INTERACTION PROGRAMMING 1

인터랙션 프로그래밍 1

3Week.

2018. 3. 22.

JavaScript Review

JavaScript 작성법

```
<!-- JavaScript -->
<script type="text/javascript">
    //code.
</script>
```

```
//Data type.
Boolean      true | false
Number       -1 0 1 2 3 4 5...
String       "a" "b" "c"...
undefined    undefined
null         null
1            Boolean 의 true 로 간주
1 이 아닌 수   Boolean 의 false 로 간주
NaN          성립이 되지 않는 수, 계산할 수 없음을 의미함.
```

일치 연산자

```
//대입 연산자(이항 연산자)
=

//동등 연산자
==

//일치연산자
===
```

Object

```
var person = {
    name : "김용원",
    job : "교수",
    phone : "010-9137-8688",
    email : "rh@102labs.com"
};
```

```
var a = 13;
a %= 3;
```

연산

```
var a = 13;
a %= 3;
a = a % 3;
```

비교 연산자

```
true === false
```

비교 연산자

```
true === 1
```

비교 연산자

```
typeof a !== undefined
```

비교 연산자

```
typeof a !== 'undefined'
```

비교 연산자

```
10 > 1
```

비교 연산자

```
11 >= '11'
```

산술 연산

```
var a = 5;
var b = a++;
```

산술 연산

```
var a = 5;
var b = ++a;
```

문자 연산

```
var a = +'10';
```

문자 연산

```
var a = 'hello' + 9;
```

조건문

if 안의 조건이 true 인 경우.
if 안의 조건이 false 인 경우.

true 일 경우 if 조건 안의 코드가 실행.
false 일 경우 else 안의 코드가 실행.

조건문

```
var a = 3;
var b = 10;
var c;
if (typeof c === 'undefined') {
    c = a % b;
};
```

조건문

```
var a = 3;
var b = 10;
var c = 0;
if (typeof c === 'undefined') {
    c = a % b;
};
```

조건문

```
var a = 3;
var b = 10;
var c = 0;
if (a && b) {
    c = a % b;
};
```

조건문

```
var a = 10;
var b = 0;
if (a && b) {
    console.log('---->');
};
```

조건문

```
var a = 10;
var b = 0;
if (b || a) {
    console.log('<----');
};
```

Array

Array 객체

한 번에 두가지 이상의 값을 포함할 수 있는 객체

사용빈도가 아주 높다.

Array 배열

```
var a = 10;
var b = 'apple';
var c = null;
var d = a;
var _array = [a, b, c, d];
console.log(_array[3]);
```

반복문

반복문

동일한 동작을 반복 수행할 수 있도록 도와주는 구문

for

while

반복문

```
var _array = ['a', 'b', 'c', 'd', 'e'];
for (var i = 0; i < _; i++) {
    console.log(_________);
};
```

반복문

```
var _array = ['a', 'b', 'c', 'd', 'e'];
for (var i = 0; i < 5; i++) {
    console.log(_array[i]);
};
```

```
var _say = ['Hello', 'My', 'Name', 'is', '_____'];
```

Hello My Name is _____.

위와 같이 출력되도록 반복문을 이용해서 작성해봅시다.

반복문

```
var _say = ['Hello', 'My', 'Name', 'is', '____'];
var say = '';
for (var i = 0; i < 5; i++) {
    say += _say[i];
};
console.log(say);
```

반복문

```
var _say = ['Hello', 'My', 'Name', 'is', '____'];
var say = '';
for (var i = 0; i < 5; i++) {
    if (i !== 0) {
        say += ' ';
    };
    say += _say[i];
    if (i === 4) {
        say += '.';
    };
};
console.log(say);
```

function

Function 함수

재사용을 위해 코드 블록을 감싸는 방법.

작업을 수행하거나 값을 계산하는 등의 역할을 수행하는 코드를 포함.
호출을 통해 내부의 코드를 동작시킨다.
유효범위를 가지고 있으며, 실행시 값을 반환하도록 되어 있다.

function 함수

```
function name() {
};
name();
```

function 함수

```
function name(param) {
};
name();
```

```
var a = 20, b = 30, c = 10;
console.log(c);
function addNumbers() {
    c = b + a;
};
console.log(c);
addNumbers();
console.log(c);
```

scope

Scope 유효범위

코드의 참조 범위.
변수와 매개변수의 접근성과 생존 기간을 의미.

전역변수(범위) / 지역변수(범위)로 구분.
전역범위는 스크립트 내의 모든 곳이 참조 가능.
지역범위는 지정된 함수 내부에서만 참조 가능.

```
var global = 'global';
var a = 'My';
var b = 'Name';
function local() {
    var local = 'local';
    var a = 'Hello';
    var b = 'World';
    console.log(a + ' ' + b);
};
local();
console.log(a + ' ' + b);
```

```
var global = 'global';
var a = 'My';
var b = 'Name';
function local() {
    var local = 'local';
    var a = 'Hello';
    var b = 'World';
    console.log(a + ' ' + b);
};
local();
console.log(a + ' ' + b);
```

Global

```
var global = 'global';
var a = 'My';
var b = 'Name';
function local() {
    var local = 'local';
    var a = 'Hello';
    var b = 'World';
    console.log(a + ' ' + b);
};
local();
console.log(a + ' ' + b);
```

Local

Global

```
var global = 'global';
function local() {
    var local = 'local';
};
```

method

Method 메서드

작업을 수행하거나 값을 연산하는 등의 역할을
수행하는 코드를 포함하고 있는 블럭단위의 뭉치.

함수와 의미적으로 동일하나, 객체에 의존되어 있는 함수를 칭함.

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

Function

```
var _obj = {
    sayHello : function() {
        console.log('Hello!');
    }
};
_obj.sayHello();
```

Method

함수의 종류

함수의 종류

JavaScript 에서 함수는 총 3가지 종류.

1. 명시적 함수
2. 익명 함수 (함수 생성자)
3. 함수 리터럴 (함수식)

명시적 함수

```
function sayHello(name) {
    return 'Hello ' + name;
};
var say1 = sayHello('World');
var say2 = sayHello('Everyone');
console.log(say1);
console.log(say2);
```

익명 함수 (함수 생성자)

```
var sayHello = new Function('name', 'return "Hello " + name');
sayHello('World');
```

함수 리터럴 (함수식, 익명함수)

```
var sayHello = function(name) {
    return 'Hello ' + name;
};
```

함수 작성법

선언식 / 표현식

일반적 함수는 선언식.
메서드 함수는 표현식.

함수 작성법

```
//함수 :: 선언식.
function sayHello() {
    console.log('Hello~');
};
```

함수 작성법

```
//함수 :: 표현식
var sayHello = function() {
    console.log('Hello!');
};
```

함수 호출(실행)

함수 호출(실행)

함수, 메서드 모두 동일하게 () 를 이용하여 호출(실행)함으로써
내부의 코드를 동작시킴.

함수 호출(실행)

```
//함수 :: 선언식.
function sayHello() {
    console.log('Hello~');
};
//함수 호출.
sayHello();
```

함수 호출(실행)

```
//함수 :: 표현식
var sayHello = function() {
    console.log('Hello!');
};
//함수 호출.
sayHello();
```

함수 호출(실행)

```
//메서드
var _obj = {
    sayHello : function(){
        console.log('Hello?');
    }
};
//메서드 함수 호출.
_obj.sayHello();
```

매개변수

Parameter 매개변수

변수의 한 종류로, 함수에 전달되는 여러 데이터 중 하나를 의미함.
매개변수의 목록은 함수를 정의하는 부분에 포함되며, 매 함수 호출시 함수에 주입된다.

매개변수

```
function sendMessage(msg) {
    console.log(msg);
};
```

전달인자

Argument 전달인자

함수에 정의된 매개변수를 통해 전달되는 실제 값.

매개변수

```
function sendMessage(msg) {
    console.log(msg);
};
sendMessage('Hello');
```

매개변수

```
function sendMessage('Hello') {
    console.log(msg);
};
sendMessage('Hello');
```

매개변수

```
function sendMessage('Hello') {
    console.log('Hello');
};
sendMessage('Hello');
```

매개변수 유효성

매개변수 유효성

함수에 전달시 사용되는 매개변수는 실행시점에 유효할 수도 유효하지 않을 수도 있다.
코드 실행시 유효성을 검증할 필요가 있다.

매개변수 유효성

```
function sendMessage(msg) {
    console.log(msg);
};
sendMessage('Hello');
```

매개변수 유효성

```
function sendMessage(msg) {
    console.log(msg);
};
sendMessage();
```

매개변수 유효성

```
function sendMessage(msg) {
    if (typeof msg === 'undefined') {
        console.log('No Message');
    } else {
        console.log(msg);
    };
};
sendMessage();
```

매개변수 활용

매개변수 활용

매개변수는 어떤 값도 전달인자로 대입될 수 있으며,
전달할 수 있는 매개변수는 다수로 사용이 가능하다.

정의된 매개변수의 수보다 많은 수의 값을 전달할 경우도 문제없이 실행 가능.
정의된 수보다 전달인자가 적을 경우 값이 없는 매개변수는 **undefined** 가 할당된다.

매개변수 활용

```
function sum(number1, number2) {
    console.log(number1 + number2);
};
sum(1, 2);
```

매개변수 활용

```
function sum(number1, number2) {
    console.log(number1 + number2);
};
sum(1, 2, 3, 4, 5, 6);
```

매개변수 활용

```
function sum(number1, number2) {
    console.log(number1 + number2);
};
sum(1);
```

함수 반환

함수는 실행시 반드시 결과를 반환한다.

return 문을 이용하여 반환할 값을 반환하며,
return 문이 없을 경우, **undefined** 를 반환한다.

함수 반환

```
function sumPrices(price1, price2) {
    price1 + price2;
};
var price = sumPrices(10, 20);
console.log(price);
```

함수 반환

```
function sumPrices(price1, price2) {
    return price1 + price2;
};
var price = sumPrices(10, 20);
console.log(price);
```

함수 반환

```
function getSize(____) {
    ______;
};
var size = getSize(30, 32);
console.log(size);
```

과제입니다.

Quest 1.

변수를 선언하고 숫자 '29' 를 할당합니다.

변수를 하나 더 선언하고 1(순서) 에서 선언한 변수와 덧셈 대입을 이용해서 '31' 이 할당되도록 코드를 작성합니다.

변수를 하나 더 선언하고 1(순서) 에서 선언한 변수와 나머지 연산 대입을 이용해서 '2' 를 할당하도록 코드를 작성합니다.

만약 2(순서) 에서 생성한 변수를 3(순서) 에서 선언한 변수로 나누었을때 나머지가 '1' 인 경우 '**일치합니다.**' 라는 문자열이 **console 에 출력**되도록 작성합니다.

Quest 2.

‘a' 부터 'z' 까지 순서대로 **console 에 출력**되도록 작성합니다.

Quest 3.

'1' 부터 '50' 까지 순서대로 출력하고, 출력된 값의 **짝/홀수 여부**를 함께 표시합니다

console 에 출력 예시 : 1은 홀수!

HOMEWORK

제출

- github에 업로드

기한

- 2018. 3. 28. 23:00 까지