HTML, CSS, JS

524730
2022년 봄학기
3/9/2022
박경신

Client-Side Web Technology

- HTML 웹 페이지의 구조와 내용
- CSS 웹페이지 스타일
- JS 웹 페이지의 동적 변경 및 응용프로그램 작성
- Tutorial
  - HTML https://www.w3schools.com/html/default.asp
  - CSS https://www.w3schools.com/css/default.asp
  - JS https://www.w3schools.com/js/default.asp

https://www.w3schools.com/html/default.asp

HTML (Hyper-Text Markup Language)

- HTML (Hyper-Text Markup Language)
  - WWW (World-Wide Web) 문서를 작성하는 Markup 언어
  - HTML은 여러 태그(tag)들로 구성되어 있으며, 각 태그들을 사용하여 원하는 형태의 문서를 만들 수 있음
- HTML 문서의 기본형식

```
<!DOCTYPE html>
<html>
  <head>
    <title>My First Web Page</title><!-- 헤더 타이틀 -->
  </head>
  <body>
    <h1>Heading</h1><!-- 본문 -->
    <p>Hello World!</p> <!-- 본문 -->
  </body>
</html>
```

HTML (Hyper-Text Markup Language)

- HTML 구성요소
  - 태그(Tag)

```
<tag attr="value">
    content.....
</tag>
```

  - 요소(Element) – HTML에서 시작태그와 종료태그로 이루어진 명령
  - 속성(Attribute) – 요소의 시작 태그 안에서 사용되는 좀 더 구체화된 명령
  - 변수(Arguments) – 속성에 값

HTML (Hyper-Text Markup Language)

- HTML 문서 기본 태그
  - <html>
  - <head> 헤드
  - <title> 타이틀
  - <body> 본문
  - <h1>… <h6> 제목(headings)
  - 단락(paragraph)

    ** 가운데로 정렬한 문단 **
  - <hr> 수평줄(horizontal rule)
  -
 줄 바꾸기(line break)

HTML (Hyper-Text Markup Language)

- HTML 텍스트 효과 태그
  - <b>, 진하게 (bold text formatting)
  - <i>, 이탤릭 (italics text formatting)
  - <small> 한 단계 작은 문자
  - <sup>, <sub> 윗/아래 첨자 (super/subscript text formatting)
  - <mark> 하이라이팅
- HTML 미디어 태그
  - <img>, <video>, <audio> images, videos, audio

    <img src="src.jpg" width="100" height="200" align="top">

HTML (Hyper-Text Markup Language)

- HTML 블록과 링크 태그
  - <pre> 포맷 그대로 출력
  - 블록 형식으로 공간 분할 (division)
  - <section> 독립적인 섹션(section)
  - <article> 내용이 완전히 독립적으로 구성할 수 있는 요소 정의
  - 인라인 형식으로 공간 분할
  - <a> 하이퍼링크(link/anchor) - 현재 HTML 페이지에서 다른 HTML 연결하는 링크를 만들고자 할 때 사용

    네이버
  - <iframe> 인라인 프레임 - 현재 HTML 페이지 내에 내장 윈도우를 만들고 다른 HTML 페이지를 출력할 때 사용
    - src : 출력할 웹 페이지의 URL 주소
    - srcdoc : HTML 문서 텍스트
    - name : 윈도우 이름 | 다른 웹 페이지에서 target 속성 값으로 사용
    - width : 프레임의 폭
    - height : 프레임의 높이

HTML (Hyper-Text Markup Language)

- HTML 리스트 태그
  - , <ol>, <li> unordered list, ordered list, list element

```
<ul>
    <li> data1 </li>
    <li> data2 </li>
</ul>
```

```
<ol>
    <li> data1 </li>
    <li> data2 </li>
</ol>
```

- HTML 테이블 태그
  - <table>, <th>, <tr>, <td> table, table head, table column, table data cell
  - <thead>, <tbody>, <tfoot> head cell group, data cell group, foot cell group

```
<table border="1" align="center">
    <tr>  <td> 1 x 1 </td>  <td> 1 x 2 </td>  </tr>
    <tr>  <td> 2 x 1 </td>  <td> 2 x 2 </td>  </tr>
</table>
```

HTML (Hyper-Text Markup Language)

- HTML 프레임 태그
  - <frameset>, <frame> frame

```
<frameset row="100,*">
    <frame src="top_menu.html">
    <frameset cols="200,*">
        <frame src="left_menu.html">
        <frame src="contents.html">
    </frameset>
</frameset>
```

HTML (Hyper-Text Markup Language)

- HTML 메타 태그
  - <base> URL과 웹 페이지가 출력될 윈도우를 지정하기 위해 사용
  - <link> CSS 파일을 불러오는 태그 - <head>에서만 사용
  - <script> 자바스크립트 코드를 담는 태그
  - <style> CSS 스타일 시트를 담는 태그
  - <meta> 웹 페이지의 저작자, 문자 인코딩 방식, 문서 내용 등 다양한 메타 데이터를 표현하기 위해 사용 - name과 content의 속성 쌍으로 구성

http://www.w3schools.com/css/default.asp

CSS (Cascading Style Sheet)

- CSS (Cascading Style Sheet)
  - HTML이 웹페이지의 정보를 표현하는 언어이고, CSS는 HTML을 보기 좋게 디자인하는 역할을 하는 언어
  - CSS는 문서의 콘텐츠와 레이아웃, 글꼴 및 시각적 요소들로 표현되는 **문서의 디자인을 분리하기 위한** 목적으로 만들어짐
  - CSS 명세는 WWW Consortium에서 관리, CSS1, CSS2, CSS3가 정의되어 있음
  - 하나의 규칙으로 여러 HTML 요소와 HTML 문서를 제어할 수 있음

CSS (Cascading Style Sheet)

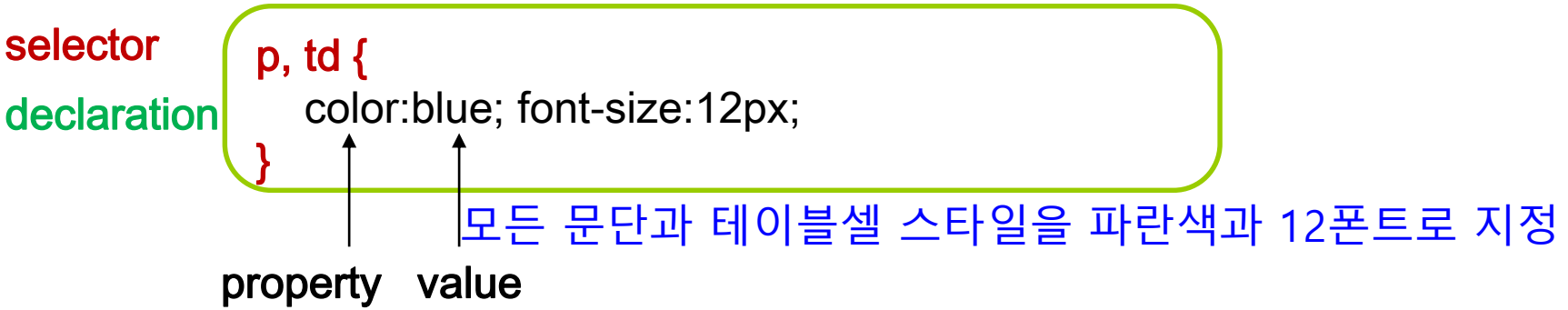
- CSS 상속
  - CSS3 스타일은 부모 요소로부터 상속

```
<p style="color:blue">Parent /*부모요소 파란색 스타일 */
  <em style="font-size:12px">Child</em> /* 자식요소 파란색 12폰트 스타일 */
</p>
```

- CSS cascading
  - 이 4가지 스타일 시트가 태그에 동시에 적용될 때 스타일이 합쳐져서 적용되는 것
    - 브라우저 디폴트
    - .css 스타일 시트 파일
    - <style></style> 태그
    - style 속성에 작성된 스타일
- CSS overriding
  - 동일한 CSS3 프로퍼티에 서로 다른 값 설정이 충돌 시 **우선순위가 높은** 스타일을 적용하는 규칙임

CSS (Cascading Style Sheet)

- CSS 기본 문법
  - CSS 규칙은 **선택자(selector)**와 선언부(declaration)로 나뉘며, 선언부는 다시 **속성(property)**과 **속성값(value)**으로 나뉨



  - 선택자는 스타일하고 싶은 **HTML tag**나 부여한 **class** 혹은 **ID**가 위치
  - 여러 선택자를 사용할 경우 **콤마(,)**로 구분
  - 선언부에 여러 개의 속성과 속성값이 있을때 **세미콜론(;)**으로 구분
  - 각각의 선언은 속성과 속성값을 **콜론(:)**으로 구분
  - CSS 주석은 **/* */**로 둘러 쌈

CSS (Cascading Style Sheet)

- HTML에 CSS 적용 방법
  - HTML 태그 내에 스타일 지정 (Inline Styles)

```
<p style="color:gray">  이 문단의 색은 회색으로  </p>
```

  - <head>안에 내부 스타일 시트 (Internal Style Sheet)

```
<head>
  <style type="text/css">
  body { font-size:9pt;}
  </style>
</head>
```

CSS (Cascading Style Sheet)

- HTML에서 CSS 적용 방법
  - <head>안에 link를 이용한 외부 스타일 시트 (External Style Sheet)

```
<head>
   <link href="style.css" rel="stylesheet" type="text/css">
</head>
```

  - <head>안에 @import를 이용한 외부 스타일 시트

```
<head>
   @import url("/css/style.css");
</head>
```

  - xml-stylesheet 처리 명령으로 외부 스타일 시트

```
<?xml-stylesheet type="text/css" href="/css/style.css" ?>
```

CSS (Cascading Style Sheet)

- CSS 적용 우선순위
  - CSS는 기본적으로 선언된 순서에 따라 적용되지만, 각종 선택자와 삽입 위치에 따라 우선순위가 달라질 수 있음
  - 우선순위를 최대한 단순하게 유지해야함
  - 선택자 우선순위
    - !important > 인라인 스타일 > 아이디 선택자 > 클래스/속성/가상 선택자 > 태그 선택자 > 전체 선택자
  - 삽입 위치 우선순위
    1. <head> 안의 style 요소
    2. <style> 안의 @import 문
    3. <link>로 연결된 CSS 파일
    4. <link>로 연결된 CSS 파일 안의 @import 문
    5. 최종 사용자가 연결한 CSS 파일
    6. 브라우저의 기본 스타일 시트

CSS (Cascading Style Sheet)

- CSS 선택자
  - tag name selector 태그 이름이 선택자로 사용

```
p, td {   color:blue; font-size:12px;   }
```

  - **class selector** 선택자 이름 앞에 '.' 기호를 붙인 경우 class 속성이 같은 모든 태그에 적용

```
.select { color: red }
<li class="select">Hello</li>
```

  - **id selector** 선택자 이름 앞에 '#' 기호를 붙인 경우 태그의 id 속성으로만 지정 가능

```
#content { width: 600px; float: left; background: green;}
<div id="content">
<h1>Content</h1>
</div>
```

CSS (Cascading Style Sheet)

- CSS 선택자
  - child selector 자식 선택자는 부모 자식 관계인 두 선택자를 '>' 기호로 조합

```
div > div > strong { color : red; } /* <div>의 자식 <div>의 자식 <strong>에 적용*/
```

  - descendent selector 자손 선택자는 자손 관계인 2개 이상의 태그를 나열한 상태

```
ul strong { color : green; } /* <ul>의 자손 <strong>에 적용 */
```

  - universal selector 전체 선택자는 '*' 기호를 사용하여 웹 페이지의 모든 HTML 태그에 적용

```
* { color : blue; } /* 모두 파란색 적용 */
```

  - attribute selector 속성 선택자는 HTML 태그의 특정 속성에 대해 값이 일치하는 태그에만 스타일을 적용

```
input[type=text] { color : blue; } /* type 속성값이 "text"인 <input>
태그에 적용 */
```

http://www.w3schools.com/cssref/css_selectors.asp

CSS (Cascading Style Sheet)

- CSS 선택자(Selectors)

선택자	예시	예시설명
.class	.intro	**class="intro"**인 모든 elements를 선택
#id	#firstname	**id="firstname"**인 element를 선택
*	*	**모든 elements** 선택
element	p	**모든 <p>** elements 선택
element,element	div, p	모든 ** 와 <p>** elements 선택
element element	div p	모든 ** 안에 모든 <p>** elements 선택
element>element	div>p	모든 ** 의 한단계 아래 자식들 <p>** 선택
element+element	div+p	모든 ** 바로 다음에 나오는 <p>** 선택
element.class	p.summary	**class="summary"인 <p>** element 선택
element#id	p#index	**id="index"인 <p>** element를 선택

http://www.w3schools.com/cssref/css_selectors.asp

CSS (Cascading Style Sheet)

- CSS 선택자(Selectors)

선택자	예시	예시설명
[attribute]	[alt]	alt attribute를 가진 모든 elements 선택
[attribute=value]	[alt="dog"]	alt="dog"인 모든 elements 선택
[attribute~=value]	[alt~="dog"]	alt attribute로 dog를 가진 모든 elements 선택
[attribute|=value]	[lang|=en]	lang attribute로 "en"로 시작하는 elements 선택
:active	a:active	active link를 선택
:focus	input:focus	focus된 입력을 선택
:lang	p::lang(kr)	lang attribute가 "kr"인 <p> 선택
::after	p::after	각 <p> 다음에 내용을 삽입
::before	p::before	각 <p> 전에 내용을 삽입
::first-letter	p::first-letter	각 <p>에 첫번째 문자를 선택
::first-line	p::first-line	각 <p>에 첫번째 줄을 선택

CSS (Cascading Style Sheet)

- CSS class 선택자
  - 클래스(class) 선택자는 여러 개를 사용할 수 있음

```
<style type="text/css">
    <!--
    p.red { color:red }
    //-->
</style>
<p> 일반적인 문단 </p>
<p class="red"> red라는 이름의 클래스가 지정된 문단 </p>
```

```
<style type="text/css">
    <!--
    .red { color:red }
    //-->
</style>
<h3 class="red"> 소제목에도 red 클래스를 지정 </h3>
<p class="red"> red라는 이름의 클래스가 지정된 문단 </p>
```

CSS (Cascading Style Sheet)

- CSS 아이디(id) 선택자
  - 아이디(id) 선택자는 고유성을 가지므로 한번만 사용할 수 있음

```
<style type="text/css">
   <!--
   #red { color:red }
   //-->
</style>
<p> 일반적인 문단 </p>
<p id="red"> 이 문단 id는 red </p>
```

CSS (Cascading Style Sheet)

HTML element	CSS rule

<p>	p {
	  font-size:14px;
	}

<p class="bp">	.bp {
	  color:gray;
	}

<p id="headline">	#headline {
	  font-size:20px;
	}

    …
 	
	.intro h1 {
	  font-size:18px;
	}

https://www.w3schools.com/js/default.asp

JS (JavaScript)

- JavaScript
  - JavaScript는 HTML과 CSS로 만들어진 웹페이지를 동적으로, 프로그래밍적으로 제어하기 위해 만들어진 언어
  - 최근 자바스크립트는 웹페이지 스크립팅(DOM) 뿐만 아니라, 서버측 스크립팅에서도 사용(**node.js**)되고, 구글 크롬 웹브라우저 플러그인, 구글 스크립트, PDF, 각종 데스크탑 위젯에서도 사용
  - **jQuery**는 자바스크립트 라이브러리
- **<script>** 태그 안에 자바스크립트 정의

```
<script>
    document.getElementById("demo").innerHTML = “Hello JavaScript";
</script>
```

자료형

- 자바스크립트의 자료형
  - Primitive Data Type
    - boolean, undefined, number, string, symbol(immutable value)
  - Complex Data Type
    - function, object, array, null
- 자바스크립트는 동적 언어
  - Dynamic Type Binding 변수에 값을 넣을 때 변수의 타입이 결정

```
var foo; // undefined
foo = 5; // number
foo = “test”; // string
foo = true;// boolean
typeof {name:‘Park', age:20} // Returns "object"
typeof [1,2,3,4]             // Returns "object" (not "array”)
typeof null                  // Returns "object"
typeof function myFunc(){}   // Returns "function"
```

함수

- 함수(function)
  - 자바스크립트에서 함수는 **first-class object(일급 객체)**임 - 즉 **변수**나 데이터 구조 안에 담을 수 있으며 **인자**로 전달할 수 있고 **반환값**으로 사용할 수 있으며, **런타임에 생성**할 수 있음
- 함수의 선언 (function declaration)

function 함수명([인자.. *[,인자]*) {
    // 코드
    return 반환값;
}

```
<script>
    var a = multiply(2, 3, 4);    // Function is called
    function multiply(x, y, z) { // function declaration
        return x * y * z;
    }
    alert(a);
</script>
```

함수

- 함수 표현 (function expression)
  - 함수 표현은 function literal 혹은 anonymous function(익명함수)
  - 실행 가능한 코드로, 변수나 데이터 구조에 할당되어지고 있음

```
<script>
    var funcName = function() { // function expression
        alert('function A');
    }
    funcName(); // function is called

    // function expression
    var fullName = function(first, last) { return first + " " + last; };
    alert(fullName('K', 'Park')); // function is called
</script>
```

함수

- **클로저 (Closure) 자유 변수(free variable)**에 엮여있는 함수
  - 자바스크립트 내에서는 함수의 생명주기는 끝났지만, 함수 내의 변수를 내부함수가 참조하고 있기 때문에 유지되어 접근할 수 있는 함수를 클로저라 부름
  - 함수 안에 있는 내부함수에서 외부함수에 선언된 변수를 사용한다면 그 내부함수는 클로저임

```
<script>
var hello = 'Hello '; // global variable
function sayHello(name) {
  var text = hello + name; // local variable
  return function() { alert(text)  }; // 내부함수에서 외부함수에 선언된 text 사용
}
// sayHello 함수 내에 text 지역변수가 hello1(),hello2() 함수호출 후에도 살아있음
var hello1 = sayHello('K');
var hello2 = sayHello('Park');
hello1(); // Hello K
hello2(); // Hello Park
</script>
```

함수

- () operator는 함수를 호출

15.5555555557

```
<p id="demo1"></p>
<script>
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}
document.getElementById("demo1").innerHTML = toCelsius(60);
</script>
```

- 함수를 () 사용하지 않고 호출시

function toCelsius(fahrenheit) { return (5/9) * (fahrenheit-32); }

```
<p id="demo2"></p>
<script>
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}
document.getElementById("demo2").innerHTML = toCelsius;
</script>
```

배열

- 배열의 선언

```
<script>
  var array = ['사과', '바나나', '망고', '딸기'];
</script>
```

인덱스 (index)	요소
0	사과
1	바나나
2	망고
3	딸기

- 배열의 구성 : 인덱스와 요소
  - 배열 요소를 사용하려면 배열 이름 뒤에 인덱스로 접근
    - array[0] → 사과
    - array[2] → 망고
- 객체와 배열
  - 배열은 **객체를 기반**으로 함
  - 배열은 요소에 인덱스로 접근/객체는 요소에 키로 접근

객체

- 객체(Object)의 생성

```
<script>
  var product = {
    제품명: '7D 건조 망고',
    유형: '당절임',
    성분: '망고, 설탕, 메타중아황산나트륨, 치자황색소',
    원산지: '필리핀'
  };
</script>
```

- 표로 나타낸 객체

키(key)	값(value)
제품명	7D 건조 망고
유형	당절임
성분	망고, 설탕, 메타중아황산나트륨, 치자황색소
원산지	필리핀

객체

- 객체의 사용
  - 객체 뒤에 대괄호를 사용하고 키를 표시하여 요소에 접근
  - product['제품명'] → '7D 건조 망고'
  - product['유형'] → '당절임'
  - product['성분'] → '망고, 설탕, 메타중아황산나트륨, 치자황색소'
  - product['원산지'] → '필리핀'
- 대괄호를 사용하지 않고 일반적으로 사용하는 방법
  - product.제품명 → '7D 건조 망고'
  - product.유형 → '당절임'
  - product.성분 → '망고, 설탕, 메타중아황산나트륨, 치자황색소'
  - product.원산지 → '필리핀'

객체

- 객체의 사용
  - 객체의 키
    - 식별자 또는 문자열 모두 사용 가능

```
<script>
  var object = {
    'with space': 273,
    'with ~!@#$%^&*()_+': 52
  };
</script>
```

  - 식별자가 아닌 문자를 키로 사용하려면 대괄호를 사용해야만 함
    - object['with space'] → 273
    - object['with ~!@#$%^&*()_+'] → 52

객체의 속성

- 속성
  - 요소: 배열 내부에 있는 값
  - 속성: 객체 내부에 있는 값

```
<script>
  var object = {
    number: 273,
    string: 'RintlanTta',
    boolean: true,
    array: [52, 273, 103, 32],
    method: function() {}
  };
</script>
```

객체의 메서드

- 메서드
  - 객체의 속성 중 함수 자료형
  - 속성과 메서드의 구분
    - 객체 person: name 속성, eat 속성
    - eat 속성은 함수 자료형이므로 eat() 메서드라 부름

```
<script>
  var person = {
    name: '나다라',
    eat: function(food) {}
  };
  person.eat(); // person의 eat 메서드 호출
</script>
```

객체의 메서드

- 메서드
  - this 키워드
    - 자기 자신이 가진 속성을 출력하고 싶을 때, 자신이 가진 속성임을 표시하는 방법

```
<script>
  var person = {
    name: '나다라',
    eat: function(food) {
      alert(this.name + '이 ' + food +'을/를 먹습니다');
    }
  };
  person.eat('밥'); // person의 eat 메서드 호출
</script>
```

객체와 반복문

- 객체와 반복문
  - 객체는 단순 for 반복문으로 객체의 속성을 살펴보는 것이 불가능
  - 객체의 속성을 모두 살펴보려면 **for in** 반복문 사용

```
<script>
  var product = {
    name: 'Microsoft Visual Studio',
    price: '15,000,000원',
    language: '한국어',
    supportedOS: 'Win32/64',
    subscription: true
  };

  // forin-loop
  var output = '';
  for (var key in product) {
    output += '·' + key + ': ' + product[key] + '₩n';
  }
  alert(output);
</script>
```

객체 관련 키워드

- 객체 관련 키워드: in, with 키워드
- in 키워드
  - 속성이 객체에 포함되어 있는 지 확인

```
<script>
  var output = '';
  var student = {
    이름: '마바사',
    국어: 92, 수학: 98, 영어: 96, 과학: 98
  };

  output += "이름 속성: " + ('이름' in student) + '₩n';
  output += "성별 속성: " + ('성별' in student);
  alert(output);
</script>
```

이름 속성: true
성별 속성: false

객체 관련 키워드

- with 키워드
  - 복잡하게 사용하는 코드를 짧게 줄여줌

```
<script>
  var output = '';
  var student = { ... }; // 이전 내용
  output += '이름: ' + student.이름 + '₩n';
  output += '국어: ' + student.국어 + '₩n';
  output += '수학: ' + student.수학 + '₩n';
  output += '영어: ' + student.영어 + '₩n';
  output += '과학: ' + student.과학 + '₩n';
  output += '총점: ' + (student.국어 + student.수학
                    + student.영어 + student.과학);
  alert(output);
</script>
```

객체 관련 키워드

- with 키워드 사용

```
<script>
  var output = '';
  var student = { ... }; // 이전 내용
  with (student) {
    output += '이름: ' + 이름 + '₩n';
    output += '국어: ' + 국어 + '₩n';
    output += '수학: ' + 수학 + '₩n';
    output += '영어: ' + 영어 + '₩n';
    output += '과학: ' + 과학 + '₩n';
    output += '총점: ' + (국어 + 수학 + 영어 + 과학);
  }
  alert(output);
</script>
```

객체 속성 추가/제거

- 객체 생성 이후 동적으로 속성을 추가/제거
- 속성 추가
  - 빈 객체 생성

```
<script>
   var student = {};
</script>
```

  - 동적으로(실행 중에) 속성 추가

```
<script>
  var student = {}; // empty object
  student.이름 = '자차카 ' ;
  student.취미 =  ' 기타 ' ;
  student.특기 =  ' 프로그래밍 ' ;
  student.장래희망 =  ' 공학자';
</script>
```

객체 속성 추가/제거

- 동적으로 메소드 추가

```
<script>
    // 이전 student 객체 메소드 추가 코드
    student.toString = function () {
      var output = '';
      for (var key in this) {
        // toString() 메서드는 출력 안함
        if (key != 'toString') {
          output += key + '₩t' + this[key] + '₩n';
        }
      }
      return output;
    };
    alert(student.toString());
</script>
```

객체 속성 추가/제거

- 속성 제거
  - delete 키워드 사용
  - delete 키워드 뒤에 삭제하고자 하는 객체의 속성을 입력
  - 객체의 속성을 입력할 때는 typeof 키워드처럼 괄호를 사용해도 되고 안해도 됨

```
<script>
  // 변수를 선언합니다
  var student = {};
  student.특기 = '프로그래밍';
  student.장래희망 = '공학자';
  delete student.특기;
</script>
```

객체와 배열을 사용한 데이터 관리

- 추상화
  - 현실에 존재하는 객체의 필요한 속성을 추출하는 작업
- 학생의 성적 총점과 평균을 계산하는 예제 작성

```
<script>
  var st0 = { 이름: '윤', 국어: 87, 수학: 98, 영어: 88, 과학: 95 };
  var st1 = { 이름: '연', 국어: 92, 수학: 98, 영어: 96, 과학: 98 };
  var st2 = { 이름: '구', 국어: 76, 수학: 96, 영어: 94, 과학: 90 };
  var st3 = { 이름: '나', 국어: 98, 수학: 92, 영어: 96, 과학: 92 };
  var st4 = { 이름: '강', 국어: 95, 수학: 98, 영어: 98, 과학: 98 };
  var st5 = { 이름: '박', 국어: 64, 수학: 88, 영어: 92, 과학: 92 };
  var st6 = { 이름: '김', 국어: 82, 수학: 86, 영어: 98, 과학: 88 };
  var st7 = { 이름: '서', 국어: 88, 수학: 74, 영어: 78, 과학: 92 };
</script>
```

객체와 배열을 사용한 데이터 관리

- 배열에 데이터 추가

```
<script>
  var st = [];
  st.push({ 이름: '윤', 국어: 87, 수학: 98, 영어: 88, 과학: 95 });
  st.push({ 이름: '연', 국어: 92, 수학: 98, 영어: 96, 과학: 98 });
  st.push({ 이름: '구', 국어: 76, 수학: 96, 영어: 94, 과학: 90 });
  st.push({ 이름: '나', 국어: 98, 수학: 92, 영어: 96, 과학: 92 });
  st.push({ 이름: '강', 국어: 95, 수학: 98, 영어: 98, 과학: 98 });
  st.push({ 이름: '박', 국어: 64, 수학: 88, 영어: 92, 과학: 92 });
  st.push({ 이름: '김', 국어: 82, 수학: 86, 영어: 98, 과학: 88 });
  st.push({ 이름: '서', 국어: 88, 수학: 74, 영어: 78, 과학: 92 });
</script>
```

객체와 배열을 사용한 데이터 관리

메서드 추가

```
<script>
  for (var i in st) {
    st [i].getSum = function () { // 총점 메서드 추가
      return this.국어 + this.수학 + this.영어 + this.과학;
    };
    st [i].getAverage = function () { // 평균 메서드 추가
      return this.getSum() / 4;
    };
  }
</script>
```

출력

```
<script>
  var output = '이름₩t총점₩t평균₩n';
  for (var i in st) {
    with (st [i]) {
      output += 이름 + '₩t' + getSum() + '₩t' + getAverage() + '₩n';
    }
  }  alert(output);
</script>
```

함수를 사용한 객체 생성

- 객체를 한 개씩 직접 만들어서 배열에 넣는 방식
  - 서로 다른 형태의 객체를 배열 안에 넣을 수 있는 장점
  - 개별적 객체를 만드는 것이 객체의 특성을 정확히 반영

```
<script>
  var students = [];
  students.push({ 이름: '윤', 국어: 87, 수학: 98,
                영어: 88, 과학: 95, 장래희망: '생명공학자' });
  students.push({ 이름: '연', 국어: 92, 수학: 98,
     영어: 96, 과학: 98, 특기: '요리', 취미: '일렉 기타' });
  students.push({ 이름: '구', 국어: 76, 수학: 96,
                영어: 94, 과학: 90, 장래희망: '프로그래머' });
</script>
```

함수를 사용한 객체 생성

- 함수를 사용하여 고정된 형태의 틀에서 찍어내서 쉽고 빠르게 객체를 생성할 수 있도록 함

```
<script>
  function makeStudent(name, korean, math, english, science) {
    var student = {
        이름: name, 국어: korean, 수학: math, 영어: english, 과학: science,
        getSum: function() {
            return this.국어 + this.수학 + this.영어 + this.과학;
        },
        getAverage: function() { return this.getSum() / 4; },
        toString: function() {
            return this.이름 + '₩t' + this.getSum() + '₩t' + this.getAverage();
        }
     };
     return student;
  }
</script>
```

함수를 사용한 객체 생성

```
<script>
  function makeStudent(name, Korean, math, English, science) …

  var students = [];
  students.push(makeStudent('윤', 87, 98, 88, 95));
  …
  students.push(makeStudent('연', 92, 98, 96, 98));
  var output = '이름₩t총점₩t평균₩n';
  for (var i in students) {
    output += students[i].toString() + '₩n';
  }
  alert(output);
</script>
```

생성자 함수

- 생성자 함수란?
  - **new** 키워드를 사용해서 객체를 생성할 수 있도록 해주는 함수
  - 생성자 함수에서는 객체를 반환하지 않음
  - 생성자는 **대문자**로 시작하는 것이 관례 (일반 함수와 구별 가능)
- 생성자 정의 및 new 키워드
  - new 키워드로 객체 생성

```
<script>
  function Student() { // 생성자 함수 정의
  }
  var student = new Student(); // 객체 생성
</script>
```

생성자 함수

- this 키워드
  - 생성자 함수로 생성될 객체의 속성 지정

```
<script>
  function Student(name, korean, math, english, science) {
    this.name = name;
    this.kor = korean;
    this.math = math;
    this.eng = english;
    this.sci = science;
  }
  var student = new Student('윤', 96, 98, 92, 98);
  for (i in student) { alert(i + ": " + student[i]); }
</script>
```

생성자 함수

- 메서드 생성

```
<script>
  function Student(name, korean, math, english, science) {
    ...
    this.getSum = function() {
      return this.kor + this.math + this.eng + this.sci;
    };
    this.getAverage = function() { return this.getSum() / 4; };
    this.toString = function() {
      return this.name + '₩t' + this.getSum() + '₩t'
            + this.getAverage();
      };
  }
  var student = new Student('윤', 96, 98, 92, 98);
</script>
```

생성자 함수

- 생성자 함수를 사용한 **객체 배열** 생성

```
<script>
  function Student(name, korean, math, english, science) { … }
  var students = [];
  students.push(new Student('윤', 96, 98, 92, 98));
  …
  students.push(new Student('아', 96, 96, 98, 92));
  var output = '이름₩t총점₩t평균₩n';
  for (var i in students) {
    output += students[i].toString() + '₩n';
  }
  alert(output);
</script>
```

프로토타입(Prototype)

- 생성자 함수
  - 기존의 객체 구조
    - name, kor, math, eng, sci 속성
    - getSum(), getAverage(), toString() 메서드

Student

name	
kor	
math	
eng	
sci	
getSum() { … }	
getAverage() { … }	
toString() { … }	

name	윤
kor	96
math	98
eng	92
sci	98
getSum() { … }	
getAverage() { … }	
toString() { … }	

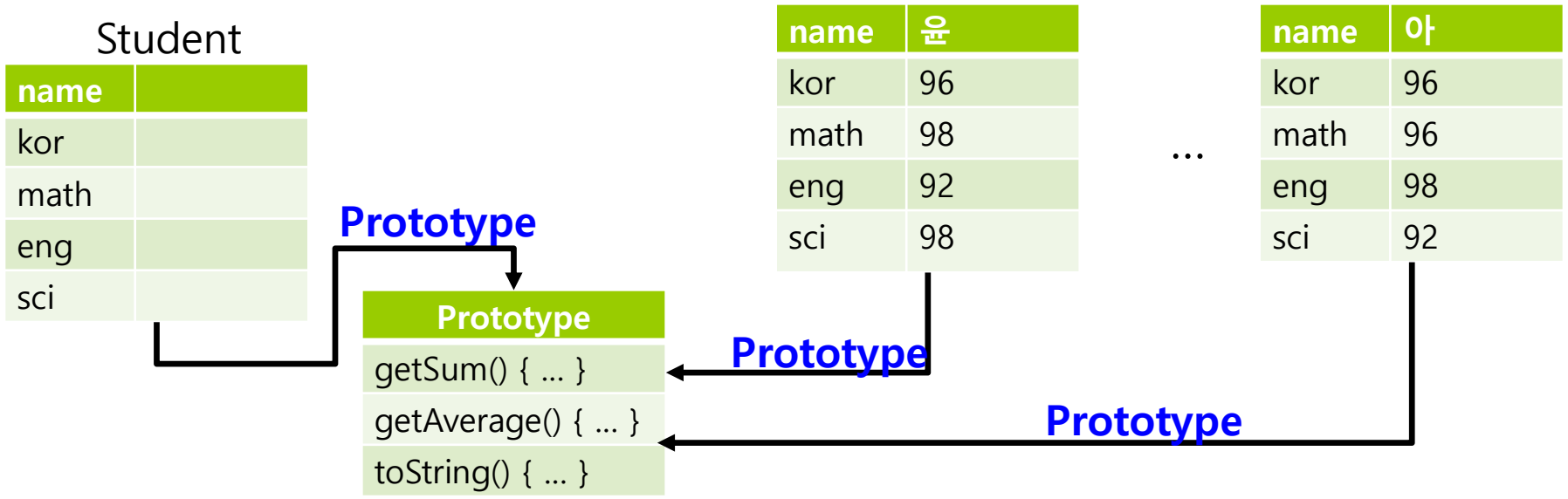
…

name	아
kor	96
math	96
eng	98
sci	92
getSum() { … }	
getAverage() { … }	
toString() { … }	

- **객체의 변수 속성은 다른 값들을 가질 수 있음**
- **하지만 메서드들은 같은 값들이 포함되기 때문에 반복적으로 들어가면 메모리 낭비가 심함**

프로토타입(Prototype)

- 메모리 낭비 문제 해결
  - **프로토타입**
    - 동일한 함수 생성에 따른 비효율적인 메모리 이용을 해결
    - 생성자 함수로 생성된 객체가 공통으로 가지는 공간
    - 메서드를 한 개만 생성해도 모든 객체가 해당 메서드를 사용할 수 있음



프로토타입

- 생성자 함수 구성
  - 한 개의 메서드를 만들면 모든 객체가 사용
  - 생성자 함수로 객체를 만들 때 **생성자 함수 내부에 속성만 넣음**
  - **메서드는 모두 프로토타입(prototype) 안에 넣음**

```
<script>
  function Student(name, korean, math, english, science) {
    this.name = name;
    this.kor = korean;
    this.math = math;
    this.eng = english;
    this.sci = science;
  }
</script>
```

프로토타입

- 프로토타입
  - prototype 속성은 모든 함수에 포함되어 있음

```
<script>
  function Student(name, korean, math, english, science) {
    this.name = name;
    this.kor = korean;
    this.math = math;
    this.eng = english;
    this.sci = science;
  }
  Student.prototype.getSum = function() { ... };
  Student.prototype.getAverage = function() { ... };
  Student.prototype.toString = function() { ... };
</script>
```

new 키워드

- new 키워드

```
<script>
  function Constructor(value) { // 생성자 함수 선언
    this.value = value;
  }
  var constructor = new Constructor('Hello'); // 변수 선언
  alert(constructor.value); // contructor에 넣은 value를 출력 Hello
</script>
```

new 키워드

- new 키워드를 사용하지 않으면?
  - 일반적인 this 키워드 사용하면, window 객체를 나타냄
  - 일반 함수 호출과 같이 **new 키워드를 사용하지 않으면**, 함수 실행 중 window 객체에 속성이 추가한 것이 됨
  - **new** 키워드로 함수 호출해야만, 객체를 위한 공간 생성되고(this 키워드가 해당 공간을 의미)

```
<script>
  function Constructor(value) {// 생성자 함수 선언
    this.value = value; // window의 value를 수정
  }
  var constructor = Constructor('Hello'); // 변수 선언
  alert(value); // value를 출력 Hello
</script>
```

캡슐화

- 캡슐화되지 않은 Rectangle 객체

```
<script>
  function Rectangle(width, height) {
    this.width = width;
    this.height = height;
  }
  Rectangle.prototype.getArea = function() {
    return this.width * this.height;
  }
  var rectangle = new Rectangle(5, 7);
  alert('AREA: ' + rectangle.getArea()); // 35
  rectangle.width = 3;
  rectangle.height = 4;
  alert('AREA: ' + rectangle.getArea()); // 12
</script>
```

캡슐화

- 캡슐화는 객체의 특정 속성이나 메서드를 사용자가 사용할 수 없도록 숨겨놓는 것임
  - Getter get○○ ( ) 형태의 메서드와 같이 값을 가져오는 메서드
  - Setter set○○ ( ) 형태의 메서드와 같이 값을 입력하는 메서드
  - Getter와 Setter를 만드는 것이 캡슐화는 아님

```
<script>
  function Rectangle(w, h) {
    var width = w;
    var height = h;
    this.getWidth = function() { return width; };
    this.getHeight = function() { return height; };
    this.setWidth = function(w) { width = w; };
    this.setHeight = function(h) { height = h; };
  }
  Rectangle.prototype.getArea = function() {
    return this.getWidth() * this.getHeight();
  };
```

캡슐화

- 캡슐화 이후 속성에 직접 접근할 수 없음

```
  var rect = new Rectangle(5, 7);
  rect.width = 3;
  rect.height = 4;
  alert('AREA: ' + rect.getArea()); // 5x7 = 35
</script>
```

- 캡슐화 이후 Getter/Setter 메서드를 이용해서 접근

```
  var rect = new Rectangle(5, 7);
  rect.setWidth(3);
  rect.setHeight(4);
  alert('AREA: ' + rect.getArea()); // 3x4 = 12
</script>
```

캡슐화

- Getter와 Setter 함수 수정

```
<script>
  function Rectangle(w, h) {
    ...
    this.setWidth = function(w) {
      if (w < 0) { throw '길이는 음수일 수 없음'; }
      else { width = w; }
    };
    this.setHeight = function(h) {
      if (h < 0) { throw '길이는 음수일 수 없음'; }
      else { height = h; }
    };
  }
  var rectangle = new Rectangle(5, 7);
  rectangle.setWidth(-3); // throw exception
  alert('AREA: ' + rectangle.getArea());
</script>
```

상속

- **상속**이란?
  - 기존의 생성자 함수나 객체를 기반으로 새로운 생성자 함수나 객체를 쉽게 만드는 것
  - 상속으로 만들어지는 객체는 기존 객체의 특성이 모두 있음
  - 상속을 사용하면 이전에 만들었던 객체와 비슷한 객체를 쉽게 만들 수 있음
- Rectangle을 상속받는 Square

```
<script>
  function Square(length) {
    this.width = length;
    this.height = length;
  }
  Square.prototype.getArea = function() {
    return this.getWidth() * this.getHeight();
  };
</script>
```

상속

- Square는 Rectangle과 유사함(width와 height가 같은 Rectangle임)
- 따라서 Rectangle에서 구현해둔 코드를 활용하도록 함(상속)
- 상속의 예
  - 생성자 함수 Square 내부에서 작성한 것

  → (1) base 속성에 생성자 함수 Rectangle을 넣고 실행한 것

  → (2) 생성자 함수 Square 프로토타입에 Rectangle의 프로토타입을 넣은 것

  - (1)을 사용해 Rectangle 객체의 속성을 Square 객체에 추가
  - (2)를 사용해 Rectangle 객체의 프로토타입이 가진 속성 또는 메서드를 Square 객체의 프로토타입에 복사

상속

- 상속 활용

```
<script>
  function Square(length) {
    this.base = Rectangle;
    this.base(length, length);
  }
  Square.prototype = Rectangle.prototype; // 상속
  Square.prototype.constructor = Square;  // 생성자 재지정
  var rectangle = new Rectangle(5, 7);
  var square = new Square(5);
  alert(rectangle.getArea() + ' : ' + square.getArea());
</script>
```

- 상속 확인

```
<script>
  var square = new Square(5);
  alert(square instanceof Rectangle); // true
  alert(typeof square); // object
</script>
```

이벤트

- 이벤트

HTML Event	설명
onchange	HTML element이 변화될 때
onclick	사용자가 HTML element를 클릭할 때
onmouseover	사용자가 HTML element에 마우스 오버 할 때
onmouseout	사용자가 HTML elemen에 마우스를 뺄 때
onkeydown	사용자가 키를 누를 때
onload	웹브라우저가 페이지 로딩을 했을 때

- 이벤트 지정 방법
  - **태그에 직접 지정**

```
<태그 ... on이벤트명(이벤트 종류) = "이벤트핸들러">내용</태그>
```

  - **script 태그 안에서 지정**
    - **객체명.on이벤트명 = 함수명; // 그리고 함수 선언**
    - **객체명.on이벤트명 = function() { 처리할 내용; ... } // 함수 표현**

이벤트

```
<button type="button" onclick="printDate()">Time is?</button> // 버튼 클릭 이벤트
<script>
function printDate() {
    document.getElementById("date").innerHTML = Date();
}
</script>
<p id="date"></p>
```

```
<script type="text/javascript">
    document.onmousedown = click; // 문서의 mousedown 이벤트
    function click() {
        if (event.button == 2) { // 마우스 우클릭 방지
            alert("Sorry, You cannot use the right mouse button.");
        }
    }
</script>
```