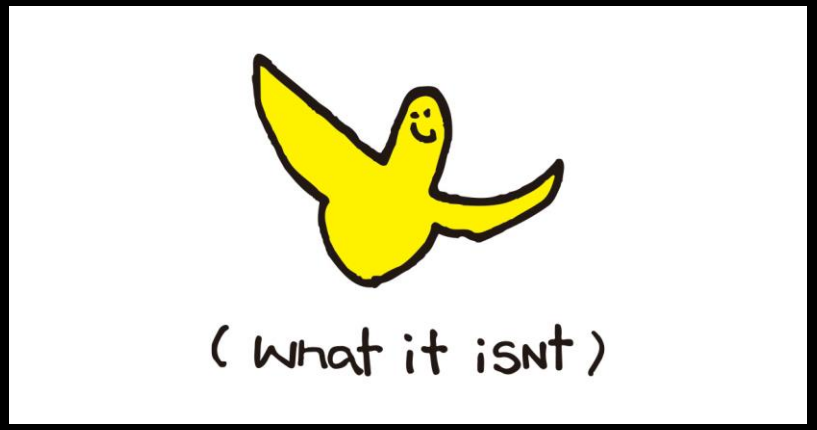
신비롭고 아름다운 Lisp 이야기

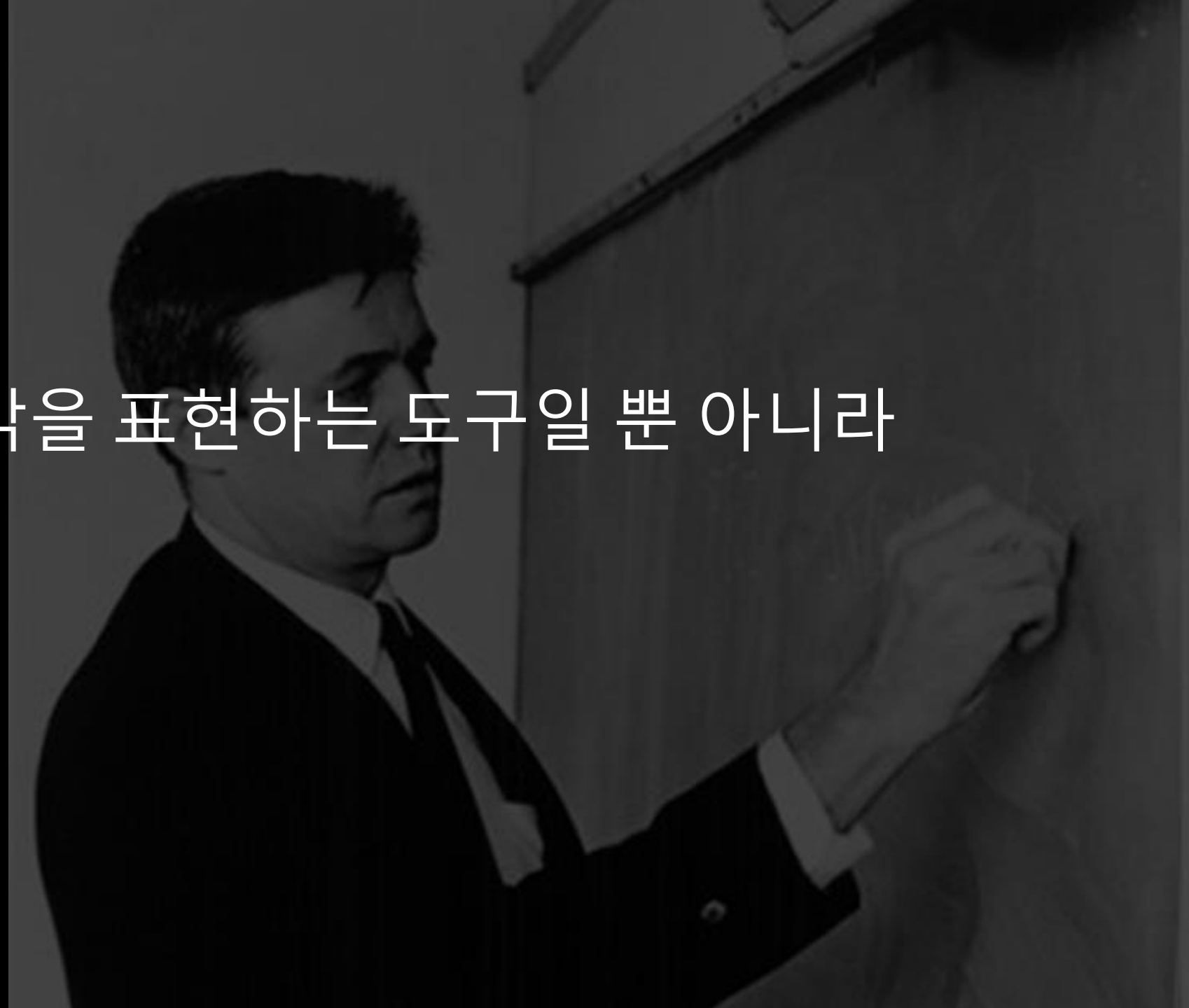
이정연

이 발표의 목적이 아닌 것

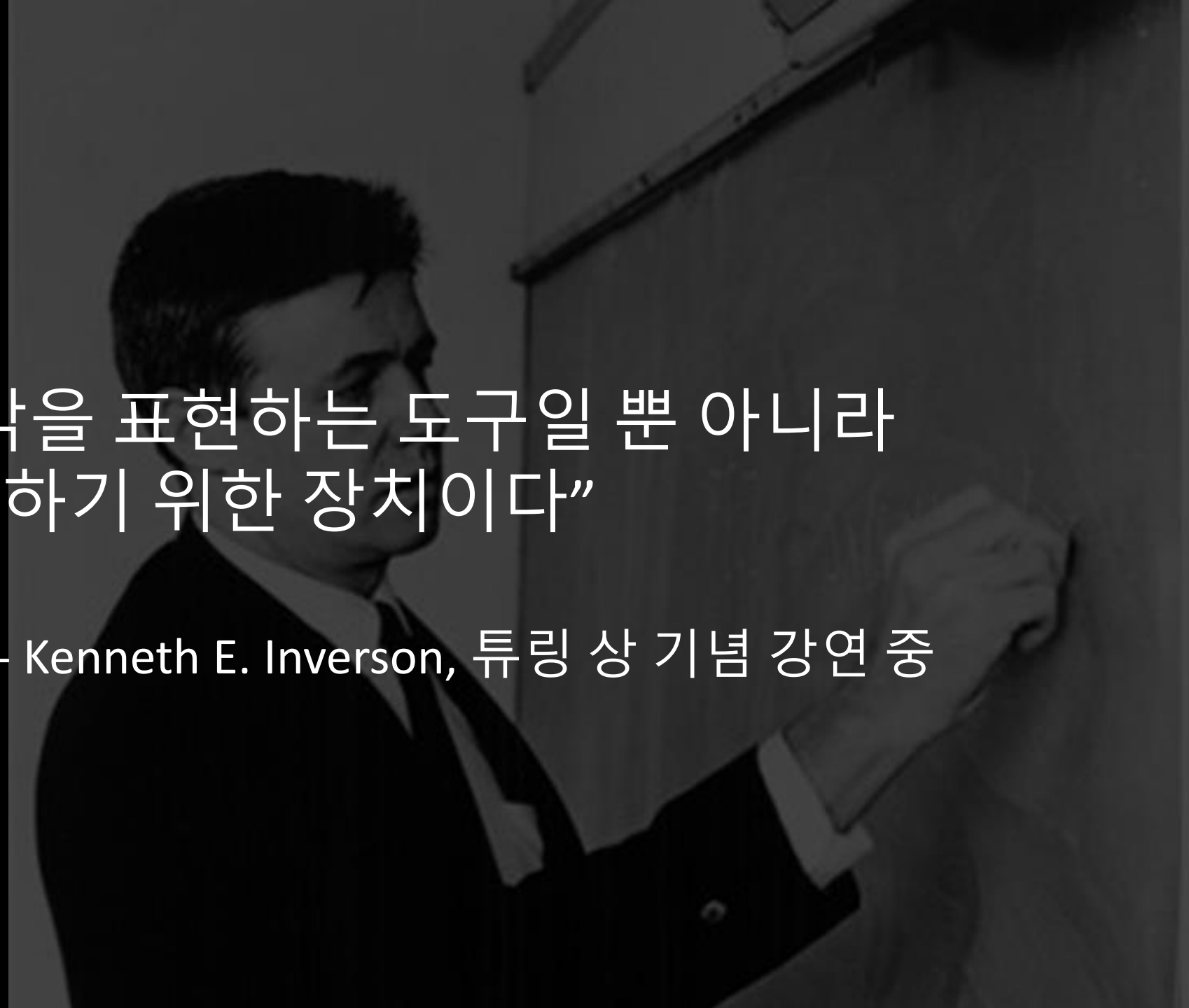


- 리스프를 가르치는 것
- 함수형 프로그래밍을 홍보하는 것
- 코드를 이해시키는 것
- 변명하는 것

LISP은 어떤 언어인가?

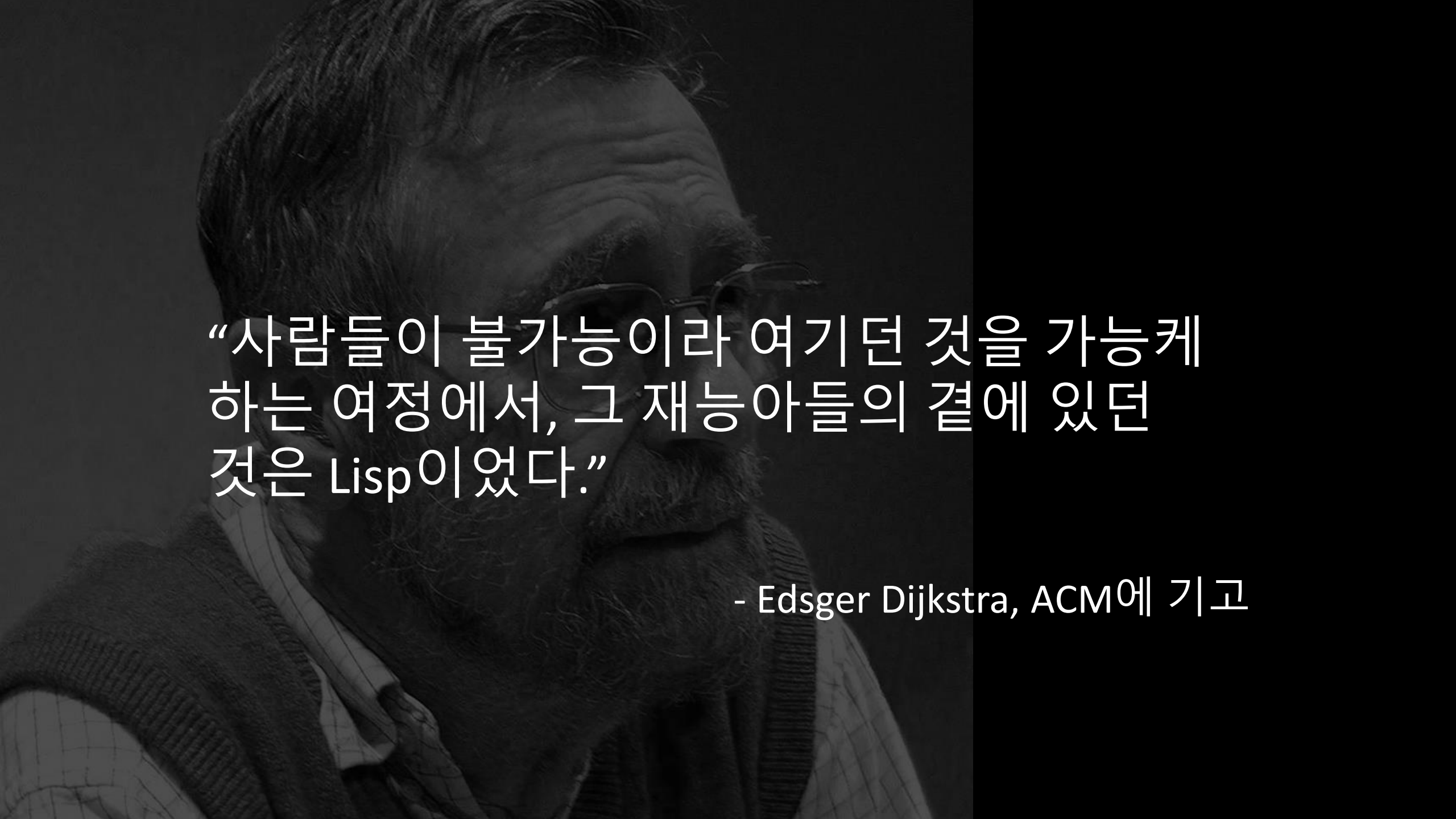


“언어는 생각을 표현하는 도구일 뿐 아니라



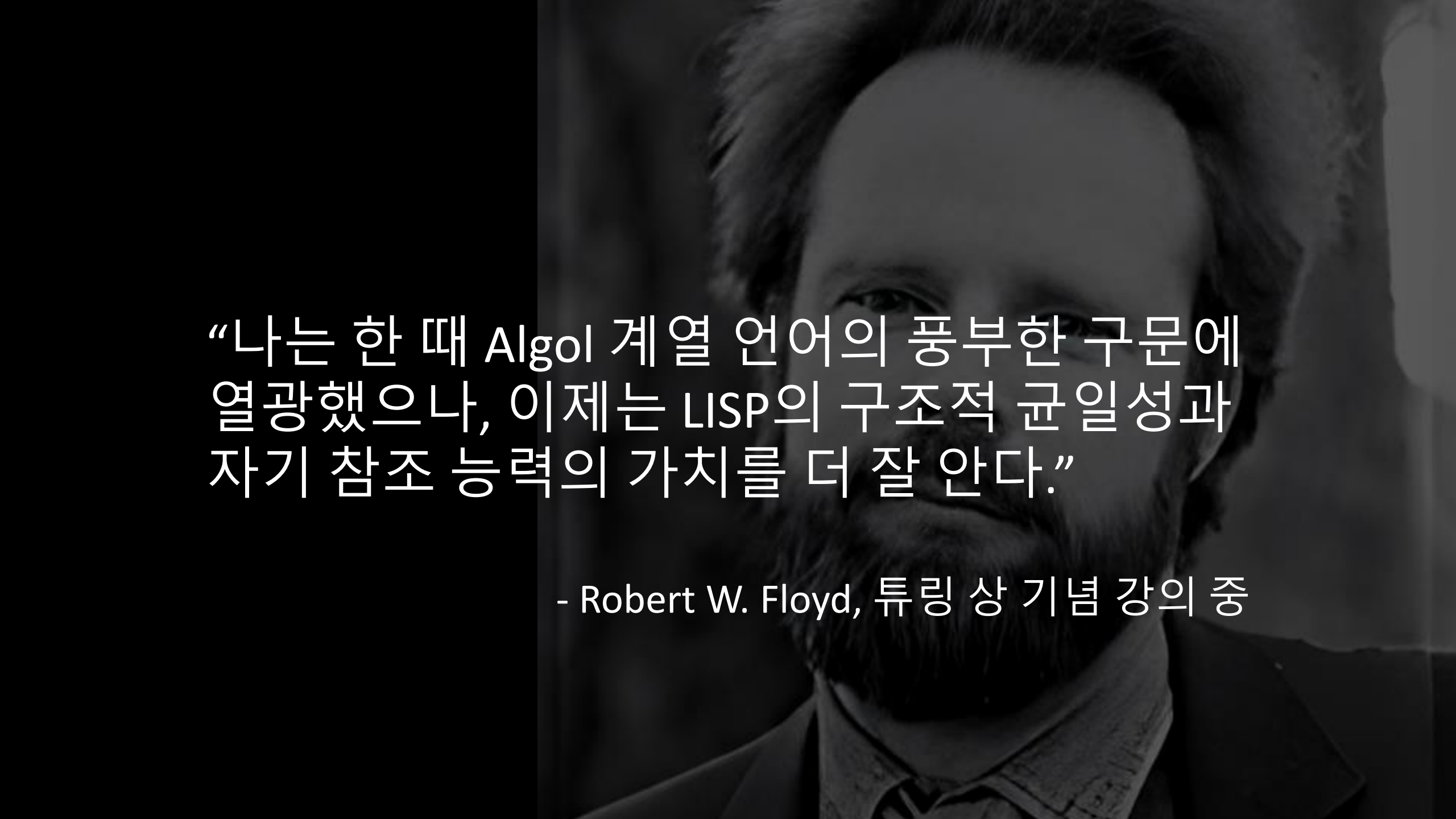
“언어는 생각을 표현하는 도구일 뿐 아니라
인간이 사유하기 위한 장치이다”

- Kenneth E. Inverson, 튜링 상 기념 강연 중



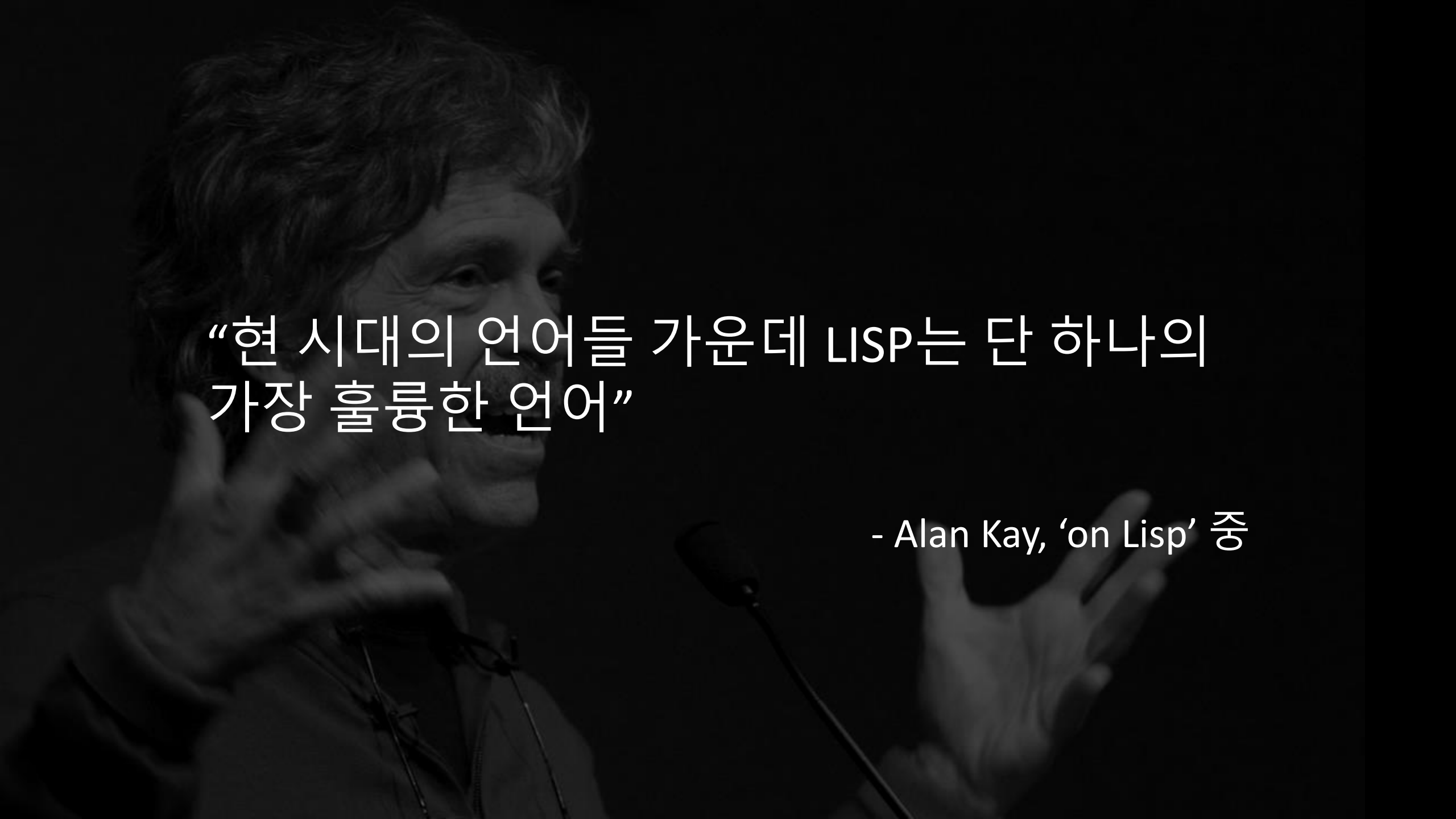
“사람들이 불가능이라 여기던 것을 가능케 하는 여정에서, 그 재능아들의 곁에 있던 것은 Lisp이었다.”

- Edsger Dijkstra, ACM에 기고



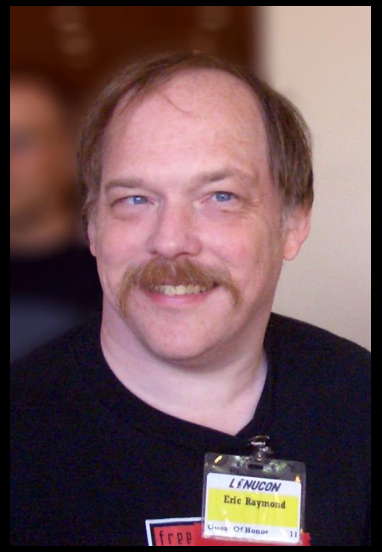
“나는 한 때 AlgoI 계열 언어의 풍부한 구문에 열광했으나, 이제는 LISP의 구조적 균일성과 자기 참조 능력의 가치를 더 잘 안다.”

- Robert W. Floyd, 튜링 상 기념 강의 중


“현 시대의 언어들 가운데 LISP는 단 하나의
가장 훌륭한 언어”
- Alan Kay, ‘on Lisp’ 중

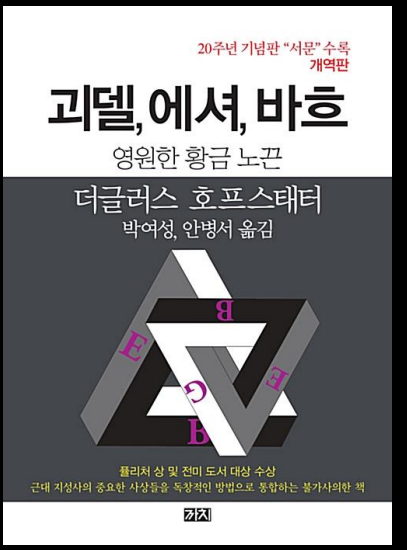
“리스프(LISP)는 마침내 도달하게 될 심오한 개발 경험을 위해 배울 가치가 있다. 그 경험은 여러분의 앞으로의 프로그래머로서의 기간동안 더욱 훌륭한 프로그래머가 되게 해주는 경험일 것이다.”

- Eric S. Raymond, ‘해커가 되려면’ 중



“가장 중요하고 흥미로운 컴퓨터 언어 가운데 하나는, AlgoI이 만들어진 시기에 함께 만들어진 존 매카시의 언어 LISP이다.”

- Douglas Hofstadter, ‘괴델, 에셔, 바흐’ 중



언어를 배우는 것은
정말로 가치가 있는가?

가장 흔한 오해[1]

한 언어를 잘 배워두면 다른 언어를 배울 때 문제가 없다

가장 흔한 오해[2]

한 언어를 잘 배워두면 다른 언어를 배울 때 문제가 없다

- C를 배우고 나서 Java를 배울 때 어려움이 없었는가?

가장 흔한 오해[3]

한 언어를 잘 배워두면 다른 언어를 배울 때 문제가 없다

- C를 배우고 나서 Java를 배울 때 어려움이 없었는가?
- **코드를 읽는 것과 쓰는 것은 아주 다른 이야기이다**

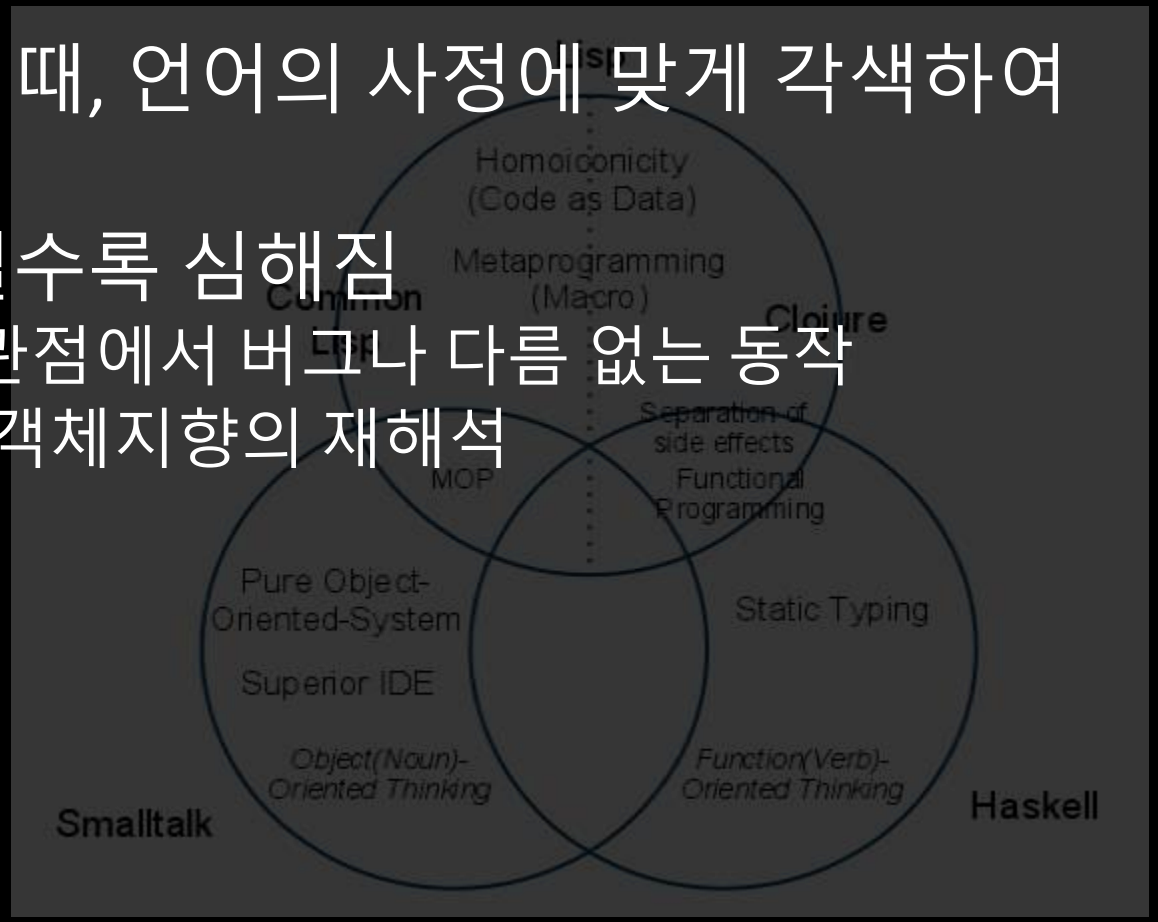
가장 흔한 오해e

한 언어를 잘 배워두면 다른 언어를 배울 때 문제가 없다

- 비슷해 보여도 다른 동작들이 아주 많음
  - C++의 템플릿 vs. Java의 제네릭스
  - Call by Value/Reference/Pointer/Sharing
  - Go와 Java의 인터페이스
- 한 언어를 통달했다고 이 모든 것을 쉽게 이해할 수 있을 리 없다

Teaching Language의 필요성[1]

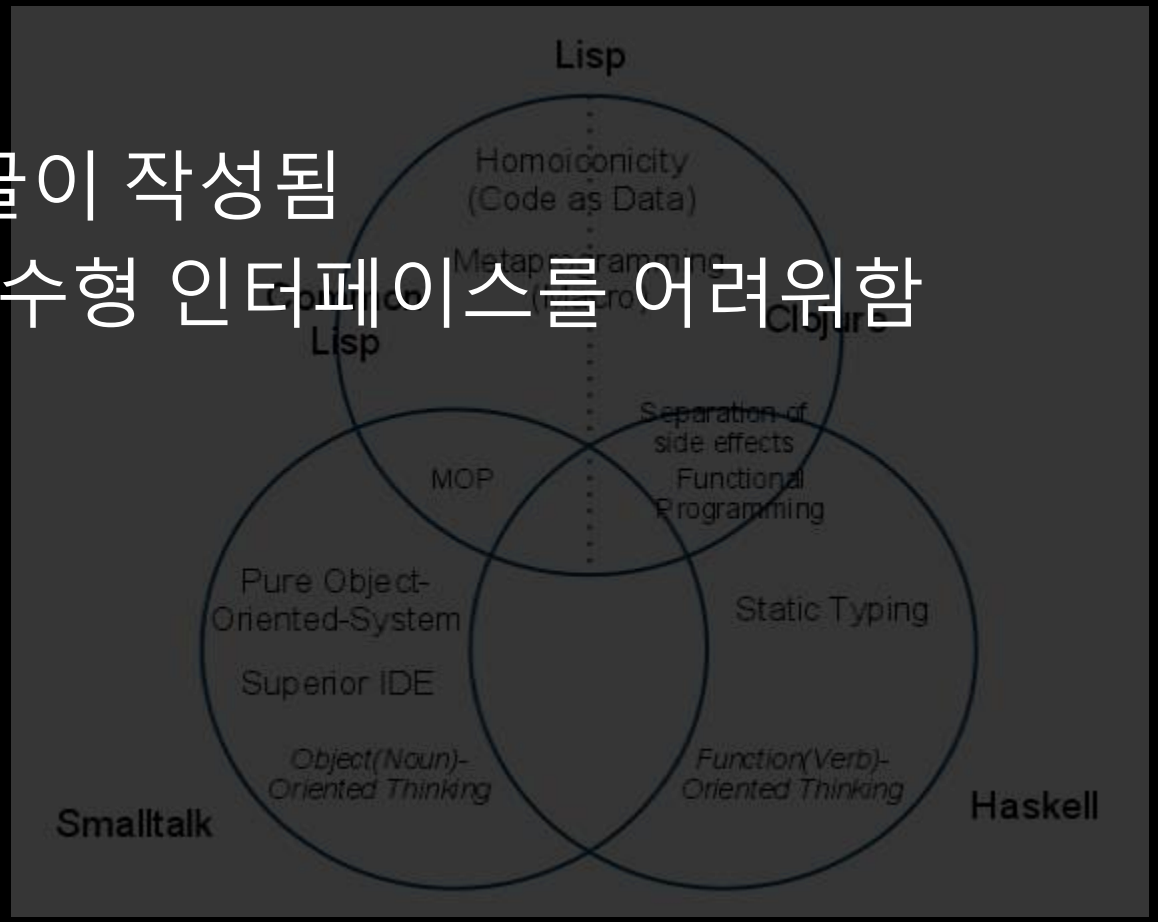
- 이론적인 개념을 언어에 구현할 때, 언어의 사정에 맞게 각색하여 옮겨지기 마련
- 특히 산업 현장에 가까운 언어일수록 심해짐
  - E.g. Java의 Type erasure – 언어론 관점에서 버그나 다름 없는 동작
  - E.g. C++의 friend – 재앙에 가까운 객체지향의 재해석



Teaching Language의 필요성[2]

‘자바 8.0 Lambda, 왜 좋나요?’

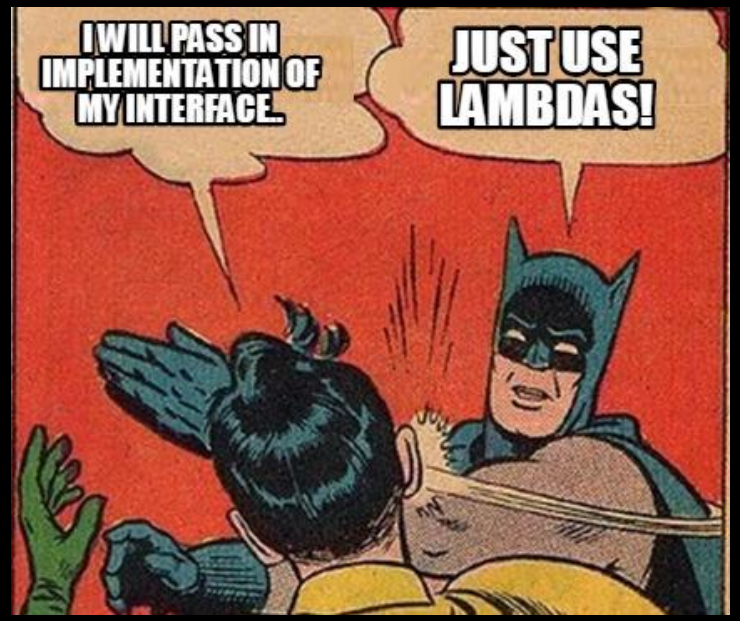
- 이에 대해 설명하는 수 천개의 글이 작성됨
- 현재까지도 여전히 사람들은 함수형 인터페이스를 어려워함



Teaching Language의 필요성[3]

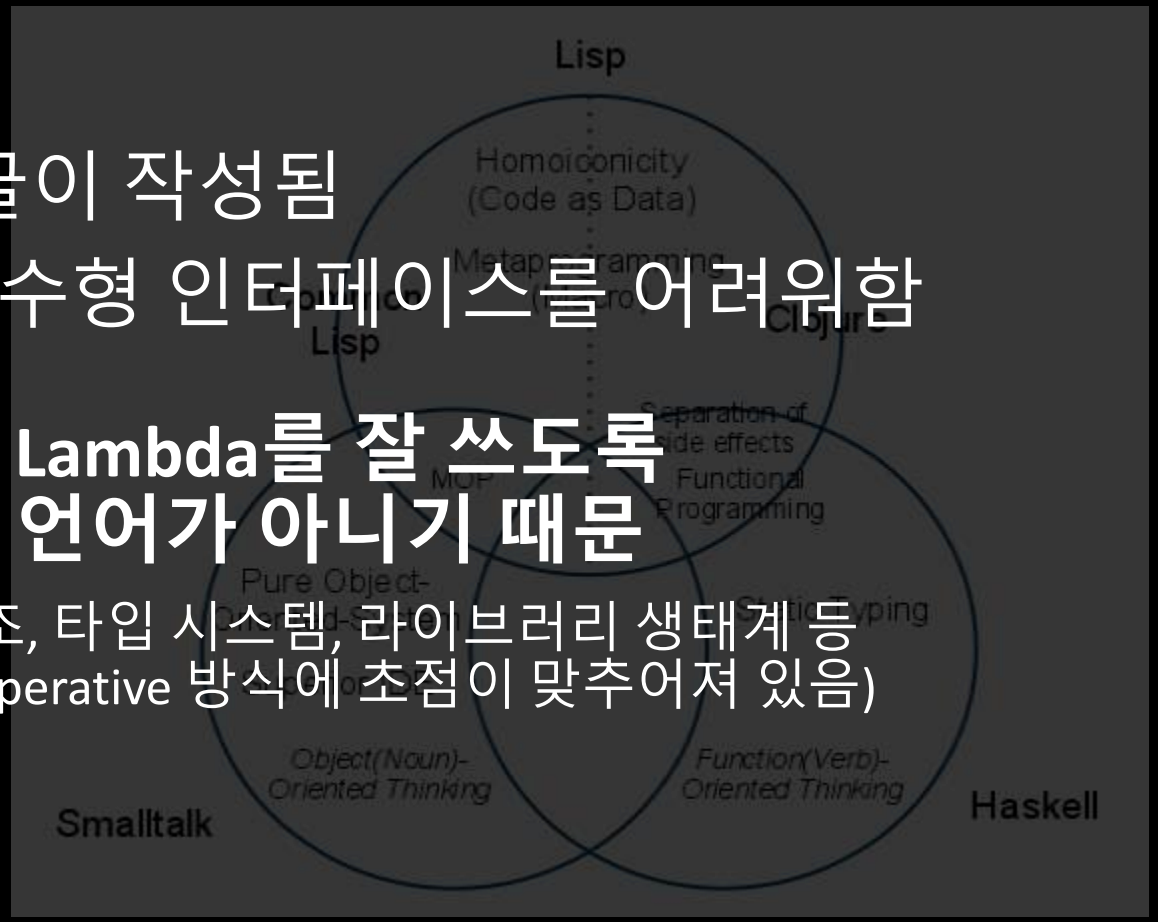
‘자바 8.0 Lambda, 왜 좋나요?’

- 이에 대해 설명하는 수 천개의 글이 작성됨
- 현재까지도 여전히 사람들은 함수형 인터페이스를 어려워함



⇒ **자바는 Lambda를 잘 쓰도록 고안된 언어가 아니기 때문**

(기본 자료구조, 타입 시스템, 라이브러리 생태계 등 모든 것이 Imperative 방식에 초점이 맞추어져 있음)

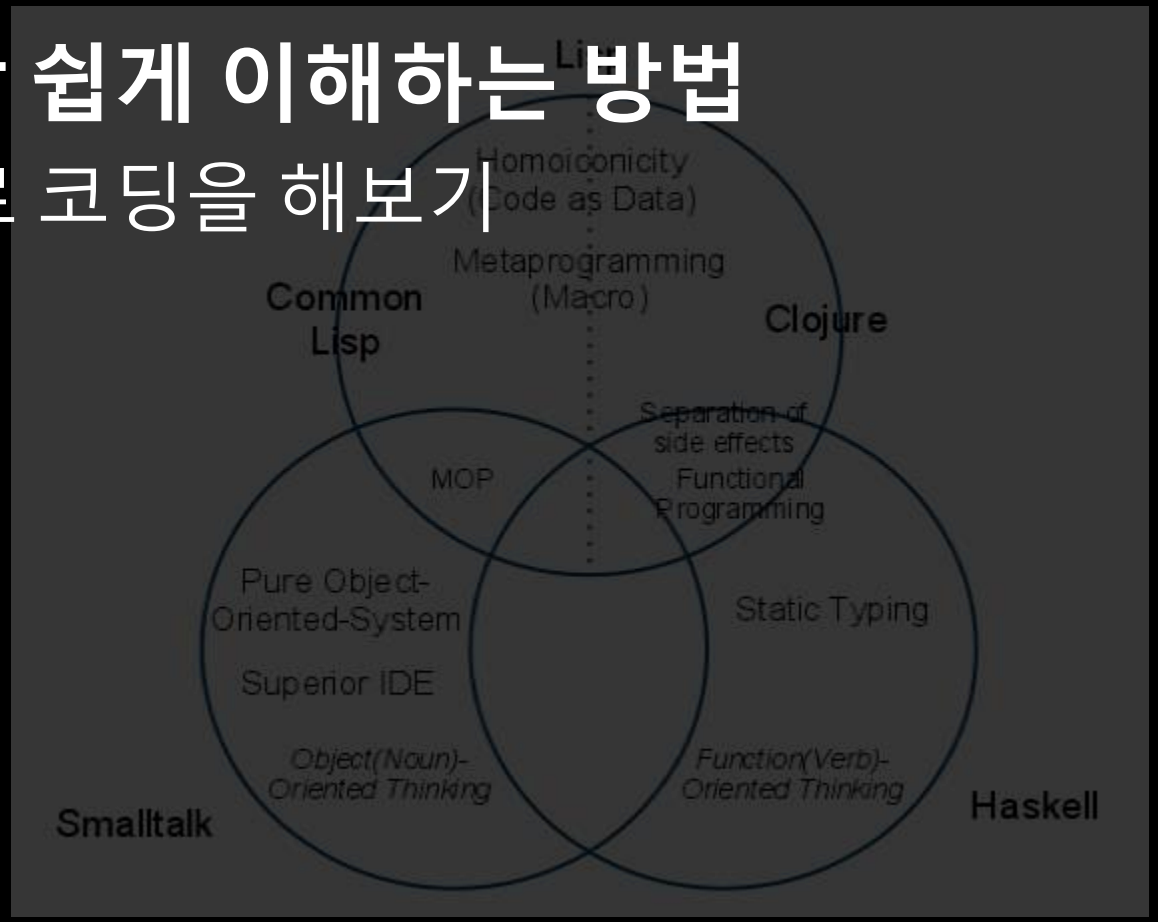


Teaching Language의 필요성[4]

프로그래밍 언어 개념을 가장 쉽게 이해하는 방법

: 그 개념을 가장 잘 설명한 언어로 코딩을 해보기

- 오래 걸리지 않나요?
  - 바른 길이 가장 빠른 길입니다

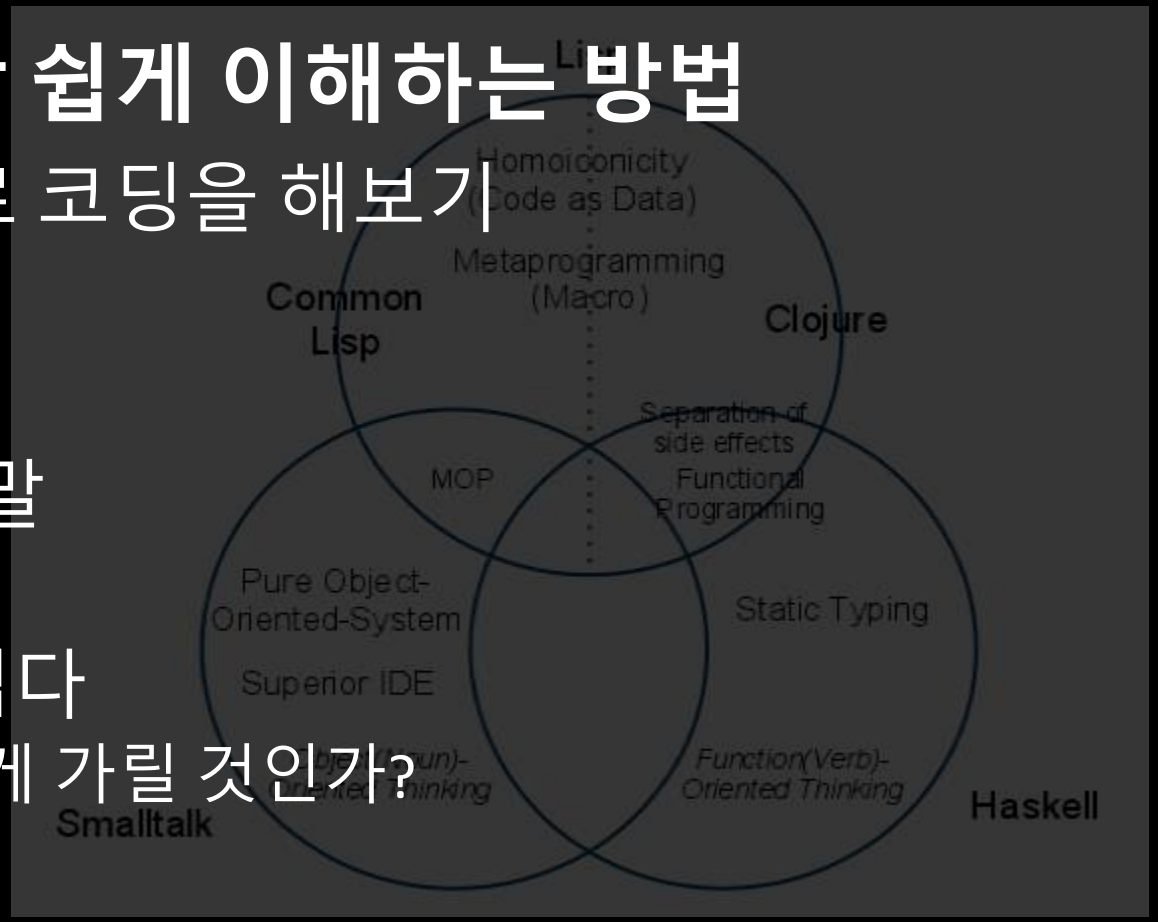


Teaching Language의 필요성[5]

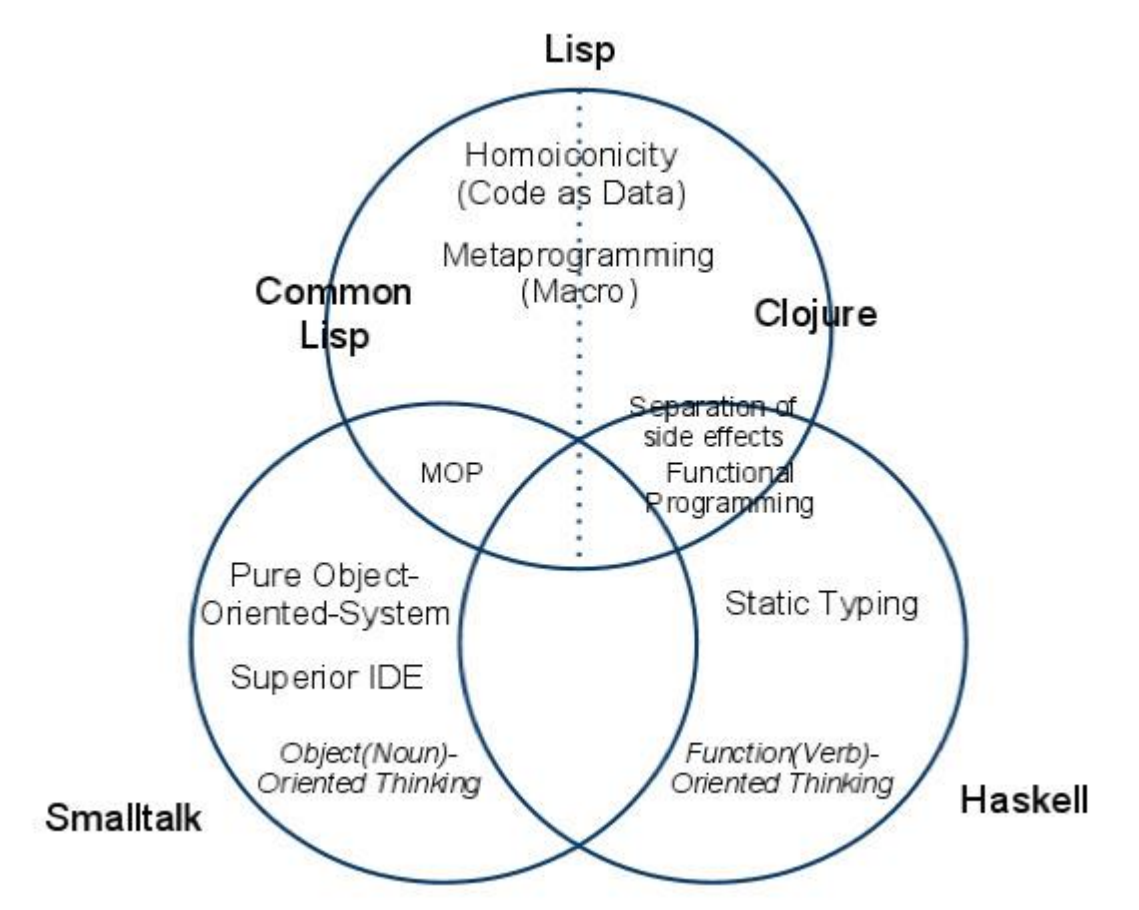
프로그래밍 언어 개념을 가장 쉽게 이해하는 방법

: 그 개념을 가장 잘 설명한 언어로 코딩을 해보기

- 오래 걸리지 않나요?
  - 바른 길이 가장 빠른 길입니다
- 대충 배워서 쓰면 흔히 듣게 될 말
  - '아 그거 그렇게 하는 거 아닌데...'
- 대충 배우는 것은 생각보다 어렵다
  - 수 천개의 글 가운데 옥석을 어떻게 가릴 것인가?
  - **제대로 된 '평가'를 내리기 어렵다**



Teaching Language의 필요성[e]



Lisp을 배우면 얻게 되는 것

Lisp에는 무엇이 들어있는가[1]

단순한 문법

동적 타입 시스템

인터프리터

함수형 패러다임

자기동형성

심볼 타입

가비지컬렉션

Lisp에는 무엇이 들어있는가2

단순한 문법

동적 타입 시스템

인터프리터

함수형 패러다임

자기동형성

심볼 타입

가비지컬렉션

Lisp에는 무엇이 들어있는가e

단순한 문법

동적 타입 시스템

인터프리터

자기동형성

함수형 패러다임

메타 프로그래밍

인터랙티브 프로그래밍

심볼 타입

가비지컬렉션

메타 프로그래밍[1]

- Lisp의 문법

  `(<op> <data> <data> …)`

  e.g. `(+ 1 2 3)` `(or T NIL)`

메타 프로그래밍2

- Lisp의 문법

  `(<op> <data> <data> …)`

  e.g. `(+ 1 2 3)` `(or T NIL)`
- 특별한 연산 규칙이 없음

메타 프로그래밍3

- Lisp의 문법

  `(<op> <data> <data> …)`

  e.g. `(+ 1 2 3)` `(or T NIL)`
- 특별한 연산 규칙이 없음
- 연산자는 함수 또는 특수 구문
  - 함수: 인자로 값을 받는다
  - 특수 구문: 인자로 코드를 받는다

    (special-form을 번역한 것)

메타 프로그래밍[4]

- 특수 구문은 어떤 코드를 실행하고, 어떤 코드를 실행하지 않을지, 어떻게 실행되어야 하는지, 그 실행 규칙을 정할 수 있다

e.g.
```
(if (> x 0)   “ok”        “fail”   )
```
c.f.
```
(if (> x 0) (print-ok) (print-fail))
```

메타 프로그래밍[5]

- 특수 구문은 어떤 코드를 실행하고, 어떤 코드를 실행하지 않을지, 어떻게 실행되어야 하는지, 그 실행 규칙을 정할 수 있다

  e.g. `(if (> x 0)  “ok”  “fail”  )`

  c.f. `(if (> x 0) (print-ok) (print-fail))`

⇒ 곧, 원하는 문법을 만들어낼 수 있다
(조건문, 반복문, short-circuit 연산자, …)

코드를 만들어내는 코드 - **메타 프로그래밍**

메타 프로그래밍6

- 언어 기능은 단지 구현의 문제
  - Object Oriented System이 필요한가?
    => Common Lisp Object System(CLOS)
  - 리스트, 배열, 해시테이블에 적용할 범용적 반복문이 필요한가?
    => LOOP Macro
  - Aspect Oriented Programming이 필요한가?
    => CLOS Meta-object Protocol

메타 프로그래밍7

- 언어 기능은 단지 구현의 문제
  - 뭐? JSX같은 게 가지고 싶어?
    => XMLisp

So a window can be defined like this:

```
(defvar *window*
  <window width="380"
          height="140">
    <button text="OK"/>
    <button text="Cancel"/>
  </window>)
```

메타 프로그래밍e

- 언어 기능은 단지 구현의 문제
  - 뭐? 명령형 스타일을 원해?
    => with-c-syntax

Using C syntax inside a Lisp function.

```
(named-readtables:in-readtable with-c-syntax:with-c-syntax-readtable)

(defun array-transpose (arr)
  (destructuring-bind (i-max j-max) (array-dimensions arr)
    #{
      int i,j;
      for (i = 0; i < i-max; i++) {
          for (j = i + 1; j < j-max; j++) {
              rotatef(arr[i][j], arr[j][i]);
          }
      }
    }#)
  arr)

(array-transpose (make-array '(3 3)
                             :initial-contents '((0 1 2) (3 4 5) (6 7 8))))
; => #2A((0 3 6) (1 4 7) (2 5 8))
```

인터랙티브 프로그래밍[1]

- 많은 사람이 눈치채지 못하는 Lisp의 중요한 가치 중 하나
  - 많은 언어확장 라이브러리들이 이 가치를 훼손하지 않고자 노력함

인터랙티브 프로그래밍2

- 많은 사람이 눈치채지 못하는 Lisp의 중요한 가치 중 하나
  - 많은 언어확장 라이브러리들이 이 가치를 훼손하지 않고자 노력함
- 인터프리터와 대화하면서 곧장 변경내용을 확인할 수 있는 기능

인터랙티브 프로그래밍[3]

- 많은 사람이 눈치채지 못하는 Lisp의 중요한 가치 중 하나
  - 많은 언어확장 라이브러리들이 이 가치를 훼손하지 않고자 노력함
- 인터프리터와 대화하면서 곧장 변경내용을 확인할 수 있는 기능
  - E.g. 클래스의 구성을 고치면 실행중인 객체들에 모두 적용됨

인터랙티브 프로그래밍[4]

- 많은 사람이 눈치채지 못하는 Lisp의 중요한 가치 중 하나
  - 많은 언어확장 라이브러리들이 이 가치를 훼손하지 않고자 노력함
- 인터프리터와 대화하면서 곧장 변경내용을 확인할 수 있는 기능
  - E.g. 클래스의 구성을 고치면 실행중인 객체들에 모두 적용됨
  - E.g. 현재 내가 보고있는 객체가 어떤 상태에 있는지 바로 확인할 수 있음

인터랙티브 프로그래밍[5]

- 많은 사람이 눈치채지 못하는 Lisp의 중요한 가치 중 하나
  - 많은 언어확장 라이브러리들이 이 가치를 훼손하지 않고자 노력함
- 인터프리터와 대화하면서 곧장 변경내용을 확인할 수 있는 기능
  - E.g. 클래스의 구성을 고치면 실행중인 객체들에 모두 적용됨
  - E.g. 현재 내가 보고있는 객체가 어떤 상태에 있는지 바로 확인할 수 있음
  - E.g. 재시작을 하지 않고도 함수의 변경사항을 적용할 수 있음

인터랙티브 프로그래밍e

- 많은 사람이 눈치채지 못하는 Lisp의 중요한 가치 중 하나
  - 많은 언어확장 라이브러리들이 이 가치를 훼손하지 않고자 노력함
- 인터프리터와 대화하면서 곧장 변경내용을 확인할 수 있는 기능
  - E.g. 클래스의 구성을 고치면 실행중인 객체들에 모두 적용됨
  - E.g. 현재 내가 보고있는 객체가 어떤 상태에 있는지 바로 확인할 수 있음
  - E.g. 재시작을 하지 않고도 함수의 변경사항을 적용할 수 있음

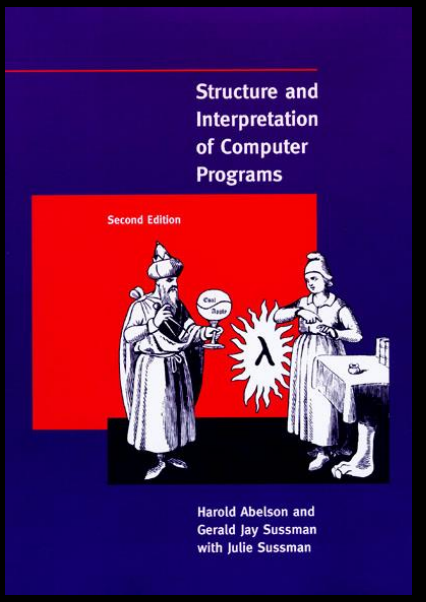
지금 보고 있는 객체에게 뭐가 문제인지 물어볼 수 있기 때문에
디버거를 켜서 재현하는데 시간을 쏟을 일이 줄어든다

고전을 읽는 언어로서의 Lisp

읽지 않기에는 아까운 책들이 너무 많다

- Lisp는 대략 30~40년간 연구자들을 사로잡은 언어
- 1960년대부터 2000년대 초반까지 업계에서 활발히 쓰였음
  - Esp. 심볼 기반/룰 베이스/전문가 시스템 형식의 고전 AI 연구에 많이 쓰임
- 그만큼 Lisp으로 쓰인 좋은 책들이 많다

컴퓨터 프로그램의 구조와 해석

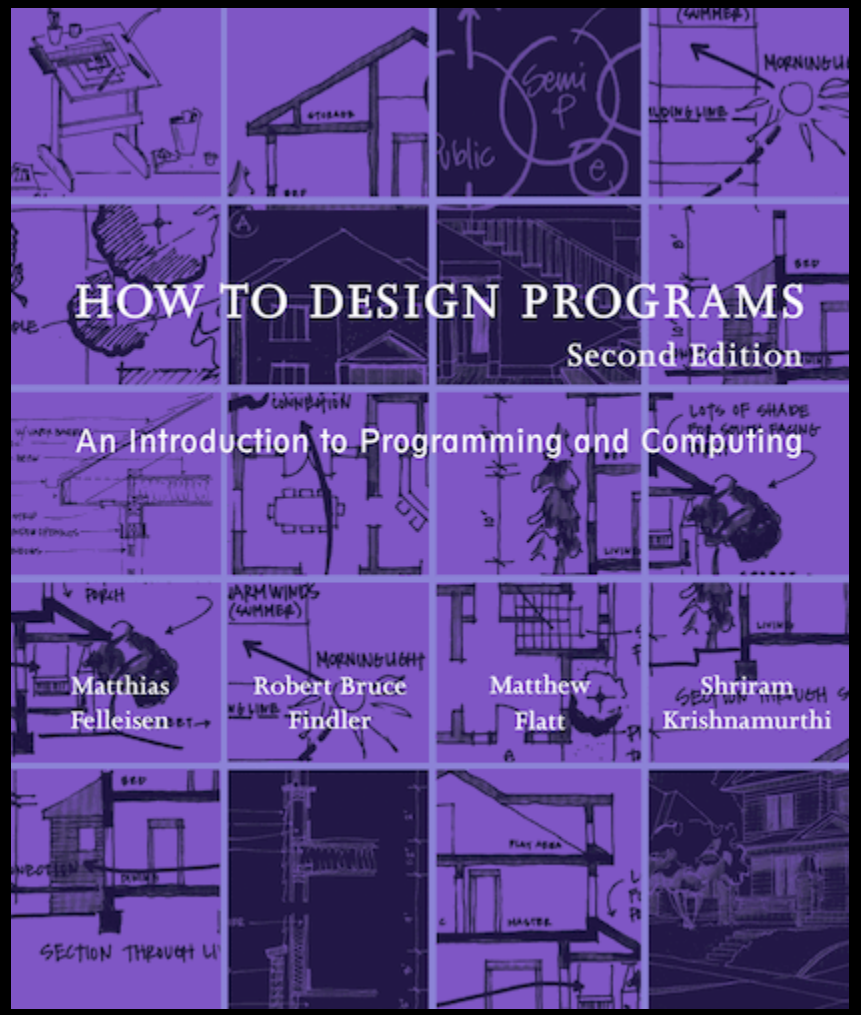


- 프로그램이란 무엇이고, 무엇을 할 수 있고, 어떻게 해야 하는가? 라는 질문의 해답
- 과거 MIT CS 학부 1학년 교재

“컴퓨터 프로그램의 구조와 해석을 읽고 진한 감동을 받아서 눈물까지 흘렸다는 내용을 자기소개서에 서사적으로 기술한 사람은 진정 프로그램을 이해하는 마니아임에 틀림없다.” – Joel on Software

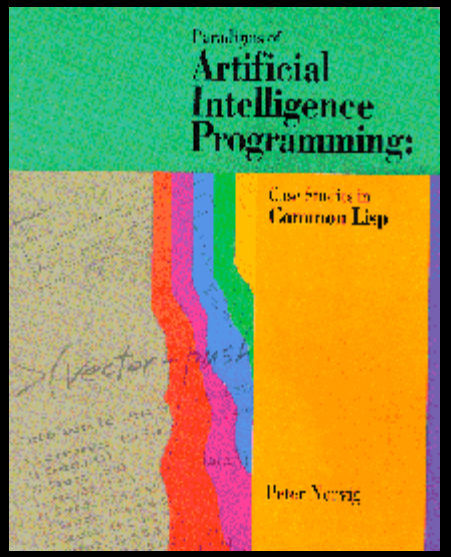
프로그램 디자인, 어떻게 할 것인가

- SICP와 비슷한 내용
- 좀 더 순한 맛의 연습문제
- 좀 더 친절한 구성
- 근데 SICP의 매운맛을 좋아하는 사람이 더 많은 듯?



PAIP

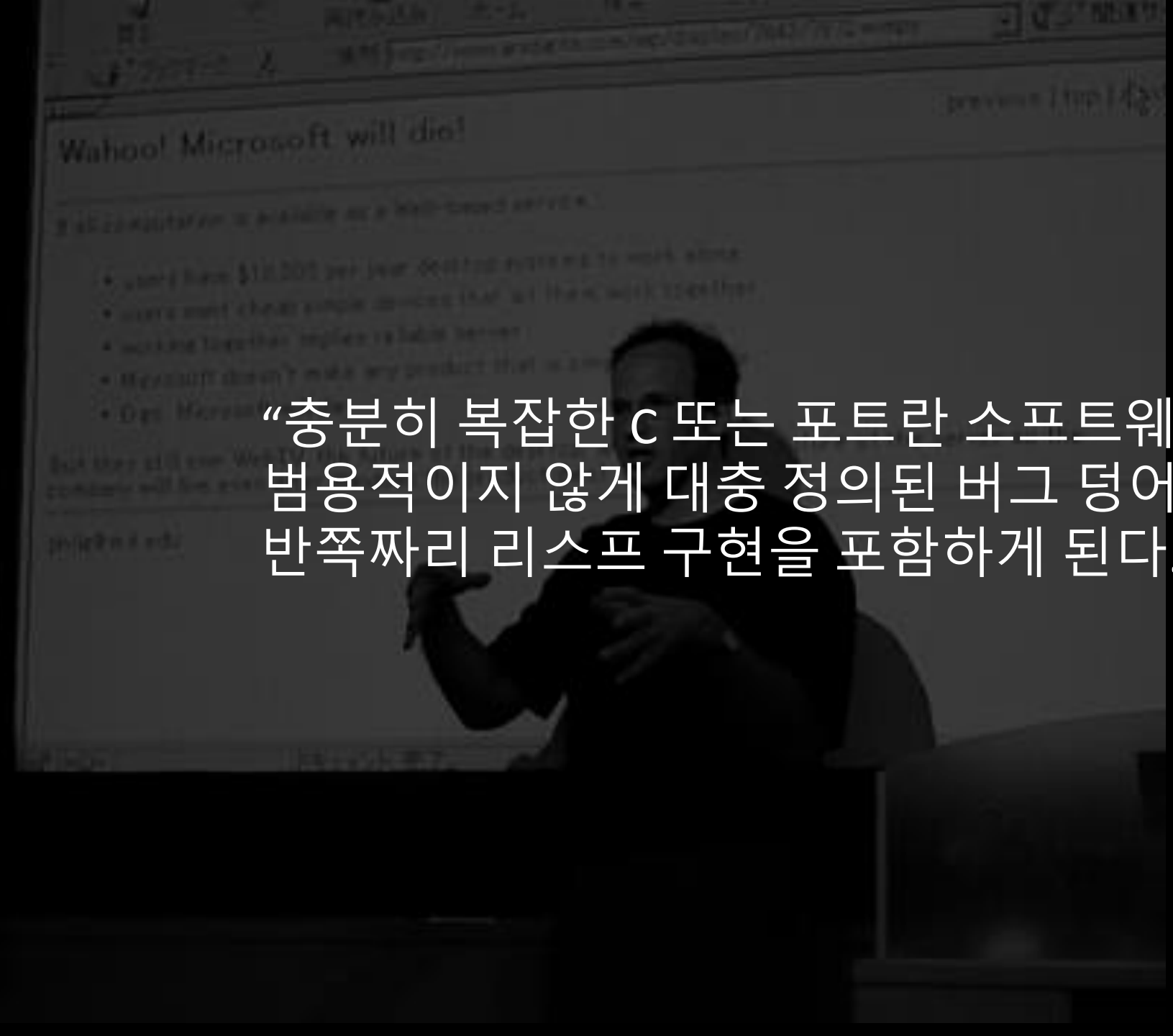
- 원제: Paradaigms of AI Programming
- AI의 거장 Peter Norvig이 쓴 책
- 휴리스틱 알고리즘, 기호 대수학, 논리 프로그래밍, 전문가 시스템, 자연어 처리 등 고전 AI 방법론의 주요 논점을 설명한 책
- 실용적인 알고리즘 설명서로도 손색이 없음



Lisp 책들

- Practical Common Lisp: 가장 대중적인 Lisp 입문서
- 만들면서 배우는 리습 프로그래밍: Lisp 입문서(국내 번역)
- On Lisp: 함수형 패러다임, 매크로 시스템을 자세히 다룬 책
- Let Over Lambda: 함수형 프로그래밍의 극적인 활용을 다룬 책
- Lambda Papers: 초창기 리습 연구자들의 언어학 고찰이 담긴 논문

여전히 살아있는 Lisp



“충분히 복잡한 C 또는 포트란 소프트웨어는 모두 범용적이지 않게 대충 정의된 버그 덩어리의 느린 반쪽짜리 리스프 구현을 포함하게 된다.”

- Philip Greenspun

대중적인 Lisp 구현체

- SBCL – 바이너리 컴파일, 타입 검사, Posix API 제공, 사실상 표준
- ECL – 임베디드 리스프, 모바일에서 사용 가능, 높은 호환성
- LispWork – 산업용 Lisp 구현체, IDE를 비롯한 편의기능 제공
- CMU-CL, CLisp, ABCL 등 목적에 따른 다양한 구현체들이 있음

활용되고 있는 Lisp 계열 언어들

- Emacs 설정 스크립트 언어
- GIMP 플러그인 언어
- AutoCAD 플러그인 언어
- Clojure – 범용 Lisp 방언
- Racket – 범용 Lisp 방언

감사합니다

Lisp을 시작하고 싶은 자는 나에게...