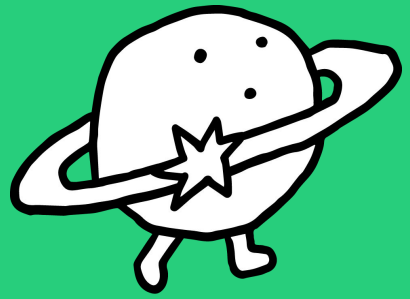
크론과 함께하는 재미난(難) 수업

우아한테크코스

#6
로또, 객체와 함수
피드백



#질문 🤔질문 잘하는 법

질문하는 법에 관한 질문을 드리려고 합니다.

저는 질문을 하기 전에 과연 질문을 통해 해결해야 하는지를 고민하는 편인 것 같아요.
근데 사실 이렇게 고민을 하다보면, '구글링으로 알아보면 해결되지 않을까?' '책에 나와있을 거 같은데?' '아직 스스로 고민을 덜해본 것 같은데?' 하는 생각이 드는 경우가 많아요!
그래서 결국에는 질문을 삼키게 되는 것 같습니다.
그런데 이런 태도가 더 빠르고 효과적으로 배울 수 있는 기회를 놓치게 되는 것 같기도 한 것 같아요.

#질문 돌아보기

딱 떨어지는 답을 구하기 위한 질문들

- ~하는 것이 맞을까요?
- ~는 정말 필요한가요?
- ~가 제 생각인데 이렇게 해도 될까요?
- ~라고 생각하는데 이 방식이 더 나을까요?
- ~중 무엇이 더 나을까요?
- ~와 같이 구현했는데 혹시 현업에서는 어떻게 진행하나요?

#질문 돌아보기

- 코치의 답변이 공신력있는 근거가 될 수 없다.
- 우아한테크코스에서 채택한 방법이 정답이 될 수는 없다.
- 현업의 경험을 묻는 것도 정답을 구하기 위해서가 아닐까?

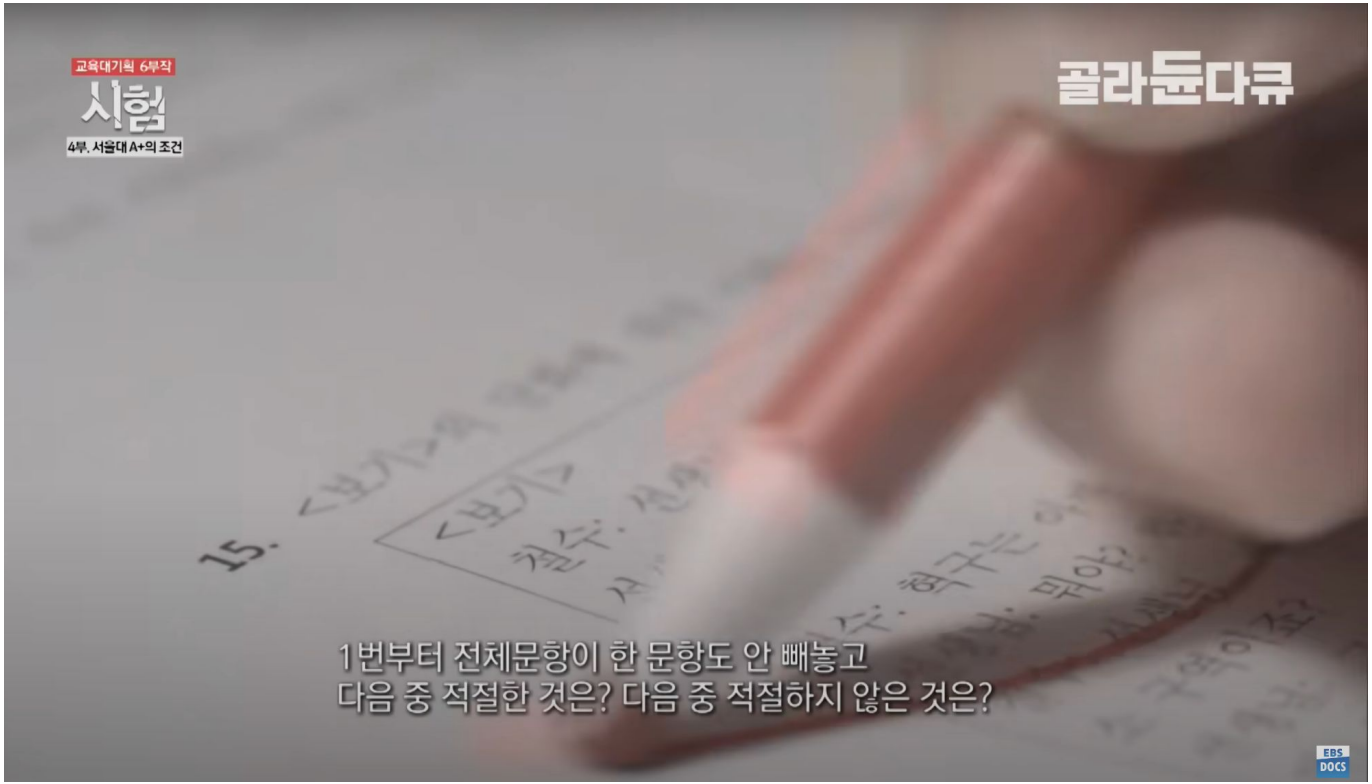
#질문 돌아보기

그런데 왜 우리는

정답을

찾으려고 하는가?

전통적인 학교의 “학습”


교육대기획 6부작
시험
4부. 서울대 A+의 조건
골라듄다큐
15. <보기>
<보기>
1번부터 전체문항이 한 문항도 안 빠놓고
다음 중 적절한 것은? 다음 중 적절하지 않은 것은?
EBS
DOCS

전통적인 학교의 “학습”

우아한테크코스의 “학습”

참고 자료


EBS DOCS
한국에서 발견된 특이한 공부 습관? 서울대가 이 상태라면 더 이상 천재는 없다 | ...
공유
EBS는 대한민국의 공영 방송 서비스입니다.
한국 학생 똑똑한 건
세계가 압니다 그런데..
서울대가 이 상태?
한국에서 더 이상 천재는 없다
다음에서 보기: YouTube

프로그래밍 패러다임

크론과 함께하는 재미난(難) 수업

로또를 하면서 드는 고민

‘이런 상황일 때 뭘 써야하지..? 어떤 게 맞지?’

#질문

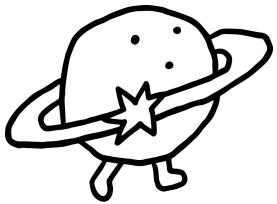
조합을 사용해서 멤버 변수로 클래스를 통해서 생성된 인스턴스를 추가한다 해도 결국 미리 구현된 클래스를 의존하는 것이니 미리 구현된 클래스가 변경된다면 조합을 사용하는 클래스에서도 이에 맞춰 변경을 해야하는건 아닐까요?

배열에 대해서 **Array** 객체를 많이 사용하시던데 그 이유가 궁금합니다.

혹시 의도를 아십니까?

JavaScript에서 객체를 만드는 다양한 방법을 이해하고 사용한다.

JavaScript에서는 클래스 말고도 객체를 만드는 방법은 여러 가지가 있다. 객체를 생성하는 방법에 대해서는 MDN 문서의 JavaScript 객체 기본과 Classes을 참고한다.

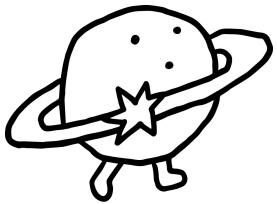


프로그래밍 패러다임 학습 목적

JavaScript에서 객체를 만드는 다양한 방법을 이해하고 사용한다.

JavaScript에서는 클래스 말고도 객체를 만드는 방법은 여러 가지가 있다. 객체를 생성하는 방법에 대해서는 MDN 문서의 JavaScript 객체 기본과 Classes을 참고한다.

패러다임이란?



패러다임이란?

국립국어원 표준국어대사전

패러다임 (paradigm)

「명사」

어떤 한 시대 사람들의 견해나 사고를 근본적으로 규정하고 있는 테두리로서의 인식의 체계. 또는 사물에 대한 이론적인 틀이나 체계.

- 어떤 인물에 대한 이해는 그가 살았던 시대의 패러다임 안에서 이루어져야 한다.

❶ 혁파 → 없앰
- 어려운 한자말 대신에 쉬운 말을 쓴다.

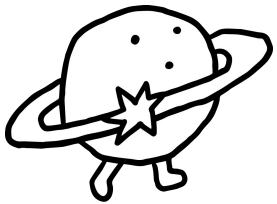
❷ 패러다임 → 틀/체계/방식

❻ 채널 → 경로/창구

❽ 리더십 → 지도력/통솔력

❾ 프로젝트 → (연구) 과제/사업

프로그래밍 패러다임이란?



프로그래밍 패러다임이란?

국립국어원 우리말샘

프로그래밍^패러다임 (programming paradigm)

분야 『정보·통신』

「001」 계산 모델을 공유하는 프로그램 언어에 의한 특징적인 프로그래밍을 추상화한 개념. 컴퓨터를 사용해서 풀어야 할 문제가 있고 그것을 프로그램으로서 기술할 때, 무엇에 착안해서 문제를 정리하고 무엇을 중심으로 프로그램을 구성할 것인가 따위에 대한 방향을 부여한다. 대표적인 예로는 명령형 프로그래밍, 객체 지향 프로그래밍 따위가 있다.

프로그래밍 패러다임 종류

명령형

선언형

프로그래밍 패러다임 종류

명령형

HOW

선언형

WHAT

프로그래밍 패러다임 종류

명령형
HOW

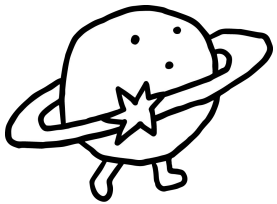
- 문제를 **어떻게(HOW)** 해결할 것인지 표현하는 데 초점
- 프로그램은 수행할 명령어들로 구성
- 알고리즘을 명시하고 목표를 명시하지 않음

프로그래밍 패러다임 종류

선언형
WHAT

- 무엇(WHAT)을 해결할 것인지 코드로 작성
- 해법을 정의하기보다는 문제를 설명하는 고급 언어
- 목표를 명시하고 알고리즘을 명시하지 않음

그래도
잘 모르겠어요



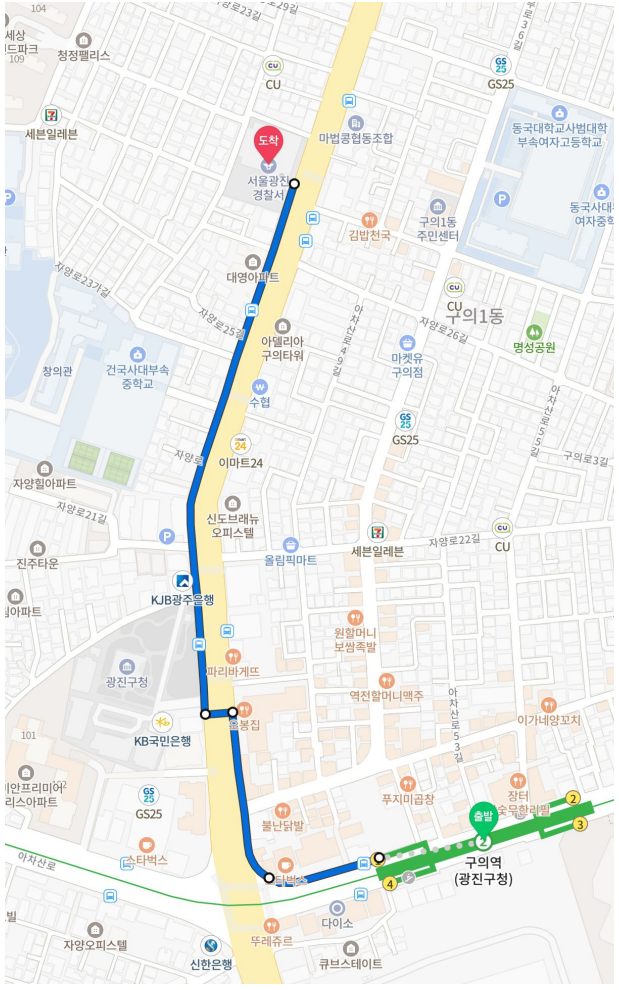
프로그래밍 패러다임 종류

구의역에서 광진경찰서에 가야 한다면?

명령형

HOW

1. 구의역 1번 출구까지 역사 내 이동
2. 구의역 1번 출구 나와서 직진 후 우회전
3. 직진 후 오봉집 앞에서 횡단보도 건너기
4. KJB광주은행 방향으로 직진
5. 대영아파트 방향으로 더 직진



프로그래밍 패러다임 종류

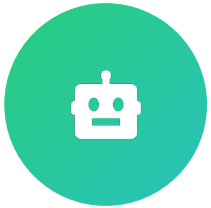
구의역에서 광진경찰서에 가야 한다면?

선언형
WHAT

- 서울 광진구 자양로 167

프로그래밍 패러다임 종류

절차적



객체지향

함수형

논리형

명령형 프로그래밍
Imperative Programming

선언형 프로그래밍
Declarative Programming

프로그래밍 패러다임 흥행 변천

절차(Procedural)적 프로그래밍

절차적 프로그래밍

절차적
procedural

- 하나의 큰 기능을 처리하기 위해
  작은 단위의 기능으로 나눠 처리 (Top down 방식)

- 비교적 작은 규모의 작업을 수행하는 함수 생성

```
procedure DisplayMessage;
begin
  WriteLn('Hello, world!');
end;
```

절차적 프로그래밍

절차적

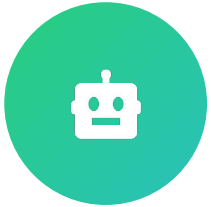
procedural

```
let shoppingCart = [];

function addItem(item) {
  shoppingCart.push(item);
}

function removeItem(item) {
  const index =
    shoppingCart.indexOf(item);
  if (index > -1) {
    shoppingCart.splice(index, 1);
  }
}
```

```
function listItems() {
  console.log('Shopping List:');

  for(let i=0;i<shoppingCart.length;i++) {
    console.log(shoppingCart[i]);
  }
}

// 예시
addItem('Milk');
addItem('Bread');
removeItem('Milk');
listItems(); // "Bread"
```

객체지향 프로그래밍

객체지향 프로그래밍



객체지향

- 명령어의 목록으로 보는 시각에서 벗어나
  여러 개의 독립적 단위, 즉 객체의 모임으로 파악하고자 함

- 구조체가 '객체'라는 추상적 개념을 담는 '클래스'로 발전

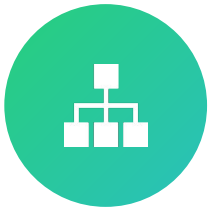
객체지향 프로그래밍


```
class ShoppingCart {
  items = [];

  addItem(item) {
    this.items.push(item);
  }

  removeItem(item) {
    const index = this.items.indexOf(item);
    if (index > -1) {
      this.items.splice(index, 1);
    }
  }

  listItems() {
    this.items.forEach(item => console.log(item));
  }
}
```

```
const myList = new ShoppingCart();

myList.addItem('Milk');
myList.addItem('Bread');
myList.removeItem('Milk');
myList.listItems(); // "Bread" 출력
```

객체지향 프로그래밍 특징

추상화



상속

다형성

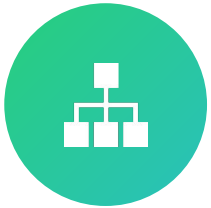
캡슐화

객체지향 프로그래밍 특징

추상화

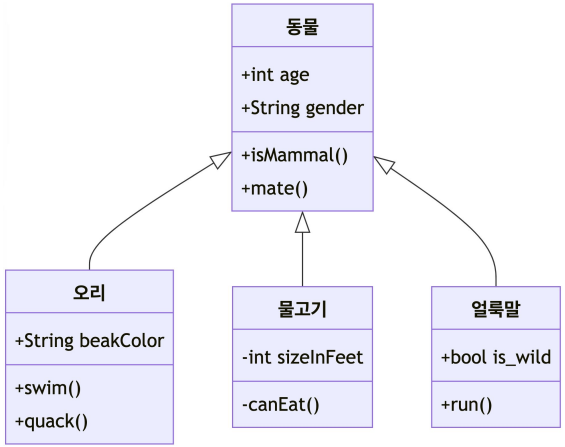
- 사물의 공통점을 뽑아 내는 과정

객체지향 프로그래밍 특징



상속

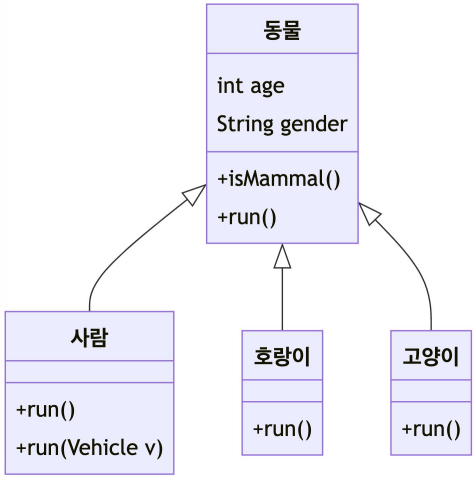
- 새로운 클래스가 기존의 클래스의 자료와 연산을 계승
- 클래스 간의 종속 관계를 형성하여 객체를 조직화함



객체지향 프로그래밍 특징

다형성

- 메서드의 인자의 개수나 자료형에 따라서 다른 기능을 함



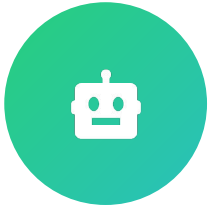
객체지향 프로그래밍 특징

캡슐화

- 객체의 세부 구현 내용을 숨김
  - 보안
  - 유지보수
  - 재사용

```
class Car {
 #name;
 #distance = 0;

  constructor(name) {
    this.#name = name;
  }

  move() {
    this.#distance += 1;
  }

  get position() {
    return this.#distance;
  }
}
```

객체지향 프로그래밍 한계점



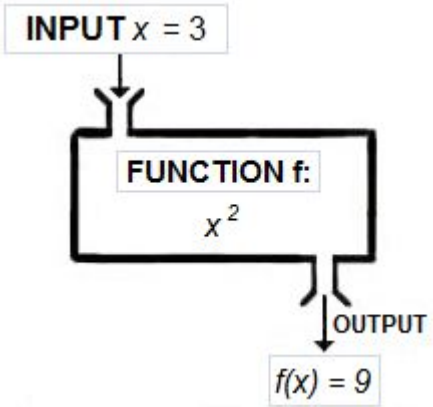
객체지향

- 함수의 비일관성
- 객체 내 상태 변화 제어의 어려움
  (특히 멀티스레드를 지원하는 타 언어에서)
- 객체 간 의존성 증가

함수형
프로그래밍

함수형 프로그래밍

함수란?



함수형 프로그래밍

함수형

- 프로그램의 계산 과정을 수학 함수의 수행으로 간주
- 프로그램은 함수의 정의들로 구성
- 함수 수행은 상태 변경을 허용하지 않음

함수형 프로그래밍 특징

함수형

- 상태를 제어하는 대신 상태를 없애고 목적에만 집중
- 객체 단위 대신 함수들의 집합으로 문제를 분해
- 주어진 입력에 대해 항상 결과가 같아야 함
- 지루한 일은 런타임이 하도록 맡기고
  개발자가 더 중요한 문제에 집중할 수 있도록 함

함수형 프로그래밍 특징

일급 함수

순수 함수

불변성

느긋한
계산법

함수형 프로그래밍 특징

일급 함수

- 변수에 함수를 할당할 수 있음
- 함수를 인자로 전달할 수 있음
- 함수를 반환할 수 있음

함수형 프로그래밍 특징

일급 시민
=시민권자
≠영주권자

- 거주의 자유
- 자유롭게 출입국할 수 있음
- 투표의 자유

  즉, 다른 요소들과 아무런 차별이 없다는 것을 의미

함수형 프로그래밍 특징

일급 함수

- 변수에 함수를 할당할 수 있음
- 함수를 인자로 전달할 수 있음
- 함수를 반환할 수 있음

```
// 함수를 변수에 할당
const greet = (name) => `Hello, ${name}!`;

// 함수를 다른 함수의 인자로 전달
const greetUser = (userName, greetingFunction) => greetingFunction(userName);

console.log(greetUser("Cron", greet)); // "Hello, Cron!"
```

함수형 프로그래밍 특징 (심화)

일급 함수

- 객체 단위 대신 함수들의 집합으로 문제를 분해

```
Array
  .from({ length: 5 }, (_, index) => index + 1)
  // [1, 2, 3, 4, 5]

  .map((number) => number * 2)
  // [2, 4, 6, 8, 10]

  .filter((number) => number > 5);
  // [6, 8, 10]
```

함수형 프로그래밍 특징 (심화)

- 개발자가 집중하는 것은 무엇이고 런타임이 집중하는 것은 무엇인가?

일급 함수

```
const numbers = [1, 2, 3];


const result = numbers.map(
  function (number, index) {
    return number * 2;
  }
); // [2, 4, 6]


// (number) => number * 2;
```

함수형 프로그래밍 특징 (심화)

- 개발자가 집중하는 것은 무엇이고 런타임이 집중하는 것은 무엇인가?

일급 함수

```
const numbers = [1, 2, 3];


const result = numbers.map(
  function (number, index) {
    return number * 2;
  }
); // [2, 4, 6]


// (number) => number * 2;
```

```
Array.prototype.map = function (callback) {
  const newArray = [];
  for (let index = 0; index < this.length; index++) {
    const element = callback(this[index], index);
    newArray.push(element);
  }
  return newArray;
};
```

함수형 프로그래밍 특징

순수 함수

- 동일한 인자가 들어오면 항상 동일한 결과를 반환
- 입력된 값을 포함하여 외부와 공유되고 있는 값 중
  함수가 참조할 수 있는 어떠한 값도 변경하지 않음

```
const add = (x, y) => {
 // ❌ x.value = 1;
 return x.value + y.value;
};

console.log(add(2, 3)); // 5
```

함수형 프로그래밍 특징

불변성

- 최초 생성된 값이 변경되지 않음

```
const originalArray = [1, 2, 3];
const newArray = [...originalArray, 4]; // 원본 배열을 변경하지 않고 새 배열을 생성

console.log(originalArray); // [1, 2, 3]
console.log(newArray); // [1, 2, 3, 4]
```

함수형 프로그래밍

함수형

```
const addItem = (list, items) => [...list, ...items];
const removeItem = (list, item) => list.filter(currentItem => currentItem !== item);
const listItems = (list) => {
  list.forEach(item => console.log(item));
}

let shoppingCart = addItem([], ['Milk', 'Bread']);
shoppingCart = removeItem(shoppingCart, 'Milk');
listItems(shoppingCart);
```

함수형 프로그래밍 한계점

함수형

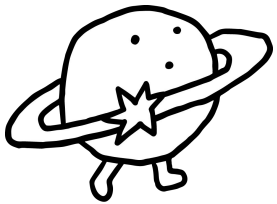
- 학습 곡선
  - 고차 함수, 커링, 모나드 등…
  - 디버깅/오류 추적의 어려움
- 성능
  - 메모리 사용량, 가비지 컬렉션 동작 빈도 증가 등

함수형 프로그래밍 한계점

- 학습 곡선에 따른 가독성 예시

```
const add = x => y => x + y;
const multiply = x => y => x * y;

// 커링을 사용하여 함수를 생성
const addFive = add(5);
const double = multiply(2);

// 배열의 각 요소에 함수 적용
const numbers = [1, 2, 3];
const result = numbers.map(addFive).map(double);

console.log(result); // [12, 14, 16]
```

함수형 프로그래밍 모든 기법을 알아야 하나요?

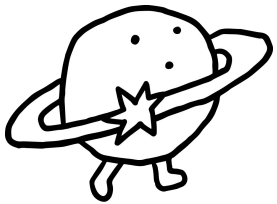


프로그래밍 패러다임 학습 목적

JavaScript에서 객체를 만드는 다양한 방법을 이해하고 사용한다.

JavaScript에서는 클래스 말고도 객체를 만드는 방법은 여러 가지가 있다. 객체를 생성하는 방법에 대해서는 MDN 문서의 JavaScript 객체 기본과 Classes을 참고한다.

다중 패러다임 프로그래밍?

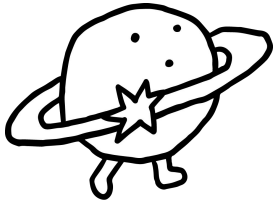


다중 패러다임 프로그래밍

Wikipedia

다중 패러다임 프로그래밍 언어(multiparadigm programming language)는 하나 이상의 프로그래밍 패러다임을 지원하는 프로그래밍 언어이다. 이것은 비야네 스트롭스트룹이 설명하는, "하나 이상의 프로그래밍 스타일을 따르는 프로그램"을 허용한다. 이런 언어들의 설계 목표는 모든 문제를 가장 쉽고 효율적으로 풀 수 있는 하나의 패러다임은 없다는 것을 인정하고, 프로그래머가 자신의 일에 가장 적합한 것을 사용할 수 있게 하는 것이다.

No Silver Bullet



패러다임별
문제 해결 방식

절차적 프로그래밍

절차적

procedural

```
let shoppingCart = [];

function addItem(item) {
  shoppingCart.push(item);
}
function removeItem(item) {
  const index =
    shoppingCart.indexOf(item);
  if (index > -1) {
    shoppingCart.splice(index, 1);
  }
}
```

```
function listItems() {
  console.log('Shopping List:');

  for(let i=0;i<shoppingCart.length;i++) {
    console.log(shoppingCart[i]);
  }
}

// 예시
addItem('Milk');
addItem('Bread');
removeItem('Milk');
listItems(); // "Bread"
```

객체지향 프로그래밍


```
class ShoppingCart {
  items = [];

  addItem(item) {
    this.items.push(item);
  }

  removeItem(item) {
    const index = this.items.indexOf(item);
    if (index > -1) {
      this.items.splice(index, 1);
    }
  }

  listItems() {
    this.items.forEach(item => console.log(item));
  }
}
```

```
const myList = new ShoppingCart();

myList.addItem('Milk');
myList.addItem('Bread');
myList.removeItem('Milk');
myList.listItems(); // "Bread" 출력
```

함수형 프로그래밍

함수형

```
const addItem = (list, items) => [...list, ...items];
const removeItem = (list, item) => list.filter(currentItem => currentItem !== item);
const listItems = (list) => {
  list.forEach(item => console.log(item));
}

let shoppingCart = addItem([], ['Milk', 'Bread']);
shoppingCart = removeItem(shoppingCart, 'Milk');
listItems(shoppingCart);
```