

프로젝트 기반 실무형 서비스

Chap02. Nodejs 기본문법

김진형

자료형 및 변수

자료형 및 변수

자료형

- 숫자 (number)
- 문자열 (string)
- 참/거짓 (boolean)
- undefined
- 심볼 (symbol)

자료형 및 변수

숫자, 문자, 참거짓

```
var intNumber = 10;  
var floatNumber = 20.3;  
var exponentialNumber1 = 10e6;  
var exponentialNumber2 = 10-e6;
```

```
var string1 = "문자열1";  
var string2 = '문자열2';  
var string3 = "문자'열'3";  
var string4 = '문자"열"';
```

```
var bool1 = true;  
var bool2 = false;
```

자료형 및 변수

typeof 연산자

```
typeof 100;           // number
typeof "Hello";       // string
typeof true;          // boolean
typeof undefined;     // undefined
typeof null;          // object
```

자료형 및 변수

undefined

- 자바스크립트에서 null 은 object 타입이며 아직 '값' 이 정해지지 않은 것을 의미한다.
- undefined는 null과 달리 '타입' 이 정해지지 않은 것을 의미한다.
- 자바스크립트에서 undefined는 초기화되지 않은 변수나 존재하지 않는 값에 접근할 때 반환된다.

```
var num;           // 초기화 하지 않았으므로 undefined
var str = null;    // object 타입의 null 값
typeof num2;       // 정의되지 않은 변수 undefined
```

자료형 및 변수

null과 undefined

- null과 undefined는 동등 연산자(==)와 일치 연산자(===)로 비교할 때 그 결과 값이 다르다.
- null과 undefined는 타입을 제외하면 같은 의미 이지만, 타입이 다르므로 일치하지 않는다.

```
null == undefined; // true  
null === undefined; // false
```

자료형 및 변수

형 변환

- 자바스크립트는 타입 검사가 매우 유연한 언어
- 타입이 정해져 있지 않으며, 같은 변수에 다른 타입의 값을 다시 대입할 수도 있다.

```
var num = 20;    // number 타입 20
num = "이십";    // string 타입 "이십"
var num;         // 재선언은 불가, 무시된다.
```


자료형 및 변수

묵시적 타입 변환(implicit)

- 특정 타입의 값을 기대하는 곳에 다른 타입이 오면, 자동으로 타입을 변환하여 사용한다.
- 아래 예제의 세번째 예제는, 문자열을 뺄셈연산을 하기위해 변환할 수 없으므로 NaN 을 반환
- NaN (Not a Number)의 축약형으로 정의되지 않은 값이나 표현할 수 없는 값이라는 의미

```
10 + "문자열"; // 10이 문자열로 변환되어 연결  
"3" * "5";    // 곱셈을 위해 둘다 숫자로 변환  
1 - "문자열"   // NaN
```

자료형 및 변수

명시적 타입 변환(explicit)

- Number()
- String()
- Boolean()
- Object()
- parseInt()
- parseFloat()

```
Number("10"); // 숫자 10
String(true); // 문자열 "true"
Boolean(0); // boolean false
Object(3); // new Number(3)와 동일한 객체
```

자료형 및 변수

변수

- 변수란 데이터를 저장할 수 있는 메모리 공간이며, 그 값이 변경될 수 있는 공간
- 자바스크립트는 var 키워드를 이용해 변수를 선언한다.

```
var month;           // month라는 이름의 변수 선언  
date = 25;           // date라는 이름의 변수를 묵시적 선언  
var year, week;      // 여러 변수를 한번에 선언
```

자료형 및 변수

변수의 타입과 초기값

- 변수는 타입이 정해져 있지 않으며, 같은 변수에 다른 타입을 다시 대입할 수도 있다.
- 한 변수에 다른 타입의 값을 여러번 대입할 수는 있지만, 한 번 선언된 변수를 재선언 할수는 없다.
- 변수는 영문자(대소문자), 숫자, 언더스코어(_) 또는 달러(\$)로만 구성된다.

```
var num = 10;           // 선언과 동시에 초기화
num = [10, 20, 30];     // 배열 대입
var num;               // 무시
```

연산자

연산자

연산자의 종류

- 산술 연산자
- 대입 연산자
- 증감 연산자
- 비교 연산자
- 논리 연산자
- 비트 연산자
- 기타 연산자

연산자

비교 연산자

비교 연산자	설명
==	왼쪽 피연산자와 오른쪽 피연산자의 값이 같으면 참을 반환함.
===	왼쪽 피연산자와 오른쪽 피연산자의 값이 같고, 같은 타입이면 참을 반환함.
!=	왼쪽 피연산자와 오른쪽 피연산자의 값이 같지 않으면 참을 반환함.
!==	왼쪽 피연산자와 오른쪽 피연산자의 값이 같지 않거나, 타입이 다르면 참을 반환함.

```
var x = 3, y = '3', z = 3;  
x == y // 타입을 맞추면 같으므로 true  
x === y // 타입이 다르므로 false  
x === z // 값과 타입이 모두 같으므로 true
```

제어문

제어문

제어문의 종류

- 조건문
- 반복문

배열

배열

배열의 기초

// 배열 리터럴을 이용하는 방법

```
var arrLit = [1, true, "JavaScript"];
```

// Array 객체의 생성자를 이용하는 방법

```
var arrObj = Array(1, true, "JavaScript");
```

// new 연산자를 이용한 Array 객체 생성 방법

```
var arrNewObj = new Array(1, true, "JavaScript");
```

배열

배열의 요소 추가

```
var arr = [1, true, "Java"];
```

```
// push() 메소드를 이용하는 방법  
arr.push("Script");
```

```
// length 프로퍼티를 이용하는 방법  
arr[arr.length] = 100;
```

```
// 특정 인덱스를 지정하여 추가하는 방법  
arr[10] = "자바스크립트";
```

배열

문자열을 배열처럼 접근하기

- 인터넷 익스플로러 7 이하 버전에서는 동작 안함

```
var str = "안녕하세요!";  
str.charAt(2);           // 하  
str[2];                   // 하
```

함수

함수

기본 정의

- 함수는 function 키워드로 시작되며, 다음과 같은 구성요소를 갖는다.
 - 함수의 이름
 - 괄호 안에 쉼표(,)로 구분되는 매개변수(parameter)
 - 중괄호({})로 둘러싸인 자바스크립트 실행문

// addNum라는 이름의 함수를 정의함.

```
function addNum(x, y) {  
    document.write(x + y);  
}
```

// addNum() 함수에 인수로 2와 3을 전달하여 호출함.

```
addNum(2, 3);
```

함수

반환 (return) 문

```
function multiNum(x, y) {  
    return x * y;  
}  
var num = multiNum(3, 4);
```


함수

값으로서의 함수

- 자바스크립트에서 함수는 문법적 구문일뿐만 아니라 값(value) 이기도 한다.
- 함수가 변수에 대입될 수도 있으며, 다른 함수의 인수로 전달될 수도 있다.

// 제공의 값을 구하는 함수 sqr를 정의함.

```
function sqr(x) {  
    return x * x;  
}
```

```
var sqrNum = sqr;  
document.write(sqr(4) + "<br>");  
document.write(sqrNum(4));
```

함수

매개변수 (parameter)

- 함수를 정의할 때는 매개변수의 타입을 따로 명시하지 않는다.
- 함수를 호출할 때에도 인수(argument)로 전달된 값에 대해 어떠한 타입 검사도 하지않는다.
- 함수의 정의보다 적은 수의 인자가 전달되더라도, 다른 언어와 달리 오류를 발생시키지 않는다.
 - 이 경우 자동으로 undefined 값을 설정한다.

```
// x, y, z라는 3개의 매개변수를 가지는 함수 addNum()을 정의함.  
function addNum(x, y, z) {  
    return x + y + z;  
}  
// 인수로 1, 2, 3을 전달하여 함수를 호출함. -> 6  
addNum(1, 2, 3);  
// 인수로 1, 2을 전달하여 함수를 호출함. -> NaN  
addNum(1, 2);  
// 인수로 1을 전달하여 함수를 호출함. -> NaN  
addNum(1);  
// 인수로 아무것도 전달하지 않고 함수를 호출함. -> NaN  
addNum();
```

함수

디폴트 매개변수

- 디폴트 매개변수란 함수를 호출할 때 명시된 인수를 전달하지 않았을 경우에 사용하게 될 기본값을 의미한다.

```
function mul(a, b) {  
    // 인수가 한 개만 전달되었을 때 나머지 매개변수의 값을  
    // undefined 값이 아닌 1로 설정함.  
    b = (typeof b !== 'undefined') ? b : 1;  
    return a * b;  
}  
mul(3, 4); // 12  
mul(3);    // 3
```

```
function mul(a, b = 1) {  
    return a * b;  
}
```

함수

익명 함수

- 익명함수는 이름이 없는 함수를 의미
- 간단하게 변수에 저장해서 사용

```
var mySquare = function(x) {  
    return x * x ;  
}
```

```
console.log("mySquare : " + mySquare(10));  
console.log("mySquare : " + typeof mySquare);
```

ES6

ES6

ECMA Script 6

- 자바스크립트를 표준화 하기위해 만들어졌음
- 기존 자바스크립트의 단점들을 보완

자료형 및 변수

숫자, 문자, 참거짓

```
var intNumber = 10;  
var floatNumber = 20.3;  
var exponentialNumber1 = 10e6;  
var exponentialNumber2 = 10-e6;
```

```
var string1 = "문자열1";  
var string2 = '문자열2';  
var string3 = "문자'열'3";  
var string4 = '문자"열"';  
var string5 = `문"자'열'`;
```

```
var bool1 = true;  
var bool2 = false;
```

자료형 및 변수

표현식 삽입법 (Expression interpolation)

```
var string1 = "문자열1";  
var string2 = '문자열2'  
var string3 = "문자'열'3";  
var string4 = '문자"열"';  
var string5 = ` "문"자'열' `;  
var string6 = `이것은 ${string1} 입니다.  
그리고 개행까지 할수있죠`;
```


자료형 및 변수

변수 선언

- var
 - 전통적인 javascript의 변수 선언
 - 유연한 처리로 대부분 에러없이 처리
- let
 - var의 단점을 보완
 - 명확한 에러
- const
 - immutable 변수

자료형 및 변수

var, let

```
var name = 'bathingape'  
console.log(name) // bathingape
```

```
var name = 'javascript'  
console.log(name) // javascript
```

```
let name = 'bathingape'  
console.log(name) // bathingape
```

```
// Uncaught SyntaxError: Identifier  
// 'name' has already been declared  
let name = 'javascript'  
console.log(name)
```

자료형 및 변수

let, const

```
let name = 'bathingape'  
console.log(name) // bathingape
```

```
name = 'react'  
console.log(name) //react
```

```
const name = 'bathingape'  
console.log(name) // bathingape
```

```
// Uncaught SyntaxError: Identifier  
// 'name' has already been declared  
const name = 'javascript'  
console.log(name)
```

```
//Uncaught TypeError: Assignment  
// to constant variable.  
name = 'react'  
console.log(name)
```

함수

익명 함수

- 익명함수는 이름이 없는 함수를 의미
- 간단하게 변수에 저장해서 사용

```
var mySquare = function(x) {  
    return x * x ;  
}
```

```
console.log("mySquare : " + mySquare(10));  
console.log("mySquare : " + typeof mySquare);
```

함수

화살표 표기법

- ES6 문법이며 함수를 간결하게 표현

```
var mySquare = function(x) {  
    return x * x ;  
}
```

```
var mySquare = (x) => {  
    return x * x ;  
}
```

```
var mySquare = (x) => x * x ;
```

객체(object)

객체 (object)

자바스크립트 객체

- 자바스크립트의 기본 데이터 타입은 객체
- 숫자, 문자열, boolean, undefined 타입을 제외한 모든것이 객체
- 숫자, 문자열, boolean과 같은 원시 타입은 값이 정해진 객체로 취급

```
var person = {  
  name: "홍길동",  
  birthday: "030219",  
  pId: "1234567",  
  fullId: function() {  
    return this.birthday + this.pId;  
  }  
};  
person.name // 홍길동  
person["name"] // 홍길동
```

객체 (object)

객체의 생성

- 리터럴 표기를 이용한 객체의 생성
- 생성자를 이용한 객체의 생성
- Object.create() 메소드를 이용한 객체의 생성

```
var 객체이름 = {  
    프로퍼티1이름 : 프로퍼티1의값,  
    프로퍼티2이름 : 프로퍼티2의값,  
    ...  
};
```

```
// new 연산자를 사용하여 Date 타입의 객체를 생성함.  
var day = new Date();
```


객체 (object)

객체의 프로토타입

- C++이나 Java같은 클래스 기반의 객체 지향 언어와는 달리 자바스크립트는 프로토타입 기반의 객체지향 언어이다.
- 프로토타입 기반이기에 상속의 개념이 클래스 기반의 객체 지향 언어와는 약간 다르다.
- 자바스크립트에서는 현재 존재하고 있는 객체를 프로토타입으로 사용하여, 해당 객체를 복제하여 재사용하는 것을 상속이라고 한다.
- 자바스크립트의 모든 객체는 프로토타입(prototype)이라는 객체를 가지고 있다.

객체 (object)

프로토타입 체인

```
// 이 객체의 프로토타입은 Object.prototype이다.  
var obj = new Object();
```

```
// 이 객체의 프로토타입은 Array.prototype이다.  
var arr = new Array();
```

```
// 이 객체의 프로토타입은 Date.prototype이다.  
var date = new Date();
```

```
// 이 객체는 Date.prototype 뿐만 아니라  
// Object.prototype도 프로토타입으로 가진다.  
var date = new Date();
```

객체 (object)

프로토타입 생성

- 프로토타입을 생성하는 가장 기본적인 방법은 객체 생성자 함수를 작성하는 것 이다.
- 생성자 함수를 작성하고 new 연산자를 사용해 객체를 생성하면, 같은 프로토타입을 가지는 객체들을 생성할 수 있다.

```
// 개에 관한 생성자 함수를 작성함.  
function Dog(color, name, age) {  
    this.color = color;  
    this.name = name;  
    this.age = age;  
}  
// 이 객체는 Dog라는 프로토타입을 가짐.  
var myDog = new Dog("흰색", "마루", 1);
```

객체 (object)

프로퍼티 및 메소드 추가

- 이미 생성된 객체에 새로운 프로퍼티나 메소드를 추가하는 방법은 다음과 같다.

```
// 개에 관한 생성자 함수를 작성함.  
function Dog(color, name, age) {  
    this.color = color;  
    this.name = name;  
    this.age = age;  
}  
// 이 객체는 Dog라는 프로토타입을 가짐.  
var myDog = new Dog("흰색", "마루", 1);  
  
// 품종에 관한 프로퍼티를 추가함.  
myDog.family = "시베리안 허스키";  
// 털색을 포함한 품종을 반환해 주는 메소드를 추가함.  
myDog.breed = function() {  
    return this.color + " " + this.family;  
}
```

객체 (object)

프로토타입에 프로퍼티 및 메소드 추가

- 프로토타입에 새로운 프로퍼티나 메소드를 추가하는 것은 생성자 함수에 직접 추가해야함

```
function Dog(color, name, age) {  
  this.color = color;  
  this.name = name;  
  this.age = age;  
  // 프로토타입에 프로퍼티를 추가할 때에는 기본값을 가지게 할 수 있음.  
  this.family = "시베리안 허스키";  
  this.breed = function() {  
    return this.color + " " + this.family;  
  };  
}
```

객체 (object)

prototype 프로퍼티

- prototype 프로퍼티를 이용하면 현재 존재하고 있는 프로토타입에 새로운 프로퍼티나 메소드를 손쉽게 추가 가능함

```
function Dog(color, name, age) {  
    this.color = color;  
    this.name = name;  
    this.age = age;  
}  
// 현재 존재하고 있는 Dog 프로토타입에 family 프로퍼티를 추가함.  
Dog.prototype.family = "시베리안 허스키";  
// 현재 존재하고 있는 Dog 프로토타입에 breed 메소드를 추가함.  
Dog.prototype.breed = function() {  
    return this.color + " " + this.family;  
};
```

객체 (object)

this 키워드

- 자바스크립트에서 this 키워드는 해당 키워드가 사용된 자바스크립트 코드 영역을 포함하고 있는 객체를 가리킨다.
- 예를들어 메소드 내부에서 사용된 this 키워드는 해당 메소드를 포함하고 있는 객체를 가리킨다.
- 객체 내부에서 사용된 this 키워드는 객체 그 자신을 가리킨다.
- 이러한 this는 변수가 아닌 키워드이므로 사용자가 임의로 가리키는 값을 바꿀 수 없다.

정규식 의 개념

정의

- 문자열에서 특정한 문자 조합을 찾기 위한 패턴.
- 구성 : /검색패턴/플래그

```
// 정규 표현식 리터럴을 이용한 생성
var regStr = /a+bc/;
// RegExp 객체를 이용한 생성
var regObj = new RegExp("a+bc");

// /a+bc/
regStr;
// /a+bc/
regObj;
```


정규식 의 개념

단순 패턴 검색

```
var targetStr = `간장 공장 공장장은 강 공장장이고, 된장  
공장 공장장은 장 공장장이다.`;  
var strReg1 = /공장/;  
var strReg2 = /장공/;  
  
targetStr.search(strReg1); // 3  
targetStr.search(strReg2); // -1
```

정규식 의 개념

플래그

플래그(flag)	설명
i	검색 패턴을 비교할 때 대소문자를 구분하지 않도록 설정함.
g	검색 패턴을 비교할 때 일치하는 모든 부분을 선택하도록 설정함.
m	검색 패턴을 비교할 때 여러 줄의 입력 문자열을 그 상태 그대로 여러 줄로 비교하도록 설정함.
y	대상 문자열의 현재 위치부터 비교를 시작하도록 설정함.

```
var targetStr = "bcabcAB";
```

```
// 검색 패턴 비교 시 기본 설정으로 대소문자를 구분함.
```

```
var strReg = /AB/;
```

```
// new RegExp("AB", "i")와 동일함.
```

```
var strUsingFlag = /AB/i;
```

```
targetStr.search(strReg); // 5
```

```
// 2 -> 대소문자를 구분하지 않고 검색함.
```

```
targetStr.search(strUsingFlag);
```

정규식의 응용

특수문자

특수 문자	설명
\	역슬래시() 다음에 일반 문자가 나오면 이스케이프 문자로 해석하고, 특수 문자가 나오면 일반 문자로 해석함.
\d	숫자를 검색함. /[0-9]/와 같음.
\D	숫자가 아닌 문자를 검색함. /[^\d]/와 같음
\w	언더스코어()를 포함한 영문자 및 숫자를 검색함. /[A-Za-z0-9_]/와 같음.
\W	언더스코어(), 영문자, 숫자가 아닌 문자를 검색함. /[^\w]/와 같음.
\s	띄어쓰기, 탭, 줄 바꿈 문자 등의 공백 문자를 검색함.
\S	띄어쓰기, 탭, 줄 바꿈 문자 등의 공백 문자가 아닌 문자를 검색함.
\b	단어의 맨 앞이나 맨 뒤가 패턴과 일치하는지를 검색함.
\xhh	16진수 hh에 해당하는 유니코드 문자를 검색함.
\uhhhh	16진수 hhhh에 해당하는 유니코드 문자를 검색함.

정규식의 응용

특수문자

```
var targetStr = "ab1bc2cd3de";
```

```
// 2 -> 0부터 9까지의 숫자를 검색함.
```

```
var reg1 = /\d/;
```

```
// 8 -> 3부터 9까지의 숫자를 검색함.
```

```
var reg2 = /[3-9]/;
```

```
var targetStr = "abc 123";
```

```
// 공백 문자를 사이에 두는 언더스코어(_)를
```

```
// 포함한 영문자 및 숫자로 이루어진 문자열을 검색함.
```

```
var reg = /\w\s\w/; // c 1
```

정규식의 응용

특수문자

```
var targetStr1 = "abc123abc";    // 7
var targetStr2 = "abc 123 abc";  // 1
var targetStr3 = "abc@123!abc";  // 1

// 단어의 맨 앞이나 맨 뒤에 부분 문자열 "bc"가 존재하는지를 검색함.
var reg = /bc\b/;
```

정규식의 응용

양화사 (quantifier)

괄호	설명
n*	바로 앞의 문자가 0번 이상 나타나는 경우를 검색함. {0, }와 같음.
n+	바로 앞의 문자가 1번 이상 나타나는 경우를 검색함. {1, }과 같음.
n?	바로 앞의 문자가 0번 또는 1번만 나타나는 경우를 검색함. {0,1}과 같음.

```
var targetStr = "Hello World!";
```

```
// 문자 'l' 다음에 문자 'o'가 0번 이상 나타나는 경우를 검색함.
```

```
var zeroReg = /lo*/;
```

```
// 문자 'l' 다음에 문자 'o'가 1번 이상 나타나는 경우를 검색함.
```

```
var oneReg = /lo+/;
```

```
// 문자 'l' 다음에 문자 'o'가 0 또는 1번만 나타나는 경우를 검색함.
```

```
var zeroOneReg = /lo?/;
```

```
targetStr.search(zeroReg);    // 2
```

```
targetStr.search(oneReg);     // 3
```

```
targetStr.search(zeroOneReg); // 2
```

정규식의 응용

양화사 (quantifier)

괄호	설명
n*	바로 앞의 문자가 0번 이상 나타나는 경우를 검색함. {0, }와 같음.
n+	바로 앞의 문자가 1번 이상 나타나는 경우를 검색함. {1, }과 같음.
n?	바로 앞의 문자가 0번 또는 1번만 나타나는 경우를 검색함. {0,1}과 같음.

```
var targetStr = "Hello World!";
```

```
// 문자 'l' 다음에 문자 'o'가 0번 이상 나타나는 경우를 검색함.
```

```
var zeroReg = /lo*/;
```

```
// 문자 'l' 다음에 문자 'o'가 1번 이상 나타나는 경우를 검색함.
```

```
var oneReg = /lo+/;
```

```
// 문자 'l' 다음에 문자 'o'가 0 또는 1번만 나타나는 경우를 검색함.
```

```
var zeroOneReg = /lo?/;
```

```
targetStr.search(zeroReg);    // 2
```

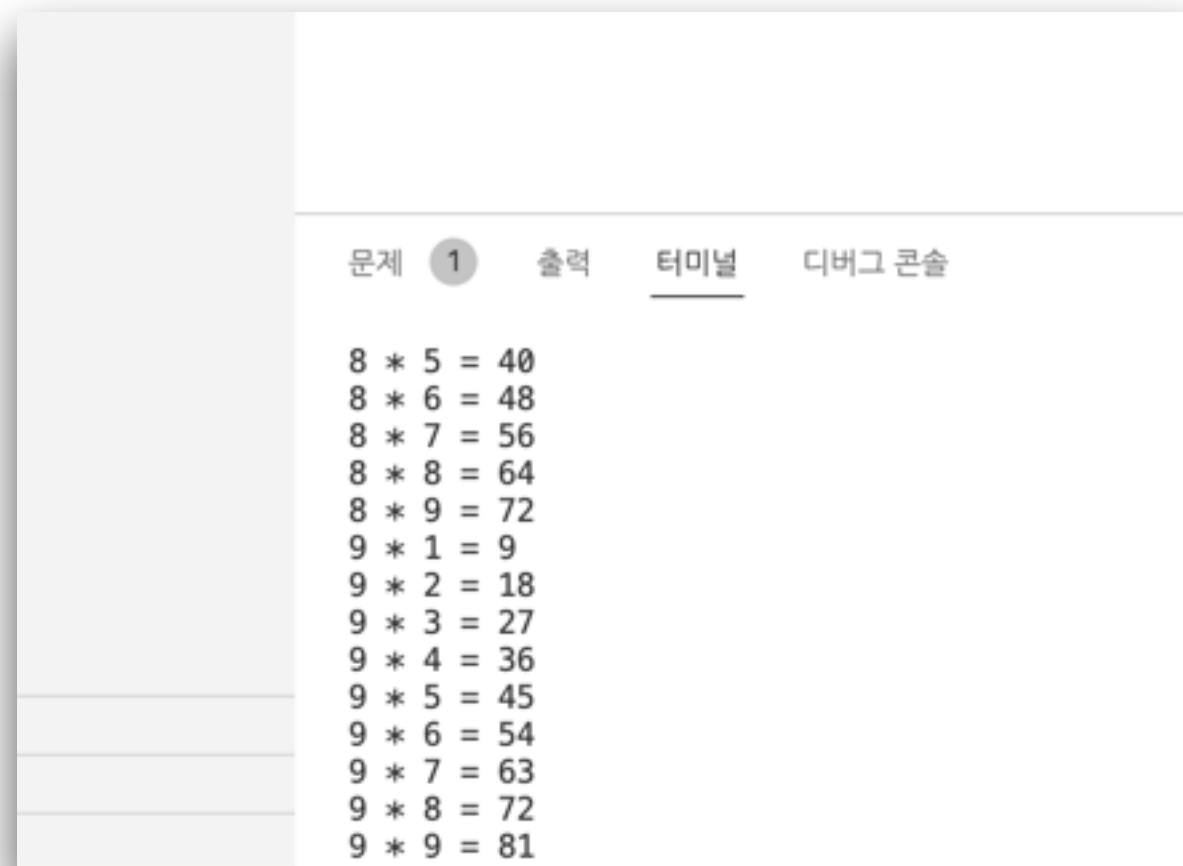
```
targetStr.search(oneReg);    // 3
```

```
targetStr.search(zeroOneReg); // 2
```

예제

구구단 출력

1단부터 9단까지 출력될수 있도록



The screenshot shows a web application interface with a sidebar on the left and a main content area on the right. The sidebar has a light gray background and contains a vertical list of items. The main content area has a white background and a header with four tabs: '문제 1', '출력', '터미널', and '디버그 콘솔'. The '출력' tab is selected, and the content area displays a list of multiplication problems and their results, starting from 8 * 5 = 40 and ending with 9 * 9 = 81.

문제	1	출력	터미널	디버그 콘솔
		8 * 5 = 40		
		8 * 6 = 48		
		8 * 7 = 56		
		8 * 8 = 64		
		8 * 9 = 72		
		9 * 1 = 9		
		9 * 2 = 18		
		9 * 3 = 27		
		9 * 4 = 36		
		9 * 5 = 45		
		9 * 6 = 54		
		9 * 7 = 63		
		9 * 8 = 72		
		9 * 9 = 81		

가위바위보 게임

무작위 난수를 받아와 0 ~ 2 사이의 정수를 넣을수 있도록

- 가위 0, 바위 1, 보 2 라고 가정
- 컴퓨터와 사용자의 값을 무작위로 생성하여 가위바위보를 하는 게임

```
var com, user;
```

```
com = // 0 ~ 2 사이의 난수 생성  
user = // 0 ~ 2 사이의 난수 생성
```

```
console.log("컴퓨터의 입력 : " + com);  
console.log("사용자의 입력 : " + user);
```

가위바위보 게임

중복된 0 ~ 2 정수를 받아오는 코드를 하나의 함수로

```
var com, user;  
  
com = Math.floor(Math.random() * 3);  
user = Math.floor(Math.random() * 3);  
  
console.log("컴퓨터의 입력 : " + com);  
console.log("사용자의 입력 : " + user);
```

가위바위보 게임

승패의 결과를 알려주는 **winnerIs** 함수 구현

```
var com, user;

com = rsp();
user = rsp();

console.log("컴퓨터의 입력 : " + com);
console.log("사용자의 입력 : " + user);

winnerIs(com, user);

/**
 * 가위 0, 바위 1, 보 2 의 값을 무작위로 return
 */
function rsp() {
    return Math.floor(Math.random() * 3);
}
```

```
var com, user;
```

```
com = rsp();  
user = rsp();
```

```
console.log("컴퓨터의 입력 : " + com);  
console.log("사용자의 입력 : " + user);
```

```
winnerIs(com, user);
```

```
/**  
 * 가위 0, 바위 1, 보 2 의 값을 무작위로 return  
 */  
function rsp() {  
    return Math.floor(Math.random() * 3);  
}
```

```
/**  
 * 컴퓨터와 사용자 값을 비교하여 승패를 출력  
 * @param {number} com 컴퓨터 입력 수  
 * @param {number} user 사용자 입력 수  
 */  
function winnerIs(com, user) {  
    if(com == user) {  
        console.log("비겼습니다.");  
    } else if(/* 사용자가 이기는 조건은? */) {  
        console.log("사용자 승");  
    } else {  
        console.log("사용자 패");  
    }  
}
```

```
(com == 0 && user == 1) || (com == 2 && user == 2) ||
(com == 2 && user == 0)
```

대괄호 [] 안의 숫자는 (com - user)의 결과값

사용자 \ 컴퓨터	가위(0)	바위(1)	보(2)
가위(0)	비김 [0]	컴퓨터 승 [1]	사용자 승 [2]
바위(1)	사용자 승 [-1]	비김 [0]	컴퓨터 승 [1]
보(2)	컴퓨터 승 [-2]	사용자 승 [-1]	비김 [0]

```
(com - user) == -1 || (com - user) == 2
```

```
((com - user + 1) % 3) == 0
```

```
!(com == user + 1) % 3
```

검증 함수

주어진 메일 형식이 우리학교 메일인지 검증

```
let mail1 = "abc@tukorea.ac.kr";  
let mail2 = "hello@korea.com";  
let mail3 = "hi@hello.net";
```

```
validation(mail1); // true  
validation(mail2); // false  
validation(mail3); // false
```

검증 함수

주어진 메일 형식이 우리학교 메일인지 검증

```
let mail1 = "abc@tukorea.ac.kr";
let mail2 = "hello world";
let mail3 = "hi@hello";

console.log(validation(mail1)); // true
console.log(validation(mail2)); // false
console.log(validation(mail3)); // false

function validation(mail) {
  const regex = new RegExp(' [a-z0-9]+@tukorea.ac.kr ');
  return regex.test(mail)
}
```