#### **JAVSCRIPT** ―――

<body> **요기 Script 요기** </body> 사이에 넣음!

| <!DOCTYPE html>  <html lang="ko">  <head>  <meta charset="UTF-8">  <title>Document</title>  </head>  <body>  html이 와바바봥왕 들어가고 그 아래 스크립트!  <script>  'use strict'; //strict mode 사용을 한다는 의미이며 코드를 엄격하게 사용하도록 함  </script>  </body>  </html> |
| --- |

| **리터럴(데이터)** | **직접 표현되는 값 그 자체** | |
| --- | --- | --- |
| 숫자 | var a1 = 10;  자바스크립트의 수치는 모두 **실수** 정수형 개념은 없다. | |
| 문자열 | **var a2 = "문자"**  // var a2 = '문자';  두개 혼용해서 쓰지 않기! (권장,,,)- | |
| 논리형 타입 (true false) | var a3 = false; | |

#### - 웹의 동작(동적) 구현하기 위한 언어

#### - 웹 브라우저에서 동작(단, node.js는 서버에서 동작)

#### - html문서 내에서 <script> ~ <script> 태그 안에 작성

#### - 대소문자를 구별

#### - 객체 기반의 스크립트 언어

#### - 웹 브라우저 자바스크립트 엔진이 내장되어 있기 때문에 사용이 가능

#### 

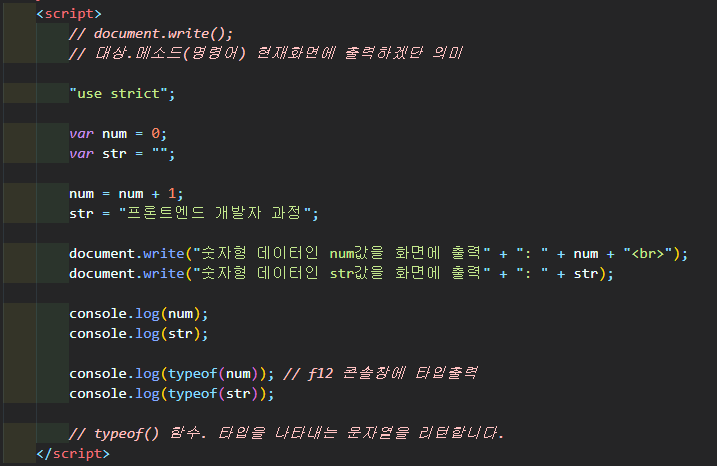
#### **▶ 문법**

#### 

#### 자바스크립트의 실행문은 세미콜론(;)으로 구분. 대소문자 구분해야함.

| **var javascript = 10; // 변수 javascript와 JavaScript는 서로 다른 두 개의 변수로 인식됨.**  **var JavaScript = 20;**  **Var Script = 30; // 변수의 선언은 var 키워드로만 할 수 있으면 Var는 동작하지 않음.** |
| --- |

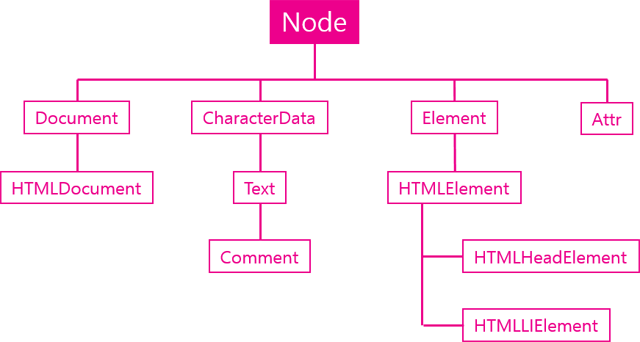
| **리터럴(데이터)** | **직접 표현되는 값 그 자체** | |
| --- | --- | --- |
| 숫자 | var a1 = 10;  자바스크립트의 수치는 모두 **실수** 정수형 개념은 없다. | |
| 문자열 | **var a2 = "문자"**  // var a2 = '문자';  두개 혼용해서 쓰지 않기! (권장,,,)- | |
| 논리형 타입 (true false)  = 불리언형(boolean) | var a3 = false;  참(true) 또는 거짓(false)으로 표현되는 값 | |



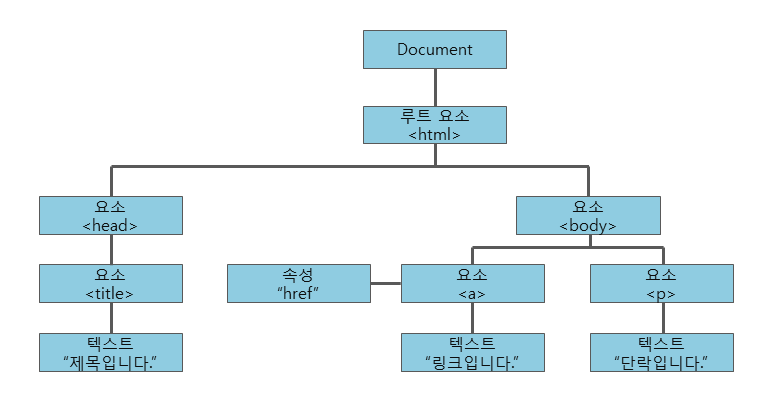
| **코드 값** | **비고** | | |
| --- | --- | --- | --- |
| typeof() 함수 | 타입을 나타내는 문자열을 리턴합니다. | | |
| alert(a2); | 변수 확인하기에 좋음!  <script>  function test() {  alert("확인을 누를 때까지 다른 작업을 할 수 없어요!");  }  </script> | | |
| null == undefined; // true | null과 undefined는 동등 연산자(==)와 일치 연산자(===)로 비교할 때 그 결과값이 다르므로 주의. null과 undefined는 타입을 제외하면 같은 의미지만, 타입이 다르므로 일치하지는 않음. | | |
| null === undefined; // false |
| == | 문자열을 등치로서 비교 | | |
| != , <> , ~= | 문자열을 동일하지 않은 것으로 비교 | | |
| + | 문자열을 연결 | | |
| alert(1==2);  alert(1==1);  alert("one"=="two");  alert("one"=="one");  alert(1=='1');  alert(1==='1');  alert(true == 1 );  alert(true === 1 );  && | false  true  false  true  이건 숫자로 자동 생각해서 멋대로 구럼,,  //형이 같은지까지 비교함  false는 0, 그 외 숫자는 1  false는 0, 그 외 숫자는 1 | | |

#### **Node 객체**

Node 객체는 DOM에서 시조와 같은 역할을 한다. 모든 DOM 객체는 Node 객체를 상속 받는다. Node 객체의 위상을 그림으로 나타내면 아래와 같다.



Node 객체는 모든 구성요소를 대표하는 객체이기 때문에 각각의 구성요소가 어떤 카테고리에 속하는 것인지를 알려주는 식별자를 제공한다.



노드 트리의 가장 상위에는 단 하나의 루트 노드(root node)가 존재합니다.

루트 노드를 제외한 모든 노드는 단 하나의 부모 노드(parent node)만을 가집니다.

모든 요소 노드는 자식 노드(child node)를 가질 수 있습니다.

형제 노드(sibling node)란 같은 부모 노드를 가지는 모든 노드를 가리킵니다.

조상 노드(ancestor node)란 부모 노드를 포함해 계층적으로 현재 노드보다 상위에 존재하는 모든 노드를 가리킵니다.

자손 노드(descendant node)란 자식 노드를 포함해 계층적으로 현재 노드보다 하위에 존재하는 모든 노드를 가리킵니다.

| **노드객체** | **비고** | | |
| --- | --- | --- | --- |
| node.parentNode  상위 노드에 접근 | var html=document.querySelector(".html");  var contUl=html.parentNode; | | |
| node.parentElement  상위 HTML노드에 접근 | var html=document.querySelector(".html");  var contUl=html.parentElement; | | |
| node.childNodes  하위 노드에 접근 | var cont=document.querySelector("#tutorial");  var contLi=cont.childNodes; | | |
| element.children  하위 HTML노드에 접근 | var cont=document.querySelector("#tutorial");  var contLi=cont.children; | | |
| node.firstChild  하위 첫 노드에 접근 | var cont=document.querySelector("#tutorial");  var contChild=cont.firstChild; | | |
| node.firstElementChild  하위 첫 HTML에 접근 | var cont=document.querySelector("#tutorial");  var contChild=cont.firstElementChild; | | |
| node.lastChild  하위 마지막 노드에 접근 | var cont=document.querySelector("#tutorial");  var contChild=cont.lastChild; | | |
| node.firstElementChild  하위 마지막 HTML에 접근 | var cont=document.querySelector("#tutorial");  var contChild=cont.lastElementChild; | | |
| node.previousSibling  이전 노드에 접근 | var css=document.querySelector(".css");  var previous=css.previousSibling; | | |
| node.previousElementSibling  이전 HTML에 접근 | var css=document.querySelector(".css");  var previous=css.previousElementSibling; | | |
| node.nextSibling  다음 노드에 접근 | var html=document.querySelector(".html");  var next=html.nextSibling; | | |
| node.nextElementSibling  다음 HTML 노드에 접근 | var html=document.querySelector(".html");  var next=html.nextElementSibling; | | |

| **노드 활용** | **비고** | | |
| --- | --- | --- | --- |
| 노드의 속성 제거 | var p = document.querySelectorAll("p");  for(var i=0; i<p.length; i++){  if(p[i].textContent == "portfolio"){  p[i].removeAttribute("id");  }  } | | |
| 노드의  속성여부 점검 | var p = document.querySelectorAll("p");  for(var i=0; i<p.length; i++){  if(p[i].hasAttribute("id")){ //id속성을 가지고 있다면  p[i].style.backgroundColor = "#666"  }  } | | |

#### **타입별 설명**

| **예문 html** |
| --- |
| <div class="container">  <ul id="tutorial">  <li class="html">HTML</li>  <li class="css">CSS</li>  <li class="css3">CSS3</li>  <li class="javascript">JavaScript</li>  <li class="jquery">jQuery</li>  <li class="blank"></li>  </ul>  <p id="portfolio">  portfolio  </p>  </div> |

| **노드객체** | **비고** | | |
| --- | --- | --- | --- |
| id로 접근 | 요소에 id 속성이 있으면 위치에 상관없이  메서드 document.getElementById(id)를 이용해 접근할 수 있습니다.  var tutorial = document.getElementById('tutorial'); //$('#tutorial')  tutorial.style.backgroundColor = "#666"; | | |

| **노드리스트** | **비고** | | |
| --- | --- | --- | --- |
| 태그 접근 | var li = document.getElementsByTagName("li");  console.log(li);  console.log(li[1]);  console.log(li.item(1));  console.log("li.length : "+li.length);  for(var i = 0; i < li.length;i++ ){  li[i].style.backgroundColor="#666";  }  var html = document.querySelector(".html");  var tagName = html.tagName;  console.log(tagName);  ㄴ 위처럼 할 경우 태그 속성값이 무엇인지 뜬다 (LI) | | |
| class로 접근 | var html = document.getElementsByClassName('html');  html[0].style.backgroundColor = 'red';  console.log(html);  console.log(html[0]);  console.log(html.item(0));  var li = document.querySelectorAll("#tutorial li");  for(var i=0; i < li.length; i++){  //element.className 노드의 클래스 이름  if(li[i].className == "html"){ li[i].style.backgroundColor = "#ccc";  }  } | | |

| **노드선택** | **비고** | | |
| --- | --- | --- | --- |
| css 기법으로  첫번째 노드 | var li = document.querySelector('#tutorial > li');  li.style.backgroundColor = "blue"; | | |

| **css 기법으로 전체 노드 선택 (querySelectorAll)** | | | |
| --- | --- | --- | --- |
| elem.querySelectorAll(css)은 다재다능한 요소 검색 메서드입니다.  이 메서드는 elem의 자식 요소 중 주어진 CSS 선택자에 대응하는 요소 모두를 반환합니다.  for  var li = document.querySelectorAll('#tutorial > li');  for(var i=0; i<li.length; i++){  console.log(li.length);  //console.log(li[i]);  //node.textContent 노드의 텍스트 내용 읽고 쓰기  //console.log(li[i].textContent)//li내에있는 텍스트 내용이 콘솔창에 출력  if(li[i].textContent == "HTML" || li[i].textContent == "CSS"){  li[i].style.backgroundColor = "#ccc";  }  }  for문으로 li를 전부 검사하여 textcontent가 HTML이거나 CSS면 배경을 #ccc로 지정하겠다  textContent == "대문자를 써야 대소문자 둘다 인식함 | | | |

| **getElementsByTagName 으로 접근** | | | |
| --- | --- | --- | --- |
| var li = document.getElementsByTagName("li");  for(var i=0; i<li.length; i++){  if(li[i].textContent == "HTML" || li[i].textContent =="CSS"){  li[i].style.backgroundColor="#666"  }  }  Tagname li를 전부 검사하여 textcontent가 HTML이거나 CSS면 배경을 #ccc로 지정하겠다 | | | |

#### **명시적 타입 변환**

| **코드 값** | **비고** | | |
| --- | --- | --- | --- |
| Number() | 문자를 숫자로 변환 | | |
| String() | 숫자나 불린등을 문자형으로 변환 | | |
| Boolean() | 문자나 숫자등을 불린형으로 변환 | | |
| Object() | 모든 자료형을 객체형으로 변환 | | |
| parseInt() | 문자형을 int형으로 변환 | | |
| parseFloat() | 문자형을 float형으로 변환 | | |

#### **변수**

#### 데이터를 담아두는 공간. 프로그램에서 사용되는 데이터를 일정 기간 동안 기억함.

#### 필요한 때에 다시 사용하기 위해 데이터에 고유의 이름인 식별자를 명시한 것.

값이 변경될 수 있으며 자바스크립트에서는 let 키워드를 사용하여 변수를 선언하고

선언되지 않은 변수를 사용하거나 접근하려고 하면 오류가 발생합니다.

| let 변수명; // 변수를 선언  변수명 = 값; // 변수를 초기화  let num;  num = 10; |
| --- |

| JavaScript는 값이 할당되는 과정에서 자동으로 변수의 타입이 결정된다.  즉, 변수는 고정된 타입이 없으며 같은 변수에 여러 타입의 값을 자유롭게 할당 가능.  변수는 애플리케이션에서 한번 쓰고 버리는 값이 아닌 값이 아닌 일정 기간 유지할 필요가 있는 값에 사용하며 변수를 사용하면 값의 의미가 명확해져서 코드의 가독성이 좋아진다.  선언하지 않은 변수명, 예약어가 아닌 단어는 기본적으로 JavaScript에서 사용할 수 없다. |
| --- |

| **명칭** | **뜻** | |
| --- | --- | --- |
| 식별자 | 어떤 대상을 유일하게 식별할 수 있는 이름   | 숫자가 포함될 수 있다.  알파벳이 포함될 수 있다.  $가 포함될 수 있다.  \_(underscore)가 포함될 수 있다.  영문자의 대소문자를 구분한다.  단, 숫자로 시작할 수 없다.  JavaScript Keyword(예약어)는 식별자가 될 수 없다. | | --- | | |
| 변수명 | 변수에 명시한 고유한 식별자 | |
| 변수값 | 변수로 참조할 수 있는 데이터 값 | |

변수는 var, let, const 키워드를 사용하여 선언하고 할당 연산자(=)를 사용해 값을 할당.

식별자인 변수명을 사용해 변수에 저장된 값을 참조함.

| var score;  score = 80;  score = 90;  score; | 변수의 선언  값의 할당  값의 재할당  변수의 참조 |
| --- | --- |

#### **상수(constant)**

#### 한번 선언된 상수는 다시 재정의 할 수 없으며, 값도 재할당 할 수 없습니다.

| const 상수명;  상수명 = 값 (X)  const 상수명 = 값 (o)  상수명 = 값 (x) // 재할당 할 수 없음 |
| --- |

| \* var를 사용하지 않는 이유  1. 같은 이름의 변수를 선언할 수 있음  2. 지역변수와 전역변수의 구별이 없음 |
| --- |

let 변수명; // 변수를 선언

변수명 = 값; // 변수를 초기화

상수: 한번 선언된 상수는 재정의X

const 상수명; - id느낌?

var를 사용하지 않는 이유:

- 같은 이름의 변수 선언할수있음

- 지역변수와 전역변수 구분X

#### **데이터 타입**

#### 프로그래밍 언어에서 사용할 수 있는 데이터 (숫자, 문자열, 불리언 등)의 종류

| **숫자형 코드값** | **뜻** | **예문** | |
| --- | --- | --- | --- |
| \* | 곱한값 | var num2 = num \* 2; | |
| % | 나눈 나머지값 | var remainder1 = 30 %4; | |
| >= | 크거나 같음 | var result2 = 3 >= 1; | |
| 일치연산자(===) | 완전히 동일 | console.log(one === two); // true | |
| a += 1; | a = a + 1 |  | |
| a \*= 3; | a = a \* 3 |  | |
| a++; | a = a + 1 |  | |
| a--; | a = a - 1 |  | |

| **문자열** | **뜻** | **예문** | |
| --- | --- | --- | --- |
| String | 텍스트 데이터를 나타냄  따옴표로 감싸져 있는 텍스트  + 기호를 사용하여 아래와 같이 문자열을 붙일 수 있다 | var s1 = "something";  var s2 = "else";  var result = s1 + s2;  console.log(result); // "somethingelse" | |
| 문자열 비교. 아래와 같이 같고 다름을 비교할 수 있다.  var s3 = "abc "; // 유의: 문자열 끝자리에 공백이 붙어있다.  var s4 = "abc";  var result2 = s3 !== s4; // s3과 s4가 다른지 확인하고 그 결과를 result2 변수에 담음  console.log(result2); // true | | | |
| length  var length | 문자열은 길이에 대한 정보  1) var txt = "ABCDEFG"  var sln = txt.length;  2) var length = 'abcdef',length;  1) alert(sln); 로 위 값을 확인할 수 있다 (7)  ※ string의 length이기 때문에 sln 사용  2) 값은 6 | var str1 = "abc"; // 3개의 텍스트를 포함하고 있는 문자열  console.log(str1.length); // 3  var str2 = " "; // 공백 3개  console.log(str2.length); // 3  console.log("graph-ql".length); // 8 | |

| **문자열** | **뜻** | **예문** | |
| --- | --- | --- | --- |
| prompt() | 문자입력창 | const cnt = prompt('뭐든 입력하세요.')); | |
| document.write() | 바로 출력됨 | document.write(`<p>입력한 수 ${arr}에서 가장 큰 값을 ${max}입니다</p>`); | |

#### **▶ JSON**

#### 이벤트 입니다!

#### **JSON.parse()**

| JSON.parse( text [ , reviver ] ) |
| --- |

JSON.parse() 메서드는 JSON 문자열의 구문을 분석하고, 그 결과에서 JavaScript 값이나 객체를 생성한다. 선택적으로, reviver 함수를 인수로 전달할 경우, 결과를 반환하기 전에 변형할 수 있다.

| const json = '{"result":true, "count":42}';  const obj = JSON**.parse**(json );  console.log(obj .count); // expected output: 42  console.log(obj .result); // expected output: true |
| --- |

| **코드 값** | **비고** | | |
| --- | --- | --- | --- |
| text | JSON으로 변환할 문자열. | | |
| reviver Optional | 함수라면, 변환 결과를 반환하기 전에 이 인수에 전달해 변형함. | | |
| 반환 값 | 주어진 JSON 문자열에 대응하는 Object. | | |
| 예외 | 변환할 문자열이 유효한 JSON이 아닐 경우 SyntaxError. | | |

#### **JSON.stringify()**

| JSON.stringify( value [ , replacer [ , space ] ] ) |
| --- |

JSON.stringify() 메서드는 JavaScript 값이나 객체를 JSON 문자열로 변환합니다. 선택적으로, replacer를 함수로 전달할 경우 변환 전 값을 변형할 수 있고, 배열로 전달할 경우 지정한 속성만 결과에 포함합니다.

| JavaScript Demo: JSON.stringify()  console.log(JSON.**stringify**({ x: 5, y: 6 }));  // expected output: "{"x":5,"y":6}"  console.log(JSON.**stringify**([new Number(3), new String('false'), new Boolean(false)]));  // expected output: "[3,"false",false]"  console.log(JSON.**stringify**({ x: [10, undefined, function(){}, Symbol('')] }));  // expected output: "{"x":[10,null,null,null]}"  console.log(JSON.**stringify**(new Date(2006, 0, 2, 15, 4, 5)));  // expected output: ""2006-01-02T15:04:05.000Z"" |
| --- |

| **코드 값** | **비고** | | |
| --- | --- | --- | --- |
| value | JSON 문자열로 변환할 값. | | |
| replacer | 문자열화 동작 방식을 변경하는 함수, 혹은 JSON 문자열에 포함될 값 객체의 속성들을 선택하기 위한 화이트리스트(whitelist)로 쓰이는 String 과 Number 객체들의 배열.  이 값이 null 이거나 제공되지 않으면, 객체의 모든 속성들이 JSON 문자열 결과에 포함된다. | | |
| space | 가독성을 목적으로 JSON 문자열 출력에 공백을 삽입하는데 사용되는 String 또는 Number 객체. 이것이 Number 라면, 공백으로 사용되는 스페이스(space)의 수를 나타낸다;  이 수가 10 보다 크면 10 으로 제한된다. 1 보다 작은 값은 스페이스가 사용되지 않는 것을 나타낸다.  이것이 String 이라면, 그 문자열(만약 길이가 10 보다 길다면, 첫번째 10 개의 문자)이 공백으로 사용된다. 이 매개 변수가 제공되지 않는다면 (또는 null 이면), 공백이 사용되지 않는다. | | |
| 반환 값 | 주어진 값과 대응하는 JSON 문자열. | | |
| 예외 | 순환 참조를 발견할 경우 TypeError(cyclic object value). | | |

JSON.stringify()는 값을 JSON 표기법으로 변환한다.

배열이 아닌 객체의 속성들은 어떤 특정한 순서에 따라 문자열화 될 것이라고 보장되지 않는다. 같은 객체의 문자열화에 있어서 속성의 순서에 의존하지 않는다.

Boolean, Number, String 객체들은 문자열화 될 때 전통적인 변환 의미에 따라 연관된 기본형(primitive) 값으로 변환된다.

undefined, 함수, 심볼(symbol)은 변환될 때 생략되거나(객체 안에 있을 경우) null 로 변환된다(배열 안에 있을 경우).

심볼을 키로 가지는 속성들은 replacer 함수를 사용하더라도 완전히 무시된다.

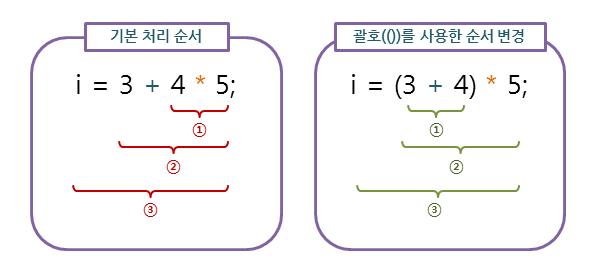
열거 불가능한 속성들은 무시된다.

#### **▶ 연산자**

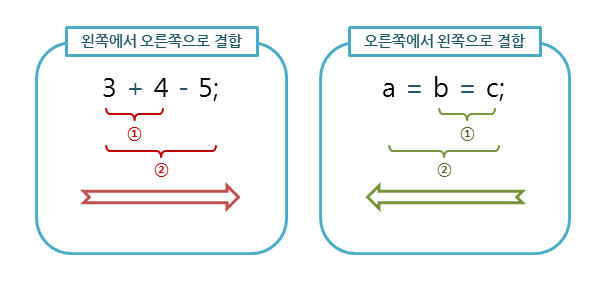
#### 이벤트 입니다!

#### **연산자 우선순위**

연산자의 우선순위는 수식 내에 여러 연산자가 함께 등장할 때, 어느 연산자가 먼저 처리될 것인가를 결정함. 아래 그림은 가장 높은 우선순위를 가지고 있는 괄호(()) 연산자를 사용하여 연산자의 처리 순서를 변경하는 것.



아래 그림은 결합 방향은 수식 내에 우선순위가 같은 연산자가 둘 이상 있을 때, 먼저 어느 연산을 수행할 것인가를 결정



#### **연산자 우선순위표**

| **우선순위** | **연산자** | **설명** | **결합 방향** |
| --- | --- | --- | --- |
| **1** | [] | 첨자 연산자 | 왼쪽에서 오른쪽으로 |
|  | . | 멤버 연산자 | 왼쪽에서 오른쪽으로 |
| **2** | ++ | 후위 증가 연산자 | 왼쪽에서 오른쪽으로 |
|  | -- | 후위 감소 연산자 | 왼쪽에서 오른쪽으로 |
| **3** | ! | 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 |
|  | (타입) | 타입 캐스트 연산자 | 오른쪽에서 왼쪽으로 |
| **4** | \* | 곱셈 연산자 | 왼쪽에서 오른쪽으로 |
|  | / | 나눗셈 연산자 | 왼쪽에서 오른쪽으로 |
|  | % | 나머지 연산자 | 왼쪽에서 오른쪽으로 |
| **5** | + | 덧셈 연산자 (이항 연산자) | 왼쪽에서 오른쪽으로 |
|  | - | 뺄셈 연산자 (이항 연산자) | 왼쪽에서 오른쪽으로 |
| **6** | << | 비트 왼쪽 시프트 연산자 | 왼쪽에서 오른쪽으로 |
|  | >> | 부호 비트를 확장하면서 비트 오른쪽 시프트 | 왼쪽에서 오른쪽으로 |
|  | >>> | 부호 비트까지 모두 비트 오른쪽 시프트 | 왼쪽에서 오른쪽으로 |
| **7** | < | 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 | 인스턴스의 실제 타입 반환 | 왼쪽에서 오른쪽으로 |
| **8** | == | 관계 연산자(와 같은) | 왼쪽에서 오른쪽으로 |
|  | != | 관계 연산자(와 같지 않은) | 왼쪽에서 오른쪽으로 |
| **9** | & | 비트 AND 연산자 | 왼쪽에서 오른쪽으로 |
| **10** | ^ | 비트 XOR 연산자 | 왼쪽에서 오른쪽으로 |
| **11** | | | 비트 OR 연산자 | 왼쪽에서 오른쪽으로 |
| **12** | && | 논리 AND 연산자 | 왼쪽에서 오른쪽으로 |
| **13** | || | 논리 OR 연산자 | 왼쪽에서 오른쪽으로 |
| **14** | ? : | 삼항 조건 연산자 | 오른쪽에서 왼쪽으로 |
| **15** | = | 대입 연산자 및 복합 대입 연산자  (=, +=, -=, \*=, /=, %=, <<=, >>=, >>>=, &=, ^=, |=) | 오른쪽 |

#### **산술 연산자**

사칙연산 및 기본적인 연산을 하는 연산자

모두 두 개의 피연산자를 가지는 이항 연산자이며, 결합 방향은 왼쪽에서 오른쪽.

| **리터럴(데이터)** | **직접 표현되는 값 그 자체** | |
| --- | --- | --- |
| + | 덧셈  왼쪽의 피연산자에 오른쪽의 피연산자를 더함. | |
| - | 뺄셈  왼쪽의 피연산자에서 오른쪽의 피연산자를 뺌. | |
| \* | 곱셈  왼쪽의 피연산자에 오른쪽의 피연산자를 곱함. | |
| / | 나눗셈  왼쪽의 피연산자를 오른쪽의 피연산자로 나눔. | |
| % | 나눈 나머지 값 (ex 10%3 =1)  왼쪽의 피연산자를 오른쪽의 피연산자로 나눈 후, 그 나머지를 반환함 | |
| \*\* | 거듭제곱 (ex 2\*\*2 = 4) | |

#### **비교 연산자**

피연산자 사이의 상대적인 크기를 판단하여, 참(true)과 거짓(false)을 반환

두 개의 피연산자를 가지는 이항 연산자이며, 피연산자들의 결합 방향은 왼쪽에서 오른쪽

1. 피연산자가 둘 다 숫자면, 해당 숫자를 서로 비교합니다.

2. 피연산자가 둘 다 문자열이면, 문자열의 첫 번째 문자부터 알파벳 순서대로 비교합니다.

| **리터럴(데이터)** | **직접 표현되는 값 그 자체** | |
| --- | --- | --- |
| > | 크다  왼쪽 피연산자의 값이 오른쪽 피연산자의 값보다 크면 참을 반환함. | |
| < | 작다  왼쪽 피연산자의 값이 오른쪽 피연산자의 값보다 크거나 같으면 참을 반환함. | |
| >= | 크거나 같다  왼쪽 피연산자의 값이 오른쪽 피연산자의 값보다 작으면 참을 반환함. | |
| <= | 작거나 같다  왼쪽 피연산자의 값이 오른쪽 피연산자의 값보다 작거나 같으면 참을 반환함. | |
| == | 같다 ("10" == 10 는 true -> 타입이 달라도 값이 같으면 참)  왼쪽 피연산자와 오른쪽 피연산자의 값이 같으면 참을 반환함. | |
| === | 같다 ("10" === 10 는 false -> 타입과 값이 같아야 참)  왼쪽 피연산자와 오른쪽 피연산자의 값이 같고, 같은 타입이면 참을 반환함. | |
| != | 다르다 (값만 같으면 false, 값이 같으면 true)  왼쪽 피연산자와 오른쪽 피연산자의 값이 같지 않으면 참을 반환함. | |
| !== | 다르다 (값, 타입 둘 중 하나만 다르면 true)  왼쪽 피연산자와 오른쪽 피연산자의 값이 같지 않거나, 타입이 다르면 참을 반환함. | |

#### **대입 연산자**

변수에 값을 대입할 때 사용하는 이항 연산자, 피연산자들의 결합 방향은 오른쪽에서 왼쪽

(산술 연산자와 결합한 다양한 복합 대입 연산자가 존재)

| = : 우측의 값을 왼쪽(변수 또는 상수)에 대입  +=, -=, \*=, /=, %=, \*\*= : 우측의 값을 왼쪽의 값과 연산 후 다시 왼쪽(변수)에 대입합니다.  let num=10;  num += 5; // num = num + 5; (같은 표현) |
| --- |

| **리터럴(데이터)** | **직접 표현되는 값 그 자체** | |
| --- | --- | --- |
| = | 왼쪽의 피연산자에 오른쪽의 피연산자를 대입함. | |
| += | 왼쪽의 피연산자에 오른쪽의 피연산자를 더한 후,  그 결과값을 왼쪽의 피연산자에 대입함. | |
| -= | 왼쪽의 피연산자에서 오른쪽의 피연산자를 뺀 후,  그 결과값을 왼쪽의 피연산자에 대입함. | |
| \*= | 왼쪽의 피연산자에 오른쪽의 피연산자를 곱한 후,  그 결과값을 왼쪽의 피연산자에 대입함. | |
| /= | 왼쪽의 피연산자를 오른쪽의 피연산자로 나눈 후,  그 결과값을 왼쪽의 피연산자에 대입함. | |
| %= | 왼쪽의 피연산자를 오른쪽의 피연산자로 나눈 후,  그 나머지를 왼쪽의 피연산자에 대입함. | |

#### **증감 연산자**

증감 연산자는 1씩 값을 증가시키거나 또는 감소시킬 때 사용하는 연산자입니다. 연산자의 위치가 어디에 있는지에 따라 결과가 달라질 수 있습니다

| let a = 10;  ++a; // 11  a++; // 12  let b = ++a; // a=13, b=13  let c = a++; // a=14, c=13 |
| --- |

| **리터럴(데이터)** | **직접 표현되는 값 그 자체** | |
| --- | --- | --- |
| ++변수 | 변수의 값을 1증가시킨 후, 다음 연산을 진행 | |
| 변수++ | 먼저 연산을 수행한 후, 변수의 값을 1증가 시킴 | |
| --변수 | 변수의 값을 1감소시킨 후, 다음 연산을 진행 | |
| 변수-- | 먼저 연산을 수행한 후, 변수의 값을 1갑소 시킴 | |

#### **논리 연산자**

논리식을 판단하여, 참(true)과 거짓(false)을 반환합니다.

and, or, xor 연산자는 두 개의 피연산자를 가지는 이항 연산자이며, 피연산자들의 결합 방향은 왼쪽에서 오른쪽. ! 연산자는 피연산자가 단 하나뿐인 단항 연산자이며, 피연산자의 결합 방향은 오른쪽에서 왼쪽입니다.

| **리터럴(데이터)** | **직접 표현되는 값 그 자체** | |
| --- | --- | --- |
| and | 논리식이 모두 참이면 참을 반환함. (AND 연산) | |
| or | 논리식 중에서 하나라도 참이면 참을 반환함. (OR 연산) | |
| xor | 논리식이 서로 다르면 참을 반환함. (XOR 연산) | |
| && | 논리식이 모두 참이면 참을 반환함. (AND 연산) | |
| || | 논리식 중에서 하나라도 참이면 참을 반환함. (OR 연산) | |
| ! | 논리식의 결과가 참이면 거짓을, 거짓이면 참을 반환함. (NOT 연산) | |

and와 && 연산자, or와 || 연산자는 정확히 같은 동작을 하며, 연산자의 우선순위에서만 차이가 발생합니다.

논리 연산자의 모든 동작의 결과를 보여주는 진리표(truth table).

| **A** | **B** | **A and B (A && B)** | **A or B (A || B)** | **A xor B** | **!A** |
| --- | --- | --- | --- | --- | --- |
| true | true | true | true | false | false |
| true | false | false | true | true | false |
| false | true | false | true | true | true |
| false | false | false | false | false | true |

#### **삼항 연산자**

피연산자가 3개인 연산자 (조건식? 연산식1:연산식2)

= 조건식의 결과값이 참이면 연산식1, 거짓이면 연산식2을 실행하라.

조건식에는 boolean형(!!/논리연산자) 조건식이 들어간다.

조건식에 따른 참, 거짓에 따라 반환값이 달라지는 연산자

| true false  ----- -----  변수(상수) = 조건식 ? 결과1 : 결과2  ------  true 또는 false |
| --- |

\*\* false외에도 null,NaN, 0, 비어있는 문자 값 (""), 그리고 undefined으로 조건문에 false 값으로 사용 가능 합니다. 이 값들이 조건문으로 사용된다면 exprIfFalse(거짓일 때 실행할 식)이 결과로 나오게 됩니다.

\*\*null 값 처리하기

이와같이 null 값을 처리할 때에도 일반적으로 사용됩니다.:

| let greeting = person => {  let name = person ? person.name : `stranger`  return `Howdy, ${name}`  }  console.log(greeting({name: `Alice`})); // "Howdy, Alice"  console.log(greeting(null)); // "Howdy, stranger" |
| --- |

#### **▶ 배열**

이름과 인덱스로 참조되는 정렬된 값의 집합. 배열을 구성하는 각각의 값: 배열 요소

배열에서의 위치를 가리키는 숫자: 인덱스

| 배열을 만드는 방법  let 배열명;  배열명 = [요소1, 요소2, 요소3 ...];  let arr;  arr = [100, 90, 200, 50];  let 배열명 = new Array(요소1, 요소2, 요소3 ..); |
| --- |

배열 순서는 0부터 시작. 0, 1, 2, 3…

아래는 예시

| **예시** | **출력값** |
| --- | --- |
| let arr = ['루시', 10, 3.5, true]; |  |
| document.write(arr[0]+"<br>"); | 루시 |
| document.write(arr[1]+"<br>"); | 10 |
| document.write(arr[2]+"<br>"); | 3.5 |
| document.write(arr[3]+"<br><br>"); | true |

| **종류** | **script** | **결과** |
| --- | --- | --- |
| 배열추가 | arr[4] = 'white'; | let arr = ['루시', 10, 3.5, true. 'white']; |
| .push() | 배열의 맨 끝에 값을 추가한다. |
| .unshift() | 배열의 맨 앞에 값을 추가한다. |
| 배열삭제 | .pop()  // arr.pop() | 루시  10  3.5  true  마지막 요소를 삭제 |
| .shift() | 배열의 맨 앞에 있는 값을 제거한다. |
| 배열수정 | arr[4] = '포메리안'; | let arr = ['루시', 10, 3.5, true. '포메리안']; |
| 배열병합 | .concat() |  |
| 요소 사이에 추가 | .join() | 배열과 배열 사이에 문자를 삽입하여 문자열로 출력이 된다.  [배열명].join([추가문자열]) 형식이다. |
| 정렬 | .sort()  .reverse() |  |
| 순서대로 출력 | for(let i=0; i < arr.length; i++){  document.write(arr[i]);  } | 루시103.5true포메리안 |

#### **length property**

배열 요소의 개수를 저장. 배열 내에서 **가장 큰 인덱스 값**에 1을 더한 값이다.

자바스크립트의 모든 배열은 length프로퍼티를 가진다.

배열의 길이를 확인할 수 있는 속성

| 배열명.length > array1.length |
| --- |

배열의 변수명이 array1일때 뒤에 .length를 입력하면 array1의 길이를 의미하게 된다.

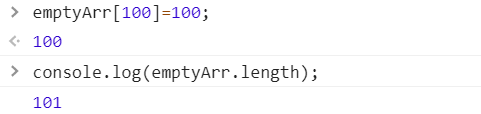
| let emptyArr = [];  console.log(emptyArr[0]) //undefined  emptyArr[0] = 200;  emptyArr[4] = 'apple';  emptyArr[10] = 'ten';  console.log(emptyArr); |
| --- |



emptyArr에 세 개의 값만 할당했지만 console.log(emptyArr)의 결과는 11개의 배열 요소 값이 출력된다. 그 이유는 자바스크립트가 배열의 크기를 현재 배열 인덱스 중 가장 큰 값을 기준으로 정하기 때문이다.

배열의 제일 큰 인덱스 값을 바꿔준다면 그에 따라 length 값도 바뀐다.

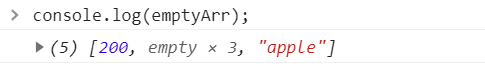
| emptyArr[100] = 100;  console.log(emptyArr); // 101 |
| --- |



length 프로퍼티는 명시적으로 값을 변경할 수도 있다.

| emptyArr.length = 50;  console.log(emptyArr.length); // 50  emptyArr.length = 5;  console.log(emptyArr.length); // 5  conso.log(empyArr); |
| --- |

length 프로퍼티를 변경했을 때, **프로퍼티 값을 벗어나는 요소는 삭제**된다.



#### **push()**

push 메서드는 인수로 넘어온 항목을 배열의 맨 끝에 추가하는 자바스크립트 표준 배열 메서드이다. push 메서드는 현재 배열의 length 값 위치에 새로운 원소값을 추가한다.

| let arr =['a','b','c']'  arr.push('d');  console.log(arr); //['a','b','c','d']  arr.length = 5;  arr.push('e');  console.log(arr); //['a','b','c','d',undefined,'e'] |
| --- |

arr 배열의 length를 5로 바꾸고 push 메서드를 실행하면 arr[5]에 값이 추가되는 것을 알 수 있다. 이를 통해 배열 메서드가 length 프로퍼티를 기반으로 동작하고 있다는 것을 알 수 있다.

#### **concet()**

두개의 문자열을 하나의 문자열로 만들어주는 역할을 하는 함수.

입력값을 문자열 대신 배열을 사용하면 두 개의 배열을 하나의 배열로 만들어주기도함

| - 문자열 합치기  "[문자열]".concat("[문자열]")'    - 배열 합치기  [배열명].concat([배열명]);    결과: arr : 배열 1,배열 2,배열 3,배열 4 |
| --- |

#### **▶ 제어문**

프로그램 내에서 주어진 조건문의 결과에 따라 별도의 명령을 수행하도록 제어하는 실행문

결과를 도출하기 위해 논리적 판단을 하는 역할.

논리적 판단에는 **조건을 비교**하거나 **일정 구간을 반복**시켜야 함.

| if / switch / for / while / do~ while |
| --- |

#### 

#### 

#### **if 문**

만약 조건식이 참이면 이것을 실행하세요.

| if(조건식) -> 만약 조건식이 참(true) 이면 {} 안에있는 실행문을 실행  else if(조건식) -> 첫번째 if(조건식)이 거짓(false)이고 지금 else if(조건식) 이 참(true) 이면 {} 안에있는 실행문을 실행 (생략 가능)  else -> 위 조건식들이 모두 거짓(false) 이면 실행. (생략 가능) |
| --- |

| **실제 표현** | **예문** | **비고** |
| --- | --- | --- |
| if (조건식){  실행문...  }else if(조건식){  실행문...  }else{  실행문...  } | var count = 0;    if(count==0){  alert('if:' + count);  }else if(count ==1 ){  alert('else if:' + count);  }else{  alert('else: ' + count);  } | var count 값이 0일때 출력값  ▶ if : 0  var count 값이 1일때 출력값  ▶ else if : 0  var count 값이 그 외 일때 출력값  ▶ else : 0 |

#### **switch 문**

뜻 그대로 스위치? else if문을 여러번 사용해야 할 경우에 유용하게 사용가능

switch문은 if ~ else if문과 마찬가지로 주어진 조건 값에 따라 프로그램이 다른 명령을 수행하도록 하는 조건

| 조건변수에 특정 값을 가진 변수가 들어감.  var count = 0; 이라는 변수가 있고 이 변수의 값에 따라 어떤 작업을 해야한다면,  switch(count) 로 작성하며 count가 어떤 값인지 case 문으로 비교, 확인.  case는 각각의 케이스를 말하는 것이며 '여기를 실행하세요' 라고 명시하는것으로 이해. (case1 : count 변수가 0입니까 ? case2 : count 변수가 1입니까? 등...)  break는 생략 가능하며 말그대로 '여기서 그만 실행하겠다'는 뜻.  break를 만나면 switch를 그만 실행(종료)하게 됨.  생략하게 되면 종료되지 않고 다음 case 로 이동하여 비교, 확인하며 switch문을 계속 실행,  default는 기본 값.  특정 case를 만족하지 못하면 실행문을 실행하지 못하고 break문도 못만났겠죠. 그때 마지막까지 내려와서 기본값을 실행하게 됩니다. 역시 생략 가능.  if 문의 else 구문과 비슷한 편! |
| --- |

| **실제 표현** | **예문** | **비고** |
| --- | --- | --- |
| switch(조건변수){  case 값1 : 실행문 1;  …  break;  case 값2 : 실행문 2;  …  break;  case 값3 : 실행문 3;  .....  break;  default :  실행문n;  ...  } | var test = prompt("숫자를 입력하세요.","")    switch (test)  {  case "1" : alert("하나");  break;  case "2" : alert("둘");  break;  case "3" : alert("셋");  break; default : alert("몰랑");  } | prompt: 문자열 입력할 때 사용.  첫번째 매개변수는 입력 창에 띄울 메세지, 두번째 메시지는 입력 부분의 기본 값  if문은  A가 아니면 B,  그게 아니면 C 느낌.  switch문은  A랑 B랑 C는 각자 a랑 b랑 c에  찾아서 들어가라는 느낌 |

#### **for 문**

반복문입니다. 특정 조건만큼 변수를 증감하고 실행문을 반복.

| for문은 for문 내부에서 사용할 지역변수(숫자)를 선언하고 조건식이 참일때 계속 실행. 증감식은 초기화 변수나 다른 특정 변수를 증가(++ , +n)하거나 감소(--, -n) 또는 변수를 변화시킬 수식이 들어갈 수 있음.  for(var i = 0; i < 5; i++){  실행문  ...  }  i 라는 변수를 0으로 초기화 하고 i 가 5보다 작을 경우 실행.  실행문이 끝나면 i를 1씩 증가시키고 다시 반복 실행.  조건식이 거짓(false)가 되면 중지. |
| --- |

| **실제 표현** | **예문** | **비고** |
| --- | --- | --- |
| for(초기화 변수; 조건식; 증감식){  실행문  ...  } | var test = 0;  for ( var a=1; a<=100; a=a\*2){  test = test + a;  }    document.write("합계 : " + test); | 증감식이 뒤에 있어야 나중에 실행 |

#### **for문 (with 배열)**

for 문을 이용해서 배열을 출력하는 방법이 이렇게 있네요….

length 활용법 눈여겨보자. length… 그 배열의 길이(갯수)만큼 for문을 돌려요~

| "use strict";  let arr1 = ["김사과", 20, 160.5, true];  for(let i=0; i<arr1.length; i++){  console.log(arr1[i]);  }  console.log("-----------------------");  for(let i in arr1){  console.log(i, arr1[i]);  }  console.log("-----------------------");  for(let i of arr1){  console.log(i);  }  console.log("-----------------------");  arr1.forEach(function(item, idx, arr){  console.log(item, idx, arr);  }); |
| --- |

| **종류** | **설명** | **비고** |
| --- | --- | --- |
| for… in 문 | 객체의 key값과 value값을 추출  키 값의 갯수만큼 반복하여 첫번째~마지막까지 반복 | 객체에 사용 가능 |
| for… of 문 | 컬렉션 객체가 [Symbol.iterator속성을 가졍야한다?]  직접명시가능?  for in 반복문 : 객체의 모든 열거 가능한 속성(property)에 대한 반복  for of 반복문 : [Symbol.iterator] 속성을 가지는 컬렉션 전용 |  |
| forEach문 | 배열의 첫번째부터 마지막까지 반복하며 item을 꺼낼 수 있다 | Array객체에만 사용 가능함 |

#### **while 문**

~ 하는 동안에. 조건식이 참(true)인 동안에 실행문을 반복.

for문처럼 초기값과 증감식은 없음.

| for문은 for문 내부에서 사용할 지역변수(숫자)를 선언하고 조건식이 참일때 계속 실행. 증감식은 초기화 변수나 다른 특정 변수를 증가(++ , +n)하거나 감소(--, -n) 또는 변수를 변화시킬 수식이 들어갈 수 있음.  for(var i = 0; i < 5; i++){  실행문  ...  }  i 라는 변수를 0으로 초기화 하고 i 가 5보다 작을 경우 실행.  실행문이 끝나면 i를 1씩 증가시키고 다시 반복 실행.  조건식이 거짓(false)가 되면 중지. |
| --- |

| **실제 표현** | **예문** | **비고** |
| --- | --- | --- |
| while(조건식){  실행문  ...  } | var a = 1;  while(a<=10){  document.write("안녕하세요(while)"+"<br>");  a++;  } | 아래와 같이 작성하면 무한 반복문. break; 구문과 함께 작성.  while(true){  실행문  }  ▶ 안녕하세요 10줄 출력됨 |

#### **do while 문**

무조건 최초 1회는 실행하고 조건식을 검사.

= 조건식이 거짓(false)여도 실행문을 먼저 실행한 후 검사 (while 과 차이점)

| for문은 for문 내부에서 사용할 지역변수(숫자)를 선언하고 조건식이 참일때 계속 실행. 증감식은 초기화 변수나 다른 특정 변수를 증가(++ , +n)하거나 감소(--, -n) 또는 변수를 변화시킬 수식이 들어갈 수 있음.  for(var i = 0; i < 5; i++){  실행문  ...  }  i 라는 변수를 0으로 초기화 하고 i 가 5보다 작을 경우 실행.  실행문이 끝나면 i를 1씩 증가시키고 다시 반복 실행.  조건식이 거짓(false)가 되면 중지. |
| --- |

| **실제 표현** | **예문** | **비고** |
| --- | --- | --- |
| do{  실행문  ...  }while(조건식) | var a = 1;  while(a<=10){  document.write("안녕하세요(while)"  +"<br>");  a++;  }    do{  alert(a);  }  while(false) | 아래와 같이 작성하면 무한 반복문. break; 구문과 함께 작성.  while(true){  실행문  }  ▶ 안녕하세요 10줄 출력됨  + while이 실패할 때 alert(11)가 뜸 |

#### **break / continue 문**

**break** : switch문과 반복문에서 사용.

반복문에서 이 명령을 만나면 반복문 다음 문장으로 빠져나감 (= 반복문 종료!)

**continue** : 반복문에서만 사용 가능.

반복문에서 이 명령을 만나면 반복문의 끝으로 이동합니다. (= 반복문 종료X !)

continue와 반복문 사이에 있는 실행문이 무시된다고 생각하면 됨.

| while문 또는 for문  {  반복중인 문장  ...  continue; // continue를 만나면 다시 반복문의 조건으로 돌아감  (현재 반복 횟수는 완료)  반복중인 문장  } |
| --- |

#### 

#### **▶ 함수**

하나의 특별한 목적의 작업을 수행하도록 설계된 독립적인 블록. 필요할 때마다 호출하여 해당 작업을 반복수행 할 수 있다.

\* 자바스크립트에서 블록이란 함수의 실행문의 중괄호({})로 묶여진 부븐을 가리킨다.

\*\* 자바스크립트에서는 함수도 하나의 타입(datatype)이다. 따라서 함수를 변수에 대입 하거나, 함수에 프로퍼티를 지정하는 것도 가능하며 다른 함수에 중첩하여 사용할 수도 있다.

함수의 구성요소:

1. 함수의 이름
2. 괄호 안에 쉼표(,)로 구분되는 함수의 매개변수
3. 중괄호 ({})로 둘러싸인 자바스크립트 실행문

| *function* 함수이름(매개변수1, 매개변수2,...) {  함수가 호출되었을 때 실행하고자 하는 실행문;  } |
| --- |

함수 선언방법

| "use strict";  //메소드()  function func1(){  console.log("func1을 호출하였습니다.")  }  const func2 = function (){  console.log("func2을 호출하였습니다.")  }  func1(); // 이게 함수 호출임  func1();  func1();  func2(); |
| --- |

함수를 선언해야 그 선언을 둘러싼 함수의 최상부나 전역범위로 끌어올려진다.

#### **매개변수**

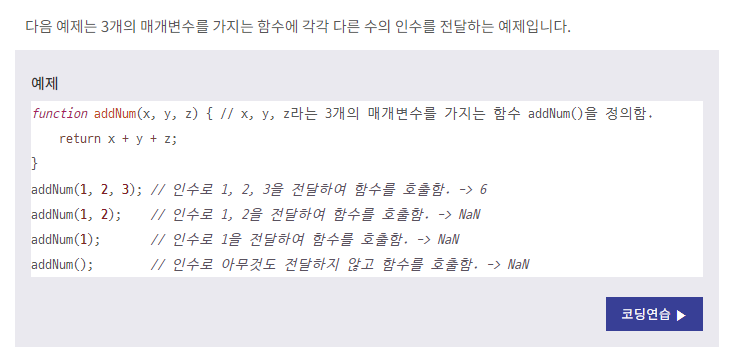
함수의 정의에서 전달받은 인수를 함수 내부로 전달하기 위해 사용하는 변수를 의미

인수: 함수가 호출될 때 함수로 값을 전달해주는 값

| 예문1  fun1 선언:  //메소드()  function func1(num){ // num이 매개변수  console.log(`전달받은 매개변수의 값은 ${num}입니다`)  fun1 활용:  func1(1000); |
| --- |

| 예문2  func2 선언:  function func2(start, end){  let sum = 0;  for (let i=start; i<=end; i++){  sum += i;  }  console.log(`${start}부터 ${end}까지의 총합은 ${sum}입니다`);  }  func2 활용:  func2(1, 100); |
| --- |

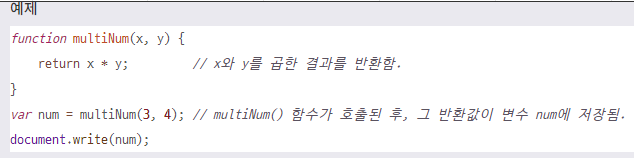
| 예문3  func3 선언:  function func3(num1=1, num2=1){  console.log(`${num1}+${num2}=${num1+num2}`);  console.log(`${num1}-${num2}=${num1-num2}`);  console.log(`${num1}\*${num2}=${num1\*num2}`);  console.log(`${num1}/${num2}=${num1/num2}`);  }  func3 활용:  func3(); |
| --- |



#### **반환(return)문**

#### 

| 자바스크립트에서 함수는 반환(return)문을 포함할 수 있다.  이러한 반환문을 통해 호출자는 함수에서 실행된 결과를 전달받을 수 있다.  반환문은 함수의 실행을 중단하고, return 키워드 다음에 명시된 표현식 값을 호출자에게 반환 |
| --- |
|



#### 

#### **화살표 함수**

함수 표현식보다 단순하고 간결한 문법으로 함수를 만들어준다.

() => {} 는 function(){} 와 동일

| $(document).ready(function(){  });  $(function(){  });  $(document).ready(() => {  });  // 위 세 개가 전부 다 같은 것!  //(단. 화살표는 자바스크립트에만 적용됨! 제이쿼리 적용XXX) |
| --- |

1. (매개변수) => {본문}

2. 매개변수가 하나뿐인 경우 괄호 생략 가능

3. 매개변수가 없을 경우 괄호가 필수

4. 본문이 return[식 or 값] 뿐일 경우 {}와 return 키워드 생략 가능

5. 위 4에서 return 할 값이 객체일 경우에는 괄호 필수

| this를 가지지 않는다..  arguments를 지원하지 않는다.  new와 함께 호출할 수 없다. |
| --- |
|

let func = function(arg1, arg2, ...argN) {

return expression;

};

위의 예문을

let func = (arg1, arg2, ...argN) => expression

처럼 축약할 수 있다.

| let sum = (a, b) => a + b;  위 화살표 함수는 아래 함수의 축약 버전입니다.  let sum = function(a, b) {  return a + b;  }; |
| --- |

#### **▶ 이벤트**

#### 이벤트 입니다!

#### **onclick**

| <input type="button" value="덧셈" **onclick="add();**"> |
| --- |

| **함수? 값?** | **직접 표현되는 값 그 자체** | |
| --- | --- | --- |
| onclick | 클릭할 때   | body  <div class="container">  <button id="call" onclick="writeTxt()">event call</button>  <textarea id="demo" class="note"></textarea>  </div>  script  function writeTxt(){ //onclick 이벤트  var area = document.querySelector(".note");  area.innerHTML = "Hello JavaScript"  }; | | --- | | body  <div class="container">  <button id="call" onclick="writeTxt('Hello JavaScript')">  event call  </button>  <textarea id="demo" class="note"></textarea>  </div>  script  function writeTxt(str){  var area = document.querySelector(".note");  area.innerHTML = str;  }; | | |
| onmouseover | hover | |
| onmouseout | 커서가 바깥으로 나갈때 | |
| onkeydown | 키보드 눌렀을때 | |
| onload | 브라우저 로딩이 끝났을때 | |

#### **addEventListener()**

#### 이벤트 대상이 수신할 때마다 호출할 함수를 설정

| addEventListener(type(이벤트유형), 이벤트(함수나 객체 등))  htmlLi.addEventListener("click", function(e){  var bool = e.target  bool.classList.toggle("act");  }); |
| --- |
|

#### **select\_onchange**

#### 내가 선택한 값이 달라졌을때 사용?? 이 값이 변했을때 이벤트 뽷

| <select name="" id="mySelect" onchange="myFuncion()">  <option value="Audi">Audi</option>  <option value="BMW">BMW</option>  <option value="Mercedes">Mercedes</option>  </select>  <script>  function myFuncion(){  var x= document. /\*문서안에서\*/ getElementById /\*개체를 가져와라, id인것을\*/ ("mySelect").value /\*현재값을 가져오는 것\*/;  }  </script> |
| --- |
|

#### **innerHTML**

#### innerHTML=" "

| 예를 들어 HTML로 [홍길동]이라는 콘텐츠를 화면에 출력하였다. 이 HTML 요소에 접근하여 [홍길동]을 [이순신]으로 바꿔 출력하게 만들려면 이 속성을 사용해야 한다. 그리고 HTML 요소에 접근하려면 document.getElementById 메서드를 함께 사용한다. |
| --- |
|

<p id="demo"></p>

<script>

function myFuncion(){

document.getElementById("demo") /\*문서에서 id가 demo인 것을 가져오라\*/ .innerHTML /\*그안에 html값을 넣는거\*/ = "You selected : " + x;

}

</script>

#### **마우스 이벤트**

#### 

#### HTML 마우스 이벤트입니다!

| <input type="button" value="덧셈" **onclick="add();**"> |
| --- |

| **이벤트** | **직접 표현되는 값 그 자체** | |
| --- | --- | --- |
| onmouseenter | 마우스 포인터가 요소 안으로 들어올 때(자식 포함) 발생하는 이벤트  요소 밖으로 나갔다 오지 않으면 계속 발생하지 않음 | |
| onmouseover | onmouseout 의 반대 개념  마우스 포인터가 요소 안으로 들어올 때 + 자식 요소 출입시 발생하는 이벤트  요소 영역을 벗어나지 않아도 자식 요소에 들어가거나 나올 때 발생함 | |
| onmouseout | onmouseover 의 반대 개념  마우스 포인터가 요소 밖으로 나갈 때 + 자식 요소 출입시 발생하는 이벤트  요소 영역을 벗어나지 않아도 자식 요소에 들어가거나 나올 때 발생함 | |
| onmousemove | 마우스 포인터가 요소 위에서 움직일 때(자식 포함) 발생하는 이벤트  요소 위에서 마우스를 움직이면 계속 발생함 | |

<span onmouseover="this.style.color='red'" onmouseout="this.style.color='black'">

Mouse over me!

</span>

> 마우스 오버할 경우 검은 글씨가 검정으로 바뀜

#### **Coordinates**

#### 커서가 있는 위치. 마우스 좌표.

| **이벤트** | **직접 표현되는 값 그 자체** | |
| --- | --- | --- |
| clientX  clientY | 클라이언트 영역 내의 가로,세로 좌표를 제공합니다.  여기서 클라이언트 영역은 현재 보이는 브라우저 화면이 기준이 됩니다.  clientX : 브라우저 페이지에서의 X좌표 위치를 반환하나 스크롤은 무시하고 해당 페이지의 상단을 0으로 측정합니다.  clientY : 브라우저 페이지에서의 Y좌표 위치를 반환하나 스크롤은 무시하고 해당 페이지의 상단을 0으로 측정합니다. | |
| offsetX  offsetY | 위 메서드는 이벤트 대상이 기준이 됩니다. ( 화면 중간에 있는 박스 내부에서 클릭한 위치를 찾을 때 해당 박스의 왼쪽 모서리 좌표가 0이됩니다. 화면의 기준이 아닙니다) 전체 문서를 기준으로 합니다(스크롤 화면 포함)  offsetX : 이벤트 대상 객체에서의 상대적 마우스 x좌표 위치를 반환합니다.  offsetY : 이벤트 대상 객체에서의 상대적 마우스 y좌표 위치를 반환합니다. | |
| pageX  pageY | 위 메서드는 전체 문서를 기준으로 x,y 좌표를 반환 합니다. 스크롤 화면을 포함해서 측정합니다.  pageX : 브라우저 페이지에서의 x좌표 위치를 반환합니다.  pageY : 브라우저 페이지에서의 Y좌표 위치를 반환합니다. | |
| screenX  screenY | 위 메서드는 모니터 화면을 기준으로 좌표를 제공합니다. 여기서 중요한 점은 브라우저 화면이 아니라 자신의 모니터 화면 전체를 기준으로 좌표를 측정한다는 점입니다.  screenX : 전체 모니터 스크린에서의 x좌표 위치를 반환합니다.  screenY :전체 모니터 스크린에서의 y좌표 위치를 반환합니다. | |

#### **indexOf()**

#### "문자열".indexOf("찾을 문자")

| 문자열에서 원하는 문자열을 검색하여 찾음  배열에서 원하는 특정 배열값의 존재여부 등을 확인  배열의 경우 위치값을 index로 반환 |
| --- |

var text = "123";

var findStr = "456"; // *123이 있는지 찾아보는 문자열 예시*

if (text.indexOf(findStr) != -1){

alert("Find!");

}

else{

alert("Not Found!");

}

| 456이 없으므로 Not Found!를 출력한다.  if 조건문에서 **-1의 값을 가지는가 여부를 확인**  indexOf()는 값을 찾고, 그 결과로 숫자를 반환하는데 없는 경우 -1을 반환함  있는 경우(존재하는 경우) 결과값으로 **문자열 시작 위치에 해당하는 index를 반환**  ! -1 이란?  ▶ 특정 문자열이 해당하는 텍스트 안에서 찾았다면 if 문에서 절대 -1이 될 수 없는 0이상의 양수 값이기 때문에, -1은 값이 없음을 의미한다. |
| --- |

#### **String**

#### 숫자를 문자열로 변환하는 함수

String(숫자 또는 숫자를 값으로 하는 변수)

| var x = 123;  x.toString(); // x는 이제 문자  var x = (123).toString; // 변수없이 숫자를 깡으로 바꿈 |
| --- |

| **함수** | **예시** | |
| --- | --- | --- |
| charAt() | 문자열에서 인자로 넘긴 index에 해당하는 문자형 데이터를 반환해주는 함수  var Sample = "Hello"  var result = Sample.charAt(1);  ▶ "e"를 반환한다. | |
| indexOf() | String.indexOf("찾을 문자", index);  문자열의 왼쪽부터 시작하여 찾을 문자와 일치하는 최초의 문자의 Index를 반환하는 함수. 두 번째 인자를 지정 하여 검색하는 시작 Index도 지정가능.  일치하는 문자를 찾지 못하면 -1을 반환 | |
| lastIndexOf() | String.lastIndexOf("찾을 문자");  IndexOf()와 비슷한 기능을 제공하지만, 차이점은 찾을 문자를 문자열 오른쪽에서부터 찾는다.  일치하는 문자를 찾지 못하면 -1을 반환. | |
| replace() | String.replace("찾을 문자", "치환할 문자");  문자열의 왼쪽에서부터 찾을 문자를 찾아 최초에 일치하는 문자에 대하여 두 번째 인자로 넘긴 치환할 문자로 대체. 문자열 왼쪽에서부터 최초로 발견된 일치하는 문자열에 대해서만 치환이 이루어짐. | |
| length | 문자열의 전체 길이에 대한 속성값  var sample = "Hello"  document.write(sample.length); //5 | |

#### **Number**

#### 문자열을 숫자로 변환하는 함수

| Number(문자열 또는 문자열을 값으로 하는 변수 등을 입력)  ▶ 숫자로 변환할 수 없는 값인 경우 NAN을 반환 |
| --- |

#### **toPrecision()**

#### 수의 길이를 제한한다.toFixed와 비슷

| var x = 9.565;  x.toPrecision(); // 9.565  x.toPrecision(2); // 9.7 (반올림)  x.toPrecision(4); // 9.656 (반올림) |
| --- |

#### **toPrecision()**

#### 소수점 자리수를 제한. toPrecision()과 비슷

| var y = 9.565;  x.toFixed(); // 9.565    y.toFixed(2); // 9.56 (반올림)  alert(y.toFixed(2)); // 단일성이라 alert(y)로 하면 기본것만 나옵니다!!  x.toFixed(4); // 9.656 (반올림) |
| --- |

#### **parselnt()**

#### 문자열을 정수로 바꾸는 함수

| parseInt( string, n )  string을 n진법일 때의 값으로 바꿉니다. n은 옵션으로 2부터 36까지 입력할 수 있습니다. 입력하지 않으면 10으로 처리합니다. string의 처리는 parseFloat()와 거의 같습니다. 소수 부분은 버립니다. 0x로 시작하면 16진법으로 처리합니다. |
| --- |

| parseInt("12.68"); //12  parseInt("100", 10); //100  parseInt("100", 2); // 4  parseInt("0x100"); 256 |
| --- |

#### **Math.random()**

#### 0.0과 같거나 크고, 1.0보다 작은 값을 출력해낸다.

Math.floor(Math.random() \* 6 )+ 1; //1~6사이의 랜덤 주사위, floor: 내림

| parseInt( string, n )  string을 n진법일 때의 값으로 바꿉니다. n은 옵션으로 2부터 36까지 입력할 수 있습니다. 입력하지 않으면 10으로 처리합니다. string의 처리는 parseFloat()와 거의 같습니다. 소수 부분은 버립니다. 0x로 시작하면 16진법으로 처리합니다. |
| --- |

#### **Date**

#### 인수로 년, 월, 일, 시, 분, 초, 밀리초를 의미하는 숫자를 전달하면 지정된 날짜와 시간을 가지는 인스턴스를 반환한다. 이때 년, 월은 반드시 지정하여야 한다. 지정하지 않은 옵션 정보는 0 또는 1으로 초기화된다????

| var d = new Date(99, 5, 24, 11, 33, 30, 0); // 년+달\_일+시+분+초\_밀리초  document.getElementById("demo1").innerHTML = d;    var a = new Date(); // 오늘날짜  a.getDate(); // 요일을 현지 시각 또는 국제 표준시로 변환  document.getElementById("demo2").innerHTML = a; |
| --- |