

Ionic Study

Day 5

오늘 할 것들

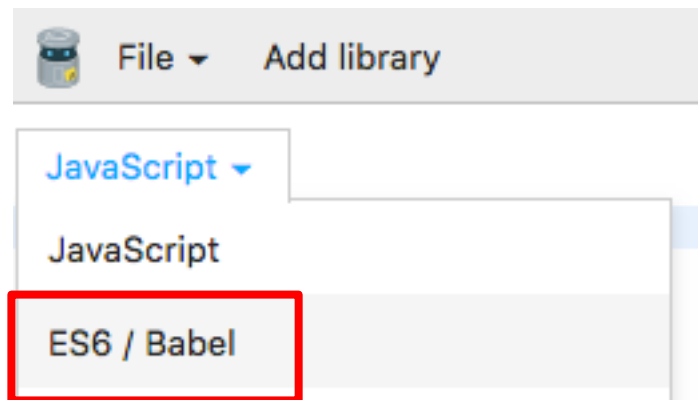
- 자바스크립트 기본 문법 알아보기
 - 함수의 스코프 및 호이스팅
 - 객체 다뤄보기

go to jsbin

- jsbin
 - 스코프와 호이스팅 개념을 설명하는 예제 코드들은 nodeJS에서 잘 동작하지 않습니다.
 - 이번 시간에는 jsbin이라는 연습용 사이트를 이용하여 자바스크립트를 작성하고 테스트함으로써 스코프와 호이스팅 개념을 이해해보도록 합니다.

go to jsbin

- jsbin
 - <http://jsbin.com/>
 - 1. jsbin 접속 후, 상단 좌측의 JavaScript를 클릭하여 ES6 / Babel로 변경.



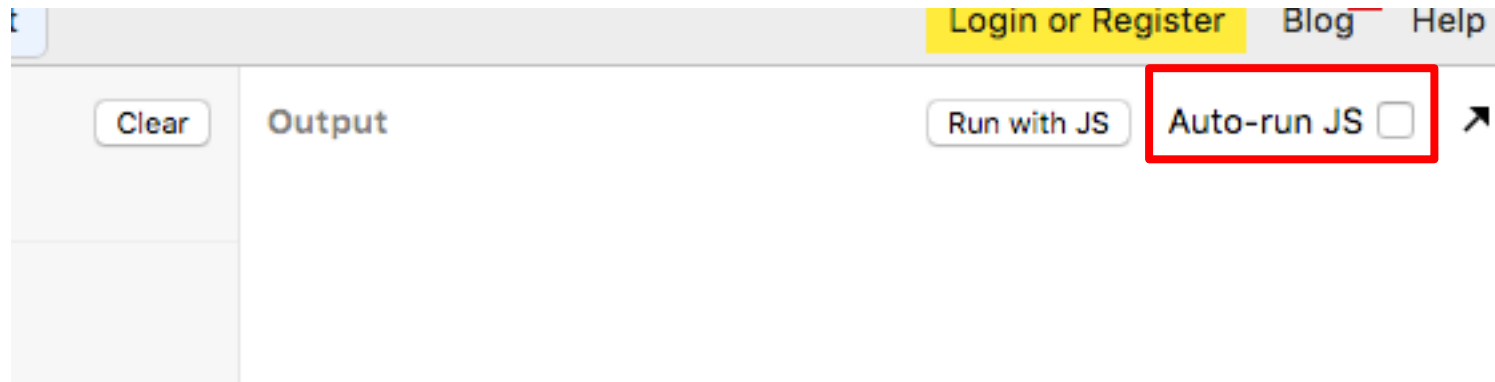
go to jsbin

- jsbin
 - 2. 상단의 Console와 Output 버튼을 클릭하여 콘솔 입력과 출력 창 활성화.



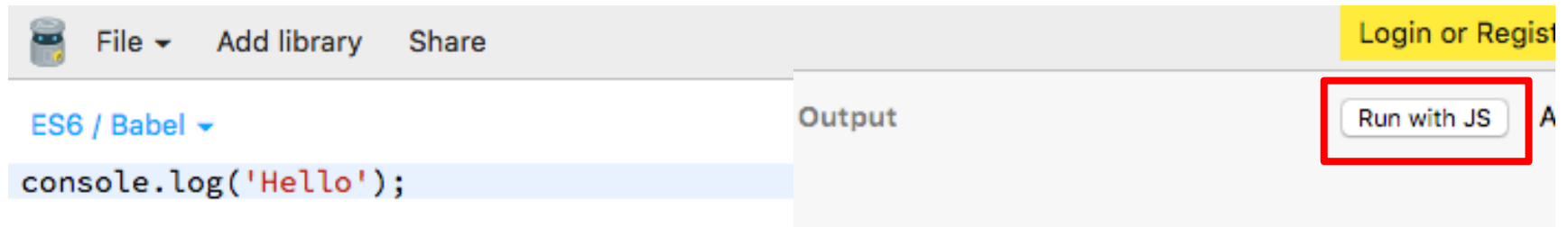
go to jsbin

- jsbin
 - 3. 우측의 'Auto-run JS' 체크 해제



go to jsbin

- jsbin
 - 스크립트 작성은 좌측에서...
 - 다 작성하고 실행해보려면 Output 패널 상단의 [Run with JS] 버튼 클릭.



함수

- 스코프와 호이스팅
 - 스코프 (Scope)
 - 범위.
 - 변수 또는 함수가 선언된 범위.
 - 어디에서 변수에 접근할 수 있는지, 그 컨텍스트에서 변수에 접근할 수 있는지를 명시적으로 나타냄.

스코프

- 지역변수
 - 함수 내부에 선언된 변수.
 - 해당 함수와 내부 함수에서만 접근이 가능.
 - 함수 내에서 전역변수랑 같은 이름의 지역변수를 선언한 경우, 지역변수가 우선함.
 - 사용하기 전에 반드시 선언해야 함

스코프

- 전역변수
 - 자바스크립트에서 제일 바깥 범위에 변수를 선언
 - 함수의 외부에서 선언된 모든 변수는 전역 범위를 가지게 됨
 - 모든 전역변수는 특정 전역 객체와 연결
 - 웹 : window 객체, nodeJS : global 객체
 - 특히 웹에서는 var 키워드 생략 가능.
 - 어디서든지 변수를 불러올 수 있음.
 - 변수가 섞이는 위험이 발생할 수 있다. (특히 라이브러리등) – 이전에 있던 변수를 덮어쓰는 사고 발생.
 - 가급적 전역변수는 생성을 지양해야 함.

스코프 - 가장 많이 하는 오해 1

```
var name = "Carry";    // 1
```

```
function sayHello(name) {  
    var name = "Mike"; // 2  
    console.log(name); // 3  
}  
console.log(name);      // 4
```

- 1은 전역 변수
- 2는 지역 변수
- 3의 결과는?
- 4의 결과는?

```
var name = "Carry";    // 1
```

```
if (name) {  
    name = "Rick";      // 2  
    console.log(name); // 3  
}  
console.log(name);      // 4
```

- 2를 건드리면 1에 영향을 끼칠까?
- 3의 결과는?
- 4의 결과는?

스코프 - 가장 많이 하는 오해 2

```
var ex1 = 'hi';
```

```
function sayHi() {  
    console.log(ex1);  
}
```

```
function sayHello() {  
    ex1 = 'Hello';  
    console.log(ex1);  
}
```

```
sayHi();  
sayHello();  
sayHi();
```

- ex1은 지역변수? 전역변수?
- 어떤 일이 일어날까요?

```
var ex2 = 'bam';
```

```
function showEx2() {  
    var ex2 = 'sam';  
    console.log(ex2);  
}
```

```
showEx2();
```

- 어떤 일이 일어날까요?

스코프

- 전역변수 - window객체를 통해 전역변수에 접근하기 (web)

	JS ex1.js	JS ex2.js	JS ex3.js
1	var globalVar1 = '전역 변수';		
2	console.log(globalVar1);		
3	console.log(window.globalVar1);		
4			
5	window.globalVar2 = '나는 천재다';		
6	console.log(globalVar2);		

스코프

- 전역변수 - var 키워드를 생략하면 자동으로 전역변수가 됨.

	JS ex2.js	JS ex3.js	JS ex4.js	JS ex5
1	function showAge() {			
2	age = 80;			
3	console.log(age);			
4	}			
5	console.log('showAge before : '+ age);			
6	showAge();			
7	console.log('showAge after : '+ age);			

함수

- 스코프와 호이스팅

- 호이스팅(Hoisting)

- 변수의 정의가 그 범위에 따라 선언과 할당으로 분리되는 것.
 - 스코프 내에서 선언된 변수는 항상 최상위에 선언한 것과 동등한 의미를 지님.
 - 변수의 선언이 초기화나 할당시에 발생하는것이 아니라, 최상위로 호이스트 됨.
 - “모든 변수선언은 호이스트 된다”

호이스팅

- 호이스팅은 '호이스트'에서 비롯.
 - 호이스트 : 소형의 화물을 들어올리는 장치 (감아서 끌어올림)
- '변수를 끌어올리다'



호이스팅



- 이런 호이 말고...

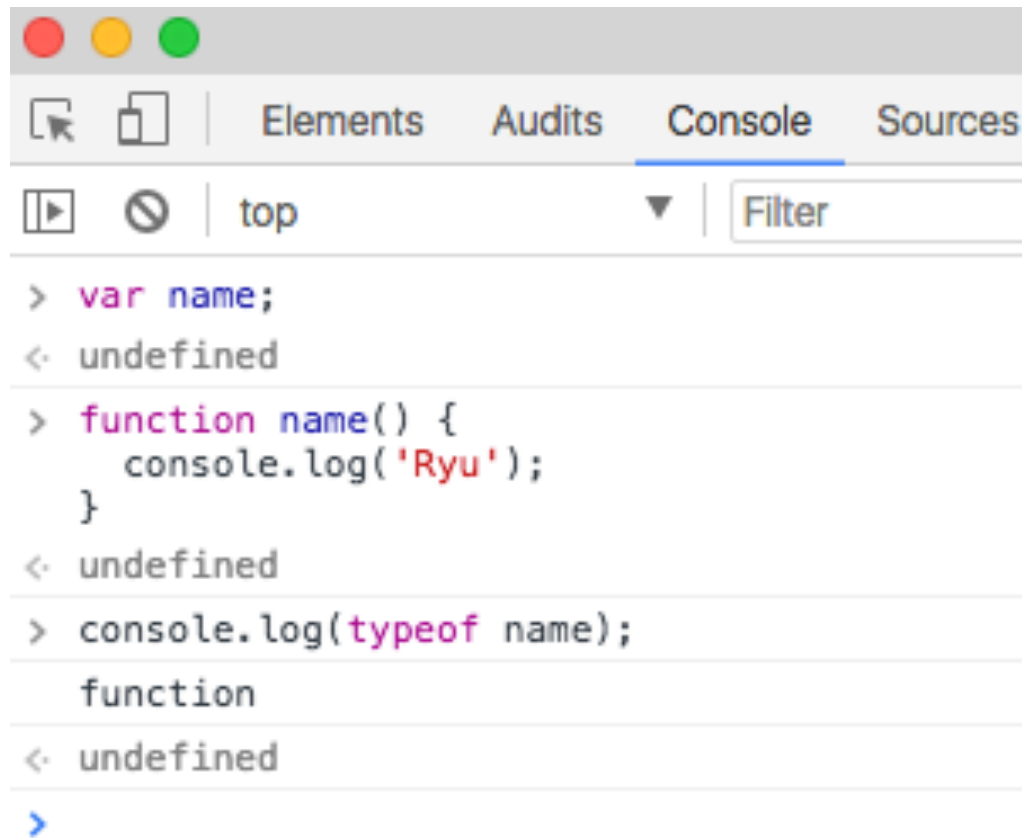
호이스팅

- 실행 결과는?

```
JS ex3.js JS ex4.js JS ex5.js JS ex6.js JS ex7.js ●
1 function sayHello() {
2     console.log('message: ' + message);
3     var message = 'Hello';
4     console.log('message: ' + message);
5 }
6 sayHello();
```

호이스팅

- 호이스팅되면 함수 선언은 변수 선언을 덮어 쓰게 됨.



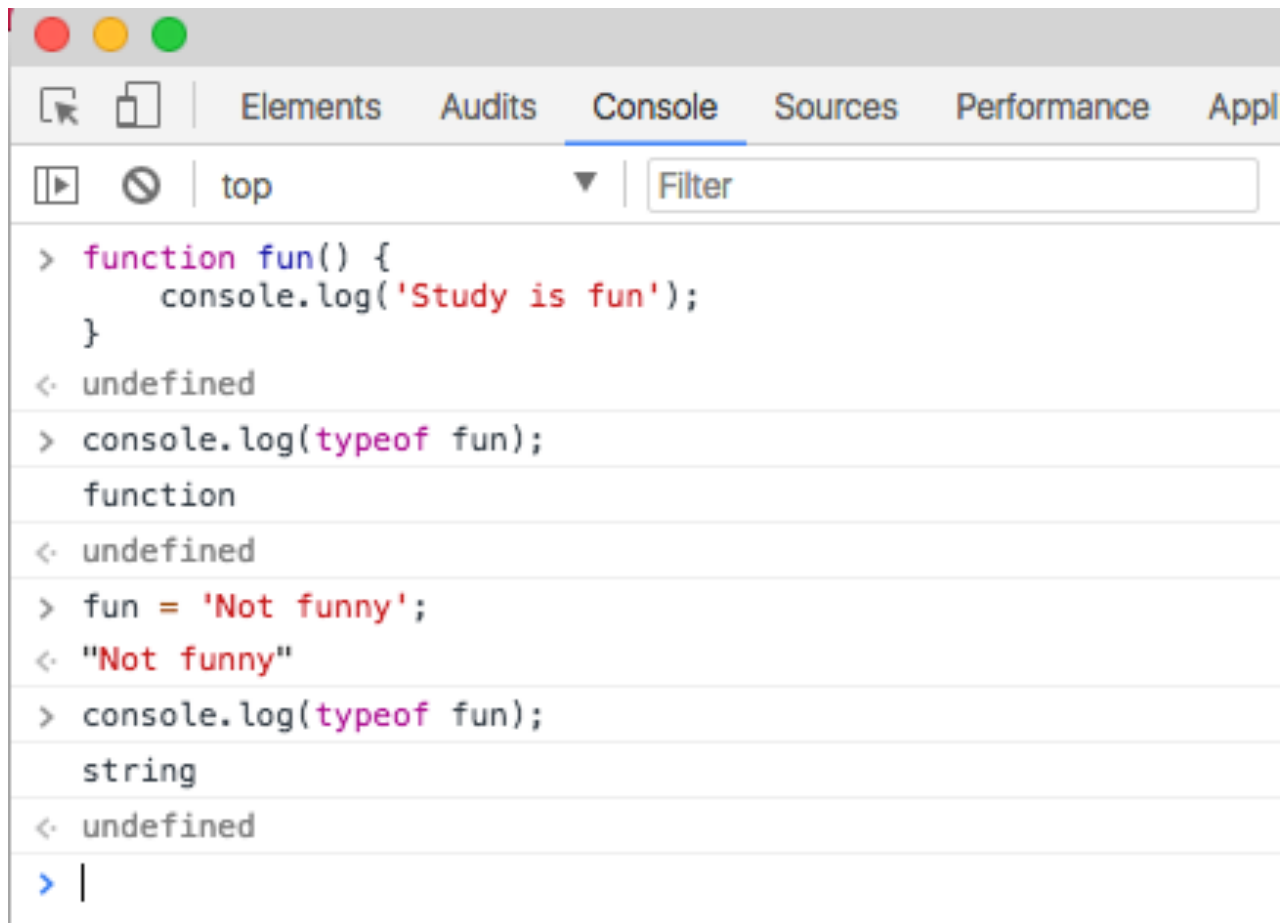
The screenshot shows a web browser's developer console with the 'Console' tab selected. The console displays the following sequence of commands and results:

```
> var name;  
< undefined  
  
> function name() {  
    console.log('Ryu');  
}  
< undefined  
  
> console.log(typeof name);  
function  
< undefined  
  
>
```

The output demonstrates that the function declaration is hoisted to the top of the scope, overriding the variable declaration. As a result, `typeof name` returns `function` instead of `undefined`.

호이스팅

- 그 반대의 경우도 가능.



The screenshot shows a web browser's developer console with the 'Console' tab selected. The console displays the following sequence of commands and outputs:

```
> function fun() {  
    console.log('Study is fun');  
}  
< undefined  
> console.log(typeof fun);  
function  
< undefined  
> fun = 'Not funny';  
< "Not funny"  
> console.log(typeof fun);  
string  
< undefined  
> |
```

The output demonstrates that the function `fun` is hoisted to the top of the scope, allowing it to be used before its definition. Similarly, the variable `fun` is hoisted from its assignment at the bottom to the top, overriding the function definition.

객체지향 프로그래밍

- 객체 - 현실의 사물을 프로그래밍에 반영
- 이러한 객체 위주로 개발을 이어나가는 것을 객체지향 프로그래밍이라고 함...

클래스

- 자바스크립트는 클래스라는 개념이 없음.
- 끝.
- 일 줄 알았죠?

클래스

- 자바스크립트는 클래스라는 개념이 없음.
- 자바스크립트에서 다루는 모든 것은 객체.
- function을 클래스로 사용
 - 함수는 1급 객체!
 - 생성자 함수라고 한다.
- 다른 언어와 다르게, private, public에 대한 개념이 없다.
 - 유사 구현 필요 (어렵다 -_-)

객체

- 객체(Object) = 클래스 인스턴스(Instance)
- 속성과 메소드로 구성
 - 속성(Property) : 객체의 전용 변수
 - 해당 객체의 특성을 기술할 때 사용.
 - 메소드(Method) : 객체의 전용 함수
 - 해당 객체의 행동을 기술할 때 사용.

객체

- 클래스 정의하기
 - 일반 함수를 이용하여 정의하는 방법
 - 빈 속성을 이용하여 정의하는 방법
 - 클래스명은 구분을 위해 첫글자를 대문자로 정의하는 것이 관례
 - ex) Apple, Person ...

객체

- 일반 함수를 이용하여 정의
 - 평범한 함수를 만들어 놓고, new 연산자를 이용하여 객체를 생성

객체

- ex8.js

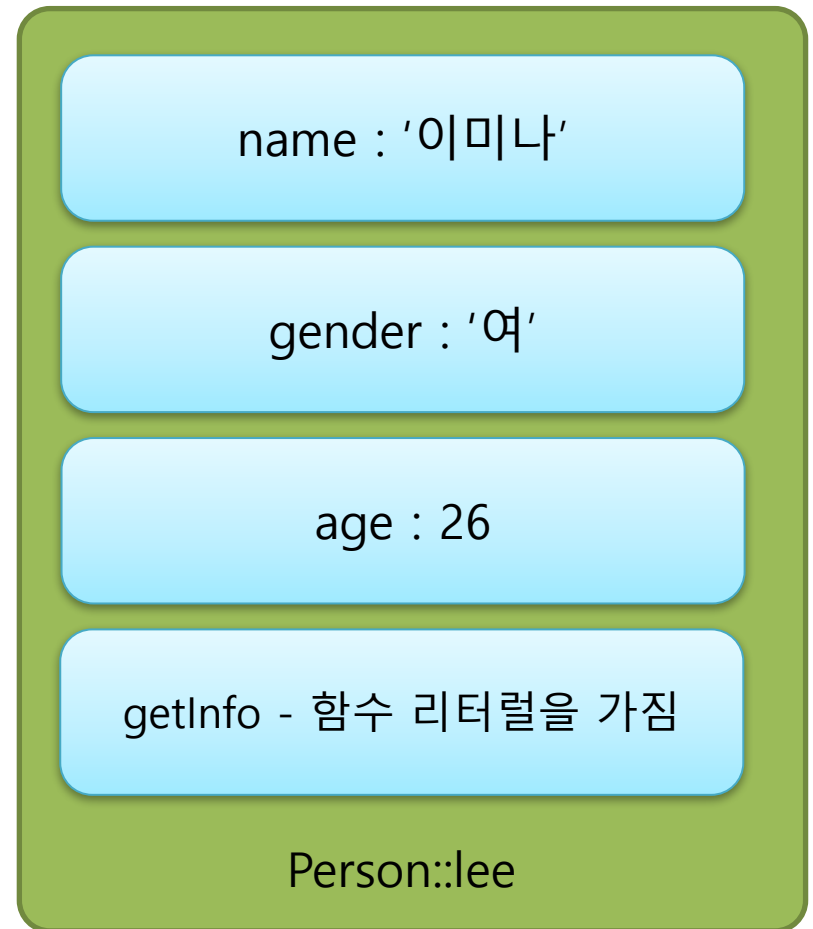
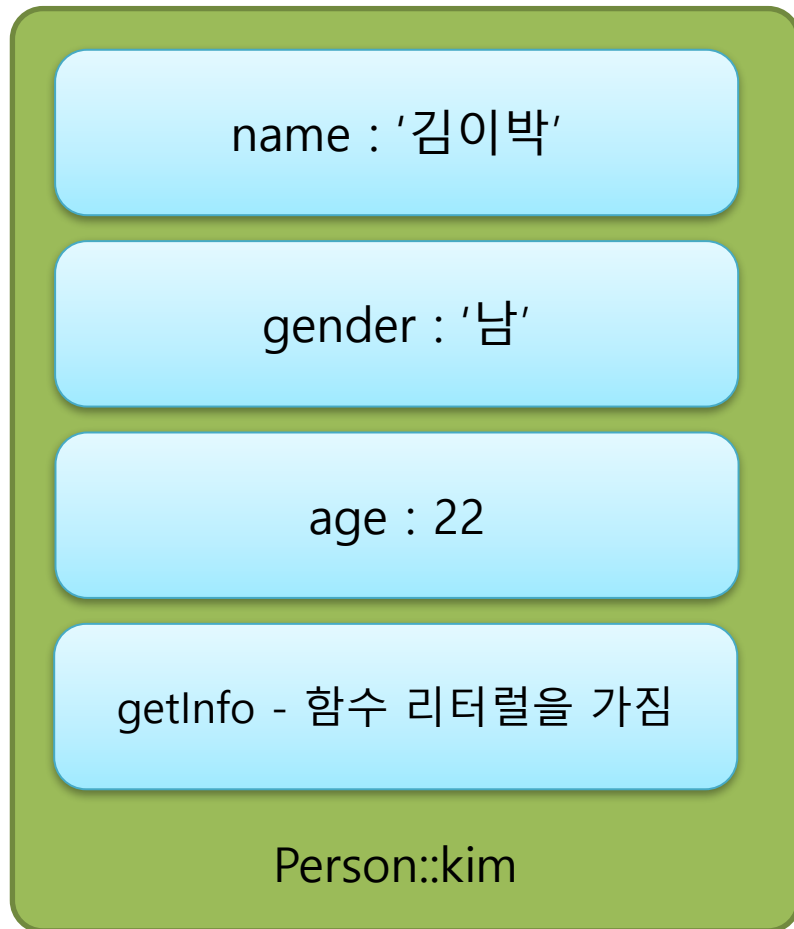
```
JS ex5.js JS ex6.js JS ex7.js JS ex8.js x JS
1 function Person(name, gender, age) {
2   this.name = name;
3   this.gender = gender;
4   this.age = age;
5   this.getInfo = getPersonInfo;
6 };
7
8 function getPersonInfo() {
9   return '이름 : ' + this.name +
10     ', 성별 : ' + this.gender +
11     ', 나이 : ' + this.age;
12 }
13
14 let kim, lee;
15 kim = new Person('김이박', '남', 29);
16 kim.age = 22;
17 lee = new Person('이미나', '여', 26);
18
19 console.log(kim.getInfo());
20 console.log(lee.getInfo());
```

객체

- new 연산자, this 키워드
 - new : 새로운 개체를 만드는 연산자.
 - 메모리에 할당...
 - this : 생성된 객체의 멤버변수를 참조

객체

- new 연산자를 사용하면 독립된 속성을 가지는 객체가 생성됨. (kim, lee)



객체

- 객체의 속성에 접근 : 연산자를 이용
 - 속성 가져오기
객체변수명.속성명
 - 속성 설정
객체변수명.속성명 = 속성값
 - 모든 속성은 개방되어있다.
 - 객체만 참조 가능하다면 어디서든지 해당 객체의 속성에 접근 가능.

객체

- 앞서 예제의 큰 단점
 - 메소드를 정의하는 함수가 전역으로 구현되어 있다.
 - 전역함수 및 변수의 큰 문제점 : 다른 라이브러리 사용 및 우연에 의하여 이름 충돌 가능성 있음.
 - 객체에 접근하는 메소드는 가급적 전역 구현을 피하는게 좋음
 - 객체지향의 특성 중, 캡슐화를 실현하지 못하기 때문에...

객체

- 해결 방안
 - 내부 함수(중첩 함수) 구현
 - prototype 구현

객체

- ex9.js - 메소드 getInfo를 내부함수로 변경

```
JS ex5.js JS ex6.js JS ex7.js ● JS ex8.js JS ex10.js x
1  function Person(name, gender, age) {
2      this.name = name;
3      this.gender = gender;
4      this.age = age;
5      this.getInfo = function() {
6          return '이름 : ' + this.name +
7              ', 성별 : ' + this.gender +
8              ', 나이 : ' + this.age;
9      }
10 };
11
12 let kim, lee;
13 kim = new Person('김이박', '남', 29);
14 kim.age = 22;
15 lee = new Person('이미나', '여', 26);
16
17 console.log(kim.getInfo());
18 console.log(lee.getInfo());
```

객체

- ex9의 장,단점
 - Good : 함수명 충돌 가능성 제거
 - 전역함수와 함수명이 겹치지 않는다 (클래스 메소드는 함수 스코프에 의해 전역함수보다 우선하게 됨)
 - Bad : 새로운 객체를 생성할 때마다 해당 메소드가 매번 재생성 됨.
 - 객체별로 독립된 메소드를 가짐.
 - . 연산자를 이용하여 메소드를 변경 시킬 가능성.
 - prototype 키워드를 사용하여 해결 가능.

객체

- ex10.js - Person의 프로토타입 추가

```
JS ex6.js JS ex7.js ● JS ex8.js JS ex10.js ✕ JS ex9.js
1  function Person(name, gender, age) {
2      this.name = name;
3      this.gender = gender;
4      this.age = age;
5  };
6
7  Person.prototype.getInfo = function() {
8      return '이름 : ' + this.name +
9          ', 성별 : ' + this.gender +
10         ', 나이 : ' + this.age;
11 };
12
13
14 let kim, lee;
15 kim = new Person('김이박', '남', 29);
16 kim.age = 22;
17 lee = new Person('이미나', '여', 26);
18
19 console.log(kim.getInfo());
20 console.log(lee.getInfo());
```

객체

- prototype 키워드
 - 함수의 속성
 - 생성자 함수에서 만든 객체의 속성.
 - 함수의 프로토타입 = 객체
 - 인스턴스의 공유가 가능.

객체

- 빈 속성을 이용하여 정의
 - 리터럴 : 자바스크립트에서 객체와 배열을 정의하는 간단한 방법

객체

- 속성의 표현 방법

- {}를 이용하여 속성을 나타낸다.
- 프로퍼티는 ,로 구분하며 정의와 할당은 :로 구분

```
var prop = {  
    name: '홍길동',  
    age: 27  
}
```

객체

- `new Object, {}`
 - 함수를 이용한 구현 이외에도, `new Object()` 또는 객체 리터럴을 이용하여 빈 객체를 생성 가능.

객체

- ex11.js

```
JS ex6.js JS ex7.js ● JS ex8.js JS ex10.js JS ex11.js x
1  const obj = {};
2  const alex = {
3      name: '알렉스',
4      gender: '남',
5      age : 15,
6      getInfo: function() {
7          return '이름 : ' + this.name +
8              ', 성별 : ' + this.gender +
9              ', 나이 : ' + this.age;
10     }
11 };
12
13 console.dir(obj);
14 obj.prop = 'value';
15 console.dir(obj);
16 console.log(obj.prop);
17
18 console.dir(alex);
19 console.log(alex.getInfo());
```


객체

- 객체 리터럴을 사용하는 경우 특징
 - 클래스 생성자 불필요
 - 객체 리터럴을 사용하여 객체를 정의하는 경우에는 해당 클래스의 인스턴스는 이미 생성되어 있음. new 연산자를 쓸 필요가 없음.
- 객체 리터럴의 단점
 - 인스턴스 여러개 생성 불가. (객체 리터럴을 사용하면 인스턴스가 자동으로 생성되므로)

객체

- private
 - 객체에서 바뀌면 안되는 속성값들이 존재
 - 이런 녀석들은 절대 외부에 공개해선 안됨
 - 캡슐화를 구현
 - C++, java, c#등... private 접근지정자 존재
 - 자바스크립트는? 읊따...

객체

- private
 - 자바스크립트에서 속성을 private로 구현하기 위해서는 클로저를 활용
 - 클로저 : 함수 안에 정의된 함수 변수.
 - 지역 변수의 성격을 띤다. 스코프는 함수 내로 한정
 - 해당 속성에 접근하기 위해 게터(getter), 세터(setter) 사용 필요.
 - 게터 : 속성값을 반환(Get an attribute)
 - 세터 : 속성값을 지정(Set an attribute)

객체

- ex12.js - 생성자 함수 내부에 게터 세터 구현

```
ex8.js  JS ex10.js  JS ex11.js  JS ex12.js x  JS ex13.js

function Person() {
  var name;
  var age;
  this.setName = function(newname) {
    name = newname;
  };
  this.setAge = function(newage) {
    age = newage;
  };
  this.getName = function() { return name; }
  this.getAge = function() { return age; }
}

let person = new Person();
person.setName('홍길동');
person.setAge(28);
person.name = '임꺽정';

var output = person.getName() + ':' + person.getAge();
console.log(output);
```

객체

- ex13.js - prototype를 이용한 게터 세터 구현

```
ex10.js  JS ex11.js  JS ex12.js  JS ex14.js  JS ex13.js x
function Person() {
    var name;
    var age;
};
Person.prototype.setName = function(newname) {
    name = newname;
};
Person.prototype.setAge = function(newage) {
    age = newage;
};
Person.prototype.getName = function() { return name; }
Person.prototype.getAge = function() { return age; }

let person = new Person();
person.setName('홍길동');
person.setAge(28);

var output = person.getName() + ':' + person.getAge();
console.log(output);
```

객체

- 퀴즈 - 아래 코드에서 1과 2는 같을까요?



```
ex10.js  JS ex11.js  1  JS ex14.js  JS ex13.js x
function Person() {
  var name;
  var age;
};
Person.prototype.setName = function(newname) {
  name = newname;
};
Person.prototype.setAge = function(newage) {
  age = newage;
};
Person.prototype.getName = function() { return name; }
Person.prototype.getAge = function() { return age; }

let person = new Person();
person.setName('홍길동');
person.setAge(28);

var output = person.getName() + ':' + person.getAge();
console.log(output);
```

객체

- 주의

- 생성자 함수 내부에서 var로 선언한 변수와 prototype에 프로퍼티로 생성한 함수 내에 존재하는 변수는 서로 다름.
- 선언한 공간이 다르기 때문 (클로저)
- 이러한 특징으로 인해 속성을 공유해야 하는 상황에서 내부 함수와 프로토타입 함수 구현의 두가지 기법을 혼용하는것은 매우 부적절한 선택이 됨.
- 서로 독립된 (다른 영역에서 접근할 필요가 없는) 상황이라면 무관하긴 하지만... 애간하면 하나로 통일하는게 좋음.

객체

- 객체 리터럴에서 private
 - 익명 함수로 감싼 다음, 즉시 실행할 수 있게 구현.
 - 이후는 생성자의 경우와 동일하게 클로저를 활용.
 - 객체 리터럴을 익명함수로 감쌌기 때문에, 공개가 필요한 속성 또는 메서드를 속성으로 만들어서 반환.

객체

- ex14.js - 객체 리터럴에서의 private

```
JS ex8.js JS ex10.js JS ex11.js JS ex12.js JS ex14.js x
1  var alvis = (function() {
2      var name;
3
4      return {
5          getName: function() {
6              return name;
7          },
8          setName: function(newname) {
9              name = newname;
10         }
11     }
12 }());
13
14 alvis.setName('Elly');
15 console.log(alvis.getName());
```

오늘은 여기까지~

See you next day!