Ch05 함수

목표 : 익명함수, 선언함수, 매개변수와 리턴값, 가변인자함수, 내부함수, 자기호출함수, 콜백함수, 클로저함수, 내장함수(<https://www.w3schools.com/jsref>, http://devdocs.io/javascript)

* 1. 익명 함수

함수는 코드의 집합을 나타내는 자료형. 익명 함수는 다음 형태로 만든다.

**var functionVar = function() { };**

* 익명 함수 생성과 출력 ; 이름이 없으므로 변수에 넣어 사용해야 한다.

<script>

var funVar = function () {

var output = prompt('숫자를 입력해 주세요');

alert('익명합수 호출됨. 입력하셨던 숫자는 ' + output);

};

alert(funVar);

alert(typeof (funVar));

funVar(); // 익명함수호출은 함수형변수에 ()해서 호출

</script>

* 기존에 사용하던 함수들도 소스를 볼 수 있을까라는 생각에 아래와 같이 실행해 보지만 내장함수는 코드를 볼 수 없게 막아 놓았다. 아래의 함수와 위의 함수의 차이는 내장함수와 사용자가 만든 함수라는 차이점 외에 함수의 이름이 있고 없고의 차이가 있다. 이렇게 이름이 있는 함수를 ‘선언적 함수’라 부른다. 함수의 이름이 없는 함수를 ‘익명함수’라 부른다.
* 자바스크립트에서 함수는 하나의 자료형이지만 여타 자료형과 다르게 뒤에 괄호를 열고 닫음으로써 코드를 실행한다. 이를 함수를 호출한다고 표현한다.

<script>

//anonymous\_fun();// 에러

var anonymous\_fun = function (i) {

alert(i);

};

var i = 10;

var anonymous\_fun = function () { alert('재정의했네'); };

anonymous\_fun();

</script>

* 1. 선언적 함수

※ **function 함수명 ( ) {**

**}**

<script>

function fun() { alert('함수 A'); }

function fun() { alert('함수 B'); }

fun();

</script>

* 익명함수와 선언적 함수의 차이

<script>

fun();

function fun() { alert('함수 A'); } //웹브라우저는 script 태그 내부의 내용을 읽기전에 선언적 함수부터 읽는다

function fun() { alert('함수 B'); } // 함수를 재정의 하면 기본의 함수는 없어지고, 재정의한 함수를 따른다

</script>

<script>

fun\_anonymous();

var fun\_anonymous = function () { alert('함수 A'); }

var fun\_anonymous = function () { alert('함수 B'); }

</script>

<script>

fun(); // 함수3 경고

function fun() {

alert('함수1');

}

function fun() {

alert('함수2');

}

fun(); // 함수3 경고

var fun = function () {

alert('함수3');

};

fun(); // 함수3 경고

</script>

* 1. 매개변수와 리턴값

function 함수이름(매개변수, 매개변수, 매개변수) {

//함수 코드

return 리턴값;

}

* 리턴값이 없는 경우

<script>

var fun = function (i) {

alert('리턴값이 없는 함수 '+i);

};

var result = fun(20);

alert('함수결과 : '+result);

</script>

* X의 Y승을 리턴하는 함수를 선언하는 코드를 구현하고 x값과 y값을 입력받아 x의 y승을 출력하는 javaScript 프로그램을 구현하시오.

<script> // 사용자로부터 x, y값을 받아 (x의 y승)을 화면출력

var x = prompt("x의 y승을 구할 거예요. x값은 ?");

var y = prompt("x의 y승을 구할 거예요. y값은 ?");

var result = pow(x);

// 함수에 선언된 매개변수보다 적게 사용되면 지정되지 않는

// 매개변수는 undefined로 입력된다

document.writeln("x(" + x + ")의 y(" + y + ")승은 " + result);

function pow(x, y) {

var temp = 1;

for (var i = 1; i <= y; i++) {

temp \*= x; // temp = temp \* x;

}

return temp;

}

</script>

* 1. 매개변수

자바 스크립트는 함수를 생성할 때 지정한 매개변수보다 많거나 적은 매개변수를 사용하는 것을 허용한다. 원래 함수에서 선언된 매개변수보다 많게 사용하면 일반적으로 추가된 매개변수는 무시한다. 원래 함수에서 선언한 매개변수보다 적게 사용하면 지정하지 않는 매개변수는 undefined로 입력된다.

<script>

alert('원래 매개변수입니다', '추가된 매개변수입니다');

function f(x, y) {

var temp = 1;

for (var i = 1; i <= y ; i++) {

temp \*= x;

}

return temp;

}

alert(f(10));

alert(f(10,2));

</script>

※ Array()함수 ; 빈 배열을 만든다.

Array(number) ; 매개변수만큼의 크기를 가지는 배열을 만든다.

Array(any, …, any); 매개변수를 배열로 만든다.

<script>

/\* 내장함수 중 Array 함수 :

\* 매개변수가 0개 : length가 0인 배열생성

\* 매개변수가 1개일 때 : 매개변수만큼의 크기를 가지는 배열생성

\* 매개변수가 2개이상일때는 매개변수로 배열을 만든다. \*/

var arr1 = [273, 2, 3, 'Hello'];

var arr2 = Array(273, 2, 3, 'Hello');

var arr3 = [ , , , , ,];

var arr4 = Array(5);

var arr5 = [];

var arr6 = Array();

document.writeln('arr1\*\* : ' + arr1 +'<br>');

document.writeln('arr2\*\* : ' + arr2 + '<br>');

document.writeln('arr3 : ' + arr3 + ' 방크기' + arr3.length +'<br> ');

document.writeln('arr4 \*\* : ' + arr4 + ' 방크기' + arr4.length + '<br> ');

document.writeln('arr5 \*\*: ' + arr5 + ' 방크기' + arr5.length + '<br> ');

document.writeln('arr6 \*\*: ' + arr6 + ' 방크기' + arr6.length + '<br> ');

</script>

* 1. 가변 인자 함수 ; 매개변수를 선언된 형태와 다르게 사용했을 때, 매개변수를 모두 활용할 수 있는 함수. 매개변수의 개수가 변할 수 있는 함수 ex. Array()함수
* arguments : 매개변수 배열

function arrayFun() {

if (arguments.length == 0) {

// 매개변수가 없을 때 할 명령어들

alert('매개변수 없이 호출했구만');

} else if (arguments.length == 1) {

// 매개변수가 1개일 때 할 명령어들

alert('매개변수 하나 ' + arguments[0] + '입력했구만');

} else if (arguments.length >= 2) {

// 매개변수가 2개 이상일 대 할 명령어들

var output = '매개변수를 2개 이상 ';

for (var idx = 0; idx < arguments.length; idx++) {

output += arguments[idx] + ' ';

}

output += ' 입력했구만';

alert(output);

}

}

arrayFun();

arrayFun(2, 3, 4);

arrayFun(9999);

</script>

<script>

var array = function () {

var result = [];

var length = arguments.length;

if (length == 0) {

result = [];

} else if (length == 1) {

// 매개변수 수만큼 result.push(); 수행

for (var i = 0; i < arguments[0]; i++) {

result.push('');

}

} else if (length >= 2) {

// 매개변수 갖지고 result에 push() 수행

for (var i = 0; i < arguments.length; i++) {

result.push(arguments[i]);

}

}

return result;

};

// 호출

var arr1 = array();

var arr2 = array(5);

var arr3 = array(2, 4, 5);

document.writeln('arr1(array()) : ' + arr1);

document.writeln('<br>');

document.writeln('arr2(array(5)) : ');

document.writeln('<br>');

document.writeln('arr3(array(2, 4, 5)) : ' + arr3);

</script>

* 연습문제 : 아래와 같이 매개변수가 없으면 -999를 리턴하고 매개변수가 1개 이상이면 누적합을 리턴하는 가변인자 함수 sumAll()을 작성한 스크립트 파일을 이용하시오.

<script>

var sum = sumAll(); // -999 (매개변수가 없으면 -999를 리턴)

document.write('sumAll() = ' + sum + '<br>');

var sum = sumAll(1); // 1 (매개변수가 1개 이상이면 누적값리턴)

document.write('sumAll(1) = ' + sum + '<br>');

var sum = sumAll(1, 2, 3, 4, 5); // 15

document.write('sumAll(1, 2, 3, 4, 5) = ' + sum + '<br>');

</script>

* 1. 리턴값 return;
  2. 내부 함수 ; 함수 내부에 선언하는 함수

<script>

/\* A씨가 만든 함수 \*/

function square(x) { return x \* x; }

function pythagoras(width, height) {

return Math.sqrt(square(width) + square(height));

}

</script>

<script>

/\* B씨가 만든 함수 \*/

function square(width, height, hypotenuse) {

if (width \* width + height \* height == hypotenuse \* hypotenuse) {

return true;

} else {

return false;

}

}

</script>

<script>

alert(pythagoras(3, 4));

</script>

* 위의 충돌을 막기 위해 내부함수 사용하면 내부함수가 우선된다. jQuery는 대부분 선언적 함수를 내부함수로 작성하고 있다.

<script>// A가 만든 부분 : 직각삼각형의 빗변을 구하는 함수

function phthagoras(w, h) {

function square(x) { // x의 제곱을 리턴하는 함수(내부함수)

return x \* x;

}

return Math.sqrt(square(w) + square(h));

//return Math.sqrt((function (w) { return w \* w; })(w) +

// (function (w) { return w \* w; })(h); 자기호출함수

}

</script>

<script> /\* B가 만든 부분 : 빗변, 밑변, 높이를 입력받아

직각삼각형이면 true/직각삼각형이 아니면 false 리턴하는 함수\*/

function square(p, w, h) {

if (w \* w + h \* h == p \* p) { // 직각삼각형

return true;

} else {

return false;

}

}

</script>

<script> // 사용자로부터 w, h을 받아 빗변을 출력

var w = prompt("직각삼각형의 밑변 길이는 ?");

var h = prompt("직각삼각형의 높이 길이는 ?");

document.writeln('밑변(' + w + '), 높이(' + h + ')에서 나온 ');

document.writeln('빗변은 ' + phthagoras(w, h));

</script>

※ 자기호출함수

(fuction(){ … })(); //다른 개발자에게 영향을 주지 않게 함수를 생성하자마자 호출하는 경우

* 1. 콜백함수 ; 자바스크립트에서는 함수도 하나의 자료형이므로 매개변수에 넣을 수 있다. 매개변수에 넣어 전달된 함수를 콜백함수라 한다.

<script>

function callTenTimes(callback) {

for (var i = 0; i < 10; i++) {

document.write(i);

callback();

}

}

var callback = function () { document.write(' 함수호출<br>'); };

callTenTimes(callback);

</script>

※ 콜백함수는 익명함수로 사용하는 경우도 많다.

<script>

function callTenTimes(callback) {

for (var i = 0; i < 10; i++) {

document.write(i);

callback();

}

}

callTenTimes(function () { document.write(' 함수호출<br>'); });

</script>

* 1. 함수를 리턴하는 함수

※ 함수가 자료형이므로 매개변수로도 전달할 수 있지만 함수를 리턴하는 함수도 만들 수 있다. 특정함수를 여러 번 수행하거나 이벤트 처리를 할 때 필요. 리턴하는 함수를 쓰는 가장 큰 이유는 클러저 때문.

<script>

function returnFunc() {

return function () { alert('Hello Function..!') };

}

returnFunc()();

</script>

* 1. 클로저 ; 리턴된 함수

<script>

function test(name) {

var output = 'Hello' + name + '님';

}

alert(output);

</script>

※ 위의 경우 output이 지역변수이므로 함수 외부에서 사용할 수 없어 아래와 같이 클로저를 사용해 본다.

<script>

function test(name) {

var output = 'Hello' + name + '님';

return function () {

alert(output);

}

}

test('홍길동')(); //140p.

</script>

※ 클로저 함수로 인해 남은 지역 변수는 각각의 클로저 함수의 고유한 변수이다. test1()과 test2()를 호출하면 각 함수가 고유한 지역 변수 output(홍길동, 김길동)이 있다. 향후 7장에서 다시 관련내용을 살펴봅니다

<script>

function test(name) {

var output = 'Hello ' + name + '님';

return function () {

alert(output);

};

}

var test1 = test('홍길동'); //변수 선언

var test2 = test('김길동');

test1(); // 함수를 호출

test2();

</script>

* 1. 자바 스크립트 내장 함수
     1. 타이머 함수
        + setTimeout(function, millisecond) ; 일정시간 후 함수를 한번 실행
        + setInterval(function, millisecond) ; 일정시간마다 함수를 반복해서 실행
        + clearTimeout(id) ; setTimeout 중지
        + clearInterval(id); setInterval 중지

<script>

setTimeout(function () { alert('3초가 지났네요') }, 3000);

</script>

<script>

var intervalID = setInterval(function () {document.write('<h4>' + new Date() + '</h4>'); }, 1000);

setTimeout(function () {clearInterval(intervalID); }, 5000);

</script>

* 다음 코드의 실행 순서는 A, C, B이다. 자바스크립트 함수 중에 브라우저에 처리를 부탁하는 함수인 타이머 함수는 웹 브라우저가 처리하고 처리가 완료되었다는 것을 자바 스크립트에게 알려주는 방식이다. 이러한 함수는 현재 단위가 끝나기전에는 실행되지 않는다.

<script>

alert('A');

setTimeout(function () {alert('B');}, 0);

alert('C');

</script>

* 다음 코드의 실행 결과는 3, 3, 3이다. 0,1,2를 출력한다고 생각하겠지만 setTimeout()함수를 호출하는 시점이 반복문이 모두 끝난 이후이므로 발생하는 문제이다.

<script>

for (var i = 0; i < 3; i++) {

setTimeout(function (){alert(i);},0);

}

</script>

* 따라서 0,1,2를 출력하려면 자기 호출 함수와 콜백함수를 아래와 같이 활용한다.

<script>

for (var i = 0; i < 3; i++) {

(function (j) {setTimeout(function () { alert(i); }, 0);})(i);

}

</script>

<script>

[0, 1, 2].forEach(function (data, idx) {

setTimeout(function () { alert(data);},0);

});

</script>

* + 1. 인코딩과 디코딩 함수 ; 인코딩은 문자를 컴퓨터에 저장하거나 통신에 사용할 목적으로 부호화하는 방법이다. 웹에서는 통신할 때 한글 같은 유니코드 문자가 오작동을 일으킬 수 있으므로 인코딩해야 한다. 문자를 부호화하는 것이 인코딩, 그 반대로 돌리는 것이 디코딩이다.

Google.com을 통해 검색하면 디코딩된 결과를 볼 수 있다.

encodeURIComponent(uriConponent) ; 대부부의 문자를 모두 인코딩

decodeURIComponent(encodedURI); 대부분의 문자를 모두 디코딩

<script>

alert(encodeURIComponent('더존'));

alert(decodeURIComponent(encodeURIComponent('더존')));

</script>

* + 1. 코드 실행함수
* eval(수식이나 자바스크립코드); 자바스크립 코드를 실행

<script>

var willEval = '';

willEval += 'var number = 10;';

willEval += 'alert(number);';

eval(willEval);

alert(number);

</script>

* + 1. 숫자 확인 함수
       - isFinite() ; number가 유한한 값인지 확인
       - isNaN() ; number가 NaN인지 확인
    2. 숫자 변환 함수

<script>

var won = '1000원';

var dollar = '1.5$';

alert(Number(won) + ' : ' + Number(dollar));

alert(parseInt(won) + ' : ' + parseFloat(dollar));

</script>

총괄 예제 : 1초마다 10초동안 body의 백그라운드 이미지가 바뀌고 시간이 출력되도록 자바스크립트로 구현 후, [yisy0703@naver.com](mailto:yisy0703@naver.com)으로 사진과 html파일을 압축하여 첨부하시오.(이미지파일 최소 3개)