함수
- 함수의 형식 -

함수의 형식

❖ **함수란?**

- 코드의 집합

- 함수의 정의

```
function 함수명(인수목록) {
        본체
}
```

-

```
함수명(인수목록);
```

함수의 형식

❖ 선언적 함수

- ㅇ 선언적 함수 생성
  - ▪ 함수 정의시 이름 배정

```
> function fnName() {
   console.log('Hello javascript')
}

> fnName();
Hello javascript
```

함수의 형식

❖ 선언적 함수

- ㅇ 선언적 함수의 재정의

```
function fn() {
   console.log('Hello javascript 1')
}
function fn() {
   console.log('Hello javascript 2')
}

> fn()
'Hello javascript 2
```

함수의 형식

❖ 선언적 함수

- ㅇ 선언적 함수의 재정의
  - 먼저 정의 후 호출해야 함
  - 정의되지 않은 함수 호출 시 에러 발생

```
> fn2()
ReferenceError: fn2 is not defined

function fn2() {
   console.log('Hello javascript 1')
}
function fn2() {
   console.log('Hello javascript 2')
}
```

함수의 형식

❖ **05_1_01_function.html**

```
<body>
   <script>
      function add(a, b) {
         return a + b;
      }

      document.write("2 + 3 = " + add(2, 3) + "<br>");
      document.write("java + script = " + add("java", "script") +
                     "<br>");
   </script>
</body>
```

함수의 형식

❖ 05_1_02_sum.html

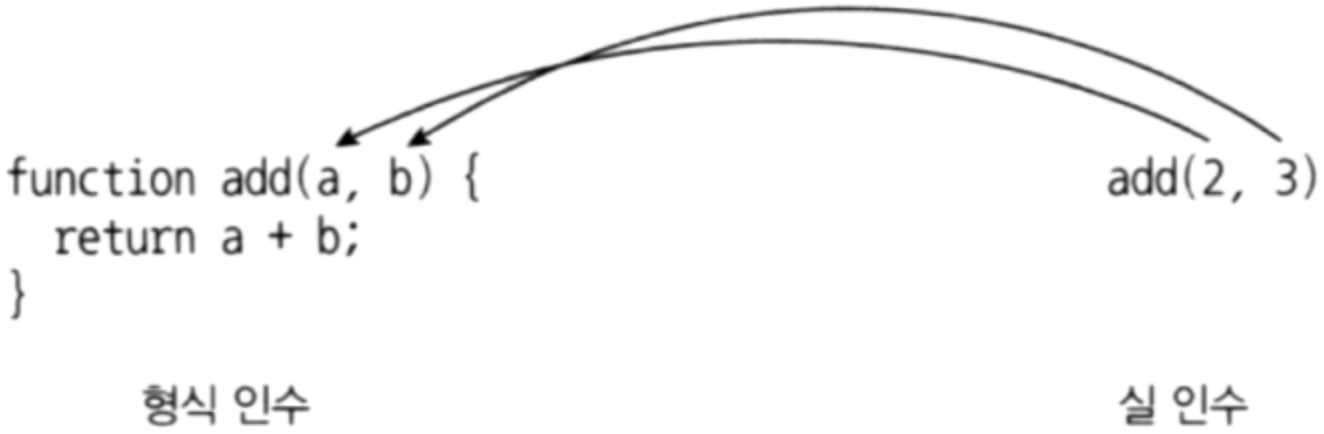
```
<body>
   <script>
      function sum(n) {
         var s = 0;
         for (var i = 1; i <= n; i++) {
            s += i;
         }
         return s;
      }

      document.write("1~100 = " + sum(100) + "<br>");
      document.write("1~200 = " + sum(200) + "<br>");
   </script>
</body>
```

함수의 형식

❖ **인수(매개변수)**

ㅇ 함수 호출과 함수 연결의 매개가 되는 변수



함수의 형식

❖ 매개변수

- 함수 생성 시 지정한 매개 변수의 수가 많거나 적은 사용도 허용
- 지정하지 않은 매개변수는 undefined로 입력

```
function fn(a, b, c) {
   console.log(a, b, c);
}
> fn()
undefined undefined undefined

> fn(10)
10 undefined undefined

> fn(10, 20)
10 20 undefined

> fn(10, 20, 30)
10 20 30
```

함수의 형식

❖ 05_1_03_noargument.html

```
<body>
   <script>
      function hello() {
         document.write("안녕하세요. <br>");
         document.write("좋은 아침입니다. <br>");
      }
      hello();
      hello();
   </script>
</body>
```

함수의 형식

❖ **05_1_04_extraargument.html**

```
<body>
   <script>
      function add(a, b) {
         return a + b;
      }
      document.write(add(2, 3) + "<br>");
      document.write(add(2, 3, 4) + "<br>");
      document.write(add(2) + "<br>");
   </script>
</body>
```

함수의 형식

❖ **05_1_05_defaultargument.html**

```
<body>
   <script>
      function sum(n) {
         if (n == undefined) n = 100; // n = || 100;
         var s = 0;
         for (var i = 0; i <= n; i++) {
            s += i;
         }
         return s;
      }
      document.write("1~10 = " + sum(10) + "<br>");
      document.write("1~100 = " + sum() + "<br>");
   </script>
</body>
```

함수의 형식

❖ 가변 인자 함수란?

- 매개변수의 개수가 변할 수 있는 함수
- 매개변수가 선언된 형태와 다르게 사용했을 때 매개변수를 모두 활용하는 함수를 의미
- 가변인자 함수의 예 : `Array( )` 함수

함수 형태	설명
Array( )	빈 배열을 만듭니다.
Array(number)	매개변수만큼의 크기를 가지는 배열을 만듭니다.
Array(any, ... , any)	매개변수를 배열로 만듭니다.

함수의 형식

❖ sumAll( ) 함수

- arguments
  - 자바스크립트 내부 변수의 기본으로 제공
  - 매개변수의 배열

```
function sumAll() {
   var sum = 0;

   for(var i=0; i<arguments.length; i++) {
      sum += arguments[i];
   }
   return sum;
}

console.log(sumAll(1,2,3,4,5,6,7,8,9))
```

```
for (var i in arguments) { }
```

함수의 형식

❖ **05_1_06_arguments.html**

```
<body>
   <script>
      function total() {
         var s = 0;
         for (var i = 0; i < arguments.length; i++) {
            s += arguments[i];
         }
         return s;
      }
      document.write(total(2, 5, 3) + "<br>");
      document.write(total(1, 5, 8, 8, 12, 14) + "<br>");
   </script>
</body>
```

함수의 형식

❖ **05_1_06_arguments2.html**

```
<body>
   <script>
      function total() {
         var s = 0;
         if (typeof(arguments[0]) == "string") {
            s = "";
         }
         for (var i = 0; i < arguments.length; i++) {
            s += arguments[i];
         }
         return s;
      }
      document.write(total(1, 2, 3) + "<br>");
      document.write(total("니들이", " 게맛을", " 알어?") + "<br>");
   </script>
</body>
```

함수의 형식

❖ **05_1_07_callby.html**

```
<body>
   <script>
      function byvalue(a) {
         a = 9999;
      }
      function byref(a) {
         a[0] = 9999;
      }

      var int = 1000;
      var ar = [1000, 2000, 3000];

      document.write("int = " + int + ", ar[0] = " + ar[0] +
                     "<br>");
      byvalue(int);
      byref(ar);
      document.write("int = " + int + ", ar[0] = " + ar[0] +
                     "<br>");
   </script>
</body>
```

함수의 형식

❖ 리턴값

- 리턴 키워드
  - 함수 실행 중 함수를 호출한 곳으로 돌아가라는 의미
  - 리턴 타입을 지정하지 않음
  - 리턴 값이 없는 경우 undefined 리턴

```
function sumAll() {
   var sum = 0;

   for(var i=0; i<arguments.length; i++) {
      sum += arguments[i];
   }
}

console.log(sumAll(1,2,3,4,5,6,7,8,9))
```

함수의 형식

❖ **05_1_08_return.html**

```
<body>
   <script>
      function add(a, b) {
         return a + b;
      }
      var c = 7 - add(2, 3);
      document.write(c);
   </script>
</body>
```

함수의 형식

❖ **05_1_08_return2.html**

```
<body>
   <script>
      function sum(n) {
         if (n < 0) return;
         var s = 0;
         for (var i = 0; i <= n; i++) {
            s += i;
         }
         return s;
      }
      document.write("1~100 = " + sum(100) + "<br>");
      document.write("1~-5 = " + sum(-5) + "<br>");
   </script>
</body>
```

함수
- 함수 고급 -

함수 고급

❖ 내부 함수

- 프로그램 개발 시 일어나는 네임 충돌을 막는 방법
- 내부 함수는 함수 내부에 선언

```
function 외부 함수() {
    function 내부 함수1() {
        // 함수 코드
    }

    function 내부 함수2() {
        // 함수 코드
    }

    // 함수 코드
}
```

함수 고급

❖ 내부 함수 이용으로 함수 충돌을 막는 법

- 내부 함수 사용 시 내부 함수 우선

```
function pythagoras(width, height) {
    function square(x) {
        return x * x;
    }

    return Math.sqrt(square(width) + square(height));
}
```

- 외부에서는 내부 함수를 호출 할 수 없음

❖ 05_2_01_nestfunction.html

```
<body>
   <script>
      function add(a, b) {
         return a + b;
      }
      function sum(n) {
         var s = 0;
         for (var i = 0; i <= n; i++) {
            s = add(s, i);
         }
         return s;
      }
      document.write("1~100 = " + sum(100) + "<br>");
   </script>
</body>
```

❖ 05_2_01_nestfunction2.html

```
<body>
   <script>
      function sum(n) {
         function add(a, b) {
            return a + b;
         }
         var s = 0;
         for (var i = 0; i <= n; i++) {
            s = add(s, i);
         }
         return s;
      }
      document.write("1~100 = " + sum(100) + "<br>");
      document.write("2 + 3 = " + add(2 + 3) + "<br>");      // 에러
   </script>
</body>
```

❖ 05_2_01_nestfunction3.html

```
<body>
   <script>
      function outer() {
         var outvalue = 5678;
         function inner() {
            var invalue = 1234;
            document.write("outvalue = " + outvalue + "<br>");
         }
         inner();
         document.write("invalue = " + invalue + "<br>");    // 에러
      }
      outer();
   </script>
</body>
```

함수 고급

❖ 익명 함수

- 이름을 가지지 않는 함수
  - 변수에 익명 함수에 대한 참조를 저장하여 사용

function(인수목록) { 본체 }

- 함수 호출 : 함수 참조(함수명)뒤에 괄호표기후 코드를 실행

```
> var fn = function() {
   console.log('Hello javascript')
}

> console.log(fn);
[Function]

> fn();
Hello javascript
```

❖ 05_2_02_funcliteral.html

```
<body>
   <script>
      var add = function(a, b) {
         return a + b;
      };
      document.write("2 + 3 = " + add(2, 3));
   </script>
</body>
```

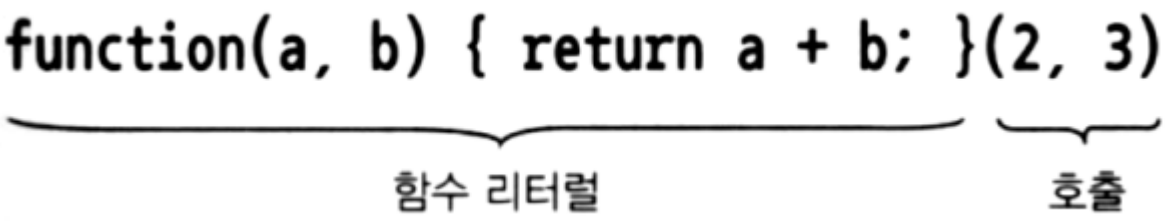
❖ 05_2_02_funcliteral2.html

```
<body>
   <script>
      document.write("2 + 3 = " + add1(2, 3) + "<br>");
      document.write("4 + 5 = " + add2(4, 5) + "<br>");   // 에러

      function add1(a, b) { return a + b; }
      var add2 = function(a, b) { return a + b; };
   </script>
</body>
```

❖ 05_2_02_funcliteral3.html

```
<body>
   <script>
      document.write("2 + 3 = " +
                  function(a, b) { return a + b; }(2, 3));
   </script>
</body>
```



❖ 05_2_03_assignfunc.html

```
<body>
   <script>
      var add = function(a, b) {
         return a + b;
      }
      var plus = add;
      document.write("2 + 3 = " + plus(2, 3));
   </script>
</body>
```

❖ 함수를 매개변수로 전달하기

- 함수적 프로그래밍

❖ 05_2_04_funcargument.html

```
<body>
   <script>
      var add = function(a, b) {
         return a + b;
      }
      var multi = function(a, b) {
         return a * b;
      }
      function calc(a, b, f) {
         return f(a, b);
      }
      document.write("2 + 3 = " + calc(2, 3, add) +"<br>");
      document.write("2 * 3 = " + calc(2, 3, multi) +"<br>");
   </script>
</body>
```

❖ 함수를 리턴하는 함수

- 함수를 리턴하는 함수의 사용은 클로저 때문임

```
function outer() {
   return function() {
      console.log('Hello Function...!');
   };
}

// 호출 1
outer()();

// 호출 2
var fn = outer();
fn();
```

함수 고급

❖ 클로저

- 지역 변수의 유효 범위

```
function test(name) {
   var output = 'Hello ' + name + '...!';
}

console.log(output)
```

- 함수 안의 지역 변수는 함수 외부에서 사용 불가능
- 지역 변수는 함수 실행 시 생성되고 종료 시 사라짐

함수 고급

❖ 클로저

- 클로저 특징 : 규칙 위반 가능

```
function test(name) {
   var output = 'Hello ' + name + '...!';

   return function() {
      console.log(output)
   }
}

test('Javascript')();
```

- 지역 변수를 남겨두는 현상
- test( ) 함수로 생성된 공간
- 리턴된 함수 자체
- 살아남은 지역 변수 output(반드시 리턴된 클로저 함수 사용)

❖ 클로저 정의

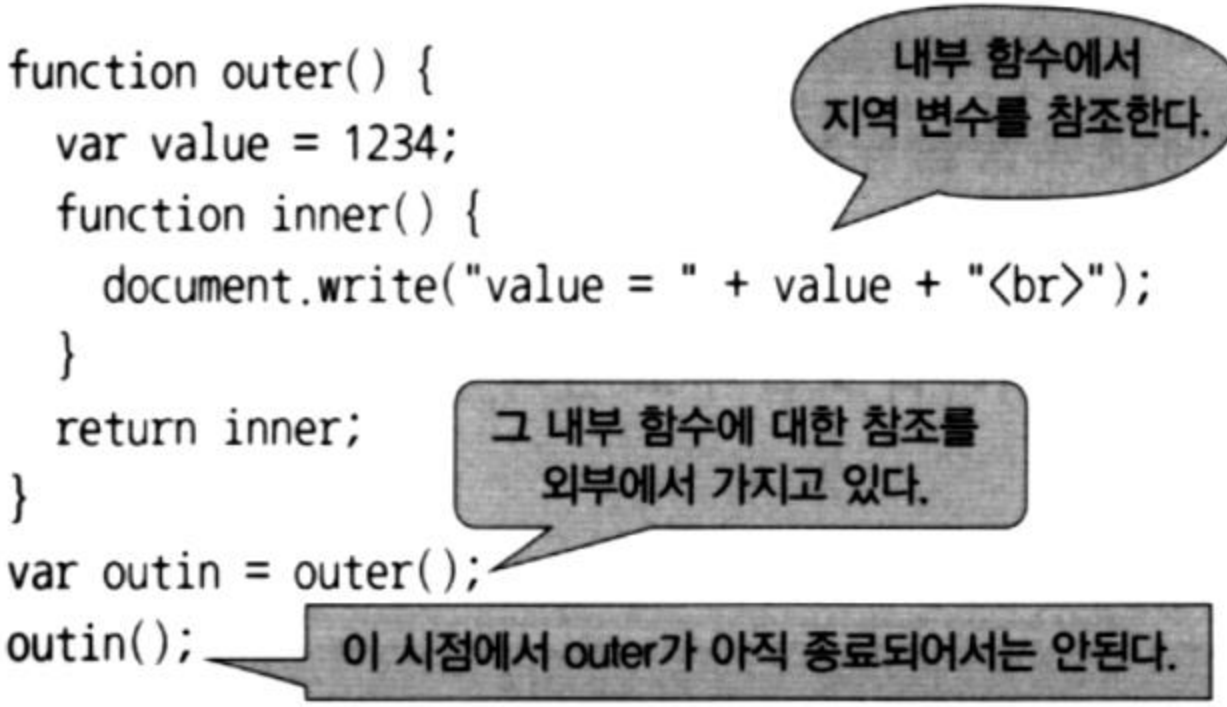
```
function test(name) {
   var output = 'Hello ' + name + '...!';

   return function() {
      console.log(output)
   }
}

var test_1 = test('Node');
var test_2 = test('Javascript');

test_1();
test_2();
```

❖ 클로저


function outer() {
var value = 1234;
function inner() {
document.write("value = " + value + "
");
}
return inner;
}
var outin = outer();
outin();
내부 함수에서
지역 변수를 참조한다.
그 내부 함수에 대한 참조를
외부에서 가지고 있다.
이 시점에서 outer가 아직 종료되어서는 안된다.

❖ 05_2_05_closure.html

```
<body>
   <script>
      function outer() {
         var value = 1234;
         function inner() {
            document.write("value = " + value + "<br>");
         }
         inner();
      }
      outer();
   </script>
</body>
```

❖ 05_2_05_closure2.html

```
<body>
   <script>
      function outer() {
         var value = 1234;
      }
      outer();
      document.write("value = " + value + "<br>");
   </script>
</body>
```

❖ 05_2_05_closure3.html

```
<body>
   <script>
      function outer() {
         var value = 1234;
         function inner() {
            document.write("value = " + value + "<br>");
         }
         return inner;
      }
      var outin = outer();
      outin();
   </script>
</body>
```

❖ 05_2_05_closure4.html

```
<body>
   <script>
      function outcount() {
         var count = 0;

         setInterval(function() {
            count++;
            document.write(count + "초 지났습니다." + "<br>");
         }, 1000);
      }
      outcount();
   </script>
</body>
```

```
function outer() {
    var value;
    이벤트 등록(function() {
        value 사용;
    });
}
```

함수 고급

❖ 동적 함수

```
var 변수 = new Function("인수1", "인수2", .... "본체");
```

❖ 05_2_06_dynamicfunc.html

```
<body>
   <script>
      var add = new Function("a", "b", "return a + b;");
      document.write("2 + 3 = " + add(2, 3) +"<br>");
   </script>
</body>
```

❖ 05_2_06_dynamicfunc2.html

```
<body>
   <script>
      var body;
      if (confirm("더할래, 곱할래") == true) {
         body = "return a + b;";
      } else {
         body = "return a * b;";
      }
      var add = new Function("a", "b", body);
      document.write("result = " + add(2, 3) +"<br>");
   </script>
</body>
```

함수 고급

❖ 재귀호출

- 자기 자신을 호출하는 함수

❖ callee

- arguments의 속성
  - 해당 함수를 호출한 함수에 대한 참조
  - 익명 함수의 재귀호출에 사용

❖ 05_2_07_recursive.html

```
<body>
   <script>
      function fact(n) {
         if (n == 1) {
            return 1;
         } else {
            return n * fact(n-1);
         }
      }
      document.write("5! = " + fact(5) + "<br>");
   </script>
</body>
```

❖ 05_2_08_callee.html

```
<body>
   <script>
      document.write("5! = " + function(n) {
         if (n == 1) {
            return 1;
         } else {
            return n * arguments.callee(n-1);
         }
      }(5) + "<br>");
   </script>
</body>
```

함수
- 내장 함수 -

내장 함수

❖ 타입 변환 함수

Number(value)
String(value)
Boolean(value)

parseInt(value, radix)
parseFloat(value)

```
Number("1234") : 1234
Number("12개") : NaN
parseInt("12개") : 12
Number("3.1415") : 3.1415
Number("3.14원주율") : NaN
parseFloat("3.14원주율") : 3.14
```

내장 함수

❖ 05_3_01_parseint.html

```
<body>
   <script>
      document.write('Number("1234") : ' + Number("1234") + "<br>");
      document.write('Number("12개") : ' + Number("12개") + "<br>");
      document.write('parseInt("12개") : ' + parseInt("12개") +
                  "<br>");
      document.write('Number("3.1415") : ' + Number("3.1415") +
                  "<br>");
      document.write('Number("3.14원주율") : ' + Number("3.14원주율")
                  + "<br>");
      document.write('parseFloat("3.14원주율") : ' +
                  parseFloat("3.14원주율") + "<br>");
   </script>
</body>
```

❖ 05_3_02_parseintradix.html

```
<body>
   <script>
      var hex = "0x1a"
      document.write("0x1a = " + parseInt(hex, 16) + "<br>");
      document.write("0x1a = " + parseInt(hex) + "<br>");
      document.write("0x1a = " + Number(hex) + "<br>");

      var decimal = "12"
      document.write("12(10) = " + parseInt(decimal, 10) + "<br>");
      document.write("12(16) = " + parseInt(decimal, 16) + "<br>");
   </script>
</body>
```

❖ 05_3_03_tostringradix.html

```
<body>
   <script>
      var hex = 0x1a;

      document.write("hex = " + hex + "<br>");
      document.write("hex = " + hex.toString(16) + "<br>");
      document.write("hex = " + hex.toString(2) + "<br>");
   </script>
</body>
```

내장 함수

❖ 값의 상태 점검

isFinite(value)

isNaN(value)

```
if (b == NaN) {

if (NaN == NaN) {
```

❖ 05_3_04_isfinite.html

```
<body>
   <script>
      var a = 2 / 0;
      if (isFinite(a) == false) {
         document.write("무한대값입니다." + "<br>");
      }

      var b = 0 / 0;
      if (isNaN(b)) {
         document.write("올바른 숫자가 아닙니다." + "<br>");
      }
   </script>
</body>
```

내장 함수

❖ 인코딩

- 인터넷 URL : 영문알파벳과 몇 개의 특수 기호만 가능
- utf-8 한글의 경우
  - 한글자 3자리 → URL 인코딩

```
query=%EC%86%8C%EB%85%80%EC%8B%9C%EB%8C%80
```

함수	설명
escape(string) unescape(string)	* @ - _ + . / 를 제외한 모든 특수 문자를 인코딩한다.
encodeURI(uri) decodeURI(uri)	. / ? : @ & = + $ # 을 제외한 모든 특수 문자를 인코딩한다.
encodeURIComponent(uri) decodeURIComponent(uri)	거의 대부분의 특수 문자를 인코딩한다. 알파벳과 숫자 정도만 원래대로 남는다. UTF-8로 인코딩된다.

내장 함수

❖ 05_3_05_encode.html

```
<body>
   <script>
      var s = "소/녀:시@대";
      document.write("원본 = " + s + "<br>");
      var e = encodeURIComponent(s);
      document.write("인코딩 = " + e + "<br>");
      var u = decodeURIComponent(e);
      document.write("디코딩 = " + u + "<br>");
   </script>
</body>
```

내장 함수

❖ 타이머 함수

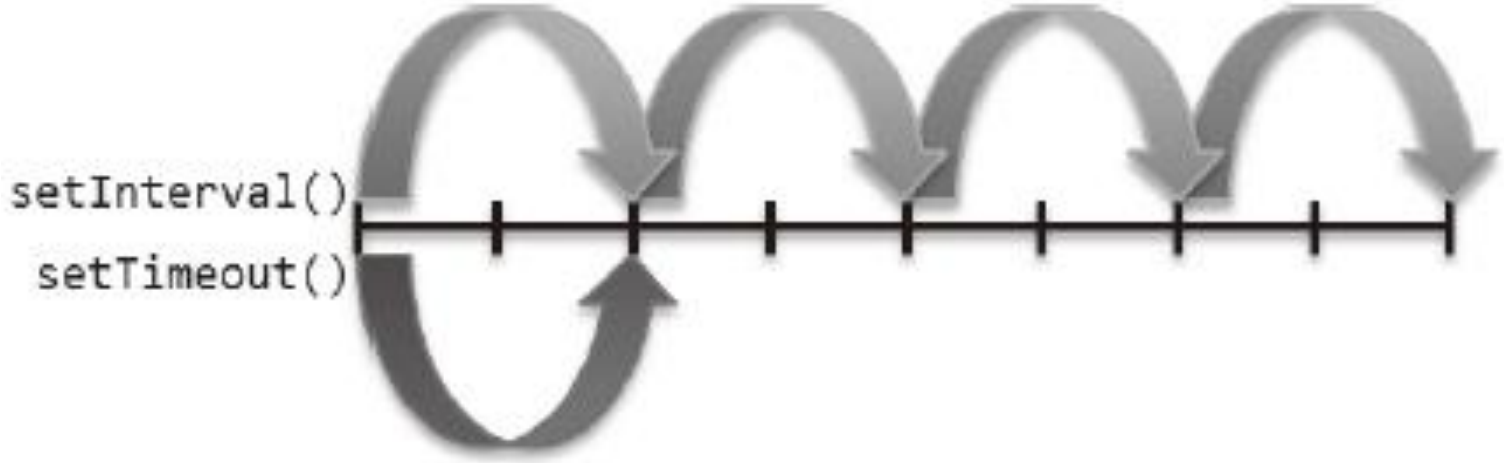
ㅇ 특정한 시간에 특정한 함수를 실행 가능하게 함

메서드 이름	설명
setTimeout(function, millisecond)	일정 시간 후 함수를 한 번 실행합니다.
setInterval(function, millisecond)	일정 시간마다 함수를 반복해서 실행합니다.
clearTimeout(id)	일정 시간 후 함수를 한 번 실행하는 것을 중지합니다.
clearInterval(id)	일정 시간마다 함수를 반복하는 것을 중단합니다.

내장 함수

❖ 타이머 함수

- setTimeout ( ) 메서드 : 특정한 시간 후에 함수를 한 번 실행
- setInterval ( ) 메서드 : 특정한 시간마다 함수를 실행



내장 함수

❖ 타이머 함수

- setTimeout ( ) 함수의 주의사항 : 특별히 없음
- setInterval ( ) 함수의 주의사항 : 지속적 자원의 소비
- 해결 방법 : 타이머를 멈춤
- clearTimeout ( )함수/clearInterval ( ) 함수를 사용

```
var intervalID = setInterval(function(){
   console.log(new Date());
}, 1000);

setTimeout(function(){
   clearInterval(intervalID);
}, 10000);
```

내장 함수

❖ 코드 실행 함수

- 자바스크립트는 문자열을 코드로 실행할 수 있는 특별한 함수를 제공
- `eval( )` 함수는 문자열을 자바스크립트 코드로 실행하는 함수

함수 이름	설명
eval(string)	string을 자바스크립트 코드로 실행합니다.

내장 함수

❖ 코드 실행 함수

- 코드 실행 함수 예

```
var willEval = '';
willEval += 'var num = 10;';
willEval += 'console.log(num);';

eval(willEval);
```