함수와 일급객체

프로토타입은 한마디로 얘기하면 자스에선 상속 매커니즘이다.

함수는 객체다.

그러면 왜 자스에서는 함수를 객체로 취급할까?

콜백함수 기억나? 얘의 특징은 함수를 인수로 줄수 잇다. 인수에 뭘 줄수 잇나?

매개변수 x 가 있다고 가정했을때 foo(x) 해서 값이 올수 있는 자리다. 왜 값이 올수 잇는 자리지? x 에 할당이 되어야 하기 때문에. 할당문은 없지만 암묵적으로 할당이 이루어진다. 따라서 여기에 올수 잇는애들은 값이다. 근데 콜백 함수 써서 알수 있지만 함수를 줄수 있다. 즉, 이런식으로 콜백 함수, 콜백함수를 받아서 사용하는 고차함수 이런것들을 가지고 코딩을 하는 패러다임을 함수형 프로그래밍이라고 한다.

함수형 프로그래밍이 되려면 함수가 값이어야 한다. 즉, 함수형 프로그램 패러다임을 실현하기 위해서는 함수가 값이어야 한다. 함수형 프로그래밍 패러다임이 없는 건 함수가 값일 필요가 없다.

일급객체를 공부한다는 얘기는 고차함수에서 고생한다는 말이다.

일급객체란 말은 뭐냐면, 일급이 있으면 이급도 잇나? 일급 객체를 다른 책에서는 퍌스트 클래스 시티즌이라고 한다. 시티즌 이 시민이잖아. 우린 시민이란 말이 익숙치 않잖아. 서양애들은 시민이란 말 쓴다.

옛날에 시민이 여러가지 있었잖아. 시민이라고 하면 투표권 있는 사람들만 시민이었다.미안한데, 옛날에 여자들은 시민이 아니었어. 노예들도 시민이 아니었어. 그럼 누가 시민이냐면 투표권이 있고 군대에 갈수 있어야 했다. 그 시민들을 일급시민이라고 했다. 그니까 객체도 일급이 있고 이급이 있다는 거다. 그러면 일급이 뭐냐면 완전히 값과 똑같은거다.

그러면 이급객체가 있나? 없다.

일급객체에는 특징이 있는데, 에크마에 일급객체라는 말이 있을까? 없다. 통상적으로 함수형 패러다임이라고 한다.

그럼 함수를 무명의 리터럴로 생성할수 있나? 있다. 무명 함수와 기명함수의 차이는 이름이 있고 없고인데,

만약에 기명이면 이름이 있기 때문에 리터럴 자체만으로도 의미가 있다. 이름이 있으니까.

근데 리터럴에 이름이 없으면? 무명 리터럴의 특징은 반드시 어딘가에 할당을 해준다. 만약에 무명 리터럴을 할당 안하고 쓰면 어떤 결과가 일어나냐? 할당 못하면 재사용 못한다.

따라서 무명 리터럴이 가능한 문법을 제공한다는 뜻은 변수에 할당하라는 의미이다.

무명 리터럴을 변수에 할당하는 것을 함수 표현식이라고 한다 .

함수 표현식의 함수 객체는 언제 생성되나? 런타임 이전이야? 할당이 언제되냐고. 런타임에 생성이 가능하다는 거다. 즉, 함수를 어케 만든다? 함수 정의를 미리 한다음 함수를 하는 이런 정해진 코스가 아니라 바로 호출하기 직전에 만들수 있다.

그리고 또하나,

무명 리터럴로 만들수 있다는 것 은 함수를 어떤식으로 사용할수 있다는 얘기냐면, 함수를 호출하면서 (foo(...))만들수 있다. <- 이게 바로 콜백이다.

함수를 변수나 자료구조, 자료구조라는 것은 복합적, 하나의 값만 갖는게 아니라 여러개의 값을 갖고 있는거(객체, 배열 등) 이런데에 넣을수 있느냐는 말이지. 함수를.

배열이나 객체에 함수를 집어넣을수 있다. 배열은 모르겠는데 객체는 넣을수 있음. 그게 메소드라고 하는거다. 배열에 넣을수 있을까? 이런거 될까? [function(){},...] 됩니다. 왜냐. 값이니까.

[ ] 안에 요소가 오는데, 요소에는 값이 오면 된다.

함수의 매개변수에 전달할수 있나? 있다.

함수가 함수의 리턴값이 될수 있나? 될수잇다.

이 4개를 만족하는 함수를 일급객체라고 한다. 자스는 이걸 다 만족한다. 이게 되면 함수형 프로그래밍이 된다는 거다.

 함수가 일급 객체라는 것은 함수를 객체와 동일하게 사용할수 있다는 거다.

객체는 값이져? 따라서 함수는 값이다. 라는게 성립이 된다.

함수를 자스에서는 값과 동일하게 취급할수 있다. 지금 이 얘기를 한마디로 설명하면 함수는 일급객체다. 라고 퉁칠수 있다.

 그런데, 함수는 객체다 라고 햇잖아. 그러면 함수하고는 객체하고는 똑같은거냐? 다르다.

 객체의 개념이 더 큰거고, 그 객체 내부에 여러가지 부분 집합이 있는데 거기에 함수, 배열, 유사배열객체, 서클객체 등등

함수는 객체인데, 둘은 다르다. 함수는 객체라고 했져? 객체의 특성은 함수가 다 갖고 있다.

함수가 갖고 있는 특성을 객체가 다 갖고 있진 않다. 함수 고유의 특성이 많다.

함수 고유의 속성(프로퍼티) : [[call]] <- 일반객체는 이거 없다. 호출 못한다는 뜻이다.

함수는 무조건 [[call]] 이 있다.

[[construct]] <- 객체가 이거 갖고 있을수 있나? 갖고있을 수 있다. 이게 있는 애는 컨스트럭터 라고 한다. 콜, 컨스트럭트 둘다 있는 애는?

클래스는 일반함수로 호출 못한다. 별도로 생각해야 한다.

클래스는 일반 함수 호출 못하니까 [[call]] 이 없다.

함수와 객체의 또 다른 차이점이 뭐가 있을까? 객체는 호출할수 없으나 함수는 호출할수 있다.

내부 메소드는 감춰진 프로퍼티니까, 객체를 가지고 있는 프로퍼티라면 함수는 다 가지고 있다.

그럼 함수만 가지고 있는 건 뭐가 있을까?

예를 들어서,

객체들이 갖고 잇는 프로퍼티는 함수는 다 가지고 있는데 함수만 가지고 있는 프로퍼티가 뭐가 있을까?

이건 지금부터 배우자.

```
function square(number) {
  return number * number;
}

square.f = function () {}  <- 이거 되는거다. 함수가 메소드를 갖고있는거다. 왜냐면 얘는 객체기 때문에. 객체는 프로퍼티를 가지고 있으니까.
square.a = 1; <- 이것도 되는거다.

string. 하고 나오는 목록들은 전부 프로퍼티다.
```

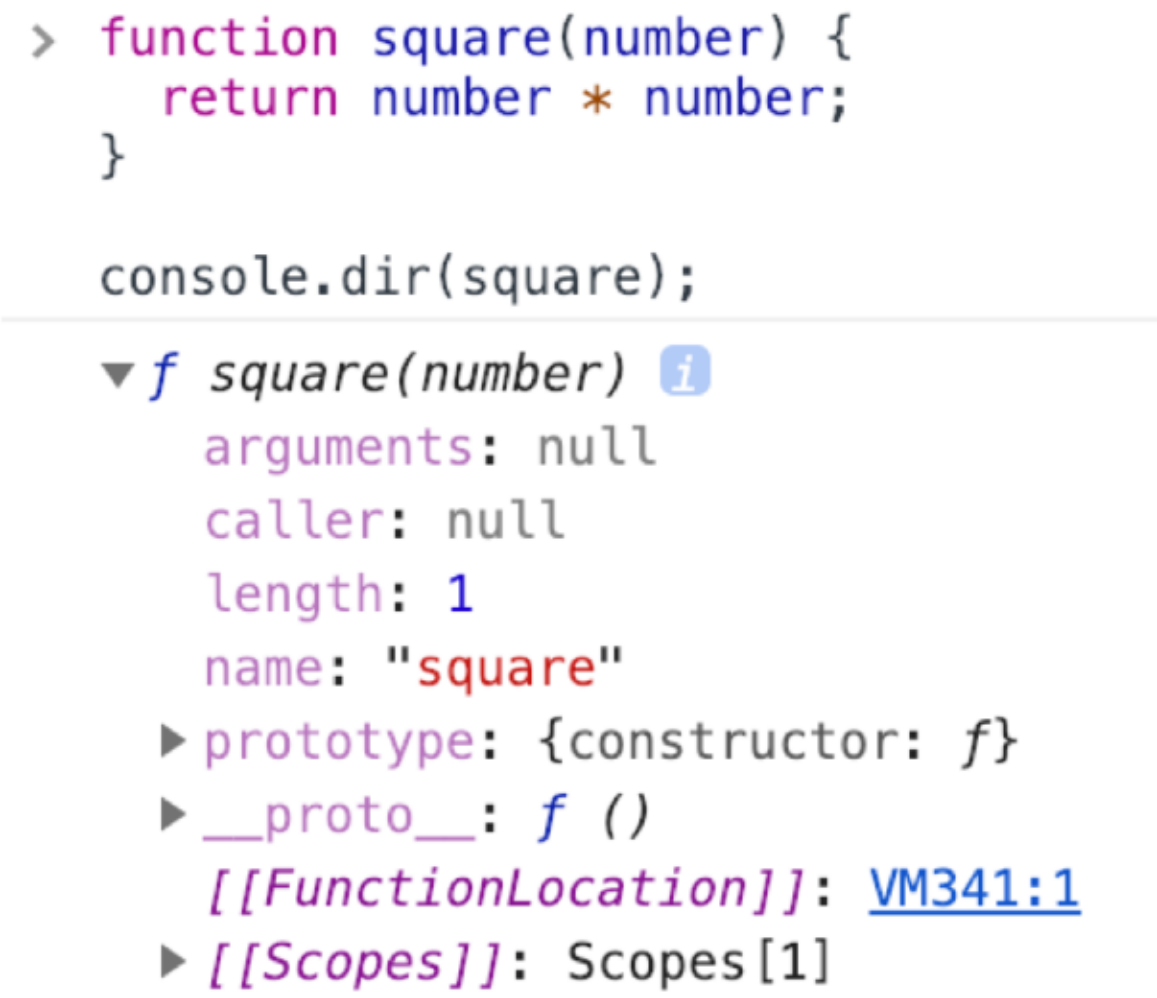
아~ 이 앞에 O 자가 붙는걸 생성자 함수라고 했는데, 얘는 빌트인 객체라고 한다. 얘가 메소드들을 제공한다. 얘는 함수이지만 객체이다. 생성자 함수는 메소드를 갖고 있다.

그럼 우리가 만든 펄슨 생성자 함수도 프로퍼티를 갖고있겠네?

함수 객체가 어떤 프로퍼티를 가지고 있는지 설명해보자,

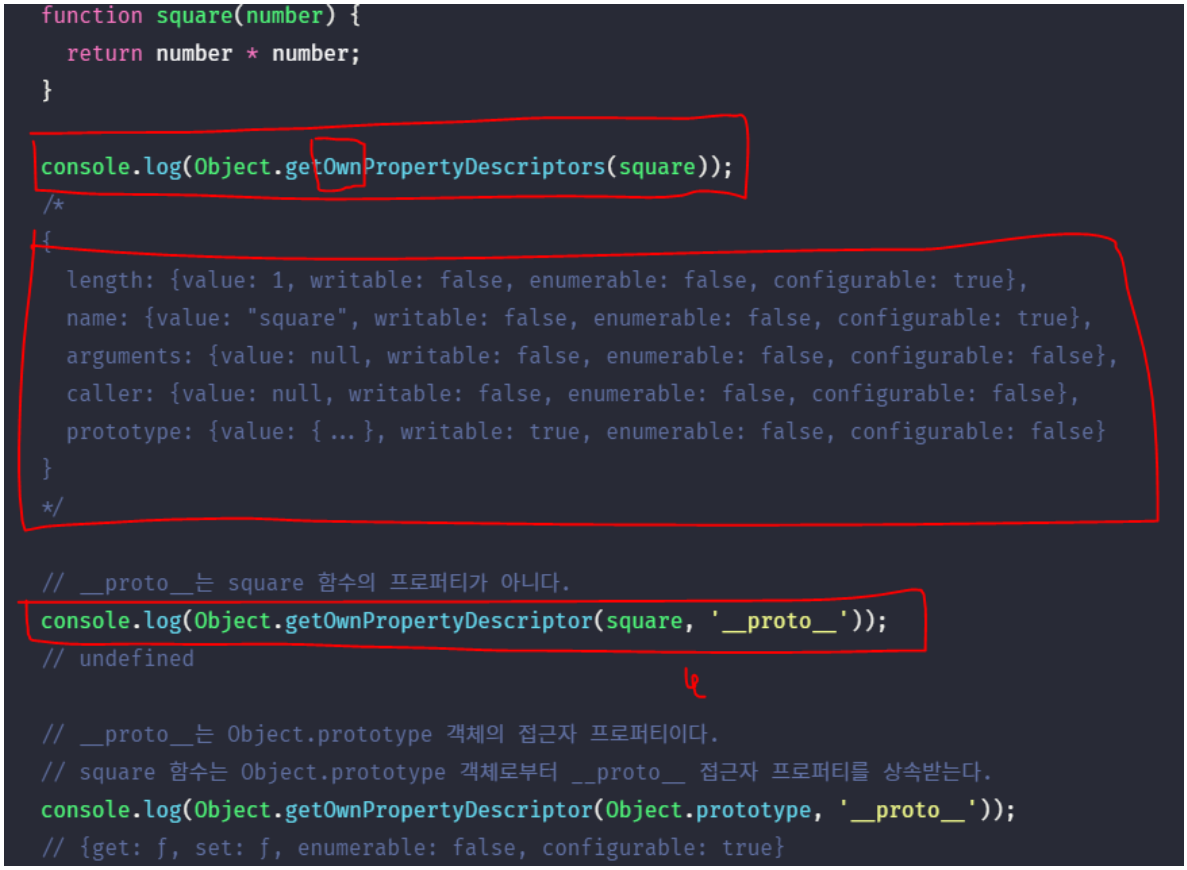
```
function square(number) {
  return number * number;
}    // 일반함수인가? 모르는거다. 어케 호출될지 모르니까.

console.dir(square);
함수는요, dir 안하고 로그 하면 소스코드의 자체를 보여준다.
객체로서 함수 객체를 객체로서 안의 프로퍼티를 들여다보려면 dir 이란 함수를 호출해줘야 한다.
```

콜론 붙은것들은 다 스퀘어라는 함수객체의 프로퍼티들이다.



함수 객체의 프로퍼티

직접이 아니라 상속받아 쓰는 애다.



```
function square(number) {
  return number * number;
}

console.log(Object.getOwnPropertyDescriptors(square));
/*
{
  length: {value: 1, writable: false, enumerable: false, configurable: true},
  name: {value: "square", writable: false, enumerable: false, configurable: true},
  arguments: {value: null, writable: false, enumerable: false, configurable: false},
  caller: {value: null, writable: false, enumerable: false, configurable: false},
  prototype: {value: { ... }, writable: true, enumerable: false, configurable: false}
}
*/

// __proto__는 square 함수의 프로퍼티가 아니다.
console.log(Object.getOwnPropertyDescriptor(square, '__proto__'));
// undefined

// __proto__는 Object.prototype 객체의 접근자 프로퍼티이다.
// square 함수는 Object.prototype 객체로부터 __proto__ 접근자 프로퍼티를 상속받는다.
console.log(Object.getOwnPropertyDescriptor(Object.prototype, '__proto__'));
// {get: ƒ, set: ƒ, enumerable: false, configurable: true}
```

.... 저 꼬랑지 부분은..... 없으니까 언디파인드가 나온다.

우리가 지금 뭘 살펴보고 있나? 제목을 보시면 함수 객체의 프로퍼티를 보고 있는거다.

근데 아규먼츠 얘는 프로퍼티인가?

아규먼츠 : 얘는 프로퍼티인가? 프로퍼티는 맞아요? 네. 그럼 왜 프로퍼티가 맞지? 위에 콜론 붙어서 나오니까. 근데 표준으로는 프로퍼티가 아니다. 폐지되었다. es3에서 폐지되었따. 근데 왜 크롬은 프로퍼티로 출력하고 있지?

모든 자스엔진이 다 표준을 깔고 있지 않다. 옛날에 폐지된것들을 그냥 쓰는 경우도 있다. 지금 이렇게 아규먼츠ㄴ 프로퍼티가 있다는것은 얘를 어케 참조 할수 잇다는 거야? 예를 참조 하고 싶어, 그럼 어케 해?스퀘어.아규먼츠로 참조하라는 말이잖아.

그럼 우리는 아규먼츠를 어케 들여다봐야해?

결론은 es3 에서 아규먼츠 프로퍼티는 폐지되었어. 폐지인데 살려두고 있는거다. 살려두고 있는건 문제가 아니다. 표준인데 구현이 안되는게 문제다. 그러면 우리는 아규먼츠를 어케 들여다봐야해?

그전에,

아규먼츠라는 프로퍼티가 이전에 존재했엇잖아. 걔 값이 있을거 아냐. 걔 값이 아규먼츠 객체다. 옛날에 아규먼츠 객체를 보려면 함수이름.아규먼츠 해서 참조했었다. 함수내부에서.

그럼 함수 내부에서 함수이름.아규먼츠 이렇게 하는거 가능한가? 가능하다.

함수 선언문은 함수 몸체 안에서 함수이름 참조할수 있나? 있따. 함수 표현식인데 무명 함수 리터럴로 만든 함수 표현식이다. 이때는 함수이름.아규먼츠 안되는거 아니야? 된다. 왜되? 식별자로 보는거니까.

함수이름으로 본게 아니라 식별자로 본거니까 되는거다. 함수 내부에서 디스를 그냥 사용할수 잇었듯이, 그거와 마찬가지로 함수 몸체 안에서 암묵적으로 다 보이는게 몇개 있는데 디스하고 아규먼츠하고 뉴.타겟이 보인다. 우리가 선언하지 않아도 보인다.

그럼 얘 정체가 뭐냐? 객체다. 객체인데 유사배열 객체로 분리된다는 말이다.

프로퍼티값은 인수다.

렝스가 있는 것을 유사배열이라고 한다.

```
const sum = function () {
let res = 0;
for (let i = 0; i < arguments.length; i++){
res += arguments[i];
}
return res;
};
console.log(sum(1, 2, 3)); //6
```

1, 2, 3 : 첫번째 인수, 두번째 인수, 세번째 인수 가 이런 순서를 의미하는 프로퍼티 이름에 할당이 되어져있는 상태에서 렝스를 가지고 있따.

이런 애들을 이런식으로 프로퍼티 키가 숫자 비슷하게 생겼고, 렝스를 가지고 있는 애들을 유사배열이라고 한다.

정해진 이름의 프로퍼티 키에 함수를 만들면 리터러블이 된다는 말이다.

리터러블은 포 오브라는 구문으로 순회할수 있다. 그리고 즉 스프레드 연산자의 대상이 될수 있따.

포문을 돌릴수 있다. 그럼 이런 아규먼츠 객체가 이런 유사배열 객체 형태를 띄고 잇을까?

위 코드에서 i < 3 이 아니라 i < 아규먼츠.렝스 이렇게 표현한 이유는 인수가 몇개가 올지 모를때 이렇게 쓴다.

포문은 알고리즘 할때만 하고 거의 안쓴다. 엄청 많이 돌때 포문을 많이 쓴다. 거의 100번 이상. 근데 100번 이상 돌게끔 한거 부터가 이상한거다.

그래서 포문을 안쓰고 싶어. 그러면 배열로 얘를 만들수 있을까? 얘 이름이 유사배열 객체잖아. 얘를 배열로 바꿔. 왜 바꿀까? 배열에 엄청 유명한 메소드가 나와. 어케 바꾸면 될까?

배열에 슬라이스라는 메소드가 있는데, 복사가 된다. 슬라이스는 원래 잘라내는거다. 그래서 결론적으로 복사가 되는거다. 슬라이스는 누구의 메소드? 배열의 메소드다. 자스는 간접호출이란게 있따.

슬라이스를 콜 메소드로 호출하면서 인수로 전달하면 이 메소드를 호출하면서 디스를 바꾼다.

슬라이스 라는 애가 디스를 쓰는거다.

포문 돌면서 배열 만들수 있다.

아규먼츠 객체: 기본적으로 유사배열 객체라서 포문으로 돌릴수 있따.

그리고 아규먼츠라고 하는 함수의 프로퍼티가 있는데, 그 함수의 프로퍼티를 잘 보면 된다. 디스와 같이 다이렉트로 보면 된다.

문제의 발단 : 매개변수를 정의할수 없었다는게 문제였는데, 매개변수를 정의할수 잇는 방법은 매개변수 앞에 '...' 붙여주면 된다. 그럼 얘가 무슨 뜻이냐면 몇개가 올지 모르지만, 몇개가 오던지 다 배열로 받아주겟다. 라는 말이다.

결국 es6 문법을 쓰면 아규먼츠 안써도 된다.

콜러 프로퍼티

콜러는 비표준이다.

비표준이란 얘기는 에크마스크립트 사양서에 안올라와있다는거다. 그래서 몰라서 된다.

근데 이름에서 알수 있듯이, 콜러는 뭘까? 그 함수를 호출한 함수를 말한다.

몰라도 된다. 넘어가도 된다.

렝스 프로퍼티

여기서 말하는 렝스는 아규먼츠 객체 내부의 렝스랑 다른거다.

이거는 함수의 렝스다.

얘는 매개변수의 개수를 가리킨다.

거의 쓸일 없다.

네임 프로퍼티

여기에 뭐가 들어가냐면 함수 이름이 들어간다.

이것도 거의 쓸일 없다.

프로토 접근자 프로퍼티

얘는 함수가 가지고 있는게 아니고 모든 객체가 가지고 있다.

얘는 뭐냐면 얘로 접근하면 프로토타입 객체가 나옴.

나중에 다음시간에 살펴보자

__ proto __ 가 있고 prototype 이 있네? 이 두개가 오늘의 핵심이다.

이게 네이밍이 안좋아서 헷갈린다.

__ proto __ 얘는 모든 객체가 갖고 있고

"프로토타입" 은 함수만 가지고 있다.

프로토타입이라은 프로퍼티를 가지고 있따. 얘는 함수 객체는 반드시 얘가 있따.

그런데 모든 함수 객체가 얘를 가지고 잇는게 아니라 컨스트럭터만 가지고 잇따.

이 프로퍼티를 따라가면 어떤 객체가 나오는데, 프로토타입 객체라고 한다.

객체지향이란건 어떤 패러다임이냐면, 우리가 인식할수 있는 모든 객체들을 객체라고 하는 독립적인 뭔가를 만드는거다.

컨스트럭터 함수를 만들면

```
function Person (name) {
    this.name = name;
    this.sayHi = function () {
        console.log(`Hi! my name is ${this.name}.`);    <- sayHi
가 여러번 만들어져서 문제점.
    };
}
const me = new Person ('Lee');
me.sayHi();
----
//세이하이는 일반함수로 호출된거야, 생성자함수로 호출된거야? 메소드로 호출된거
다.
```

모든 객체가 [[prototype]] 라는 내부슬롯을 갖고잇따. 그럼 이 내부슬롯의 값은 언제 결정되나?

그럼 me 라는 인스턴스는 프로퍼티가 몇개? 두개다. 네임이랑 쎄이하이.

인스턴스가 메소드를 가지고 있는게 문제다.

식별자하고 프로퍼티를 혼동하지 않도록 주의하자.

결국은 프로퍼티도 값을 찾아내는 하나의 키잖아. 근데 문법적으로 식별자하고 프로퍼티하고 다르다.

프로토타입

자스는 멀티패러다임 언어다.

자스는 클래스가 없다. 아~ 자스는 객체지향 언어가 아니라는 오해를 받았지만

자스는 객체지향 언어이긴 하다. 근데 클래스 기반이 아니라 프로토타입 기반이다.

프로토타입 기반이 클래스기반보다 유연하다.

데이터를 주고받을수 있어야 한다. 그 중에 중요한게 상속이다. 프로퍼티를 찾을때... 부모를 찾으려고 하는게 상속구조다.

자스는 훌륭한 객체지향 언어다.

객체지향 프로그래밍

객체지향이라는 것은 여러개의 독립된 객체들을 만든것이다. 그 객체들을 연관을 지어서 프로그래밍을 하려는 패러다임을 말한다. 그럼 객체는 뭘까? 개발자 또는 프로그램이 주체고 ... 대상들이 다 객체다.

근데 그 객체를 어케 표현하냐면 속성으로 표현한다. 모든 속성ㄷ을 다 막라 해야하나요?

... 그것을 추상화라고 한다.

```
const person = {
    name: 'Lee'
};
console.log(person.constructor); // object
```

위 코드는 뭘 보려고 하는거야?

```
const person = {
    name: 'Lee'
};
console.log(person.__proto__ === object.prototype); //true

const arr = {};  <- 얘도 객체다.
console.log(arr.__proto__ === Array.prototyoe); // true
```

```
// 생성자 함수
function Circle(radius) {
  this.radius = radius;
  this.getArea = function () {
    // Math.PI는 원주율을 나타내는 상수이다.
    return Math.PI * this.radius ** 2;
  };
}

// 인스턴스 생성
// 반지름이 1인 인스턴스 생성
const circle1 = new Circle(1);
// 반지름이 2인 인스턴스 생성
const circle2 = new Circle(2);

// Circle 생성자 함수는 인스턴스를 생성할 때마다 동일한 동작을 하는
// getArea 메소드를 중복 생성하고 모든 인스턴스가 중복 소유한다.
// 따라서 getArea 메소드는 하나만 생성하여 모든 인스턴스가 공유하는 것이 바람직하다.
console.log(circle1.getArea === circle2.getArea); // false

console.log(circle1.getArea()); // 3.141592653589793
console.log(circle2.getArea()); // 12.566370614359172
```

this 레디우스는 ....

프로토타입 객체

모든 함수가 만들어질때에라고 하면 안된다. 컨스트럭터와 함께 호출할수 있는 함수들을 생성하면 함수 객체들이 만들어지는데, 프로토타입이 쌍으로 만들어진다. 동시에 만들어지는 것인가? 동시라는게 아니라 처리를 따로따로 되긴 하지만 만들고 바로 만드니까 같이 만들어진다고 이해하고 있자.

우리가 예를 들어서 생성자 함수를 만들고 ... 프로토타입이라고 하는 프로토타입 객체가 만들어질텐데.. 비어있는 상태이다. 왜 이렇게 비어놧을까? 생성자 함수를 만들었어, 이게 평가되어져서... 프로토타입 객체가 만들어질텐데 .... 뭘 넣을지 모른다. 자스엔진은. 일단 비워놓는다. 그럼 알아서 채워넣어야 한다.

꼭 채워넣어야 할 필요는 없다.

proto 접근자 프로퍼티

언더바 프로토 라고 하는걸 접근자 프로퍼티라고 한다.

모든 객체들은 그 객체가 함수가 되든 배열이 되든 모든 객체는 프로토타입이라고 하는 내부 슬롯을 가지고 있다. 이 내부 슬롯의 값은 언제 결정되나? 그 값은 어케 결정되고 언제 결정되나? 자신이 태어날때 결정해야 하는데 그 결정을 어케 하냐? 자신과 연결되어진 생성자 함수가 반드시 있다 그 생성자 함수... 객체가 프로토타입의 값이 되는것이다.

어떤 객체가 태어날라고 하고 있어. 태어나면. 뭘 해야 하냐면 내 부모가 누군지 결정해야 한다. 객체가 결정하는게 아니라 자스엔진이 얘를 태어나게 해주면서 결정해주는데 그 결정에 규칙이 있는데 그 규칙이 뭐냐면 모든 객체는 생성자 함수와 연결이 되어있다. 어케 연결되어있나? 그 생성자 함수를 어케 찾을수 있지? 예를 들어서

펄슨 생성자 함수를 뉴로 해서 객체를 만들었다. 그럼 걔는 자신의 프로토타입을 어케 만들지?

... 이거의 값이다. 모든 객체들은 생성자 함수와 연결 되어잇다.

객체들은 태어날때 자기의 부모를 안다. 그 부모의 참조를 프로토타입에 연결한다.

내부슬롯의 값을 우리가 참조할수 없는데 언더바 프로토로 참조 또는 갱신을 할수가 있다.

모든 프로퍼티는 다 참조가 된다. 이렇게 언더바 프로토라고 이름을 짓지는 않을거잖아. 그리고 참조하기 어렵게 하는 이유가 있다. 심볼이 나오기 이전의 얘기다. 내부적으로 쓰려고 언더바 프로토 라고 이름을 지었다. 이런 이름을 비표준이지만 이런 이름을 가지고 어떠한 이유에서인지 브라우저 밴더들이 이런 프로퍼티 를 갖기 시작했다. 계속 비표준이다가 es6 되서 표준이 되었다.

기본적으로 참조도 가능하고 할당도 가능하다. 접근을 하면 함수가 돈다. 게터가 돈다. 게터가 돌면서 뭘하지? 그 객체에 [[프로토타입]]에 참조 값을 반환한다.

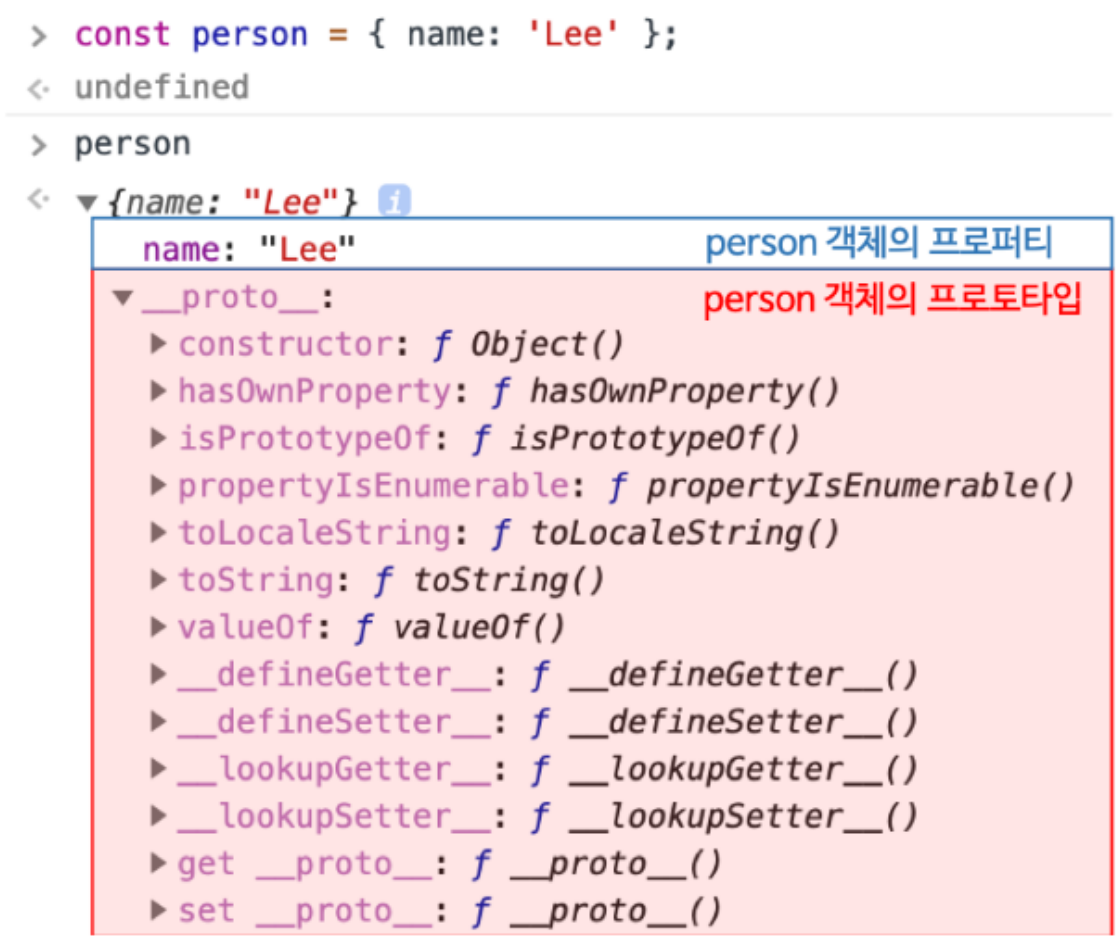
그 객체에 프로토타입에 참조를 반환한다.그리고 저 언더바 프로토에 어떤 값을 할당하면 무슨 일이 일어날까? 세터가 돈다. 세터가 돌면서 [[프로토타입]]의 참조값을 바꾼다.

.... 자신의... 를 교체할수 있다. 언더바 프로토라는 접근자 프로터피는 누가 사용할수 있을까?

이걸 생각하려면 일케 생각하자. 쟤가 하는 일이 뭐지? 프로토타입에 접근하거나 교체하는 거잖아.

그러면 프로토타입을 가지고 있는 애는 누구냐? 모든 객체다. 따라서 언더바 프로토는 누가 가지고 있어야 하는 기능이지? 모든 객체가 가지고 있어야 하는 기능이다. 그러면 모든 객체가 일일히 가지고 있어야 하나? 그럴 필요없다.

언더바 프로토는 존재의 위치가 어디냐? 오브젝트 프로토타입이다.



크롬 브라우저의 콘솔에서 출력한 객체의 프로퍼티

네임: 리는 펄슨이라는 객체...

언더바 프로토 는 프로터티 키네. 언더바 프로토 밑에 잇는 애들은 참조한 애들이다. 네임의 내용이다.

그럼 저 많은것들을 다 사용할수 있다.

```
const p = {
console.log(p.hasOwnProperty('name')); //true
console.log(object.getOwnPropertyDescripts(p));
}
```

가능성이 있는 곳이 3개다.

함수 객체의 프로토타입 프로퍼티

언더바 프로토, [[프로토타입]], 프로토타입 <- 이 3개가 헷갈리게 한다. 앞에 두개가 세트이고,

뒤에 프로토타입 앞에는 함수가 와야 하고, 언더바 프로토 앞에는 모든 객체가 와야 한다.

모든 객체긴 한데 괄호열고(오브젝트. 프로토타입) 이걸 상속받는 객체다.

맨 뒤의 프로토타입 앞에 잇는 함수가 가리키고 있는 값이 뭐지? 이 함수가 생성할 인스턴스.. 프로토타입을 가진다.

예습: 21번까지 20번은 가볍게만 읽어보고 오기. 소감 물어볼거야. 뭘 얘기하는지 주제만 찾아오면 되.