**본인참고자료**

객체란?

데이터와 기능을 논리적으로 묶어 놓은 것을 의미하며 데이터는 프로퍼티 값

기능은 메서드.

프로퍼티(키와 벨류의 쌍)는 키와 할당된 벨류가 있다.

메서드: 메서드명 함수=>기명함수,

무기명함수(무명함수:예시 화살표함수)

**인사**

종호형님이 서울에 세미나를 들으러 올라가셨기 때문에 오늘은 제가 이어서 지난번 수업을 복습을 주도하도록 하겠습니다. 여러분들과 자주 볼 기회가 없었고, 처음으로 강단에 서다보니 긴장을 많이 하였습니다. 부족하지만 최선을 다하도록 하겠습니다. 잘 부탁드립니다. 자 오늘 내용은 지난 시간에 배운 내용을 다시 복습한 뒤에 이어 준비한 내용으로 진도를 조금 나갈 수 있도록 준비하였습니다.

**개념 설명**

러닝자바스크립트라는 책이 존재합니다. 이 책의 저자 이선 브라운은 객체를 다음과 같이 정의하였습니다. “객체는 데이터와 기능을 논리적으로 묶어놓은 것이다. ” 여기서 말하는 데이터는 프로퍼티라고 하며 기능은 메서드라고 합니다. 객체를 이용하는 이유는 대부분이 현실세계의 물리, 논리적인 법칙들로 어떤 것을 프로그래밍으로 구현 하기 위함 입니다.

~~추가로 객체의 진짜 특징은 키를 통해 프로퍼티에 접근할 수 있다는 점입니다.~~

(이는 객체의 진짜 특성이 아니다 . 객체의 특징은 다형성과 상속이다. )

객체란? 하나의 식빵과 같습니다. 각 식빵이 별개의 물건이기 때문이죠. 이 식빵을 객체 혹은 인스턴스라고 합니다. 그리고 중요한 것은 이 인스턴스라고 불리우는 식빵은 식빵을 만드는 틀인 ‘클래스’ 라는 것을 통해 만들어집니다.

q. 이 틀이 존재 하는 이유가 무엇 일까요 ? => 특정 형태를 가진 객체를 대량 생산하기 위함

그렇다면 식빵의 틀에 국화빵 혹은 붕어빵재료를 넣으면 어떻게 될까요 ? 맛과 모양이 엉망이라도 식빵의 모양을 한 틀에 구웠기 때문에 형태와 일부 특성은 식빵을 닮았을 것 입니다.

방금 문장에서 핵심은 ‘형태’ 입니다. 다시 클래스를 정리하면 “특정 형태를 가진 객체를 만들기 위한 레시피” 입니다.

(기존 내용 파기)

ppt넣는곳

**코드 설명**

이론적인 개념들은 잠시 멈추고 이제 객체를 직접 만들어 보도록 하겠습니다. 객체를 만드는 방법은 es6이전과 이후에 많은 변화를 겪었기 때문에 먼저 es6이전에는 어떻게 객체를 만들었는지 보도록 하겠습니다.

**객체 직접 할당**

먼저 es5이전에는 객체리터럴이라고 하는 것을 변수에 할당하여 객체를 만들었습니다. 이 객체리터럴은 함수나 제어문 등에서 쓰이던 블락{} 인데, 변수에 블락을 할당하게 되면 이때부터는 객체 리터럴이라고 불리며 변수에 객체를 사용할 수 있도록 합니다.

처음은 식빵 객체를 손으로 하나하나 빚는 방법입니다.

var bread = {

*//객체의 프로퍼티는 키와 밸류의 쌍으로 이루어져있다. 프로퍼티는 key : value의 형태로 작성하며,*

*//' , '쉼표로써 프로퍼티를 구분 한다.*

name : "우유 식빵",

flour : "밀가루",

milk : "우유",

}

이때 객체를 콘솔로 찍어보면 다음과 같이 나오는 것을 볼 수 있습니다.

예시) console.log(bread) *// {name : "우유 식빵", flour : "밀가루", milk : "우유",}*

이 객체의 프로퍼티를 변경하거나 접근 하고 싶으면 다음과 같이 작성한다.

console.log(bread.flour);*//"밀가루"*

bread.flour = "쌀가루";

console.log(bread.flour)*//"쌀가루"*

지금처럼 객체리터럴을 변수에 직접 할당하여 정의하는 것은 바람직하지 않습니다. 왜냐면 객체를 대량 생산할 수 없기 때문이다. 또한 객체를 처음부터 커스터마이징 할 수 없기 때문이다.

**함수를 이용한 객체 생성 방법**

*// 2. 함수를 이용한 객체 생성 방법*

function milkBread(flour,milk){

*//객체 리터럴을 반환한다.*

return {

name : "우유 식빵",*//고정*

flour : flour,

other : milk,

}

}

var first = milkBread("밀가루","딸기 우유");

var second = milkBread("쌀가루","초코 우유");

함수를 이용한 방법은 객체 리터럴을 반환함으로써 변수에 객체를 할당해주는 방법입니다. 이 방법을 쓰면 전달인자로 전달 받은 값들을 매개변수에 따라 프로퍼티를 달리 할당 할 수 있다.

**개별 적인 인스턴스**

여러분들이 보기에 first변수와 second변수는 프로퍼티가 다르기 때문에 first === second의 결과는 false라고 생각하실겁니다. 실제로 찍어보면 그렇구요.

console.log(first === second); *// false*

그렇다면 first와 동일한 프로퍼티를 가진 녀석을 만들어 비교 해보도록 하겠습니다.

“ 결과는 어떻게 될까요 ? “

var third = milkBread("밀가루","딸기 우유");

console.log(first === third); *// false*

정답은 false입니다. 그 이유는 각기 다른 인스턴스는 개별적인 존재이기 때문입니다. 예를 들어 우리는 생긴 것도 하는 것도 똑같은 일란성 쌍둥이를 닮았다고 할 수 있지만, 똑같은 하나의 사람이라고 하지 않을 겁니다. 또 다른 예로 같은 식빵틀에서 나와 생김새와 맛도 똑같은 두개의 식빵을 우리는 같은 존재라고 하지 않습니다.

객체 또한 메모리 내부에서 주소를 가지고 있는 포인터 형태이기 때문에 인스턴스가 가지고 있는 주소가 각기 다르기 때문에 === 일치 연산자의 결과가 false라고 출력되는 것입니다. 이런 특성들 덕분에 우리는 객체를 이용해 조금 더 현실을 반영한 프로그래밍을 만들 수 있는 것 입니다.

**Class를 이용한 객체 생성 방법**

이전 까지는 객체를 함수로 만들었기 때문에 조금 코드의 일관성이나 가독성이 그렇게 좋지 못했습니다. Es6에서는 클래스 문법이 도입되었기 때문에 클래스와 함수가 확실히 분리 될 수 있게 되었습니다. 아까와 같은 내용을 es6문법으로 만들고 메서드(기능)을 만들어 보도록 하겠습니다.

class milkBread2{*//클래스 선언 후 클래스명을 작성한 뒤 블럭을 연다*

constructor(flour,milk){

this.name = "식빵";

this.flour = flour;

this.milk = milk;

}

moracmorac(){

console.log("모락 모락")

}

}

처음 보는 것들이 많이 생겼는데, constructor는 이 객체로 전달 받은 인자를 배치시키기 위한 생성자 함수 입니다. 또한 이 constructor는 return 값이 존재 하지 않습니다. 그리고 this라는 것이 생겼습니다. 이 this는 객체 내부에서 자신을 가리키는데 사용됩니다.

이 클래스로 객체를 생성할 때는 다음과 같이 작성합니다.

let four = new milkBread2("밀가루","우유");

클래스명 옆에 new라는 생성자를 붙여 만든다. 이때 전달인자로 값을 전달 하면 생성자 함수에서 전달인자로 받은 값들을 처리 할 수 있게 된다.

그리고 이제 메서드를 사용해보도록 하겠습니다. 이 메서드는 우리가 프로퍼티에 접근 하듯 객체명에 ‘.’을 찍고 호출자 ()를 적어주면 됩니다.

four.moracmorac(); *//모락 모락*

**리터럴에 메서드를 정의하는 방법**

function milkBread(flour,milk){

*//객체 리터럴을 반환한다.*

return {

name : "우유 식빵",*//고정*

flour : flour,

other : milk,

moracmorac : function(){

console.log("모락 모락")

}

}

}

메서드명 : 무기명 함수(매개변수){} 블락의 형태로 메서드를 만들면 됩니다.

**This의 증명**

클래스 내부에 returnThis라는 메서드를 만든다.

class milkBread2{*//클래스 선언 후 클래스명을 작성한 뒤 블럭을 연다*

constructor(flour,milk){

this.name = "식빵";

this.flour = flour;

this.milk = milk;

}

moracmorac(){

console.log("모락 모락")

}

returnThis(){

return this;

}

}

이 정체불명의 this를 확인 해보려 return this하였다.

console.log(four.returnThis()); *// milkBread2 { name: '식빵', flour: '밀가루', milk: '우유' }*

콘솔로 해당 메서드를 호출하여 찍어보니 방금 만든 four객체가 반환된 것을 확인 할 수 있다.

더 나아가 four와 해당 메서드가 일치하는 지 보자

console.log(four === four.returnThis()); *//결과는 true*

결과는 true . 이 this는 객체 스스로를 내부에서 칭하는 키워드 이다.

**쉬는 시간**

**객체를 다시 설명**

( 이제 세세한 부분으로 나누어서 설명한다. )

~~객체=> object~~

객체는 영어로 오브젝트 object이다.

간단한 예시

var zero = {

firstName: 'Zero',

lastName: 'Cho'

};

그 다음에는 속성에 대한 정의설명

속성(Property)

=>

객체 안을 보면 firstName과 lastName이 왼쪽에 있고, 'Zero'와 'Cho'가 오른쪽에 있네요.

콤마로 구분되는 것들을 객체의 속성이라고 부릅니다.

zero 객체에는 firstName: 'Zero'와 lastName: 'Cho'까지 두 개의 속성이 있는거죠.

속성끼리는 쉼표로 구분해줍니다.

위에서는 보기 좋게 줄 바꿈을 했지만, 꼭 줄바꿈을 해야하는 것은 아닙니다.

var zero = { firstName: 'Zero', lastName: 'Cho' }; 처럼

쉼표로 구분되기만 하면 충분합니다.

키와 값에 대한 부분적인 설명을 한다.

속성에서 firstName과 lastName 같은 것들을 객체의 키라고 부르고

'Zero'와 'Cho'를 값이라고 부릅니다.

즉, 속성은 키: 값의 관계로 이루어져있죠. 키는 속성명이라고 생각하시면 됩니다.

객체한테 속성의 이름을 부여하는 방법

zero.firstName; // 'Zero'

zero['firstName']; // 'Zero' (이렇게도 가능합니다)

zero.lastName; // 'Lee'

zero['lastName']; // 'Lee'

마침표를 이용해서 객체안의 속성들에 접근하는 방식이고

[]안에 속성명을 적어서 접근할 수도 있습니다.

예시를 하나 든다

var zero = {

body: {

height: 180,

weight: 64

}

};

zero.body.height; // 180

이렇게 마침표로 계속 찍으면서 생성이 가능하므로

복잡한 데이터구조도 짤수 있습니다.

new를 사용해서 만드는 방법도 존재하지만

보통 {}를 사용합니다 {}을 이용해서 만든 객체를

객체 리터럴이라고 부릅니다.

반대로 아닌경우에는 그냥 리터럴이라고 부릅니다.

객체란 결국 껍데기를 말하는 것이고 실제 객체를 완성하는 구성요소는

프로퍼티와 메소드입니다.

위의 예시로 식빵에 대해서 설명을 했는데

이해하셨나요??

이제 객체에 key와 value를 추가하는 방법에 대해서 알아보도록 하겠습니다

먼제 예시문을 들고 시작하겠습니다.

var person = {

name: "Surim Son"

};

console.log(person.name); // 결과 : "Surim Son"

console.log(person.age); // undefined (존재하지 않는 key/value)

// 추가하는 방법

person.age = 22; 혹은 person["age"] = 22;

console.log(person.age); // 결과 : 22

이런식으로 추가할 수 있습니다

그러면 crud에서 수정도 가능해야 하겠죠?

수정방법은 이렇게 하시면 됩니다.

var person = {

name: "Surim Son",

age: 22

};

console.log(person.name); // "Surim Son"

console.log(person.age); // 22

// 수정하는 방법

person.age = 23; 혹은 person["age"] = 23;

console.log(person.age); // 23

삭제는

ar person = {

name: "Surim Son",

age: 22

};

console.log(person.name); // "Surim Son"

console.log(person.age); // 22

// 삭제하는 방법

delete person.age;

console.log(person.age); // undefined (더 이상 존재하지 않는 key/value)

이렇게 하시면 됩니다.

제가 오늘 상속까지 진도를 나갈생각이었는데

러닝자바스크립트로 공부하다가 제가 이해를 잘 못해서

내일 종호행님이 오시면 강의를 해주실 겁니다.

감사합니다.