함수

함수 : 한 번 정의하면 몇 번이든 실행할 수 있고 호출 할 수 있는 자바스크립트 코드블록
매개변수 : 함수 몸체 내에서 지역변수처럼 취급
메서드 : 어떤 객체의 속성으로 저장된 자바스크립트 함수
생성자 : 새로 생성된 객체를 초기화 하는데 쓰이는 함수
자바스크립트 함수는 다른 함수 내에 중첩되어 정의될 수 있고 , 함수는 해당 함수가 정의되어있는 유효범위에 속한 어떤 변수에도 접근 가능하다 (클로저)

1. 함수 정의하기

- function
- 함수이름 식별자(함수이름)
- 쉼표로 구분된 0개혹은 임의 개수의 식별자들과 이 식별자들을 둘러싼 한 쌍의 괄호
- 0개 혹은 임의 개수의 자바스크립트 구문을 포함하는 한쌍의 중괄호

```
function 함수이름(전달인자1, 전달인자2, ... ,전달인자n) {
        구문
}
```

* 함수 이름을 짧게 쓰기보다는 함수의 의미 및 기능을 잘 설명할 수 있는 이름으로 선택해야 함

1.1 중첩 함수

자바스크립트에서 함수는 다른 함수와 중첩될 수 있다
변수 범위규칙이 중첩된 함수는 해당 함수가 속한 함수의 매개변수와 변수에 접근할 수 있다

```
Function(a,b){
        function square(x) { return x*x; }
        return Math.sqrt(square(a) + square(b));
}
```

2. 함수 호출하기

함수를 정의했어도 함수 몸체의 자바스크립트 코드는 함수를 호출하지 않으면 실행되지 않는다
자바스크립트 함수 호출 법

- 일반적인 함수 형태
- 메서드 형태
- 생성자
- 해당 함수의 call()과 apply() 메서드를 통한 간접적 방식

2.1 함수 호출

순서

- 각 전달인자 표현식(괄호 사이의 값들)이 평가
- 평가 결과가 전달인자가 된다
- 전달인자 값들은 함수정의에 지정한 매개변수와 대응된다
- 함수 몸체에서 매개변수에 대한 참조는 해당 전달인자 값을 표현한다

반환값

- return 문의 값이 있으면 그 값을 반환 아니면 undefined을 반환한다

2.2 메서드 호출

함수 : f, 객체 : o, 메서드 : m 라고 한다면 **o.m = f;** 로 정의 할 수 있다
객체 o에 메서드 m()을 정의한 다음에 **o.m();** 로 호출 할 수 있다
(m이 2개의 인자를 받는다면 o.m(x,y); 로 호출하면 된다 (여러 개를 인자로 받을 수 있다))
메서드 호출과 함수호출의 다른 점은 호출 컨텍스트다

- 객체와 속성이름으로 구성된다
- . 뿐만 나이라 [ ]로도 메서드 호출을 할 수 있다

```
var calculator = {
        operand1: 1,
        operand2: 1,
        add: function() {
                //자기의 객체를 참조하기 위해 this 키워드 사용
                this.result = this.operand1 + this.operand2;
        }
};
calculator.add(); //add 메서드 호출
caculator.result //2
```

2.3 생성자 호출

함수나 메서드 호출 앞에 new 키워드가 있으면 생성자 호출이다
한 쌍의 빈 괄호()는 생략 가능하다
새로 생성된 객체를 this 키워드로 참조할 수 있다
var o = new Object(); // 생성자 호출 () 빈 괄호는 생략 가능

2.4 간접 호출

함수의 메서드 중에 call(), apply() 가 간접 호출을 한다
호출 시에 this 값을 명시적으로 지정할 수 있다

- 어떤 함수든지 특정 객체의 메서드로 호출할 수 있다

call() 메서드는 자신에게 주어진 전달인자를 호출 할 함수의 전달인자로 사용한다
apply() 메서드는 값 배열을 전달인자로 사용한다

3. 함수 전달인자와 매개변수

함수 매개변수에 대한 형식을 명시하지 않는다
함수를 호출할 때도 전달된 인자 값에 대한 형식 검사를 수행하지 않는다

3.1 생략 가능한 매개변수

생략 가능한 전달인자를 사용하여 함수를 설계할 때 생략할 수 있는 인자는 전달인자 목록의 제일 뒤쪽에 두어야 한다
함수를 정의 할 때 생략할 수 있음을 /*optional*/ 주석을 통해 강조해준다

```
function getPropertyNames(o, /*optional*/ a) {
if ( a=== undefined ) a = []; //a에 대한 전달인자가 없을 때 새로운 배열 생성
for(var property in o) a.push(property);
return a;
}
var a = getPropertyNames(o); //전달인자 한 개 호출 새 배열에 o의 속성을 넣음
getPropertyNames(p,a); //전달인자 두 개 호출 p의 속성 a에 추가
```

3.2 가변길이 전달인자 목록 : Arguments 객체

함수몸체 내에서 arguments 식별자는 해당 호출에 대한 Arguments 객체를 참조한다
Arguments 객체는 유사 배열 객체이다
이름이 아닌 인덱스 숫자를 통해 함수의 전달인자를 얻어올 수 있다
arguments 라는 프로퍼티는 함수 내에서 전달인자를 참고하는 것과 같음
callee caller 속성
- caller 속성을 통해 호출 스택에 접근 할 수 있다
- callee 속성은 이름없는 함수를 재귀적으로 호출하는 데 유용하다

```
function add(){
   var sum = 0;
   for (var i = 0; i < arguments.length; i++) {
      sum += arguments[i];
   }
   return sum;
}
function test(){
   var sum1 = add(1, 2, 3);
   var sum2 = add(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
}
```

3.3 객체의 속성을 전달인자로 사용하기

```
//전달인자를 순서에 상관없이 이름/값의 쌍으로 함수에 전달하기
function easyCopy(args) {
        testcopy(args.from,
                args.from_start || 0,      //기본 값 0을 설정
                args.to_start || 0,
                args.length);
}
//easycopy() 사용법
var a = [1,2,3,4] , b = [];
easyCopy({from: a, to: b, length: 4});
```

3.4 전달인자 형식

메서드의 매개변수에는 정의된 형식도 없고, 함수에 전달한 값에 대한 검사도 하지 않는다
- 주석을 이용해 인자형식을 작성해 주는 것이 좋다

4. 값으로서의 함수

자바스크립트 함수는 문법 뿐만 아니라 값이기도 하다
- 변수에 할당, 객체의 프로퍼티나 배열 요소로 저장, 다른 함수의 인자로 전달 등이 가능하다

```
//다른 함수의 인자로 전달하는 예 ( 사칙연산 )
// 사칙연산 함수 정의
function add(x,y) { return x + y; }
function subtract(x,y) { return x - y; }
function multiply(x,y) { return x * y; }
function divide(x,y) { return x / y; }
// 정의한 함수 중 하나를 인자로 받아 두 개의 피연산자와 같이 호출
function operate(operator, operand1, operand2) {
    return operator(operand1, operand2);
}
// 함수이용 (1 + 2) * (4 - 3)
var i = operate(multiply, operate(add, 1, 2), operate(subtract, 4, 3));
```

4.1 자신만의 함수 프로퍼티 정의하기

정적 변수가 필요할 때는 함수의 프로퍼티를 사용하는 것이 좋다

```
//함수 객체의 카운터 프로퍼티를 초기화한다.
//uniqueInteger 함수는 hoisting 되기 때문에 먼저할당 가능
uniqueInteger.counter = 0;
//다음반환값을 기억하기위해 자신의 프로퍼티사용
function uniqueInteger() {
        return uniqueInteger.counter++;
}
```

5. 네임스페이스로서의 함수

자바스크립트는 단일 코드 블록 내에서만 유효한 변수를 정의하는 방법을 제공하지 않는다
- 간단한 임시 네임스페이스처럼 작동하는 함수를 정의하는 기법을 이용

하나의 전역변수만 정의

function myModule{
//모듈 코드
//모듈에서 사용하는 어떤 변수이건 이 함수의 지역변수다
//전역 네임스페이스를 어지럽히지 않는다
}
myModule(); //함수호출을 꼭 해주어야 한다!!

단일 표현식으로 익명 함수를 정의

(function() { //이름이 없는 표현식으로 위의 함수 재작성
//모듈 코드
}()); //함수 리터럴을 끝내고 바로 호출함

6. 클로저

따로 정리해 두었음

7. 함수 프로퍼티, 메서드, 생성자

7.1 length 프로퍼티

몸체 내에서의 arguments.length는 실제로 전달된 인자의 개수

함수 자체의 length 프로퍼티는 함수를 정의할 때 명시한 인자 개수를 반환

```
function check(args) {
    var actual = args.length; //인자의 실제 개수
    var expected = args.callee.length; //인자의 요구 개수
    if (actual !== expected) //두 값이 다르면 예외 발생
    throw Error("Expected " + expected + "args; got " + actual);
}
function f(x,y,z) {
    check(arguments); //실제 인자 개수가 요구 개수와 같은지 검사한다
    return x +y + z; //함수의 나머지 로직 수행
}
```

7.2 prototype 프로퍼티

모든 함수는 서로 다른 프로토타입을 가지고 있고, 프로토타입 객체를 참조한다

함수가 생성자로 사용될 때 새로 생성된 객체는 함수의 프로토 타입 객체로부터 프로퍼티들을 상속 받는다

7.3 call()과 apply() 메서드

call()과 apply()는 어떤 함수를 간접적으로 호출할 수 있게 하며 특정함수를 다른 객체의 메서드인 것처럼 다룰 수 있다

첫 번째 인자는 호출되는 함수와 관련이 있는 객체 (호출 컨텍스트, 함수 몸체에서 this키워드의 값)

함수.call(객체, 전달할 인자, ...,전달할 인자); 함수.apply(객체, [전달할 인자, ...,전달할 인자]);

7.4 bind() 메서드

함수와 객체를 서로 묶는 것

새로운 함수를 반환 하는데 새로운 함수에 전달한 모든 인자는 원래 함수에도 전달된다

새로 만든 함수를 호출하면 원래 함수가 객체.원래 함수로 호출이 된다

function.bind(thisArg[,arg1[,arg2[,argN]]])

```
function f(y) { return this.x +y; } //바인드되어야 하는 함수
var o = { x : 1}; //바인드될 객체
var g = f.bind(o); //g(x)를 호출하면 o.f(x)가 호출된다
g(2) //3
//최대값 최소값 사이의  값인지 알려주는 함수
var checkNumericRange = function (value) {
    if (typeof value !== 'number')
        return false;
    else
                return value >= this.minimum && value <= this.maximum;
    }
//최소값 최대값 설정
var range = { minimum: 10, maximum: 20 };
// Bind 로 checkNumericRange 함수생성
var boundCheckNumericRange = checkNumericRange.bind(range);
// 바인딩된 함수사용
    var result = boundCheckNumericRange (12);
document.write(result);             // true
```

7.5 toString() 메서드

함수 선언 구문 다음에 나오는 문자열을 반환

실제 대부분은 toString() 메서드의 구현체들은 함수의 전체 소스코드를 반환

7.6 Function() 생성자

Function() 생성자를 통해서 함수를 정의 할 수 있다

```
var f = new Function("x","y","return x*y;");   //생성자를 사용한 정의
var f = function(x, y) { return x*y; }             //키워드를 사용한 정의
```

임의 개수의 문자열 인자를 요구 (마지막 인자는 함수 몸체에 대한 텍스트)

익명 함수를 생성한다 (함수 리터럴과 같음)

중요한 점

- 동적으로 자바스크립트 함수를 생성하고 실행 시간에 컴파일 되는 것을 가능케 한다
- 생성자가 호출될 때마다 함수 몸체를 분석하고 새로운 함수 객체를 생성한다
- 함수생성자가 생성하는 함수는 어휘적 유효범위를 사용하지 않는다
  (언제나 최상위 레벨 함수로 컴파일)

new Function()으로 생성된 함수 객체를 호출하면 eval()함수를 호출하는 것과 같다(전역의 eval())

eval은 컨텍스트 및 스코프 관리가 어려워지므로 사용하지 않는 것이 좋다

- new Function도 사용하지 않는 것이 좋다

7.7 호출 가능한 객체

함수 호출 표현식을 통해 호출할 수 있는 객체

IE9부터 진짜 함수를 사용하도록 바뀌어서 호출가능객체는 사라지고 있다

RegExp 객체를 직접 호출 할 수 있다고 해도 이를 직접 호출하는 코드는 작성하지 않도록 한다
 - 더는 사용되지 않을 것이고 미래에는 제거될 것이다
진짜 함수 객체인지 알아보는 법

```
function isFunction(x) {
        return Object.prototype.toString.call(x) ==="[object Function]";
}
```

8. 함수형 프로그래밍

프로그램 전체를 함수의 집합으로 구성하는 프로그램
자바스크립트는 함수형 프로그래밍을 구현할 수 있는 스크립트언어이다
map(), reduce() 같은 배열 메서드는 함수형 프로그래밍 스타일에 적합한 구조를 지니고 있다