Web CSS

자바스크립트

• 명령형(imperative), 함수형(functional), 프로토타입 기반(prototype-based) 객체지향프로그래밍을 지원하는 멀티 패러다임프로그래밍 언어

• HTML로는 웹의 내용을 작성하고, CSS로는 웹을 디자인하며, 자바스크립트로는웹의 동작을 구현 가능 • 인터프리터 언어

• 타입을 명시하지 않음 • ECMAScript

ECMAScript

| 버전 | 출시  년도 | 특징 |
| --- | --- | --- |
| ES1 | 1997 | 초판 |
| ES2 | 1998 | ISO/IEC 16262 국제 표준과 동일한 규격을 적용 |
| ES3 | 1999 | 정규 표현식, try…catch 예외 처리 |
| ES5 | 2009 | HTML5와 함께 출현한 표준안. JSON, strict mode, 접근자 프로퍼티(getter, setter), 향상된 배열 조작 기능(forEach, map, filter, reduce, some, every) |
| ES6 (ECMAScript 2015) | 2015 | let, const, class, 화살표 함수, 템플릿 리터럴, 디스트럭처링 할당, spread 문법, rest 파라미터, Symbol, Promise, Map/Set, iterator/generator, module import/export |
| ES7 (ECMAScript 2016) | 2016 | 지수(\*\*) 연산자, Array.prototype.includes, String.prototype.includes |
| ES8 (ECMAScript 2017) | 2017 | async/await, Object 정적 메소드(Object.values, Object.entries, Object.getOwnPropertyDescriptors) |
| ES9 (ECMAScript 2018) | 2018 | Object Rest/Spread 프로퍼티 |

관련 기술

• Ajax(Asynchronous JavaScript and XML)

– 1999년, 자바스크립트를 이용해서 비동기적(Asynchronous) 으로 서버와 브라우저가 데이터를 교환할 수 있는통신기능 • jQuery

– 2006년, 다소 번거롭고 논란이 있던 DOM(Document Object Model)을 보다 쉽게 제어할 수 있게 됨 – 크로스 브라우징 이슈도 어느 정도 해결 • Node.js

– 2009년, 브라우저에서만 동작하던 자바스크립트를브라우저 이외의 환경에서 동작시킬 수 있는 자바스크립트실행환경 – 자바스크립트는 웹 브라우저를 벗어나 서버 사이드애플리케이션 개발에서도 사용되는 범용 프로그래밍 언어로발전

웹 브라우저

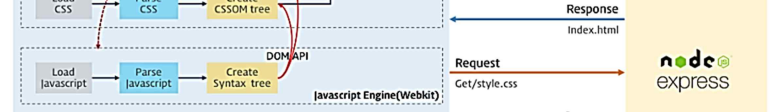
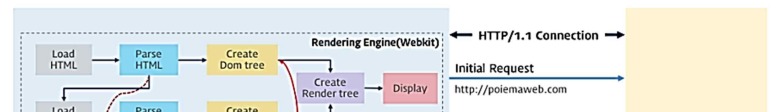
• 개발자 도구의 활용 – 크롬브라우저( V8자바스크립트엔진)

| 패널 | 설명 |
| --- | --- |
| Elements | 로딩된 웹 페이지의 DOM과 CSS를 편집하여 렌더링된 뷰를 확인해 볼다. 단, 편집한 내용이 저장되지는 않는다. 웹 페이지가 의도된 대로 렌더링되지 않았다면 이 패널을 확인하여 유용한 힌트를 얻을 수 있다. |
| Console | 로딩된 웹 페이지의 에러를 확인하거나 자바스크립트 소스코드에 포함시킨console.log 메소드의 결과를 확인해 볼 수 있다. |
| Sources | 로딩된 웹 페이지의 자바스크립트 코드를 디버깅할 수 있다. |
| Network | 로딩된 웹 페이지에 관련한 네트워크 요청(request) 정보와 퍼포먼스를할 수 있다. |
| Application | 웹 스토리지, 세션, 쿠키를 확인하고 관리할 수 있다. |

수 있

확인

동작원리



선언

• 내부 선언 방식 

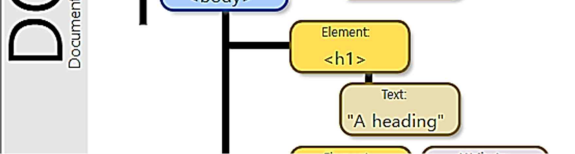
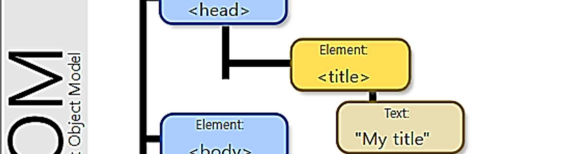
• 외부 선언 방식 • 위치

– <head>

– <body>

– both

DOM



DOM 요소 찾기

• HTML 태그 이름(tag name)을 이용한선택• 아이디(id)를 이용한 선택 • 클래스(class)를 이용한 선택 • name 속성(attribute)을 이용한 선택• CSS 선택자(selector)를 이용한 선택• HTML 객체 집합(object collection)을이용한선택

출력문

• innerHTML

– 해당 HTM요소 안에 출력 • **document.write()**

– **HTML문서에 출력**

• window.alert()

– alert 박스 안에 출력 • **console.log()**

– **브라우저 콘솔에 출력**

주석

• 주석(Comment)은 작성된 코드의의미를설명하기 위해 사용 • 한줄 주석

– // 다음에 작성

• 여러 줄 주석

– /\*과 \*/의 사이에 작성 • 주석은 해석기(parser)가 무시하며실행되지 않는다.

기본문법



변수(variable)

• 변하는 값(value)을 저장(할당)하고그저장된 값을 참조하기 위해 사용하는메모리공간

• 메모리 상의 주소(address)를 기억하는저장소

• 메모리 주소 접근을 위해 사람이이해할수 있는 언어로 지정한 식별자(identifier)

var x; // 변수의 선언

x = 6; // 정수값의 할당

값(value)

var str = 'Hello World';

• str이라는 이름의 변수를 선언하고문자열리터럴 ‘Hello World’를 값으로 할당

| 용어 | 의미 |
| --- | --- |
| 데이터 타입(Data Type) | 프로그래밍 언어에서 사용할 수 있는 값의 종류 |
| 변수(Variable) | 값이 저장된 메모리 공간의 주소를 가리키는식별자(identifier) |
| 리터럴(literal) | 소스코드 안에서 직접 만들어 낸 상수 값 자체를말하며 값을 구성하는 최소 단위 |

리터럴 표기법(literal notation)

10.50

1001 숫자리터럴

'Hello'

"World" 문자열 리터럴

true

false 불리언 리터럴

null 리터럴 null

undefined 리터럴 undefined 객체 리터럴 {name:’Lee’,gender:’male’} 배열 리터럴 [1, 2, 3]

/AB/

/ab/gi 정규표현식 리터럴

함수 리터럴 function() {}

변수 호이스팅(Variable Hoisting)

• var 선언문이나 function 선언문 등 모 든 선언문이 해당 Scope의 선두로 옮 겨진 것처럼 동작하는 특성

console.log(foo);

• 모든 선언문(var, let, const, function,

var foo = 123;

function\*, class)이 선언되기 이전에 참 console.log(foo);

조 가능

1. 선언 단계(Declaration phase)

{

var foo = 456; }

– 변수 객체(Variable Object)에 변수를

등록

– 스코프가 참조하는 대상 2. 초기화 단계(Initialization phase)

console.log(foo); function test(){ console.log(xoo); var xoo=1000;

– 등록된 변수를 메모리에 할당

– 변수는 undefined로 초기화

console.log(xoo);

3. 할당 단계(Assignment phase)

};

test();

console.log(foo,xoo);

– undefined로 초기화된 변수에 실제값 을 할당

스코프(scope)

• 참조 대상 식별자(identifier, 변수, 함수의 이름과 같이 어떤 대상을 다른 대상과 구분 하여 식별할 수 있는 유일한 이름)를 찾아

내기 위한 규칙

• 전역 스코프 (Global scope) – 코드 어디에서든지 참조 • 지역 스코프 (Local scope or Function level scope)

var global = 'global'; function foo() { var local = 'local'; console.log(global); console.log(local);

– 함수 코드 블록이 만든 스코프로 함수 자신

과 하위 함수에서만 참조 • 전역 변수 (Global variable)

}

foo();

console.log(global);

– 전역에서 선언된 변수이며 어디에든 참조 console.log(local);

• 지역 변수 (Local variable)

// Uncaught

ReferenceError: local

– 지역(함수) 내에서 선언된 변수이며 그 지역 is not defined

과 그 지역의 하부 지역에서만 참조

ES6 let const

• let

– 재선언 불가

– 블록 레벨 스코프

– 선언 이전에 사용 불가 • const

– 재선언 불가 / 재할당 불가 – 블록 레벨 스코프

scope

• 자바스크립트

– 함수 레벨 스코프(function-level scope)

• 블록 레벨 스코프(block-level scope) – ECMAScript 6에서 도입된 let keyword를 사용

**함수 레벨 스코프(Function-level scope)**

함수 내에서 선언된 변수는 함수 내에서만 유효하며 함수 외부에서는 참조할수 없다. 즉, 함수 내부에서 선언한 변수는 지역 변수이며 함수 외부에서선언한 변수는 모두 전역 변수이다.

**블록 레벨 스코프(Block-level scope)**

모든 코드 블록(함수, if 문, for 문, while 문, try/catch 문 등) 내에서 선언된변수는 코드 블록 내에서만 유효하며 코드 블록 외부에서는 참조할 수 없다. 즉, 코드 블록 내부에서 선언한 변수는 지역 변수이다.

데이터 타입

• 원시 타입 (primitive data type) – number

– string

– boolean

– null

– undefined

– symbol (New in ECMAScript 6)

• 객체 타입 (Object data type) – object

타입 변환

• 동적 타이핑

– 자바스크립트는 C나 Java외는 다르게변수를선언할 때 데이터 타입을 미리 지정하지않음. • 한 변수에 여러 번 대입은 가능하지만재선언은 무시

• 묵시적 타입 변환(implicit type conversion)• 명시적 타입 변환(explicit type conversion)– Number(), String(), Boolean(), Object(), parseInt(), parseFloat()

연산자(Operator)

• 피연산자(Operand)

| Operand | Operator | Operand |
| --- | --- | --- |
| 100 | + | 50 |

• 산술연산자(Arithmetic Operator)

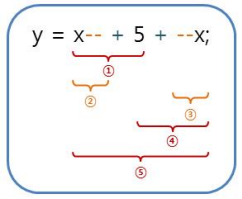
| Operator | Description |
| --- | --- |
| + | 더하기 Addition |
| - | 빼기 Subtraction |
| \* | 곱하기 Multiplication |
| \*\* | 거듭제곱 Exponentiation (ES2016) |
| / | 나누기 Division |
| % | 나머지 Modulus (Remainder) |

대입 연산자(assignment operator)

**Operator Example Same As** = x = y x = y += x += y x = x + y -= x -= y x = x - y \*= x \*= y x = x \* y /= x /= y x = x / y %= x %= y x = x % y <<= x <<= y x = x << y >>= x >>= y x = x >> y >>>= x >>>= y x = x >>> y &= x &= y x = x & y ^= x ^= y x = x ^ y |= x |= y x = x | y \*\*= x \*\*= y x = x \*\* y

증감 연산자(increment and decrement operator)

**증감 연산자 설명** ++x 먼저 피연산자의 값을 1 증가시킨 후에 해당 연산을 진행함. x++ 먼저 해당 연산을 수행하고 나서, 피연산자의 값을 1 증가시킴. --x 먼저 피연산자의 값을 1 감소시킨 후에 해당 연산을 진행함. x-- 먼저 해당 연산을 수행하고 나서, 피연산자의 값을 1 감소시킴.

10 

x y

9

8 23

x y

비교연산자(comparison operator)

| **비교 연산자** | **설명** |
| --- | --- |
| == | 왼쪽 피연산자와 오른쪽 피연산자의 값이 같으면 참을 반환함. |
| === | 왼쪽 피연산자와 오른쪽 피연산자의 값이 같고, 같은 타입이면 참을환함. |
| != | 왼쪽 피연산자와 오른쪽 피연산자의 값이 같지 않으면 참을 반환함. |
| !== | 왼쪽 피연산자와 오른쪽 피연산자의 값이 같지 않거나, 타입이 다르면참을 반환함. |
| > | 왼쪽 피연산자의 값이 오른쪽 피연산자의 값보다 크면 참을 반환함. |
| >= | 왼쪽 피연산자의 값이 오른쪽 피연산자의 값보다 크거나 같으면반환함. |
| < | 왼쪽 피연산자의 값이 오른쪽 피연산자의 값보다 작으면 참을 반환함. |
| <= | 왼쪽 피연산자의 값이 오른쪽 피연산자의 값보다 작거나 같으면반환함. |

반

참을

참을

논리 연산자(logical operator)

| **논리 연산자** | **설명** |
| --- | --- |
| && | 논리식이 모두 참이면 참을 반환함. (논리 AND 연산) |
| || | 논리식 중에서 하나라도 참이면 참을 반환함. (논리OR 연산) |
| ! | 논리식의 결과가 참이면 거짓을, 거짓이면 참을 반환함. (논리NOT 연산) |

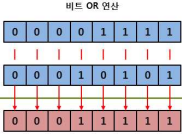
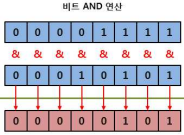
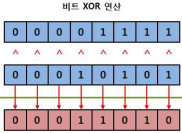
| **A** | **B** | **A && B** | **A || B** | **!A** |
| --- | --- | --- | --- | --- |
| true | true | true | true | false |
| true | false | false | true | false |
| false | true | false | true | true |
| false | false | false | false | true |

단축평가

| 단축 평가 표현식 | 평가 결과 |
| --- | --- |
| true || anything | true |
| false || anything | anything |
| true && anything | anything |
| false && anything | false |

비트 연산자(bitwise operator)

| **비트**  **연산자** | **설명** |
| --- | --- |
| & | 대응되는 비트가 모두 1이면 1을 반환함. (비트 AND 연산) |
| | | 대응되는 비트 중에서 하나라도 1이면 1을 반환함. (비트 OR 연산) |
| ^ | 대응되는 비트가 서로 다르면 1을 반환함. (비트 XOR 연산) |
| ~ | 비트를 1이면 0으로, 0이면 1로 반전시킴. (비트 NOT 연산) |
| << | 지정한 수만큼 비트를 전부 왼쪽으로 이동시킴. (left shift 연산) |
| >> | 부호를 유지하면서 지정한 수만큼 비트를 전부 오른쪽으로 이동시킴. (right shift 연산) |
| >>> | 지정한 수만큼 비트를 전부 오른쪽으로 이동시키며, 새로운 비트는 전부 0이 됨. |



삼항 연산자(ternary operator)

문법

표현식 ? 반환값1 : 반환값2

물음표(?) 앞의 표현식에 따라 결과값이 참이면 반환값1을 반환 결과값이 거짓이면 반환값2를 반환

타입의 판단

• typeof 연산자

– 피연산자의 타입을 반환

| **값** | **typeof 연산자의 결괏값** |
| --- | --- |
| 숫자, NaN | "number" |
| 문자열 | "string" |
| true, false | "boolean" |
| null | "object" |
| undefined | "undefined" |
| 함수 | "function" |
| 함수가 아닌 객체 | "object" |

• instanceof 연산자

– 피연산자인 객체가 특정 객체의 인스턴스인지 아닌지를확인

void 연산자

• void 연산자는 피연산자로 어떤타입의값이 오던지 상관없이 언제나 undefined 값만을 반환

<a href="javascript:void(0)">이 링크는 동작하지 않습니다.</a> <a href="javascript:void(document.body.style.backgroundColor='yellow')">이 링크도 동작하지 않지만, HTML 문서의 배경색을 바꿔줍니다. </a>

연산자의 우선순위(operator precedence)

우선순위표

| **우선**  **순위** | **연산자** | **설명** | **결합 방향** |
| --- | --- | --- | --- |
| **1** | () | 묶음(괄호) | - |
| **2** | . | 멤버 접근 | 왼쪽에서 오른쪽으로 |
|  | new | 인수 있는 객체 생성 | - |
| **3** | () | 함수 호출 | 왼쪽에서 오른쪽으로 |
|  | new | 인수 없는 객체 생성 | 오른쪽에서 왼쪽으로 |
| **4** | ++ | 후위 증가 연산자 | - |
|  | -- | 후위 감소 연산자 | - |
| **5** | ! | 논리 NOT 연산자 | 오른쪽에서 왼쪽으로 |
|  | ~ | 비트 NOT 연산자 | 오른쪽에서 왼쪽으로 |
|  | + | 양의 부호 (단항 연산자) | 오른쪽에서 왼쪽으로 |
|  | - | 음의 부호 (단항 연산자) | 오른쪽에서 왼쪽으로 |
|  | ++ | 전위 증가 연산자 | 오른쪽에서 왼쪽으로 |
|  | -- | 전위 감소 연산자 | 오른쪽에서 왼쪽으로 |
|  | typeof | 타입 반환 | 오른쪽에서 왼쪽으로 |
|  | void | undefined 반환 | 오른쪽에서 왼쪽으로 |
|  | delete | 프로퍼티의 제거 | 오른쪽에서 왼쪽으로 |
| **6** | \*\* | 거듭제곱 연산자 | 오른쪽에서 왼쪽으로 |
|  | \* | 곱셈 연산자 | 왼쪽에서 오른쪽으로 |
|  | / | 나눗셈 연산자 | 왼쪽에서 오른쪽으로 |
|  | % | 나머지 연산자 | 왼쪽에서 오른쪽으로 |
| **7** | + | 덧셈 연산자 (이항 연산자) | 왼쪽에서 오른쪽으로 |
|  | - | 뺄셈 연산자 (이항 연산자) | 왼쪽에서 오른쪽으로 |

| **우선**  **순위** | **연산자** | **설명** | **결합 방향** |
| --- | --- | --- | --- |
| **8** | << | 비트 왼쪽 시프트 연산자 | 왼쪽에서 오른쪽으로 |
|  | >> | 부호 비트를 확장하면서 비트 오른쪽 시프트 | 왼쪽에서 오른쪽으로 |
|  | >>> | 부호 비트를 확장하지 않고 비트 오른쪽 시프트 | 왼쪽에서 오른쪽으로 |
| **9** | < | 관계 연산자(보다 작은) | 왼쪽에서 오른쪽으로 |
|  | <= | 관계 연산자(보다 작거나 같은) | 왼쪽에서 오른쪽으로 |
|  | > | 관계 연산자(보다 큰) | 왼쪽에서 오른쪽으로 |
|  | >= | 관계 연산자(보다 크거나 같은) | 왼쪽에서 오른쪽으로 |
|  | instanceof | 인스턴스 여부 판단 | 왼쪽에서 오른쪽으로 |
| **10** | == | 동등 연산자 | 왼쪽에서 오른쪽으로 |
|  | === | 일치 연산자 | 왼쪽에서 오른쪽으로 |
|  | != | 부등 연산자 | 왼쪽에서 오른쪽으로 |
|  | !== | 불일치 연산자 | 왼쪽에서 오른쪽으로 |
| **11** | & | 비트 AND 연산자 | 왼쪽에서 오른쪽으로 |
| **12** | ^ | 비트 XOR 연산자 | 왼쪽에서 오른쪽으로 |
| **13** | | | 비트 OR 연산자 | 왼쪽에서 오른쪽으로 |
| **14** | && | 논리 AND 연산자 | 왼쪽에서 오른쪽으로 |
| **15** | || | 논리 OR 연산자 | 왼쪽에서 오른쪽으로 |
| **16** | ? : | 삼항 연산자 | 오른쪽에서 왼쪽으로 |
| **17** | = | 대입 연산자  (=, +=, -=, \*=, /=, %=, <<=, >>=, >>>=, &=, ^=, |=) | 오른쪽에서 왼쪽으로 |
| **18** | ... | 전개 | - |
| **19** | , | 쉼표 연산자 | 왼쪽에서 오른쪽으로 |

입력

• Prompt

– window.prompt(message,default); – 문자열입력

• confirm

– window.confirm(message);

– Boolean 값 입력

제어문(control flow statements)

• 프로그램의 순차적인 흐름을 제어해야할때 사용하는 실행문

• 조건문(conditional statements) – 프로그램 내에서 주어진 표현식의 결과에따라 별도의 명령을 수행하도록 제어하는실행문

• if 문, if / else 문, if / else if / else 문, switch 문• 반복문(iteration statements)

– 프로그램 내에서 똑같은 명령을 일정횟수만큼 반복하여 수행하도록 제어하는 실행문• while 문, do / while 문, for 문, for / in 문, for / of 문

조건문

• if / else if / else 

문법

if (조건1) {

표현식1의 결과가 참일 때 실행;

}

else if (조건2) {

조건2의 결과가 참일 때 실행;

}

else {

모든 조건이 거짓일 때 실행;

}

하나의 조건문 안에서 if 문과 else 문은 단 한 번만 사용될 수 있습니다. 하지만 else if 문은 여러 번 사용되어 다양한 조건을 설정할 수 있습니다.

조건문

• switch

**문법**

switch (조건) {

case 값1:

조건의 값이 값1일 때 실행;

break;

case 값2:

조건의 값이 값2일 때 실행;

break;

...

default:

조건 값이 어떠한 case 절에도 해당하지 않을 때 실행; break;

}

반복문(iteration statements) • while

**문법**

while (조건) {

조건의 결과가 참인 동안 반복적으로 실행; 

}

**예제**

*var* i = **1**;

while (i < **10**) {

document.write(i + "<br>");

i++;

}

무한 루프(infinite loop) 주의

반복문(iteration statements) • do / while

**문법**

do { 

조건의 결과가 참인 동안 반복적으로 실행;

} while (조건);

**예제**

*var* j = **1**;

do {

document.write(“j : “ + (j++) + ”<br>”);

} while (j > **3**);

반복문(iteration statements) • for 

**문법**

for (초기식; 표현식; 증감식) {

표현식의 결과가 참인 동안 반복적으로 실행; }

**예제**

for (*var* i = **1**; i < **10**; i++) {

document.write(i + "<br>");

}

break & continue

• break

– 반복문(for, for…in, for…of, while, do…while) 또는 switch 문의 코드 블록을 탈출• continue

– 반복문(for, for…in, for…of, while, do…while) 의 코드 블록 실행을 현 지점에서 중단하고반복문의 증감식으로 이동

– 반복문을 탈출하지 않음 • label

– label 문은 프로그램 내의 특정 영역을식별할수 있도록 해주는 식별자

반복문(iteration statements)

• for / in

– 해당 객체의 모든 열거할 수 있는 프로퍼티(enumerable properties)를 순회할 수 있도록해주는 반복문

**문법**

for (변수 in 객체) {

객체의 모든 열거할 수 있는 프로퍼티의 개수만큼 반복적으로 실행; }

반복문(iteration statements)

• for / of

– 반복할 수 있는 객체(iterable objects)를순회할 수 있도록 해주는 반복문 • Array, Map, Set, arguments 객체 등

**문법**

for (변수 of 객체) {

객체의 모든 열거할 수 있는 값의 개수만큼 반복적으로 실행; }

**예제**

*var* arr = [**3**, **4**, **5**];

for (*var* value of arr) {

document.write(value + " ");

}

배열(array)

• 이름과 인덱스로 참조되는 정렬된값의집합• 배열 요소(element)라고 하며, 배열에서의위치를 가리키는 숫자를 인덱스(index) – 배열 요소의 타입이 고정되어 있지 않음• 같은 배열에 있는 배열 요소끼리의 타입이 다른수있음– 배열 요소의 인덱스가 불연속 가능 • 특정 배열 요소가 빈요소일 수 있음



배열 생성

**문법**

1. *var* arr = [배열요소1, 배열요소2,...];

2. *var* arr = Array(배열요소1, 배열요소2,...);

3. *var* arr = new Array(배열요소1, 배열요소2,...);

var arrLit = [1, true, "JavaScript"];

// 배열 리터럴을 이용하는 방법

var arrObj = Array(1, true, "JavaScript");

// Array 객체의 생성자를 이용하는 방법

var arrNewObj = new Array(1, true, "JavaScript");

// new 연산자를 이용한 Array 객체 생성 방법

var arr = new Array(2);

console.log(arr);

arr = new Array(1, 2, 3);

console.log(arr); // [1, 2, 3]

배열 생성

• Array객체의 static함수 (of, from)

// of

const arr = Array.of(1, 2);

console.log(arr); // [ 1, 2 ]

// from

const arr = [1, 2, 3, 4, 5, 6];

const arr2 = Array.from(arr);

console.log(arr2); // [ 1, 2, 3, 4, 5, 6 ]

값의 참조/ 배열의 길이

**문법**

배열이름[인덱스]



**문법**

배열이름.length

배열 요소의 추가

**문법**

1. arr.push(추가할 요소);

2. arr[arr.length] = 추가할 요소; 3. arr[특정인덱스] = 추가할 요소;

• 배열의 홀(hole)

✔ 인덱스에 대응하는 배열 요소가 없는 부분 ✔ undefined 값을 가지는 요소

• 희소 배열(sparse array) <> 밀집 배열(dense array)

✔ 자바스크립트의 배열은 지금까지 살펴본 일반적인 의미의 배열과다르다.

✔ 배열의 요소를 위한 각각의 메모리 공간은 동일한 크기를 갖지않아도되며 연속적으로 이어져 있지 않을 수도 있다.

array function

• isArray

– 특정한 오브젝트가 배열인지 체크 – ES5 미지원시 instanceof

• indexOf

– 특정한 아이템의 위치를 찾을 때 • includes

– 배열안에 특정한 아이템이 있는지 체크

배열 함수

• push

– 배열 맨 뒤에 요소를 추가 (배열 자체를 수정하고 길이를 반환) • unshift

– 배열 맨 앞에 요소를 추가 (배열 자체를 수정하고 길이를 반환) • pop

– 배열 맨 뒤에 요소를 제거 (배열 자체를 수정하고 제거된 요소를반환) • shift

– 배열 맨 앞의 요소를 제거(배열 자체를 수정하고 제거된 요소를 반환) • splice

– 배열 특정 위치에 요소를 추가, 제거 (배열 자체를 수정하고 제거된요소를 배열 형태로 반환)

배열의 함수

• slice

– 잘라진 새로운 배열을 만듬 (배열 자체는 유지하고새로운배열을 반환) • concat

– 여러개의 배열을 연결 (배열 자체는 유지하고 새로운배열을 반환) • sort

– 배열의 요소를 적절하게 정렬(원본 배열을 직접 변경하며정렬된 배열을 반환) • reverse

– 배열 요소의 순서를 거꾸로 배치 (배열 자체를 수정하고변경된 요소를 배열 형태로 반환) • flat

– 중첩 배열을 하나의 배열로 평탄화 (배열 자체는 유지하고새로운 배열을 반환)

배열의 함수

• fill

– 특정한 값으로 배열을 채우기 (배열자체를수정하고 변경된 요소를 배열 형태로 반환) • join

– 배열을 문자열로 변환 (배열 자체는유지하고새로운 배열을 반환)

배열의 함수



배열의 반복/순회(iteration)

• 배열 내부의 값들은 사용할 때 반복문을통해 값들을 가져다 사용 • for 문

• for..in 문

• for..of 문

• forEach 함수

– break문 사용불가

다차원 배열

• 배열 안에 또 다른 배열이 포함되게하여차원을 표현

const array = [

[1, 2, 3],

[4, 5, 6],

[7, 8, 9]

];

배열 고차 함수(Higher Order function) • 고차 함수(Higher order function) – 함수를 인자로 전달받거나 함수를 결과로반환하는 함수 

• sort, forEach

• map

– 주어진 콜백 함수의 반환값(결과값)으로새로운배열을 생성하여 반환

• filter

– 배열을 순회하며 각 요소에 대하여 인자로주어진콜백함수의 실행 결과가 true인 배열 요소의값만을 추출한 새로운 배열을 반환

배열 고차 함수(Higher Order function) • reduce

– 배열을 순회하며 각 요소에 대하여 이전의콜백함수 실행 반환값을 전달하여 콜백함수를실행하고그 결과를 반환 

• some

– 배열 내 일부 요소가 콜백 함수의 테스트를통과하는지 확인하여 그 결과를 boolean으로반환• every

– 배열 내 모든 요소가 콜백함수의 테스트를통과하는지 확인하여 그 결과를 boolean으로 반환

배열 고차 함수(Higher Order function) • find

– 배열을 순회하며 각 요소에 대하여 인자로주어진 콜백함수를 실행하여 그 결과가참인첫번째 요소를 반환

– 참인 요소가 없으면 undefined • findIndex

– 배열을 순회하며 각 요소에 대하여 인자로주어진 콜백함수를 실행하여 그 결과가참인첫번째 요소의 인덱스를 반환 – 참인 요소가 존재하지 않는다면 -1을반환

함수(function)

• 하나의 특별한 목적의 작업을 수행하도록설계된 독립적인 블록

**문법**

**예제** *function* addNum(x, y) {

*function* 함수이름(매개변수1, 매개변수2,...) { 함수가 호출되었을 때 실행하는 실행문;

return x + y;

}

}

document.write(addNum(**2**, **3**));

• 자바스크립트 함수 – 자바스크립트에서는 함수도 하나의 타입(datatype)

– 함수를 변수에 대입하거나, 함수에 프로퍼티를지정하는 것도 가능

– 다른 함수 내에 중첩되어 정의 가능

함수

• 반환문(return)

– 함수는 return 문을 포함할 수 있음 – 함수내 실행된 결과를 전달 • 함수 호이스팅(hoisting)

– 함수내 선언된 모든 변수는 함수 전체에걸쳐유효

매개변수 / 인수

• 매개변수(parameter)

– 함수의 정의에서 전달받은 인수를 함수 내부로 전달하기 위해 사용하는변수 • 인수(argument)

– 함수가 호출될 때 함수로 값을 전달해주는 값 • ES6--------------

• 디폴트 매개변수(default parameter)

– 함수 호출시 명시된 인수를 미전달 시 사용할 기본 값 – 자바스크립트에서 매개변수 기본 값은 undefined

• 나머지 매개 변수

– 나머지 매개변수는 생략 접두사(...)를 사용하여 특정 위치의 인수부터마지막 인수까지를 한 번에 지정 가능

자바스크립트에서는 함수를 정의할 때는 매개변수의 타입을 따로 명시하지 않습니다.

함수를 호출할 때에도 인수(argument)로 전달된 값에 대해 어떠한 타입검사도 하지 않습니다.

함수를 호출할 때 함수의 정의보다 적은 수의 인수가 전달되더라도, 다른언어와는 달리 오류를 발생시키지 않습니다.

arguments 객체

• 함수의 정의보다 더 많은 수의 인수가전달되면, 매개변수에 대입되지 못한인수들은 참조불가능

• arguments 객체를 이용하 함수로전달된인수의 총 개수를 확인하거나, 각각의인수에도 접근가능

• arguments 객체는 함수가 호출될때전달된 인수를 배열의 형태로 저장

전역 함수

• 자바스크립트는 사용자의 편의를 위해다양한 기능의 여러 전역 함수를 미리 정의하여제공 • 전역 함수는 자바스크립트의 어떤타입의객체에서도 바로 사용 가능

eval()

문자열로 표현된 자바스크립트 코드를 실행하는함수

전달된 값이 유한한 수인지를 검사하여 그 결과를

반환 isFinite()

전달된 값이 NaN인지를 검사하여 그 결과를 반환isNaN()

문자열을 파싱하여 부동 소수점 수(floating point

number)로 반환 parseFloat()

문자열을 파싱하여 정수로 반환 parseInt()

decodeURI()

전역 함수

URI에서 주소를 표시하는 특수문자를 제외하고, 모든 문자를 이스케이프 시퀀스(escape sequences) 처리하여 부호화

URI에서 encodeURI() 함수에서 부호화하지 않은모

든 문자까지 포함하여 이스케이프 시퀀스 처리decodeURIComponent() encodeURI() 함수나 다른 방법으로 만들어진

URI(Uniform Resource Identifier)를 해독 encodeURI() encodeURIComponent() 함수나 다른 방법으로만

들어진 URI 컴포넌트를 해독 encodeURIComponent() 전달받은 문자열에서 특정 문자들을 16진법 이스케

이프 시퀀스 문자로 변환 escape() escape() 함수나 다른 방법으로 만들어진 16진법이

스케이프 시퀀스 문자를 원래의 문자로 변환 unescape() Number() 전달받은 객체의 값을 숫자로 반환 String() 전달받은 객체의 값을 문자열로 반환

함수의 형태

• 즉시 실행 함수(IIFE, Immediately Invoke Function Expression) – 함수의 정의와 동시에 실행되는 함수 • 내부함수(Inner function)

– 함수 내부에 정의된 함수

• 재귀 함수(Recusive function)

– 자기 자신을 호출하는 함수

– 재귀 함수는 반복 연산을 간단히 구현할 수 있다는장점이 있지만 무한 반복에 빠질 수 있고, stackoverflow에러를 발생시킬 수 있으므로 주의 • 콜백 함수(Callback function)

– 함수를 명시적으로 호출하는 방식이 아니라특정이벤트가 발생했을 때 시스템에 의해 호출되는함수

객체(object)

• 자바스크립트의 기본 타입(data type)은객체(object) – 자바스크립트에서는 숫자, 문자열, 불리언, undefined 타입을 제외한 모든 것이 객체• 숫자, 문자열, 불리언과 같은 원시 타입은 값이정해진객체로 취급되어, 객체로서의 특징도 함께 가짐• 이름(name)과 값(value)으로 구성된프로퍼티(property)의 정렬되지 않은 집합 • 프로퍼티의 값으로 함수가 올 수도있는데, 이러한 프로퍼티를 메소드(method)

**예제**

*var* cat ="나비";

*var* kitty = { name: "나비", family: "코리안 숏 헤어", age: **1**, weight: **0.1** }; cat;

kitty.name; //참조

*kitty[“name”];* //참조

값 참조

• 객체의 프로퍼티 참조

**문법**

객체이름.프로퍼티이름

또는

객체이름["프로퍼티이름"]

• 객체의 메소드 참조

**문법**

객체이름.메소드이름()

객체의 생성

• 자바와 같은 클래스 기반 객체 지향언어는 클래스를 사전에 정의하고 필요한시점에 new 연산자를 사용하여 인스턴스를생성하는 방식으로 객체를 생성

• 자바스크립트는 프로토타입 기반객체지향 언어로서 클래스라는 개념이없고별도의 객체 생성 방법이 존재

객체의 생성

• 객체 리터럴

– 중괄호({})를 사용하여 객체를 생성 – {} 내에 1개 이상의 프로퍼티를 기술하면 해당프로퍼티가 추가된 객체를 생성

– {} 내에 아무것도 기술하지 않으면 빈 객체가생성

**문법**

*var* 객체이름 = { 프로퍼티1이름 : 프로퍼티1의값, 프로퍼티2이름 : 프로퍼티2의값, ...

};

var person1 = {

name: 'Lee',

gender: 'male',

sayHello: function () { console.log(this.name); }

};

객체의 생성

• Object 생성자 함수(Constructor function)

– new 연산자와 Object 생성자 함수를 호출하여 빈 객체를 생성할수있다. 빈 객체 생성 이후 프로퍼티 또는 메소드를 추가하여 객체를완성하는 방법 • 생성자 함수

– new 키워드와 함께 객체를 생성하고 초기화하는 함수 • 생성자 함수 이름은 일반적으로 대문자로 시작

– 프로퍼티 또는 메소드명 앞에 기술한 this는 생성자 함수가 생성할인스턴스(instance) • 인스턴스(instance) : 생성자 함수를 통해 생성된 객체

– this에 연결(바인딩)되어 있는 프로퍼티와 메소드는 외부에서 참조가능– 생성자 함수 내에서 선언된 일반 변수는 외부에서 참조 불가능

function Person(name, gender) {

var person = new Object(); person.name = 'Lee'; person.gender = 'male';

this.name = name; this.gender = gender; this.sayHello = function(){

person.sayHello = function () {

console.log(this.name);

console.log(this.name); };

};

}

참고

• 자바스크립트는 Object 생성자 함수이외에도 String, Number, Boolean, Array, Date, RegExp 등의 빌트인 생성자함수를제공

– 일반 함수와 생성자 함수를 구분하기위해생성자 함수의 이름은 파스칼 케이스 (PascalCase)/ 카멜 표기법(Camel case)을사용하는 것이 일반적

예약어

• abstract arguments boolean break byte • case catch char class\* const

• continue debugger default delete do • double else enum\* eval export\*

• extends\* false final finally float

• for function goto if implements

• import\* in instanceof int interface

• let long native new null

• package private protected public return • short static super\* switch synchronized • this throw throws transient true

• try typeof var void volatile

• while with yield

• // \*는 ES6에서 추가된 예약어

프로토타입(prototype)

• 자바스크립트의 모든 객체는 최소한 하나 이상의 다른객체로부터 상속을 받으며, 이때 상속되는 정보를 제공하는객체• 상속(inheritance)

– 새로운 클래스에서 기존 클래스의 모든 프로퍼티와 메소드를사용할 수 있는 것 – 상속을 통해 새로운 프로그램의 요구에 맞게 기존 클래스를수정하여 재사용 – 클래스 간의 종속 관계를 형성함으로써 객체의 관계를 조직화– 추상화, 캡슐화와 더불어 객체 지향 프로그래밍을 구성하는중요한 특징 중 하나

• 자바스크립트는 프로토타입 기반(prototype-based)의객체지향 언어 – C#이나 C++과 같은 클래스 기반(class-based)의 객체 지향언어– 자바스크립트에서는 현재 존재하고 있는 객체를 프로토타입으로사용하여, 해당 객체를 복제하여 재사용하는 것을 상속이라고정의 • 프로토타입 체인(prototype chain)

객체 프로퍼티의 참조 및삭제• 참조

**문법**

객체이름.프로퍼티이름

또는

객체이름["프로퍼티이름"]

• 삭제

**문법**

delete 객체이름.프로퍼티이름;

객체 프로퍼티의 순회• for-in 문

– 객체의 문자열 키(key)를 순회하기 위한문법• for-of 문

– 배열의 값을 순회하기 위한 문법

객체 레퍼런스(object reference) • pass by reference

// Pass-by-reference

var foo = { val: 10 }

var bar = foo;

console.log(foo.val, bar.val); // 10 10 console.log(foo === bar); // true

bar.val = 20;

console.log(foo.val, bar.val); // 20 20 console.log(foo === bar); // true



객체 메소드

• 모든 자바스크립트 객체는 Object 객체와 Object.prototype 객체의모든 프로퍼티와 메소드를 상속 • hasOwnProperty()

– 특정 프로퍼티가 해당 객체에 존재하는지를 검사 • propertyIsEnumerable()

– 특정 프로퍼티가 해당 객체에 존재하고, 열거할 수 있는 프로퍼티인지를 검사 – Object.defineProperty() 메소드는 객체에 프로퍼티와 추가하고 그속성까지 설정하는 메소드 (ECMAScript 5) • isPrototypeOf()

– 특정 객체의 프로토타입 체인에 현재 객체가 존재하는지를 검사• isExtensible()

– 객체에 새로운 프로퍼티를 추가할 수 있는지 여부를 반환 – preventExtensions() 메소드를 사용하여 해당 객체에 새로운 프로퍼티를추가할 수 없도록 설정 • toString()

– 메소드를 호출한 객체의 값을 문자열로 반환

• valueOf()

– 특정 객체의 원시 타입(primitive type)의 값을 반환

객체의 분류



네이티브 객체

• Native objects or Built-in objects • ECMAScript 명세에 정의된 객체를말하며애플리케이션 전역의 공통 기능을제공• 네이티브 객체는 애플리케이션의환경과관계없이 언제나 사용 가능 • Object, String, Number, Function, Array, RegExp, Date, Math와 같은 객체생성에관계가 있는 함수 객체와 메소드로구성

호스트 객체(Host Object)

• 브라우저 환경에서 제공하는 window, XmlHttpRequest, HTMLElement 등의 DOM 노드객체와 같이 호스트 환경에 정의된 객체• BOM(Browser Object Model)

• DOM(Document Object Model)

