**JavaScript**

JavaScript

#### 기초

#### 관례

- **생성자 함수** 이름은 **대**문자로 시작
- Variable, Instance, Function, Method 이름은 소문자로 시작
- 작명은 CamelCase 이용.
- 문자열은 작은 따옴표('')를 이용한다.

#### 자료형

- string
- number
  - 정수, 소수 모두 포함.
- boolean
- function
- object
  - 배열도 포함.
- undefined

### string

- 더하기 연산자를 제외한 사칙 연산자는 숫자가 우선된다!
  - '23' + 12 = '2312'
  - '23' - 13 = 10

### bool

- true : 1 (2 이상도 true로 인식), false : 0
  - bool 끼리 비교가 가능

### undefined

선언하지 않거나 초기화 하지 않은 변수

#### 배열

- **대괄호**를 이용하거나 Array() Method 이용.
- 배열 요소의 type 상관 없이 전부 담을 수 있음.

```
var arr = [123, 'Hello', true, function () { }, {}, [111, 222]];
// var arr = Array(123, 'Hello', true, function () { }, {}, [111, 222]);
arr.push(arr.length); // 배열 길이 값을 배열 끝에 추가
```

#### 입력

### 문자열

```
var input = prompt(String message, String defaultValue);
```

### bool

```
var input = confirm();
```

#### 자료형 변환

### string to number

Number('number');

### type to bool

Boolean();

아래의 경우를 제외하고 모두 true

```
Boolean(0);
Boolean(NaN);
Boolean('');
Boolean(null);
Boolean(undefined);
```

#### 연산자

### 일치

- == : value 비교
- === : type 비교

```
//모두 true
'' == false
'' == 0
0 == false
'111' == 111
```

#### 제어문

### 짧은 조건문

좌변이 참이면 우변을 실행하지 않음.

```
// 실행되지 않음.
10 % 2 == 0 || alert("ok?");
```

### for in

### 요소 접근이 아닌 index 접근!!!

```
var arr = ['Hello', 273, 253.23, false];

/* 요소 접근이 아닌 index 접근이기 때문에 indexing 필요! */
for (var i in arr) {
    alert(typeof (arr[i]));
}
```

#### 함수

##### 익명함수와 선언적함수

익명함수는 아래와 같이 선언적함수와 의미가 같다. **단, 함수명이 같을 때 실행순서가 달라진다.**

```
var func = function () { }
function func () { }
```

- Case 1

```
        f();
```

```
// 마지막 함수가 실행되어 2 출력
      /*
      function f() { alert('1') };
      function f() { alert('2') };
      */
// 실행되지 않는다.
      /*
      var f = function () { alert('1') };
      var f = function () { alert('2') };
      */
```

- Case 2

```
      var f = function () { alert('1') };
      function f() { alert('2') };

//첫 번째 함수가 실행되므로 '1' 출력.
      f();
```

### 가변 인자

모든 함수는 내부에 **arguments**라는 배열 변수가 있고, parameter 유무에 상관없이 지원.

```
      function sumAll () {
          var sum = 0;
          for (var i in arguments)
              sum += arguments[i];
          return sum;
      }

      var result = sumAll(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

// 55 출력
      alert(result);
```

#### 내부 함수

모든 함수의 arguments로 인해 method override 불가능. 때문에 같은 이름의 함수 호출시 충돌 발생 가능.
☞ 함수 안에 함수를 사용.

#### 콜백 함수

함수형 프로그래밍. parameter로 함수 사용.

```
function fx(callback) {
    for (var i = 0 ; i < 3 ; i++)
        callback();
}

// Case 1
var c = function () { alert('!'); }
fx(c);

// Case 2
fx(function () {
    alert("?");
});
```

#### 함수를 반환하는 함수

#### Closure

자신의 Scope 밖에 있는 변수에 접근할 수 있는 함수

```
function fx(text) {
    var output = "안녕하세요. " + text
    return function () {
        alert(output);
    }
}

fx('Jake')(); //함수의 함수 호출
```

#### 객체

- Method : 객체의 속성 중 함수 자료형인 속성(Property)
- 객체 내부에서 "**this**" keyword를 생략할 수 없다!

#### Prototype

생성자 함수로 생성된 객체가 공통으로 가지는 공간. (static과 비슷한 개념.)

- 왜 사용하는가? 이점?
  - Memory Lick 방지!
    - 생성자 함수에 Property와 Method가 공존할 때, Property는 값이 다르지만 **Method의**

**Logic은 항상 같음**!

- 여러개의 객체 생성시 같은 Logic의 Method를 계속 생성하므로 **메모리 낭비**!
  - 그러므로 생성자 함수 생성시에는 Property만 넣는다.

◦ 동적 Method를 추가할 수 있음.

```
function Person (name, age) {
    this.name = name;
    this.age = age;
    /*
    this.showInfo = function () {
        var str = '';

        for (var key in this) {
            if (key == 'showInfo')
                break;

            str += key + '\t' + this[key] + '\n';
        }

        return str;
    };
    */
}

Person.prototype.showInfo = function () {
    var str = '';

    for (var key in this) {
        if (key == 'showInfo')
            break;

        str += key + '\t' + this[key] + '\n';
    }

    return str;
};


var person = new Person('Luke', 29);

alert(person.showInfo());
```

**Capsulation**

```
function Person (name, age) {
    this.name = name;
```

```
            var age1 = age;
            this.getAge = function () { return age1; }
            this.setAge = function (age)
            {
                if (age < 0)
                    throw '나이가 음수일 수 없습니다!';

                age1 = age;
            }
        }

        Person.prototype.showInfo = function () {
            return this.name + ' / ' + this.getAge();
        };


        var person = new Person('Luke', 29);

        person.setAge(-1);

        alert(person.showInfo());
```

**Inheritance**

```
    <script>
        function Person (name, age) {
            this.name = name;
            var age1 = age;
            this.getAge = function () { return age1; }
            this.setAge = function (age)
            {
                if (age < 0)
                    throw '나이가 음수일 수 없습니다!';

                age1 = age;
            }
        }

        Person.prototype.showInfo = function () {
            return this.name + ' / ' + this.getAge();
        };

    </script>

    <script>
        function Employee(name, age, id) {
            // base는 임의 지정 속성. 아무거나 상관 없음.
```

```
            // C#의 base, Java의 super와 같은 개념
            this.base = Person;
            this.base(name, age);

            this.id = id;
        }

        /* Prototype Method 초기화*/
        Employee.prototype = Person.prototype;
        // Prototype이 부모가 아닌 자기 자신의 생성자 가리키도록 설정. (위에서 부모를 가리키도록 했으니까..)
        Employee.prototype.constructor = Employee;

        var obj = new Employee("Luke", 28, 'A1234');
        alert(obj.showInfo());
        alert(obj.constructor); //누구의 생성자인지 확인
        alert(obj.id);
        alert(obj instanceof Person); //다형성 확인
    </script>
```

#### AJAX

##### 생성

- XML2JSON

```
<!DOCTYPE html>
<html>
<head>
    <title>XMLHttpRequest</title>
</head>
<body>
    <script>
        function createRequest() {
            try {
                return new XMLHttpRequest();
            } catch (e) {
                //IE 6 이하 버전

                var versions = [
                    'Msxml2.XMLHTTP.6.0',
                    'Msxml2.XMLHTTP.5.0',
                    'Msxml2.XMLHTTP.4.0',
                    'Msxml2.XMLHTTP.3.0',
                    'Msxml2.XMLHTTP',
                    'Microsoft.XMLHTTP'
                ];
```

```
                for (var i in versions) {
                    try {
                        return new ActiveXObject(versions[i]);
                    } catch (e2) {
                    }
                }
            }
        }


        var request = createRequest();
        request.onreadystatechange = function () {
            //request.readyState == 4 : 모든 Data 받음.
            if (request.readyState == 4 && request.status == 200) {
                document.body.innerHTML += request.responseText;
            }
        };
        request.open('GET', '/data.html', true);
        request.send();
    </script>
</body>
</html>
```