- 3주차 : 클래스(생성자 함수)와 프로토타입, this -

/* 객체지향 프로그래밍 (OOP - Object Oriented Programming) */

- 변수나 함수 등을 작성하고 여러 곳에다가 가져다가 사용할 수 있도록 작성하는 것을 말한다.
- 예를 들어 타다(?) 기능을 만들고 이 기능을 자동차, 자전거, 바이크 등에 사용할 수 있다. 이렇듯 재사용을 고려하여 개발하는 것을 “OOP”라고 한다.
- 객체지향 프로그래밍 작업을 도와주는 것이 바로 “클래스”이다. 클래스는 속성(맴버변수)과 메서드(맴버함수)로 구성되어 있다.
- 자바스크립트에서는 프로토타입 기반으로 OOP를 할 수 있기 때문에, **프로토타입 기반 객체지향 프로그래밍**이라고 이해하는 것이 좋다.

/* 클래스 (생성자 함수) */

- 생성자 함수란 객체를 생성하는 함수를 말한다. Java와 같은 언어에서는 class가 있지만, 자바스크립트에서는 class가 생성자 함수 역할을 대신한다. (단, ES6에서는 class 개념이 추가되었다.)
- 생성된 클래스(생성자 함수)를 재사용하고 싶다면 new 연산자를 사용하여 새로운 객체, 즉 인스턴스를 생성하면 된다.
- 보통 클래스를 붕어빵 틀, 인스턴스를 붕어빵에 비유하곤 한다.
- 클래스(생성자 함수)는 일반 함수와 구분하기 위해 대문자로 작성된다.
- this는 생성자 함수 자신을 가르키고, this에 저장된 내용물은 new를 통해 객체를 만들 때 해당 객체에 적용

```
function Person(firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;

    this.fullName = function() {
        console.log(this.firstName + " " + this.lastName);
    }
}

var kim = new Person("inkweon", "Kim");
var park = new Person("David", "Park");

kim.fullName();
park.fullName();

console.log(kim);
console.log(park);
```

- 이 경우 fullName()를 인스턴스별로 따로 생성하고 있어 불필요하게 자원을 낭비하게 된다.
- 이 때 사용되는 것이 prototype 프로퍼티이다.

/* prototype */

```
function Person(firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
}

Person.prototype.fullName = function() {
    console.log(this.firstName + " " + this.lastName);
};


var kim = new Person("inkweon", "Kim");
var park = new Person("David", “Park");

kim.fullName();
park.fullName();

console.log(kim);
console.log(park);
```

- prototype 프로퍼티로 함수를 생성하면 인스턴스별로 함수를 생성하지 않아 메모리를 세이브할 수 있다.

/* __proto__ */

- "__proto__"는 자바스크립트 조상을 탐색하는 코드이다.
- 조상을 탐색하면 그 후손(?)이 조상이 갖고 있는 프로퍼티와 메서드를 전수하고 이를 사용할 수 있다.

```
var arr = new Array(1);
console.dir(arr);

var str = new String("K");
console.dir(str);

var func = new Function("x, y", "return x * y");
console.dir(func);
```

- 참고 : https://blog.naver.com/gi_balja/221254920628

```
var person = new Object();

person.name = "Kim In Kweon";
person.age = 20;
person.sayHi = function() {
    console.log("Hi")
};

console.dir(person);
console.log(person.__proto__ === Object.prototype);  // true 출력
```

- 이는 객체 뿐 아니라 문자열, 배열, 함수에서도 동일하게 나타난다.

```
var str = new String("Hello World");
console.log(str.__proto__ === String.prototype); // true 출력

var arr = new Array(10, 20, 30);
console.log(arr.__proto__ === Array.prototype); // true 출력

var func = new Function("x, y", "return x * y");
console.log(func.__proto__ === Function.prototype); // true 출력
```

- 그리고 이들 String.prototype, Array.prototype, Function.prototype 객체는 최종적으로 Object.prototype 객체를 조상으로 갖고 있다.

```
console.log(String.prototype.__proto__ === Object.prototype); // true 출력
console.log(Array.prototype.__proto__ === Object.prototype); // true 출력
console.log(Function.prototype.__proto__ === Object.prototype); // true 출력
```

- 이러한 이유로 자바스크립트에서 문자열, 배열, 함수 모두를 객체라고 얘기한다.

- 용어 정리

```
// str : String.prototype 객체의 인스턴스
// String() : 생성자 함수
var str = new String("Hello World");


// __proto__ : str의 조상을 가르키는 프로퍼티
// String.prototype 객체 : str의 조상
console.log(str.__proto__ === String.prototype);
```

/* constructor */

- 각각의 prototype 객체는 constructor를 갖고 있다. constructor은 인스턴스 입장에서 자신을 만든 클래스(생성자 함수)를 가르키게 된다.

```
var str = new String("Hello World");

console.log(str.constructor === String); // true 출력
```

**** 한 단계 더 추가하자면 String.prototype.construcotr는 String() 생성자 함수를 가르키게 된다.**

```
var str = new String("Hello World");

console.log(str.constructor === String);
console.log(String.prototype.constructor === String); // 출력
```

**** String 생성자 함수의 조상을 탐색하면 Object.prototype 객체가 등장한다.**

```
var str = new String("Hello World");

console.log(String.__proto__ === Function.prototype); // true 출력
console.log(Function.prototype.__proto__ === Object.prototype); // true 출력
```

/* 사용자가 정의한 생성자 함수의 조상 탐색하기 */

```
function Person(firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
}

Person.prototype.fullName = function() {
    console.log(this.firstName + " " + this.lastName);
};


var kim = new Person("inkweon", "Kim");


// 인스턴스 kim과 Person.prototype 객체의 조상
console.log(kim.__proto__ === Person.prototype); // true 출력
console.log(Person.prototype.__proto__ === Object.prototype); // true 출력
```

```
// Person 생성자 함수의 조상
console.log(Person.__proto__ === Function.prototype); // true 출력
console.log(Function.prototype.__proto__ === Object.prototype); // true 출력
```

```
// kim, Person.prototype의 constructor
console.log(kim.constructor === Person); // true 출력
console.log(Person.prototype.constructor === Person); // true 출력
```

- 참고 : https://blog.naver.com/gi_balja/221256287861

/* this */

- “this”는 일반적으로 메서드를 호출한 객체가 저장되어 있는 속성을 말한다.
- “this”의 대상은 다음 상황에 따라 달라진다.

1. 일반 함수에서 this -> window
2. 중첩 함수에서 this -> window
3. 이벤트에서 this -> 이벤트 객체
4. 메서드에서 this -> 메서드 객체
5. 메서드 내부의 중첩 함수에서 this -> window

```
var personOne = {
    name: "Kim",
    age: 100,
    getName: function() {
        console.log(this.name);
    }
};

var personTwo = {
    name: "Park",
    age: 999,
    getName: function() {
        console.log(this.name);
    }
};

personOne.getName();
personTwo.getName();
```

- 중첩 함수의 안의 “this”는 window를 가르킨다.

```
var personOne = {
    name: "Kim",
    age: 100,
    getName: function() {
        function test() {
            console.log(this);
        }
        test();
    }
};


personOne.getName();
```

- 이 경우 that 변수 안에 “this”를 담아 중첩 함수의 this 경로(?)를 수정해야 한다.

```
var personOne = {
    name: "Kim",
    age: 100,
    getName: function() {

        var that = this;

        function test() {
            console.log(this);
        }
        test();
    }
};


personOne.getName();
```

- **템플릿 리터럴 (Template literal) : `${}`**
- ES6에 등장한 새로운 문자열 표기법으로 백틱(backtick) 문자 `를 사용한다.

```
var name = "기발자";

console.log("내 이름은 " + name + "입니다.");
console.log(`내 이름은 ${name}입니다.`);
```

/* 슬라이드 효과 */

```
(function () {

    var customers = [];
    var index = 0;

    // 생성자 함수
    function Customer(name, img, text) {
        this.name = name;
        this.img = img;
        this.text = text;
    }

    function createCustomer(name, img, text) {
        var fullImg = `img/customer-${img}.jpg`;
        var customer = new Customer(name, fullImg, text);
        customers.push(customer);
    }

    createCustomer("아이언맨", 0, "Hello World");
    createCustomer("토르", 1, "Nice to meet you");
    createCustomer("헐크", 2, "Great!!");
    createCustomer("캡틴 아메리카", 3, "Welcome to Real World");

    console.log(customers);
    console.log(customers[0].name);



    document.querySelectorAll('.btn').forEach(function(item) {

        …… 이전과 동일한 코드

    })


})();
```

/* 글자 타이핑 효과 */

```
(function () {

    var arr = [10, 20, 30];
    var index = 0;

    var loop = function () {

        var current = index % arr.length;
        console.log(arr[current]);
        index++;

        setTimeout(function() {
             return loop();
        }, 2000);

    }

    loop();

})();
```

```
(function() {

    // TypeWriter 생성자 함수
    function TypeWriter(txtElement, words, wait) {

        this.txtElement = txtElement;
        this.words = words;
        this.wait = parseInt(wait, 10);

        this.txt = "";
        this.wordIndex = 0;
        this.isDeleting = false;

        this.type();

    }

})();
```

```
(function() {
    ……
    TypeWriter.prototype.type = function() {

        var current = this.wordIndex % this.words.length;    // 배열 안 텍스트 좌표
        var fulltxt = this.words[current];                   // 배열 안 텍스트 가져오기

        if(this.isDeleting) {
            // 전체글자 기준 뒤에서 한 글자씩 제거
            this.txt = fulltxt.substring(0, this.txt.length - 1);

        } else {
            // 전체 글자 기준 앞에서 한 글자씩 붙이기
            this.txt = fulltxt.substring(0, this.txt.length + 1);
        }

        // 글자가 들어갈 HTML 코드
        this.txtElement.innerHTML = `<span class="txt">${this.txt}</span>`;

        var typeSpeed = 300;        // 기본 속도값

        if(this.isDeleting) {
            typeSpeed /= 2;         // 글자가 사라지는 속도값
        }

        if(!this.isDeleting && this.txt === fulltxt) {    // 전체 단어가 완성되었다면
            typeSpeed = this.wait;
            this.isDeleting = true;
        } else if(this.isDeleting && this.txt === "") {   // 단어가 모두 사라졌다면
            this.isDeleting = false;
            this.wordIndex++;
            typeSpeed = 500;
        }
        var that = this;
        setTimeout(function() { return that.type(); }, typeSpeed);
    };
})();
```

- substring() 설명 : https://webclub.tistory.com/20

```
(function() {

    ……
    ……

    var txtElement = document.querySelector('.txt-type');
    var words = ["에이지 오브 울트론", "인피니티 워", "엔드게임"];
    var wait = 3000;

    new TypeWriter(txtElement, words, wait);

})();
```