HTML5プロフェッショナル認定試験 レベル2ポイント解説無料セミナー

株式会社クリーク・アンド・リバー社 認定講師

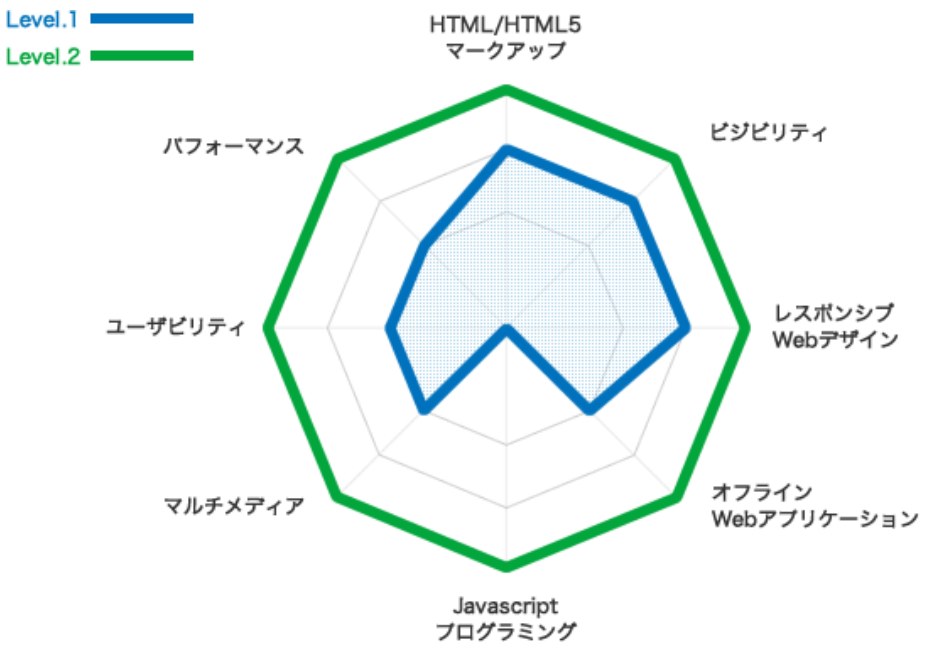
高井 歩

本日の内容

- 試験概要
- 出題範囲
- 学習方法
- JavaScript対策
  - JavaScript基本編
    - おやくそく
    - 変数の基本
    - 関数の基本
  - JavaScript応用編
    - 変数のスコープ
    - オブジェクト
    - クロージャ

試験概要

試験概要



HTML5プロフェッショナル認定試験
レベル２

所要時間：90分（アンケート等の時間を含む）
試験問題数：40～45問
受験料：\15,000 （税別）
認定条件：HTML5 レベル2試験に合格し、
かつ有意なHTML5レベル1認定を保有していること。
認定の有意性の期限：5年間

試験概要

認定証

認定カード

認定証・認定カードは、認定されてから2週間程度で
ご登録されたご住所にお届けしています。

認定者ロゴ（名刺用）

認定者ロゴは、認定後すぐに名刺等でご利用していただけます。

出題範囲

出題範囲

	重要度	知識問題	コードリーティング問題	記述問題
JavaScript				
JavaScript文法	10	○	○	○
WebブラウザにおけるJavaScript API				
イベント	8	○	○	○
ドキュメントオブジェクト／DOM	6	○	○	
ウィンドウオブジェクト	8	○	○	
Selectors API	4	○	○	
テスト・デバッグ	2	○		

出題範囲

	重要度	知識問題	コードリーティング問題	記述問題
グラフィックス				
Canvas(2D)	6	○	○	
SVG	1	○		
マルチメディア				
video要素,audio要素	2	○	○	
オフラインアプリケーションAPI				
アプリケーションキャッシュの制御	2	○	○	
Session History and Navigation				
History　API	3	○	○	

出題範囲

	重要度	知識問題	コードリーティング問題	記述問題
表示制御				
Page Visibility	2	○		
Timing control for script-based animations	2	○	○	
ストレージ				
Web Storage	4	○	○	
Indexed Database API	2	○	○	
File API	2	○	○	

出題範囲

	重要度	知識問題	コードリーティング問題	記述問題
通信				
WebSocket	2	○		
XMLHttpRequest	4	○	○	
Geolocation API				
Geolocation APIの基本と位置情報の取得	2	○		
Web Workers				
並列処理の記述	4	○	○	
パフォーマンス				
Navigation Timing	4	○		
High Resolution Time	1	○		

学習方法

学習方法 (個人の感想です)

- 試験範囲をリストアップ
  - テキストエディタやアウトラインエディタで1行1用語
    - 重複や似た項目が分散しているのを整理
- 自分の知識、経験を棚卸し
  - 全ての用語について、簡単な解説が出来るか
- あやふやな点については書籍、Webで調査
  - エディタの項目に追記

サンプル

- JavaScript,Canvas,SVG,Multimedia APIなどについてはWeb検索などで一通りサンプルを作成してみる。
- JavaScriptは基本的な文法を理解しているだけでは足りない
  - デベロッパーツール等で動作を解析してみる
  - JavaScriptの仕組みについての参考書を確認しておく
- 重要度が低い項目(マイナーな機能)については、ブラウザ毎の対応状況も違うため、サンプルよりも用語と解説重視?

用語

- JavaScriptはプログラム言語としては、少ない仕組みでいろいろな応用が出来るように作られています。
- そのため、実体としては同じ仕組みに対して、用途や使用されるタイミングなどによって説明のために別の名称が付けられていることがあります。
- 混乱しがちですが、仕組みと用途をきちんと整理して覚えるようにしてください。

参考書籍

JavaScript 第6版

JavaScript本格入門　～モダンスタイルによる基礎からAjax・ｊQueryまで

フロントエンドエンジニア養成読本 [HTML、CSS、JavaScriptの基本から現場で役立つ技術まで満載!] (Software Design plus)

独習JavaScript 第2版

開眼! JavaScript ―言語仕様から学ぶJavaScriptの本質

付録:JavaScriptデバッガ

参考サイト

http://news.mynavi.jp/column/wc/011/

Chrome
http://www.buildinsider.net/web/chromedevtools/01#page-5

Firefox
https://developer.mozilla.org/ja/docs/Tools/Debugger

IE
http://d.hatena.ne.jp/replication/20130311/1363012914

Safari(英語)
https://developer.apple.com/library/safari/documentation/AppleApplications/Conceptual/Safari_Developer_Guide/Debugger/Debugger.html#//apple_ref/doc/uid/TP40007874-CH5-SW1

JavaScript基本編

おやくそく

scriptタグ

```
<script type="text/javascript">
</script>
```

```
<script ></script>
```

```
<script src="外部ファイル">
</script>
```

HTML5では、JavaScriptを記述する場合、type="text/javascript"を省略できます。

~~<script src="外部ファイル" />~~

終了タグの省略はできません

document.write

Webブラウザ上のJavaScriptでは、

document.write(‘出力したい文字列’);

と記述すると、documnt.writeを含む<script>タグの直後に’出力したい文字列’が書き出されます。

書き出された内容はHTMLの一部として処理されます。

※文字列以外の値は、自動的に文字列に変換されます。

変数の基本

変数の基本

```
<script type="text/javascript">
var a = 10;
var b = 20;
var c = 'JavaScript';
document.write(a+'<br />');
document.write(b+'<br />');
document.write(c+'<br />');
</script>
```

変数名	値

変数の基本

```
<script type="text/javascript">
var a = 10;
var b = 20;
var c = 'JavaScript';
document.write(a+'<br />');
document.write(b+'<br />');
document.write(c+'<br />');
</script>
```

変数名	値
a	*10*

変数の基本

```
<script type="text/javascript">
var a = 10;
var b = 20;
var c = 'JavaScript';
document.write(a+'<br />');
document.write(b+'<br />');
document.write(c+'<br />');
</script>
```

変数名	値
a	*10*
b	*20*

変数の基本

```
<script type="text/javascript">
var a = 10;
var b = 20;
var c = 'JavaScript';
document.write(a+'<br />');
document.write(b+'<br />');
document.write(c+'<br />');
</script>
```

変数名	値
a	*10*
b	*20*
c	*‘JavaScript’*

変数の基本

```
<script type="text/javascript">
var a = 10;
var b = 20;
var c = 'JavaScript';
document.write(a+'<br />');
document.write(b+'<br />');
document.write(c+'<br />');
</script>
```

変数名	値
a	*10*
b	*20*
c	*‘JavaScript’*

変数の基本

```
<script type="text/javascript">
var a = 10;
var b = 20;
var c = 'JavaScript';
document.write(a+'<br />');
document.write(b+'<br />');
document.write(c+'<br />');
</script>
```

変数名	値
a	*10*
b	*20*
c	*‘JavaScript’*

変数の基本

```
<script type="text/javascript">
var a = 10;
var b = 20;
var c = 'JavaScript';
document.write(a+'<br />');
document.write(b+'<br />');
document.write(c+'<br />');
</script>
```

変数名	値
a	*10*
b	*20*
c	*‘JavaScript’*

変数の上書き

```
<script type="text/javascript">
var a = 10;
var b = 20;
b = 'JavaScript';
document.write(a+'<br />');
document.write(b+'<br />');
</script>
```

変数名	値

変数の上書き

```
<script type="text/javascript">
var a = 10;
var b = 20;
b = 'JavaScript';
document.write(a+'<br />');
document.write(b+'<br />');
</script>
```

変数名	値
a	*10*

変数の上書き

```
<script type="text/javascript">
var a = 10;
var b = 20;
b = 'JavaScript';
document.write(a+'<br />');
document.write(b+'<br />');
</script>
```

変数名	値
a	*10*
b	*20*

変数の上書き

```
<script type="text/javascript">
var a = 10;
var b = 20;
b = 'JavaScript';
document.write(a+'<br />');
document.write(b+'<br />');
</script>
```

変数名	値
a	*10*
b	*‘JavaScript’*

undefined

```
<script type="text/javascript">
var a;
document.write(a);
</script>
```

変数名	値
a	*undefined*

関数の基本

関数の基本

```
<script type="text/javascript">
function add (a,b) {
  var c = a+b;
  return c;
}
var d = add(3,4);
document.write(d+'<br />');
document.write(add(10,20));
</script>
```

関数の基本

```
<script type="text/javascript">
function add (a,b) {
  var c = a+b;
  return c;
}
var d = add(3,4);
document.write(d+'<br>');
document.write(add(10,20));
</script>
```

関数の定義

関数の
呼び出し

関数の定義

```
function add (a,b)
{
  var c = a+b;
  return c;
}
```

add	関数名
(a,b)	引数リスト
return c;	戻り値

関数の呼び出し

関数の呼び出しは、関数の戻り値で置き替えられます

```
var d = add(3,4);
```

```
var d = 7;
```

```
document.write(add(10,20));
```

```
document.write(30);
```

※関数にreturn文が無い場合、戻り値はundefinedになります

問題

```
<script type="text/javascript">
function add (a,b) {
  var c = a+b;
  return c;
}
var d = add(add(1,2),3);
document.write(d);
</script>
```

このスクリプトを実行した結果、表示されるのは?

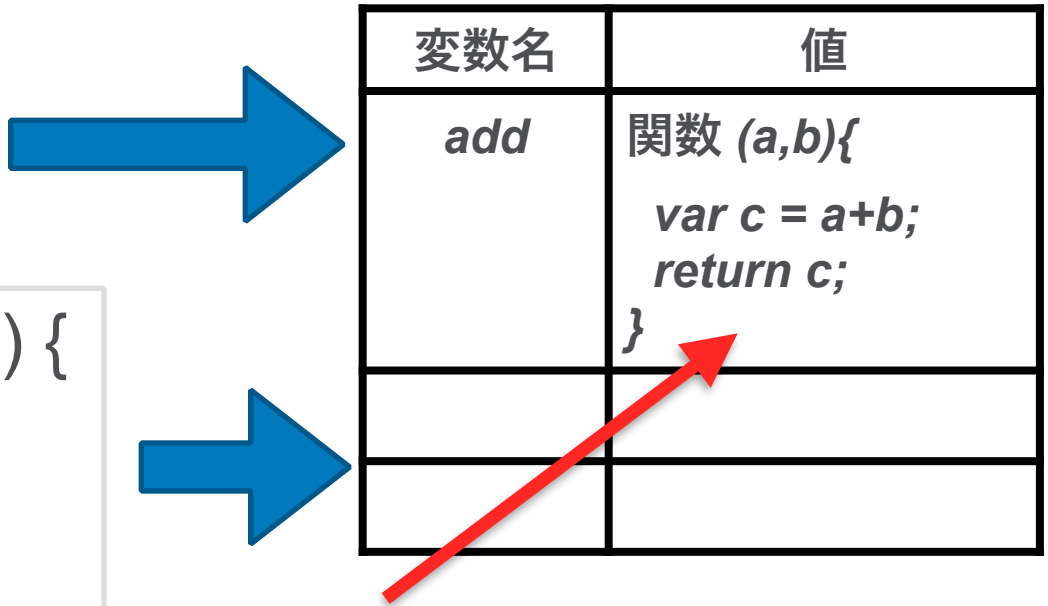
A. 1
B. 2
C. 3
D. 6

無名関数

```
function add (a,b) {
  var c = a+b;
  return c;
}
```

```
var add = function(a,b) {
  var c = a+b;
  return c;
}
```

結果は同じ



無名関数

- JavaScriptでは、関数も値の一種。
  - 文字列や数値と同じように、変数に代入できます。
- 関数そのものには名前は付いていません。
  - 変数に代入≒関数名を付ける
- 変数に代入されていない、名前の無い関数のことを、無名関数または匿名関数と呼びます。
- 代入した変数名と()を使って関数を呼び出せます。

無名関数

```
var add = function(a,b) {
  var c = a+b;
  return c;
}
var a = add;
document.write(a(5,6));
```

関数の代入された変数から、
他の変数へ代入しても、
同じように関数を呼び出せます。

無名関数

```
var d = (function (a,b) {
  var c = a+b;
  return c;
})(5,6);
document.write(d);
```

無名関数は、変数に代入しなくても、
(無名関数の定義)(引数)という形で直接実行できます。
関数の名前を変数表に登録せずに関数を実行できるので、JavaScriptプログラムでは良く使用されます。

JavaScript応用編

変数のスコープ

変数のスコープ

```
<script type="text/javascript">
function add (a,b) {
  var c = a+b;
  return c;
}
var d = add(3,4);
document.write(d);
document.write(add(10,20));
</script>
```

変数名	値

変数のスコープ

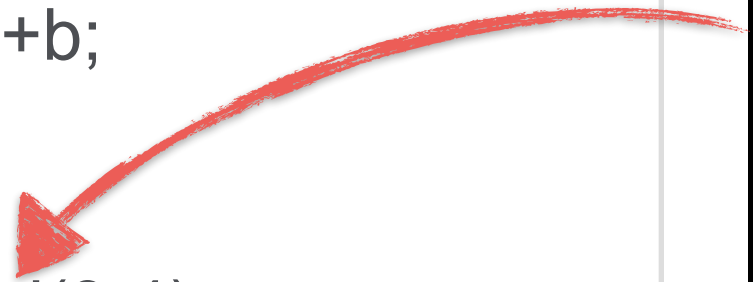
```
<script type="text/javascript">
function add (a,b) {
  var c = a+b;
  return c;
}
var d = add(3,4);
document.write(d);
document.write(add(10,20));
</script>
```

変数名	値

add	*関数 (a,b){
	var c = a+b;
	return c;
	}*

変数のスコープ

```
<script type="text/javascript">
function add (a,b) {
  var c = a+b;
  return c;
}
var d = add(3,4);
document.write(d);
document.write(add(10,20));
</script>
```



変数名	値

add	*関数 (a,b){
	var c = a+b;
	return c;
	}*

変数のスコープ

```
<script type="text/javascript">
function add (a,b) {
  var c = a+b;
  return c;
}
var d = add(3,4);
document.write(d);
document.write(add(10,20));
</script>
```

関数が呼び出されると、その関数用の新しい表が用意されます

変数名	値
a	*3*
b	*4*

変数のスコープ

```
<script type="text/javascript">
function add (a,b) {
  var c = a+b;
  return c;
}
var d = add(3,4);
document.write(d);
document.write(add(10,20));
</script>
```

変数名	値
a	*3*
b	*4*
c	*7*

変数のスコープ

```
<script type="text/javascript">
function add (a,b) {
  var c = a+b;
  return c;
}
var d = add(3,4);
document.write(d);
document.write(add(10,20));
</script>
```

変数名	値
a	*3*
b	*4*
c	*7*

変数のスコープ

```
<script type="text/javascript">
function add (a,b) {
  var c = a+b;
  return c;
}
var d = add(3,4);
document.write(d);
document.write(add(10,20));
</script>
```

7

関数の処理が終わると、関数用の表は破棄されます。

変数名	値
add	*関数 (a,b){ var c = a+b; return c; }*

変数名	値
a	3
b	4
c	7

変数のスコープ

```
<script type="text/javascript">
function add (a,b) {
  var c = a+b;
  return c;
}
var d = add(3,4);
document.write(d);
document.write(add(10,20));
</script>
```

関数が呼び出されると、その関数用の新しい表が用意されます

変数名	値
a	*10*
b	*20*

変数のスコープ

```
<script type="text/javascript">
function add (a,b) {
  var c = a+b;
  return c;
}
var d = add(3,4);
document.write(d);
document.write(add(10,20));
</script>
```

変数名	値
a	*10*
b	*20*
c	*30*

変数のスコープ

```
<script type="text/javascript">
function add (a,b) {
  var c = a+b;
  return c;
}
var d = add(3,4);
document.write(d);
document.write(add(10,20));
</script>
```

変数名	値
a	*10*
b	*20*
c	*30*

変数のスコープ

```
<script type="text/javascript">
function add (a,b) {
  var c = a+b;
  return c;
}
var d = add(3,4);
document.write(d);
document.write(add(10,20));
</script>
```

30

関数の処理が終わると、関数用の表は破棄されます。

変数名	値
add	*関数 (a,b){ var c = a+b; return c; }*

変数名	値
a	10
b	20
c	30

変数のスコープ

ブラウザで確認

変数のスコープ

```
<script type="text/javascript">
var b = 10;
function add_b (a) {
  var c = a+b;
  return c;
}
document.write(add_b(5));
</script>
```

変数名	値

変数のスコープ

```
<script type="text/javascript">
var b = 10;
function add_b (a) {
  var c = a+b;
  return c;
}
document.write(add_b(5));
</script>
```

変数名	値
b	*10*

変数のスコープ

```
<script type="text/javascript">
var b = 10;
function add_b (a) {
  var c = a+b;
  return c;
}
document.write(add_b(5));
</script>
```

変数名	値
b	*10*
add_b	*関数 (a){* *var c = a+b;* *return c;* *}*

変数のスコープ

```
<script type="text/javascript">
var b = 10;
function add_b (a) {
  var c = a+b;
  return c;
}
document.write(add_b(5));
</script>
```

変数名	値
b	*10*
add_b	*関数 (a){* *var c = a+b;* *return c;* *}*

変数のスコープ

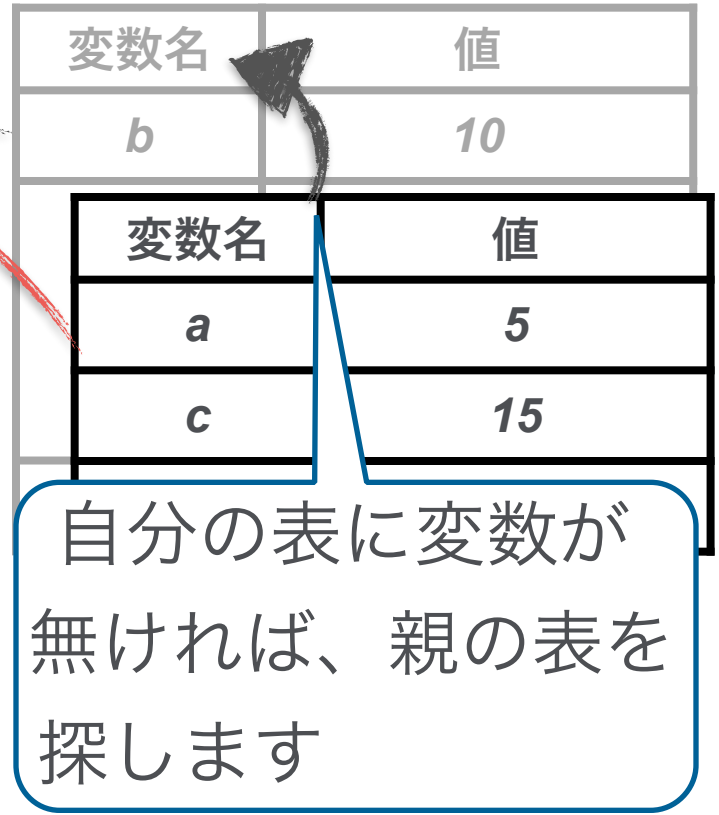
```
<script type="text/javascript">
var b = 10;
function add_b (a) {
  var c = a+b;
  return c;
}
document.write(add_b(5));
</script>
```

変数名	値
b	*10*

変数名	値
a	***5***

変数のスコープ

```
<script type="text/javascript">
var b = 10;
function add_b (a) {
  var c = a+b;
  return c;
}
document.write(add_b(5));
</script>
```



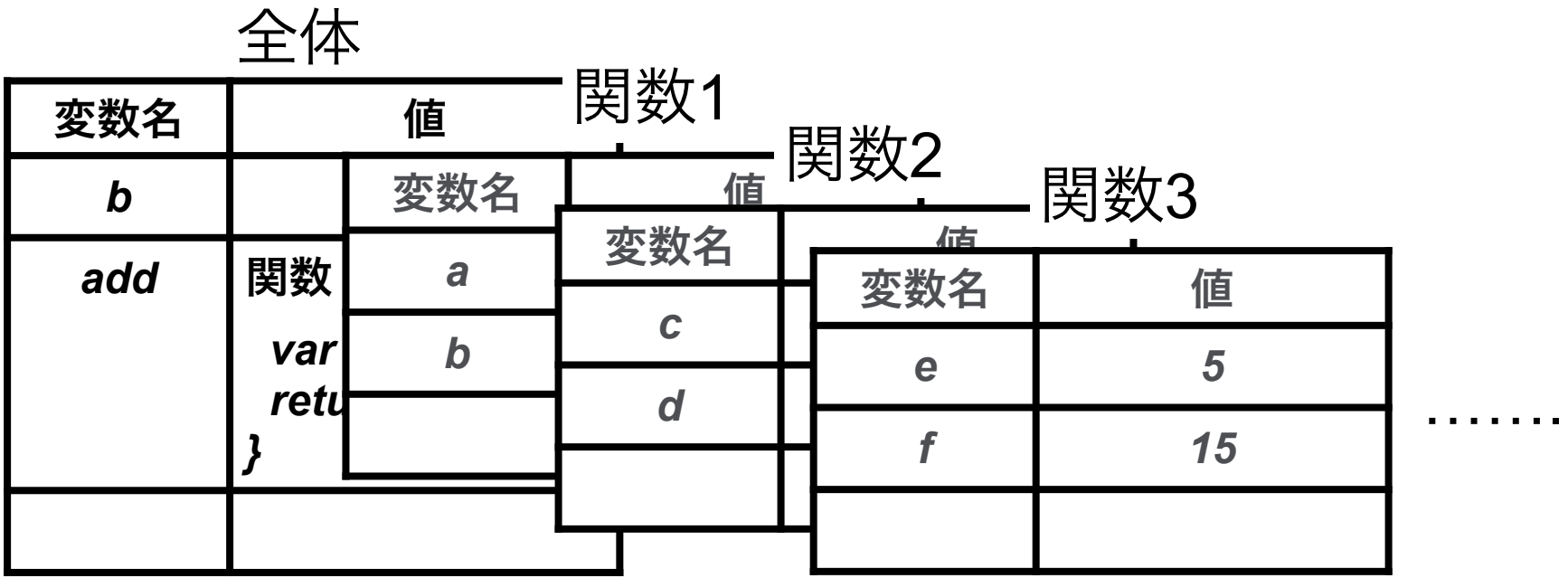
変数のスコープ

```
<script type="text/javascript">
var b = 10;
function add_b (a) {
  var c = a+b;
  return c;
}
document.write(add_b(5));
</script>
```

変数bが使える範囲

変数a,cが使える範囲

スコープチェーン



関数の中から関数を呼び出すと、鎖のように変数の表がつながっていきます。変数を参照すると、子から親へ順に変数名を検索していき、もしも最後まで見つからなければエラーになります。

変数の宣言

```
<script type="text/javascript">
function add_b (a) {
  b = 20;
  var c = a+b;
  return c;
}
document.write(add_b(5)+'<br  />');
document.write(b);
</script>
```

全体

変数名	値
add_b	関数 *(a){* *b = 20;* *var c = a+b;* *return c;* *}*

変数の宣言

```
<script type="text/javascript">
function add_b (a) {
  b = 20;
  var c = a+b;
  return c;
}
document.write(add_b(5)+'<br />');
document.write(b);
</script>
```

変数名	値
add_b	*関数 (a){* *b = 20;* *var c = a+b;* *return c;* *}*

変数名	値
a	*5*

変数の宣言

```
<script type="text/javascript">
function add_b (a) {
  b = 20;
  var c = a+b;
  return c;
}
document.write(add_b(5)+'<br  />');
document.write(b);
</script>
```

変数名	値
add_b	*関数 (a){* *b = 20;* *var c = a+b;* *return c;* *}*
b	*20*

変数名	値
a	*5*

変数の宣言

```
<script type="text/javascript">
function add_b (a) {
  b = 20;
  var c = a+b;
  return c;
}
document.write(add_b(5)+'<br  />');
document.write(b);
</script>
```

変数名	値

add_b	*関数 (a){
	b = 20;
	var c = a+b;
	return c;
	}*

b	*20*

変数名	値
a	*5*
c	*25*

変数の宣言

var 付きの変数宣言は実行した箇所のローカル変数表に登録

var が付かない変数宣言は全体(グローバル)な変数表に登録

※グローバルな変数名表を乱用すると、別のプログラムの変数を上書きする可能性があるため推奨できません。

問題

```
<script type=“text/javascript">
var b = 10;
function set_b (a) {
  b = a;
}
set_b(5);
document.write(b);
</script>
```

このスクリプトを実行した結果、表示されるのは?

A. 5
B. 10
C. undefined
D. null

変数のスコープ

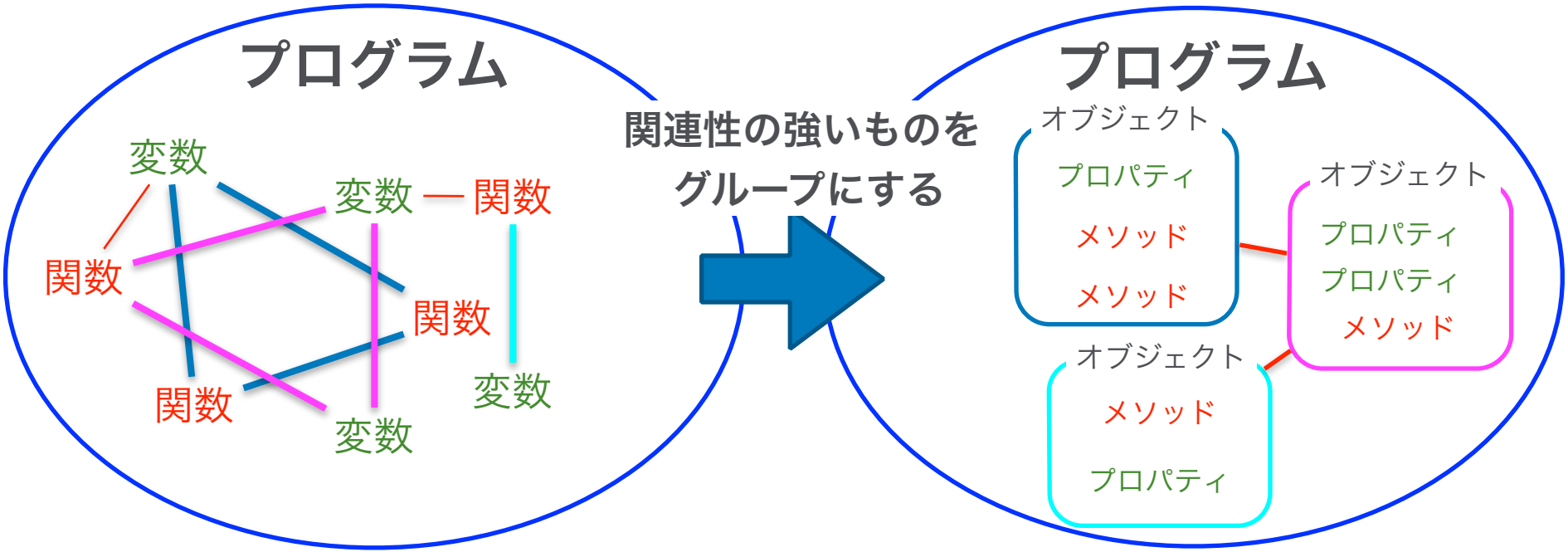
ブラウザで確認

オブジェクト

オブジェクト指向プログラム

- JavaScriptはオブジェクト指向プログラム言語のひとつです。
- オブジェクト指向プログラミングとは、プログラムで扱う対象を”物(オブジェクト)”に見立ててプログラムを作成する手法です。
- プログラムで扱う対象とは、必ずしも実体を共なうものである必要性はありません。時間や数値など概念的なものも、オブジェクトとして扱います。
- JavaScriptでは全ての値がオブジェクトとして扱えるようになっています。
  - これにはちょっとしたトリックがありますが、後で解説します。

オブジェクト



オブジェクトとは、変数と関数をグループにして
プログラムの構造をわかりやすくする仕組み

オブジェクトリテラル

```
<script type="text/javascript">
var obj = { a : 1 , ‘b’ : 2 , 3 : ‘A’  };
document.write(obj.a+’<br />’);
document.write(obj.b+’<br />’);
document.write(obj[3]);
</script>
```

変数名	値

オブジェクトリテラル

```
<script type="text/javascript">
var obj = { a : 1 , ‘b’ : 2 , 3 : ‘A’  };
document.write(obj.a+’<br />’);
document.write(obj.b+’<br />’);
document.write(obj[3]);
</script>
```

変数名	値
obj	オブジェクト プロパティ名 / 値 *a* / *1* *‘b’* / *2* *3* / *‘A’*

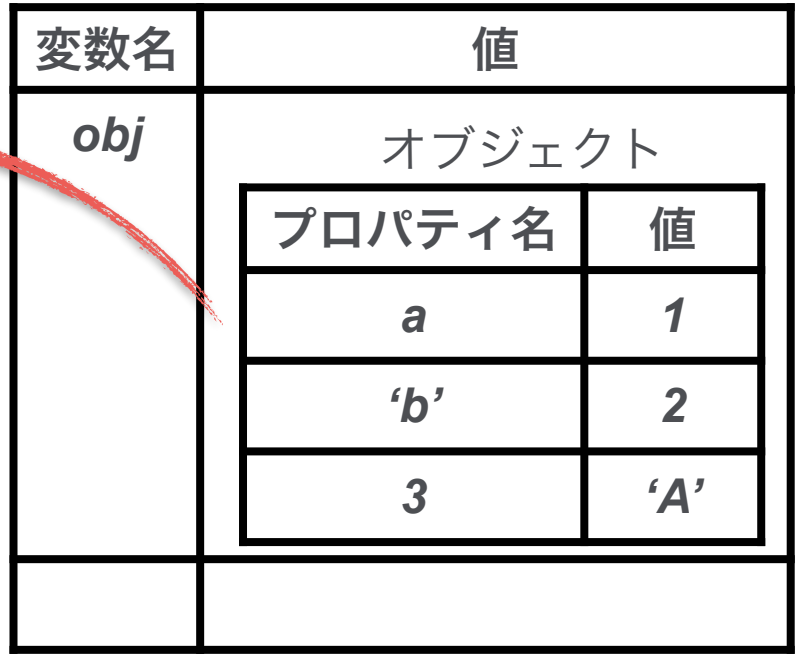
プロパティアクセス

```
<script type="text/javascript">
var obj = { a : 1 , ‘b’ : 2 , 3 : ‘A’  };
document.write(obj.a+’<br />’);
document.write(obj.b+’<br />’);
document.write(obj[3]);
</script>
```

変数名	値
obj	オブジェクト プロパティ名 / 値 *a* / *1* *‘b’* / *2* *3* / *‘A’*

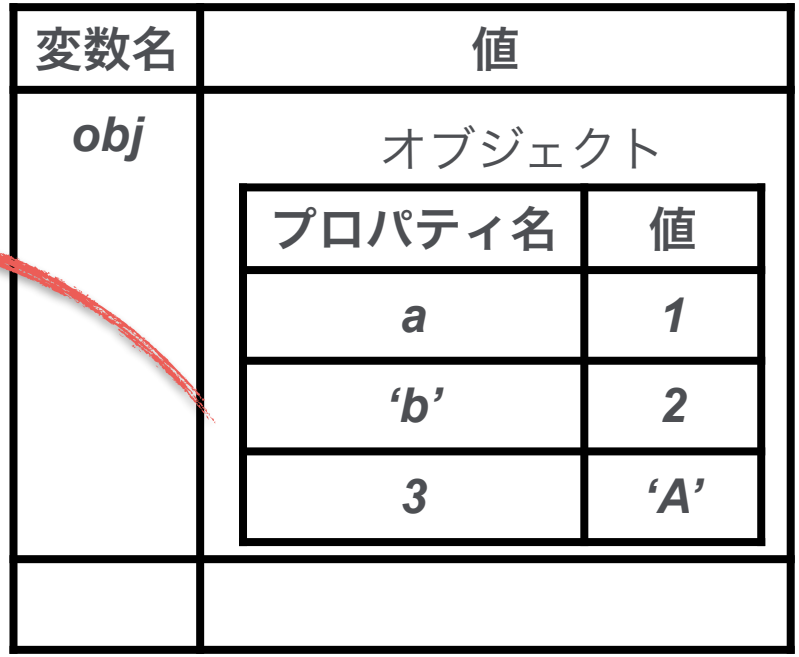
プロパティアクセス

```
<script type="text/javascript">
var obj = { a : 1 , ‘b’ : 2 , 3 : ‘A’  };
document.write(obj.a+’<br />’);
document.write(obj.b+’<br />’);
document.write(obj[3]);
</script>
```



プロパティアクセス

```
<script type="text/javascript">
var obj = { a : 1 , ‘b’ : 2 , 3 : ‘A’  };
document.write(obj.a+’<br />’);
document.write(obj.b+’<br />’);
document.write(obj[3]);
</script>
```



プロパティアクセス

```
<script type="text/javascript">
var obj = { a : 1 , ‘b’ : 2 , 3 : ‘A’  };
document.write(obj.a+’<br />’);
document.write(obj.b+’<br />’);
document.write(obj[3]);
</script>
```

変数名	値
obj	オブジェクト

プロパティ名	値
a	*1*
‘b’	*2*
3	*‘A’*

プロパティアクセス

```
<script type="text/javascript">
var obj = { };
obj.a = 1;
obj[‘b’] = 2;
obj[3]  = ‘A’;
</script>
```

変数名	値

プロパティアクセス

```
<script type="text/javascript">
var obj = { };
obj.a = 1;
obj[‘b’] = 2;
obj[3]  = ‘A’;
</script>
```

変数名	値
obj	オブジェクト

プロパティ名	値
a	*1*
‘b’	*2*
3	*‘A’*

メソッド

```
<script type="text/javascript">
var obj = { add : function (a,b) {
  return a+b;
} };
document.write( obj.add(3,4) );
</script>
```

変数名	値
obj	オブジェクト

プロパティ名	値
add	*function (a,b) { return a+b; }*

this

```
<script type=“text/javascript">
function showThis () {
  document.write(this+’<br />’);
}
showThis();
var obj = { ‘showThis’:showThis}
obj.showThis();
</script>
```

JavaScriptには、thisという読取りしかできない(代入できない)変数のようなものがあります。thisは厳密には変数やプロパティではありません。

this

```
<script type=“text/javascript">
function showThis () {
  document.write(this+’<br />’);
}
showThis();
var obj = { ‘showThis’:showThis}
obj.showThis();
</script>
```

オブジェクトが指定されていない場合はthisはwindowを指します

オブジェクトが指定された場合はthisはオブジェクトを指します

グローバル オブジェクトの例

オブジェクト名	用途/機能
Array	配列を管理する。要素の追加、削除や要素数の取得など
Boolean	論理値*(true,false)*のラッパーオブジェクト
Date	日付、日時の管理。現在時刻の取得、表示用の整形など
Error	プログラムでエラーが発生すると生成。エラーの内容。
JSON	*JSON*形式データの管理。
Math	四則演算など基礎的な計算以外の演算。三角関数など
Number	数値のラッパーオブジェクト
Object	全てのオブジェクトの元になるオブジェクト
String	文字列のラッパーオブジェクト

Arrayオブジェクト

```
<script type="text/javascript">
var array = new Array('HTML5');
document.write( array.length+'<br />' );
array.push('JavaScript');
document.write( array.length+'<br />' );
</script>
```

Arrayオブジェクト

```
<script type="text/javascript">
var array = new Array('HTML5');
document.write( array.length+'<br />' );
array.push('JavaS
document.write( a                    );
</script>
```

グローバルオブジェクトから新しいオブジェクトを作る場合はnewを付けます

Arrayオブジェクト

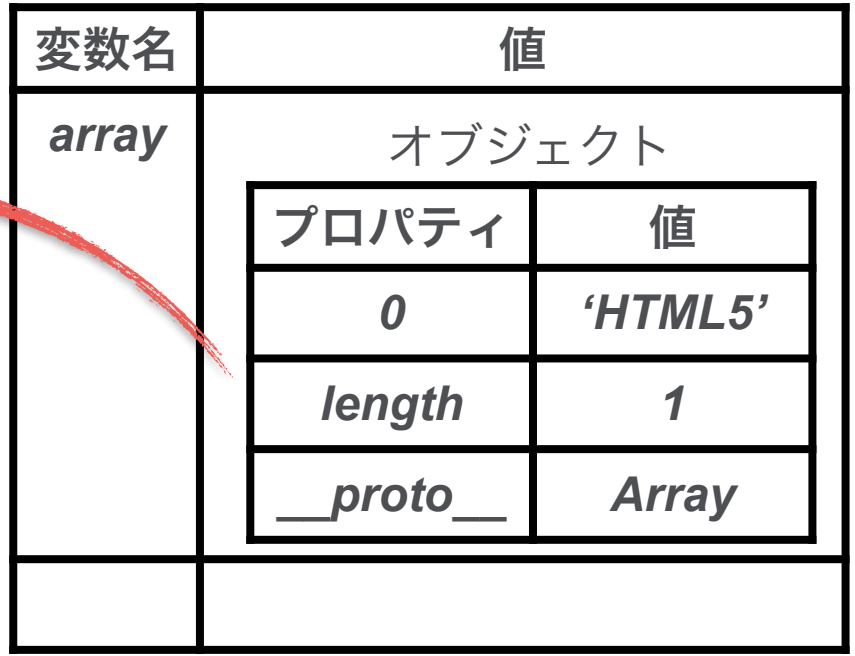
```
<script type="text/javascript">
var array = new Array('HTML5');
document.write( array.length+'<br />' );
array.push('JavaScript');
document.write( array.length+'<br />' );
</script>
```

変数名	値
array	オブジェクト

プロパティ	値
0	*'HTML5'*
length	*1*
__proto__	*Array*

Arrayオブジェクト

```
<script type="text/javascript">
var array = new Array('HTML5');
document.write( array.length+'<br />' );
array.push('JavaScript');
document.write( array.length+'<br />' );
</script>
```



Arrayオブジェクト

pushメソッドは配列の末尾に要素を追加します

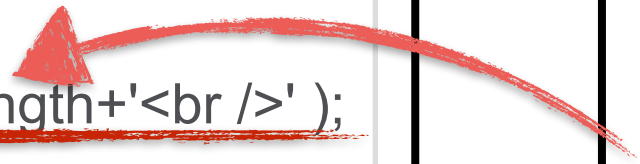
```
<script ty                    >
var array                     L5');
document.w      ite( array.length+'<br />' );
array.push('JavaScript');
document.write( array.length+'<br />' );
</script>
```

変数名	値
array	オブジェクト

プロパティ	値
0	*'HTML5'*
1	*'JavaScript'*
length	*2*
__proto__	*Array*

Arrayオブジェクト

```
<script type="text/javascript">
var array = new Array('HTML5');
document.write( array.length+'<br />' );
array.push('JavaScript');
document.write( array.length+'<br />' );
</script>
```



変数名	値
array	オブジェクト

プロパティ	値
0	*'HTML5'*
1	*'JavaScript'*
length	*2*
__proto__	*Array*

コンストラクタ関数

```
<script type="text/javascript">
function Item(name,price) {
  this.name = name;
  this.price = price;
  this.getTaxIncluded = function() {
    return this.price * 1.08;
 }
}
var item = new Item('apple',100);
document.write( item.name);
document.write('/price:¥'+item.getTaxIncluded());
</script>
```

new

new

thisに空のオブジェクトをセットする
コンストラクタ関数を呼び出す

コンストラクタ関数

thisに対して
プロパティ、メソッドをセットする

thisが参照しているオブジェクトを戻り値にする

コンストラクタ関数

```
<script type="text/javascript">
function Item(name,price) {
  this.name = name;
  this.price = price;
}
var item = new Item('apple',100);
</script>
```

変数名	値

this

windowオブジェクト

コンストラクタ関数

```
<script type="text/javascript">
function Item(name,price) {
  this.name = name;
  this.price = price;
}
var item = new Item('apple',100);
</script>
```

変数名	値

Item	function Item(name,price) {
	this.name = name;
	this.price = price;
	}

this

windowオブジェクト

コンストラクタ関数

```
<script type="text/javascript">
function Item(name,price) {
  this.name = name;
  this.price = price;
}
var item = new Item('apple',100);
</script>
```

変数名	値

Item	function Item(name,price) {
	this.name = name;
	this.price = price;
	}

this

プロパティ	値

新しい空のオブジェクト

コンストラクタ関数

```
<script type="text/javascript">
function Item(name,price) {
  this.name = name;
  this.price = price;
}
var item = new Item('apple',100);
</script>
```

this

プロパティ	値
name	*'apple'*

変数名	値

Item	function Item(name,price) {
	this.name = name;
	this.price = price;
	}

コンストラクタ関数

```
<script type="text/javascript">
function Item(name,price) {
  this.name = name;
  this.price = price;
}
var item = new Item('apple',100);
</script>
```

this

プロパティ	値
name	*'apple'*
price	*100*

変数名	値

Item	function Item(name,price) {
	this.name = name;
	this.price = price;
	}

コンストラクタ関数

```
<script type="text/javascript">
function Item(name,price) {
  this.name = name;
  this.price = price;
}
var item = new Item('apple',100);
</script>
```

this

windowオブジェクト

コンストラクタ関数から出ると元に戻る

変数名	値

Item	function Item(name,price) {
	this.name = name;
	this.price = price;
	}

item	

プロパティ	値
name	*'apple'*
price	*100*

関数オブジェクト

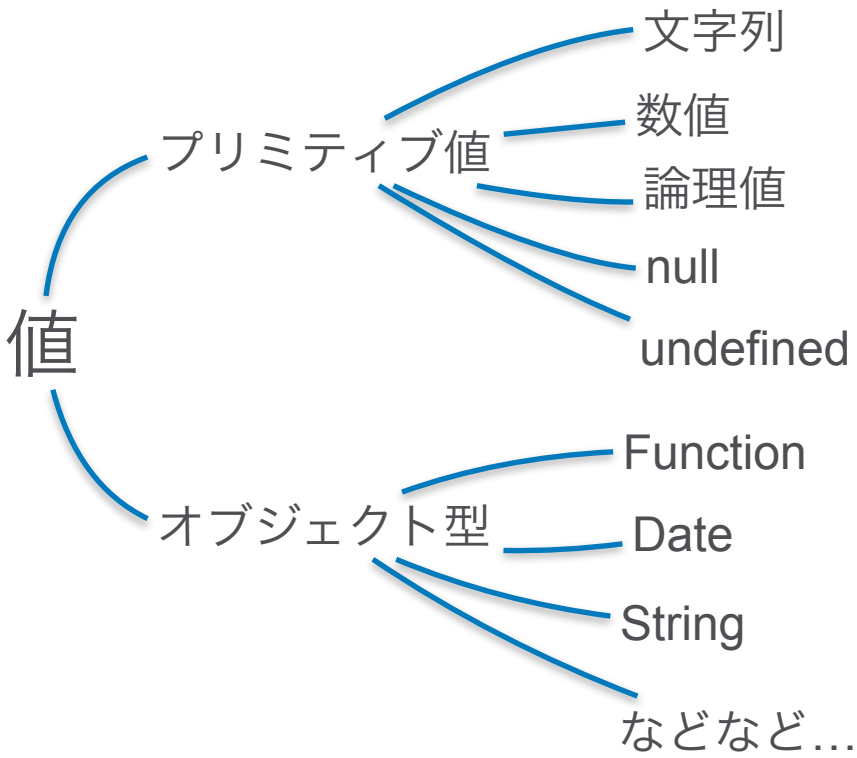
```
<script type="text/javascript">
document.write(Number.MAX_VALUE);
document.write('<br />');
document.write(Number.MIN_VALUE);
</script>
```

最大値(MAX_VALUE) 約1.79×10^{308}

最小の絶対値(MIN_VALUE) 約5×10^{-324}

- Numberコンストラクタ関数のリファレンスを見ると、MAX_VALUE,MIN_VALUEというプロパティがあります。
- 実は、JavaScriptでは関数もオブジェクトなのです。つまり、Numberという変数の値は、関数であり、かつオブジェクトでもあります

JavaScriptの値



オブジェクト型とプリミティブ型

- プリミティブ型(基本データ型)
  - 数値(number)、文字列(string)、真偽値(boolean)、null、undefined
- オブジェクト(object)型
  - 上記プリミティブ型以外
  - プリミティブ型には、対応するオブジェクト(ラッパーオブジェクト)がある
    - Number,String,Boolean,null,undefined
  - 実際にプログラムで使用されるのは、NumberとStringぐらい。
    - Boolean,null,undefinedは「全ての値をオブジェクトとして扱える」という仕組みの整合性のために用意されている

オブジェクト型とプリミティブ型

- オブジェクトはメモリも(比較的)必要だし、処理が(比較的)遅い
  - 頻繁につかうものは特別扱いしよう
    - プリミティブ型
      - プリミティブ型の処理は軽量で高速な分、基本的なことしかできない
      - でも色々な機能を追加したい
        - 必要な時だけオブジェクト化(ラッピング)

オブジェクト型とプリミティブ型

- ‘Text’はプリミティブ型
- new String(‘Text’)は
  - オブジェクト型
- String(‘Text’)は
  - プリミティブ型
- new String(‘Text’).slice(2)というメソッドの戻り値は…
  - ‘xt’
  - プリミティブ型

コンストラクタ関数の戻り値

- new String(‘Text’)とString(‘Text’)、同じ関数なのに、newの有無で戻り値が違うのはどのような仕組みによるものでしょうか。
- JavaScriptには、newを使って関数を呼び出すと、その関数の戻り値によって結果が変わるというルールがあります。

関数の戻り値	newの結果
undefined (関数にreturnが無い)	this
オブジェクト	オブジェクト
プリミティブ型	this

問題

```
<script type=“text/javascript">
var a = Number(‘10’);
var b = new Number(20);
</script>
```

正しいデータ型は?

A)

変数名	値
a	プリミティブ
b	オブジェクト

B)

変数名	
a	プリミティブ
b	プリミティブ

C)

変数名	値
a	オブジェクト
b	プリミティブ

typeof演算子

```
<script type=“text/javascript">
var a = Number(10);
var b = new Number(20);
document.write(typeof a+’<br />’);
document.write(typeof b+’<br />’);
</script>
```

クロージャ

クロージャ

```
function adder(num) {
  return function ( a ) {
    return num + a;
  };
}
var add_2 = adder(2);
document.write( add_2(5)+'<br />' );
var add_10 = adder(10);
document.write( add_10(5) );
```

変数名	値

クロージャ

```
function adder(num) {
  return function ( a ) {
    return num + a;
  };
}
var add_2 = adder(2);
document.write( add_2(5)+'<br />' );
var add_10 = adder(10);
document.write( add_10(5) );
```

変数名	値

adder	function adder(num) {
	  return function (a) {
	  return num + a;
	  }
	}

クロージャ

```
function adder(num) {
  return function ( a ) {
    return num + a;
  };
}
var add_2 = adder(2);
document.write( add_2(5)+'<br />' );
var add_10 = adder(10);
document.write( add_10(5) );
```

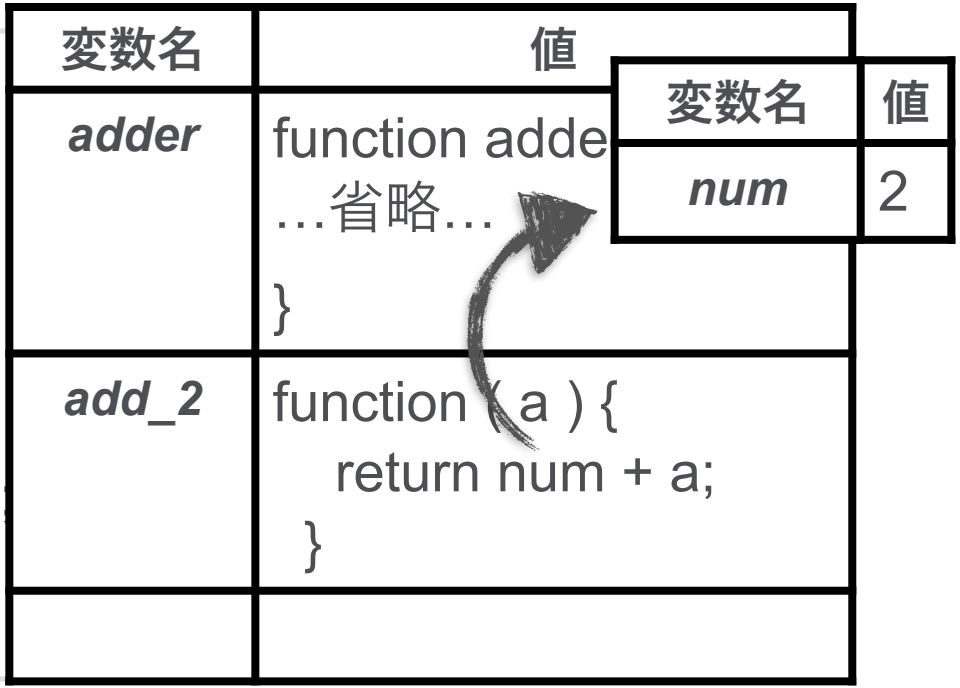
変数名	値
num	2

変数名	値

ler	function adder(num) {
	  return function (a) {
	  return num + a;
	  }
	}

クロージャ

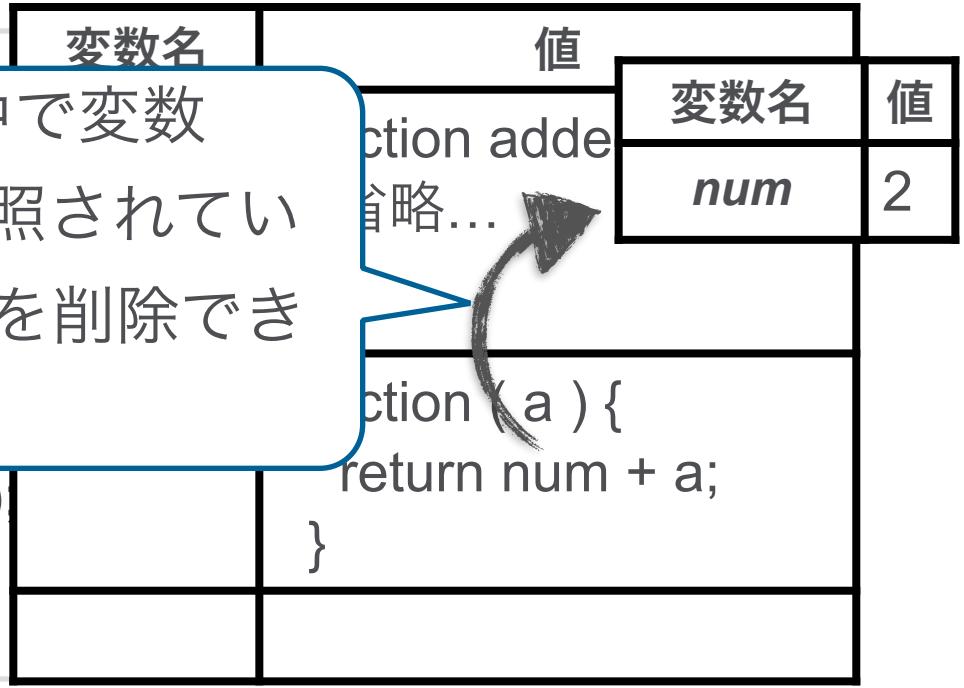
```
function adder(num) {
  return function ( a ) {
    return num + a;
  };
}
var add_2 = adder(2);
document.write( add_2(5)+'<br />' );
var add_10 = adder(10);
document.write( add_10(5) );
```



クロージャ

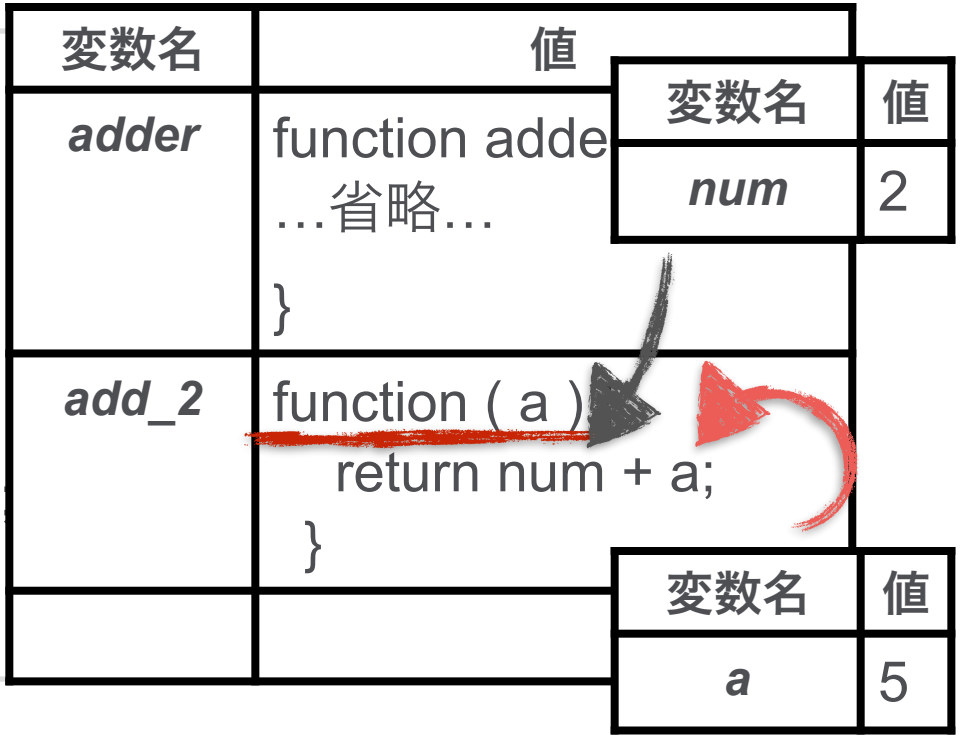
```
function adder(num) {
  return function ( a ) {
    return num + a;
  };
}
var add_2 = adder(2);
document.write( add_2(5)+'<br />' );
var add_10 = adder(10);
document.write( add_10(5) );
```

関数の中で変数numが参照されているので表を削除できません



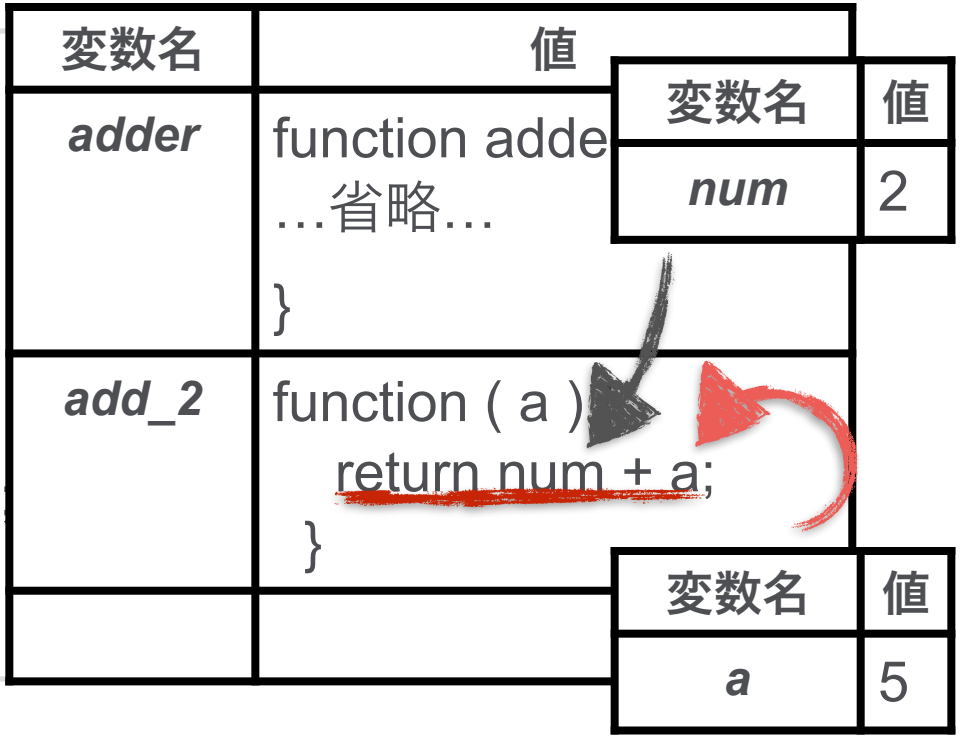
クロージャ

```
function adder(num) {
  return function ( a ) {
    return num + a;
  };
}
var add_2 = adder(2);
document.write( add_2(5)+'<br />' );
var add_10 = adder(10);
document.write( add_10(5) );
```



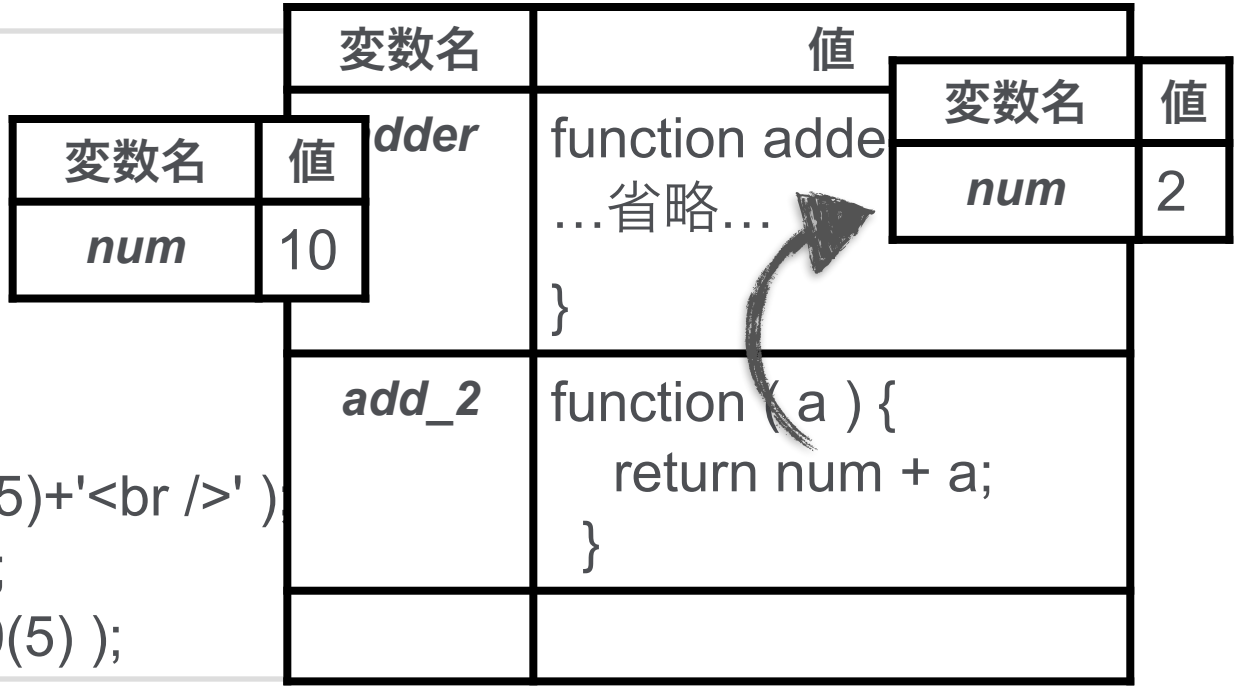
クロージャ

```
function adder(num) {
  return function ( a ) {
    return num + a;
  };
}
var add_2 = adder(2);
document.write( add_2(5)+'<br />' );
var add_10 = adder(10);
document.write( add_10(5) );
```



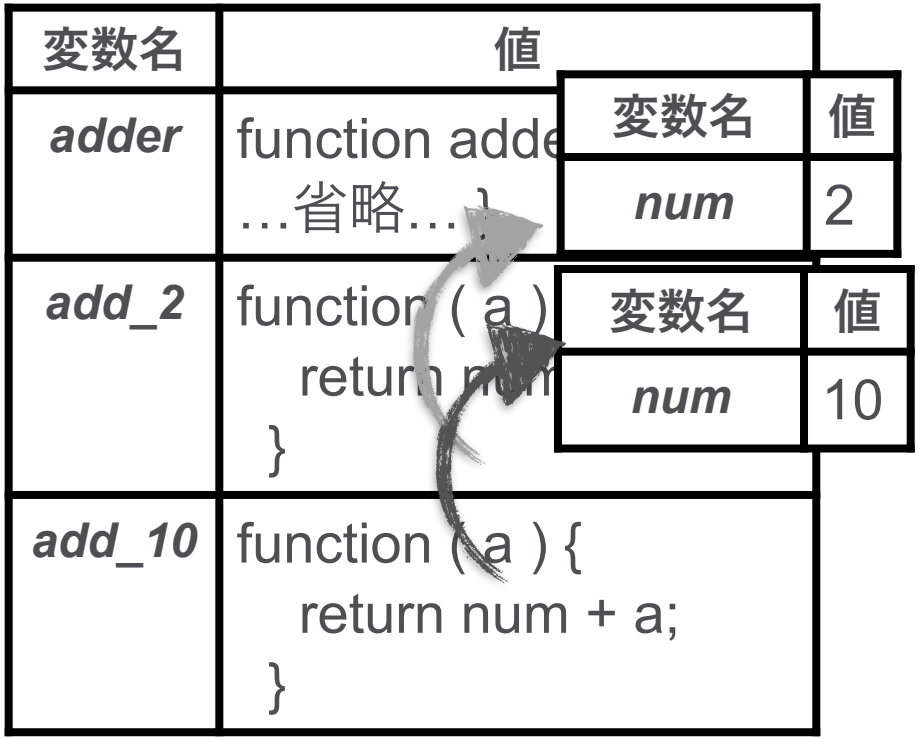
クロージャ

```
function adder(num) {
  return function ( a ) {
    return num + a;
  };
}
var add_2 = adder(2);
document.write( add_2(5)+'<br />' );
var add_10 = adder(10);
document.write( add_10(5) );
```



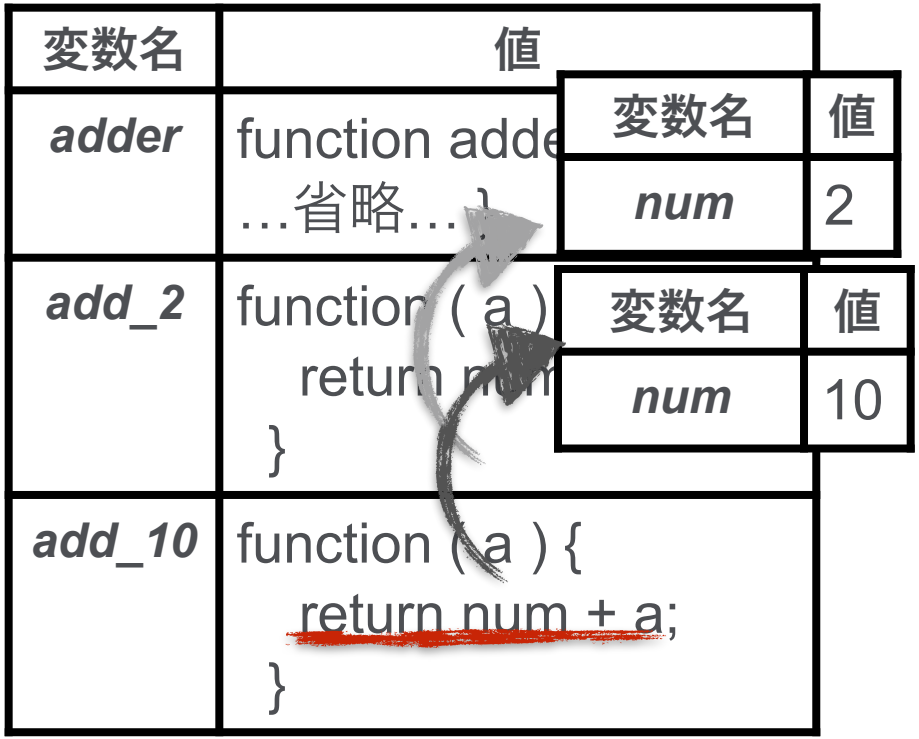
クロージャ

```
function adder(num) {
  return function ( a ) {
    return num + a;
  };
}
var add_2 = adder(2);
document.write( add_2(5)+'<br />' );
var add_10 = adder(10);
document.write( add_10(5) );
```



クロージャ

```
function adder(num) {
  return function ( a ) {
    return num + a;
  };
}
var add_2 = adder(2);
document.write( add_2(5)+'<br />' );
var add_10 = adder(10);
document.write( add_10(5) );
```



クロージャの見分けかた

1. 関数の中で新しい関数(または関数リテラル)が作られている
2. 新しい関数定義に祖先の関数のローカル変数(引数も含む)が含まれている

```
function adder(num) {
  return function ( a ) {
    return num + a;
  }
}
```

1.新しい関数リテラル

2.祖先の関数のローカル変数

クロージャのメリット

- グローバルな変数を作らずに関数の動作を変えられる
- 実行するときに余分な引数を必要としない関数を作ることができる
- クロージャを作成したときの情報を持ち続けられる

- オブジェクトと用途的には似ている(状態を持つ)
  - プロパティ≒祖先の変数、メソッド≒クロージャ本体

Callオブジェクト

Callオブジェクト

プロパティ	値
ローカル変数/引数	変数名 / 値
this	
親の *Call*オブジェクト	
などなど	

いままで変数表という名前で説明してきたものは、正しくはCallオブジェクトと言うオブジェクトです。

関数が呼び出されると、新しいCallオブジェクトが作られ、引数や関数内のローカル変数、this等がプロパティとして登録されます。

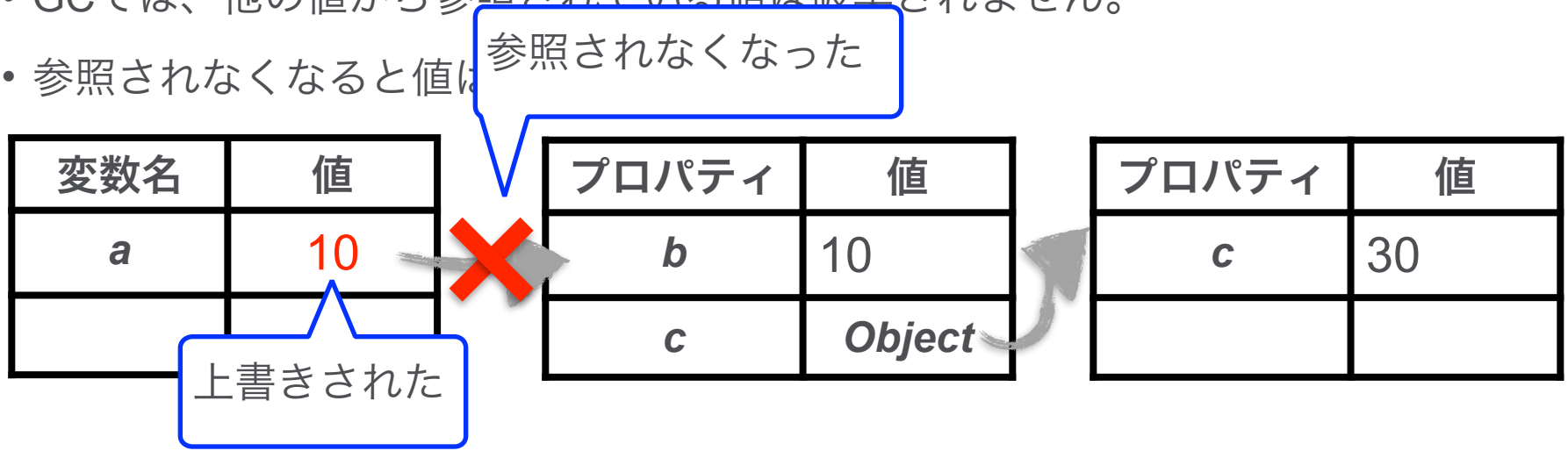
CallオブジェクトはJavaScriptのプログラムからは直接操作できません。

ガーベージコレクション

- 不要になった値は、メモリを空けるために破棄されます。
- このしくみをがガーベージコレクション(ごみ集め)、略してGCと呼びます。
- GCでは、他の値から参照されている値は破棄されません。
- 参照されなくなると値は破棄されます。

ガーベージコレクション

- 不要になった値は、メモリを空けるために破棄されます。
- このしくみをがガーベージコレクション(ごみ集め)、略してGCと呼びます。
- GCでは、他の値から参照されている値は破棄されません。
- 参照されなくなると値は



ガーベージコレクション

- 不要になった値は、メモリを空けるために破棄されます。
- このしくみをがガーベージコレクション(ごみ集め)、略してGCと呼びます。
- GCでは、他の値から参照されている値は破棄されません。
- 参照されなくなると値は破棄されます。

ガーベージコレクション

- 不要になった値は、メモリを空けるために破棄されます。
- このしくみをがガーベージコレクション(ごみ集め)、略してGCと呼びます。
- GCでは、他の値から参照されている値は破棄されません。
- 参照されなくなると値は破棄されます。

変数名	値
a	10

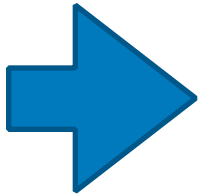
ガーベージコレクション

- 関数の実行が終わると、Callオブジェクト(変数表)もGCの対象になります。
- しかし、クロージャという形で変数表の中の変数への参照が残ったままだと、GCによる破棄がおこなわれません。(破棄されるとクロージャを実行できない)
- 場合によっては不要な値が残りメモリを圧迫することがありますので、クロージャへの参照を破棄する必要があります。

小ネタ

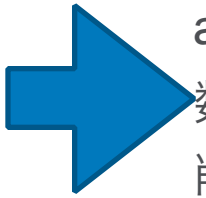
Arrayオブジェクトの要素数

```
<script type="text/javascript">
var array = new Array();
array[0] = 'HTML5';
array[9] = 'JavaScript';
document.write(array.length);
</script>
```



array.lengthはインデックスのうち
最大の数値+1を返す

```
<script type="text/javascript">
var array = new Array('a','b','c','d');
array.length = 1;
document.write(array);
</script>
```



array.lengthに数値を代入すると
数値以上のインデックスの要素が
削除される

strictモード

```
(function () {
   "use strict”;
   b = 1; // 暗黙のグローバル変数は禁止
   NaN = 1; // NaNは書き込み禁止
   Infinity = 1; // Infinityは書き込み禁止
   undefined = 1; // undefiendは書き込み禁止
   var obj = {a:1,a:2}; // 同名プロパティ禁止
   function f(a,a) { return a;}  // 同名仮引数禁止
})( );
```

“use strict”;とプログラム先頭、または関数先頭に記述するとstrictモードになります。

strictモードでは、気付きにくい落とし穴をエラーにしてくれます。

質疑応答

ご静聴 ありがとうございました