객체의 메서드

- 메서드는 객체가 가지고 있는 동작
- 수행하기 위해서는 객체를 통해서 해당 메서드 수행
- 함수와 메서드의 차이
  - 함수 : 그 동작을 수행하기 위해 객체에게 어떤 동작을 수행하라고 명령하지 않아도 됨, 함수자체가 그 동작을 정의한 함수객체이기 때문에 자기 자신을 수행하는 것

```
메서드명 : function() {
    실행명령 1;

    return 리턴값;
}
```

DOM을 사용하는 이벤트 처리기 등록

window.onload 사용

- DOM에서 이벤트처리기 등록하는 가장 큰 목적은 **HTML 코드와 자바스크립트 코드의 분리**
  - script요소를 head요소의 자식요소로 배치
  - DOM 사용시, **body 요소의 바깥에서 body 요소 안에 있는 HTML요소를 조작**할 수 있음
- 이벤트 처리기를 등록하는 작업의 실행 시점은 **HTML 문서 전체를 읽어들인 이후로**
  - window 객체의 onload 프로퍼티에 이벤트 처리기를 등록하는 작업을 수행하는 초기 설정 함수 정의 필요

```
window.onload = function() {...};
```

- 이 코드에 의해 웹브라우저가 HTML 문서 전체를 **모두 읽어들인 후, 우변의 함수 실행**

Document.getElementById()

- **주어진 문자열과 일치하는 id속성을 가진 요소를 찾고, Element 객체 반환**
  - ID는 문서 내에서 **유니크해야함**
  - ID가 없는 요소에 접근하려면, Document.querySelector()를 사용

```
document.getElementById(id);
```

이벤트 처리기 프로퍼티에 동작함수 등록

- 이벤트 처리를 위한 동작을 수행할 이벤트 처리기 등록

```
button.onclick = displayTime; // displayTime 함수를 onclick 프로퍼티에 등록
```

HTML요소의 <INPUT>, <OUTPUT> 태그 사용

- <input> 태그는 사용자에게 입력을 받을 수 있는 필드를 생성, </input> 태그는 없음

    - type (필수속성)
        - 어떤 타입으로 입력을 받을지 속성값으로 결정
        - text, password, radio, checkbox 등
        - number 지정시 숫자만 입력 가능(다른 타입은 표시 X)
    - value : 입력태그의 초기값을 설정
    - id : 하나의 페이지에서 유일한 값으로, 화면을 구성하는 모든 것들에게 따로따로 접근할 때 이용
- <output> 태그는 스크립트 등에 의해 수행된 계산의 결과나
  사용자의 액션에 의한 결과를 나타낼 때 사용

document.getElementById().value

- 웹 페이지에 있는 엘리먼트의 객체를 가져와,
  데이터를 직접 잡아내는 대신 자바스크립트 객체를 사용해서 **HTML 필드자체를 제공**
    - value 프로퍼티를 통해 데이터에 접근 가능

```
var h = document.getElementById("height").value;
```

innerHTML 프로퍼티 사용

- 요소객체의 innerHTML 프로퍼티
    - 그 HTML요소의 내용을 가리키며, 해당 요소의 내용을 읽거나 쓸 수 있다

```
var bmi = document.getElementById("bmi");
bmi.innerHTML = (w/h/h).toFixed(1);
```

ES2015+

- ES2015 이전에는 var로 변수를 선언
    - ES2015 부터는 const와 let이 대체
        - 차이점 : **블록 스코프** (var은 **함수 스코프**)
- const : 상수
    - 상수에 할당한 값은 다른 값으로 변경 불가
    - 변경하고 할 때는 let으로 변수 선언
    - 선언 시에 초기화 필요
    - 초기화 하지 않고 선언시 에러

템플릿 문자열

- 문자열을 합칠 때 + 기호때문에 지저분한 점 개선
    - ES2015부터 `(백틱) 사용 가능
    - 백틱 문자열 안에 ${변수}처럼 사용

```
const string2 = `${num3} 더하기 ${num4}는 ${result}`;
```

객체 리터럴

- ES5 시절의 객체 표현 방법은 속성 표현 방식에 주목

- ES5+는 훨씬 간결한 문법으로 객체 리터럴 표현
  - 객체의 메서드에 : function을 붙이지 않아도 됨
  - sayNode : sayNode ▶ sayNode로 축약 가능
  - [변수 + 값] 등으로 동적 속성명을 객체 속성명으로 사용 가능

```
const newobject = {
    sayJS() {
        console.log('JS');
    },
    sayNode,
    [es + 6]: 'Fantastic',
};
```

화살표 함수

- function 선언 대신 => 기호로 함수 선언

```
function add1(x, y) {
  return x + y;
}

const add2 = (x, y) => {
  return x + y;
};

const add3 = (x, y) => x + y;

const add4 = (x, y) => (x + y);

function not1(x) {
  return !x;
}

const not2 = x => !x;
```

  - not1, 2도 같은 기능
- 기존 function() {}를 대체하는 것은 아님
  - 중간 변수를 이용해 전달하는 과정 필요

```
var relationship1 = {
  name: 'zero',
  friends: ['nero', 'hero', 'xero'],
  logFriends: function() {
    var that = this; // relationship1을 가리키는 this를 that에 저장
    this.friends.forEach(function(friend) {
      console.log(that.name, friend);
    });
  },
};
relationship1.logFriends();
```