함수

함수

❖ 함수의 정의

```
function 함수명(인수, ...) {
   // 함수 로직

   return 반환값;
}
```

```
function getTriangle(base, height) {
  return base * height / 2;
}

console.log('삼각형의 면적:' + getTriangle(5, 2));
```

함수

❖ 함수 리터럴 표현으로 정의

- 익명함수
- 변수에 배정해서 사용

```
var getTriangle = function(base, height) {
  return base * height / 2;
};

console.log('삼각형의 면적:' + getTriangle(5, 2));
```

함수

❖ 화살표 함수

- 다른 언어에서는 람다 함수라고 함

```
(인수, ...) => { 함수 본체 }
```

- 인수가 1개인 경우 괄호 생략 가능
  - 인수 => { 함수 본체 }
- 함수 본체가 1줄인 경우 { } 생략 가능, return 키워드 생략 가능

```
let getTriangle = (base, height) => {
  return base * height / 2;
};

console.log('삼각형의 면적:' + getTriangle(5, 2));
```

```
let getTriangle = (base, height) => base * height / 2;

let getCircle = radius => radius * radius * Math.PI;
```

함수

❖ 함수 정의시 주의 사항

- return 문
  - 중간에 줄바꿈 하지 않음

```
var getTriangle = function(base, height) {
  return
  base * height / 2;
//return base * height / 2;
};

console.log('삼각형의 면적:' + getTriangle(5, 2));
```

함수

❖ 스코프

- 전역 스코프: 함수 밖에서 선언한 변수
- 지역 스코프: 함수 안에서 선언한 변수

```
scope = 'Global Variable';

function getValue() {
  scope = 'Local Variable';
  return scope;
}

console.log(getValue());
console.log(scope);
```

```
var scope = 'Global Variable';

function getValue() {
  var scope = 'Local Variable';
  return scope;
}

console.log(getValue());
console.log(scope);
```

함수

❖ 호이스팅(hoisting)

```
var scope = 'Global Variable';
function getValue() {
  console.log(scope);
  var scope = 'Local Variable';
  return scope;
}

console.log(getValue());
console.log(scope);
```

함수

❖ 매개변수

```
var value = 10;

function decrementValue(value) {
  value--;
  return value;
}

console.log(decrementValue(100));
console.log(value);
```

```
var value = [1, 2, 4, 8, 16];
function deleteElement(value) {
  value.pop();
  return value;
}

console.log(deleteElement(value));
console.log(value);
```

함수

❖ 블록 레벨 스코프

- let
  - var는 함수 레벨 스코프

```
if (true) {
  var i = 5;
}

console.log(i);
```

```
if (true) {
  let i = 5;
}

console.log(i);
```

함수

❖ 블록 레벨 스코프

- switch 문에서의 let 사용 주의

```
switch(x) {
  case 0:
    let value = 'x:0';
  case 1:
    let value = 'x:1';    // 에러
}
```

인수의 다양한 표기법

❖ 인수의 수를 체크하지 않음

```
function showMessage(value) {
  console.log(value);
}

showMessage();
showMessage('철수');
showMessage('철수', '영희');
```

인수의 다양한 표기법

❖ 인수의 수를 체크하지 않음

- arguments 객체
  - 함수의 속성
  - 실제 전달한 인자를 가지는 유사 배열 객체

```
function showMessage(value) {
  if (arguments.length !== 1) {
    throw new Error('인수의 수가 서로 다릅니다 : ' + arguments.length);
  }
  console.log(value);
}

try {
  showMessage(' 철수', ' 영희');
} catch(e) {
  window.alert(e.message);
}
```

인수의 다양한 표기법

❖ 인수의 디폴트 값 설정

```
function getTriangle(base, height) {
  if (base === undefined) { base = 1; }            // base = base || 1
  if (height === undefined) { height = 1; }        // height = height || 1
  return base * height / 2;
}

console.log(getTriangle(5));
```

```
function getTriangle(base=1, height=1) {

  return base * height / 2;
}

console.log(getTriangle(5));
```

인수의 다양한 표기법

❖ 필수 인수

```
function show(x, y = 1) {
  console.log('x = ' + x);
  console.log('y = ' + y);
}

show();
```

```
function required() {
  throw new Error('인수가 부족합니다.');
}

function hoge(value = required()) {
  return value;
}

hoge();
```

인수의 다양한 표기법

❖ 가변 길이 인수

```
function sum() {
  var result = 0;
  for (var i = 0, len = arguments.length; i < len; i++) {
    var tmp = arguments[i];
    if (typeof tmp !== 'number') {
      throw new Error('인수값이 숫자가 아닙니다. : ' + tmp);
    }
    result += tmp;
  }
  return result;
}

try {
  console.log(sum(1, 3, 5, 7, 9));
} catch(e) {
  window.alert(e.message);
}
```

인수의 다양한 표기법

❖ 가변 길이 인수 - 나마지 인수

```
function sum(...nums) {
  let result = 0;
  for (let num of nums) {
    if (typeof num !== 'number') {
      throw new Error('지정값이 숫자가 아닙니다:' + num);
    }
    result += num;
  }
  return result;
}

try {
  console.log(sum(1, 3, 5, 7, 9));
} catch(e) {
  window.alert(e.message);
}
```

인수의 다양한 표기법

❖ ...를 이용한 배열의 전개

```
console.log(Math.max(15, -3, 78, 1));
console.log(Math.max([15, -3, 78, 1]));
```

```
console.log(Math.max(...[15, -3, 78, 1]));
```

인수의 다양한 표기법

❖ 이름 있는 인수 - 객체

```
function getTriangle(args) {
  if (args.base === undefined) { args.base = 1; }
  if (args.height === undefined) { args.height = 1; }
  return args.base * args.height / 2;
}

console.log(getTriangle({ base:5, height:4 }));
```

```
function getTriangle({ base = 1, height = 1 }) {
  return base * height / 2;
}

console.log(getTriangle({ base:5, height:4 }));
```

인수의 다양한 표기법

❖ 이름 있는 인수 - 객체 프로퍼티 추출

```
function show({name}) {
  console.log(name);
};

let member = {
  mid: 'Y0001',
  name: '정시온',
  address: 'shion_jung@example.com'
};

show(member);
```

함수 호출과 반환값

❖ 복수의 반환값을 개별 변수에 대입하기

```
function getMaxMin(...nums) {
  return [Math.max(...nums), Math.min(...nums)];
}

let result = getMaxMin(10, 35, -5, 78, 0);
console.log(result);

let [max, min] = getMaxMin(10, 35, -5, 78, 0);
//let [,min] = getMaxMin(10, 35, -5, 78, 0);
console.log(max);
console.log(min);
```

함수 호출과 반환값

❖ 재귀함수

```
function factorial(n) {
  if (n != 0) { return n * factorial(n - 1); }
  return 1;
}

console.log(factorial(5));
```

함수 호출과 반환값

❖ 고차 함수

- 매개변수 또는 리턴값으로 함수를 사용하는 함수

```
function arrayWalk(data, f) {
  for (var key in data) {
    f(data[key], key);
  }
}

function showElement(value, key) {
  console.log(key + ' : ' + value);
}

var ary = [1, 2, 4, 8, 16];
arrayWalk(ary, showElement);
```

함수 호출과 반환값

❖ 고차 함수

- 매개변수 또는 리턴값으로 함수를 사용하는 함수

```
function arrayWalk(data, f) {
   for (var key in data) {
     f(data[key], key);
   }
}

var result= 0;
function sumElement(value, key) {
  result += value;
}

var ary = [1, 2, 4, 8, 16];
arrayWalk(ary, sumElement);
console.log('합계:' + result);
```

함수 호출과 반환값

❖ 익명함수

- 이름이 없는 일회용 함수
- 함수를 매개변수로 전달할 때 주로 사용

```
function arrayWalk(data, f) {
  for (var key in data) {
    f(data[key], key);
  }
}

var ary = [1, 2, 4, 8, 16];
arrayWalk(
  ary,
  function (value, key) {
    console.log(key + ' : ' + value);
  }
);
```

함수 호출과 반환값

❖ 익명함수

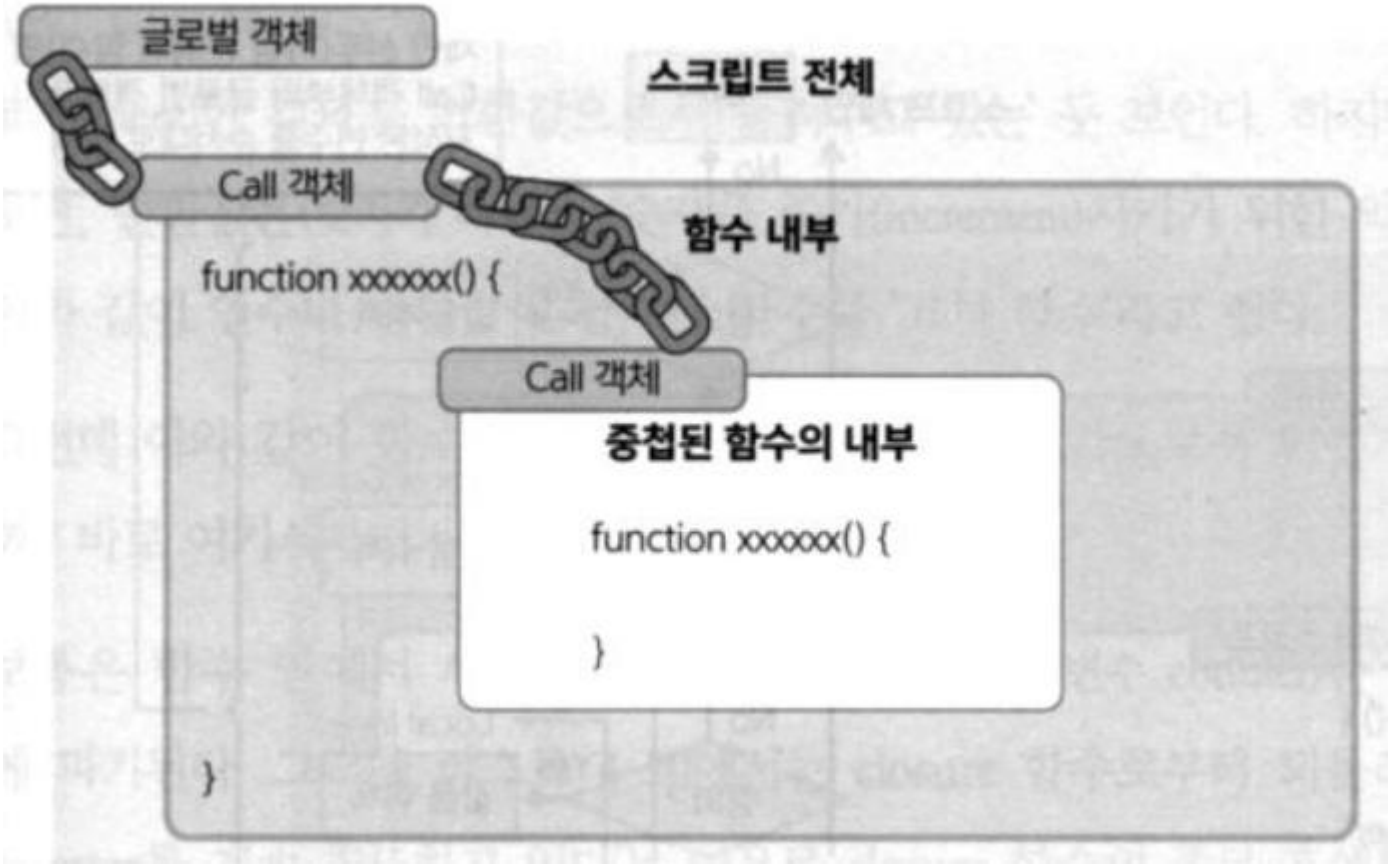
- 간단한 경우 화살표 함수 정의

```
function arrayWalk(data, f) {
  for (var key in data) {
    f(data[key], key);
  }
}

var ary = [1, 2, 4, 8, 16];
arrayWalk(
  ary,
  (value, key) => console.log(key + ' : ' + value)
);
```

함수의 고급 기법

❖ 스코프 체인



함수의 고급 기법

❖ 스코프 체인

```
var y = 'Global';
function outerFunc() {
  var y = 'Local Outer';

  function innerFunc() {
    var z = 'Local Inner';
    console.log(z);
    console.log(y);
    console.log(x);
  }
  innerFunc();
}
outerFunc();
```

함수의 고급 기법

❖ 클로저

- 지역 변수를 참조하고 있는 함수 내의 함수(내부 함수)

```
function closure(init) {
  var counter = init;

  return function() {
    return ++counter;
  }
}

var myClosure = closure(1);
console.log(myClosure());
console.log(myClosure());
console.log(myClosure());
```

함수의 고급 기법

❖ 클로저

- 바깥 함수를 호출할 때마다 클로저 인스턴스가 생성

```
function closure(init) {
  var counter = init;

  return function() {
    return ++counter;
  }
}

var myClosure1 = closure(1);
var myClosure2 = closure(100);

console.log(myClosure1());
console.log(myClosure2());
console.log(myClosure1());
console.log(myClosure2());
```