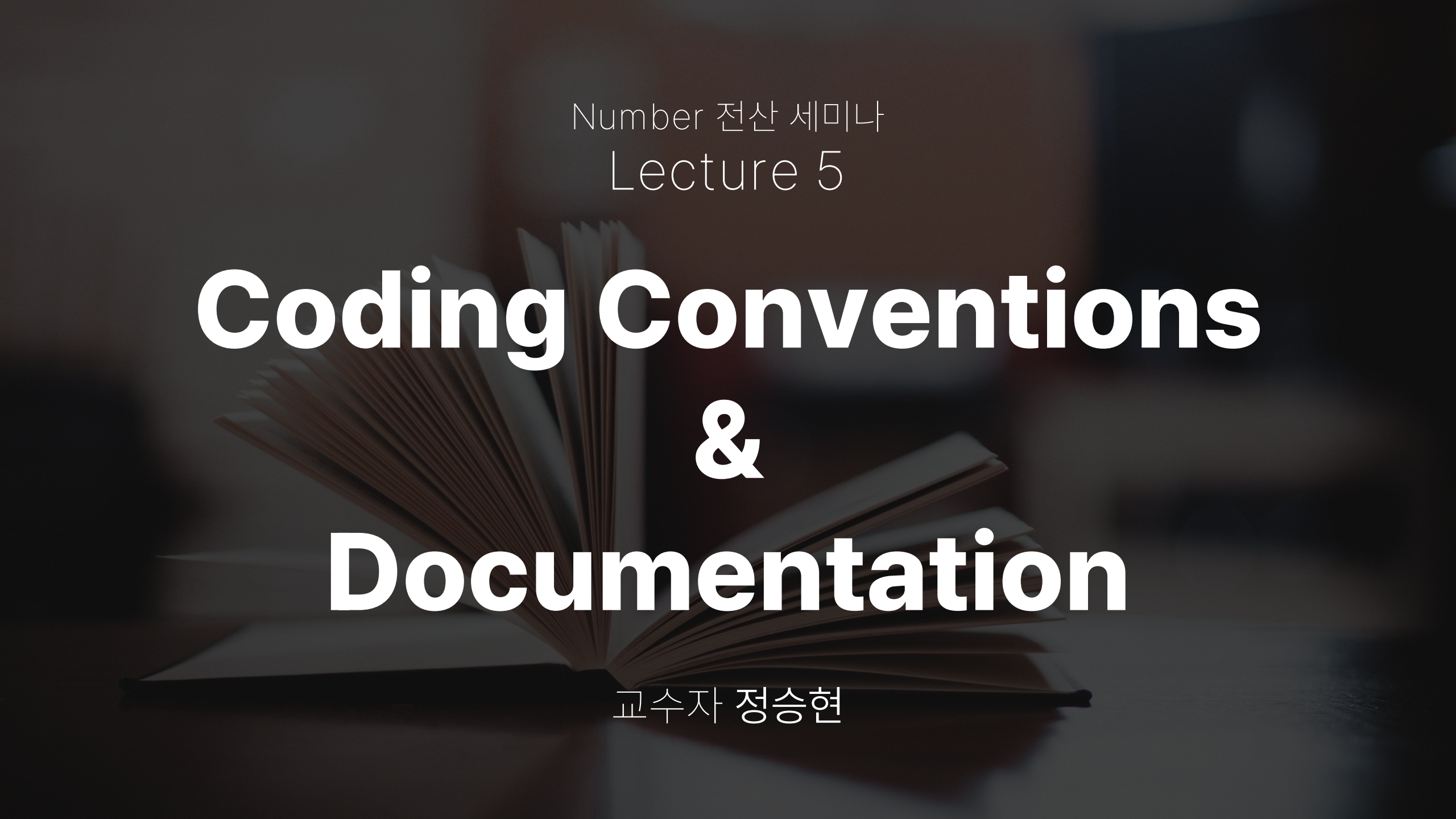


Number 전산 세미나

Lecture 5

Coding Conventions & Documentation

교수자 정승현

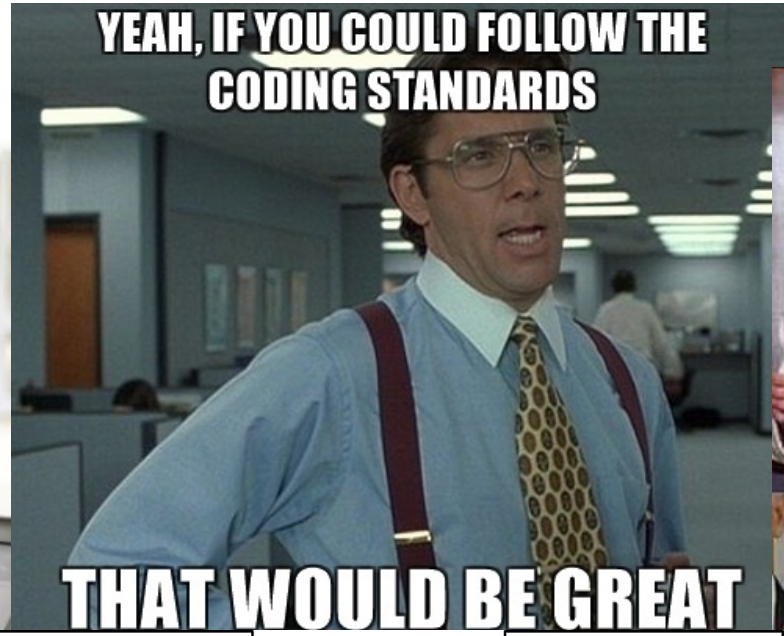
목차

A

Coding Conventions

우리의 경험 혹은 미래

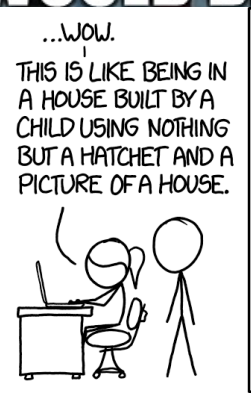

Knows who changed his code
By the style of indentation

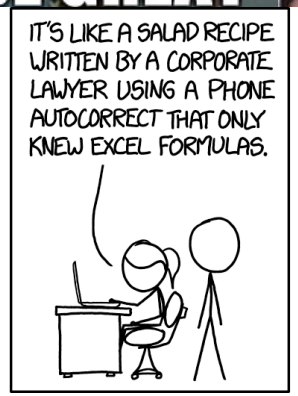

YEAH, IF YOU COULD FOLLOW THE CODING STANDARDS
THAT WOULD BE GREAT


Follow coding standards


*junior devs


KEEP IN MIND THAT I'M SELF-TAUGHT, SO MY CODE MAY BE A LITTLE MESSY.
LEMME SEE- I'M SURE IT'S FINE.


...WOW.
I
THIS IS LIKE BEING IN A HOUSE BUILT BY A CHILD USING NOTHING BUT A HATCHET AND A PICTURE OF A HOUSE.


IT'S LIKE A SALAD RECIPE WRITTEN BY A CORPORATE LAWYER USING A PHONE AUTOCORRECT THAT ONLY KNEW EXCEL FORMULAS.

IT'S LIKE SOMEONE TOOK A TRANSCRIPT OF A COUPLE ARGUING AT IKEA AND MADE RANDOM EDITS UNTIL IT COMPILED WITHOUT ERRORS.
OKAY, I'LL READ A STYLE GUIDE.

Coding Convention이란

말 그대로 coding하는 convention.

특정 프로젝트 혹은 특정 조직에서 개발을 할 때, 서로 지키기로 명시적/묵시적으로 약속한 코딩하는 스타일.

그 스타일이라는 것에 어떤 것이 포함되는지는 이따 알아보도록 하자.

https://youtube.com/shorts/_Eta6jYEkoU?si=TJDiaKMUPvpkhSfy

Coding Convention의 목적

문제: 코드는 쓸 일보다 읽을 일이 훨씬 많음. (심지어 혼자 작업해도)

협업자가 있다면 누가 누구의 코드를 보게 될지 모름.

새로운 협업자는 완전히 새로운 코드를 파악해야 함.

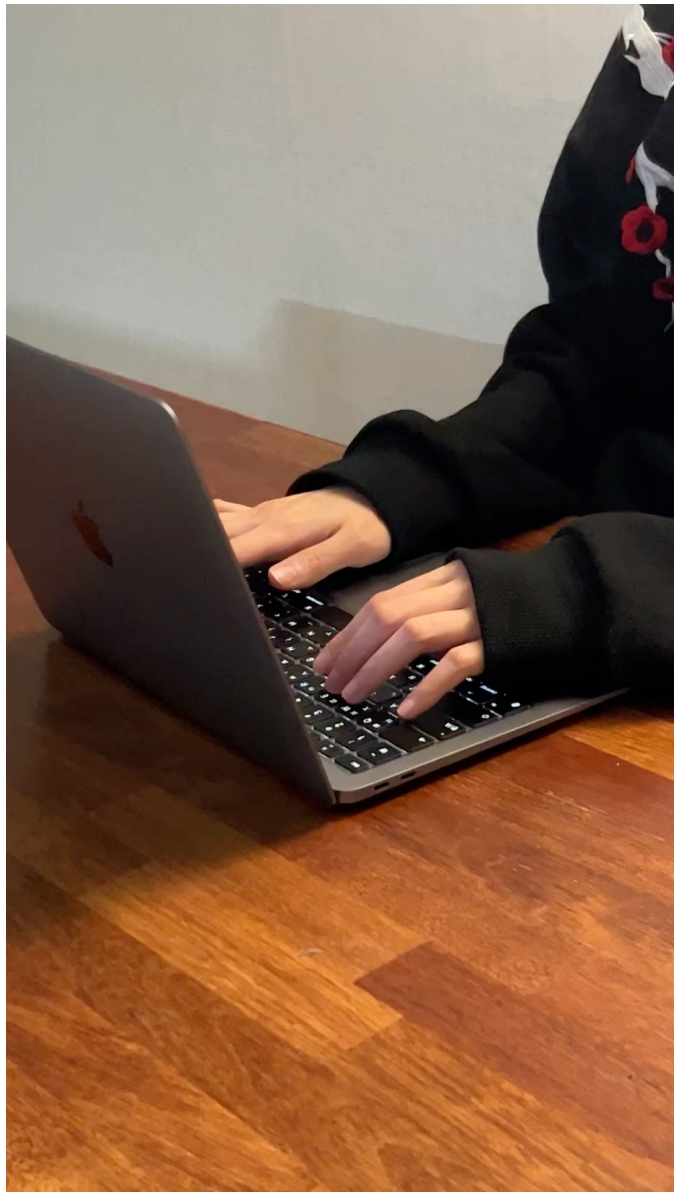
“Coding Convention을 정하자”

목적: (문제의 해결)

가독성이 좋은 코드를 쓰기 위해.

작업자가 달라도 읽는 방식을 일치시키기 위해.

유지와 보수가 용이하도록 코드와 소스를 구성하기 위해.



Formatting

"코드를 어떤 모양으로 쓸 것인가?"

코드의 모양은 읽을 때 이정표 내지는 신호가 됨.

Examples:

- 적절한 공백과 들여쓰기로 가독성을 높이기
- 괄호의 위치를 일관적으로 하기
- 너무 긴 줄은 나누어 쓰기

```
//look into the eye of wisdom and if you are lucky you shall get
//your code back correctly compiled
//idk how it works but it does

            package flipcoins; public   class
    monetaryCoin extends Coin   { int   appropratelyNamed;
  public monetaryCoin(int   amount)  {   appropratelyNamed
 =amount;       }                              public    void
       setValue                                (int     Value)
{       this.              /**/                 appropratelyNamed
=     Value;}              /**/                   public     int
 getIntValue               /*[]*/                  ()          {
      return               /*[]*/                   appropratelyNamed;
}      public              /*[]*/                  String     add
(monetaryCoin              /**/                  []    mc) { int
total  =  this.            /**/                appropratelyNamed;
if(mc.length >=0)                             { for (monetaryCoin
mc1 : mc) { total                             += mc1.getIntValue
 ( ) ;  }  }  return   Integer .toString  (total ) ;   }  public
   String    getValue() { String result = Integer.toString
      ( appropratelyNamed   ) ;  return  result ;  }  }
```

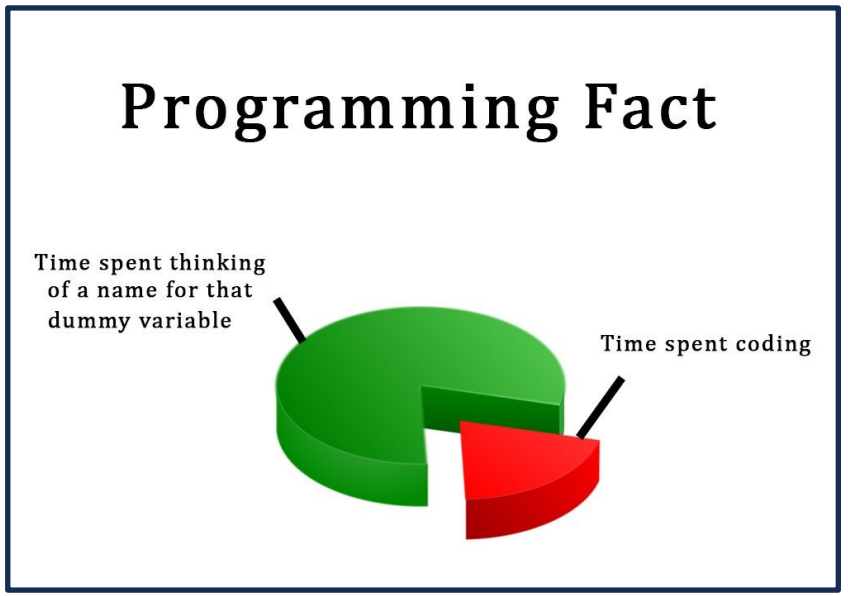
Naming

"심볼의 이름을 어떻게 정할 것인가?"

좋은 이름을 쓰면 무얼 하는 코드인지 알기 쉬움.

Examples:

- (코드를 쓰던 때의) 나만 알아보는 이름 피하기
- 심볼의 종류(상수, 변수, 클래스 등)를 한눈에 알 수 있도록 하기
- 흔한 종류의 메서드들의 이름 형식을 통일하기



Code Structure

"코드를 어떻게 구성할 것인가?"

코드 구성이 일관적이면 원하는 부분을 찾기 쉬움.

Examples:

- 함수의 정의를 쓰는 순서를

    일의 단위가 큰 것부터 vs 일의 단위가 작은 것부터

- 클래스의 정의에서

    필드 먼저 vs 메서드 먼저

- 클래스의 메서드 정의는

    getter끼리 setter끼리 vs 각 getter와 setter 쌍은 함께

Project Directory Structure

"프로젝트 디렉터리를 어떻게 구성할 것인가?"

프로젝트 디렉터리 구성 방식이 일관되면

오래 된 프로젝트를 다루거나, 새로운 협업자가 프로젝트를 다룰 때 편리함.

Examples:

- 프로젝트를 위해 만든, 여러 군데서 활용할 수 있는 간단한 일을 수행하는 부분을 따로 모아 한 파일/디렉터리에 담기
- 프로젝트에 포함되어 있지만 코드가 아닌 리소스를 한 디렉터리에 담기
- 디렉터리와 파일 이름을 짓는 규칙 정하기

Practices of Programming Principles

"중요한 원칙들을 어느 선까지, 또 어떤 방식으로 고수하도록 할 것인가?"

다양한 디자인/프로그래밍 원칙들 (전산학부 "소프트웨어 공학 개론" 수업에서 배울 수 있습니다.)

- **Separation of Concerns** 프로그램을 작은 부분으로 나누어 각각이 하나의 작은 일을 하도록 하기
- Keep it Simple, Stupid (KISS) 단순하게 설계하기
- Don't Repeat Yourself (DRY) 반복을 줄이기

이런 원칙들은 대개 정도껏 적용해야 하며(그러므로 **DO NOT BE DOGMATIC!**), 적용하는 방식 또한 다양하다.

그 구체적인 적용 계획

또...

앞서 나온 것들 외에도 많은 것들을 coding convention으로 정할 수 있다.

가장 좋은 Coding Convention이란 게 있나?

https://sandimetz.com/blog/2017/6/1/why-we-argue-style

결국 다 정해야 한다

Formatting (이것 안에도 정해야 할 것이 많다.)

Naming (이것 안에도 정해야 할 것이 많다.)

Code Structure (이것 안에도 정해야 할 것이 많다.)

Project Directory Structure (ㅇㄱ ㅇㅇㄷ ㅈㅎㅇ ㅎ ㄱㅇ ㅁㄷ.)

Practices of Programming Principles (ㅇㅇㅈㅎㅁ)

...

"이 많은 걸 언제 다 정해서 약속합니까?"

그러게 말이에요 ☹

여러... 줄기 빛: Style Guide

"이 convention 잡솨봐요"

앞서 등장한 다양한 coding conventions를 (내가) 하나씩 정하기에는 무리가 있다. 그런데...

"우리 협업자들 뿐 아니라 다들 우리 convention을 쓴다면 참 편리할 텐데..."

"우리 convention 정말 훌륭한데... 널리 알리고 싶은데..."

"한 땀 한 땀 만든 convention인데... 모두 이걸 따르면 좋을 텐데..."

몇몇 이들이 conventions를 세트 메뉴처럼 조합해 그 설명서를 공개했으니... 그것이 바로 **style guide**. ☺

(실은, 공개해야만 style guide는 아니고, 조직 내에서 보기 위해 만든 conventions 설명서를 style guide라 부른다.)

Style Guide의 예시

언어	Style Guide	Link
C++	Google	https://google.github.io/styleguide/cppguide.html
Python	PEP 8	https://peps.python.org/pep-0008/
JavaScript	AirBnB	https://github.com/airbnb/javascript
Rust		https://doc.rust-lang.org/nightly/style-guide/
Scala		https://docs.scala-lang.org/style/
OCaml		https://ocaml.org/docs/guidelines
Go	Effective Go	https://go.dev/doc/effective_go

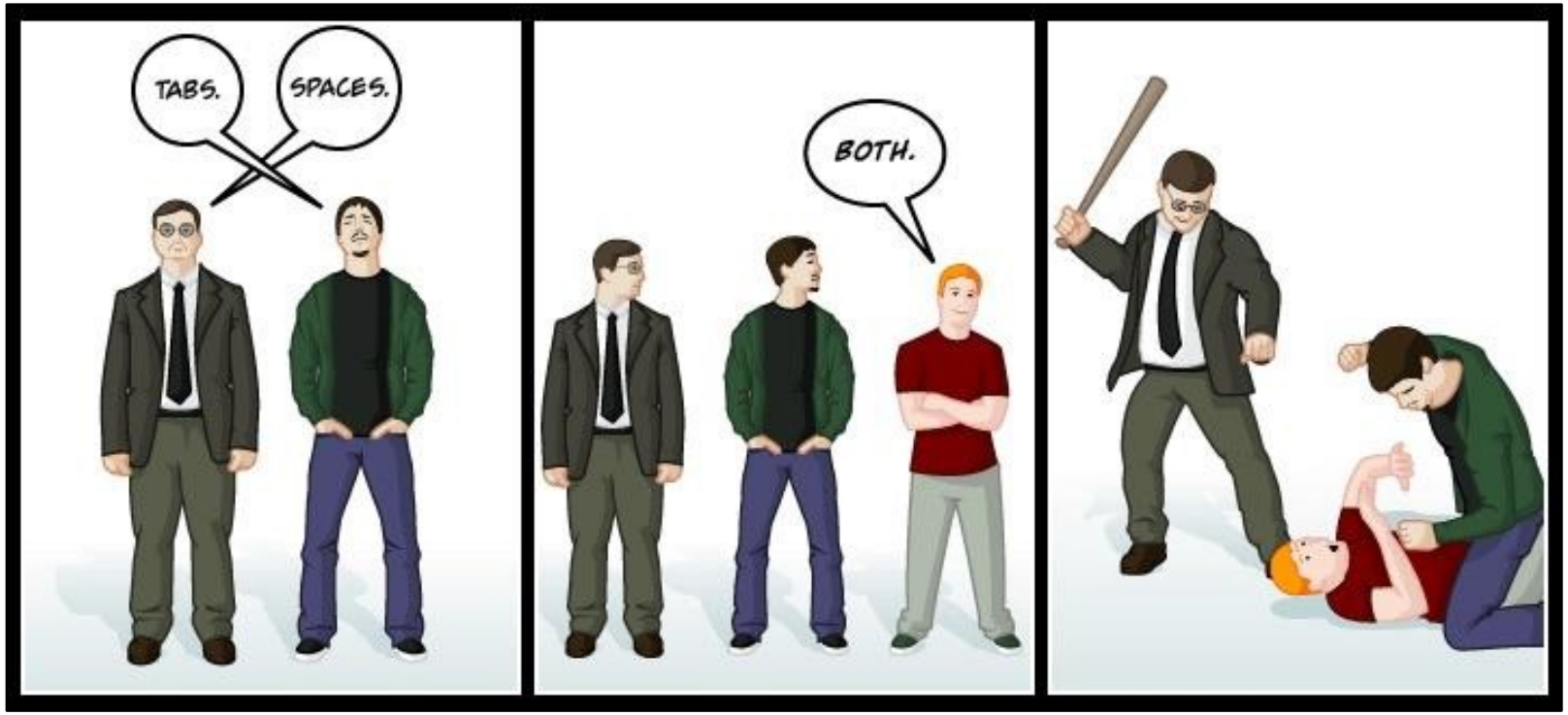
Case

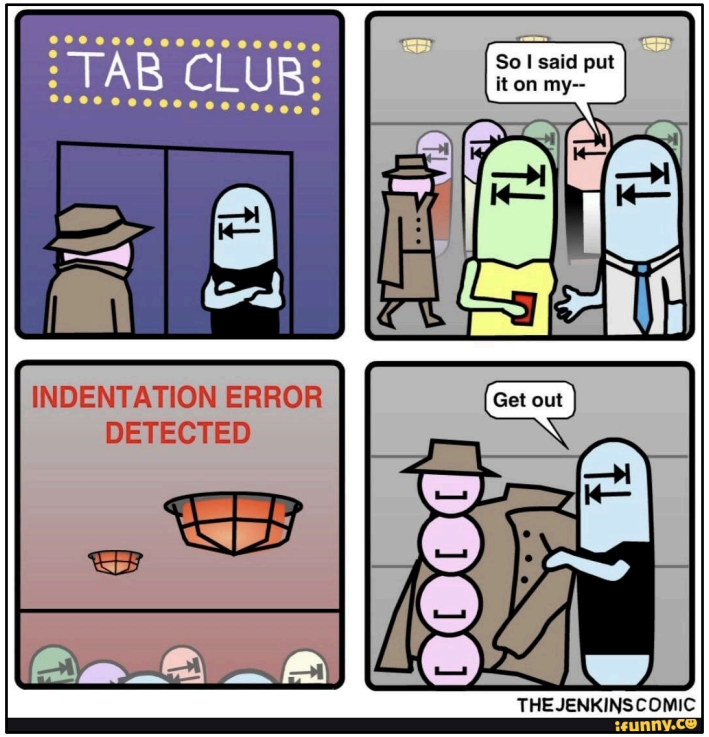
심볼이나 매크로 상수 등 이름을 정할 때 쓰인다. 대문자/소문자, 단어 구분 방법에 따라 나뉜다.

예시	명칭	사용처
`namingCase`	(lower) camel case	C#, OCaml, Scala에서 변수 이름 등
`NamingCase`	(upper) camel case, Pascal case	대부분의 언어에서 클래스/구조체 이름, C#에서 메서드 이름, Scala에서 상수 이름 등
`naming_case`	snake case	Python, Rust에서 변수/함수 이름 등
`NAMING_CASE`	screaming case, macro case, app caps	상수 이름, C/C++ 매크로 이름 등
`naming-case`	kebab case, CSS case	디렉터리/파일 이름, CSS property 이름 등
`Naming-Case`	HTTP header case	HTTP 헤더 이름 등

Indent Character

들여쓰기를 TAB으로 하느냐, space 여러 개로 하느냐.





Indent Size

들여쓰기를 얼마나 할 것인가. 대부분 2칸에서 8칸(Linux 커널 소스) 안으로 한다.

대부분은 4칸을 사용하고,

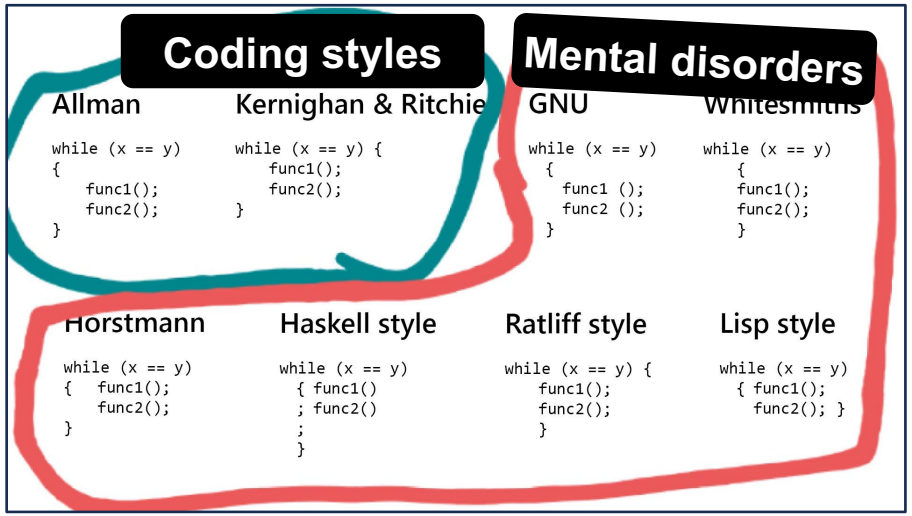
들여쓰기로 들어가는 깊이가 깊은 상황이 많은 언어들은 2칸을 사용하기도 한다. (예시: JavaScript)

들여쓰기 목적에 따라 혼용하는 경우도 있다. (예시: Google C++ Style Guide)

Indentation Style

들여쓰기를 어떻게 할 것인가.

코드를 읽기 쉽게 만드는 아주 기본적인... 예의범절에 가깝다...



Allman

```
while (x == y)
{
    func1();
    func2();
}
```

Kernighan & Ritchie

```
while (x == y) {
    func1();
    func2();
}
```

GNU

```
while (x == y)
  {
    func1 ();
    func2 ();
  }
```

Whitesmiths

```
while (x == y)
    {
    func1();
    func2();
    }
```

Horstmann

```
while (x == y)
{   func1();
    func2();
}
```

Haskell style

```
while (x == y)
  { func1()
  ; func2()
  ;
  }
```

Ratliff style

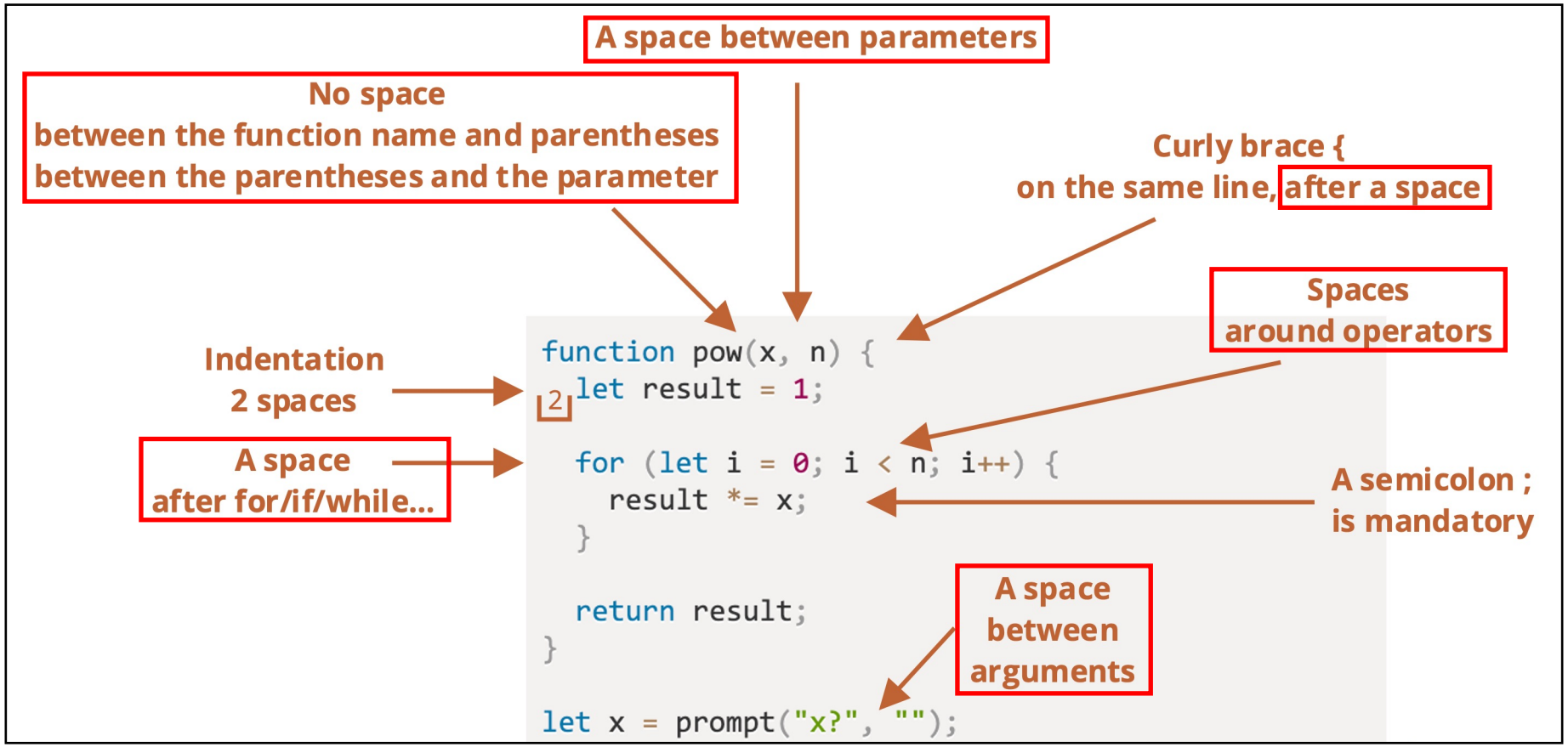
```
while (x == y) {
    func1();
    func2();
    }
```

Lisp style

```
while (x == y)
  { func1();
    func2(); }
```

Spaces

가독성을 높이기 위해 공백 넣기 혹은 넣지 않기.



Maximum Line Length

너무 긴 줄을 금지하는 규칙. 대표적인 Style Guide들은 다음과 같이 줄 길이를 제한하고 있다.

언어	Style Guide	Maximum Line Length
C++	Google	80
JavaScript	Douglas Crockford	80
Python	PEP 8	79
Java	Android	100
Ruby	AirBnB	100
PHP	PSR-2	120
R	Google	80
Go	Effective Go	없음 (줄 길이 무제한)

Project Directory Structure

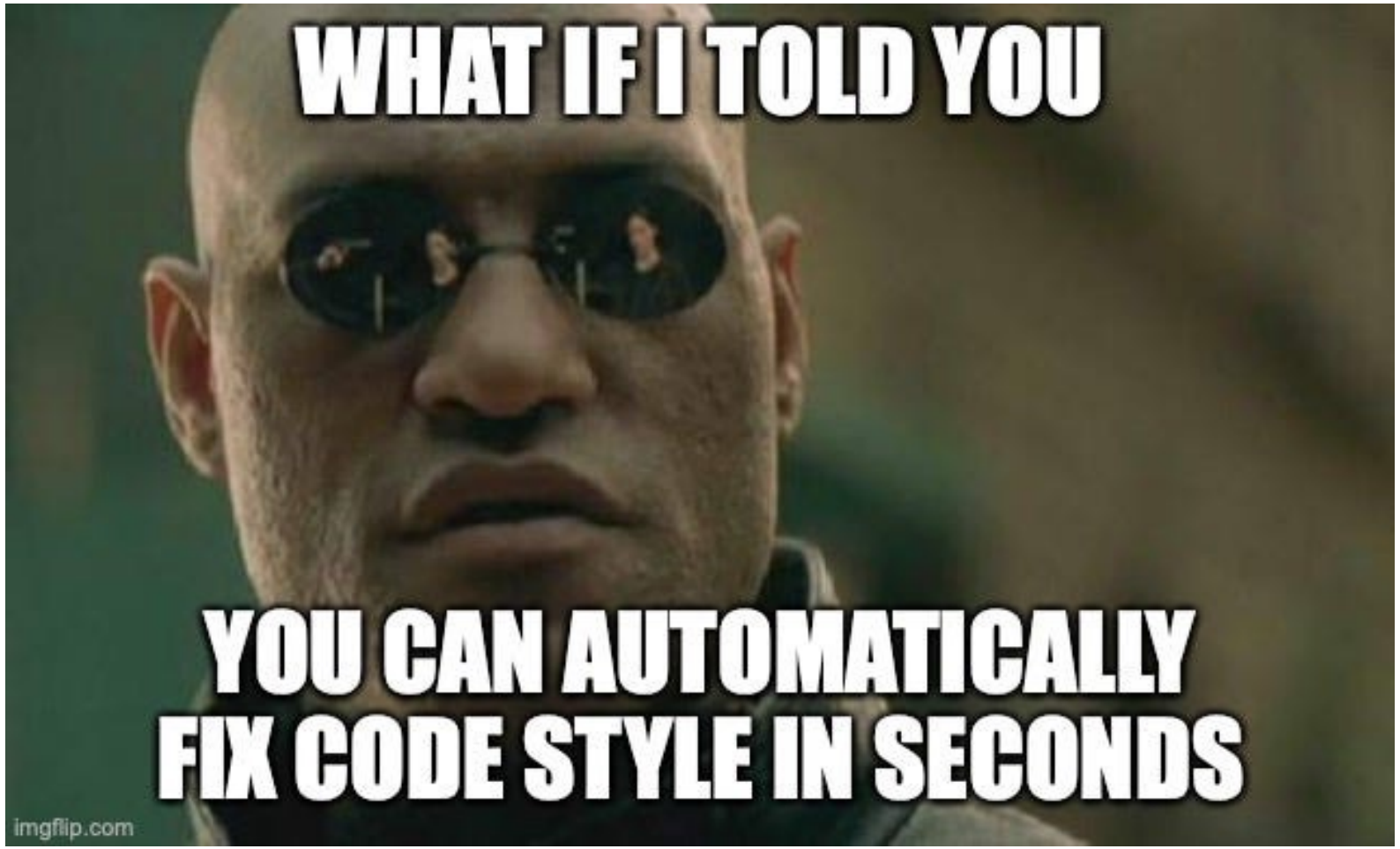
언어가 추천하는 가이드가 있거나, 빌드 시스템이 추천하는 가이드가 있을 것이므로 그대로 따라가면 될 것.

찾아 봤는데 없다면...

> 해당 언어로 만든 **다른 프로젝트를 구경해 보고 모방하거나**
>
> **스스로 잘 설계하는 걸로.** (그 구조에 대한 설명은 Documentation으로 해야 한다.)

Formatter & Linter

[시연]



혼자서 작업할 때는

Indentation style같은 건 둘 다 써 보고 편한 걸로 쓴다. (한 프로젝트 안에서는 같은 걸 씁시다.)

적절한 formatter를 사용한다. (Format on Save 강력 추천)

괜찮다면 적절한 linter를 사용한다.

Documentation의 예시 코드나, 인터넷 검색으로 보게 된 남의 코드가 나와 좀 다른 convention을 쓰는 것 같다?

→ 해당 부분과 관련된, 해당 언어의 추천 convention이 있는지 찾아본다.

협업에 참여하게 됐을 때는

사용하는 coding convention이 있는지 문의한다/알아본다.

있다면 사용하고 있는 formatter나 linter가 있는지 문의한다/알아본다.

있다면 같은 formatter와 linter를 사용한다.

없다면 사용을 제안하는 건 어떨까?

없다면 style guide가 작성되어 있는지 문의한다/알아본다.

있다면 style guide를 읽는다.

~~있다면 formatter나 linter 제작을 제안하는 건 어떨까?~~

없다면 공개된 널리 쓰이는 style guide를 따르는 걸 제안하자.

안 된다면 ~~나가는 건 어떨까?~~ style guide를 작성하는 걸 제안하자.

없다면 공개된 널리 쓰이는 style guide를 따르는 걸 제안해보자.

안 된다면 ~~나가는 건 어떨까?~~ coding convention을 만들고 style guide를 작성하는 걸 제안하자.

어떤 경우든 style guide가 있다면 꼭 읽고 지키는 것을 추천합니다. Formatting 관련 내용만 있는 건 아니니까요. 근데 하다 보면 눈치껏 적당히 되긴 함.

B

Documentation

우리가 겪는/겪을 상황

선대개 MATLAB 과제

프밍기 과제 in Python

문DS 과제 in Java

허기홍 교수님의 OCaml

강지훈 교수님의 Rust

류석영 교수님의 Scala

KENS의 C++

이런 모르겠는 언어들...

또

소공개에서 프론트엔드 개발을 하기 위한 React

교수님과 조교님들이 과제를 위해 손수 만들어 오신 라이브러리

heap memory를 trace하고 싶은데? heaptrack

백엔드 개발을 하는데 credential을 별도 파일로 보관하고 싶은데? dotenv

이런 프레임워크, 라이브러리와 CLI Tool들...

설명서를 봐야겠네

Documentation이란?

우리말로 하면 “문서화”인데, 소프트웨어에서 “Documentation”은

해당 소프트웨어의

(1) 사용 방법과

(2) 사용 효과,

더 넓게는

(3) 작동 원리까지

기술하는 것, 혹은 기술한 그 문서 자체

를 의미한다. 그야말로 소프트웨어의 사용 설명서.

"우리 사용 설명서 안 읽잖아요"

사용 설명서를 읽는 것들은?

정확한 사용법 없이 사용하기엔

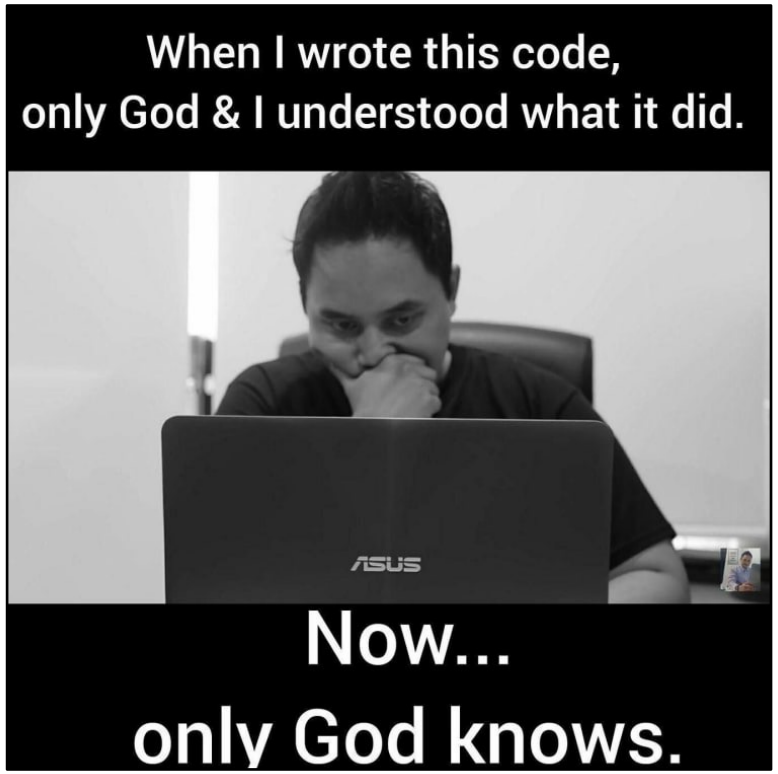
감도 안 잡히거나

위험하거나

손해를 보게 되는

그런 물건들.

"잊히지 않기 위해서"



소프트웨어도 사람에게서 잊혀지면 죽는다.

Documentation이 없거나 부실하면?

사용자: "쓰기 힘들어", "쓰기 꺼려져"

→ 사용자 감소, 중요한 일에 사용하지 않음

개발자: "수정 힘들어", "버그 패치 어려워", "기능 추가/수정/삭제 어려워"

→ 버그 패치 늦어짐, 시대에 뒤쳐짐, 개발자 감소, 개발 의지 꺾임, 개발자 구인 난항

➔ 악순환 ➔ 사람들의 기억에서 사라짐 ➔ 소프트웨어의 죽음

(그마저도 Documentation이 잘 되어 있으면 사람들 모아서 적당히 부활이 가능한데, 아니라면 부활도 쉽지 않음.)

네 가지 Documentation

https://documentation.divio.com/

종류	초점	필요 기능	형태
Tutorials	학습	새로 온 사람이 시작할 수 있도록 해야 함	강의
How-to guides	목표	특정 문제를 어떻게 해결하는지 보여야 함	Step-by-step 안내
Reference	정보	소프트웨어의 동작 방식을 기술함	정확하고 간결한 내용 (describe)
Explanation	이해	배경 지식과 작동 원리 등을 설명해야 함	광범위한 정보에 대한 설명문 (explain)

	Useful for studying	Useful for working
Practical steps	Tutorials	How-to guides
Theoretical knowledge	Explanation	Reference

좋은 예시: Django

https://docs.djangoproject.com/en/4.2/#how-the-documentation-is-organized

네 가지 접근 방식을 모두 제공하고 있다.

README

프로젝트 디렉터리에, 바이너리라면 압축 파일의 최상위 디렉터리에 존재하는 파일로,

해당 소프트웨어에 대한 대표적인 정보(버전, 의존성, 개략적인 사용 방법 등)가 쓰여 있다.

- 예시: https://github.com/kaist-plrg/esmeta

주로 다음과 같은 언어로 작성된다.

마크업 언어	README 파일 이름	
Markdown	`README.md`	(나는 이걸 권장함.)
Plain Text	`README` 혹은 `README.txt`	
reStructuredText	`README.rst`	

Generated Documentation

프로젝트에서 사용 중인 documentation generator와

프로젝트의 소스로부터 생성할 수 있는 documentation.

라이브러리의 경우 공식 API 문서가 되는 경우가 많다.

- 예시: https://github.com/antony-jeong/torch-fx-rs/tree/dev

언어	Documentation generator
Python	Sphinx
C/C++	Doxygen (다른 언어들도 쓸 수 있긴 함.)
Java	Javadoc
Rust	rustdoc
Scala	Scaladoc

공식 웹 사이트

[별도 장소에서 user에게 공개되는 documentation]

언어, 프레임워크, 라이브러리나 CLI tool이 그에 대한 공식 웹 사이트를 가지고 있다면,

대부분은 해당 소프트웨어에 대한 documentation가 게시되어 있다.

- 예시: https://vuejs.org/

https://www.scala-lang.org/

Non-docs Comments

[소프트웨어의 소스에서 maintainer에게 공개되는 documentation]

User에게 공개되지 않는 private 메서드나 필드에 대한 설명

복잡한 로직에 대한 부연 설명

진행 중인 작업 혹은 해야 하는 일에 대한 표시

써야 할지 말아야 할지 의견이 나뉘는 소재임.

많이 복잡한 로직이라면 comments를 써야 한다는 의견과

많이 복잡한 로직은 human-readable한 코드로 충분히 표현 가능하다는 의견이 있음.

Documentation인가?

7가지 원칙

짧은 문장을 사용한다.

간결한 문장을 사용한다.

능동형 문장을 사용한다.

시각 자료를 활용한다.

두괄식 구성을 사용한다.

사용자를 중심으로 하는 서술 시점을 유지한다.

문제 해결형 서술 기법을 사용한다.

바로 접할 수 있는 것 & 과제에서 쓸 수 있는 것

- README 작성하기
- Documentation generator 활용하기
- Comments 작성하기

Python 프로젝트로 Documentation 다루기

읽기 실습:

Python library “`sh`”를 사용해 “`ls -la`”를 해 주는 파이썬 프로그램 만들기

쓰기 실습:

Comments와 `README.md` 작성하기

맺음말