**자바 스크립트 위치**

1. 내부 자바스크립트
   1. <haed> 섹션에 <script>~</script> 섹션을 넣어서 그 안에 자바스크립트 작성
2. 외부 자바스크립트
   1. <head> 섹션에 <script src=”파일주소”></script> 작성. 파일은 .js 파일이어야한다.
3. 인라인 자바스크립트
   1. HTML 태그 내부에 이벤트 속성으로 삽입될 수 있음. 꼭 필요할 때만 사용.
      1. EX) <button type=”button” onclick=”alert(‘반갑습니다.’)”>~~~</button>
4. 자바스크립트로 콘텐츠를 처리할 땐, <body>의 맨 끝에 넣어야한다.
   1. 자바스크립트는 읽혀지면서 바로 실행되므로. <head>나 <body>맨끝에 넣어야함.

**대소문자와 공백**

1. 자바스크립트는 대소문자를 구별함. 따라서 자바스크립트에서 HTML의 태그와 속성이름은 반드시 소문자로 적어줘야한다.
2. 자바스크립트에서 공백은 무시한다.

**변수**

var x = 10; // 전역변수

var name = “Shin”, age = 23, job=”student”; //한줄에 여러개 선언 가능

**변수이름 규칙**

1. 변수이름은 문자로 시작한다.
2. $나 \_로 시작할 수 있다
3. 변수이름은 대소문자를 구분한다

**변수 자료형 :** 수치형, 문자형, 부울형, 객체형, undefined

1. 변수 자료형 아는 법 : typeof(변수)
   1. 수치형 : “number” / 문자열 : “string” / 부울형 : “boolean” / 호출이 불가능한 객체 : “object” / 호출이 가능한 객체 : “function” / Null : “object” / Undefined : “undefined”
2. 객체형 : var myCar = {model : “bmz”, color:”red”, hp:100} / myCar.model 처럼 쓴다.

**변수 기타 규칙**

1. 문자열을 나타낼 때는 “”, ‘’ 모두 사용 가능하다.

**연산자**

1. 다른 건 다 같음
2. === : 값과 타입이 모두 같으면 참 / !== : 값 타입이 모두 같으면 true, 그외 false

**숫자와 문자열 사이의 변환**

1. parseInt(문자열) : 수치형으로 변경
2. String(수치형) : 문자열로 바뀜.

**조건문 :** 같음

**반복문**

1. 나머진 다 같음
2. for(var i in myCar) : 자바의 for(int I : myCar) 와 같음. 단, 자료형관계없이 쓸 수 있다.

**배열**

1. 리터럴로 배열 생성
   1. var fruits = [“apple”, “banana”, “peach”];
2. Array 객체로 생성
   1. var fruits = new Array(); // var fruits = new Array(“Apple”, “banana”) 도 물론 가능
   2. fruits[0] = “Apple” 로 값 추가.
3. Array.length 로 길이 확인가능
4. 키를 이용해 값을 저장할 수도 있다.
   1. fruits[“a”] = “Apple”; / fruits[“b”] = “Banana” / fruits[“p”] = “Peach”

**함수**

1. function 함수이름 () { } 로 나타내고 함수이름을 통해 호출할 수 있다.
2. 외부에서 버튼을 누르는 등의 행동이 발생하면 fucntion 호출 가능
3. () 부분엔 인수를 넣을 수 있음.(자료형 안써도 됨)
4. 무명함수
   1. var greeting = function() { alert(“안녕하세요”); };
   2. 호출할 땐 greeting() 로 가능
   3. 이벤트 처리함수를 작성할 때 많이 사용됨.
5. 다른 언어와 마찬가지로 반환가능.
6. 함수 내부 – 지역변수 / 함수 외부 – 전역변수

**입출력**

1. prompt(“출력메세지”, “입력창에 뜨는 문자열”); : 입력 메시지를 문자열로 반환
   1. input = prompt(“정수를 입력하세요”, “정수로”);
2. alert(문자열) : 경고 메시지를 화면에 띄우는 것.
3. confirm(문자열) : 확인을 누르면 true, 취소를 누르면 false 를 반환

**객체**

객체 종류

1. 내장객체 : 생성자가 미리 작성되어 있다.
2. 사용자 정의 객체 : 사용자가 생성자를 정의한다.

객체 생성

1. 객체 상수로부터 객체 생성 : 객체를 딱 하나만 사용(임시 객체 같은 느낌)
   1. var myCar {  
       model : “520d”,  
       speed : 60,  
       color : “red”,  
       brake : function() {this.speed -= 10; },  
       accel : function () {this.speed +=10;}  
      };
   2. 사용은 myCar.model / myCar.brake() 처럼 사용하면 된다.
2. 생성자 함수 사용
   1. function Car(model, speed, color) {  
       this.model = model;  
       this.speed = speed;  
       this.color = color;  
       this.brake = function() {  
       this.speed -= 10;  
       }  
       this.accel = function() {  
       this.speed += 10;  
       }  
      }
   2. this.가 안붙으면 지역변수와 헷갈릴 수 있으므로 웬만하면 붙이자
   3. var myCar = new Car(“520d”, 60, “white”); 으로 생성
   4. 따로 생성자를 만들어줄 필요는 없다.
   5. class로 선언하는 것이 아닌 function으로 선언하여 함수처럼 다루자.
   6. 클래스 대신 prototype이라는 것을 클래스처럼 사용할 수 있다.

**프로토타입**

1. 위의 객체 섹션처럼 객체를 선언하면, Car를 100개 선언하면 100개의 변수에 모두 메소드가 들어감.
2. 이를 방지하기위해 객체는 변수만 놔두고, 메소드는 공유 메소드를 만드는 게 프로토타입
3. function Point(xpos, ypos) {  
    this.x = xpos;  
    this.y = ypos;  
    }  
   Point.prototype.getDistance = function() {  
    return Math.sqrt(this.x\*this.x + this.y\*this.y);  
   };
4. 위처럼 선언하면, 아무리 많은 객체를 선언하여도 메소드는 하나만 사용된다.
5. 물론 변수도 위처럼 프로토타입으로 선언할 수 있다.(전역 상수변수)

**Object 객체**

1. 모든 객체의 기본. object 내 객체의 속성/메소드를 재정의하여 사용할 수 있다.
2. 속성/메소드  
   constructor 속성으로 생성자 함수를 가리킴.   
    var d = new Date();  
    d.constructor는 Date()와 같다.  
   valueOf() 메소드로서 객체를 숫자로 변환한다  
   toString() 메소드로서 객체의 값을 문자열로 변환한다.  
   hasOwnProperty() 전달 인수로 주어진 속성을 가지고 있으면 true 반환  
   isPrototypeOf() 현재 객체가 전달 인수로 주어진 객체의 프로토타입이면 true 반환
3. 재정의 할땐 그냥  
   myCar.toString = function() {  
    return “새로 생성된 객체입니다.”;  
   }  
   로 하면 된다.

**내장 객체**

1. Date 객체

생성자

new Date() // 현재 날짜와 시간

new Date(milliseconds) // 1970/01/01 이후의 밀리초

new Date(dateString) // 다양한 문자열

new Date(year, month, date[, hours[, minutes[, seconds[, ms]]]])

메소드

책 참고

1. Number 객체

-수치형 값을 감싸서 객체로 만들어주는 랩퍼 객체

-생성 : var num = new Number(7);

1.234.toString(); // 자동으로 랩퍼 생성. 1.234를 감쌈.

-속성값 : MAX VALUE – 표현할 수 있는 가장 큰 값

MIN VALUE – 표현할 수 있는 가장 작은 값

NaN – Not a Number 의 약자

-메소드 :

toExponential([digits]) : 지수형으로 반환. 인수는 소수점 이하 숫자의 개수

1232.34567.toExponential() -> 1.23234567e+3

1232.34567.toExponential(1) -> 1.2e+3

toFixed([digits]) : 고정소수점 방식으로 반환, 인수는 소수점 이하의 숫자의 개수

123.456789.toFixed() -> 123

123.456789.toFixed(1) -> 123.5

toPrecision([precision]) – 유효숫자수를 지정한다.

123.456789.toPrecision(1) -> 1e+2

123.456789.toPrecision(2) -> 1.2e+2

toString([radix]) – 주어진 진법으로 숫자를 반환함

1. String 객체
   1. .length : 문자열의 길이
   2. 메소드 : toUpperCase() / toLowerCase() / s1.concat(s2) / indexOf(str) / match(/~/opt)

replace(s1, s2) / split(delimiter)

1. Math 객체 : 숫자관련 상수 및 메소드. 자세한건 책 참조
2. Array 객체
   1. 선언 : var myArray = new Array();
   2. 배열의 크기는 자동으로 조절됨
      1. myArray[0] = “apple”;
      2. myArray[99] = “banana”; // 크기가 100으로 변함.
   3. 하나의 배열에 여러 자료형을 혼합하여 사용가능하다.
      1. myArray[4] = new Date();
      2. myArray[6] = 2.34;
   4. 배열의 크기보다 큰 인덱스 값으로 배열 요소에 접근하면 오류가 발생하지 않고 undefined 값이

반환된다.

**오류처리**

1. 실행 중 오류(예외)가 발생하면 오류 대화상자가 뜨고, 그걸 보고 고치면 됨
2. try catch 문으로 예외 처리 해준다.
   1. catch(error) 로 받으면 됨 / error.message 안에 오류메시지가 들어있음
   2. throw 문으로 날릴 수도 있음

**메소드**

1. document.write(문자열); : 기존의 HTML 내용을 지우고 그 위에 넣은 문자열 출력.
   1. 끝에 “<br>” 를 추가하면 줄바꿈 가능 / document.write(myCar.model + “<br>”);
2. document.getElementById(id) : id에 해당하는 요소를 가져옴
3. document.getElementById(id).value : id에 해당하는 요소의 값을 가져옴.