String 객체(1)

string 변수 VS String 객체

- 기본 타입과 참조 타입(객체)
- string 기본 타입의 변수는 마침표(.) 연산자에 의해 암시적으로 객체로 변환
- 따라서 기본 타입의 변수에서 String객체의 메서드를 사용할 수 있음

```
> var a1 = "hello";
< undefined
> var a2 = new String("hello");
< undefined
> console.dir(a1)
  ▼hello i
< undefined
> console.dir(a2)
  ▼String i
     0: "h"
     1: "e"
     2: "l"
     3: "l"
     4: "o"
     length: 5
   ▶  proto  : String
     [[PrimitiveValue]]: "hello"
< undefined
> var a3 = a1.toUpperCase()
< undefined
>
```

String 객체(2)

String 객체 메서드

[**표 05-01** : String 객체의 HTML 관련 메서드]

big( )	글자를 한 단계 더 크게	메서드 이름과 동일한 시작 태그, 마감 태그로 문자열을 둘러싼 값을 리턴함. ex) a변수값이 hello인 경우 a.big( ) → 〈big〉hello〈/big〉
small( )	글자를 한 단계 더 작게	
blink( )	깜빡이도록	
bold( )	더 굵게	
fixed( )	타자기체로	
italics( )	이탤릭체로	
strike( )	strike 처리	
sub( )	아래 첨자	
sup( )	위 첨자	
fontsize(size)	〈font size="[크기]"〉hello〈/font〉 형태의 문자열 리턴	
fontcolor(color)	〈font color="[색상]"〉hello〈/font〉 형태의 문자열 리턴	
anchor(anchorname)	〈a name="[앵커명]"〉hello〈/a〉 형태의 문자열 리턴	
link(url)	〈a href="[URL]"〉hello〈/a〉 형태의 문자열 리턴	

[**표 05-02** : String 객체의 주요 메서드]

charAt(index)	index 위치의 문자열을 리턴함.
indexOf(value, startIndex)	startIndex 위치로부터 value 문자열이 발견된 위치를 리턴한다. value 문자열을 찾지 못한 경우는 -1을 리턴한다. startIndex를 생략하면 처음부터 찾는다.
lastIndexOf(value, startIndex)	indexOf와 유사하지만 뒤쪽부터 검색한다.
concat(v1, v2, ...)	문자열들을 이어 붙여 하나의 문자열로 리턴한다.
toLowerCase( )	소문자로 변환한 값을 리턴한다.
toUpperCase( )	대문자로 변환한 값을 리턴한다.
replace(searchString,newValue)	문자열을 찾아 치환한다. 검색 문자열은 정규식으로 표현한다.
substring(start, end)	start에서 end 사이의 문자열을 추출하여 리턴한다. end를 생략할 경우 start 이후의 모든 문자열을 추출한다.
substr(start,length)	start에서 length 길이만큼의 문자열을 추출하여 리턴한다. length를 생략할 경우 start 이후의 모든 문자열을 추출한다.
slice(start, end)	start에서 end까지 문자열을 추출한다. substring과의 차이점이 있다면 음수를 사용하여 마지막부터의 위치를 지정할 수 있다.
split(separator, limit)	separator 문자열로 분리해 배열로 리턴한다. separator가 여러 개인 경우는 정규식으로 사용할 수 있다.
trim( )	앞 뒤의 공백을 제거한다.
match(regexp)	정규식에 해당하는 문자열을 추출하여 배열로 리턴한다.

String 객체(3)

간단한 예제

[예제 05-01 : String 객체의 메서드]

```
01: var a = "간장 공장 공장장은 강 공장장이고, 된장 공장 공장장은 공 공장장이다.";
02: var b = "hello";
03: var c = "world";
04:
05: console.log("charAt(3) : " + a.charAt(3));
06: console.log("indexOf('공장') : " + a.indexOf("공장"));
07: console.log("b.concat(' ' + c) : " + b.concat(" " + c));
08: console.log("b.toUpperCase() : " + b.toUpperCase());
09: console.log("a.replace(/\s/g, '_') : " + a.replace(/\s/g, "_"));
10: console.log("a.substring(3, 10) : [" + a.substring(3, 10) + "]");
11: console.log("a.slice(1, -3) : " + a.slice(4, -6));
12:
13: var arr1 = a.split(" ");
14: console.log("a.split(' ') : 다음 값을 확인합니다.");
15: console.log(arr1);
```

```
charAt(3) : 공                                                    05-01.html:11
indexOf('공장') : 3                                               05-01.html:12
b.concat(' ' + c) : hello world                                   05-01.html:13
b.toUpperCase() : HELLO                                           05-01.html:14
a.replace(/s/g, '_') : 간장_공장_공장장은_강_                      05-01.html:15
공장장이고,_된장_공장_공장장은_공_공장장이다.
a.substring(3, 10) : [공장 공장장은]                              05-01.html:16
a.slice(1, -3) : 장 공장장은 강 공장장이고, 된                     05-01.html:17
장 공장 공장장은 공
a.split(' ') : 다음 값을 확인합니다.                              05-01.html:20
▼Array[10] i                                                      05-01.html:21
   0: "간장"
   1: "공장"
   2: "공장장은"
   3: "강"
   4: "공장장이고,"
   5: "된장"
   6: "공장"
   7: "공장장은"
   8: "공"
   9: "공장장이다."
   length: 10
  ▶ __proto__ : Array[0]
```

Number 객체

[**표 05-03** : Number 객체의 함수]

Number.isFinite( )	값이 유한수인지를 true/false로 리턴한다.
Number.isInteger( )	값이 정수인지를 true/false로 리턴한다.
Number.isNaN( )	값이 NaN(숫자 아님:Not a Number)인지를 true/false로 리턴한다.
Number.parseInt( )	다른 형식의 값을 정수로 변환한다.
Number.parseFloat( )	다른 형식의 값을 실수로 변환한다.

[**표 05-04** : Number 객체의 상수]

Number.MAX_VALUE	number 타입의 최댓값
Number.MIN_VALUE	number 타입의 최솟값
Number.NaN	NaN 숫자가 아님(Not a Number)
Number.POSITIVE_INFINITY	양의 무한대 값
Number.NEGATIVE_INFINITY	음의 무한대 값

[**표 05-05** : Number 객체의 메서드]

toExponential( )	number 값을 지수표기법으로 표현한 문자열을 리턴한다.
toFixed( )	고정 소수점으로 나타낸 문자열을 리턴한다. 실제 값의 소수점 자리수보다 작은 크기의 고정 크기를 지정하면 값이 반올림된다.

Date 객체(1)

시간을 다루는 객체

Date 생성자 함수

[**표 05-06** : Date 생성자 함수의 사용법]

new Date( )	현재 시각으로 객체 생성
new Date(milliseconds)	GMT 1970.1.1 기준으로 흐른 밀리초값을 이용해 객체 생성
new Date(dateString)	날짜 형식의 문자열을 이용해 객체 생성
new Date(year, month, day)	연, 월, 일 값을 이용해 객체 생성
new Date(year, month, day, hours, minutes, seconds, milliseconds)	연, 월, 일, 시, 분, 초, 밀리초 값을 숫자로 입력하여 객체 생성. 뒤쪽부터 값을 생략할 수 있다.

[**표 05-07** : Date 객체의 메서드]

getFullYear( ), setFullYear( )	연도를 반환, 설정한다.
getMonth( ), setMonth( )	월을 반환, 설정한다(0~11).
getDate( ), setDate( )	일자를 반환, 설정한다(1~31).
getDay( )	요일을 반환한다(0~6). 일요일:0, 토요일:6
getHours( ), setHours( )	시간을 반환, 설정한다(0~23).
getMinutes( ), setMinutes( )	분을 반환, 설정한다(0~59).
getSeconds( ), setSeconds( )	초를 반환, 설정한다(0~59).
getMilliseconds( ), setMilliseconds( )	밀리초를 반환, 설정한다(0~999).
parse( )	다른 형식의 값을 1970.1.1 이후 흐른 밀리초 단위로 변환한다.
toDateString( )	날짜 형식의 문자열을 리턴한다(ex: "Fri Jan 15 2016").
toTimeString( )	시각 형식의 문자열을 리턴한다(ex: "16:36:38 GMT+0900 (대한민국 표준시)").
toUTCString( )	UTC 형식의 문자열을 리턴한다(ex: "Fri, 15 Jan 2016 07:36:38 GMT").

[예제 05-02 : Date 객체]

```
//2050년 크리스마스의 요일은?
var a = new Date(2050,11,25, 10,30);
console.log("요일(0~6) : " + a.getDay());
console.log("날짜 정보 : " + a.toString());
```

```
요일(0~6) : 0
날짜 정보 : Sun Dec 25 2050 10:30:00 GMT+0900 (대한민국 표준시)
```

Date 객체(2)

Date prototype에 메서드 추가

[예제 05-03 : Date 객체에 메서드 추가]

```
01: //--mode : 변환 형식 지정. "date", "time", "datetime"
02: //--isMilli : 밀리초까지 보여줄지 여부를 지정
03: Date.prototype.toKrString = function(mode, isMilli) {
04:     var strout, year, month, day, hour, minute, second, milli;
05:
06:     if (typeof(mode) == "undefined" ||
07:         mode != "date" && mode != "time" && mode != "datetime") {
08:         mode = "date";      //date를 기본값으로...
09:     }
10:     strout = "";
11:     year = this.getFullYear();
12:     month = this.getMonth()+1;
13:     if (month < 10) { month = "0" + month; }
14:     day = this.getDate();
15:     if (day < 10) { day = "0" + day; }
16:     hour = this.getHours();
17:     if (hour < 10) { hour = "0" + hour; }
18:     minute = this.getMinutes();
19:     if (minute < 10) { minute = "0" + minute; }
20:     second = this.getSeconds();
21:     if (second < 10) { second = "0" + second; }
22:     milli = this.getMilliseconds();
23:     if (milli < 10) {
24:         milli = "00" + milli;
25:     } else if(milli < 100) {
26:         milli = "0" + milli;
27:     }
28:
29:     if (mode == "date" || mode == "datetime") {
30:         strout += year + "-" + month + "-" + day;
31:     }
32:     if (mode == "datetime") { strout += " "; }
33:     if (mode == "time" || mode == "datetime") {
34:         strout += hour + ":" + minute + ":" + second;
35:         if (isMilli) {
36:             strout += " " + milli;
37:         }
38:     }
39:     return strout;
40: }
41:
42: var a = new Date(2050,11,25, 10,30,11, 2);
43: console.log("a.toKrString() : " + a.toKrString());
44: console.log("a.toKrString('date') : " + a.toKrString("date"));
45: console.log("a.toKrString('datetime') : " + a.toKrString("datetime"));
46: console.log("a.toKrString('datetime', true) : " + a.toKrString("datetime", true));
47: console.log("a.toKrString('time') : " + a.toKrString("time"));
48: console.log("a.toKrString('time', true) : " + a.toKrString("time", true));
```

Math 객체

[**표 05-08** : Math 객체의 정적 속성]

LOG2E	2를 밑으로 하는 자연 로그(약 1.4427)
LOG10E	10을 밑으로 하는 자연 로그(약 0.43429)
PI	원주율 PI(약 3.141593)
SQRT2	2의 제곱근(약 1.4142)

[**표 05-09** : Math 객체의 정적 메서드]

abs(a)	절대값을 리턴함 ex) Math.abs(−2) → 2
sin(a)	사인값을 리턴함
cos(a)	코사인값을 리턴함
tan(a)	탄젠트값을 리턴함
ceil(a)	소숫점 이하 올림 처리한 값을 리턴함
floor(a)	소숫점 이하 내림 처리한 값을 리턴함
exp(a)	거듭 제곱으로 반올림된 지수값를 리턴함
log(a)	E를 밑으로 하는 로그값을 리턴함
max(a,b,c,...)	인자값 중 최대값을 리턴함
min(a,b,c,...)	인자값 중 최소값을 리턴함
pow(a,b)	a의 b제곱값을 리턴함
random()	0~1사이의 난수값을 리턴함
round(a)	반올림값을 리턴함
sqrt(a)	a의 제곱근값을 리턴함

Array 객체(1)

Array 객체의 메서드

- 배열 객체의 메서드는 중요함
- 서버와 AJAX 통신하여 받아오는 데이터 형식이 JSON
- 여러건의 데이터인 경우 배열 형태의 JSON으로 받아옴.
- 받아온 데이터를 이용해 필터링, 정렬, 추출, 가공 하는 경우가 많아지고 있음
- 조금 복잡하지만 알아두면 유용하다.

[표 05-10 : Array 객체의 메서드]

concat(arr1, arr2)	두 개의 배열을 이어 붙여서 새로운 배열을 리턴함.
join(separator)	배열의 요소값을 separator로 이어 붙여 하나의 문자열을 리턴함.
pop( )	배열의 마지막 요소값을 제거하고 제거된 요소값을 리턴함.
push( )	배열의 마지막에 새로운 값을 추가함.
reverse( )	배열의 순서를 역순으로 바꿈.
shift( )	배열의 첫 번째 요소값을 제거하고 제거된 요소값을 리턴함.
slice(start, end)	start index에서 end index 이전까지의 배열값을 새로운 배열을 만들어 리턴함.
splice(index, deleteCount, insertItem1, insertItem2, ...)	배열에 값을 삽입, 삭제, 치환할 수 있는 메서드.
sort(fn)	배열값을 기준으로 정렬함. 함수를 인자로 전달할 경우, 함수가 리턴하는 값을 기준으로 정렬함.
indexOf(val)	특정 값을 가진 요소의 인덱스 번호를 리턴함.
filter(fn)	필터 조건을 만족하는 값만을 가진 요소로 필터링함. 인자로 함수 또는 값이 전달됨.
forEach(fn)	요소값을 이용해 루프를 돌면서 전달한 함수를 호출함.
map(fn)	요소값을 이용해 루프를 돌면서 전달한 함수를 호출함. forEach()와의 차이점은 함수에서 값을 리턴할 수 있음.
reduce(callbackfn, initValue)	콜백 함수를 호출하여 집계된 결과를 리턴함.

Array 객체(2)

Splice() 메서드

- 삽입, 삭제, 치환 등 모든 기능을 제공하는 다목적 메서드

splice(index, deleteCount, insItem1, insItem2, ...)

[예제 05-04 : splice() 메서드]

```
01: //사용법 : splice(index, deleteCount, insItem1, insItem2, ...)
02: var arr = [10,20,30,40,50];
03: //---삽입
04: arr.splice(1,0,"hello");    //2번째부터 삭제는 하지 않고 "hello" 추가
05: //---현재 상태 : [10, "hello", 20, 30, 40, 50]
06: arr.splice(3, 0, "world 1","world 2")       //여러개 요소를 삽입
07: //---현재 상태 : [10, "hello", 20, "world 1", "world 2", 30, 40, 50]
08: arr.splice(5, 2)    //5번 인덱스에서 2개 삭제
09: //---현재 상태 : [10, "hello", 20, "world 1", "world 2", 50]
10: arr.splice(2, 1, "jquery!!");   //2번 인덱스 위치에서 1개 요소 삭제하고 "jquery!!" 추가
11: //---현재 상태 : [10, "hello", "jquery!!", "world 1", "world 2", 50]
12: console.log(arr);
```

Array 객체(3)

- sort() 메서드
  - 정렬 기능 제공
  - 함수의 리턴값을 정렬기준으로 사용할 수 있음

```
05: var data2 = [ 70,20,40,50,60,100 ];
06: data2.sort(function(a,b) {
07:     return b-a;
08: });
```

```
24: data3.scores.sort(function (a, b) {
25:   if (a.score > b.score) {
26:         return 1;
27:     } else if (a.score < b.score) {
28:         return -1;
29:     } else {
30:         return 0;
31:     }
32: });
```

Array 객체(4)

(이어서)

- 직접 만든 sort 메서드 형태

```
01: Array.prototype.sort2 = function(fn) {
02:     if (typeof(fn) == "function" ) {
03:         for (var i=0; i < this.length-1; i++) {
04:             for (var j=i+1; j < this.length; j++) {
05:                 var result = fn(this[i], this[j]);
06:                 if (result > 0) {
07:                     var temp = this[i];
08:                     this[i] = this[j];
09:                     this[j] = temp;
10:                 }
11:             }
12:         }
13:     } else {
14:         //요소의 값만으로 정렬하도록 함.
15:     }
16: }
```

Array 객체(5)

filter() 메서드

- 전달한 함수의 조건을 만족하는 경우의 요소만 모아서 새로운 배열 리턴

arr.filter(fn [, thisArgs]);

fn : 조건 기준을 실행할 함수이다.

thisArgs : fn 함수 내에서 this로 사용할 값이다. optional한 값이다. 즉 필수 파라미터가 아니므로반드시 전달해야 하는 것은 아니다.

fn

function(item, index, arr) { }

item : 배열의 요소값

index : 배열 요소에 대한 인덱스 번호

```
23: var result2 = data2.scores.filter(function(val, index) {
24:     if (val.score >= 80) {
25:         return true;
26:     }
27: });
```

```
34: var result3 = data2.scores.filter(function(val, index) {
35:     if (val.score >= this.min && val.score <= this.max ) {
36:         return true;
37:     }
38: }, condition);
```

Array 객체(6)

forEach() 메서드

- for, while문과 같은 반복문

forEach(fn [,thisArgs])

fn : function(item, index) { }

item : 각 요소의 값

index : 각 요소의 인덱스 번호

thisArgs : fn 함수 내에서 사용할 this 객체를 전달할 수 있음. optional 함. 즉 필수 파라미터가 아니므로 반드시 전달해야 하는 것은 아니다.

```
15: data.scores.forEach(function(item, index) {
16:     console.log((index+1) + " : " + item.name + ", " + item.score);
17: });
```

```
for (var i=0; i < data.scores.length; i++) {
   var item = data.scores[i];
   console.log((i+1) + " : " + item.name + ", " + item.score);
}
```

Array 객체(7)

map 메서드

- forEach와 유사하지만 fn 함수 내에서 값을 리턴할 수 있으며 이 값으로 새로운 배열을 만들어 리턴한다.

map(fn [,thisArgs])

fn : function(item, index) { }

item : 각 요소의 값

index : 각 요소의 인덱스 번호

thisArgs : fn 함수 내에서 사용할 this 객체를 전달할 수 있음. optional 함. 즉 필수 파라미터가 아니므로 반드시 전달해야 하는 것은 아니다.

리턴값 : fn이 리턴한 값으로 만든 배열을 리턴함.

```
15: var obj = { sum : 0, count : 0 };
16: var arr = data.scores.map(function(item, index) {
17:     var score = item.correct / item.total * 100;
18:     console.log(item.name + " : " + score);
19:     this.sum += score;
20:     this.count++;
21:     return { "name" : item.name, "score" : score };
22: }, obj);
```

Array 객체(8)

reduce() 메서드

- 배열 요소마다 함수를 호출해 누적된 값을 구할 수 있음

reduce(fn [, initialValue])

fn : function(previousValue, currentValue, currentIndex) { }

previousValue : fn 함수에 대한 이전 호출로부터 리턴받은 값

currentValue : 현재 배열의 요소값

currentIndex : 현재 배열의 요소에 대한 인덱스 번호

initialValue : 누적을 시작하는 초기값. optional함. 즉 필수 파라미터가 아니므로 반드시 전달해야 하는 것은 아니다.

initialValue가 지정되었을 경우와 아닌 경우에 대해 첫 번째 fn 함수 호출 처리가 달라진다.

- **지정** previousValue는 initialValue가 되며, currentValue는 배열의 첫 번째 요소값이 됨.
- **미지정** previousValue는 배열의 첫 번째 요소값이 되며, currentValue는 배열의 두 번째 요소값이 됨.

```
16: var initialValue = "";
17: var result = data.scores.reduce(function(preVal, curVal, curIdx) {
18:     if (curVal.score >= 80) {
19:         return  preVal + curVal.name + " ";
20:     } else {
21:         return preVal;
22:     }
23: }, initialValue);
```

Array 객체(9)

concat() 메서드

- 2개 이상의 배열을 이어 붙이는 기능

[예제 05-11]

```
var arr1 = [1,2,3];
var arr2 = [4,5,6];
var arr3 = [7,8,9];

var result = arr1.concat(arr2, arr3);
console.log(result); // 결과 : [1,2,3,4,5,6,7,8,9]
```

정규식 객체(1)

정규식이란?

- 특정한 패턴을 가진 문자열 집합을 표현하는 형식 언어
- 특정 패턴의 문자열을 찾아내거나 치환하는 작업을 간단하게 처리할 수 있음.

정규식 객체 만들기

- **정규식 리터럴 이용** var 변수명 = /패턴/플래그;
- **RegExp 생성자 함수 이용** var 변수명 = new RegExp("패턴", "플래그");

정규식 객체(2)

문자 표현 패턴

[**표 05-11** : 문자 표현 패턴]

패턴	설명
/abc/	'abc' 문자열이 연속된 문자열 패턴을 의미한다.
/[abc]/	a,b,c 중에서 임의의 한 문자를 나타내는 패턴이다.
/[^abc]/	a,b,c가 아닌 임의의 한 문자를 나타내는 패턴이다. [ ] 안에서 사용되는 ^기호는 not을 의미한다.
/./	/./ 마침표(.) 기호는 줄바꿈 문자가 아닌 임의의 한 문자를 나타내는 패턴이다.
₩w	₩w(역슬래시 소문자w)는 알파뉴머릭(Alphanumeric)* 문자 중 임의의 한 문자를 나타내는 패턴이다.
₩W	₩W는 대문자인 경우에 not Alphanumeric의 의미이다. not ₩w이다. 알파뉴머릭이 아닌 문자열 중 임의의 한 문자를 나타낸다.
₩d	디지트(digit) 문자열 중 임의의 한 문자를 나타낸다.
₩D	디지트(digit)가 아닌 문자열 중 임의의 한 문자를 나타낸다.

정규식 객체(3)

수량/반복 표현 패턴

[표 05-12 : 수량/반복 표현 패턴]

패턴	설명
{n,m}	n번 이상 m번 이하로 반복되는 패턴을 의미한다.
{n,}	n번 이상 반복되는 패턴을 의미한다.
{n}	n번 반복되는 패턴을 의미한다.
?	0번 또는 한 번 나타내는 패턴을 의미한다. {0,1}과 같은 의미다.
+	한 번 이상 반복되는 패턴을 의미한다. {1,}와 같은 의미다.
*	0번 또는 여러 번 반복되는 패턴을 의미한다. {0,}와 같은 의미다.

기타 패턴

[표 05-13 : 기타 패턴]

패턴	설명
^	지정된 문자열로 시작되는 패턴을 의미한다. 예) /^a/
$	지정된 문자열로 끝나는 패턴을 의미한다. 예) /a$/
|	OR의 기능이다. /ab|cd|ef/는 ab 또는 cd 또는 ef 문자열을 포함하는 패턴을 의미한다.

정규식 객체(4)

플래그

[표 05-14 : 플래그]

플래그	설명
g	글로벌(Global) 옵션이다. 이 플래그가 부여되어 있다면, 패턴을 찾는 작업을 전체 문자열에 걸쳐 여러 번 반복한다.
i	문자열 패턴을 찾을 때 대소문자를 구분하지 않고 패턴을 찾는 작업을 수행한다.
m	여러 줄의 문자열에서 패턴을 찾는 작업을 가능하게 한다.

정규식 사용 예

- 우리나라 휴대폰 전화 번호 패턴

  /01[016789]-₩d{3,4}-₩d{4}/

- 알파벳2글자-숫자5개로 이루어진 상품 코드(ex PI-12462)

  /[A-Za-z]{2}-₩d{5}/

- A로 시작하고 임의의 문자가 여러 번 위치한 뒤 마지막에 C로 끝나는 문자열 패턴

  /^A.+C$/

- URL 주소 형식 패턴

  /(http(s)?:₩/₩/)?₩w+(₩.₩w+)+/

- 한글 패턴

  /[ㄱ-ㅎㅏ-ㅑㅣ가-힣]/

정규식 객체(5)

자바스크립트에서 정규식 사용

[예제 05-12 : 패턴 매칭]

```
01: var pat = /[ㄱ-ㅎㅏ-ㅑ가-힣]/;
02: var str1 = "한글과 English 비교";
03: var str2 = "Hello World!!";
04: console.log(pat.test(str1));   //결과 : true
05: console.log(pat.test(str2));//결과 : false
```

[예제 05-13 : 문자열 추출]

```
01: var pat1 = /01[016789]-\d{3,4}-\d{4}/;
02: var pat2 = /01[016789]-\d{3,4}-\d{4}/g;
03: var str = "홍길동님 번호는 010-2224-3331이고, 이몽룡님 번호는 016-316-9765이다.";
04:
05: var list1 = str.match(pat1);
06: var list2 = str.match(pat2);
07: console.log(list1);
08: console.log(list2);
```

[예제 05-14 : 문자열 치환]

```
01: //한글 또는 공백이 아닌 문자열 패턴
02: var pat = /[^ㄱ-ㅎㅏ-ㅑ가-힣\s]/g;
03: var str = "'한글도' 있고, English도 있고, 日本語도 있다.";
04: var str2 = str.replace(pat, "_");
05: console.log(str2);
```