 제공되지 않기 때문에 개발자가 책임을 지고 구현해야 합니다.
>
> - 계정, 인증, 인가, 역할 등
> - 객체 지향 매핑을 통한 데이터베이스 추상화
> - 입력 확인 및 입력 정제
> - 웹 탬플릿 엔진
>
> 위와 같은 사항을 프레임워크가 지원하지 않으므로 개발자는 구현해도 되고 안해도 됩니다. 그만큼 구현해야 할 범위가 작아져서 빠르게 개발할 수 있습니다. 하지만 위 기능이 필요한 시스템이라면 개발자가 스스로 구현해야 합니다.

Flask와 Django는 다음과 같은 차이점이 있습니다. 우리는 Flask에 한정해서 공부해 볼 것입니다.

Framework	Flask	Django
출시년도	2010	2005
ORM 지원	X	O
아키텍처	MSA(Micro Service Architecture)	Monolithic
지원기능	적음	많음
초기 학습량	적은 학습으로 사용 가능	상대적으로 배워야할 것이 많음
소스코드 양	코딩량이 상대적으로 적음	Flask보다 소스코드 크기가 큼(무거움)
개발 자유도	높음	낮음
개발자 책임	높음	낮음(프레임워크)에서 대부분 지원

ORM(Object Relational Mapping)은 프레임워크에서 데이터베이스를 객체로 관리할 수 있도록 지원하는 기능입니다. ORM을 사용하면 별도로 공부하지 않아도 편리하게 데이터베이스를 조작할 수 있습니다.

MSA와 Monolithic에 대한 구체적 설명은 여기를 참고하기 바랍니다.

3.2. 어떤 프레임워크를 더 많이 쓸까요?

Django와 Flask의 인기는 github.com에서의 좋아요(스타) 숫자를 보면 대충 짐작할 수 있습니다. 아래 바로가기 이미지를 클릭하여 Django Github 또는 Flask Github 페이지로 이동하여 'Star' 숫자를 확인해 보세요.

2022년 1월 기준으로 Github의 좋아요(stars)와 퍼가기(Fork) 숫자는 다음과 같습니다.

Framework	Flask	Django
좋아요(Stars)	57,600개	61,600개
복제횟수(Forks)	14,800회	26,300회

위 표만 보면 Django가 보다 많은 좋아요/복제가 발생한 것으로 보아 좀 더 많이 쓰는 것처럼 보입니다. Django의 나이가 더 많으니 어찌보면 당연해 보이기도 합니다. 하지만 짧은 기간에도 불구하고 많은 사람들이 Flask를 사용하고 있습니다. 하지만 단순히 숫자만 가지고 어떤 프레임워크가 좋다고 말하는 것은 바보같은 짓입니다. 각각의 프레임워크는 나름대로의 장단점이 있기 때문입니다.

> **Git의 기능: fork, clone**
>
> - `fork`는 다른 사람이 만든 Git 저장소(repository, 이하 repo)를 내 repo로 복제하는 기능입니다. 원본 repo (내가 복사해온 repo 주인)와 나의 repo는 서로 연결되어 있습니다.
>   - 원본 repo가 변경 \(\to\) 내 repo로 반영: `fetch`, `pull` 사용
>   - 내 repo가 변경 \(\to\) 원본 repo로 반영: `push` 사용, 원본 repo 주인이 승인해 주면 반영
> - `clone`은 내가 만든 원격 저장소(repo) 또는 `fork`를 통해 내 원격 repo로 복제한 것을 로컬 컴퓨터로 복사하는 것입니다.
>   - 내 컴퓨터에서 작업 \(\to\) 원격 repo로 전송: `add`, `commit`, `push` 사용
>   - `fork`한 경우 `push` 내 원격 repo만 업데이트 됩니다.
>   - `fork` 해온 원본 repo에 내 원격 repo의 변경사항을 반영하기 위해서는 별도의 과정을 거쳐야 합니다.
>   - 내 원격 repo 변경을 원본 repo에 반영하는 절차는 이 블로그를 참고하세요.

Flask는 지원 기능이 적기 때문에 필요한 기능을 구현할 때마다 별도의 라이브러리를 설치해야 합니다. 그리고 라이브러리를 설치하면 그 때마다 Flask와 연결하는 `binding` 작업을 해주어야 합니다. Flask는 빠르고 가볍게 구현할 수 있지만 기능이 늘어날 수록 개발에 필요한 `cost (비용)`도 같이 증가합니다. 간단한 시스템을 구현할 때는 flask를 사용하는 것이 답일 수 있지만 복잡하거나 `커스터마이징(customizing)`이 많이 필요한 시스템에는 적당하지 않습니다.

다음은 Python과 Web Framework에 대한 2021년도 Survey 결과입니다. 구체적인 내용은 Jetbrain 홈페이지를 참고하기 바랍니다.

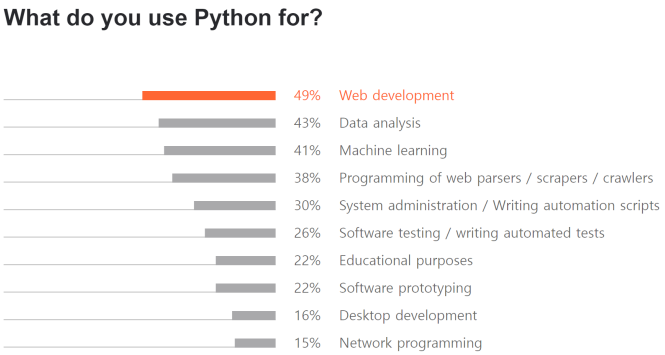

Fig. 3.1 Python 사용의 목적

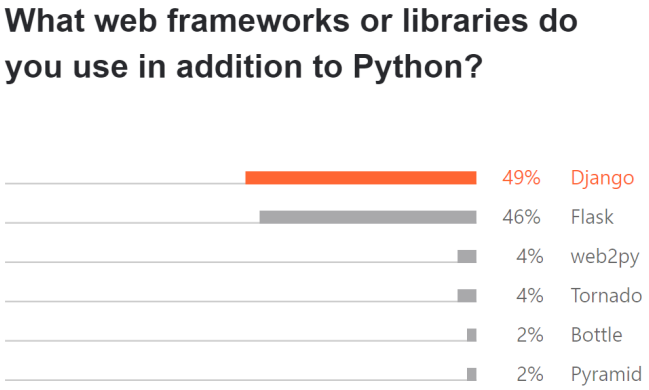

Fig. 3.2 Python 개발자의 웹 프레임워크 사용 비율

4. Flask 실행을 위한 사전 준비

Flask를 실행하기 위해서는 몇 가지 사전 준비사항이 필요합니다. 몇 가지 해주어야 할 일들 (To Do List)는 다음과 같습니다.

- VS code 설치
- 가상환경 (Virtual Environment) 설치
- `flask` 설치
- `.gitignore` 세팅
- 의존성 파일 `requirement.txt` 생성

4.1. VS (Visual Studio) code 설치

Python 언어를 이용해 시스템이나 서비스를 개발하기 위해서는 다양한 기능을 제공하는 **편집기(editor)**를 사용하는 것이 여러모로 편리합니다. 불필요한 시간낭비(프로그래머들은 종종 '삽질' 이라고 부르기도 함)을 줄여줄 수 있습니다. 개발자들이 주로 사용하는 편집기(editor) 목록은 다음과 같습니다. 아래 목록은 Stack Overflow에서 2021년에 실시한 Survey 결과를 참조하였습니다.

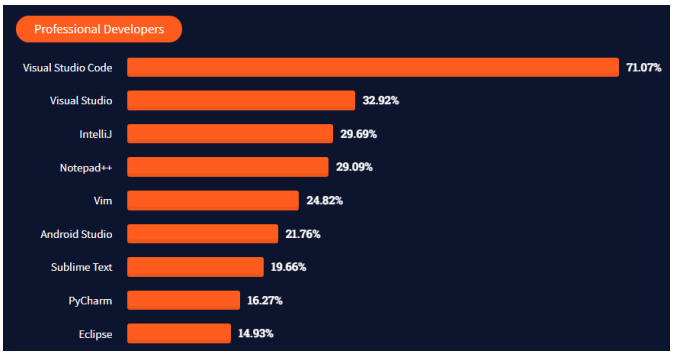

Fig. 4.1 전세계 개발자들이 선호하는 에디터(Stack Overflow 2021 조사)

위에서 열거한 편집기 중 Python을 지원하는 편집기라면 어떤 것을 선택하더라도 상관 없습니다. 본인이 평소에 자주 사용하여 친숙한 편집기가 있다면 그대로 사용해도 무방합니다. 각 편집기별 특징을 정리하면 아래 표와 같습니다.

순번	편집기**(editor)**	특징	가격	**Python** 지원
1	VS code (Visual Studio Code)	범용(다양한 프로그래밍 언어 지원)	무료	O
2	Visual Studio	범용(다양한 프로그래밍 언어 지원)	무료(상업용인 경우 유료)	O
3	IntelliJ	Java 개발	유료 및 무료(교육기관)	X
4	Notepad++	범용(다양한 프로그래밍 언어 지원)	무료	O
5	Vim	범용(다양한 프로그래밍 언어 지원), 리눅스에 기본 탑재	무료	O
6	Android Studio	안드로이드 전용 Application (앱) 개발	무료	O
7	Sublime Text	범용(다양한 프로그래밍 언어 지원)	무료	O
8	PyCharm	범용(다양한 프로그래밍 언어 지원)	무료(상업용인 경우 유료)	O
9	Eclipse	범용(다양한 프로그래밍 언어 지원)	무료	O

5. Monolithic vs. MSA

Flask를 배우다 보면 선배 개발자들이 "플라스크는 MSA라는 것은 알고 있지?"라는 말을 자주 해줍니다. 그런데 처음 웹 프레임워크를 배우는 학생들은 **MSA**라는 용어 자체가 생소한 경우가 많습니다.

MSA라는 말을 처음 들어보는 것도 멘붕인데 선배 개발자들이 이런 말까지 합니다. “Django는 Monolithic 아키텍처야. 그래서 Django와 Flask는 근본적으로 달라. 알지?”

이런말을 처음 접하게 된다면 적지 않게 당황하게 됩니다. 하지만 알고 보면 별로 복잡하지 않은 개념입니다. Flask를 배우는 김에 MSA와 Monolisic(모놀리식)의 개념에 대해 간단히 살펴보도록 하겠습니다.

Monolithic과 MSA에 대한 설명은 [MSA] Monolithic Architecture VS Micro Service Architecture 참조하여 재정리 하였으니 관심있는 독자는 해당 블로그를 방문하기 바랍니다.

5.1. Monolithic Architecture

Monolithic(모놀리식)을 네이버 영어사전에서 찾아보면 ‘(조직, 단결 등이) 단일체인, 한 덩어리로 뭉친’이라고 나옵니다. 사전적 의미 그대로 모놀리식 아키텍처는 어떤 서비스(또는 프로젝트)를 하나의 큰 덩어리로 된 어플리케이션(Application, 이하 앱)으로 만든다는 뜻입니다. 우리가 일반적으로 생각하는 방식과 비슷합니다.

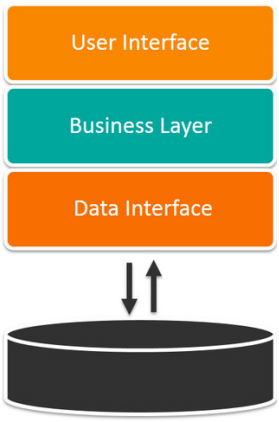

Fig. 5.1 모놀리식 아키텍처 (이미지 출처: sangmin7476)

모놀리식 아키텍처는 장단점이 존재합니다. 장점과 단점에 대해 간단히 알아보겠습니다.

- 장점
  - 로컬 머신(예: 개인 컴퓨터)에서 편리하게 개발 \(\to\) 한 덩어리로 만들기 때문에 분산 개발 환경이 불필요합니다.
  - 통합된 시나리오를 활용해서 전체 기능을 쉽게 테스트할 수 있습니다.
  - 배포가 간단합니다(하나의 앱만 배포하면 끝)
- 단점
  - 큰 덩어리로 되어 있기 때문에 유지보수(코드 수정 및 추가)가 어렵습니다.
  - 필요에 따라 자원(CPU, Memory, Storage 등)을 적절히 분배하기 어렵습니다.
  - 신속한 업데이트가 어렵습니다.
  - 새로운 기술이 개발되었을 경우 적용하기 어렵습니다(유지보수가 어려운 것과 같은 이치)
  - 부분적 장애가 발생해도 전체 시스템 작동이 안될 수 있습니다.
  - Scale out이 어렵습니다.

> 스케일 아웃**(scale out)**에 대한 설명
>
> 갑자기 이용자(접속자)가 늘어나면 서버를 몇 대 늘여서 서비스 능력을 높여줘야 합니다. 이렇게 서버를 증설하는 것을 `scale out` 한다고 말합니다. 앱이 기능별로 분리가 되어 있다면 증설이 쉽습니다. 청주대 온라인 강의 시스템에 학생들이 몰리면 온라인 강좌만 담당하는 서버만 추가하면 됩니다. 만약 청주대의 모든 시스템이 하나의 큰 덩어리로 코딩되어 있으면 scale out 하기 매우 어려워 집니다.

모놀리식 구조는 한 덩어리만 생각하면 되므로 단순합니다. 그래서 개발도, 테스트도, 배포도 편리합니다. 하지만 개발 규모가 커지면 이런 단순함이 독이 됩니다. 왜냐하면 모듈간 의존성이 높아지게 되고, 개발자들은 코드 이해가 어렵고, 하나의 프로젝트에 많은 개발자가 투입되어 의사소통도 어려워집니다. 그래서 대안으로 등장한 것이 **MSA** 입니다. 이제 MSA

Architecture에 대해 간단히 살펴보도록 하겠습니다.

5.2. MSA Architecture

MSA는 'Micro Service Architecture'의 약어입니다. 단어 그대로 아주 작은 서비스를 위한 아키텍처라는 말입니다. 기술적(Technical)으로 말하면 하나의 큰 앱을 작은 앱으로 나누어 만드는 방법입니다. 우리가 직관적으로 생각하는 앱은 전체가 한 덩어리로 된 하나의 앱으로 만드는 것입니다.

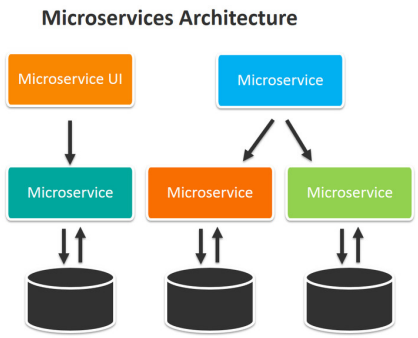

Fig. 5.2 MSA 아키텍처 (이미지 출처: sangmin7476)

물론 MSA도 장단점이 있습니다.

- 장점
  - 빌드 및 테스트 시간이 줄어듭니다. 이미 만들어진 서비스는 다시 빌드하지 않고 새롭게 추가되거나 수정된 부분만 처리하기 때문에 시간을 아낄 수 있습니다.
  - 서비스를 분리해서 개발하므로 유연하게 기술을 적용할 수 있습니다. 인증 기능은 django auth로 구현하고, 채팅 기능은 node를 이용해서 구현하는 예를 들 수 있습니다.
  - 손쉽게 scale out 가능합니다.
  - 서비스 사이에 결합도가 낮아져 유지보수가 편리하고 시스템을 안정적으로 운영할 수 있습니다.
- 단점
  - 하나의 서비스가 다른 서비스를 호출할 경우 네트워크 통신이 추가적으로 발생하므로 속도가 느려질 수 있습니다.
  - 서버를 나누고 각각의 서버가 담당하는 앱이 있기 때문에 서버간 데이터베이스 처리를 해야하는 경우 트랜잭션을 처리하는 알고리즘을 추가로 작성해야 합니다.
  - 서버가 증가하기 때문에 로깅, 모니터링, 배포, 테스트, 데이터베이스 관리 등 추가적인 작업이 필요합니다.

5.3. 모놀리식과 MSA, 어떤 것이 좋은 건가요?

앞에서 살펴본 것과 마찬가지로 어떤 아키텍처든 장단점이 있기 마련입니다. 모놀리식과 MSA도 각각 장단점이 있습니다. 따라서 어떤 것이 좋은 아키텍처라고 콕 짚어서 말할 수 없습니다. MSA가 더 좋을 것 같지만 아직도 많은 서비스가 모놀리식 아키텍처를 적용하고 있는 것으로 알려져 있습니다.

아키텍처는 것은 다양한 개발 경험과 운영 노하우를 바탕으로 선택해야 합니다. 아키텍처를 선택하거나 새롭게 고안하는 것을 아키텍처 설계(Architecture Design)라고 합니다. 소프트웨어공학 측면에서 볼때 '상위 설계'에 해당합니다. 이런 아키텍처 설계를 담당하는 사람을 아키텍처 설계자 (Architecture Designer) 또는 소프트웨어 시스템 설계자 (Software System Designer)라고 부릅니다.
