Classes

Class는 객체를 생성하기 위한 템플릿입니다. 클래스는 데이터와 이를 조작하는 코드를 하나로 추상화합니다. 자바스크립트에서 클래스는 프로토타입을 이용해서 만들어졌지만 ES5의 클래스 의미와는 다른 문법과 의미를 가집니다.

Class는 사실 "특별한 함수"입니다. 함수를 함수 표현식과 함수 선언으로 정의할 수 있듯이 class 문법도 class 표현식 and class 선언 두 가지 방법을 제공합니다.

Class 선언

Class를 정의하는 한 가지 방법은 class 선언을 이용하는 것입니다. class를 선언하기 위해서는 클래스의 이름과 함께 `class` 키워드를 사용해야 합니다

```
class 클래스명 {
    constructor(매개변수, 매개변수){
        this.속성 = 매개변수
        this.속성 = 매개변수
    }
    매서드명(){ 실행문 }
}
오브젝트명 선언 = [ new 클래스명(인수, 인수,인수),
                    new 클래스명(인수, 인수,인수) ];
```

```
class Rectangle {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
}
```

함수 선언과 클래스 선언의 중요한 차이점은 함수의 경우 정의하기 하기 전에 호출할 수 있지만, 클래스는 반드시 정의한 뒤에 사용할 수 있다는 점입니다.

```
class Rectangle {}
const p = new Rectangle();
```

```
const p = new Rectangle(); // ReferenceError
class Rectangle {}
```

extends를 통한 클래스 상속(sub classing)

- extends 키워드는 클래스 선언이나 클래스 표현식에서 다른 클래스의 자식 클래스를 생성하기 위해 사용됩니다.
- super 키워드는 객체의 부모가 가지고 있는 메서드를 호출하기 위해 사용됩니다.
- subclass에 constructor가 있다면, "this"를 사용하기 전에 가장 먼저 super()를 호출해야 합니다.

```
class 부모클래스명 {
  constructor(매개변수, 매개변수){
        this.속성 = 매개변수
        this.속성 = 매개변수
        }
  매서드명(){
        실행문
        }
}
```

```
class 자식클래스명 extends 부모 클래스명 {
  constructor(매개변수, 매개변수){
        super()
        this.속성 = 매개변수
        }
  매서드명(){
        실행문
        }
  }
```

```
class Car {
 constructor(name) {
 this.brand = name;
}
present() {
return 'I have a ' + this.brand; }
}
class Model extends Car {
  constructor(name, mod) {
    super(name);
    this.model = mod;
}
  show() {
  return this.present() + ', it is a ' + this.model ;
  }
}
const mycar = new Model("Ford", "Mustang");
document.write(mycar.show());
```

구조 분해 할당 (Destructuring)

구조 분해 할당 구문은 배열이나 객체의 속성을 해체하여 그 값을 개별 변수에 담을 수 있게 하는 JavaScript 표현식입니다.

구조화 이전버전

```
const str = ['red','blue','green'];
const car = str[0];
const truck = str[1];
const suv = str[2];
```

구조화1

```
const str = ['red','blue','green'];
const [car, truck, suv] = str;
console.log(car);   // red
```

구조화2

```
const str1 = ['red','blue','green'];
const [car1, ,suv1] = str1;
console.log(car1);    //red
```

함수 사용

```
function calc(a,b){
const add = a+b;
const sub = a-b;
const multi = a*b;
return [add, sub, multi];
}
const [add,sub,multi] = calc(4,7);
console.log(add);
console.log(sub);
console.log(multi);
```

객체사용

```
const vehicleOne = {
  brand: 'brand1',
  model: 'model1',
  type: 'car',
  year: 2021,
  color: 'red'
}
  myVehicle(vehicleOne);

function myVehicle({type, color, brand, model}) {
  const message = 'My ' + type + ' is a ' + color + ' ' + brand + ' ' + model + '.';
  document.getElementById("demo").innerHTML = message;
}
```

스프레드 연산자(...)

전개 구문을 사용하면 배열이나 문자열과 같이 반복 가능한 문자를 0개 이상의 인수 (함수로 호출할 경우) 또는 요소 (배열 리터럴의 경우)로 확장하여, 0개 이상의 키-값의 쌍으로 객체로 확장시킬 수 있습니다

```
const no01 = [1, 2, 3];
const no02 = [4, 5, 6];
const noHap= [...no01, ...no02];
console.log(noHap);
```

정규표현식

정규 표현식, 또는 정규식은 문자열에서 특정 문자 조합을 찾기 위한 패턴입니다. JavaScript에서는 정규 표현식도 객체로서, RegExp의 exec()와 test() 메서드를 사용할 수 있습니다.

1. 정규표현식 형식

/패턴/플래그

- 슬래시(/) "사이"에는 매칭시킬 "패턴"을 써준다.
- 슬래시(/) "다음"에는 옵션을 설정하는 "플래그"를 써준다.

(플래그는 하나만 찾을지, 모두 다 찾을지 등을 설정하는 옵션이라고 보면 됩니다.)

2. 정규표현식 매칭 패턴(문자, 숫자, 기호 등)

a-zA-Z	영어알파벳(-으로 범위 지정)
ㄱ-ㅎ가-힣	한글 문자(-으로 범위 지정)
0-9	숫자(-으로 범위 지정)
₩d	숫자
₩D	숫자가 아닌 것
₩w	영어 알파벳, 숫자, 언더스코어(_)
₩W	/w 가 아닌 것
₩s	space 공백
₩S	space 공백이 아닌 것
₩특수기호	특수기호

3. 정규표현식 검색 패턴

|	OR
[]	괄호안의 문자들 중 하나
[^문자]	괄호안의 문자를 제외한 것
^문자열	특정 문자열로 시작(괄호 없음 주의!)
문자열$	특정 문자열로 끝남
()	그룹 검색 및 분류(match메서드에서 그룹별로 묶어줌)

(?: 패턴)	그룹 검색(분류X)
₩b	단어의 처음/끝
₩B	단어의 처음/끝이 아님

4. 정규표현식 갯수(수량) 패턴

특정 패턴이 몇번 반복되는지도 필터링 가능합니다.

?	최대 한번(없음 || 한개)
*	없거나 있거나 (없음 || 있음): 여러개 포함
+	최소 한개(한개 || 여러개)
{n}	n개
{Min,}	최소 Min개 이상
{Min, Max}	최소 Min개 이상, 최대 Max개 이하

5. 정규표현식 플래그

g	Global: 모든 문자 검색(안 쓰면 매칭되는 첫 문자만 검색)
i	Ignore Case: 대소문자 구분 안함
m	Multi line: 여러 행의 문자열에 대해 검색

6. 정규표현식 주요 메서드

자바스크립트 코드 상에서는 아래 메서드를 통해 패턴을 검사하고, .매칭되는 문자열을 추출, 변환합니다.

("문자열").**match**(/정규표현식/플래그)	"문자열"에서 "정규표현식"에 매칭되는 항목들을 배열로 반환
("문자열").**replace**(/정규표현식/, "대체문자열")	"정규표현식"에 매칭되는 항목을 "대체문자열"로 변환
("문자열").**split**(정규표현식)	"문자열"을 "정규표현식"에 매칭되는 항목으로 쪼개어 배열로 반환
(정규표현식).**test**("문자열")	"문자열"이 "정규표현식"과 매칭되면 true, 아니면 false반환
(정규표현식).**exec**("문자열")	match메서드와 유사(단, 무조건 첫번째 매칭 결과만 반환)

JSON

JSON(제이슨 JavaScript Object Notation)은 속성-값 쌍(attribute–value pairs and array data types (or any other serializable value)) 또는 "키-값 쌍"으로 이루어진 데이터 오브젝트를 전달하기 위해 인간이 읽을 수 있는 텍스트를 사용하는 개방형 표준 포맷이다. 비동기 브라우저/서버 통신 (AJAX)을 위해, 넓게는 XML(AJAX가 사용)을 대체하는 주요 데이터 포맷이다. 특히, 인터넷에서 자료를 주고 받을 때 그 자료를 표현하는 방법으로 알려져 있다. 자료의 종류에 큰 제한은 없으며, 특히 컴퓨터 프로그램의 변수 값을 표현하는 데 적합하다.

JSON의 기본 자료형은 다음과 같다:

한 파일에는 하나의 데이터형만 사용할 수 있다.

속성명은 꼭 “”를 사용한다.

- 수(Number)

- 문자열(String): 0개 이상의 유니코드 문자들의 연속. 문자열은 큰 따옴표(")로 구분하며 역슬래시 이스케이프 문법을 지원한다.
- 참/거짓(Boolean): true 또는 false 값
- 배열(Array): 0 이상의 임의의 종류의 값으로 이루어진 순서가 있는 리스트. 대괄호로 나타내며 요소는 쉼표로 구분한다.
- 객체(Object): 순서가 없는 이름/값 쌍의 집합으로, 이름(키)이 문자열이다.
- null: 빈 값으로, null을 사용한다.
- undefined 는 사용할 수 없다.

JSON.stringify()	JavaScript 값이나 객체를 JSON 문자열로 변환합니다.
JSON.parse()	JSON 문자열의 구문을 분석하고, 그 결과에서 JavaScript 값이나 객체를 생성합니다.

Import , export

Import

import 문은 다른 모듈에서 내보낸 바인딩을 가져올 때 사용합니다.

경로 지정시 js 파일은 확장자를 생략해도 된다.

```
Import  이름 from ‘ 경로’ ;
Import  {모듈에서 하나의 멤버만 가져올 때 } from ‘ 경로’ ;
```

import myModule from “my-module";

import myDefault, {foo, bar} from “my-module";

Export

export 문은 JavaScript 모듈에서 함수, 객체, 원시 값을 내보낼 때 사용합니다. 내보낸 값은 다른 프로그램에서 import 문으로 가져가 사용할 수 있습니다.

내보내는 모듈은 "use strict"의 존재 유무와 상관없이 무조건 엄격 모드입니다. export 문은 HTML 안에 작성한 스크립트에서는 사용할 수 없습니다

내보내기에는 두 종류, 유명(named)과 기본(default) 내보내기가 있습니다. 모듈 하나에서, 유명 내보내기는 여러 개 존재할 수 있지만 기본 내보내기는 하나만 가능합니다. 각 종류는 위의 구문 중 하나와 대응합니다

Named 방법

```
// 먼저 선언한 식별자 내보내기
export { myFunction, myVariable };

// 각각의 식별자 내보내기
// (변수, 상수, 함수, 클래스)
export let myVariable = Math.sqrt(2);
export function myFunction() { ... };
```

Default 방법

```
// 먼저 선언한 식별자 내보내기
export { myFunction as default };

// 각각의 식별자 내보내기
export default function () { ... };
export default class { ... }
```