**DB 연결 웹 앱**

**\* 자바스크립트 변수-연산자-타입**

변수는 var, let, const 키워드로 선언할 수 있다. 어떤 키워드를 사용하는가에 의해서 변수의 유효범위(scope)가 달라진다.

|  |  |
| --- | --- |
| **var** | var a= 2;  var a= “I love you”  var a= true;  var a= []; // 배열  var a= {}; // 오브젝트  var a= undefined; |
| **let** | ES6 버전 |
| **const** | ES6 버전. 자바의 상수라고 생각하면 된다 |

자바 스크립트에서의 연산자는 일반적인 사칙연산, MOD 등등 자바와 비슷하다. 자바 스크립트에서 OR 연산자는 조금 독특하다.

const name= “crong”;

/\* name에 값이 할당되어 있으면 name을 쓰고 아니면 오른쪽 값을 사용한다. \*/

const myname= name || “defaultName”

**\* 비교 연산자 주의할 점**

자바 스크립트에서는 일반적으로 == 보다는 ===을 많이 사용한다. 여러가지 문제점을 야기할 수 있기 때문이다. 아래 예를 보면,

0 == false, 0 == “”, 0 == “0”, null == undefined 의 결과가 모두 true가 나온다. 자바 스크립트에선 타입까지 정확하게 비교하기 위해서는 === 를 사용한다. == 를 사용하면 자바 스크립트가 암묵적으로 타입을 자동 변경하여 비교하게 된다.

**\* 자바 스크립트의 타입**

자바 스크립트의 타입은 선언 시점이 아닌 실행 시점에 결정된다. 타입을 체크하는 또렷한 방법은 없고 typeof, isArray 함수를 호출하여 타입을 체크할 수 있다.

|  |
| --- |
| **undefined** |
| **null** |
| **boolean** |
| **number** |
| **string** |
| **object** |
| **function** |

**\* 자바 스크립트 비교-반복-문자열**

|  |  |
| --- | --- |
| **if, else if, else** | if 대신 삼항 연산자( a ? “true”:”false”)도 많이 사용 |
| **switch** | 자바와 마찬가지로 case, default로 구성. break문 사용 |
| **for** | 자바와 동일  for(var i=0, len=arr.length; i<len; i++) { } |
| **foreach** | 배열은 foreach를 지원한다. |
| **for-of** | 새로 나온 반복문으로 버전 이슈가 있을 수 있으므로 사용에 주의 필요 |

**\* 문자열 처리**

자바스크립트에서 문자열과 문자는 동일한 개념으로 사용된다. 즉, typeof “abc”; 와 typeof ‘a’;의 결과 모두 string이 나온다.

자바스크립트에서 문자열은 자바와 마찬가지로 문자열 객체이고 때문에 split, replace, trim과 같은 메소드가 존재한다.

**\* 자바스크립트 함수**

자바스크립트에서의 함수는 여러개의 인자를 받아서 그 결과를 출력한다. 아래는 함수를 선언한 예제 소스이다.

|  |
| --- |
| function printName(firstname) {  reutnr “name is “ + firstname;  }  console.log(printName()); // 결과는 name is undefined 라고 나옴..  console.log(printName(“Jisu”, “Crong”)); // name is Jisu 라고 나옴.. |

자바스크립트에서는 함수 명세와 다른 인자를 전달하며 함수를 호출 한다고 해서 함수가 없다고 에러가 나는게 아니라, 적절한 처리를 하여 (인자가 부족한 경우 파라미터에 undefined를 넣어서 등등) 호출하게 된다.

다음으로 함수 표현식이라는 개념이 있다. 변수 값에 함수 선언문을 담아 놓은 것을 의미한다.

|  |
| --- |
| function printName(firstname) {  console.log(“name is “ + firstname);  var inner= function() {  return “inner value”;  }  return inner(); // inner 호출이 inner의 선언보다 먼저 나오면 에러 발생  } |

하지만 아래와 같이 함수 표현식을 함수 선언문으로 변경하면 그 순서가 바뀌어도 에러가 발생 안하고 동작할 수 있다.

|  |
| --- |
| function printName(firstname) {  console.log(inner()); // 선언보다 먼저 호출 되지만 에러 발생 안함.  function inner() {  return “inner value”;  }  } |

그렇다면 왜 함수 선언문에서는 에러가 발생 안하고 함수 표현식에서는 에러가 발생할까? 그 이유는 자바스크립트가 동작전 파싱할 때 수행하는 호이스팅 이라는 개념 때문이다. 호이스팅이란 자바스크립트 함수 실행 전 파싱하며 간단히 정보를 모아 함수 안의 변수들을 함수 바디의 맨 위로 올리는 것을 의미한다. 즉 왼쪽 함수가 오른쪽과 같이 변경되고 변수 a에는 아직 함수가 정의 안되어 undefined 에러가 발생하게 되는 것이다.

|  |  |
| --- | --- |
| function printName(firstName) {  console.log(inner());  console.log(a);  function inner() {  return “Jisu”;  }  var a= function() {  return “Crong”;  }  } | function printName(firstName) {  function inner() {  return “Jisu”;  }  var a;  console.log(inner());  console.log(a);  a= function() {  return “Crong”;  }  } |

자바스크립트에선 return을 안하면 함수 호출 시 그 결과로 undefined가 리턴된다.

**\* 함수 arguments 속성**

함수가 실행되면 그 안에서 arguments라는 특별한 지역변수가 생성된다. 자바스크립트 함수는 선언한 파라미터보다 더 많은 인자를 보낼 수 있는데 이 때 넘어온 인자를 arguments로 배열의 형태로 하나씩 접근할 수가 있다. 단 arguments는 배열 타입은 아니기 때문에 배열의 메서드를 사용할 수는 없다.

|  |
| --- |
| function printArgs() {  for(var i=0; i<arguments.length; i++) {  console.log(arguments[i]);  }  }  printArgs(1,2,3,4); |

**\* 자바스크립트 함수 호출 스택**