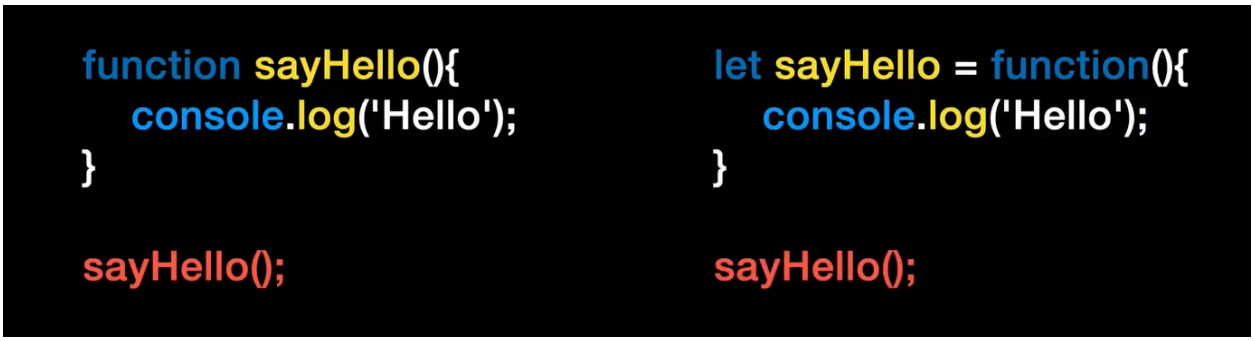
JS기본-4

태그	
날짜	@2023년 5월 30일

함수 표현식

화살표 함수

함수 선언문과 함수 표현식



함수 선언문 : 어디서든 호출 가능

자바스크립트는 위에서 아래로 한줄씩 차례대로 읽으면서 실행한다. 이렇게 순차적으로 실행되고 즉시 결과를 반환하는 언어를 인터프리터 언어(Interpreted language)라고 한다.

```
function sayHello(){
  console.log('Hello');
}

sayHello();

--------------------------------
sayHello();
function sayHello(){
  console.log('Hello');
}

이 코드도 실행된다.
왜일까
이건 자바스크립트 내부 알고리즘 때문이다.
자바스크립트의 실행 전 초기화 단계에서 코드의 모든 함수
선언을 찾아서 생성해준다.
즉 눈으로 봤을때는 저 모습이지만 저 함수를 사용할 수 있는
범위는 코드 위치보다 위로 올라가는 것이다.
이를 호이스팅(Hoisting)이라고 한다.
오해 할 수 있지만 코드위치가 실제로 올라가는 것은아니다.
```

함수 표현식 : 코드에 도달하면 생성

```
... // 1
... // 2
let sayHello = function(){ //3 생성
    console.log('Hello');  // 사용가능
}
sayHello(); // 4
```

함수 표현식은 자바스크립트가 한줄 씩 읽으면서 실행하고 해당 코드의 도달해서야 비로소 생성된다.

그래서 무엇이 더 좋은가

함수 선언문 vs 함수 표현식

함수 선언문이 자유롭고 편하게 할 순 있다.

화살표 함수 (arrow funcion)

```
let add = function(num1, num2){
  return num1 + num2;
}

이 함수를 화살표 함수로 바꾸면

let add = (num1, num2)=>{
  return num1 + num2;
}

이렇게 바꿀 수 있다.
function이 사라지고 대신 괄호 뒤쪽에 화살표가 생긴다.
크게 다르지 않다.
지금 이 예제는 코드 부분이 한줄이고 리턴문이 있기 때문에

let add = (num1, num2) =>(
    num1 + num2;
)

이렇게 바꿀수 있다.
리턴문이 한줄이라면 괄호도 생략 할 수 있다.
let add = (num1, num2) => num1 + num2;

인수가 하나라면 괄호를 생략할 수 있다.
let sayHello = name => `Hello, ${name}`;

인수가 없는 함수라면 괄호를 생략할 수 없다.
let showError =()=>{
  alert('error!');
}

리턴문이 있다고 해도 리턴 전에 여러 줄의 코드가 있을 경우
()괄호를 사용할 수 없다.
let add= function(num1, num2){
    const result = num1 + num2;
    return result;
}
```

```
// 함수 표현식

showError();

let showError = function(){
  console.log('error');
}
```

에러 발생

```
// 함수 선언문
showError();
function showError(){
  console.log("error")
}
```

"error"

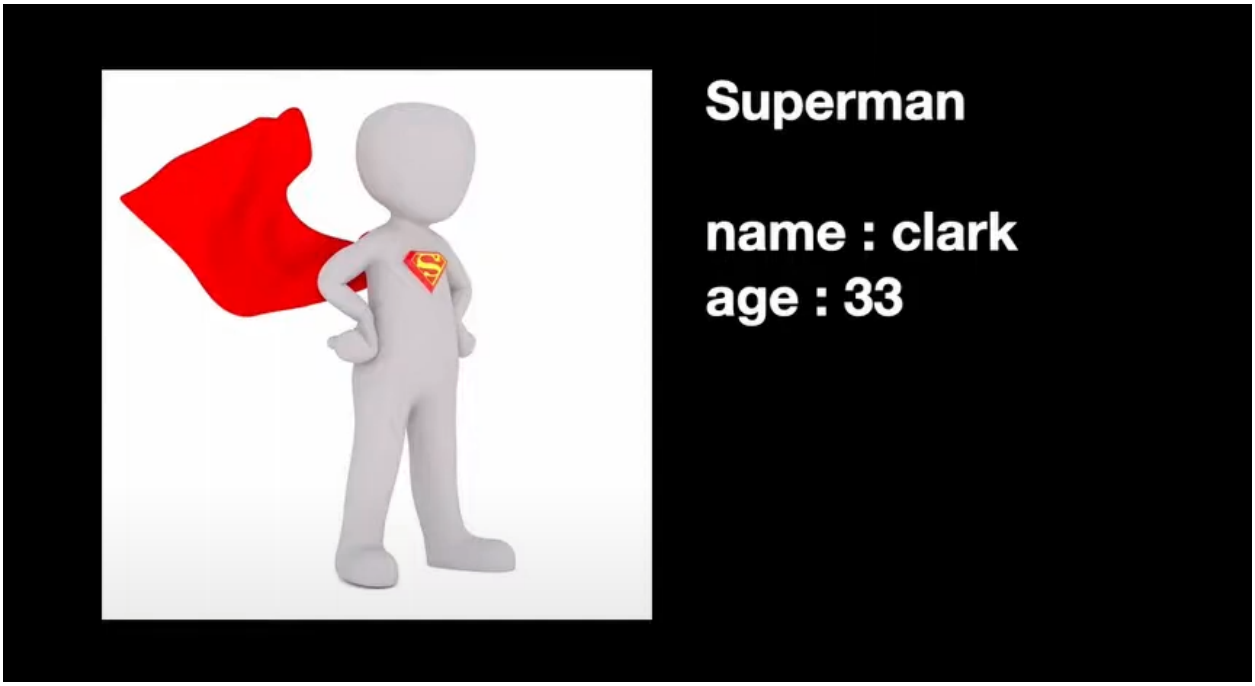
나온다.

```
// 화살표 함수

let showError =()=> {
  console.log("error")
}


// 화살표 함수
let showError =()=> {
  console.log("error")
}
showError();
```

"error"

```
// 화살표 함수

const sayHello = (name) =>{
  const msg = `Hello, ${name}`;
  console.log(msg);
};
```

객체 Object



위 사진으로 객체를 만든다

```
const superman={
  name : 'clark',
  age : 33,
}

객체이다.
객체는 중괄호로 작성하고 key와 value로 구성된 프로퍼티가
들어간다.각 프로퍼티는 쉼표로 구분한다.
마지막 쉼표는 생략가능하지만 추가나 수정할때 용이하게 작성한다.
```

object - 접근 , 추가 ,삭제

```
const superman = {
    name : 'clark',
    age : 33,
}

접근
superman.name // 'clark'
```

```
superman['age'] // 33

추가
superman.gender = 'male';
superman['hairColor'] = 'black';

삭제
delete superman.hairColor;
```

object - 단축 프로퍼티

```
const name = 'clark';
const age = 33;

const superman = {
  name : name,
  age : age,
  gender : 'male',
}

객체 name은 clark이고, age는 33이다.
위 코드는

const superman = {
  name, // name : name
  age, // age : age
  gender : 'male',
}

이렇게 쓸 수 있다.
```

object - 프로퍼티 존재 여부 확인

만약 존재하지 않는 프로퍼티에 접근하면 어떻게 될까.

```
const superman = {
    name : 'clark',
    age : 33,
}
```

```
superman.birthDay;
// undefined

에러가 발생하지 않고 undefined가 발생한다.

이때 in 연산자를 사용하면 프로퍼티가 있는지 확인할 수 있다.
'birthDay' in superman;
// false

'age' in superman;
//true

점이나 대괄호를 이용하면 될 텐데 언제 in을 쓸까
어떤 값을 너무 일찍 확신할 수 없을때 사용하면 된다.
함수 인자로 받거나 api통신을 통해 데이터를 받을때.
```

for… in 반복문

for in 반복문을 사용하면 객체를 순회하면서 값을 얻을 수 있다.

```
for (let key in superman){
  console.log(key)
  console.log(superman[key])
}
```

```
// 객체

const superman = {
  name : 'mike',
  age : 30,
}

console.log(superman.name);
console.log(superman['age']);
console.log(superman);

"mike"
30
// [object Object]
{
  "name": "mike",
```

```
  "age": 30
}
```

```
// 객체 추가

const superman = {
  name : 'mike',
  age : 30,
}

superman.hairColor = 'black';
superman['hobby'] = 'football';
console.log(superman);


// [object Object]
{
  "name": "mike",
  "age": 30,
  "hairColor": "black",
  "hobby": "football"
}
```

```
// 객체 삭제
const superman = {
  name : 'mike',
  age : 30,
}

delete superman.age;
console.log(superman);

// [object Object]
{
  "name": "mike"
}
```

```
// 객체
// 이름과 나이를 받아서 객체를 반환하는 함수

function makeObject(name, age) {
  return {
    name : name,
    age : age,
    hobby : 'football'
  }
}

const Mike = makeObject('Mike', 30);
console.log(Mike);

// [object Object]
{
  "name": "Mike",
  "age": 30,
  "hobby": "football"
}

-----------------------------------------------

function makeObject(name, age) {
  return {
    name,
    age,
    hobby : 'football'
  }
}

const Mike = makeObject('Mike', 30);
console.log(Mike);

축약해서 사용가능
```

```
// 프로퍼티 존재여부 확인
function makeObject(name, age) {
  return {
    name,
    age,
    hobby : 'football'
  }
}

const Mike = makeObject('Mike', 30);
console.log(Mike);
```

```
console.log("age" in Mike);
console.log("Birth" in Mike);

// [object Object]
{
  "name": "Mike",
  "age": 30,
  "hobby": "football"
}
true
false
```

보통 이런 경우 그냥 점이나 대괄호로 확인가능하다.

```
// 객체 in
// 나이를 확인하고 성인인지 아닌지 확인하는 함수

function isAdult(user){
    if (user.age < 20){
      return false;
    }
      return true;
}

const Mike = {
  name : 'Mike',
  age : 30
};

const Jane = {
  name : 'Jane'
};

console.log(isAdult(Mike))
console.log(isAdult(Jane))

true
true

Jane은 age가 없는데도 true가 나온다.
user.age가 undefined이기 때문에 false를 반환하고
아래에 true를 반환한다.

--------------------------------------------------

수정
```

```
function isAdult(user){
    if (!('age' in user) ||  // user에 age가 없거나
        user.age < 20){      // 20살 미만이거나
      return false;
    }
      return true;
}

const Mike = {
  name : 'Mike',
  age : 30
};

const Jane = {
  name : 'Jane'
};

console.log(isAdult(Mike))
console.log(isAdult(Jane))

true
false
```

```
// 객체 for ... in

const Mike = {
  name : "Mike",
  age : 30
};

for(key in Mike){ // key는 Mike가 가지고 있는 프로퍼티 다른 문자가 와도 상관없다
  console.log(key)
}

"name"
"age"

-----------------------------------------------------
// 객체 for ... in

const Mike = {
  name : "Mike",
  age : 30
};

for(x in Mike){ // key는 Mike가 가지고 있는 프로퍼티 다른 문자가 와도 상관없다
  console.log(Mike[x]) // Mike
```

```
}

"Mike"
30
```

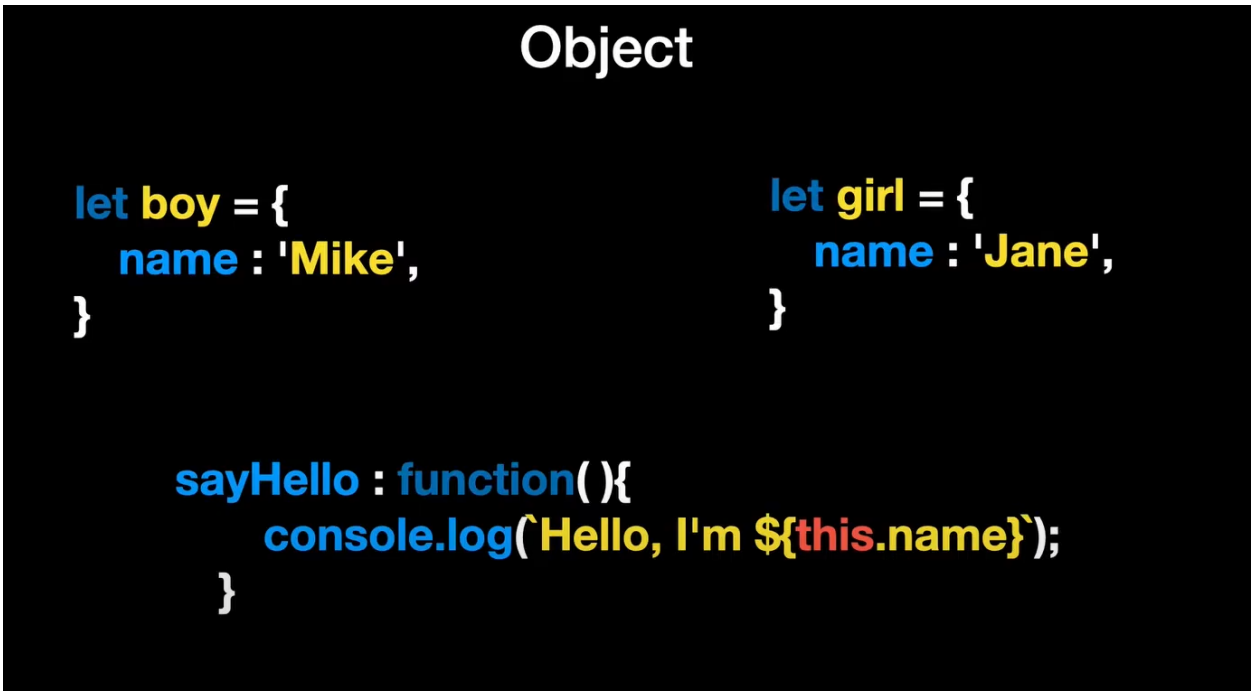
객체 method/ this

```
const superman = {
  name : 'Mike',
  age : 33,
  fly : function(){
    console.log('날아갑니다')
  }
}

이렇게 함수를 추가하고
superman.fly();를 호출하면
날아갑니다. 가 찍힌다.

이렇게 객체 프로퍼티로 할당 된 함수를 method라고 한다.
fly함수가 superman객체의 method인것이다.

단축구문으로도 작성할 수 있다.

const superman = {
  name : 'Mike',
  age : 33,
  fly(){
    console.log('날아갑니다')
  }
}
function 키워드를 생략할 수 있다.
```

객체와 method의 관계

```
const user = {
  name = "mino",
  sayHello : function () {
```

```
  console.log(`Hello, i'm  ???`);
  }
}

method 로그에 객체인 프로퍼티를 넣고 싶다면 어떻게
해야할까?

const user = {
  name = "mino",
  sayHello : function () {
  console.log(`Hello, i'm  ${user.name}`);
  }
}
이렇게 하면된다. 하지만 이 방식은 문제가 발생할 수 있다.
이럴때는 this를 사용하면 된다.

const user = {
  name = "mino",
  sayHello : function () {
  console.log(`Hello, i'm  ${this.name}`);
  }
}

user.sayHello();
// Hello i`m mino
user.sayHello로 호출하면 이 .앞에 user가 sayHello method의
this가 된다.
```




```
boy 와 girl을 만들고 함수를 만들었다.
여기서 this는 아직 결정되지 않았다
이제 이 함수를 각 객체에 method로 넣어준다.
그 후 호출하면 this는 실행하는 시점 즉 런타임에 결정한다.

boy.sayHello(); // i'm Mike
girl.sayHello(); // i'm Jane

만약
sayHello : ()=>{
  console.log(`i'm ${this.name}`);
}
이렇게 화살표 함수로 했다면 값이 전혀 달라지게 된다.
```

화살표 함수는 일반 함수와는 달리 자신만의 this를 가지지않는다.

화살표 함수 내부에서 this를 사용하면, 그 this는 외부에서 값을 가져온다.

```
let boy = {
  name : 'Mike',
  sayHello : ()=>{
```

```
    console.log(this); // 전역객체
  }
}

boy.sayHello();
```

이렇게 화살표 함수로 method를 만들고 코드를 실행하면 this
는 점 앞 boy객체를 가리키지 않는다.
화살표함수는 this를 가지고 있지 않기 때문에 여기서 this를
사용하면 전역 객체를 가리키게 된다.

- 브라우저 환경에서 전역 객체 : window
- node js 환경에서 전역 객체 : global

```
// method

let boy = {
    name : "Mike",
    showName : function () {
      console.log(boy.name)
    }
};

let man = boy;
```

man을 만들어서 boy를 할당
객체를 새로 만든것은 아니고 boy로도 접근할 수 있고, man으로도
접근할 수 있다.
사람은 mike한명이고 별명이 2개인것

```
let boy = {
    name : "Mike",
    showName : function () {
      console.log(boy.name)
    }
};
let man = boy;
man.name = "tom"
console.log(boy.name)
// man의 이름을 tom으로 변경해도
// boy의 이름도 tom으로 찍힌다.
```

```
let boy = {
    name : "Mike",
    showName : function () {
      console.log(boy.name)
    }
};

let man = boy;
boy = null;

man.showName();

// 에러 발생
// 이땐 아래처럼 변경

let boy = {
    name : "Mike",
    showName : function () {
      console.log(this.name)
    }
};

let man = boy;
boy = null;

man.showName();

메소드에서는 객체명을 직접사용하는것보다 this를 쓰는것이 좋다.
```

배열 array

: 순서가 있는 리스트

```
1번에 철수
2번에 영
…
30번에 영수

let students = ['철수', '영희', … '영수'];

배열은 대괄호로 묶어주고  쉼표로 구분해서 만들 수 있다.
```

```
배열을 탐색할 때는 고유 번호를 사용한다.
이를 index라고 한다. index는 0부터 시작한다.
위 예제는 0부터 29까지 있다.

console.log(students[0]);  // 철수
console.log(students[1]); // 영희
console.log(students[29]); // 영수

수정
students[0] = ‘민정’;
console.log(students) → [’민정’,’영희’,…

배열은 문자 뿐만 아니라, 숫자, 객체, 함수
등도 포함 할 수 있다.

let arr = [
  '민수', // 문자
  3, // 숫자
  false, // 불린
  {
    name : 'Mike',
    age : 30,
  }, // 객체
  function(){
      console.log('Test');
  } // 함수
]

length : 배열의 길이

students.length // 30
```

```
배열이 가진 method

push() : 배열 끝에 요소 추가
let days = ['월', '화', '수'];
days.push('목')
console.log(days) // ['월', '화', '수', '목']

pop() : 배열 끝 요소 제거
let days = ['월', '화', '수'];
days.pop()
console.log(days) // ['월', '화']


shift, unshift : 배열 앞에 제거/추가
추가
days.unshift('일');
console.log(days) // ['일','월', '화','수']
```

```
제거
days.shift();
console.log(days) // ['월', '화','수']

days.unshift('금','토','일');
console.log(days) // ['금','토','일','월', '화','수']
shift와 unshift모두 여러 요소를 한번에 추가 제거할 수 있따.
```

```
반복문 : for
let days = ['금','토','일'];

for(let index = 0; index < days.length; index++){
  console.log(days[index]) // 0~2까지 반복
}


반복문 : for ... of
let days = ['금','토','일'];

for(let day of days){
  console.log(day);
}

객체를 순회하는 for in과 헷갈리지 않게 주의
물론 배열도 객체형이기 때문에 for in 을 쓸 수 있지만
장점보다 단점이 많기 때문에 권장하지 않는다.

배열 days를 돌면서 요소를 day라는 이름으로 접근할 수 있다.
for문 보다는 간단하지만 index를 못얻는다는 단점이 있다.
```

```
// array

let days = ['mon' , 'tue' , 'wed'];
console.log(days);

// [object Array] (3)
["mon","tue","wed"]

-------------------------------------------

let days = ['mon' , 'tue' , 'wed'];
```

```
days[1] = '화요일';
console.log(days);
// [object Array] (3)
["mon","화요일","wed"]

-----------------------------------------

let days = ['mon' , 'tue' , 'wed'];
days[1] = '화요일';
console.log(days.length);

// 3

-----------------------------------------

let days = ['mon' , 'tue' , 'wed'];
days.push('thu');
days.unshift('sun');
console.log(days);

// [object Array] (5)
["sun","mon","tue","wed","thu"]

-----------------------------------------

let days = ['mon' , 'tue' , 'wed'];
days.push('thu');
days.unshift('sun');

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

"sun"
"mon"
"tue"
"wed"
"thu"

-----------------------------------------------

let days = ['mon' , 'tue' , 'wed'];
days.push('thu');
days.unshift('sun');

for(let day of days){
  console.log(day);
}

"sun"
"mon"
"tue"
"wed"
"thu"
```

```
// day 는 아무 이름으로 사용해도 상관없다.

let days = ['mon' , 'tue' , 'wed'];
days.push('thu');
days.unshift('sun');

for(let x of days){
  console.log(x);
}

"sun"
"mon"
"tue"
"wed"
"thu"
```