함수

함수

- **일급 함수(first class function(object))**
  - 자바스크립트에서는 함수도 객체다!
  - 이것은 함수를 값처럼 쓸 수 있다는 것, 즉
    - 함수를 변수에 할당할 수 있음
    - 함수를 함수의 인자로 사용할 수 있음
    - 함수를 반환값으로 사용할 수 있음

함수

- **함수를 변수에 할당**
  - 자바스크립트를 매우 유연한 언어로 만듬

```
var f = getGreeting;
f();    //”Hello, World!”
```

  - 함수를 객체 프로퍼티에 할당할 수도 있음

```
var o = {};
o.f = getGreeting;
o.f();          //”Hello, world!”
```

  - 배열 요소로 할당할 수도 있음

```
var arr = [1,2,3];
arr[1] = getGreeting
arr[1]();
```

```
function getGreeting(){
    return "Hello world!";
}
//함수를 저장한 변수에 "()"를 붙이면 함수 호출을 의미
console.log(getGreeting());
console.log(getGreeting);
```

함수 정의

함수

- **함수 정의 방법**
  - 함수 선언식

```
function 함수명() {
  함수 로직
}
```

  - 함수 표현식

```
var 함수명 = function () {
  함수 로직
};
```

  - 익명함수(즉시 실행이 필요한 경우, 주로 콜백함수로 사용)

```
(function () {
   함수 로직
})();
```

함수

- **익명 함수(anonymous function)**
  - 이름이 없는 함수

```
function x(y) {
   return y * y;
};
x(3)
```

[일반적인 형태]

```
var x = function(y) {
   return y * y;
};
x(3)
```

[익명 함수]

```
var g=function f(y){
   return y * y;
}

f(3)// 에러
```

외부에서 호출할 땐
g에 우선순위가 있음

함수

- **즉시 실행 함수 표현(Immediately Invoked Function Expression, IIFE)**
  - 선언과 동시에 바로 실행하는 함수
  - 함수를 괄호()로 묶음

```
( function (y) {
    return y * y;
} )(3);
```

[익명 즉시 실행 함수]

```
( function x(y) {
    return y * y;
} )(3);
```

[기명(이름이 있는) 즉시 실행 함수]

```
( fVal = function x(y) {
    return y * y;
} )(3);
fVal(4)
```

[변수에 저장도 가능]

함수

- **클로저(closure)**
  - 생성될 당시의 환경을 기억하는 함수

    →스코프가 해제되어야 할 시점에도 사라지지 않는 특징
  - 함수 안에(외부 함수) 또 다른 함수(내부함수)를 정의
  - 내부 함수가 외부 함수의 스코프에 접근 할 수 있게 하는 것

```
function outFunc(name) {
    var outVar = 'my name is '
    function innerFunc() { return outVar + name }
    return innerFunc
}
var result = outFunc('bono')
console.log('result: ' + result())
```

// result: my name is bono

outFunc함수 실행이 종료된 시점이므로 name 과 지역변수인 outVar 는 메모리에서 정리되어야 하는데..

innerFunc함수가 선언될 때, name과 outVar를 기억하고 있음

https://blueshw.github.io/2017/04/12/javascript-closure/

함수

- **클로저(closure)**

```
var out = 'out value'              //전역스코프
function outFunc() {               //outFunc스코프
    var inner = 'in value'
    function inFunc(inParam) {  //inFunc 스코프
        console.log('out: ' + out)
        console.log('inner: ' + inner)
        console.log('inParam: ' + inParam)
    }
    return inFunc
}
var param = 'this is param'
var outResult = outFunc()
outResult(param)
// out: out value
// inner: in value
// inParam: this is param
```

inFunc은 선언되었지만
언제 호출될 지 모르니
안전빵(?)으로 변수들을 저장하고 있음

함수가 선언되는 순간에, 함수가 실행될 때 실제 외부변수에 접근하기 위한 객체
가비지컬렉션 되어야하는 변수가 메모리상에 강제로(?) 남아있게 되므로 오버플로우의 위험이 존재

함수

- **클로저(closure)**
  - 은닉화: 외부 함수의 매개변수를 마치 private 변수처럼 관리 할 수 있음

```
function makeAdder(x) {
    return function(y) {
    return x + y;
    };
}
var add5 = makeAdder(5);
var add10 = makeAdder(10);

console.log(add5(2));  // 7
console.log(add10(2)); // 12
```

- add5의 변수(매개변수) x는 오직 add5라는 개체에 의해서만 초기화
- add10의 변수 (매개변수) x는 오직 add5라는 개체에 의해서만 초기화

함수

- **클로저(closure)**
  - 클로저를 이용해서 private 메소드 흉내내기

```
var makeCounter = function() {
    var privateCounter = 0;
    function changeBy(val) {
        privateCounter += val;
    }
    return {
        increment: function() {
        changeBy(1);
    },
    decrement: function() {
        changeBy(-1);
    },
    value: function() {
        return privateCounter;
    }
  };
};
```

실행 후 makeCounter에 남아 있는 요소들

함수 실행 후 반환되는 요소들
즉, 객체에 담긴 세 개의 함수

```
var counter1 = makeCounter();
var counter2 = makeCounter();
```

두 개의 카운터가 독립성을 유지

privateCounter와 changeBy는
익명함수 외부에서 접근될 수 없다.
대신 세 개의 퍼블릭 함수를
통해서만 접근 가능

함수

- **클로저(closure)**
  - 클로저를 이용해서 private 메소드 흉내내기

```
var counter = (function() {
   var privateCounter = 0;
   function changeBy(val) {
      privateCounter += val;
   }
   return {
      increment: function() {
      changeBy(1);
   },
   decrement: function() {
      changeBy(-1);
   },
   value: function() {
      return privateCounter;
   }
  };
})();
```

```
console.log(counter.value()); // logs 0

counter.increment();

counter.increment();

console.log(counter.value()); // logs 2

counter.decrement();

console.log(counter.value()); // logs 1
```

함수

- **호이스팅(Hoisting)**
  - 자바스크립트는 함수나 전역 스코프 전체를 살펴보고(코드를 다 뒤져보고) var로 선언한 변수(let은 해당 안됨)를 맨 위(유효범위의 최상단)로 끌어 올림
  - 전역 범위(global scope)
    - 전역 범위에서는 스크립트 단위에서 최상단으로 끌어 올려짐
  - 함수 범위(function scope)
    - 함수 범위에서는 해당 함수의 최상단으로 끌어 올려짐


```
x;          //undefined
var x=3;
x;          //3
```

var를 사용하면 선언하기 전에도 사용할 수 있다

```
var x;  //선언(할당은 아닌)이 끌어올려짐
x;
x=3;
x;
```

할당은 끌어 올려지지 않음

함수

- **호이스팅(Hoisting)**
  - 유효 범위의 최상단?

```
function ho1(){
    if(true){
        var name = 'yuddomack';
    }
    console.log(name);
}
function ho2(){
    for(var i=0; i<5; i++){
    // do something
    }
    console.log(i);
}
if(true){
    var score = 100;
}
console.log(score);
```

```
var score; // 선언
function ho1(){
    var name; // 선언
    if(true){
        name = 'yuddomack'; // 할당
    }
    console.log(name);
}
function ho2(){
    var i; // 선언
    for(i=0; i<5; i++){ // 할당
    // do something
    }
    console.log(i);
}
if(true){
    score = 100; // 할당
}
console.log(score);
```

함수

- **호이스팅(Hoisting)**
  - 함수도 호이스팅이 일어남

```
f()
function f(){
    console.log('f');
}
```

'f'가 출력

함수 호이스팅 예

```
sayName();
var sayName = function(){
        console.log('yuddomack');
}
```

```
var sayName;
sayName();
sayName = function(){
        console.log('yuddomack');
}
```

이것은 익명함수를 sayName에 할당

  - 호이스팅이 일어난다는 뜻이지, 선언되지 않은 변수를 사용하는 것을 추천하는 것은 아님
  - 이러한 문제를 차단하기 위해 let이 소개

함수

- 고계함수(higher-order function)
  - 인자나 반환값으로 함수를 사용하는 함수
  - 함수형 프로그래밍에서 나오는 개념(코틀린, 스칼라..)

```
function twice(f,x){

	return f(f(x));

}
function f(x){

	return x*3;

}
twice(f,7);
```