

A Brain-Friendly Guide

Head First

JavaScript

頭とからだで覚えるJavaScriptの基本



Improve your user experience with web page interactivity



Whip up working JavaScript code in a snap



Stop fearing event handling—it won't hurt a bit



Load important JavaScript concepts directly into your brain



Slice and dice HTML with help from the DOM



Band your mind around dozens of puzzles and exercises

O'REILLY®

オライリー……、ジャパン pools@maruhouse.org personal use only.

Michael Morrison 著
豊福 剛 訳

Head First JavaScript

頭とからだで覚えるJavaScriptの基本

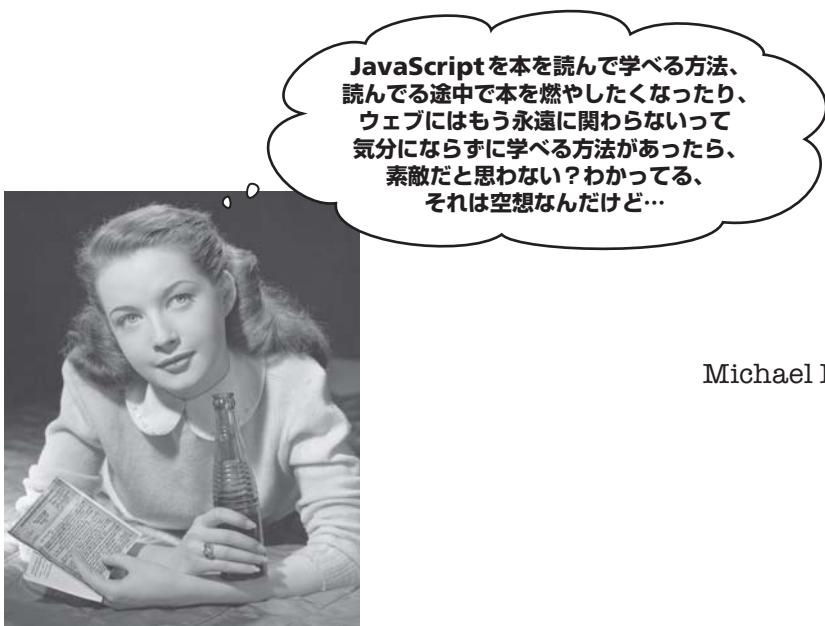
Michael Morrison 著

豊福 剛 訳

O'REILLY®
オライリー・ジャパン

本文中の製品名は、一般に各社の登録商標、商標、または商品名です。
本文中ではTM、[®]、[©]マークは省略しています。

Head First JavaScript



Michael Morrison

O'REILLY®

Beijing · Cambridge · Farnham · Köln · Sebastopol · Taipei · Tokyo

© 2008 O'Reilly Japan, Inc. Authorized translation of the English edition © 2007 O'Reilly Media, Inc.
This translation is published and sold by permission of O'Reilly Media, Inc., the owner of all rights to
publish and sell the same.

本書は、株式会社オライリー・ジャパンがO'Reilly Media, Inc.との許諾に基づき翻訳したものです。日本語版
についての権利は、株式会社オライリー・ジャパンが保有します。

日本語版の内容について、株式会社オライリー・ジャパンは、最大限の努力をもって正確を期していますが、
本書の内容に基づく運用結果についての責任は負いかねますので、ご了承ください。

何も実行できないページのリンクを集めた巨大なオンラインブックではなく、それ以上のウェブが可能だと夢みた前世紀のNetscapeの人たちへ。

もちろん彼らはぞつとするような<blink>タグを思いついたりもしましたが…思いつくのは勝手だけど、あまり変なタグを作らないで！

『Head First JavaScript』を書いた人

Michael Morrison
JavaScriptの神童。



Michael Morrison
成長を拒否した実物大の
ナード。



Michael Morrisonはずっとコンピュータをいじってきました。最初のPCはTI-99/4A。人間工学を駆使したキーボード、最先端の白黒TVモニター、カセットテープ記憶装置のセットでした。その頃から他にも数台持っていましたが、裏庭でNerfのフットボールゲームを楽しんでいるときも、TIでシューティングゲームParsecに夢中だった日々を懐かしく思っていました。

いまや大人になったMichaelの関心は、インタラクティブなウェブアプリケーションの開発とスケートボードです。切り傷、擦り傷、ときには足をひきづりながら、ハイリスクのスポーツと同じように、がむしゃらに技術に挑戦しています。

ビデオゲームの開発、おもちゃの発明、数十冊のコンピュータ本の執筆、オンラインコースの作成を手掛けてきたMichaelは、ついに Head First JavaScript に取り組む準備ができたと感じました。もう気ままな彼ではありません。

その結果、Head First シリーズの本を執筆する準備ができる人なんていないことがわかりました。ここで最善の選択は、赤のピルを飲み込んで、Head Firstのマトリックスに飛び込むことです。知的な傷を負いながら、あちらの世界からこちらの現実世界に戻ってきたMichaelは、学ぶこと(教えること)を以前と同じようには考えないでしょう。この事実に彼は高揚しています。そしてインタラクティブなウェブの不思議な世界を反映したコイの池のほとりで妻と一緒に暮らしています。

目次 (要約)

序章	xxi
1 章	インタラクティブなウェブ：応答するバーチャルワールド	1
2 章	データを格納する：あらゆる物には然るべき場所がある	33
3 章	ブラウザを調べる：ブラウザを探検する	85
4 章	意思決定：道が分かれていたら、どっちに進むか決めなさい	135
5 章	ループ：同じことが重複するのは危険	189
6 章	関数：節約、再利用、リサイクル	243
7 章	フォームと検証：ユーザに洗いざらい話してもらう	289
8 章	ページの部品を書き集める：HTMLをDOMで切る	343
9 章	データを活きづける：オブジェクトはフランケンデータ	393
10 章	カスタムオブジェクトを作成する：カスタムオブジェクトを 思い通りに	449
11 章	バグをなくせ：良いスクリプトも悪くなる	485
12 章	ダイナミックなデータ：感度良好なウェブアプリケーション	537
	索引	599

目次

序章

これからあなたの脳はJavaScriptに取り組みます。何かをぼんやりと学ぼうとしているかぎり、あなたの脳はこうした学習を重要ではないと言い続けます。「もっと重要なことのために余裕を残しておかなきゃ。野獣に襲われないようにするとか、裸で水上スキーをするのはよした方がいいかなとか」とあなたの脳は言うわけです。あなたの脳にJavaScriptを学ぶことがとっても重要だと考えさせるにはどんな工夫をすればいいでしょう？

この本を読むのにふさわしいのは誰でしょう？	xxii
あなたがどう思っているのか、わかってます	xxiii
メタ認知：思考についての思考	xxv
あなたの脳を思い通りに使うためにできること	xxvii
読んでね	xxviii
テクニカルレビューチーム	xxx
謝辞	xxxii

1

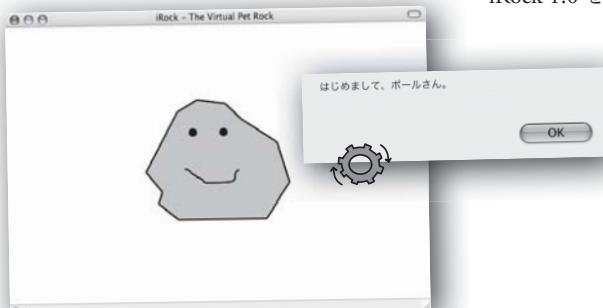
章 インタラクティブなウェブ

応答するバーチャルワールド

ウェブを受け身なページと考えるのに飽きていませんか？

受け身なページといえば本がそうです。本は読んだり学んだりするのには適していますが**インタラクティブ**ではありません。ウェブもJavaScriptのちょっとした助けがなければ**インタラクティブ**ではありません。もちろんフォームを送信できるし、HTMLとCSSを駆使してあちこちに仕掛けをつくることもできるでしょうが、無機的なウェブページを底上げしているにすぎません。ほんとうに**インタラクティブ**になるには、**もう少し賢く動作させる**必要がありますが、**それ以上の見返り**があります。

(オンラインの) ユーザが必要としていること	2
壁にむかって話しかけても、何も起こりません	3
でも JavaScriptなら応答します	4
ライト、カメラ、インターアクション	6
<script> タグを使ってブラウザに JavaScript が書かれていることを知らせます	11
ウェブブラウザは HTML と CSS、そして JavaScript を処理できます	12
バーチャルなお友だちは、かまってほしいみたいですね	15
iRock をインタラクティブにする	16
iRock ウェブページを作成する	17
テスト運転	17
JavaScript のイベントを使って iRock の「声」を実現する	18
関数を使ってユーザにアラート表示する	19
iRock に挨拶機能を追加する	20
iRock を本格的にインタラクティブにしてみましょう	22
インタラクティブとは双方向のコミュニケーション	23
ユーザの名前を受け取る関数を追加する	24
即席リプレイで動作を確認	27
iRock 1.0 をテスト運転	28



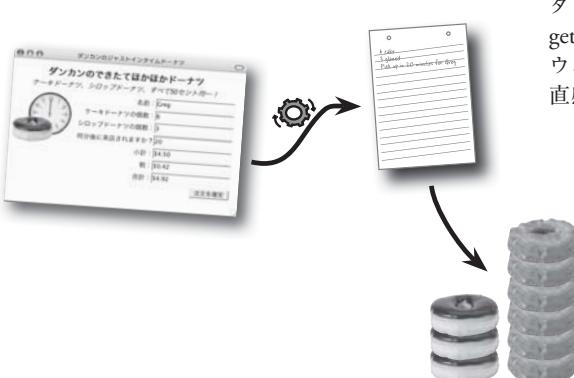
9

章 データを格納する

あらゆる物には然るべき場所がある

現実世界では人々は物を収納する場所の重要性をしばしば見落としますが、JavaScriptでは見落とすわけにはいきません。ウォークインクローゼットや車が3台もに入るガレージのような豪華さはありませんが、JavaScriptではあらゆる物に然るべき場所があるので、然るべき場所に置く責任はあなたにあります。データを表現する方法、格納する方法、いったんどこかに置いた後にそれを見つける方法、これらすべてはデータをどう扱うかという問題です。JavaScriptは収納の専門家です。JavaScriptのデータで散らかった部屋に入って仮想のラベルや収納箱を使えば、あなたの思い通りにデータを扱うことができます。

スクリプトはデータを格納できます	34
スクリプトはデータ型で考える	35
定数は一定ですが変数は変化します	40
変数は値がなくても作成できる	44
変数を "=" で初期化する	45
定数は変更しようとしても変更できません	46
名前に使える文字は?	50
変数名と定数名の合法性	51
変数名にはキャメルケースを使います	52
ダンカンドーナツのウェブページを計画する	56
ドーナツ計算の最初の一歩	58
データを初期化しましょう、そうしないと	61
NaNは数値でない	62
加算できるのは数値だけではありません	64
parseInt() や parseFloat() を使ってテキストを数値に	
変換します	65
どうして大量のドーナツ注文になったのか?	66
ダンカンが見つけた営業妨害	70
getElementById() を使ってフォームデータをつかむ	71
ウェブフォームのデータを検証する	72
直感的なユーザ入力を追求する	77



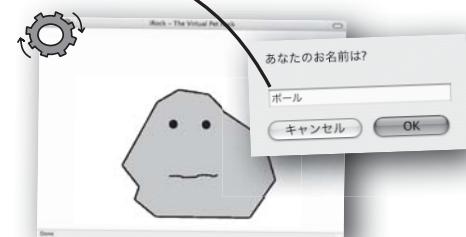


章 ブラウザを調べる

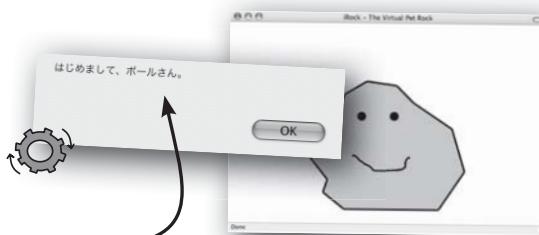
ブラウザを探検する

ときどき JavaScript は自分のまわりで何が起きているか知る必要があります。あなたのスクリプトはウェブページのコードとして開始されるかもしれません、最終的にはクライアントであるブラウザによって作成された世界で動作します。賢いスクリプトは自分が実行される環境について知る必要があるときがあるので、**ブラウザとやり取り**をして環境について情報を得ます。画面の大きさを調べたりブラウザのボタンにアクセスしたりといったブラウザとの関係を通して、スクリプトはおそらくたくさんのことを得ます。

ここから 開始！



終わり！

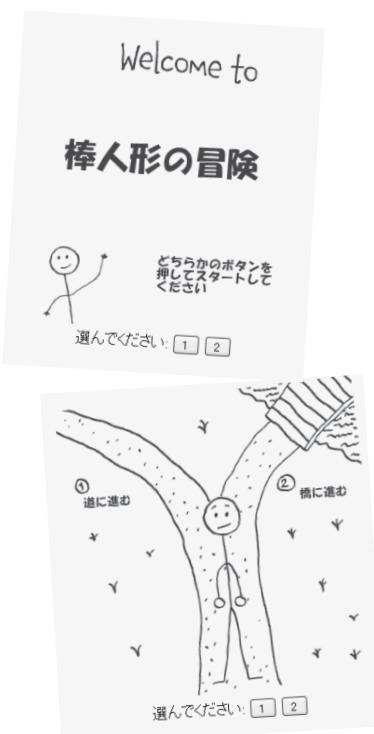


クライアント、サーバ、JavaScript	86
ブラウザはあなたのために何ができる？	88
iRock はもっと反応する必要があります	90
タイマーはアクションと経過時間を結びつけます	92
タイマーを分解する	93
setTimeout() でタイマーを設定する	94
setTimeout() の詳細	95
画面の大きさに不満が殺到	99
document オブジェクトを使ってクライアントウィンドウの幅を取得する	100
document オブジェクトのプロパティを使ってクライアント ウィンドウの幅を設定します	101
iRock 画像の高さと幅を設定する	102
ページに合わせて iRock の大きさを調整する	103
onresize はブラウザの大きさが変更されたとき発生します	107
onresize イベントを使って iRock の大きさを変更します	108
お会いしたことありました？	110
スクリプトにはライフサイクルがある	111
クッキーはスクリプトのライフサイクルを越える	112
クッキーには名前と値と期限があります	117
JavaScript はウェブページの外に置いてもかまいません	119
クッキーを使ってユーザに挨拶しましょう	120
greetUser() がクッキー対応になりました	121
クッキーを設定するのもお忘れなく	122
クッキーはブラウザのセキュリティに影響します	124
クッキーのない世界	126
何もしないより、ユーザに話しかける方がベターです	129

4 章 意思決定

道が分かれていたら、どっちに進むか決めなさい

人生は決断の連続です。止まるか進むか、煮るか焼くか、司法取引に応じるか裁判にかけられるか。決断を決断を下すことができなければ、何も達成することができません。JavaScriptも同様です。決定によってスクリプトはさまざまな可能性からひとつを選びます。意思決定によってスクリプトの「物語」は進行するので、どんなにありふれたスクリプトであってもある種の物語があります。ユーザが入力したものを信用して雪男探検のツアーの予約を受け付けてもいいでしょうか？ それともほんとはバスに乗りたいだけなのか確認した方がいいのでしょうか？ その選択はあなた次第です。



当選者の方、こちらへどうぞ！	136
"if" これが真ならば...何かを実行する	138
if文は条件を評価し、結果に応じてアクションを実行します	139
ifを使って2つのどちらかを選ぶ	141
ifを使って複数の決定ができます	142
if文にelseを追加する	143
変数を使って物語を進行させる	146
物語の一部が見つかりません	147
JavaScriptのアクションを複数にする	148
if/elseを使って段階的決定にする	154
ifの中に別のifを書くことができます	155
ページを関数で制御する	157
擬似コードを使って冒険の綿密な計画を立てる	158
棒人形の不具合	162
!= あのさ、言いたいことは何もないんだけど	163
比較演算子を使って決定を作り込む	164
コメントとドキュメンテーション	166
//で始まるJavaScriptのコメント	167
スコープとコンテキストでデータのライフサイクルが決まる	169
変数のスコアをチェックする	170
データはどこに生存している？	171
5つの選択	174
入れ子になったif/elseは複雑です	175
switch文には複数の選択肢があります	177
switch文の内部構造	178
「棒人形の冒険」のswitchバージョンをテスト運転する	183

5 章 ループ

同じことが重複するのは危険

繰り返しに人生の趣がある、という人もいます。新しくて面白いことはたしかに刺激的ですが、毎日の生活を支えているのは些細なことの繰り返しです。でも、強迫神経症的に手を洗うこと、緊張したときの癖、受け取った異常なメッセージで「全員に返信」をクリックすることなど、現実には繰り返しがいつもいいとは限らないようです。しかし、JavaScriptの世界では繰り返しは最高にお手軽です。スクリプトでは驚くほどコード断片を何度も繰り返し実行する必要があります。ループの力が発揮されるのはそのときです。ループがなければ多くの時間を浪費して無駄なコードのかたまりをカット&ペーストしたことでしょう。

空席



seat_avail.png

空席でない



seat_unavail.png

選択



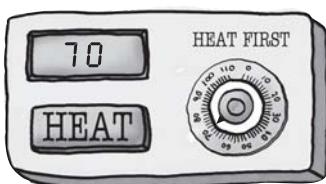
seat_select.png

お宝はXに隠されている	190
デジャヴュのように何度も繰り返すならforループ	191
forループを使ってお宝探し	192
forループを解剖する	193
Mandango：マッチョのための映画館探し	194
最初に空席をチェックします	195
ループとHTMLと座席	196
座席を変数にする	197
配列は複数のデータ断片を集めたもの	198
配列の値はキーとともに格納される	199
JavaScriptからHTMLを操作する	203
Mandangoの座席を表示する	204
座席検索をテスト運転してみる	209
ループが終わらないのは問題です	210
ループには脱出条件が必要	211
アクション中のbreak	212
論理値を扱う論理演算子について	218
whileを使って、ある条件が満たされる間だけループする	222
whileループを分解する	223
問題に適したループを使う	225
映画館の座席データをモデル化する	231
2次元配列は配列の配列	232
2次元データにアクセスするにはキーが2つ必要	233
2次元のMandango	235
全座席からマッチョのための席を探す	238

6 章 関数

節約、再利用、リサイクル

もしJavaScriptの世界に環境運動があるとしたら、運動のリーダーは関数です。関数はJavaScriptのコードをより効率的に、そしてここが肝心なのですが、再利用できるものにします。関数のアピールポイントは、タスク指向であること、コードの編成に優れていること、問題解決の達人であることです。よくできた履歴書みたいですね。実際、単純なスクリプトを除けば、ほとんどのスクリプトは関数で再編成されることで恩恵がもたらされます。平均関数の二酸化炭素排出量を数字で出すのは難しいですが、スクリプトができるだけ環境に優しく作ることで、自分の役割を果たすことにしましょう。



すべての問題の根源	244
問題解決のための関数	246
関数の仕組み	247
これまで見てきた関数	248
データを増やしてサーモスタットを改良する	251
関数に情報を渡す	252
データとしての引数	253
関数を使ってコードの重複をなくす	254
座席を設定する関数を作る	257
setSeat()でMandangoを改良します	259
フィードバックの意義	261
関数からデータを返す	262
返す値を複数もたせる	263
座席の状態を取得する	267
座席の状態を表示する	268
画像と関数を結びつける	269
繰り返しの多いコードは良くない	270
コンテンツから機能を分離する	271
関数なんてただのデータ	272
関数の呼び出しと参照	273
イベント、コールバック、HTML属性	277
関数参照を使ってイベントとつなぐ	278
関数リテラルで解決	279
どこでつながっている？	280
HTMLページの殻	283

7 章 フォームと検証

ユーザに洗いざらい話してもらう

JavaScriptを使ってユーザの情報をうまく聞き出すには、人あたり良く振る舞つたり、あるいはコソコソする必要はありませんが、慎重さは必要です。オンラインのフォームで提供されたデータが正確かつ有効であると前提できないのに、それをそのまま受け取ってしまい失敗することがあります。フォームにデータが入力されたとき **JavaScript**のコードを経由してフォームデータを送信すれば、ウェブアプリケーションの信頼性はもっと高くなりサーバの負荷も軽減されます。人目をひくビデオやかわいいペットの写真などの重要なコンテンツのために、**貴重な帯域を節約する**必要があります。

Bannerocity の HTML フォーム	291
HTML ではできること	292
フォームデータにアクセスする	293
フォームフィールドで発生するイベントの連鎖	295
フォーカスがなくなったときは onblur	296
アラートボックスを使って検証メッセージを表示できます	297
フィールドが空でないか検証する	301
うるさいアラートボックスを使わない	302
空でない検証をさらに改善する	303
サイズの問題	305
データの長さを検証する	306
郵便番号を検証する	311
日付を検証する	316
正規表現は普通の表現ではありません	318
正規表現はマッチさせるパターンを定義します	319
メタ文字は正規表現を作るときに使う記号です。	321
量化子を使って正規表現を強化する	322
正規表現を使ってデータ検証する	326
繰り返し回数の最小と最大を指定する	329
これか、あれかのパターンで 3 桁数字をなくす	331
ほかのフィールドも見逃さない	332
電話番号を検証する	333
メールアドレスを検証する	334
例外に学ぶ	335
オプション文字を集合にする	336
メールアドレスの検証関数を作る	337

Bannerocity...banner ads in the sky!

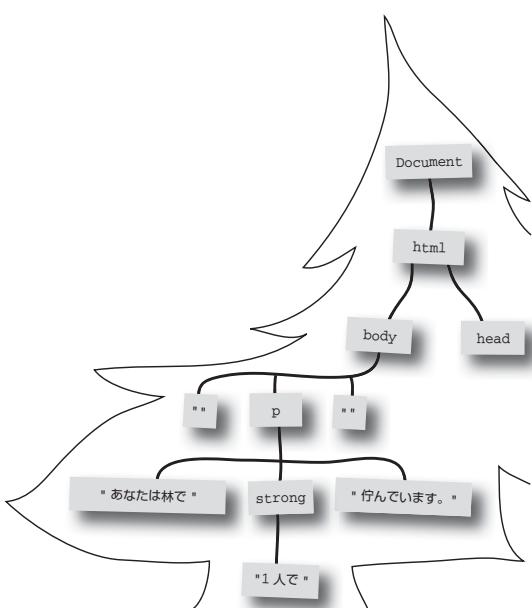


8

章 ページの部品をかき集める HTMLをDOMで切る

JavaScriptを使ってウェブページのコンテンツを制御するのは、オープンで焼く料理に近いところがあります。でも食堂がないのはもちろん、残念なことに食べられるご褒美もありません。とはいえウェブページの材料であるHTMLにアクセスして、ここが重要なんですが、ページのレシピを替えることができます。JavaScriptを使えばウェブページの中にあるHTMLコードをあなたの思いのままに操作できるのです。**DOM** (Document Object Model)と呼ばれる標準オブジェクトの集まりをJavaScriptで操作すれば、ありとあらゆる興味深い操作が可能になります。

機能的だけど退屈なインターフェース	344
アラートボックスを使わずにシーンを説明する	345
HTML要素にアクセスする	347
HTMLの内部にアクセスする	348
木に林を見る：DOM (Document Object Model)	353
ページはDOMノードの集まり	354
DOMツリーをプロパティでたどる	357
DOMを使ってノードのテキストを変更する	360
「冒険」を標準に準拠させる	365
きれいな選択肢ボタンを設計する	367
ノードテキストの置き換えについて再考する	368
関数を使ってノードのテキストを置き換える	369
オプションが動的に変わるのは良いことです	370
オプションをインタラクティブにする	371
CSSとDOMでスタイルを変更する	372
スタイルクラスを入れ替える	373
クラス対応のオプション	374
スタイル対応のオプションをテスト運転する	375
空の選択肢ボタンが表示されてしまう	376
スタイル調整のアラカルト	377
空の選択肢はもう表示しない	379
オプションを増やして、より複雑にする	380
決定ツリーをたどる	382
決定履歴をHTMLにする	383
HTMLコードを組み立てる	384
冒険物語をトレースする	387



9

章 データを活氣づける

オブジェクトはフランケンデータ

JavaScript オブジェクトはフランケンシュタイン博士が作った人造人間みたいにおぞましくはありませんが、**JavaScript** 言語の断片や部品をつなぎ合わせて、より強力なものになります。オブジェクトはデータとアクションを組み合わせた新しいデータ型になります。これまでみてきたデータ型より活動的です。自分自身をソートできる配列、自分自身を検索できる文字列。毛がはえて月に吠える狼男スクリプトなんてどうですか？

データ

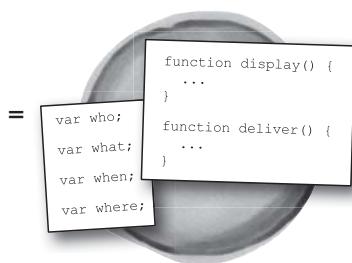
```
var who;
var what;
var when;
var where;
```

+

アクション

```
function display(what, when, where) {
}
function deliver(who) {
}
```

オブジェクト



JavaScriptでパーティの招待状を作る	394
データ + アクション = オブジェクト	395
オブジェクトには固有のデータがある	396
オブジェクトのメンバーをドットで参照	397
カスタムオブジェクトでJavaScriptを拡張する	401
カスタムオブジェクトを拡張する	402
コンストラクタの中はどうなっている？	403
ブログオブジェクトを実際に作成する	404
ソートする必要がある	409
日付のためのJavaScriptオブジェクト	410
時間の計算	411
ブログの日付を再考する	412
オブジェクトの中にあるオブジェクト	413
オブジェクトをテキストに変換する	416
日付の断片にアクセスする	417
オブジェクトとしての配列	420
配列のソートを調整する	421
関数リテラルを使ってソートを簡潔にする	422
ブログ配列を検索する	425
文字列内を検索するにはindexOf()	427
ブログ配列を検索する	428
検索も動くようになりました！	431
Mathオブジェクトは編成されたオブジェクト	434
Math.randomを使って乱数を生成する	436
関数をメソッドに変える	441
新しいブログオブジェクトのお披露目	442
オブジェクトがYouCubeにもたらしたもの	443

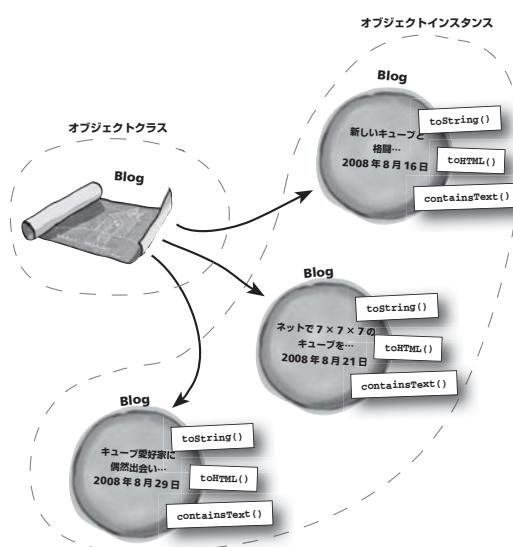
10

章 カスタムオブジェクトを作成する

カスタムオブジェクトを思い通りに

JavaScriptには返金保証はありませんが、好きなように利用できます。JavaScriptのカスタムオブジェクトは、スタバでコーヒーを注文するようなものです。キャラメルマキアートのホット、サイズはグランデ、バニラシロップとフォームミルク…。はい、カスタムコーヒーの出来上がりです。JavaScriptのカスタムオブジェクトを使えば、プロパティとメソッドの利点を生かしながら、あなたの思い通りの処理を行うコードを作ることができます。再利用可能なオブジェクト指向のコードによって、拡張されたJavaScript言語があなたのものになります。

YouCube の Blog メソッドを再考する	450
メソッドのオーバーロード	451
クラス × インスタンス	452
インスタンスはクラスから作成される	453
thisを使ってインスタンスのプロパティにアクセスする	454
クラスが持つメソッドは、ひとつのメソッドを共有して実行できます	455
prototypeを使ってクラスレベルで動かす	456
クラスと prototype と YouCube	457
クラスプロパティも共有される	462
prototypeを使ってクラスプロパティを作成する	463
署名が完成しました	465
日付を整形するメソッド	468
標準オブジェクトを拡張する	469
カスタム日付オブジェクトで YouCube を改良する	470
クラスに独自のメソッドをもたせる	471
ソート用比較関数を調べる	473
クラスメソッドを呼び出す	474
一枚の写真は千語に匹敵する	475
YouCube に画像を組み込む	476
YouCube に画像を追加する	478
オブジェクトで強化された YouCube	480

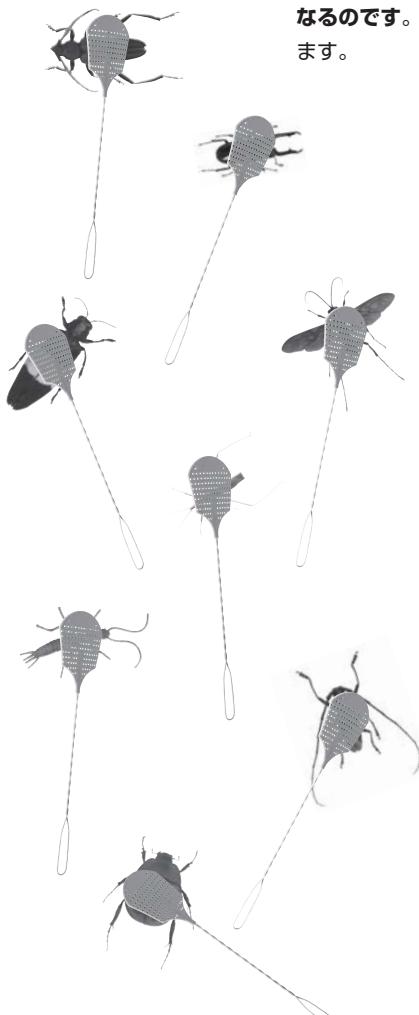


11

章 バグをなくせ

良いスクリプトも悪くなる

どんなに慎重に考えられたJavaScriptでもエラーになることはあります。そんなときは、憤てないことです。JavaScriptの優秀な開発者だったら、ぜったいバグのないスクリプトを書ける、なんて嘘です。バグの原因をつきとめて、それをなくすことができるのが優秀な開発者なのです。JavaScriptのバグを最高の腕前で駆除できる人は、目に見えにくく意地の悪いバグができるだけ少なくするために、優れたやり方で開発を進めます。ちょっとした予防策をとることで、長持ちするコードになります。とはいって、バグがみつかったら、それと戦うための武器が必要になります。

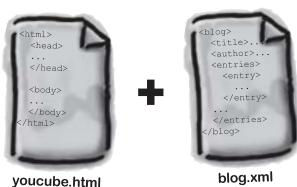


現実のデバッグ	486
バグだらけのIQ計算のケース	487
別のプラウザを試してみる	488
デバッグを楽にする	491
変数が未定義になる	495
IQの計算が成功しました	497
ラジオ番組の着呼のバグ	498
調査の開始	499
構文が妥当か調べる（バグその1）	500
文字列に注意する	501
二重引用符と単一引用符を一貫させる	502
引用符を引用符として使わないときはエスケープ文字を 追加する	503
undefinedは変数のためだけにあるのではない（バグその2）	504
全員が勝利してしまう（バグその3）	506
アラートボックスを使ってデバッグ	507
アラートを使って変数の値をウォッチする	508
論理エラーは合法ですがバグになります	510
今度は誰も当選者になれない！（バグその4）	514
アラートの連発に打ちのめされる	515
デバッグのためにカスタムコンソールを作る	517
実行時のしつこいエラー	524
JavaScriptの3大バグ	525
コメントを使って一時的にコードを無効にする	528
影の変数は危険	530

12 章 ダイナミックなデータ

感度良好なウェブアプリケーション

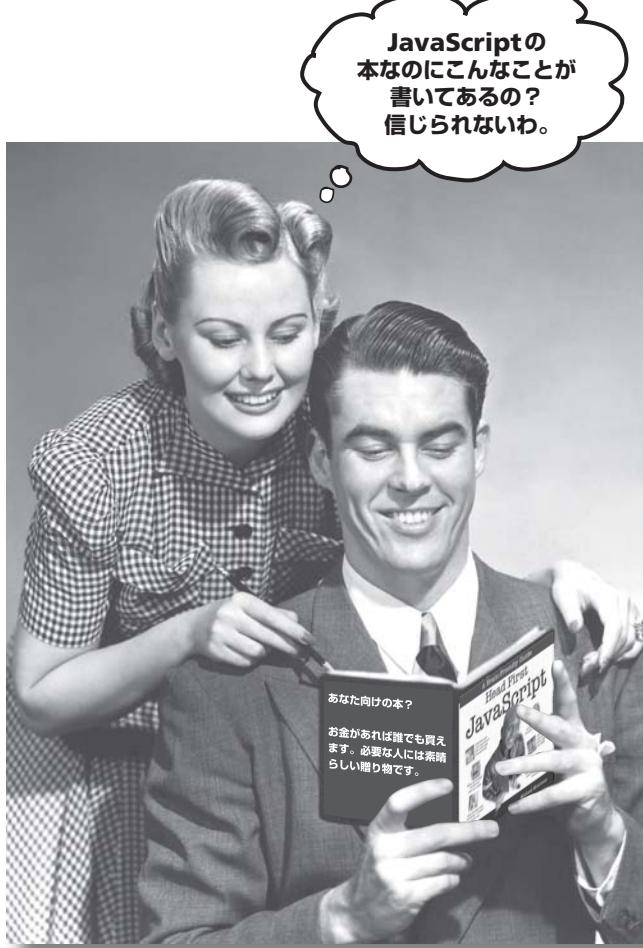
現代のウェブではユーザのあらゆる思いつきに反応できるページが求められています。それは多くのウェブユーザと開発者の夢と言えるでしょう。JavaScriptはこの夢を実現する重要な役割を担っています。Ajaxと呼ばれるプログラミング手法は、ウェブページの「感触」を劇的に変える機能を実現します。Ajaxを使うと、ウェブページの更新やブラウザの裏技を使わずに、**リアルタイムにユーザの操作に応答しながら、データの読み込みと保存が動的にすばやく実行できる**、本格的なアプリケーションになります。



動的なデータを実現したい	538
データ駆動の YouCube	539
Ajax とは要するに通信のこと	541
XML を使うとあなたのデータをあなたのやり方で扱えます	543
XML + HTML = XHTML	545
XML を YouCube プログデータに適用する	547
YouCube に Ajax を導入する	550
JavaScript と Ajax の架け橋 : XMLHttpRequest	552
GET か POST か ? XMLHttpRequest のリクエスト	555
Ajax リクエストの意味を理解する	559
リクエストオブジェクトを使ってページを	
インタラクティブにする	563
仕事が終わったら呼び出して	564
レスポンスを継ぎ目なく処理する	565
DOM からデータを引き出す	566
YouCube はそのデータで稼動する	571
ボタンの機能不全	573
ボタンに必要なデータ	574
ウェブでブログを追加して時間を節約する	577
ブログデータを書く	578
PHP を動かすのに必要なこと	581
PHP スクリプトにデータを送り込む	582
ブログデータをサーバにポストしてみましょう	585
YouCube をもっと使いやすくする	590
自動入力フィールドにする	591
タスクが重複していたら、関数にしてみては？	592
索引	599

この本の読み方

序章



このセクションでは「JavaScriptの本なのにどうしてこんな内容なの？」という疑問に答えます。

この本を読むのにふさわしいのは誰でしょう？

次の 3 つの質問の答がすべてイエスなら、

- ① ウェブブラウザ、テキストエディタ、インターネット接続が使えますか？
- ② ウェブをインタラクティブな体験に変える、活気があふれるページを作る方法を学んで理解して習得したいですか？
- ③ 無味乾燥な学校の講義よりディナーパーティでの会話の方が好きですか？

HTMLだけでは実現できない、もうもうのかっこいい機能をウェブページで実現する JavaScript コードの書き方を学ぶお手伝いをします。

この本はあなたのためになります。

たぶんこの本に向いていないのは誰でしょう？

次の 3 つの質問のいずれかの答がイエスなら、

- ① ウェブページを作るのはまったく初めてですか？（HTMLを熟知している必要はありませんが、HTMLとCSSを組み合わせてウェブページにする基本を理解している、そしてこれらのファイルをオンラインにアップロードする方法を知っていること。）
- ② Script Fu (Gimp のスクリプト) が黒帯 9 段の腕前で JavaScript リファレンスを探していますか？
- ③ 変わったことに挑戦するのは怖いですか？ ストライプの服に格子縞を合わせるより、歯根治療をしたいですか？ JavaScript コードを人間に喻えて説明してある技術書なんて真面目な本になるはずがないと思いますか？

HTMLを復習したいときは『Head First HTML with CSS & XHTML』が勧めです。

この本はあなた向きではありません。



〔マーケティングからのコメント：
クレジットカードをお持ちでしたら
レジに直行してください〕

あなたがどう思っているのか、わかってます

「このJavaScript本のどこが面白いの？」
 「どのページもグラフィックスばかりじゃない？」
 「こんな作りの本でほんとに学べるの？」

あなたの脳が何を考えているのか、わかっています

脳は目新しいことが大好きです。何か珍しいことがないか、あちこちを探しまわっています。そのようにできているおかげで生きていられるのです。では決まりきっと、ありきたりの、平凡なことを脳はどう扱っているでしょう？ こうしたことは、重要なことを記憶するという脳の仕事に関わってきません。脳は退屈なことをわざわざ記憶しようとしません。「これはたいして重要じゃない」というフィルタが働くからです。

脳は何が重要なのかをどうやって知るのでしょうか？ ハイキングの途中で虎が襲ってきたら、頭と体で何が起きるでしょう？

神経も感情も化学物質も活動が急激に活発になります。

このようにして脳は知るのです…

これは重要だ！ 忘れちゃいけない！

では自宅や図書館という、虎に襲われることもない安全で暖かい場所で勉強しているとしましょう。試験の準備、あるいはボスから10日くらいかけて目を通しておくように言われた難しい技術テーマを学習しているとします。

脳はあなたにお願いします。このたいして重要な内容で脳の希少なリソースをかき回さないでくれと。脳のリソースはほんとうに重要なこと、たとえば虎が襲ってくる、火事の危険がある、アナコンダを飼ってる友達に留守番を頼まれたのをどうやって断ればいいか、といったことに優先的に使われるためあります。

「脳さんよ、どうもどうも、この本は退屈で、ゆさぶられる感情なんてまったくないんだけど、なんとかこいつを頭の中に入れておいてよ」と脳に伝える簡単な方法がないのです。

脳はこれを重要だと
考えます。



無味乾燥で
退屈な内容が600
ページもあるぜ。

脳はこれを記憶する
までもないことだと
考えます。



「Head First」の読者は学ぶひとです。

何かを学習するとき、それを憶えて、忘れないようにする必要があります。知識を頭に詰め込むのではありません。認知科学、神経生物学、教育心理学の最近の研究によると、学習するにはページの文章だけでは十分でないことがわかつています。必要なのは脳のスイッチを入れることです。

Head Firstシリーズには学習についてのいくつかの方針があります

ビジュアルを重視。画像は文字だけの場合よりもはるかに記憶されやすいので、学習効果を高めるのに役に立ちます（最大 89 パーセント）。また内容がより理解しやすくなります。グラフィックスに関連する説明をグラフィックスの中や近くに配置しています。グラフィックスに関連する説明をページの下や別ページに配置すると、内容に関連する問題を 2 回も考えられる必要があるかもしれません。

会話みたいで親密な感じのスタイル。最近の研究によるところ、読者に語りかけるようなスタイルの方が無愛想なスタイルよりも最大 40% も学生の理解度が向上するそうです。講義ではなく物語を語るスタイルです。くだけた言葉を使って、あまり深刻にならないようにしています。ディナーパーティーで人と会話する方が講義よりも、話に身が入りますよね？



フォーカス！



アンフォーカス！

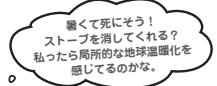
考えを深めながら学べるように。ニューロンが活性化しないから頭の中は何も変わりません。読者に必要なのは、目的をはっきりもって、熱心に楽しみながら問題の解決に集中し、結論をひねり出します。そのためには、挑戦し、練習し、新しい知識を獲得することです。そのためには、挑戦し、練習し、問題を考え、脳と感覚の両方を駆使することが必要です。

読者の関心を引きつけ、飽きさせない。「真剣に学びたいんだけど、1 ページも読み終わらないうちに眠くなってしまう」という体験は誰にでもあります。脳は、ありきたりでないもの、面白く、変わっていて、目を引く、予想に反したものに関心を集めます。新しくて難しい技術的なテーマを学ぶことは、かららずしも退屈ではありません。内容が退屈でなければ、脳はもっとスムーズに学習するはずです。

感情に訴える。何かを記憶する能力はその内容の感情的な側面にかなり依存します。「フランダースの犬」のような物語について話しているではありません。ここで

強調している感情とは、驚いたり、もっと知りたいと思ったり、楽しめたたり、これは何？と思ったり、パズルが解けたときの「やった！」かたり、誰もが難しいと思っている何かを学んだり、エンジニアのと感じたり、誰もが難しいと思っている何かをわかつたりしたときの感情です。

ボブより技術に詳しいことがわかつたりしたときの感情です。



メタ認知：思考についての思考

学びたい、それも能率よく深く学びたいのであれば、関心の持ち方に関心を持つといいでしよう。考え方について考える、学び方について学ぶ、ということです。

ほんとの人は大人になるまでにメタ認知や学習理論の授業を受けたことはありません。私たちは学ぶことを期待されていますが、学ぶ事について教えられることはまずありません。

この本を手にされた方は、インターラクティブなウェブページを作成する方法を学びたいけれども、あまり時間をかけたくないのでしょうか。この本で読んだことを使いたいと思ったら、読んだ内容を記憶する必要があります。そのためには、理解できなければなりません。この本に限らず本や学習経験を最大限に活用するのは、あなたの脳の責任です。

あなたが学んでいる新しい物事をほんとに重要なものと脳に思わせるのが秘訣です。虎に襲われるのと同じくらい重要だと。そうしないと、新しいコンテンツを脳に留めるために脳とずっと格闘することになってしまいます。

ではJavaScriptを飢えた虎と同じように脳に扱わせるにはどうしたらいいでしょう？

そのためには、スローで着実なやり方と、速くてより効率的なやり方があります。スローなやり方は、ひたすら繰り返すことです。脳に同じことを叩き込み続ければ、砂をかむような退屈なことでも学習できるのはご存知ですよね。十分に繰り返すことで、脳は「これが彼にとって重要とは思えないけど、彼は何度も何度も同じことを繰り返しているから、きっと重要なことだと思うことにしよう」と考えるわけです。

速いやり方は脳の活動を活発にさせる、とくに脳にいろいろな種類の活動をさせることです。前のページではやり方の大枠を紹介しましたが、どれも脳をあなたの望み通りに動かせるのに役立つことが証明されています。たとえば、図版の中に内容を説明する言葉を書き入れると、キャプションや本文のようにページの他の場所にあるときと比べて、言葉と絵の関係をより意味のあるものとして脳が捉えようとして、ニューロンが活性化するという研究報告もあります。ニューロンが活性化するほど、関心を払うにふさわしいもの、記憶するにふさわしいものと脳が捉える可能性が高くなります。

会話みたいな文体にすると人は会話に参加しているように思うので、話の結末を追うことを期待し、より多くの関心を払うものです。面白いのは、会話が本との間でなされていることを脳はそれほど気にしていないことです。逆に、文体がかしこまつて無味乾燥だと、脳は大勢の受け身な聞き手と一緒に座って講義を受けているのと同じように感じます。目をさます必要がないからです。

図版と会話的なスタイルは、しかし、始まりにすぎません…



この本で実現したこと

図版を使いました。テキストではなく視覚要素に脳は反応するからです。脳に関するかぎり、図版は千語に値します。テキストと図版を組み合わせるときは、図版の中にテキストを埋め込みました。テキストが関係する図版の中にテキストがある方が、キャプションや本文にあるより、脳が効率的に働くからです。

冗長な言い回し。同じ内容をさまざまなスタイルと素材で表現し、複数の意味をもたせることで、脳の中の複数の領野に内容が刻まれる可能性を高めました。

概念と図版を思いもよらないやり方で使いました。脳が新しいものに反応するからです。また脳が感情の生化学に関心を払うことを考慮して、概念と図に感情的な内容を盛り込んで、記憶に残るようにしています。ちょっとしたユーモア、驚き、興味であっても感情的な内容は効果的です。

話しかけるような会話のスタイル。一方的な説明を受け身で聞くより、会話に参加しているように思えるときの方が脳はより多くの関心をもちます。そうした脳の働きは本を読むときでも同じです。

練習問題が80問以上あります。何かを読むときより何かをしているときの方が脳の学習記憶がよくなるからです。挑戦しがいのあるレベルにしたもの、その方が好まれるからです。

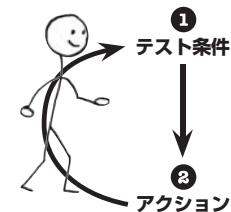
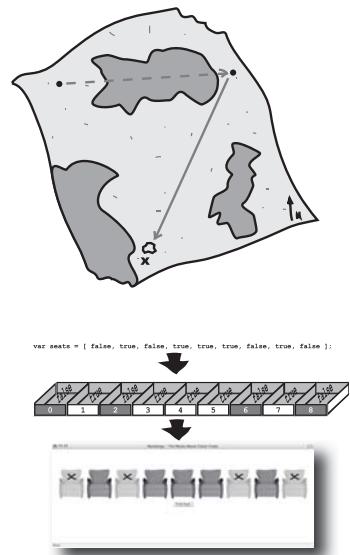
複数の学習スタイルを使いました。ひとつひとつ段階をふんだ学習を好み人もいれば、まず概要を理解したがる人もいますし、例題だけみればいいという人もいます。どんな学習方法が好みかは別にして、ある内容を複数のやり方で見せればどんな人でも満足できるでしょう。

両方の脳を考慮した内容。脳の働きが活発になるほど、学習記憶もよくなり、集中できる時間も永くなります。片側の脳を使っている間にもう片側の脳は休むことができるので、長い時間でみると学習効率がよくなります。

複数の視点をもたらすストーリーと練習問題。脳は評価や判断が強いられているときより深く学習します。

練習問題や直接的な答がない問いかけ。脳は何かやるべきことがあるとき学習記憶がよくなります。ジムで人を見ているだけではシェイプアップできません。でもあなたが頑張ればそれだけ成果があがるように、最善を尽くしています。理解しにくい例題、難解な専門用語、あまりに簡潔すぎる文章を処理することに余計な神経を使わないですむようにしています。

ストーリー、例題、写真などに人物を登場させています。脳は物に対してよりも人に対してより関心をもつからです。



頭の体操





切り取って、冷蔵庫にでも貼つておくといいでしょう。

あなたの脳を思い通りに使うためにできること

この本は最善をつくしていますので、あとはあなた次第です。以下に示すヒントを参考にして読み始めてください。脳に耳を傾け、何が役に立ち、何が役に立たないか、把握しましょう。新しいことに挑戦してみてください。

① ゆっくり読みましょう。理解すればするほど、記憶すべきことは少なくなります。

ただ読むだけでなく、ときどき読むのを止めて、考えましょう。本の中で質問があつたら、読み飛ばさずに考えてください。誰かがほんとうに質問していると想像しましょう。脳を深く考えさせれば、それだけ学習記憶が向上します。

② 練習問題に取り組みましょう。ノートを作りましょう。

練習問題は他の誰かのためではなく、あなたに解いてもらうために用意しました。練習問題を見るだけではなく、鉛筆を使いましょう。多くの研究で指摘されているように、実際に体を動かした方が学習の効果が高くなります。

③ 「素朴な疑問に答えます」を読みましょう。

どれも意味があります。補足的な読み物ではなく、本文の一部なのです。読み飛ばさないように。

④ 眠る前に読む、あるいは読み後しばらく何もしない。

学習の一部（とくに長期記憶）は本を閉じた後に行なわれます。脳をよく働かせるには、脳のテンポを大切にすることです。脳がまだ働いている間に新しいことを記憶させようとすると、憶えたての記憶の一部が失われてしまいます。

⑤ 水を飲みましょう。それもたくさんの水を。

脳は水がたっぷりのお風呂の中にあるとき最高の働きをします。脱水状態（のどの渴きを感じる前に脳はそうなります）になると、認知の働きが低下します。

⑥ 声に出しましょう、大きな声で。

声を出すことで脳の別の部分が活発になります。何かを理解しようとしているとき、あるいは何かを記憶するチャンスを増やすためには、大きな声で読むことです。さらによいのは、内容を誰かに大きな声で説明してみることです。ただ読んだだけでは理解できていなかったことが明らかになり、学習効率がよくなります。

⑦ 脳に耳を傾けましょう。

脳の負担が大きくなりすぎていないか注意を払いましょう。表面をなぞっているだけみたいな気分になったり、いま読んだばかりの内容を忘れてしまうようなときは、休憩しましょう。ある限界を越えると、それ以上何を詰め込んでも学習の効率は上がらず、かえって学習を妨げることもあります。

⑧ 実際に置き換えて考えましょう。

脳は重要なことを知る必要があります。ストーリーを楽しみましょう。写真に自分なりのキャプションを付けてみましょう。下手なジョークに文句を言うのは、何も言わないよりはるかに良いです。

⑨ 実行あるのみ。

JavaScriptを学ぶ方法はひとつだけです。**とにかくたくさんのJavaScriptコードを書くこと**です。この本を通してあなたがやることも同じです。JavaScriptの練習問題を読み飛ばさないでください。課題を解決することでたくさんのことを学習できます。棒人形の冒険、マッチョのための座席探し Mandango、YouCube ブログといったちょっと変わった例題が用意されています。練習問題に十分取り組んでから次に読み進むようにしましょう。もしインタラクティブなウェブのプロジェクトを作りたいと思っているなら、この本を読みながら作ってみましょう。恐れることはありません。JavaScriptプログラミングのノウハウを新しいピカピカのバッグに入れるのです。

読んでね

この本はリファランス本ではなく学習体験のためのチュートリアルです。そのため学習の妨げになりそうな内容についてはあえて割愛しています。この本をはじめて読むときは、1章から順番に読み進む必要があります。それまでの章で学んだことを前提にして後の章が構成されているからです。

「知る必要があること」を基本にJavaScriptを教えます。

この本はJavaScriptの歴史を調べるのには向いていないので、他の本を探しましょう。この本の目的は、JavaScriptでかっこいい実用的なことを実現する方法を教えることがあります。ウェブページをもっとインタラクティブにする、みんなが体験したくなるようないい応答をするウェブアプリケーションにするための方法です。形式的な説明ではなく、現実のプロジェクトにおいて知る必要にせまるれるJavaScriptの概念を説明しています。

JavaScript言語の隠された細部のすべてをカバーしてはいません。

JavaScriptの文、オブジェクト、イベント、キーワードをすべてこの本に詰め込みこもできましたが、フォークリフトを使わなくともあなたの机からジムに運べる手軽な本が好まれると考えました。運動しながら読むのもいいですが、汗に強い鉛筆に投資したくなるかもしれませんね。ともかく、知る必要があるJavaScriptの部品に焦点をあてたので、現場のニーズの95%は解決できるでしょう。この本を読み終わる頃には、シャワーを浴びながら夢みたすごいスクリプトを完成させるに必要なメソッドを自信をもって探せるでしょう。

JavaScriptには再利用可能なコードの組み込みライブラリが膨大にあるので、カスタムコードを自作するのではなく、標準のJavaScriptコードをいつ使うべきか理解するのは重要です。「カスタム」という言葉を目についたら、JavaScriptの組み込みコードの一部ではなく、あなたが自分で作成するコードを意味します。

この本では複数のブラウザを使うことをお勧めします。

最近のウェブブラウザはJavaScriptをサポートしていますが、特定のJavaScriptコードの処理の仕方に些細な違いがあることもあります。そのため最新のブラウザを少なくとも2種類使ってスクリプトをテストするのをお勧めします。FirefoxはJavaScriptのコーディングエラーを追うのに役立つ優秀なブラウザですが、あなたのスクリプトはさまざまな種類のブラウザで同じ動作をする必要があります。あなたのスクリプトがどんなブラウザでも動作するかテストするときは、友達、家族、仕事仲間にも協力してもらいましょう。

練習問題を省略しない。

練習問題は付録ではありません。この本の本文の一部です。記憶したり、理解したり、学んだことを応用するのに役立ちますので、練習問題を読み飛ばさないように。クロスワードパズルは興味がなければ省略してもかまいませんが、学んだ用語を別の文脈で考えるのは脳にとってよい刺激になります。折り畳みページについても、ページを折り畳んで汚くなるのが嫌なら省略してもかまいません。でも楽しみが減ることになりますね。

あえて冗長にしています。

Head Firstシリーズの特徴のひとつは、ほんとうに理解してもらうことを目的にしている点です。この本を読み終えたときには、何を学んだか憶えてもらえるようにしたいのです。ほとんどのリファレンス本は記憶され思い出されることを目的にしていますが、この本は学習が目的なので、同じ概念が何度も登場していることがわかるでしょう。

できるだけ簡潔なサンプルにしています。

たった2行のコードを理解するために200行もあるサンプルコードを読むのは読者にとて不満だらけです。この本のサンプルはできるだけ小さなないように収まるようにしていますので、明快かつ簡潔にサンプルから学ぶことができるでしょう。すべてのサンプルが堅牢で完璧だとは思わないでください。あくまで学習のために書かれたコードなので、あらゆる機能が備わっているわけではありません。ウェブにすべてのサンプルの完全なコードを公開しているので、テキストエディタでコピー&ペーストすることができます。完成されたスクリプトをオンラインで試してみることもできます。この本のウェブサイトは

<http://www.headfirstlabs.com/books/hfjs/>

「頭の体操」には答がありません

「頭の体操」は学習体験の一部ですが、そのいくつかには正解がありません。答が正しいかどうかを決めるのはあなたです。あくまで正しい方向を示すヒントであることがわかるでしょう。あなたの脳のを感じてください。

テクニカルレビューチーム

TW Scannell



Fletcher Moore



Elaine Nelson



Stephen Tallent



Alex Lee



Katherine St. John



Zachary Kessin



Anthony T. Holdener III



テクニカルレビュー担当：

Alex Leeはヒューストン大学で経営情報システムを専攻している学生です。ランニング、ビデオゲーム、新しいプログラミング言語を夜中まで習得するのを楽しんでいます。

オレゴン州シスターズ在住の**TW Scannell**は1995年以來ピットをいじっています。現在はRuby on Railsプログラマです。

Elaine Nelsonはウェブサイトのデザインを10年近く手掛けました。彼女は母親に英語の学位は何かと便利だと言っています。Elaineの現在の心象はelainenelson.orgでわかります。

Fletcher MooreはGeorgia Techのウェブ開発者/デザイナです。休日は自転車、音楽演奏、ガーデニングで過ごします。レッドソックスのファンです。アトランタで妻のKatherine、娘のSailor、息子のSatchelと暮らしています。

Anthony T. Holdener IIIはウェブアプリケーションプログラマで『Ajax: The Definitive Guide』(O'Reilly)の著者です。

Zachary Kessinは岩がまだやわらかくゴミがまだ新鮮なアイデアだった頃からウェブのプログラミングをしています。いまから15年くらい昔でしょうか。イスラエルで妻と3人の子供と暮らしています。

Katherine St. Johnはニューヨーク市立大学のコンピュータ科学と数学の助教授です。研究テーマは計算生物学とランダム構造です。

Stephen Tallentはテネシー州ナッシュビルで暮らしています。スポーツのアプリケーションを開発しながら、小さな子供の子育てに混乱しながら取り組んでいますが、それで毎日が忙殺されているわけではなく、スケートボードを楽しんだり、即席の料理人としての第二のキャリアの準備をしています。

謝辞

編集のスタッフ：

小学生のとき、外国の子供たちと日々の生活での出来事なんかを手紙でやり取りしたことはありませんか？このHead Firstのプロジェクトが始まってから、Catherine Nolanは私のペンパルになりました。電話、チャット、電子メール、FAXはもちろん、OMG (Oh My God)、LOL (Laugh Out Loud)、そして私のお気に入りのBHH (Bless Her Heart) といった略語が通じるメディアを通して、私たちはコミュニケーションを続けました。このやり取りを通して、CatherineはJavaScript認知学習オンラインコラボレーションの相手という以上の存在、私の友達になりました。いつもJavaScriptの仕事の打ち合わせがジャムセッションのバンドの話や家のリハウスの話に脱線していくわけではありませんが。この山あり谷ありのプロセスを完璧なプロフェッショナルである彼女と乗り切るのは愉しかった。ありがとう、Catherine！マティーニをおごってもらいましたね。



Lou Barr
デザインの女神。

O'Reillyチーム：

Head Firstチームの栄光について語り尽くすのは難しいですが、なんとか挑戦してみましょう。

Head Firstの研修に参加した私に教育的でサイコな狼たちを与えてくれた Brett McLaughlin は、それ以来方針を変えていません。学習プロセスのリバースエンジニアリングに対する彼の情熱はギターと同じくらい真剣なものです。彼は毎日寝る前に「私の目的は何？」と自問しているに違いないと私は信じています。彼の不滅の闘争によってこのシリーズは驚くべきものになりました。ありがとう、Brett！

もうひとりのバーチャルペンパルが Lou Barr です。合衆国と彼女の母国イギリスのちょっとした文化の違いをいろいろ教えてくれました。彼女はデザインの神々の世界から私たちのところに派遣されたのだと思います。彼女の魔法がなかったら、この本のレイアウトはまず不可能だったでしょう。

Sanders Kleinfeld はあまり目立たずに進行を管理しますが、彼の存在感はひしひしと伝わってきます。製作工程をスムーズに進め、とらえどころのないでかいアイデアを提案します。O'Reillyチームの他のメンバーへの感謝も忘れてはいけませんね。私を信頼してこのプロジェクトにゴーサインを出した Laurie Petrycki、魅力的なサポートサイト (www.headfirstlabs.com) を管理する Caitrin McCullough、軍隊並みの正確さでギャップを埋めてくれる Keith McNamara。みんな、ありがとう！

最後になりましたが、Head Firstシリーズに対する Kathy Sierra と Bert Bates の驚くべきビジョンは最大級の賞賛に値するでしょう。この本がシリーズの1冊になるのは名誉なことです…



Catherine Nolan
Phishとデューイ
十進分類のファン。

Brett McLaughlin
Head First探検隊の
リーダー。ブルースマン。



1章 インタラクティブなウェブ

応答するバーチャルワールド

まあ素敵。ウェブがこんなに
気がきくなんて思ってなかつたわ。
いま私が考えていることを
わかっているのかしら？



ウェブを受け身なページと考えるのに飽きていませんか？

受け身なページといえば本がそうです。本は読んだり学んだりするのには適していますが、**インタラクティブ**ではありません。ウェブもJavaScriptのちょっとした助けがなければインタラクティブではありません。もちろんフォームを送信できるし、HTMLとCSSを駆使してあちこちに仕掛けをつくることもできるでしょうが、無機的なウェブページを底上げしているにすぎません。ほんとうにインタラクティブになるには、**もう少し賢く動作させる必要**がありますが、**それ以上の見返り**があります。

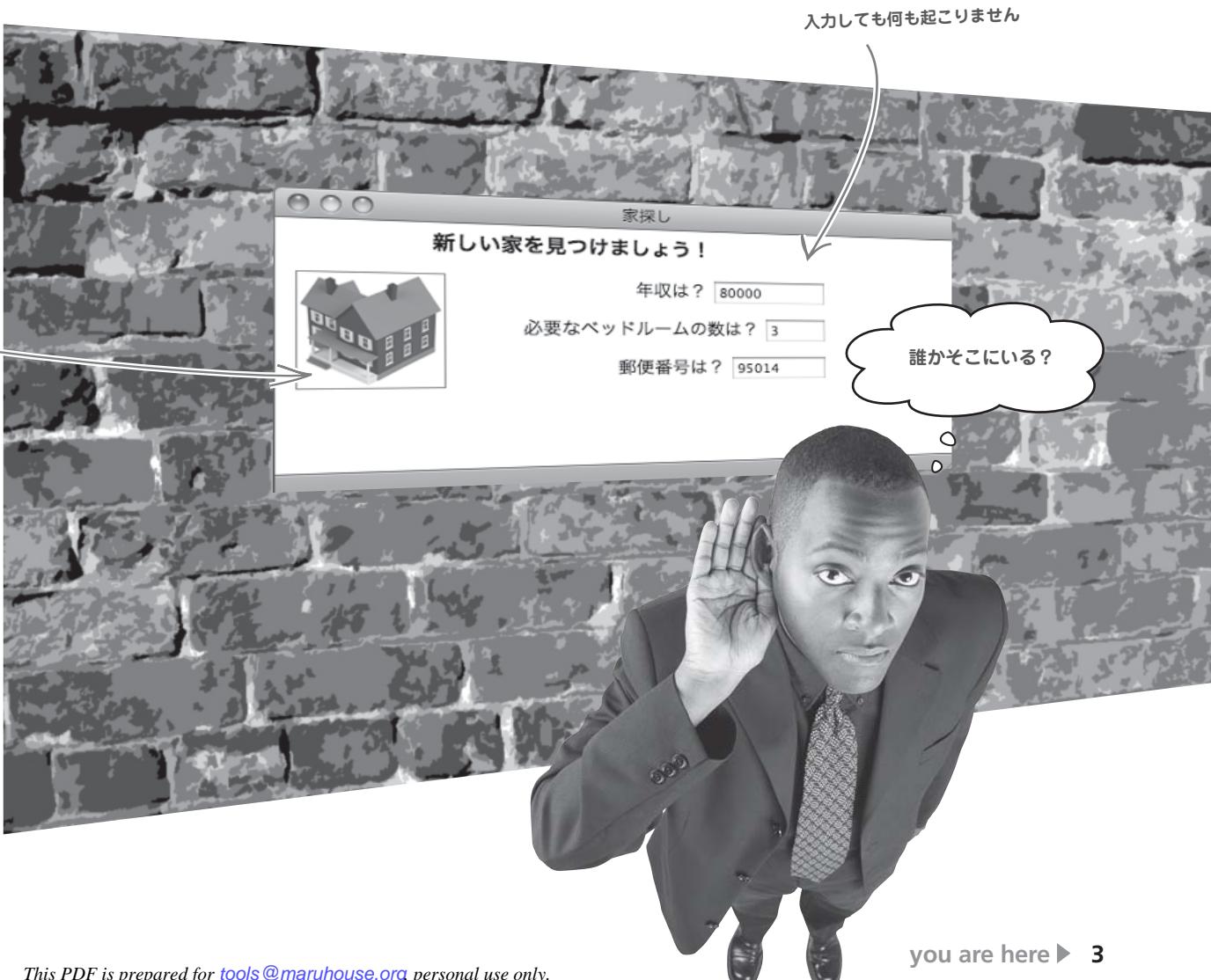
(オンラインの)ユーザが必要としていること

ご存知のとおりウェブはバーチャルですが、ウェブを使うユーザは現実の人間ですし現実世界で必要としていることがあります。とびきり美味しいミートローフのレシピを探したり、Meatloafが歌っている曲をダウンロードしたり、新しい家を購入することだってあります。幸いウェブはあなたが必要としていることを優先するようになると変わることです。



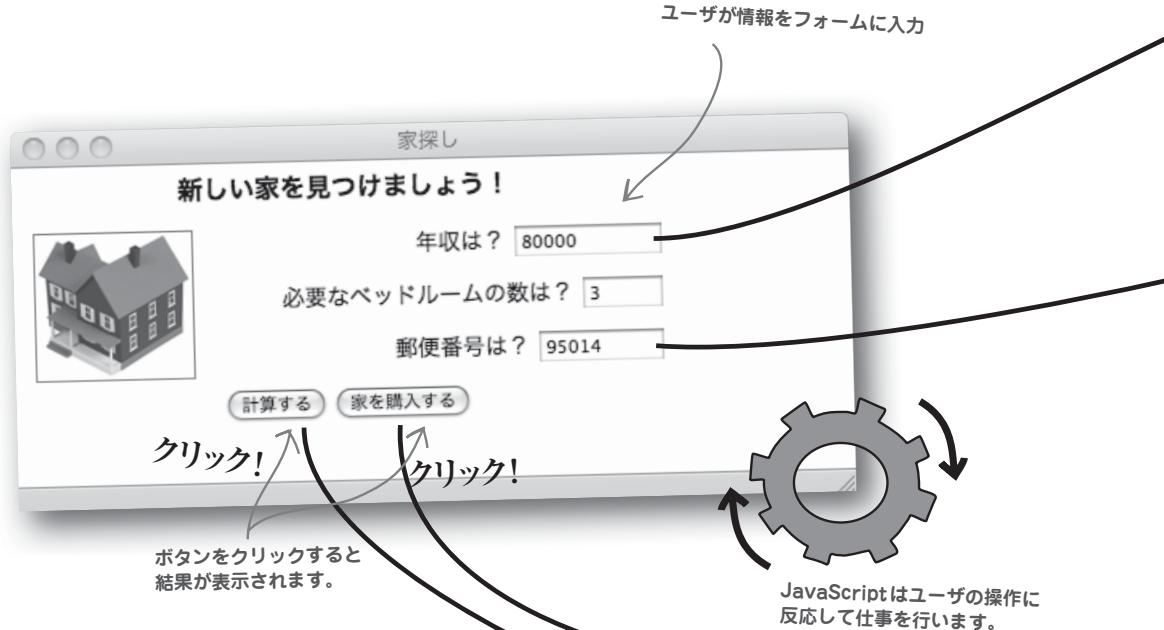
壁にむかって話しかけても、何も起こりません

ウェブはいつでもできるだけ応答してくれるとはかぎりません。まったく冷たく無表情になり、外の世界から孤立して多くのユーザが必要としていることに反応しなくなることもあります。あなたはデータを入力すれば何か反応が返ってくると期待しているかもしれません、何も起きないこともあります。それは性格の問題ではなく、静的なウェブがよいやり方を知らないだけなのです。

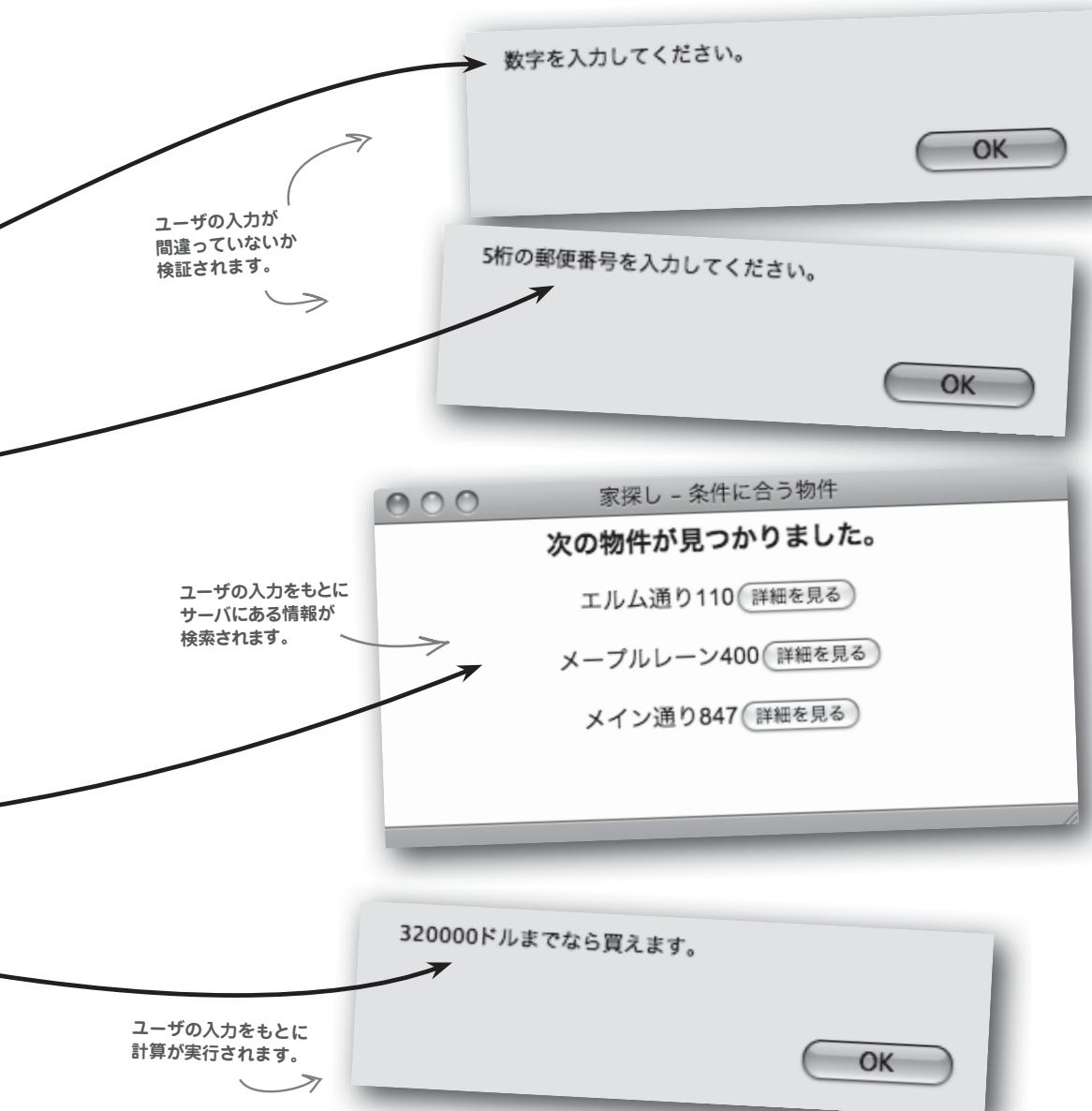


でもJavaScriptなら応答します

JavaScriptはウェブページをインタラクティブな体験に変えます。あなたが必要としていることを聞き、入力を処理し、深い要求に応えます。JavaScriptはウェブページを拡張するだけでなく、静的で活気のないページをインタラクティブなアプリケーションに変えることができるのです。

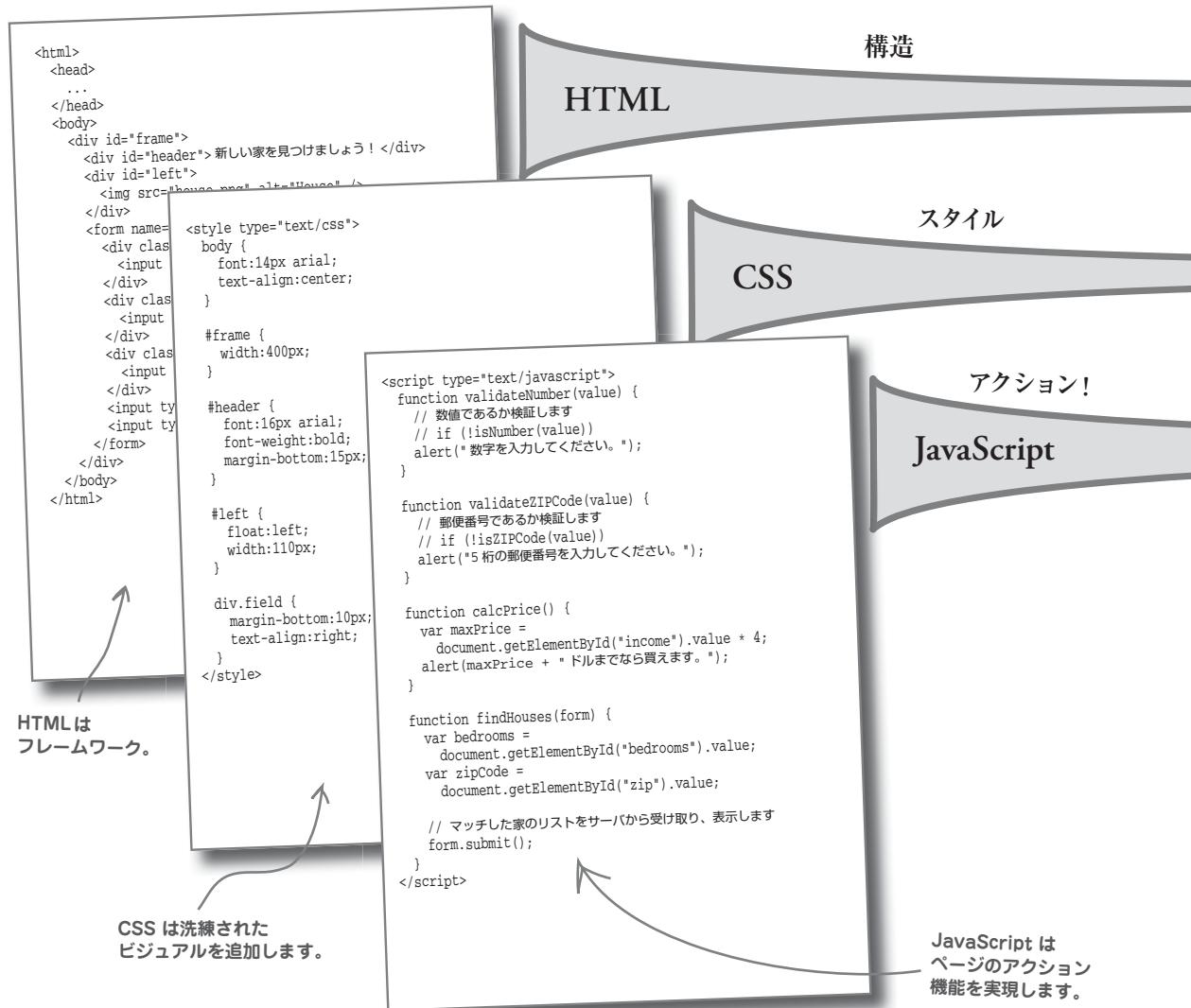


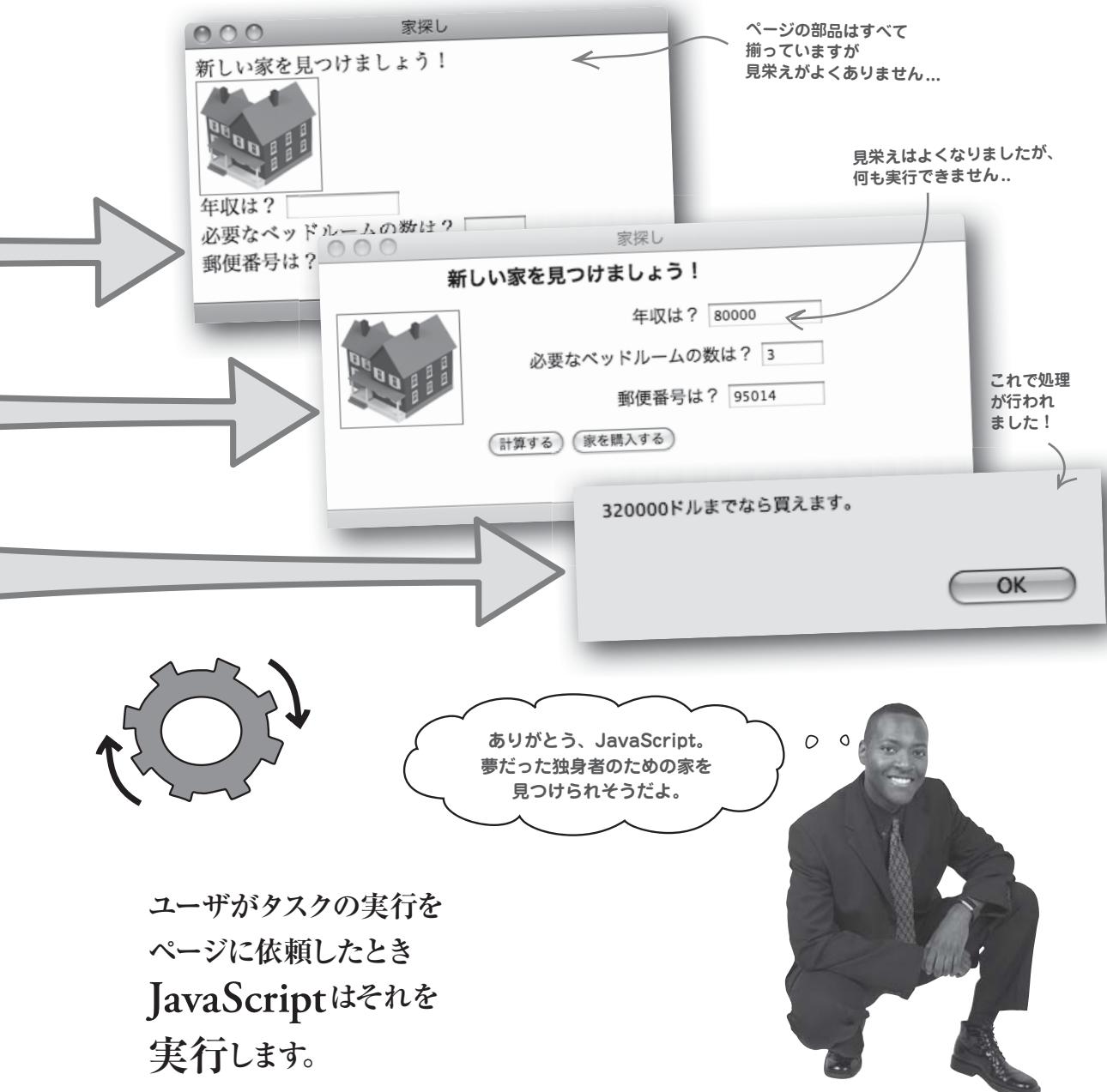
JavaScriptを使うとウェブページは
ユーザの入力に応答できるようになります。



インター ライト、カメラ、アクション

JavaScriptはHTMLとCSSと並ぶ現代のウェブページの3大構成要素のひとつです。HTMLが構造を提供し、CSSはスタイルを付加し、JavaScriptは道路にタイヤを走らせて動きをもたらします。インタラクティブなウェブページを目指すならば、構造（HTML）からスタイル（CSS）そしてその先のアクション（JavaScript）をたどる必要があります。CSSと同様JavaScriptコードもウェブページの然るべき場所にあります。







HTMLとCSSを使って
同じことが実現できないの？
JavaScriptが登場する前でも
かっこいいページが
あったわよね。

HTMLとCSSではインタラクティブになりません

問題なのは HTML と CSS だけでは、ほんとうはインタラクティブにならないことです。たしかに CSS の仕掛けを使えば、リンクの上にあるマウスの形状を変えたりできますが、そうしたスタイル操作が可能なのは非常に特殊な状況に限られます。HTML と CSS だけを使って実現できることはかなり限られているのです。

JavaScript を使うと、ボタンのクリックやブラウザウィンドウの大きさ変更、テキストフィールドへのデータ入力などウェブページで起きることを検知することができます。JavaScript はスクリプト言語なので、計算の実行、ページの画像の動的な入れ替え、データの検証といったユーザとのやり取りに応答するコードを書くことができます。



いまの時点ではまだ JavaScript の詳細
を気にすることはありません。

JavaScript はさまざまなことが実行できますが、まだ旅は始まったばかりです。イベントや関数など JavaScript のパズルの部品がこれから登場してきます。ゲームが始まるのはあなたが思っているほど遠くではありません。

HTML + CSS + JavaScript = インタラクティブ



あなたが思っている以上にあなたは理解しています。家探しウェブページのコードをみて、丸で囲まれたJavaScriptコードが何を実行しているか書いてみてください。

```

<html>
  <head>
    <title>家探し</title>
    <script type="text/javascript">
      function validateNumber(value) {
        // 数値であるか検証します
        if (!isNumber(value))
          alert("数字を入力してください。");
      }

      function validateZIPCode(value) {
        // 郵便番号であるか検証します
        if (!isZIPCode(value))
          alert("5桁の郵便番号を入力してください。");
      }

      function calcPrice() {
        var maxPrice = document.getElementById("income").value * 4;
        alert(maxPrice + "ドルまでなら買えます。");
      }

      function findHouses(form) {
        var bedrooms = document.getElementById("bedrooms").value;
        var zipCode = document.getElementById("zip").value;

        // マッチした家のリストをサーバから受け取り、表示します
        form.submit();
      }
    </script>
  </head>

  <body>
    <div id="frame">
      <div id="header">新しい家を見つめよう！</div>
      <div id="left">
        
      </div>
      <form name="orderform" action="..." method="POST">
        <div class="field">年収は？
          <input id="income" type="text" size="12"
            onblur="validateNumber(this.value)" /></div>
        <div class="field">必要なベッドルームの数は？
          <input id="bedrooms" type="text" size="6"
            onblur="validateNumber(this.value)" /></div>
        <div class="field">郵便番号は？
          <input id="zip" type="text" size="10"
            onblur="validateZIPCode(this.value)" /></div>
        <input type="button" value="計算する"
          onclick="calcPrice();"/>
        <input type="button" value="家を購入する"
          onclick="findHouses(this.form);"/>
      </form>
    </div>
  </body>
</html>
```



自分で考えてみよう の答え

あなたが思っている以上にあなたは理解しています。家探しウェブページのコードをみて、丸で囲まれたJavaScriptコードが何を実行しているか書いてみてください。

```

<html>
  <head>
    <title>家探し</title>
    <script type="text/javascript">
      function validateNumber(value) {
        // 数値であるか検証します
        if (!isNumber(value))
          alert("数字を入力してください。");
      }

      function validateZIPCode(value) {
        // 郵便番号であるか検証します
        if (!isZIPCode(value))
          alert("5桁の郵便番号を入力してください。");
      }

      function calcPrice() {
        var maxPrice = document.getElementById("income").value * 4;
        alert(maxPrice + "ドルまでなら買えます。");
      }

      function findHouses(form) {
        var bedrooms = document.getElementById("bedrooms").value;
        var zipCode = document.getElementById("zip").value;

        // マッチした家のリストをサーバから受け取り、表示します
        form.submit();
      }
    </script>
  </head>

  <body>
    <div id="frame">
      <div id="header">新しい家を見つめましょう！</div>
      <div id="left">
        
      </div>
      <form name="orderform" action="..." method="POST">
        <div class="field">年収は？
          <input id="income" type="text" size="12"
            onblur="validateNumber(this.value)"/></div>
        <div class="field">必要なベッドルームの数は？
          <input id="bedrooms" type="text" size="6"
            onblur="validateNumber(this.value)"/></div>
        <div class="field">郵便番号は？
          <input id="zip" type="text" size="10"
            onblur="validateZIPCode(this.value)"/></div>
        <input type="button" value="計算する"
          onclick="calcPrice()"/>
        <input type="button" value="家を購入する"
          onclick="findHouses(this.form)"/>
      </form>
    </div>
  </body>
</html>

```

郵便番号を 5 行の数字
で入力するように知ら
せます。

収入を 4 倍して家の
上限価格を計算します。

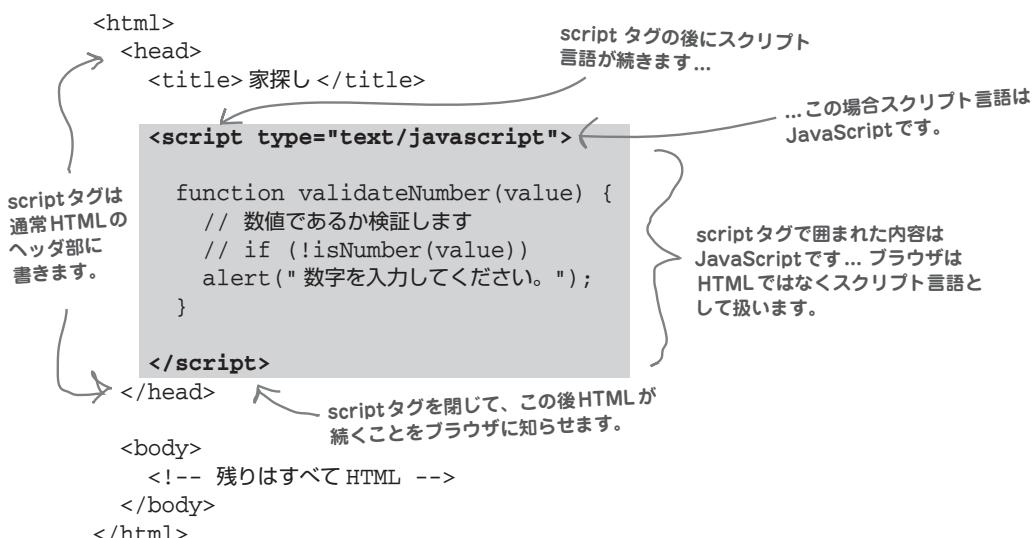
収入フィールドが数値
で入力されているか
検証します。

郵便番号フィールドの
値。

「計算する」がクリック
されたとき家の上限価格
を計算します。

<script>タグを使ってブラウザにJavaScriptが書かれていることを知らせます。

前のページで見たようにJavaScriptをHTMLウェブページに直接書こうとしています。ブラウザにコードがHTMLではなくJavaScriptであることを知らせるには、まず最初に<script>タグを書く必要があります。HTMLのどこにでも<script>タグを追加できますが、以下のようにウェブページの<head>の中に書くのが通常はベストです。



素朴な疑問に

Q: <script>タグの中のコードはJavaScriptでなければならないのでしょうか？

A: 必ずしもそうとは限りません。<script>タグはブラウザに、これからスクリプト言語になりますよと知らせるわけですが、それがJavaScriptである必要はありません。type="text/javascript"で属性を指定して、ブラウザにその言語がJavaScriptであることを知らせています。

答えます

Q: 他のスクリプト言語でも使えますか？

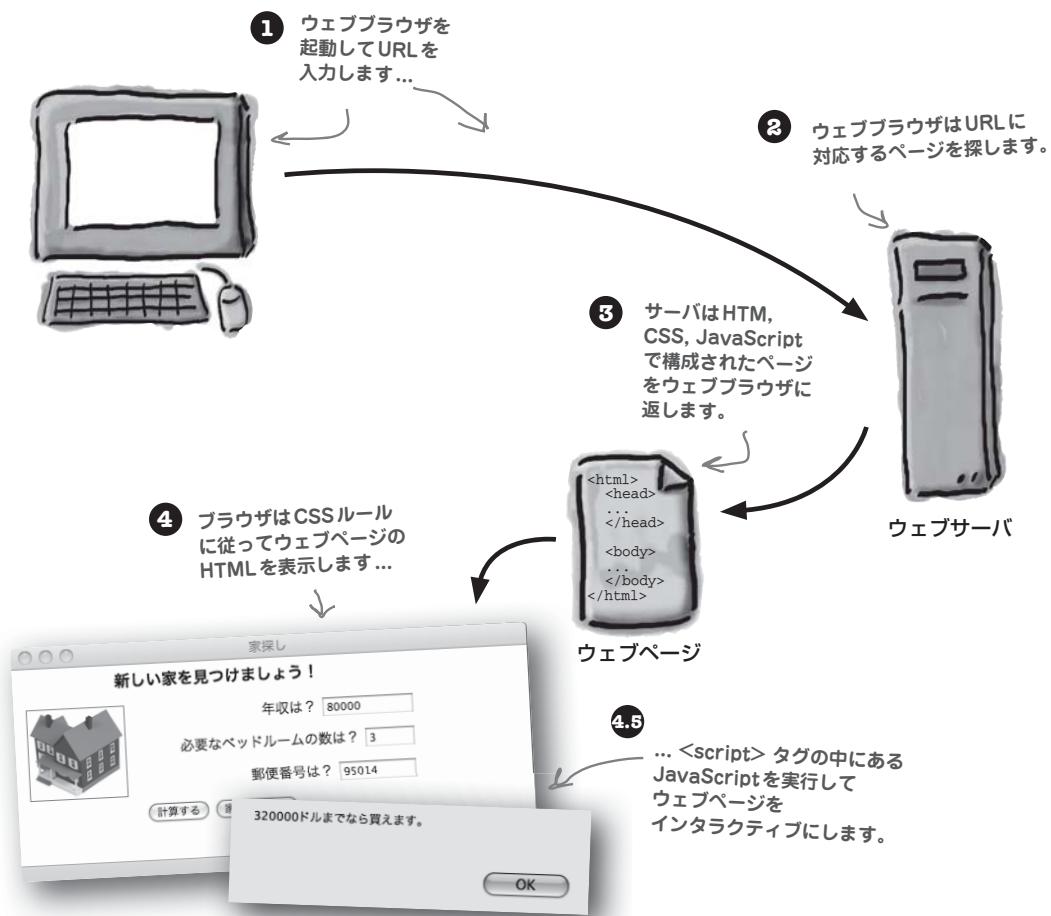
A: もちろんです。MicrosoftはVBScript (Visual Basicのスクリプト版) のAjax対応版であるASP.NET AJAXを提供しています(Ajaxについては12章で説明します)。他にもスクリプト言語はいくつかありますが、本書ではtext/javascriptだけを使います。

Q: <script>要素はHTMLページの<head>の中にある必要がありますか？

A: いいところに気がつきましたね。<script>要素はウェブページのどこにでも書けますが、ウェブページの<head>以外の場所に書くのは良くないやり方とされています。ウェブページの真ん中にCSSを書くようなものだからです。通常はJavaScriptを区別できるようにした方がいいので、ページの<head>の中に書くのがベストです。

ウェブブラウザはHTMLとCSS、そしてJavaScriptを処理できます

ウェブブラウザはHTMLを受け取って表示する方法を知っています。CSSを使ってHTMLの一部の表示を変えるようにブラウザに指示しています。JavaScriptはブラウザに対するもうひとつの指示だと考えてください。表示の仕方(HTMLやCSS)ではなく、ブラウザに実行させるコマンドを指示しているのです。



素朴な疑問に答えます

Q：ウェブブラウザはJavaScriptコードをどのようにして実行するのですか？

A：ウェブブラウザにはJavaScriptインターペリタと呼ばれるソフトウェアを内蔵しています。ページの中に現れたJavaScriptコードを実行するのがインターペリタの仕事です。JavaScriptがコンパイル言語ではなくインターペリタ言語と呼ばれるのはこのためです。C++やC#などのコンパイル言語はコンパイラと呼ばれるツールによって実行可能なプログラムファイルに変換されます。JavaScriptコードはブラウザが直接解釈するので、JavaScriptプログラムをコンパイルする必要はありません。

Q：JavaScriptコードの実行開始をウェブページに知らせるにはどうすればいいですか？

A：ほとんどのJavaScriptコードはページの中で何かが起きたときに実行されます。ページが読み込まれたとか、ユーザがボタンをクリックした、といったことです。ページで何か興味のあることが起きたとき、それをきっかけにしてJavaScriptコードの断片を実行開始できるように「イベント」として知られる機構が提供されています。

Q：ウェブのセキュリティ問題を考えるとJavaScriptは安全なんでしょうか？

A：はい、大部分は安全です。JavaScriptは悪意のあるコードが問題を起こさないようにはじめから設計されています。たとえば、JavaScriptではユーザのハードディスクにあるファイルに対して読み書きすることができます。この制約によってウイルスをはじめとする悪意のあるコードによる危険をなくしています。もちろんウェブページをめちゃくちゃにするバグのあるJavaScriptコードがありえないということではなく、JavaScriptによってユーザが深刻な危険にさらされることはない、という意味です。現実にはブラウザのバグや巧妙なハッカーによってJavaScriptのセキュリティを破る方法が過去に見つかったことはあるので、防弾装備というわけではありません。

Q：家探しコードにある<script>タグはHTMLですかJavaScriptですか？

A：<script>タグそのものはHTMLですが、その目的はHTMLコードを使ってスクリプトを混在させる手段を提供することにあります。<script>タグの中に現れるコードはJavaScriptコードです。<script>タグは複数のスクリプト言語をサポートで

きるように設計されていて、コードがJavaScriptであることをtype属性で指定します。

Q：JavaScriptを使っていないのに、たとえば入力されたデータが正しいか確認したりできるインタラクティブなウェブページを見たことがあります。こうしたことは可能なのですか？

A：はい。JavaScriptがなくてもウェブページをインタラクティブにすることは可能です。ただし多くの場合、効率が悪かったり格好悪かったりします。たとえばフォームのデータ検証をフォームが送信されたときにウェブサーバで処理することはできますが、フォーム全体をいったん送信して、サーバがデータを検証し、新しいページを結果として返すまで待っていなければなりません。その間、紙と鉛筆でフォームの検証ができるくらい待たされるかもしれません。JavaScriptは新しいページを読み込むことなくブラウザの中でインタラクティブに動作するので、サーバとの間でのデータの不必要なやり取りを行う必要がありません。それだけでなく、JavaScriptで実現可能などをJavaScriptを使わずに実現しようとすると、サードパーティのブラウザAddonを使う必要があったりもします。



エクササイズ

以下に示すコードの部品は、標準JavaScript言語の一部なのか家探しのコードのカスタム部品なのか、どちらでしょう？ 該当する方を丸で囲んでください。

alert
calcPrice
zipCode
var

JavaScript / カスタム
JavaScript / カスタム
JavaScript / カスタム
JavaScript / カスタム

onblur
onclick
findHouses
value

JavaScript / カスタム
JavaScript / カスタム
JavaScript / カスタム
JavaScript / カスタム



エクササイズの 答え

以下に示すコードの部品は、標準JavaScript言語の一部なのか家探しのコードのかスタム部品なのか、どちらでしょう？ 該当する方を丸で囲んでください。

```

<html>
  <head>
    <title> 家探し </title>
    <script type="text/javascript">
      function validateNumber(value) {
        // 数値であるか検証します
        // if (!isNumber(value))
        alert("数字を入力してください。");
      }

      function validateZIPCode(value) {
        // 郵便番号であるか検証します
        // if (!isZIPCode(value))
        alert("5桁の郵便番号を入力してください。");
      }

      function calcPrice() {
        var maxPrice = document.getElementById("income").value * 4;
        alert(maxPrice + "ドルまでなら買えます。");
      }

      function findHouses(form) {
        var bedrooms = document.getElementById("bedrooms").value;
        var zipCode = document.getElementById("zip").value;

        // マッチした家のリストをサーバから受け取り、表示します
        form.submit();
      }
    </script>
  </head>

  <body>
    <div id="frame">
      <div id="header">新しい家を見つけましょう！</div>
      <div id="left">
        
      </div>
      <form name="orderForm" action="...." method="POST">
        <div class="field">年収は？
          <input id="income" type="text" size="12"
            onblur="validateNumber(this.value)"/>
        <div class="field">必要なベッドルームの数は？
          <input id="bedrooms" type="text" size="6"
            onblur="validateNumber(this.value)"/>
        <div class="field">郵便番号は？
          <input id="zip" type="text" size="10"
            onblur="validateZIPCode(this.value)"/>
        <input type="button" value="計算する"
          onclick="calcPrice();"/>
        <input type="button" value="家を購入する"
          onclick="findHouses(this.form);"/>
      </form>
    </div>
  </body>
</html>

```

数値でないことを示す
ポップアップボックスを表示。

alert JavaScript / カスタム

家の価格を計算する
カスタムコード。

calcPrice JavaScript / カスタム

価格データのための記憶領域を設定。

var JavaScript / カスタム

条件に合致する家を
探すカスタムコード。

findHouses JavaScript / カスタム

zipCode JavaScript / カスタム

ユーザーが入力した郵便番号の
ための記憶領域。

onblur JavaScript / カスタム

ユーザーが次の入力フィールドに
移ったことを知らせます。

value JavaScript / カスタム

郵便番号入力フィールドの
現在の値。

onclick JavaScript / カスタム

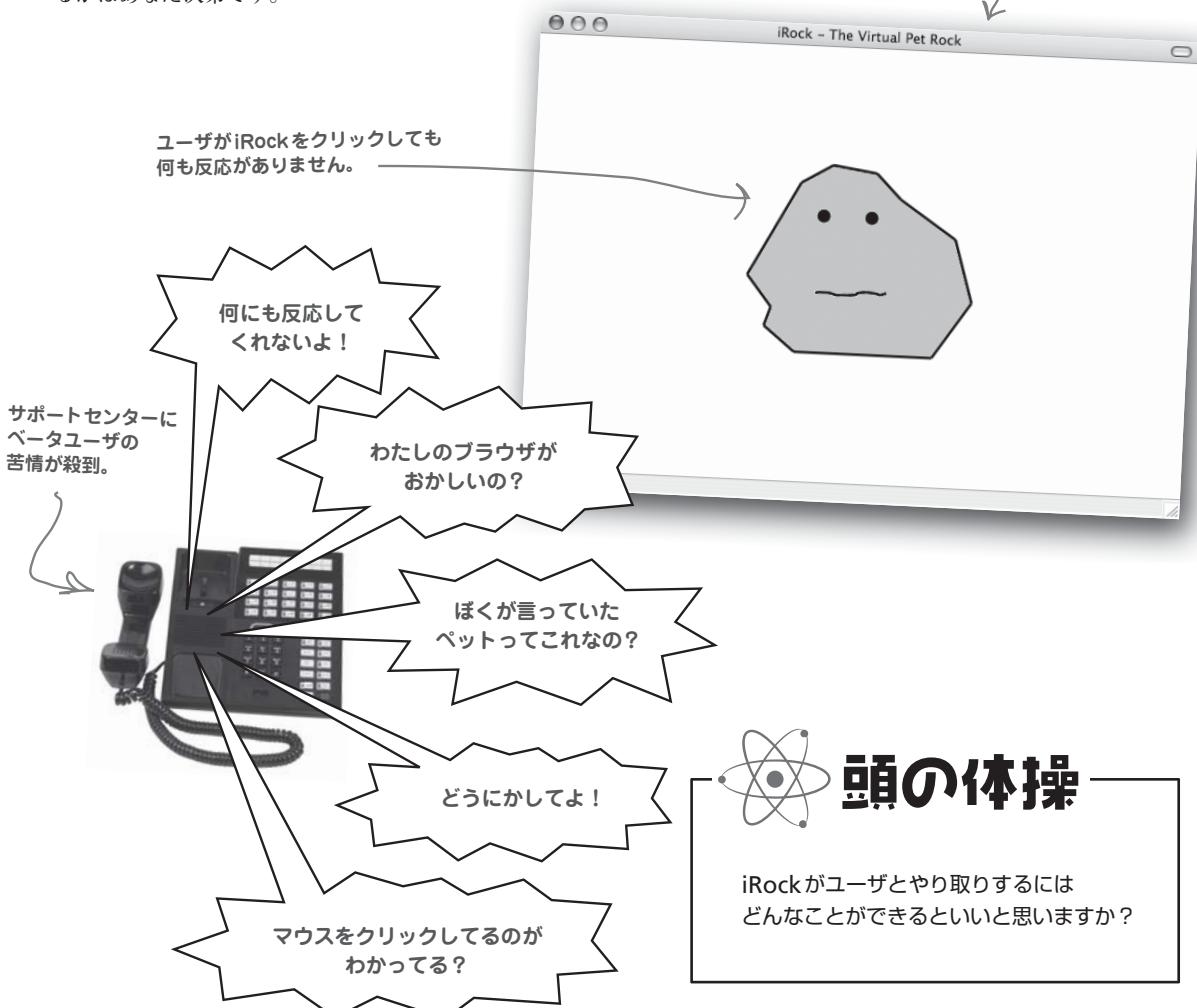
「家を購入する」ボタンがクリック
されたことを知らせます。

バーチャルなお友だちは、かまってほしいみたいです

HTMLとCSSがうまく書けたところで、あなたはボスの事務所に呼ばされました。彼が見せてくれたのは最新のオンラインバーチャルペットiRockです。玩具展示会で注目を集めましたが、オンラインペットを使ってみたベータユーザの評判は芳しくありません。

ユーザは何か面白いことが起きるのを期待してiRockをクリックしているのに、ボスは何も考えていませんでした。そこで**iRockをインタラクティブにする仕事**があなたにまわってきました。成功して栄光を勝ち取るかiRockと一緒に罵声を浴びるかはあなた次第です。

このiRockはHTMLとCSSだけなのでユーザに反応できません。



iRockをインタラクティブにする

iRockをインタラクティブする仕事を任せられたらJavaScriptを学ぶ必要があります。心配しなくて大丈夫です。すぐにiRockに「こんにちは」って挨拶させることになりますから。

この章ではこれから以下のことを学習します。

- ① iRockのHTMLウェブページを作成する。

このやり方はすでに
学習済みですね。

- ② iRockのウェブページが読み込まれたときユーザに挨拶
するようにJavaScriptのアラートを追加する。

JavaScriptではalertを
使って簡単なメッセージを
ポップアップボックスで
表示します。

- ③ ユーザの名前を尋ねて、名前で語りかける挨拶を表示し、
最後にiRockを微笑ませる、以上の処理を実行する
JavaScriptコードを書く。

iRockがクリック
されたときのアクションを
設定します…

- ④ ユーザがiRockをクリックしたときのイベントハンドラを
追加して、ステップ3で書いたコードを実行する。

…ここでアクションを
設計します。

- ⑤ 称賛され、ボスからも感謝される。

iRockウェブページを作成する

iRockくらい簡単なHTMLページもそうそうないでしょう。お好みのエディタでこのHTMLを打ち込んで、iRock.htmlというファイル名で保存してください。画像はダウンロードできます。

iRockのHTMLページは岩みたいに退屈。ボスがあなたの助けを借りたくなるのも当然ですね。

```
<html>
  <head>
    <title>iRock - The Virtual Pet Rock</title>
  </head>

  <body>
    <div style="margin-top:100px; text-align:center">
      
    </div>
  </body>
</html>
```

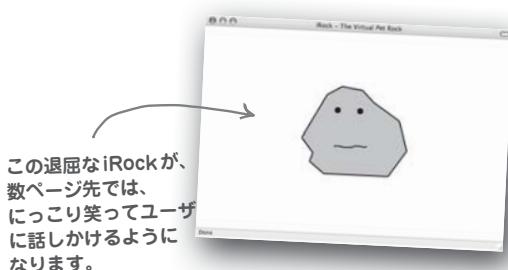


irock.html

rock.png は Head First Labs の
サイトからダウンロードできます。

テスト運転

先に進む前にiRockのウェブページをブラウザでテストしてみてください。下の画面と同じになれば問題ありません。ではこれからJavaScriptを使ってインタラクティブにしていきましょう。



この退屈なiRockが、
数ページ先では、
にっこり笑ってユーザ
に話しかけるよう
になります。

素朴な疑問に

答えます

Q: <div>タグの中にCSSがあってもいいですか？

A: 大丈夫です。いいところに気がつきましたね。

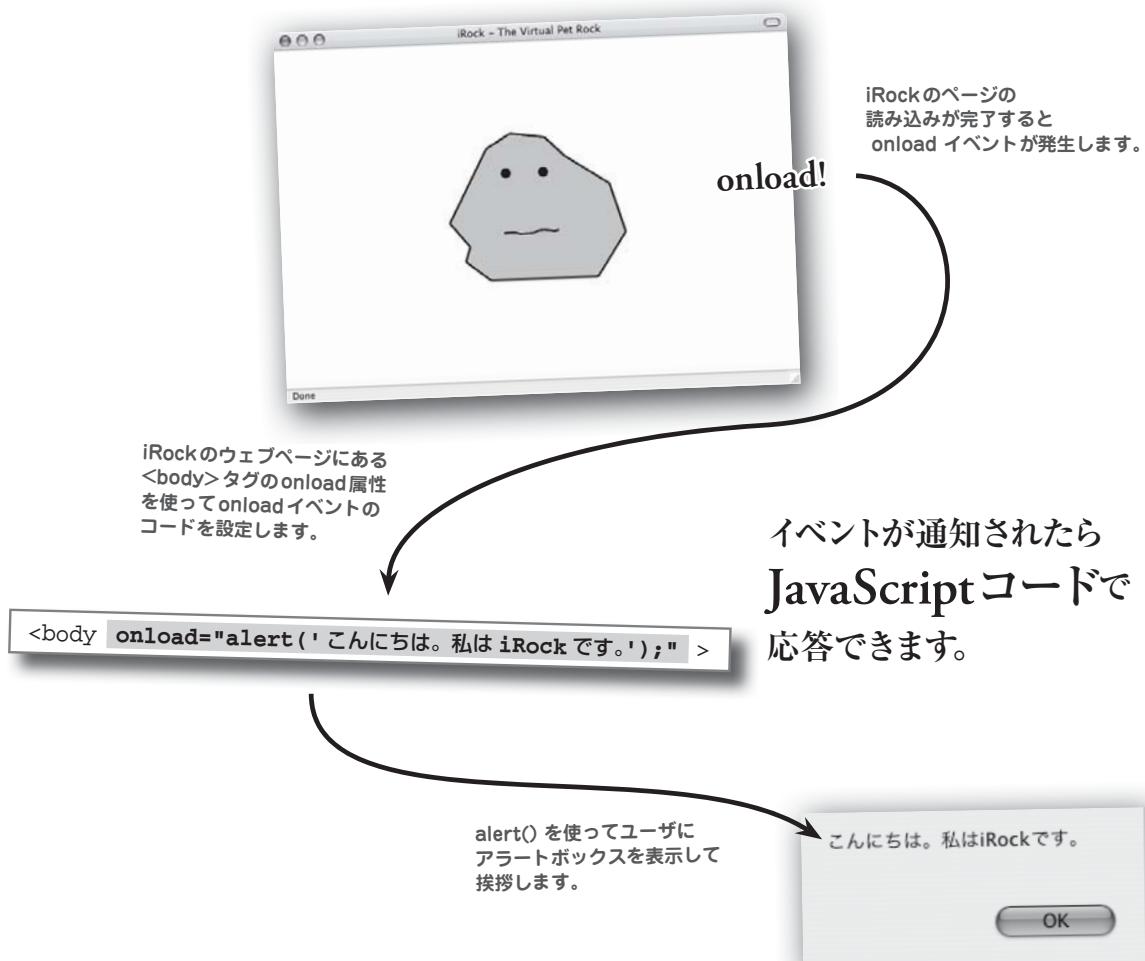
Q: HTMLページにCSSを直接書くのは、ほんとは良くないのではと思いますが、いかがでしょうか？

A: 『Head First HTML with CSS & XHTML』を読まれたのかもしれませんね。あなたのおっしゃる通りです。CSSはページの<head>の中の<style>タグの中に書くか、スタイルシートを外部ファイルにするのが普通です。あなたのボスがいいコードを書いていないとしても、コードそのものは簡潔になっています。もちろんiRockの外部スタイルシートを書いてみるのもとても格好いいと思いますよ。

JavaScriptのイベントを使ってiRockの「声」を実現する

ページが最初に読み込まれたとき JavaScript を使ってユーザに挨拶させるには、JavaScript に関する 2 つの問題を解決しておく必要があります。ひとつはページの読み込みがいつ完了したかを知る方法、もうひとつユーザに挨拶を表示する方法です。

最初の問題はイベント（ページ読み込みが完了したときのイベント）に応答することで、2 番目の問題は JavaScript の組み込み機能であるアラートボックスを使うことで解決します。イベントは JavaScript の通知の仕掛けで、これを使うとページの読み込み (onload) やボタンのクリック (onclick) といったイベントが発生したことがわかります。イベントには独自に作成した JavaScript コードで応答できます。



関数を使ってユーザにアラート表示する

JavaScriptのアラートはユーザに情報を表示するのに使えるpopupアップウインドウ(またはボックス)です。アラートボックスを表示するには、表示したいテキストをJavaScriptの関数alert()に引数で渡して呼び出します。関数は再利用可能なJavaScriptコードのかたまりで、popupアップウインドウに情報を表示するといったタスクを実行します。

`alert()` ←
名前の直後に括弧が続いて
いれば関数になります。

alert の詳細



`alert` はアラートボックスを表示する組み込み関数の名前です。

アラートボックスに表示するテキストを単一引用符で囲んで書きます。

`alert`

+ (

' 表示するテキスト '

) + ;

JavaScript の関数に渡される情報を括弧で囲みます。ここでは表示するテキストになります。

文の最後にピリオドがあるのと同じように、セミコロンは JavaScript の文の終わりの印です。

これらをひとつにまとめると、挨拶のテキストをアラートボックスに表示する関数を呼び出すJavaScriptコードの1行が完成します。

```
alert('こんにちは。私は iRock です。');
```

表示するテキストは
単一引用符か二重引用符
で囲みます。



リラックス

イベントのこと
で悩まないでね。

イベントのことはわかりにくいでしょう
が、心配しないでください。この本を読み進むにつれてイベントの意味がだんだん明らかになっていきます。

関数はタスクを実行する
再利用可能なコード断片です。

iRockに挨拶機能を追加する

iRockページが読み込まれたときユーザにご挨拶するには、`onload`イベントハンドラを追加して、JavaScriptの`alert()`を使って挨拶を表示します。`irock.html`にJavaScriptの行を追加します。

```
<html>
  <head>
    <title>iRock - The Virtual Pet Rock</title>
  </head>

  <body onload="alert('こんにちは。私はiRockです。');">
    <div style="margin-top:100px; text-align:center">
      
    </div>
  </body>
</html>
```

onload
イベントは
ページのどこで
でも使えますが、
ページの本体は
ブラウザに表示
される部分なので
<body>タグの
属性にします。

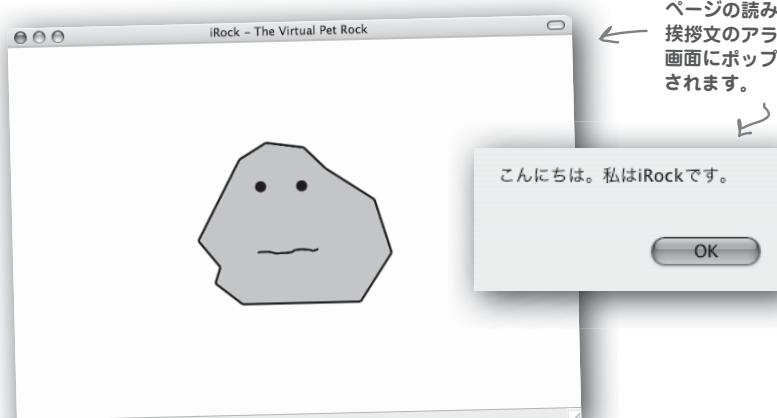
JavaScriptはウェブページの中にあります。
ウェブブラウザはHTMLやCSSと同じように
JavaScriptの扱い方をわかっています。

```
<html>
  <head>
    ...
  </head>
  <body>
    ...
  </body>
</html>
```

irock.html

インタラクティブなiRockをテスト運転する

iRockのページは`onload`イベントに応答してアラートボックスに挨拶が表示されるようになり、以前よりインタラクティブになりました。`irock.html`をウェブブラウザで表示して、動作を確認してみてください。



素朴な疑問に答えます

Q: イベントはどこから来るのでしょうか?

A: イベントはユーザの操作によって発生しますが、最終的にはブラウザから来ることになります。たとえばユーザがキーを操作して「key press」イベントが発生すると、ブラウザはそのイベントに関する情報(どのキーが押されたのかなどをひとつにまとめて、このイベントに応答するように指定された関数)を渡します。

Q: コードに結びつけられていないイベントが発生したらどうなりますか?

A: 木が倒れても、そのまわりに誰もいなければ、音がしたと言えるでしょうか? イベントに関しても同様です。もしイベントに応答しなければ、ブラウザは自分の仕事を続けるだけで、ユーザにはわかりません。言い換えると、 onload に応答するにせよ応答しないにせよ、実際に読み込まれるページには影響しません。

Q: JavaScriptコードは<script>タグの中に属すると説明されましたよね?

A: 通常はそうです。しかし、 onload イベントで行ったように、イベントハンドラに直接書くこともできます。iRock のようにJavaScriptを1行実行させるだけであれば、こうした簡単なアプローチもよく使われます。



重要ポイント

Q: alert()のような組み込み関数は他にもありますか?

A: はい、たくさんありますね。alert()はJavaScriptの再利用可能な組み込み関数のほんの一角です。JavaScriptの機能をめぐる旅の中で、たくさんの標準関数を取り上げていきます。本書を読み終える頃には、自分でカスタム関数を作れるようになっているでしょう。

Q: iRockの onload コードでは、なぜ二重引用符と単一引用符が混在しているのですか?

A: HTMLとJavaScriptでは、別のテキストが始まる前でテキストのシーケンスを閉じる必要があります。ただし別の区切り文字(二重引用符か単一引用符)を使っていれば別ですが。HTML属性にJavaScriptコードが現れる(テキストの中にテキストがある)場合、二重引用符と単一引用符を混在させることで、この問題を解決できます。属性あるいはJavaScriptテキストのどちらにどの引用符を使ってもかまいませんが、一貫性をもたせる必要があります。二重引用符と単一引用符が混在した普通の言語の例をみればはつきりするでしょう。iRockにちなんで“ユーザがクリックすると 'Hello there' と応える”。

- イベントはウェブページで起きたことをJavaScriptコードで応答するときに使います。
- onload イベントはページの読み込みが完了したときに発生します。
- <body>タグの onload 属性を設定すれば onload イベントに応答できます。
- 関数を使うとJavaScriptコードを再利用可能なモジュールにすることができます。
- いくつかの関数にはそのタスクを実行するために情報を渡す必要があります。
- alert()は小さなポップアップウィンドウにテキストメッセージを表示するJavaScriptの組み込み関数です。

誰が何をする?

JavaScriptのそれぞれの断片をその実行内容と対応づけてください。

onload

ポップアップウィンドウにメッセージを表示する

(

JavaScriptのコードの行の終端

alert

ウェブページの読み込み完了を示す

;

関数に渡される情報を囲む

誰が何をする？の答え

JavaScriptのそれぞれの断片をその実行内容と対応づけてください。



iRockを本格的に インタラクティブにしてみましょう

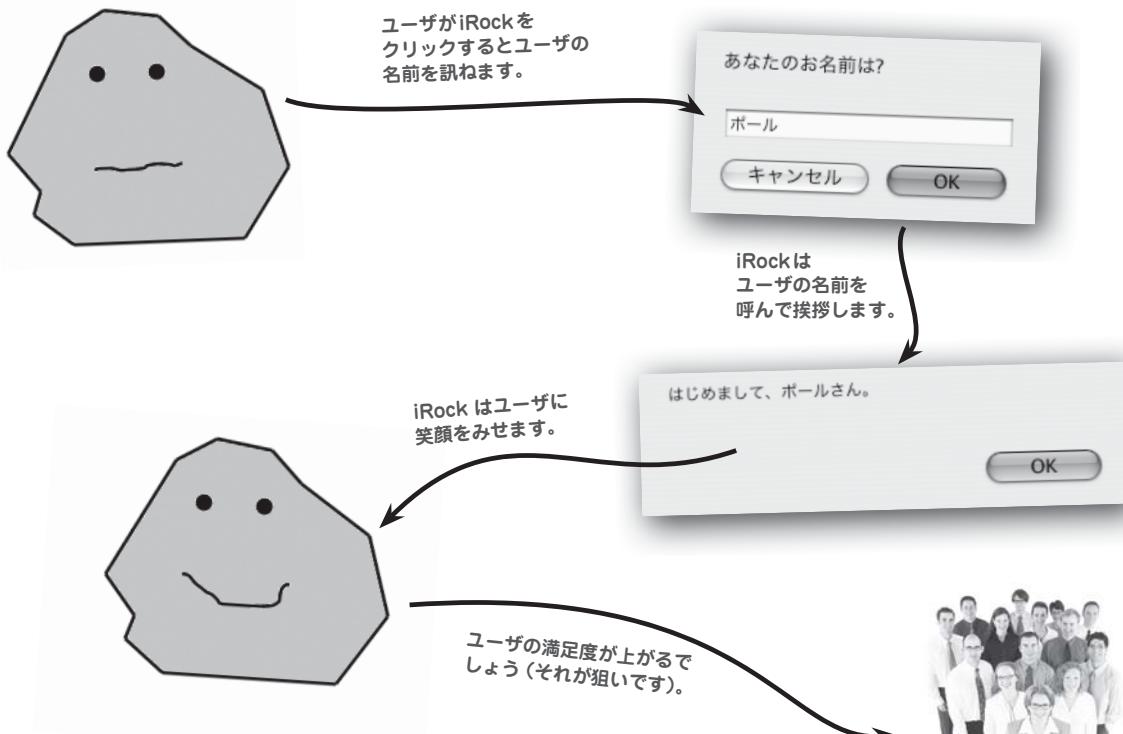
iRockをインタラクティブにする計画は着実に前進していますが、
バーチャルペットとして成功するにはまだいくつかの課題が残って
います。チェックリストを憶えていますか？

これは済み！

- ① iRockのHTMLウェブページを作成する。 ←
- ② iRockのウェブページが読み込まれたときユーザに挨拶する ← これも完了。
ようにJavaScriptのアラートを追加する。
- ③ ユーザの名前を尋ねて、名前で語りかける挨拶を表示し、
最後にiRockを微笑ませる、以上の処理を実行する
JavaScriptコードを書く。
- ④ ユーザがiRockをクリックしたときのイベントハンドラを
追加して、ステップ3で書いたコードを実行する。
- ⑤ 称賛され、ボスからも感謝される。

インタラクティブとは双方のコミュニケーション

なんとかご挨拶ができるようになったiRockですが、ユーザはとくにすることはありません。iRockがユーザに応答できるようにしたいのです。でもJavaScriptの力をすこし借りれば、顔の表情を変化させたり、挨拶で相手の名前を読んだりできるようになるので、iRockを驚くほど社交的で親しみやすいペットに変えることができます。



JavaScriptを使うとユーザはキーを押したりマウスをクリックする操作でiRockとやり取りできるようになります。ペットと飼い主の関係みたいになります。



マウスクリックに応答するときに使われるJavaScriptのイベントの名前を当ててみてください。



自分で考えてみよう の答え

マウスクリックに応答するときに使われるJavaScriptのイベントの名前を当ててみてください。

onclick

ページの要素をユーザがマウスでクリックしたとき onclick イベントが発生します。ウェブページの要素はどれも onclick を設定することができます。

ユーザの名前を受け取る関数を追加する

以下に示すJavaScriptの関数は調理済みなのですぐに使えます。「後は焼くだけJavaScript」があったら、コードをそっくりタイプするだけで使えます。このコードの詳細は追々わかってきますから、そのうち自分で関数を書けるようになるでしょう。

このtouchRock()という名前のカスタム関数のコードは、ユーザに名前を入力してもらうように促し、その名前が入った挨拶文をアラートボックスに表示します。この関数はiRockの画像を笑顔の表情に変えます。必要な作業はこの関数をiRockに追加するだけです。



後は焼くだけ
JavaScript

```
function touchRock() {  
    var userName = prompt("あなたの名前は?");  
  
    if (userName) {  
        alert("はじめまして、" + userName + "さん。");  
        document.getElementById("rockImg").src = "rock_happy.png";  
    }  
}
```

alert() と同様、JavaScriptの関数には名前があります。この関数の名前はtouchRockです。

prompt() はボックスを表示しユーザから値を受け取る関数です。

名前を受け取ったら、ユーザを名前で呼んで挨拶し…

…さらに笑顔のiRockの画像に変更します。



頭の体操

この関数をirock.htmlのどこに書いたらいいかわかりますか？



JavaScriptマグネット

ユーザフレンドリなiRockのコードからいくつかコード断片が消えてしまいました。コード断片を埋めてページを完成させてください。

ヒント：答に自信がないときは
irock.htmlに答をタイプして
テストしてみましょう。



```
<html>
<head>
    <title>iRock - The Virtual Pet Rock</title>

    < ..... type="text/javascript">
        function touchRock() {
            var userName = prompt("あなたの名前は？");

            if (userName) {
                alert("はじめまして、" + userName + "さん。");
                document.getElementById("rockImg").src = "rock_happy.png";
            }
        }
    </script>
</head>

<body ..... =".....( ..... );">
    <div style="margin-top:100px; text-align:center">
        
    </div>
</body>
</html>
```

touchRock()

alert

onload

script

onclick

'こんにちは。私は iRock です。'



JavaScript マグネットの答え

ユーザフレンドリな iRock のコードからいくつかコード断片が消えてしまいました。コード断片を埋めてページを完成させてください。

JavaScript 関数はページの
<head> にある <script>
タグの中に書きます。

<script> タグの type 属性は
スクリプト言語の種類を指定するのに
使います。この場合 JavaScript です。

```

<html>
  <head>
    <title>iRock - The Virtual Pet Rock</title>
    <script type="text/javascript">
      function touchRock() {
        var userName = prompt("あなたの名前は？");

        if (userName) {
          alert("はじめまして、" + userName + "さん。");
          document.getElementById("rockImg").src = "rock_happy.png";
        }
      }
    </script>
  </head>
<body>
  <div style="margin-top:100px; text-align:center">
    
  </div>
</body>
</html>

```

iRock の画像の onclick イベント
属性を使うと iRock がクリックされた
とき touchRock() が呼び出されます。

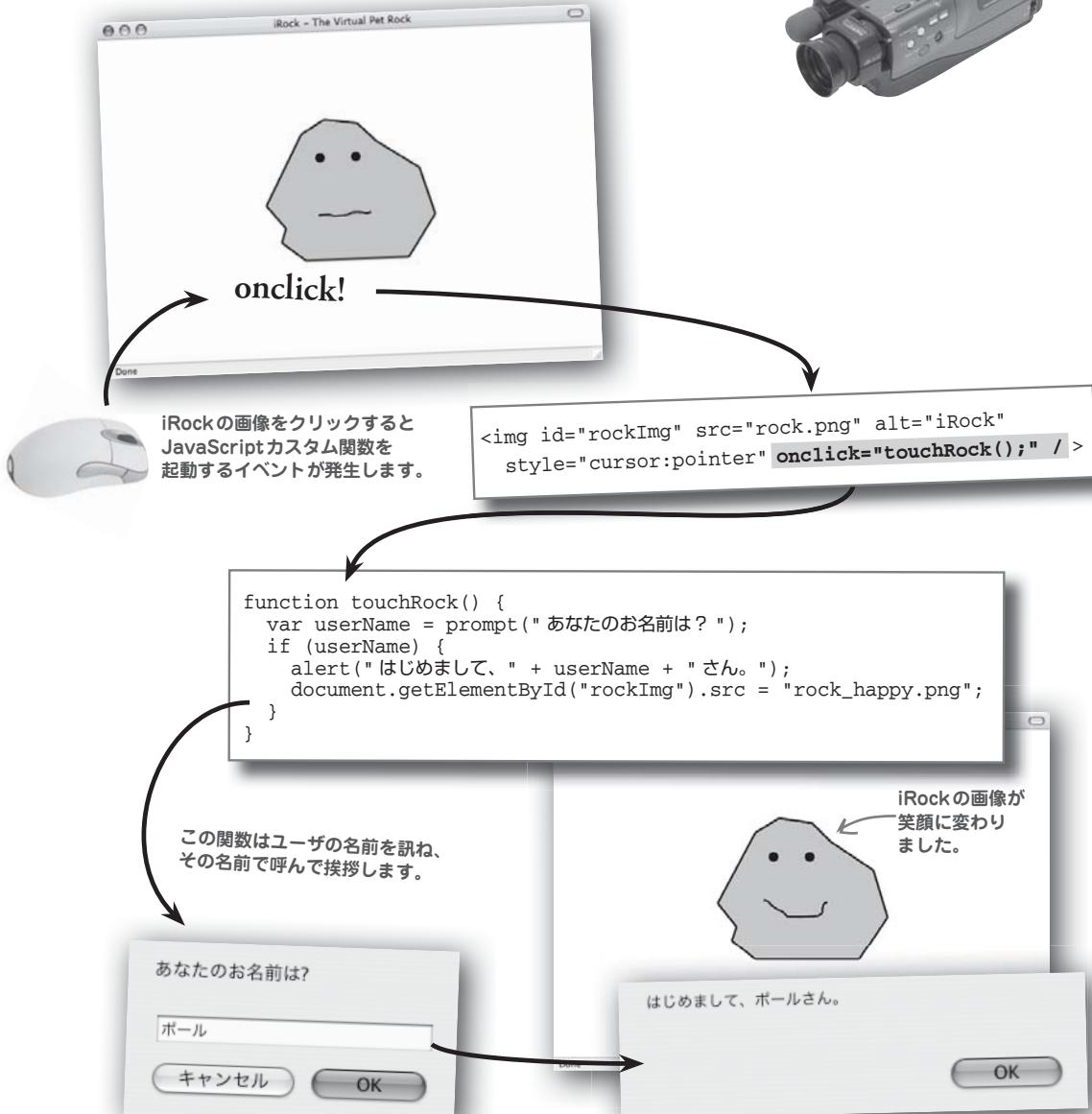
<body> タグの onload イベント属性を
挨拶用のアラートボックスにつなぎます。

iRock の画像を笑顔の
画像に変えます。

iRock の上にマウスカーソルが
あるときは手のひらの形状に
変えます。

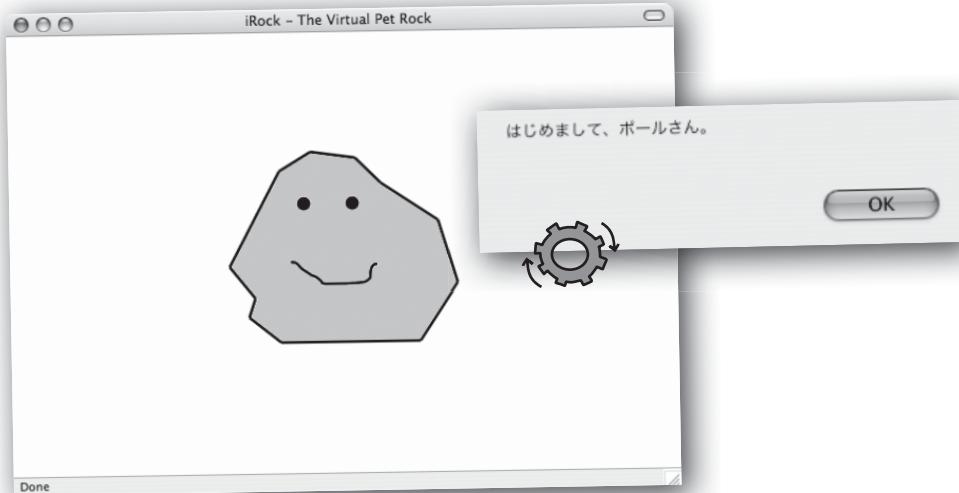
即席リプレイで動作を確認

ほんの少しのJavaScriptで大変身、いとしいiRockになりました。ページの何を変えて、その結果どうなったのか、即席でリプレイしてみましょう。



iRock 1.0 をテスト運転

あなたが作成した irock.html が 26 ページのと同じになっているか確認してください。画像がダウンロードされますよね。このウェブページを開いて、iRock を動かしてみましょう。



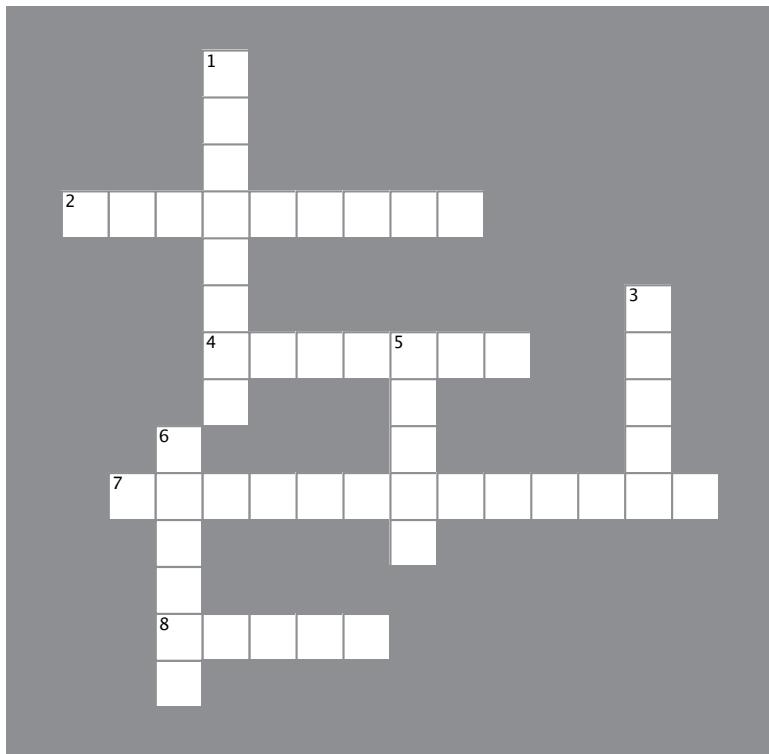
- ① iRock の HTML ウェブページを作成する。 ← これは済み！
- ② iRock のウェブページが読み込まれたとき ← これも完了。
ユーザーに挨拶するように JavaScript の
アラートを追加する。
- ③ ユーザの名前を尋ねて、名前で語りかける挨拶を
表示し、最後に iRock を微笑ませる、以上の ←
処理を実行する JavaScript コードを書く。 ←
ここで
touchRock() を
使いました。
- ④ ユーザが iRock をクリックしたときのイベント
ハンドラを追加して、ステップ 3 で書いたコード
を実行する。 ←
onclick イベント
ハンドラを
使いました。
- ⑤ 称賛され、ボスからも感謝される。 ←
ボスは大喜び…大型モニタを
買ってくれるかも？

JavaScript を使うと、
何かを見せたり
語るだけでなく、
何かを実行する
ウェブページに
なります。



JavaScriptクロスワード

ここで一休みして右脳を使ってみましょう。よくあるクロスワードパズルです。答のワードはこの章にあります。



ヨコのカギ

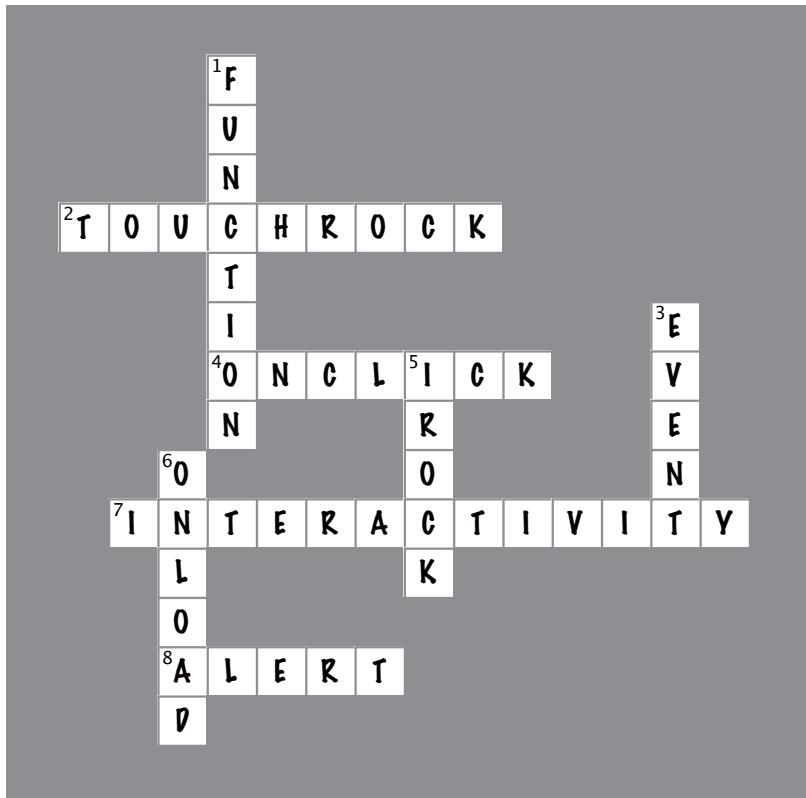
2. iRockに挨拶させる機能を提供するコードのかたまりの名前。
4. マウスクリックに応答するように、HTML要素の 属性にJavaScriptコードを設定します。
7. これがないとHTMLとCSSを使ってなんとかするしかありません。
8. ユーザにテキストを表示するときは関数 を呼び出します。

タテのカギ

1. あるタスクを実行する再利用可能なJavaScriptコード断片のこと。
3. 何かが発生したときブラウザがあなたに知らせようすること。
5. 「今シーズンのかわいいオンラインペット」といえば？
6. ウェブページの読み込みが完了したことを知らせるイベント。



JavaScript クロスワードの答え



折り畳みページ

脳のところで
折り畳んでから
謎を解決しましょう。

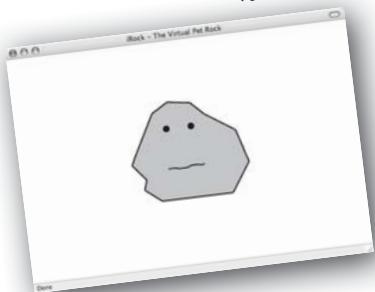
JavaScriptはウェブページに
何をもたらしますか？



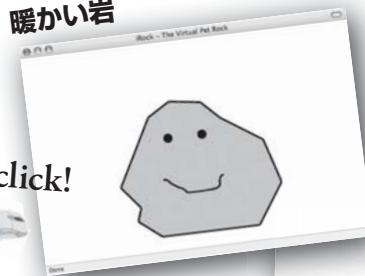
左脳と右脳がご対面！



冷たい岩…



…暖かい岩



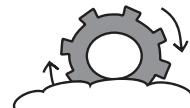
でも求めているものは誰もみな同じ！



わうう！



現実の動物と同じものを
iRockも持っています。
それは何？



あれ、どうしたの？



みやー…



その答をインターネットで探しても

あまり役に立たないでしょうね。

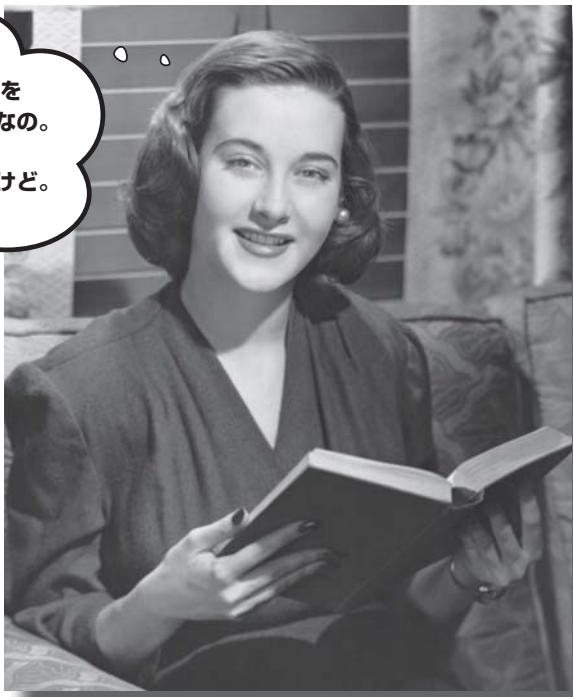
大切なのはユーザと一緒に時間を過ごすことです。

ウェブページはそれを求めています。

2章 データを格納する

あらゆる物には
然るべき場所がある

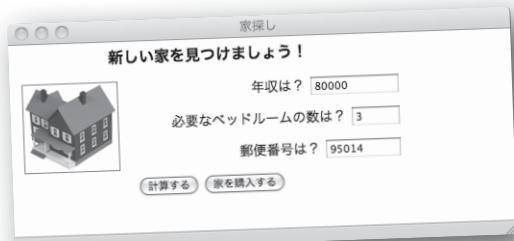
女性には大切な所持品を
しまう特別な場所が必要なの。
現金と逃亡用の偽の
パスポートはもちろんだけど。



現実世界では人々は物を収納する場所の重要性をしばしば見落としますが、JavaScriptでは見落とすわけにはいきません。ウォークインクローゼットや車が3台も入るガレージのような豪華さはありませんが、JavaScriptではあらゆる物に然るべき場所があるので、然るべき場所に置く責任はあなたにあります。データを表現する方法、格納する方法、いったんどこかに置いた後にそれを見つける方法、これらすべてはデータをどう扱うかという問題です。JavaScriptは収納の専門家です。JavaScriptのデータで散らかった部屋に入って仮想のラベルや収納箱を使えば、あなたの思い通りにデータを扱うことができます。

スクリプトはデータを格納できます

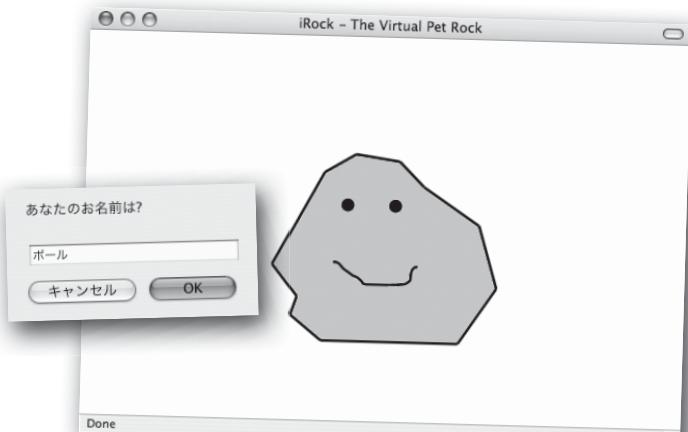
あらゆるスクリプトはなんらかの形でデータを扱う必要があり、通常はデータをメモリに格納します。ウェブブラウザが内蔵するJavaScript インタープリタはJavaScriptのデータのために記憶領域を確保する責任がありますが、データが何でありそれをどのように使うつもりであるか正確に記述するのはあなたの仕事です。



家探しに関連する情報はすべて、計算を実行するスクリプトの中に格納する必要があります。

スクリプトは格納されたデータを使って計算を実行したりユーザに関する情報を記憶します。データを格納することができなければ、新しい家を探したり iRock とほんとに仲良くなることはできないでしょう。

iRock のページで入力された
ユーザの名前はスクリプトが
挨拶文を表示するときのために
格納されます。



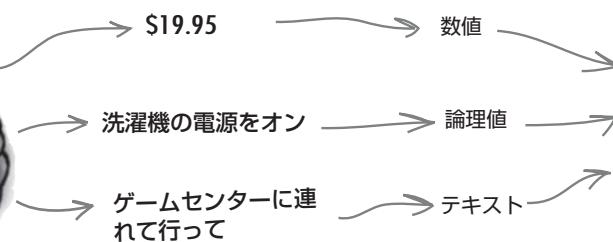
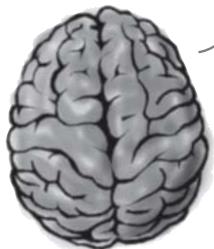
頭の体操

日常的に扱うさまざまな現実の情報の断片について、それらの類似点や違いについて考えてみましょう。こうしたさまざまなデータの断片をどのように整理していますか？

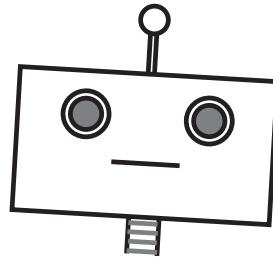
スクリプトはデータ型で考える

現実世界のデータは名前、数値、音声といった型で分類されます。JavaScriptもデータをデータ型で分類します。データ型はあなたの頭の中にある情報をJavaScriptに対応づける鍵になります。

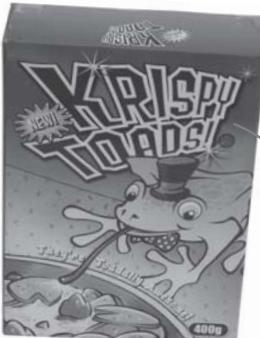
人間の脳



JavaScript



JavaScriptの3つの基本データ型:
テキスト、数値、論理値。



テキスト

テキストデータは文字のシーケンスにすぎません。テキストは通常は語や文ですが、それ以外のテキストもあります。JavaScriptのテキストは**文字列**とも呼ばれます。テキストは二重引用符 ("") や 単一引用符 (' ') で囲みます。

数値

数値は物の重さや量といった数値データを格納するのに使われます。JavaScriptの数値は整数(2 ポンド)か小数(2.5 ポンド)になります。



論理値

論理値データは常に値が true か false のどちらかになります。そのためトースターのオン/オフのスイッチのように二値をもつものは論理値を使って表現できます。論理値はつねに白黒がはっきりしているので、何かを決定するのに役立ちます。意思決定については 4 章で説明します。

データ型は JavaScript コードにあるデータを扱う方法に直接影響します。たとえばアラートボックスは数値ではなくテキストを表示します。そのため画面の裏で数値はテキストに変換されてから表示されています。



自分で考えてみよう

JavaScriptのデータ型で表現できるものをすべて探し出して、そのデータ型を書いてください。





自分で考えてみよう の答え

JavaScriptのデータ型で表現できるものすべて探し出して、そのデータ型を書いてください。





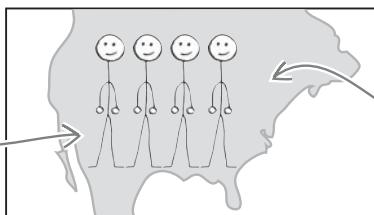
定数は一定ですが変数は変化します

JavaScriptでデータを格納するときデータ型だけでなくその**目的**も重要です。そのデータを使って何をしたいのか？ あるいはもっと具体的にスクリプトの実行を通してデータが変化するのかどうか？ その答によってJavaScriptのデータ型を変数と定数のどちらを使ってコードにするかが決まります。変数はスクリプトの実行を通して値が変わりますが、定数は値が変わりません。

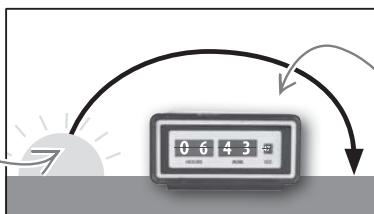
変数データは
値を変更できますが、
定数データは
値が一定です。

定数

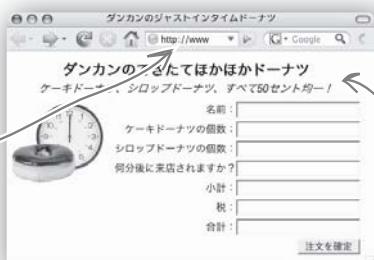
面積 350 万平方マイル。
定数（地殻変動が起こる
まで待てば別ですが）



1 日は 24 時間。定数
(月世界での一日は遅
いです)



ウェブページのURLが
www.duncansdonuts.com。
定数（ドーナツ業界が不況に
ならなければ）



人口 3 億人。合衆国の人口は
増えているので変数。

午前 6:43 に日の出。日の出の
時刻は毎日変わるので変数。

総ページヒットが 324。
ユーザがページに訪問
するたびにヒット数は
変わるので変数。

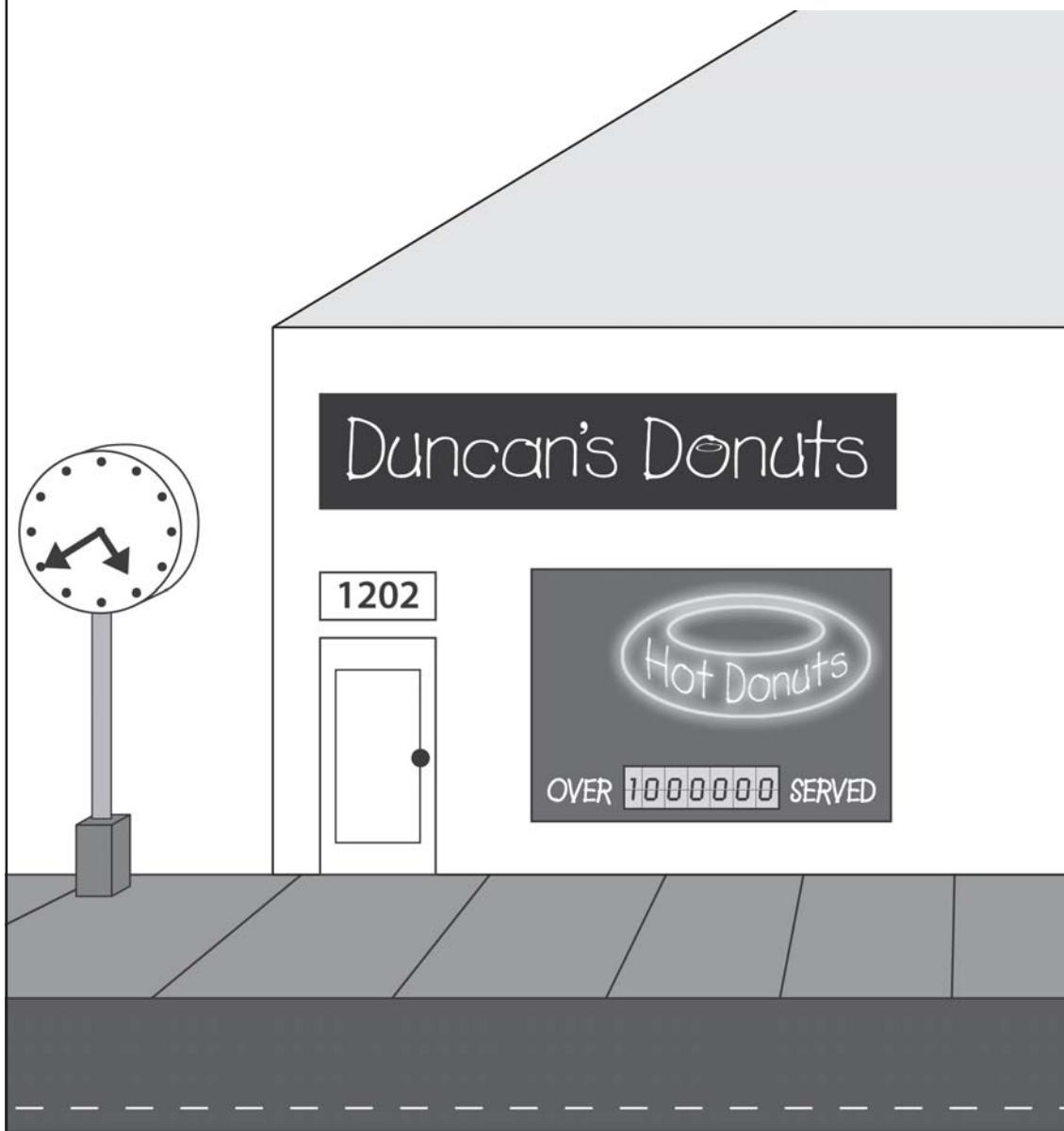


頭の体操

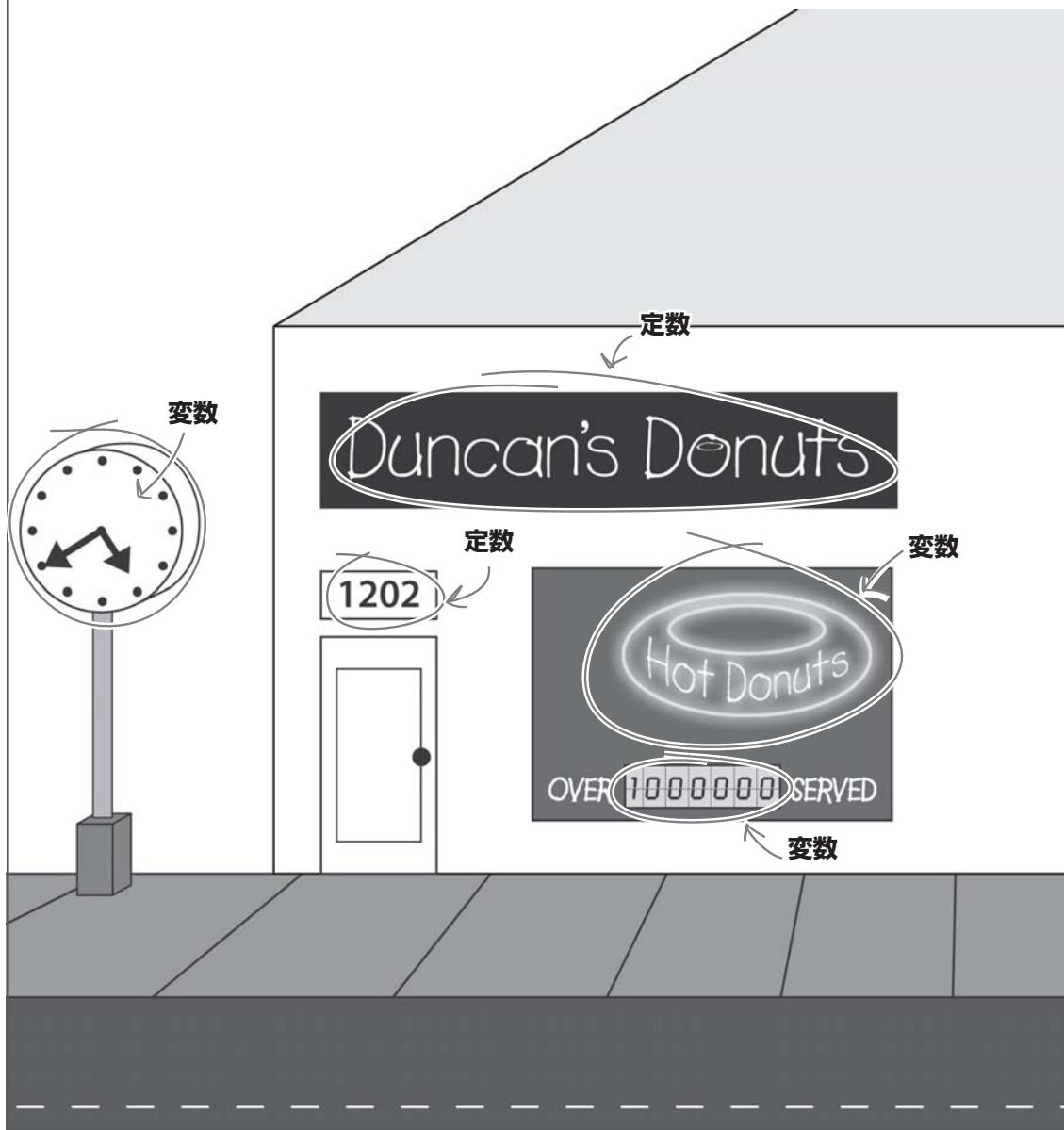
変数と定数の両方を含むことができる情報の型を考えてみましょう。



ダンカンドーナツにあるデータをすべて丸で囲み、それが変数なのか定数なのか記入してください。



自分で考えてみよう
の答え



特別座談会



今夜の対談：
変数と定数がデータ記憶について討論します。

変数：

データを柔軟に格納できる点では誰にも負けないね。いつでも好きなときに値が変えられる。今設定した値を後から別の値に変えられる。これが自由ってやつさ。

まあ、きみみたいな変化を拒む頑固者は、始終データの値が変わらざるをえない状況では通用しないさ。ロケットの発射のカウントダウンを考えてみろよ。きみにあれが扱えるかい？

それはそうかもな。まあ、変化を悪いことのように言うのは止めてくれないかな。変化は良いことだとは思わないのかい？ ユーザが入力した情報を格納して、計算を実行するときのことを考えてみなよ。

お互いの違いを認めるしかないってことだな。

定数：

そういうのをちゃんとばらんって言うんだよ。手にした値は離さない。私の頑固なまでの一貫性がスクリプト書きに評価されてるわけさ。いつも同じ値のままで、道を外れないからね。

なんだか、自分だけがミッションクリティカルなアプリケーションのためのデータ記憶オプションだと思ってるみたいだけど、そいつは違うな。発射台に据え付けられる前のロケットのことを考えてみなよ。誰か賢いやつがいて、発射日を定数にしているわけさ。もしそれが変数だったら、プロジェクトは予定より遅れまくりだろうね。

変化が起きれば起きるほど、同じであることも重要になるのさ。実際のところ、なんで変化が一番なんだい？ 最初から良い値を設定しておいて、ずっとそのままにしておけばいいんだよ。何が起きようと、ある値のまま変わらないって考えると安心できるしね。

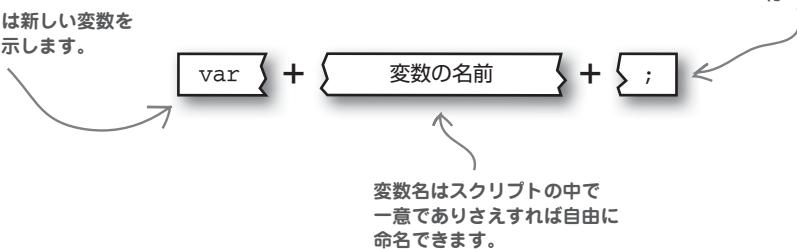
まったく、きみとはずっと話が合わないよ。

変数は値がなくても作成できる

変数はメモリ領域に確保される場所であり、一意的な名前を持っています。ちょうど物を収納する箱にラベルが張ってあるのと同じです。変数を作成するときはJavaScriptの特別なキーワードvarを使い、varの後に変数名を続けます。キーワードとはJavaScriptの予約語で、変数を作成するといった特別なタスクを実行します。

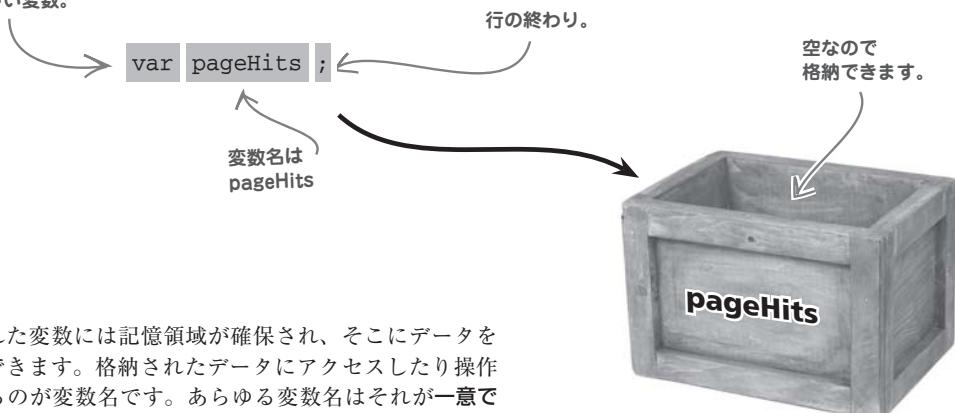
セミコロンは
JavaScriptコードの
行の終わりを示します。

キーワードvarは新しい変数を作成することを示します。



キーワードvarを使って変数を作成すると、変数の値は最初は空です。つまり値がありません。変数に値をまだ代入していない状態で値を読もうとしないかぎり、変数が最初は空であっても問題にはなりません。MP3プレイヤで音楽のダウンロードが終わらないと曲を再生できないのと同じです。

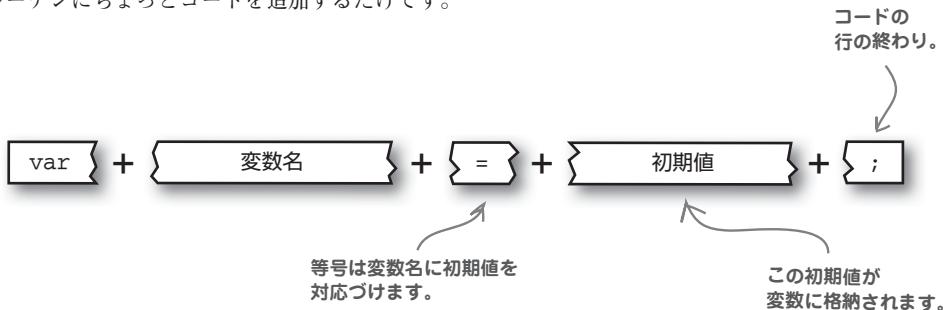
これは
新しい変数。



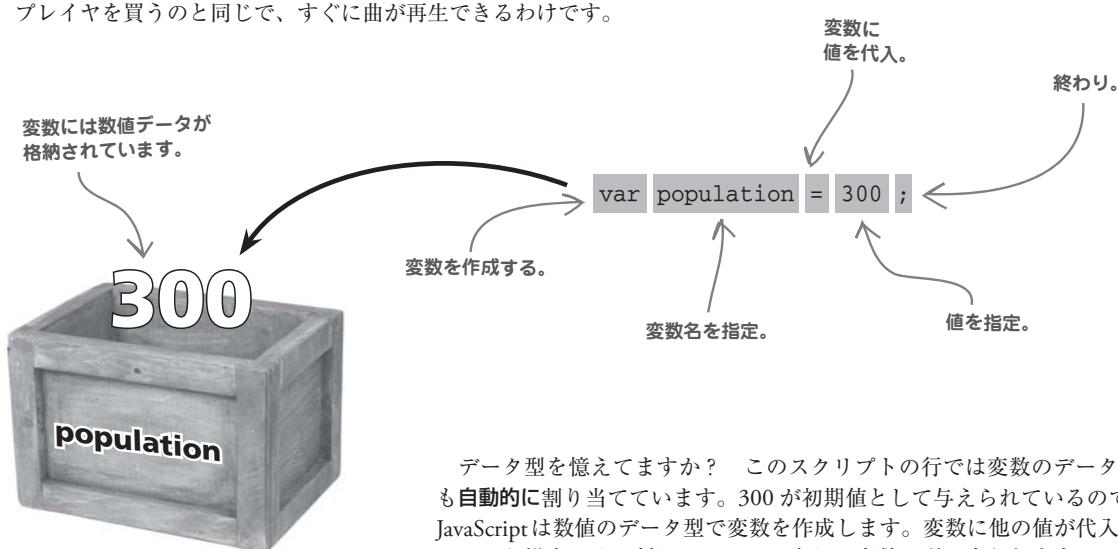
新たに作成された変数には記憶領域が確保され、そこにデータを格納することができます。格納されたデータにアクセスしたり操作する際の鍵になるのが変数名です。あらゆる変数名はそれが一意であり意味のある名前にすることが重要です。たとえば、pageHitsという変数名の場合、その変数にどんなデータが格納されるか一日でわかります。この変数名がxやgerkinだったらほとんど意味がわかりませんね。

変数を "=" で初期化する

初期値なしで変数を作成する必要はありません。変数を作成するとき変数に値を与える（変数を初期化する）のはいいことです。必要な作業は通常の変数作成ルーチンにちょっとコードを追加するだけです。



初期化されていない変数と違って、初期化された変数はすでに変数に値があるので、すぐに使い始めることができます。すでに曲が入ってるMP3プレイヤを買うのと同じで、すぐに曲が再生できるわけです。



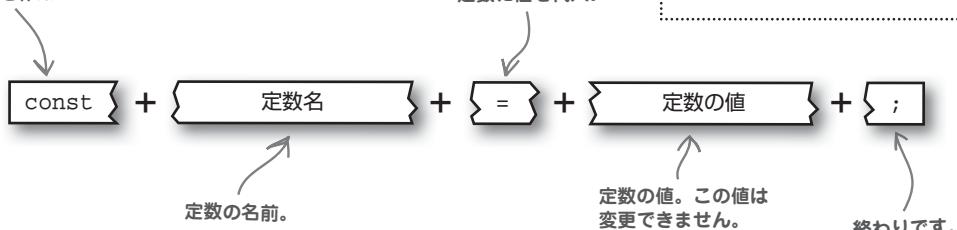
データ型を憶てますか？ このスクリプトの行では変数のデータ型も自動的に割り当てています。300 が初期値として与えられているので、JavaScriptは数値のデータ型で変数を作成します。変数に他の値が代入されていた場合、その新しいデータに応じて変数の型が変わります。ほとんどの場合JavaScriptは自動的にこの処理を行います。場合によっては明示的に別のデータ型に変換する必要がありますが、これについては後で説明します。

定数は変更しようとしても 変更できません

変数の初期化は変数の最初の値を設定するだけです。変数の値は後から変えることができます。データ断片の値を変更できないように格納するには定数にする必要があります。定数は初期化された変数と同じように作成できますが、使用するキーワードはvarではなくconstになります。定数の初期値は恒久的な値になるので値は変わりません。

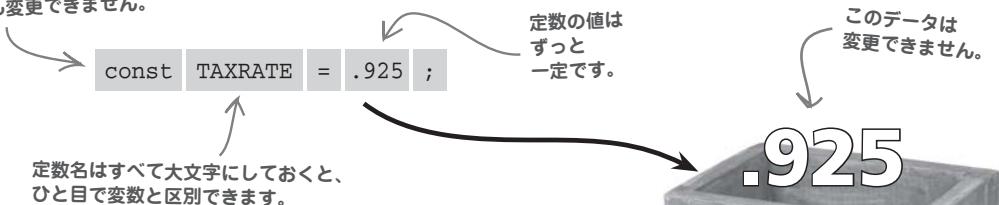
値を変更できない
定数を作成します。

定数に値を代入。



定数の作成と変数の作成の最も大きな違いは、使用するキーワードが前者がconstで後者がvarになる点です。構文は変数を初期化するのと同じになります。ただし定数の名前はコードの中で変数と区別できるように多くの場合すべて大文字になります。

このデータはこれまで、
これからも変更できません。



定数はたとえば消費税率のようにスクリプトで直接コードに書く情報を格納するのに便利です。0.925という数値を使うのではなくTAXRATEという内容のわかる名前の定数を使うことでコードが読みやすくなります。スクリプトの定数の値を別の値に変更する必要があれば、定数が定義されている箇所だけを変更すればいいので、変更は1カ所ですみます。定数にしていないとスクリプトのあちこちを探して変更する必要があるので作業が面倒になります。



要注意！

あらゆるブラウザで
constがサポート
されているわけでは
ありません。

constは比較的最近JavaScriptに導入されたキーワードなので、あらゆるブラウザでサポートされているわけではありません。constを使うJavaScriptをリリースする前にターゲットとなるブラウザで動作確認を行ってください。

TAXRATE

ちょっと待って、
定数は変更できないと
思っていたんだけど。



定数はテキストエディタを使わないかぎり 変更できません。

たしかにスクリプトの実行中に定数を変更することはできません。しかし定数を最初に作成するとき値を変更することはできます。スクリプトから見れば定数は絶対に変えられませんが、あなたから見れば定数を作成する時点に戻って値を変えることはできます。消費税の定数をスクリプトの実行中に変えることはできませんが、税率を設定している初期化コードを変更すれば、スクリプトには新しい定数の値が反映されます。



エクササイズ

以下の情報断片は変数と定数のどちらにすべきか決めて、作成するコードを書いてください。必要ならば初期化してください。



現在の気温。初期値は不明です。



人間の年を犬の年にするときの換算単位（人間の 1 年 = 犬の 7 年）



ロケット発射のカウントダウン（10 から 0 に）



甘いドーナツの値段（50 セント）



エクササイズの
答え

以下の情報断片は変数と定数のどちらにすべきか決めて、作成するコードを書いてください。必要ならば初期化してください



現在の気温。初期値は不明です。



人間の年を犬の年にするときの換算単位（人間の1年 = 犬の7年）



ロケット発射のカウントダウン（10から0に）



甘いドーナツの値段（50セント）

```
var temp;
```

↑
気温はつねに変わらし
値もわからないので、
初期値は設定しません。

```
const HUMANTODOG = 7;
```

↑
この換算単位は変わらないので
定数に最適です。

```
var countdown = 10;
```

↑
10から0のカウントダウンなので
変数にします。初期値は10にします。

```
var donutPrice = 0.50; または const DONUTPRICE = 0.50;
```

↑
ドーナツの値段が変わるので
変数にします。初期値は現在の
値段にします。

↑
ドーナツの値段が
変わらないなら定数に
した方がいいです。

素朴な疑問に

答えます

Q: JavaScriptのデータのデータ型を指定していないのですが、データ型を調べる方法ってあるのですか？

A: 他のいくつかのプログラミング言語と違ってJavaScriptでは定数や変数の型を明示的に指定することはできません。そのかわりデータの値を設定したときに暗黙の型が設定されます。JavaScriptの変数が柔軟なのは、別の値が設定されたときにデータ型が変わるからです。たとえばxという変数に数値の17を代入すると変数の型は数値になります。同じ変数に文字列「seventeen」を代入すると変数の型は文字列に変わります。

Q: JavaScriptのデータの型が自動的に設定されるとしたら、どうしてデータ型を気にする必要があるんですか？

A: JavaScriptが自動で行うデータ型の扱いに頼れない状況がたくさんあるからです。たとえば計算で使いたい数値がテキストとして格納されているかもしれません。数値として計算を実行するためには、テキストから数値に型を変換する必要があります。アラートボックスに数値を表示するときは逆で、数値を文字列に変換する必要があります。JavaScriptは数値からテキストへの変換を自動的に行いますが、期待通り

に変換されないかもしれません。

Q: 前もって値がわからない場合、変数を初期化しなくても平気ですか？

A: 大丈夫です。初期化の狙いは、変数に値がないとき変数にアクセスしようとして発生する問題をあらかじめ防ぐことにあります。変数を作成するとき変数の値を調べられないことがあります。その場合、その変数を使う前に変数に値が設定されているか確認する必要があります。変数はいつでも「何もない」値に初期化できます。テキストの場合は""、数値の場合は0、論理値の場合はfalseです。このように初期化しておけば、初期化されていないデータに誤ってアクセスしてしまう危険を減らせます。

Q: 変数と定数をいつ使い分ければいいのか、コツがあつたら教えてください。

A: 定数は変更できない、変数は変更できる、と言うのは簡単ですが、それだけでは終わりません。多くの場合、最初はなんでもかんでも変数にしておいて、後から変数のいくつかを定数に変えることになるでしょう。そのような場合でも、変数を定数

に変えることはそんなにありません。むしろ、挨拶の言葉や変換率など、固定されたテキストや数値の断片があって、それらがいくつかの場所で使われていることもあるでしょう。そうしたテキストや数値を何度も重複させるのではなく、それらを定数として作成し、定数を使うようにしておけば、定数の値を後で調整したり変更する必要が生じた場合でも、コードの一ヵ所を変更するだけで済みます。

Q: ウェブページが再読み込みされたとき、スクリプトのデータはどうなってしまいますか？

A: スクリプトのデータは初期値に再設定されます。それまでの状態は残りません。言い換えると、ウェブページを再読み込みで更新すると、スクリプトを最初に実行するときと同じ結果になります。

**変数や定数の値が設定されたとき
データ型が決まります。**

重要ポイント

- 通常スクリプトのデータは、テキスト、数値、論理値、これらのデータ型のいずれかで表現されます。
- 変数はスクリプトの実行を通して値が変更できるデータ断片です。
- 定数は値を変更できない情報断片です。
- キーワードvarは変数の作成に使います。constは定数の作成に使います。
- JavaScriptのデータ断片のデータ型はデータにある値が設定されたときに決まります。変数のデータ型は変更できます。

名前に使える文字は？

変数や定数、そしてJavaScriptの構文の構成要素は、**識別子**(identifier)として知られる一意の名前を使ってスクリプトの中で識別されます。JavaScriptの識別子は現実世界の人名と同じようなものですが、人名ほど柔軟ではありません（人名は同姓同名の場合もありますが、JavaScriptでは異なる変数を同じ名前にすることはできません）。識別子はスクリプトの中で一意にするだけでなくJavaScriptの命名規則に従う必要があります。



識別子の長さは1文字以上にします。



識別子の最初の文字に使えるのは、英字、アンダースコア(_)、ドル記号(\$)です。



識別子の2文字目以降に使えるのは、英字、アンダースコア(_)、ドル記号(\$)、数字です。



空白と(_)と\$(以外の)特殊文字は識別子に使えません。

変数や定数にJavaScript識別子で名前をつけるとき、スクリプトの中で**意味のある情報断片**に対して名前をつけるわけです。そのため識別子の命名規則に従うだけでは十分ではありません。データ断片の名前を見ただけで内容がすぐにわかるように変数の意味を考慮した名前にしましょう。

もちろんxといった簡単な識別子で十分なこともあります。スクリプトの中のデータ断片がすべて内容が簡単にわかる用途であるとはかぎりません。

**識別子は命名規則に従うのはもちろん、
内容が簡単にわかる名前にしましょう。**

識別子に関する抜破りを
許すつもりはないね。

正義の保安官
J.S. ジャスティス。



変数名と定数名の合法性



違法：数字で開始できません。

`_topSecret`

合法：先頭文字に`_`を使うのは問題ありません。変数名に特別な意味を持たせたいとき使います。

`firstName`

合法：どの文字も問題なしです。

`top100`

合法：数字が先頭にはないのでOKです。

`ka_chow`

合法：英字と下線は問題なしです。

`$total`

合法：先頭文字に`$`を使うのは合法です。

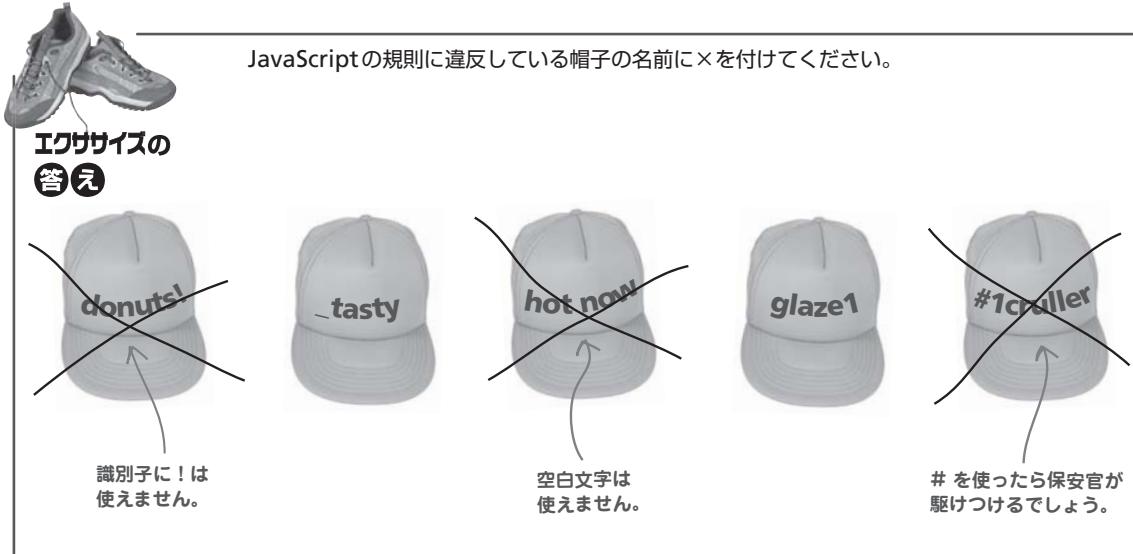
違法：`_`と`$`以外の特殊文字で開始できません。



エクワサイズ

ダンカンドーナツのペストリー職人はプロモーションのために帽子のデザインを考えています。残念ながらいくつかのデザインがJavaScriptの識別子の命名規則に違反していることに気づいていません。JavaScriptの規則に違反している帽子の名前に`×`を付けてください。





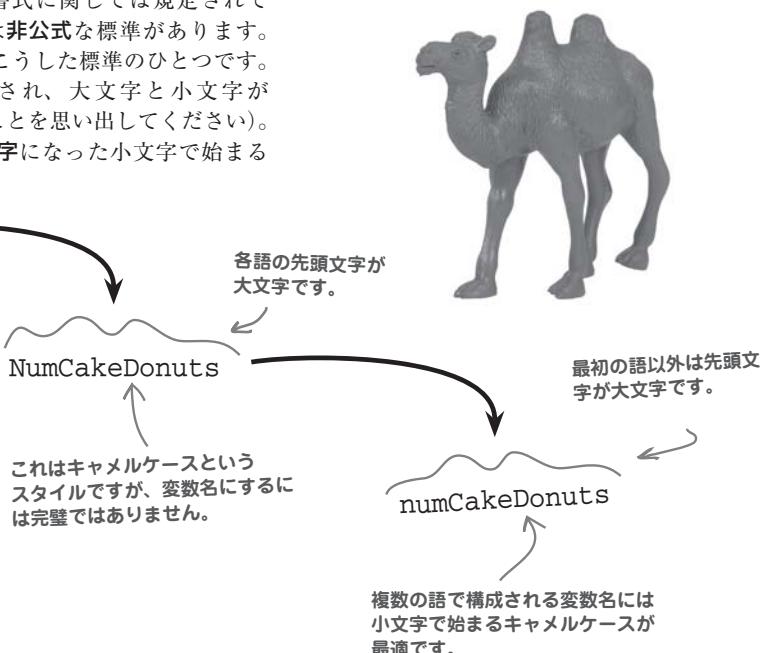
変数名にはキャメルケースを使います

JavaScriptの文法では識別子の名前の書式に関しては規定されていませんが、JavaScriptコミュニティには非公式な標準があります。キャメルケース (CamelCase) を使うのはこうした標準のひとつです。キャメルケースは複数の語から構成され、大文字と小文字が混在します（変数名には空白が使えないことを思い出してください）。変数名には通常は最初の語がすべて小文字になった小文字で始まるキャメルケースが使われます。

num_cake_donuts

変数の識別子で語を下線で区切るのは違法ではありませんが、もっと良いやり方があります。

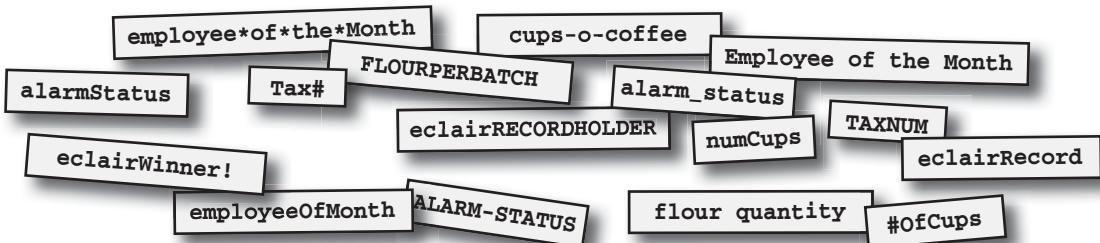
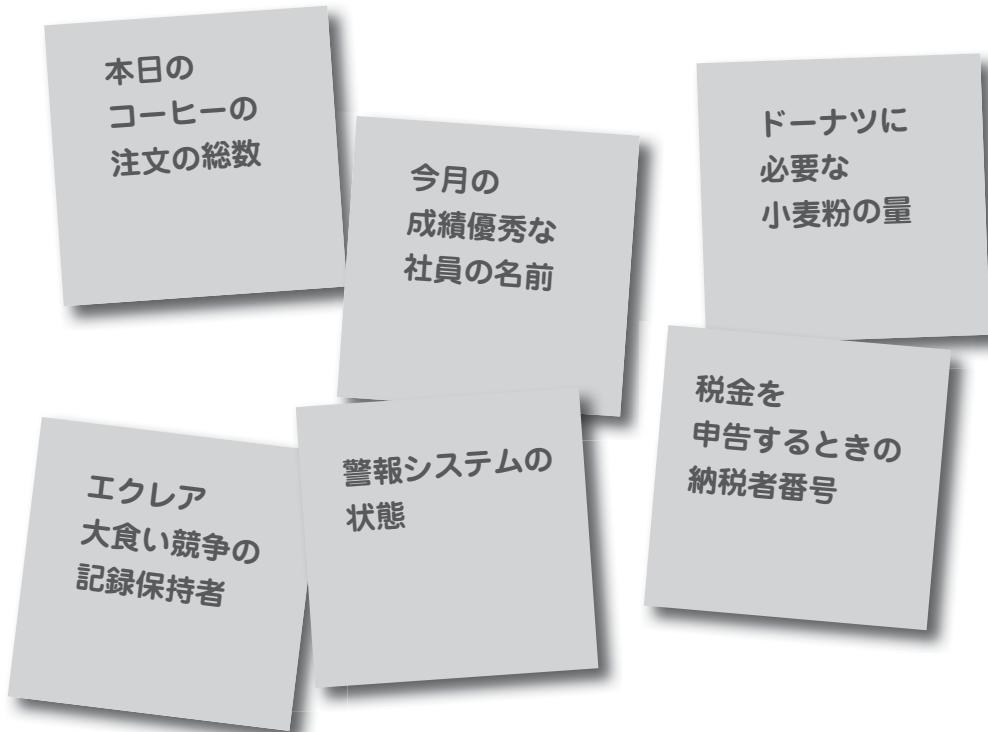
小文字で始まる
キャメルケースは
複数の語で構成される
変数名に使います。





JavaScriptマグネット

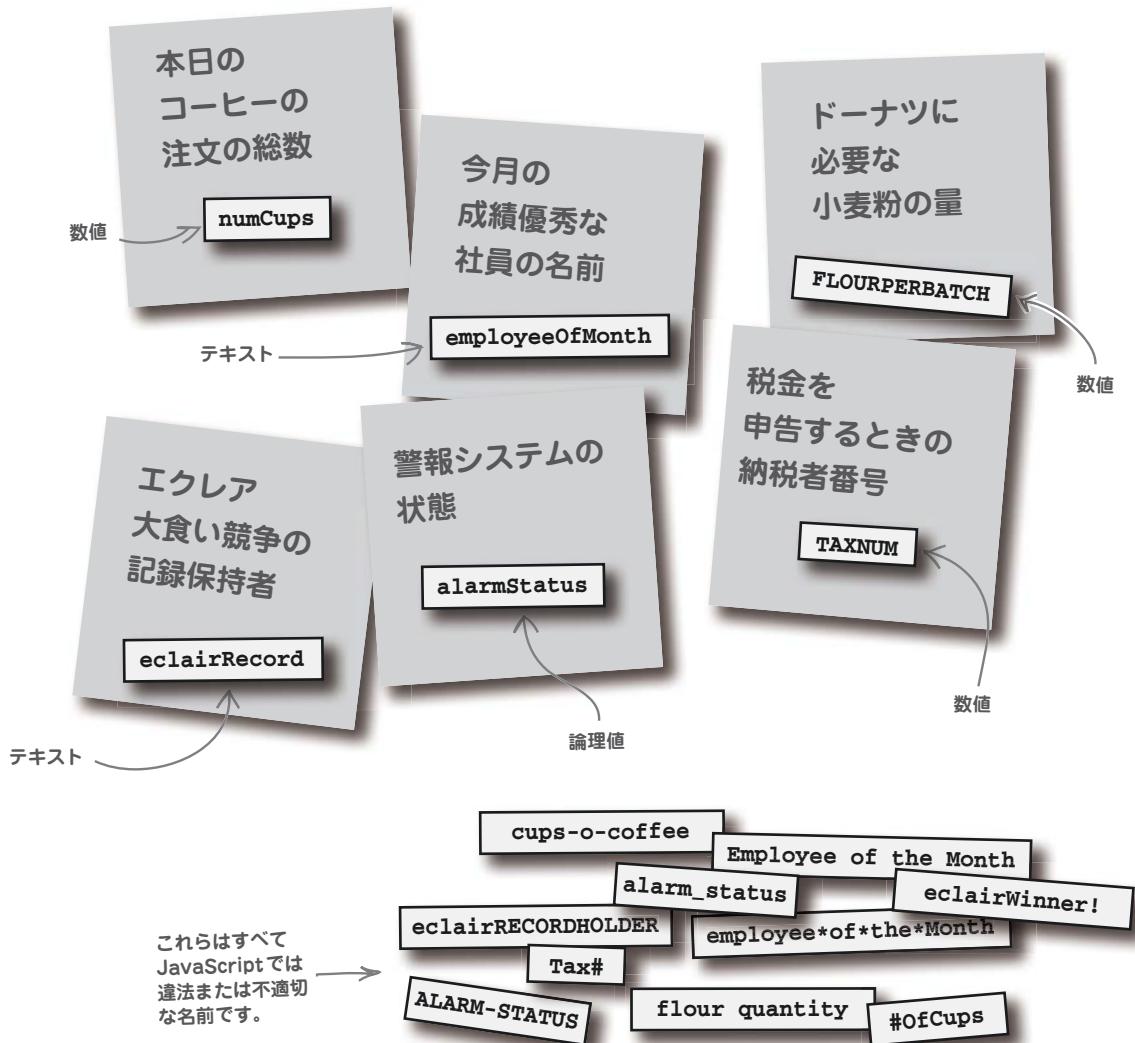
ダンカンドーナツで指定された変数と定数が外れて、識別子のマグネットになっています。変数/定数の説明に対応する正しいマグネットを選んでください。命名規則に違反したマグネットがあるので注意しましょう。データ型も記入するとボーナスポイントになります。





JavaScriptマグネットの答え

ダンカンドーナツで指定された変数と定数が外れて、識別子のマグネットになっています。変数/定数の説明に対応する正しいマグネットを選んでください。命名規則に違反したマグネットがあるので注意しましょう。データ型も記入するとボーナスポイントになります。



次なる野望はドーナツ

ダンカンドーナツのことはご存知かもしれません、ダンカンがドーナツ市場をゆさぶる大きな計画を進めていることは知られていません。ダンカンはドーナツの事業をオンラインで展開しようと思っています。ジャストインタイムのドーナツという企画の内容は、オンラインで注文を受け取るとき受け取り時間を入力してもらって、その時間までに必ずできたてドーナツを作ってくれというものです。あなたの仕事は、消費税と注文の合計金額を計算するだけでなく、ユーザが必要なデータを入力しているか確認することです。



ダンカンのジャストインタイムドーナツ

ダンカンのできたてほかほかドーナツ
ケーキドーナツ、シロップドーナツ、すべて50セント均一!

| | |
|--------------------------------------|--------|
| 名前: | Paul |
| ケーキドーナツの個数: | 0 |
| シロップドーナツの個数: | 12 |
| 何分後に来店されますか? | 45 |
| 小計: | \$6.00 |
| 税: | \$0.55 |
| 合計: | \$6.55 |
| <input type="button" value="注文を確定"/> | |

はい、私がダンカンです。
このオンライン注文システムで
ドーナツを予約すると
すごいことになりますよ。

JavaScript がユーザの
入力を受け付けて税金と
合計を計算します。



ダンカンドーナツのウェブページを計画する

ジャストインタイムドーナツの注文を処理するには、注文フォームから必須データがあるかチェックして、そのデータとともに注文の合計を計算する必要があります。データが入力されたら小計と合計が計算されるので、ユーザは合計金額をすぐ確認できます。「注文を確定」ボタンは注文を送信するためがあるので、JavaScriptの問題とは別なのでここでは気にしないでください。



これらは注文に必須な情報
なので JavaScript で検証
すべきです。

ダンカンのジャストインタイムドーナツ

ダンカンのできたてほかほかドーナツ
ケーキドーナツ、シロップドーナツ、すべて50セント均一！

名前: Paul

ケーキドーナツの個数: 0

シロップドーナツの個数: 12

何分後に来店されますか? 45

小計: \$6.00

税: \$0.55

合計: \$6.55

注文を確定

これら情報は JavaScript を
使って自動的に計算されます。



最終フォームのサーバへの
送信は JavaScript とは
関係ありません。



小計はドーナツの数の合計とドーナツ単価を掛けて計算します



$$(\text{ケーキドーナツの数} + \text{シロップドーナツの数}) \times \text{ドーナツ単価}$$



税額は小計と税率を掛けて計算します



$$\text{小計} \times \text{税率}$$



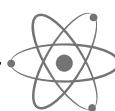
合計金額は小計と税額を足して計算します



$$\text{小計} + \text{税率}$$

ダンカンはかなりたくさんのデータをフォームで管理しようとしているようです。ユーザが入力したさまざまな情報断片だけでなく、JavaScriptコードで計算されたデータ断片もいくつかあります。

JavaScriptをちょっと
使うだけで、注文がすぐに
埋まるなんて、こいつは
天才だね！



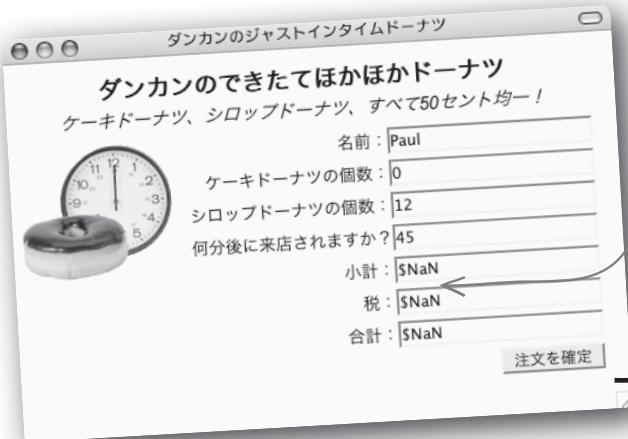
頭の体操

これらの計算を行うにはどんな変数と定数が必要でしょう？ それらの名前はどうなりますか？



ドーナツ計算の最初の一手

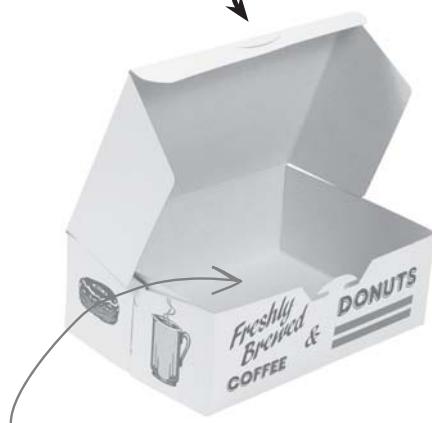
ダンカンはJavaScriptを書いて計算してみましたが、問題が生じました。ユーザーがドーナツの数を入力するとすぐに計算がめちゃくちゃになってしまいます。計算結果の値が\$NaNになって、これでは意味がありません。もっと悪いことに注文が埋まらないので、ダンカンの技術の向上をユーザはちっとも喜べません。



\$NaNになるのはコードが間違っているからでしょうか？

これではダメです！
x 0

ドーナツのスクリプトのコードをみて、何を実行しているか調べてみましょう。次のページをざっとみて何が起きているか考えてみましょう。



ドーナツがない = 大問題

このコードが呼び出されると小計と合計が計算され、注文を更新します。

ユーザ入力には問題がなさうなので、定数に問題があるはずです。

このコードは注文をサーバに送信します。

ドーナツの個数が変わると注文が更新されます。

「注文を確定」ボタンがクリックされると注文が送信されます。

```
<html>
  <head>
    <title> ダンカンのジャストインタイムドーナツ </title>
    <link rel="stylesheet" type="text/css" href="donuts.css" />
    <script type="text/javascript">
      function updateOrder() {
        const TAXRATE;
        const DONUTPRICE;
        var numCakeDonuts = document.getElementById("cakedonuts").value;
        var numGlazedDonuts = document.getElementById("glazeddonuts").value;
        var subTotal = (numCakeDonuts + numGlazedDonuts) * DONUTPRICE;
        var tax = subTotal * TAXRATE;
        var total = subTotal + tax;
        document.getElementById("subtotal").value = "$" + subTotal.toFixed(2);
        document.getElementById("tax").value = "$" + tax.toFixed(2);
        document.getElementById("total").value = "$" + total.toFixed(2);
      }
      function placeOrder() {
        // サーバに注文を送信
        form.submit();
      }
    </script>
  </head>
  <body>
    <div id="frame">
      ...
      <form name="orderform" action="donuts.php" method="POST">
        ...
        <div class="field">
          ケーキドーナツの個数: <input type="text" id="cakedonuts" name="cakedonuts" value="" onchnage="updateOrder();"/>
        </div>
        <div class="field">
          シロップドーナツの個数: <input type="text" id="glazeddonuts" name="glazeddonuts" value="" onchnage="updateOrder();"/>
        </div>
        ...
        <div class="field">
          <input type="button" value="注文を確定" onclick="placeOrder(this.form);"/>
        </div>
      </form>
    </div>
  </body>
</html>
```

自分で考えてみよう



ダンカンのジャストインタイムドーナツのスクリプトコードで問題が発生した理由を書いてください。



ダンカンのジャストインタイムドーナツのスクリプトコードで問題が発生した理由を書いてください。

TAXRATEとDONUTPRICE、この2つの定数が初期化されていないので、これらの定数を使う計算が失敗しています。



定数がいつも同じ値なのは
わかったけど、初期化しなかった
場合はどうなるの？

定数を初期化しないのは良くありません。

定数に値をあたえなければ定数を初期化せずにおくこともできますが、**とても悪い考えです**。定数を作成するとき初期化しないと、その定数には値がないだけでなく、もっと悪いことに値を代入することもできません。初期化されていない定数があるのは、ブラウザがエラーを報告しないとしても、**コーディングのエラー**です。

定数を作成するときは
必ず初期化するように。



データを初期化しましょう、そうしないと

データ断片を初期化しないと未定義 (undefined) と見なされます。未定義のときは値がないことになり、情報が含まれていないことを意味します。初期化されていない変数や定数を使おうとすると問題が生じます。

```
const DONUTPRICE;  
var numCakeDonuts = 0;  
var numGlazedDonuts = 12;  
var subTotal = (numCakeDonuts + numGlazedDonuts) * DONUTPRICE;
```

初期化されていません

初期化されています

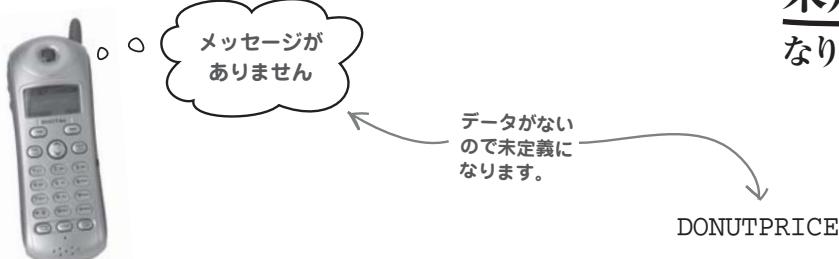
JavaScript では
× ではなく * を使って
掛け算を実行します

? ?

これが
大問題ですね。

`DONUTPRICE`は初期化されていないので、値がありません。JavaScriptには「値がない」状態を示す特殊な値があります。それが未定義(`undefined`)です。これは留守電で「メッセージがありません」と報告されるのと同じです。「メッセージがありません」はメッセージがないことを示すのが目的のメッセージなのです。未定義も同じでデータがないことを示します。

データ断片に
値がないと
未定義に
なります。



NaN は数値でない (Not a Number)

未定義が特殊なデータ条件を表現するのと同じように、NaNはJavaScriptの変数が特殊な状態であることを示す重要な値です。NaNはNot a Numberの略です。subTotalには計算を実行するのに十分な情報がないのでこの値が設定されます。つまり値がないものを数値として扱おうとするとNaNになります。

```
数値 →
subtotal = (0 + 12) * ? = NaN ← 数値でない!
↑
データが未定義なので
計算できません。
```

NaNの問題を解決するにはDONUTPRICEを作成するときに初期化する必要があります。

```
const DONUTPRICE = 0.50;
```

NaNは
数値であると
期待しているのに
数値でない
ことを示します。

素朴な疑問に 答えます

Q: スクリプトの中で識別子が一意でなければならるのは、なぜですか？

A: 識別子の要点は、スクリプトの中の情報断片を特定するために使える一意な名前として機能することです。現実世界では名前が同じ人がいることはそんなに珍しいことではありませんが、同姓同名の人がいてもその人を本人として対応することができます。JavaScriptはこうした曖昧さに対応できないので、情報断片を注意深く区別できるように名前を別にする必要があります。そのためスクリプトの中の識別子がすべて一意でなければならないのです。

Q: 作成する識別子は、あるスクリプトの中でだけ一意であればいいのですか、それともすべて一意でなければならないのですか？

A: 識別子が一意でなければならぬのは、ひとつのスクリプト

の中において、場合によってはひとつのスクリプトの中のある部分においてだけです。しかしだ規模なウェブアプリケーションになると大きなファイルを多数使うことになります。すべての識別子が一意であることを確認するのは簡単ではありません。とはいってもスクリプトの中で識別子を一意に保つのはそれほど難しくはありません。識別子の名前をできるだけ内容がわかる説明的な名前にするのがコツです。

Q: キャメルケースと小文字で始まるキャメルケースは、どう使い分けたらいいんでしょうか？

A: キャメルケース（最初の語は小文字で始まります）はJavaScriptオブジェクトの名前にだけ適用します。これについては9章で説明します。小文字ではじまるキャ

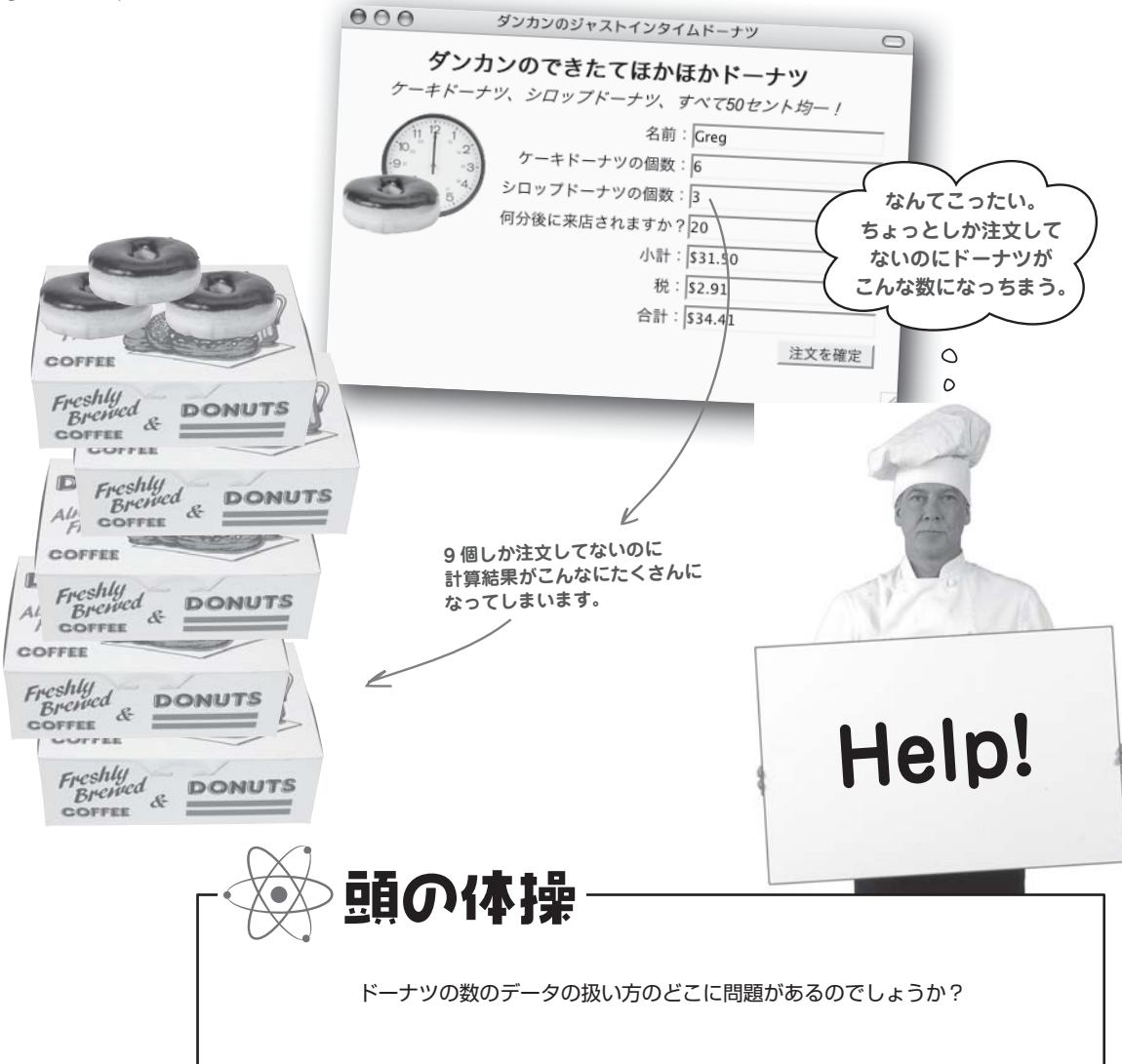
メルケースは変数と関数に適用します。オブジェクトの名前がDonutなどの場合はキャメルケースだからで、関数の名前がgetDonut()で変数の名前がnumDonutsなどの小文字ではじまるキャメルケースだからです。定数の名前はすべて大文字にします。

Q: テキストと論理値のデータはNaNとみなされるのでしょうか？

A: 理論的にはどちらも数値ではありません。NaNの目的は、数値と思っていたのに数値ではないことを示すことにあります。言い換えると、NaNは通常のJavaScriptデータの説明というより、むしろ数値データ型のエラーを示すものなのです。計算の実行中に数値と思って処理したら実は数値でなかったときにNaNが発生します。

さて、ダンカンドーナツはどうなったでしょう？

ダンカンドーナツに戻ってみたら、事態はさらに悪化していました。箱が空になるかわりにあちこちがドーナツだらけになりました。計算が間違っていて注文の数が多くなりすぎています。ダンカンはドーナツが多すぎるとの苦情に参っています。



加算できるのは数値だけではありません

JavaScriptでは文脈がすべてです。特に与えられたコード断片でデータを使って何をしているかだけではなく、どんな種類のデータを操作しているのかも重要なです。2つの情報断片を加算するという簡単な操作であっても、データの型によってまったく異なる結果になります。

$1 + 2 = 3$



数値の加算

2つの数値の加算は期待した結果になります。2つの値の和が計算結果になります。

`"do" + "nuts" = "donuts"`



あれとこれをくっつけることを意味します。

文字列の連結

2つの文字列の「加算」も期待した結果になりますが、数値の和とは異なる結果になります。2つの文字列を連結した結果になります。

テキストの文字列の加算は数値の加算と異なることがわかりました。では2つの数値を示すテキストを加算しようとしたら何が起きると思いますか？

`"1" + "2" = ?` ← 加算か連結か、どっちになる？

JavaScriptはテキストの文字列の内容についてはまったく気にしません。それは文字の集まりでしかないです。文字列の内容が数値であっても違いはありません。文字列の連結が実行されるだけなので、数値の加算を期待していた場合、予期せぬ結果になります。

`"1" + "2" = "12"` ←

結果は文字列なので加算の結果とは異なります。

数値ではなく文字列なので文字列の連結になります。



要注意！

加算する対象が加算できる対象なのか常に注意しましょう。

数値を加算するつもりなのに文字列の連結になってしまるのはJavaScriptでよくある失敗です。数値として加算するのであれば、文字列を数値に確実に変換しておく必要があります。

parseInt() や parseFloat() を使ってテキストを数値に変換します

加算/連結の問題はともかくとして、文字列として格納されたデータを数値として計算する必要がある状況があります。このような状況では、計算処理を行う前に文字列を数値に変換する必要があります。こうした変換を実行するためJavaScriptでは2つの便利な関数が提供されています。

parseInt()

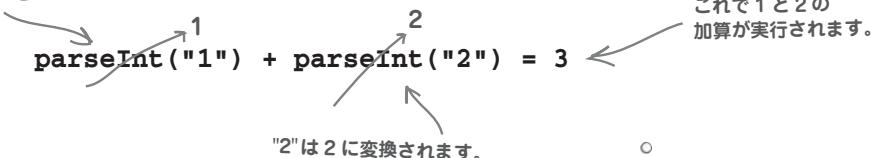
引数で文字列を渡すと
数値に変換します。

parseFloat()

引数で文字列を渡すと（10進数の）
浮動小数点数に変換します。

どちらの組み込み関数も、受け取った文字列を数値に変換して返します。

`parseInt()` は "1" を 1 に
変換します。



`parseInt()` と `parseFloat()` は常に正しく動作するとはかぎりません。あなたが正しく情報を渡したときだけ正しく動作します。これらの関数はベストをつくして文字列から数値に変換しますが、そのためには数値を構成する文字だけを含む文字列を渡さなければなりません。

`parseFloat("$31.50") = NaN`

文字 \$ は関数を混乱させるので、このコードは問題あります。

ああ、びっくり、結果は数値ではないです。



この関数のこと
少し混乱したとし
ても心配しないで
ください。

すこし後で関数の正式な内実について説明しますので、いまのところは関数というのは情報を渡すと何かを返すものだと考えてください。

どうして大量のドーナツ注文になったのか？

ジャストインタイムドーナツの注文フォームをよく見てみましょう。
どうして大量のドーナツが誤って注文されたのかがわかります。

ダンカンのジャストインタイムドーナツ

ダンカンのできたてほかほかドーナツ
ケーキドーナツ、シロップドーナツ、すべて50セント均一！



名前 : Greg

ケーキドーナツの個数 : 6

シロップドーナツの個数 : 3

何分後に来店されますか? 20

小計 : \$31.50

税 : \$2.91

合計 : \$34.41

注文を確定

ドーナツの単価で小計を割り算しています。これで
ドーナツが何個注文されたかわかります。

注文の小計。
 $\$31.50 / \$0.50 = 63 \text{ 個}$

実際に注文された
ドーナツの個数…ははあ。

ドーナツの単価

フォームのデータがその内容は何であれ文字列として格納されることを考慮すると、数値の文字列で加算を行ったのが原因が問題が発生したと考えて間違いないでしょう。フォームフィールドに入力されたのは数値ですが、JavaScriptからみると単なるテキストにすぎません。間違って文字列の連結として解釈されないように、文字列を実際の数値に変換してから数値の加算を行う必要があります。

"1" + "2" = "12"でしたね。
どうやらこれが原因のようです。



自分で考えてみよう

以下のコード断片はドーナツの数量をフォームフィールドから取得します。これを使ってドーナツの数量が文字列から数値に変換されるように、ダンカンのupdateOrder()のコードの空欄を埋めてください。

```
document.getElementById("cakedonuts").value
```

このコードはドーナツフォームに
入力されたケーキドーナツの
個数を取得します。

```
document.getElementById("glazeddonuts").value
```

このコードはドーナツフォームに
入力されたシロップドーナツの
個数を取得します。

```
function updateOrder() {
  const TAXRATE = 0.0925;
  const DONUTPRICE = 0.50;
  var numCakeDonuts =
  .....  

  var numGlazedDonuts =
  .....  

  if (isNaN(numCakeDonuts))
    numCakeDonuts = 0;
  if (isNaN(numGlazedDonuts))
    numGlazedDonuts = 0;
  var subTotal = (numCakeDonuts + numGlazedDonuts) * DONUTPRICE;
  var tax = subTotal * TAXRATE;
  var total = subTotal + tax;
  document.getElementById("subtotal").value = "$" + subTotal.toFixed(2);
  document.getElementById("tax").value = "$" + tax.toFixed(2);
  document.getElementById("total").value = "$" + total.toFixed(2);
}
```



自分で考えてみよう の答え

以下のコード断片はドーナツの数量をフォームフィールドから取得します。これを使ってドーナツの数量が文字列から数値に変換されるように、ダンカンのupdateOrder()のコードの空欄を埋めてください。

```
document.getElementById("cakedonuts").value
```

どちらも整数なのでparseInt()
を使って変換します。

```
document.getElementById("glazeddonuts").value
```

```
function updateOrder() {  
    const TAXRATE = 0.0925;  
    const DONUTPRICE = 0.50;  
    var numCakeDonuts =  
        parseInt(document.getElementById("cakedonuts").value);  
    var numGlazedDonuts =  
        parseInt(document.getElementById("glazeddonuts").value);  
    if (isNaN(numCakeDonuts))  
        numCakeDonuts = 0;  
    if (isNaN(numGlazedDonuts))  
        numGlazedDonuts = 0;  
    var subTotal = (numCakeDonuts + numGlazedDonuts) * DONUTPRICE;  
    var tax = subTotal * TAXRATE;  
    var total = subTotal + tax;  
    document.getElementById("subtotal").value = "$" + subTotal.toFixed(2);  
    document.getElementById("tax").value = "$" + tax.toFixed(2);  
    document.getElementById("total").value = "$" + total.toFixed(2);  
}
```

toFixed()は小数点以下2桁で
四捨五入します。



重要ポイント

- JavaScriptで厳格に決められているわけではありませんが、定数はすべて大文字で、変数は小文字ではじまるキャメルケースで、それぞれコーディングするのがいいでしょう。
- 定数を作成するときは必ず初期化しますが、変数はできるかぎり初期化しましょう。
- 変数が初期化されないと値が代入されるまで未定義のままになります。
- NaNはNot a Numberの略で、データ断片が数値であると期待される文脈で数値でないことを示すのに使われます。
- 文字列の連結は数値の加算とは違っていますが、どちらも同じプラス記号(+)を使います。
- 組み込みのparseInt()とparseFloat()は文字列を数値に変換するのに使う関数です。

問題が解決しました

JavaScriptが修正されたのでダンカンは大喜びです。ついに彼は正しい注文を受け取れるようになり、ビジネスも大好評です。



ダンカンが見つけた営業妨害

ダンカンは新しい問題をかかえています。フランキーが営業妨害をしかけているのです。フランキーはダンカンの店の通りの向かいでホットドッグを売っています。問題なのは、フランキーが匿名で偽の注文を送信するという汚い手を使っていることです。注文した人の名前がない注文があるのは良くありません。

ダンカンのジャストインタイムドーナツ

ダンカンのできたてほかほかドーナツ
ケーキドーナツ、シロップドーナツ、すべて50セント均一！

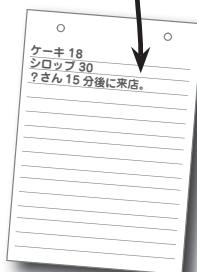
名前：
ケーキドーナツの個数：18
シロップドーナツの個数：30
何分後に来店されますか？15
小計：\$24.00
税：\$2.22
合計：\$26.22

注文を確定



名前が入力されなくても注文が受け付けられてしまいます。

フランキーのことは気にしないけど、ドーナツのコードはデータの受け取り方を貰くする必要があるな。

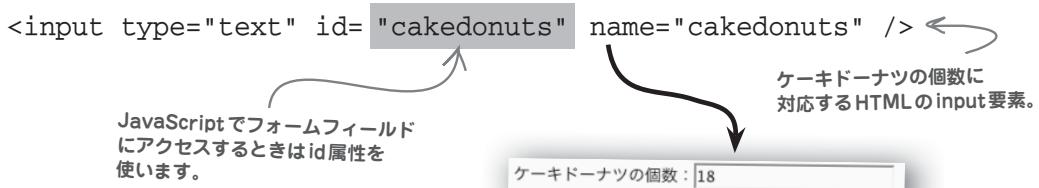


ダンカンは偽の注文のせいで時間とエネルギーとドーナツを無駄にしています。フォームデータが送信される前にデータがすべて入力されているか確認する必要があります。

getElementById()を使ってフォームデータをつかむ

フォームデータを検証するには、ウェブページからデータをつかむ方法が必要です。

JavaScriptでウェブページの要素にアクセスする鍵になるのがHTMLタグのid属性です。



JavaScriptではgetElementById()を使ってウェブページの要素をIDで取得することができます。この関数は要素のデータを直接つかむのではなく、HTMLフィールド自体をJavaScriptのオブジェクトとして提供します。そのためこのフィールドのvalueプロパティ経由でデータにアクセスします。

正確にはgetElementById()は関数ではなくdocumentオブジェクトのメソッドです。

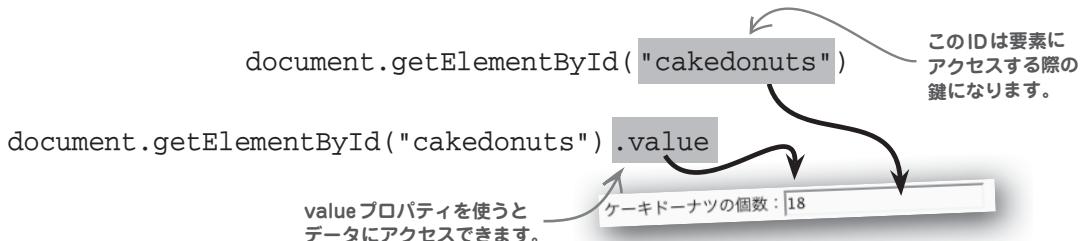
document.getElementById()
ウェブページの要素のIDをこのメソッドに与えると、その要素自体が戻ってくるので、ウェブデータのアクセスに使えます。

getElementById()は
documentオブジェクトに
属します。



オブジェクト、プロパティ、メソッドについて、いまは気にしないでください。

JavaScriptはオブジェクトと呼ばれる高度なデータ型をサポートしています。JavaScript言語そのものが実はオブジェクトの集まりなのです。オブジェクトについては本書の後半で詳しく説明しますので、いまはメソッドは関数のようなもの、プロパティは変数のようなものと考えておけば十分です。

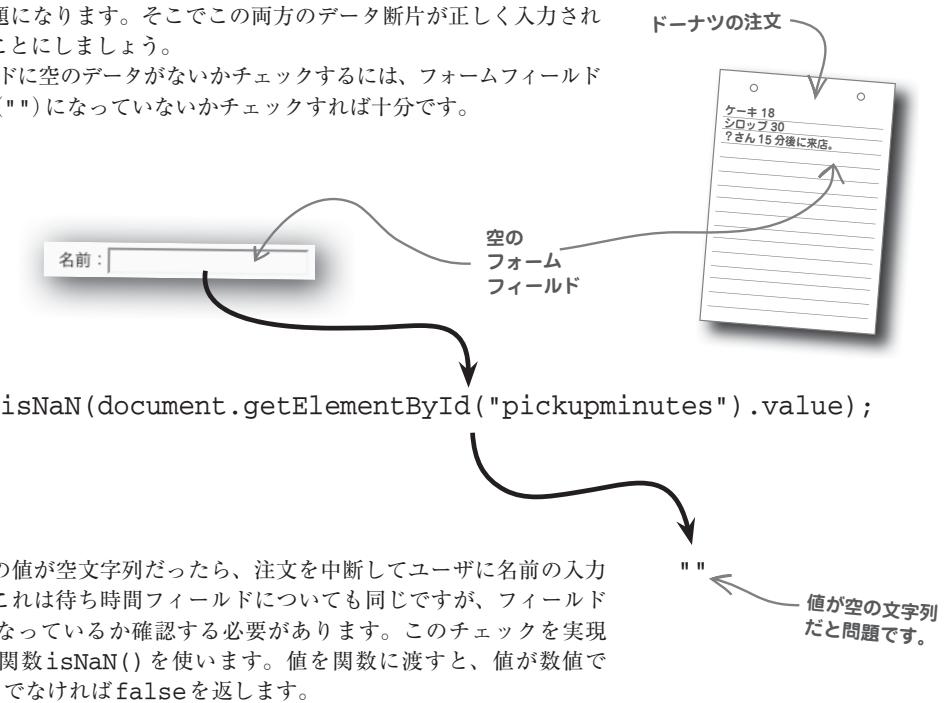


このコードを使ってダンカンのフォームデータをチェックしてみましょう。
注文を受け取る前にフィールドが空になっていないか確認します。

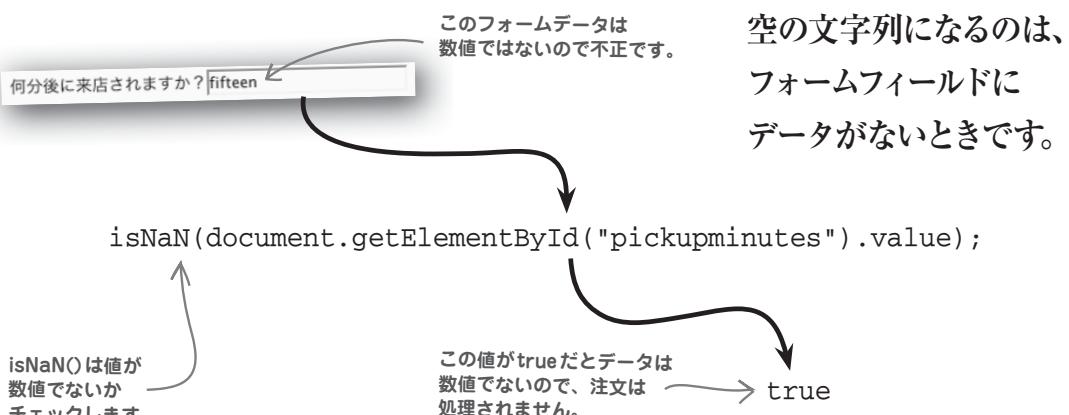
ウェブフォームのデータを検証する

ドーナツのフォームに名前が入力されているかチェックする必要があります。ドーナツをジャストインタイムでお届けするのが売りなので、待ち時間が入力されていないのも問題になります。そこでこの両方のデータ断片が正しく入力されているか確認することにしましょう。

フォームフィールドに空のデータがないかチェックするには、フォームフィールドの値が空の文字列 (" ") になっていないかチェックすれば十分です。



名前フィールドの値が空文字列だったら、注文を中断してユーザに名前の入力を促すべきです。これは待ち時間フィールドについても同じですが、フィールドのデータが数値になっているか確認する必要があります。このチェックを実現するには組み込み関数 `isNaN()` を使います。値を関数に渡すと、値が数値であれば `true`、そうでなければ `false` を返します。





JavaScriptマグネット

`placeOrder()`は名前と待ち時間のデータ検証を行います。マグネットを使ってコードを完成してください。名前と待ち時間のデータが存在するか確認し、さらに待ち時間が数値で入力されているかも確認します。同じマグネットを何回使ってもかまいません。

“if”は条件のテストに使われ、結果に応じてアクションが実行されます。

これは2つの値が等しいかテストしています。

これは2つの条件、これがまたは(OR)あればアクションを実行します。

```
function placeOrder() {
  if ( ..... == ..... ) {
    alert("申し訳ありませんが、注文を送信する前に名前を入力してください");
  } else if ( ..... == ..... || ..... ) {
    alert("申し訳ありませんが、注文を送信する前に何分後に来店されるかを入力してください");
  } else
    // サーバに注文を送信
    form.submit();
}
```

pickupminutes

name

.

""

(

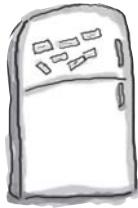
document

)

isNaN

value

getElementById



JavaScriptマグネットの答え

`placeOrder()`は名前と待ち時間のデータ検証を行います。マグネットを使ってコードを完成してください。名前と待ち時間のデータが存在するか確認し、さらに待ち時間が数値で入力されているかも確認します。同じマグネットを何回使ってもかまいません。

これは名前の値が空だったら
アラートを表示し、そうで
なければ別の処理を
行います。

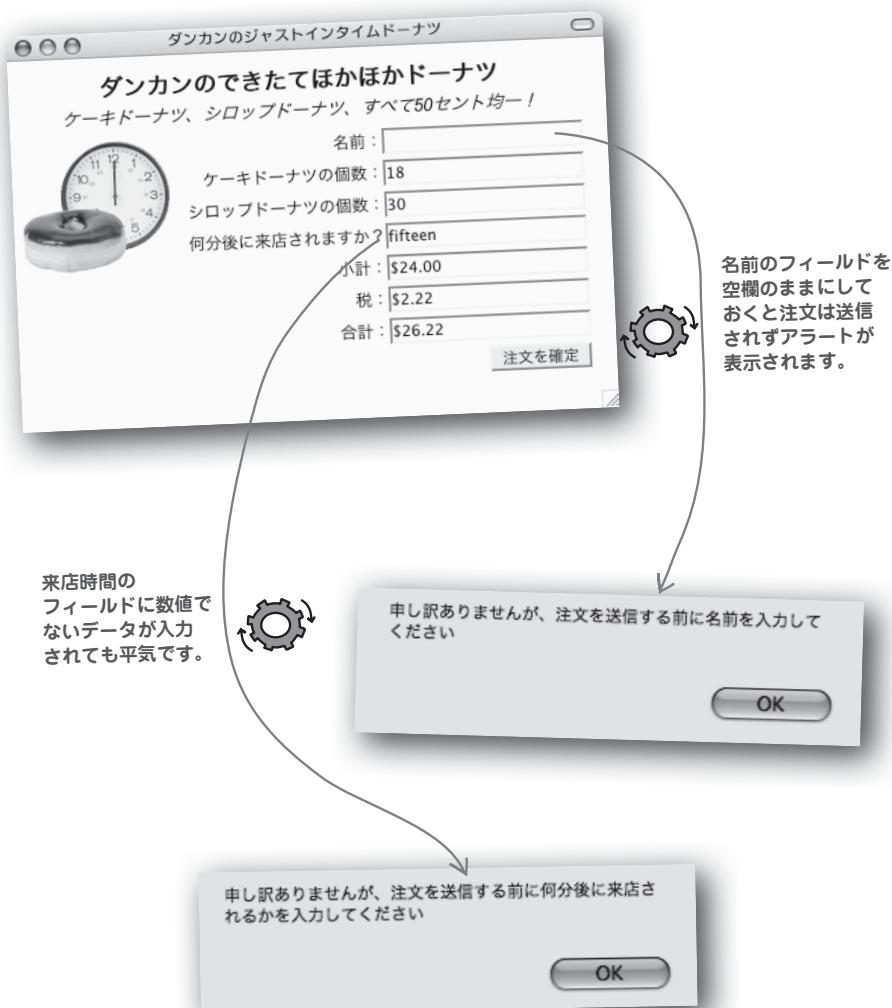
名前フィールドの値が
""と等しいかチェックします。

値が空であるか、または
値が数値でなければ、
という意味です。

```
function placeOrder() {
  if (document.getElementById("name").value == "") {
    alert("申し訳ありませんが、注文を送信する前に名前を入力してください");
  } else if (document.getElementById("pickupminutes").value == "" ||
    isNaN(document.getElementById("pickupminutes").value)) {
    alert("申し訳ありませんが、注文を送信する前に何分後に来店されるかを入力してください");
  } else
    // サーバに注文を送信
    form.submit();
}
```

ダンカンドーナツがまたも改善されました！

ジャストインタイムドーナツのフォームはデータ検証を備えて改善されたので、フランキーの営業妨害に終止符がうたれました。ユーザが入力したデータはJavaScriptを使って整合性が保証されるので、競争の激しい朝食ビジネスでは特に有益です。



素朴な疑問に答えます

Q: プラス記号(+)が加算なのか連結なのか、どうしてわかるのですか？

A: JavaScriptでは文脈によって機能が決まることが少なくありません。プラス記号は項を2つとりますが、それらを数値として加算するのか、それともテキストとして連結するのかは、項のデータ型によって決まります。項のデータ型が想定したものと実際のものとで違う場合、問題が起きることがあります。このため数値の加算を行うときは、それが数値であるか確認し、テキストの連結を行うときは、それがテキストであるか確認した方がいいのです。

Q: 数値に文字列を加算しようとしたら、どうなりますか？

A: JavaScriptでは、数値と文字列を加算しようとすると、数値から文字列への変換が自動的に実行され文字列の連結になるので、結果は常に文字列になります。文字列を数値として加算したいときは、`parseInt()`や`parseFloat()`を使って明示的に文字列を数値に変換する必要があります。

Q: 10進数の数値を含む文字列を`parseInt()`で変換するとどうなりますか？

A: 心配ありません。`parseInt()`が使われた場合、JavaScriptは小数部を無視して、整数部だけを返します。

Q: HTMLのid属性はどのようにしてウェブ要素をJavaScriptコードと結びつけるのですか？

A: JavaScriptコードがHTMLコンテンツにアクセスする際、`id`属性が窓口になると考えてください。JavaScriptがウェブページで実行される

とき、文字通りウェブページそのものではなく、実際にはブラウザで実行されています。JavaScriptコードはHTMLコードから隔離されているので、特別な機構を通さないとアクセスできません。こうした機構のひとつが`id`属性です。JavaScriptがHTML要素を取り出すのを可能にします。ウェブ要素にIDを付加すると、JavaScriptコードから要素が探せるようになります。スクリプトの可能性が拡大します。

Q: 漠然としていて、よくわかりません。JavaScriptコードからHTML要素にアクセスするには、具体的どうすればいいのですか？



A: `document`オブジェクトの`getElementById()`を使ってJavaScriptからHTML要素にアクセスすることができます。このメソッドは要素の`id`属性を使ってページから要素を探します。HTMLのIDはJavaScriptの識別子のようなもので、ページの中で一意でなければなりません。そうでないと、`getElementById()`はどのウェブ要素を返せばいいのかわからなくなります。

Q: 9章で説明されるとのことですが、すでにオブジェクトは何度か登場しています。オブジェクトとは何ですか？

A: すこし先走り過ぎたようですね。オブジェクトはJavaScript

の高度なデータ型です。関数と定数と変数を論理的にひとつにまとめたものです。メソッドはオブジェクトの中の関数、プロパティはオブジェクトの中の変数や定数になります。JavaScriptはオブジェクトを使ってあらゆるものを表現します。ウェブページの文書がオブジェクトであるのと同様に、ブラウザのウィンドウもオブジェクトです。`getElementById()`は`document`オブジェクトから呼び出される必要があるのはこのためです。このメソッドは、ウェブページ全体を表現するオブジェクトの一部なのです。説明はこれくらいにして、2章に戻りましょう。

Q: ウェブページの要素とその値の違いがまだよくわかりません。違いは何ですか？

A: ウェブページの要素はJavaScriptにはオブジェクトとして見えます。つまり、そのオブジェクトにはプロパティやメソッドがあり、それを使ってオブジェクトを操作することができます。プロパティのひとつである`value`には、要素の値が格納されています。たとえば、フォームフィールドの値はそのフィールドに入力されたデータになります。

Q: 値が数値でないか調べる必要があるのはなぜですか？ 数値であるか調べる方がよくありませんか？

A: いい質問です。値が数値であるかどうかをなぜ気にするのか、これが問題の本質です。多くの場合、数値を扱う前提であれば、(予期せぬ)例外を検査する意味があります。そのため`NaN`かどうかをチェックすれば、数値を扱うスクリプトコードがもっと堅牢になり、数値でなかったときの予期せぬ計算結果を減らせるでしょう。

直感的なユーザ入力を追求する

もはや火の粉を払う苦労もなくなり、ダンカンはジャストインタイムドーナツのフォームの使い勝手を改善したいと思っています。店頭にある「Hot Donuts」の看板が**直感的**なのと同じように、オンラインのフォームも直感的にしたいと思っています。ドーナツは通常はダース単位で注文されます。12個や24個で注文する人はごくわずかで、通常は1ダースや2ダースで注文されるわけです。ダンカンはドーナツフォームでユーザがもっとできるだけ自然にデータを入力できるようにしたいと考えています。

現在のスクリプトではユーザがドーナツの数量に「ダース」という語を入力することを想定していないので問題があります。

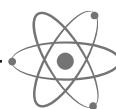
The form has the following fields:

- 名前 : Dierdre
- ケーキドーナツの個数 : 3ダース
- シロップドーナツの個数 :
- 何分後に来店されますか? : 60
- 小計 : \$1.50
- 税 : \$0.14
- 合計 : \$1.64

A callout points to the input field for 'ケーキドーナツの個数' with the text: "'3 ダース'" は parseInt() のおかげで 数値の 3 に 変換されます。

Below the diagram, there's an illustration of three donuts. One is labeled '3' above it, and another is labeled 'これは文字列ではなく数値です。' (This is a string, not a number).

このスクリプトはユーザが数値と一緒に語「ダース」を入力しても文句を言いません。parseInt() は文字列の中の数値の後にあるテキストを無視します。そのため語「ダース」はただ破棄され、数値だけが取り出されます。



頭の体操

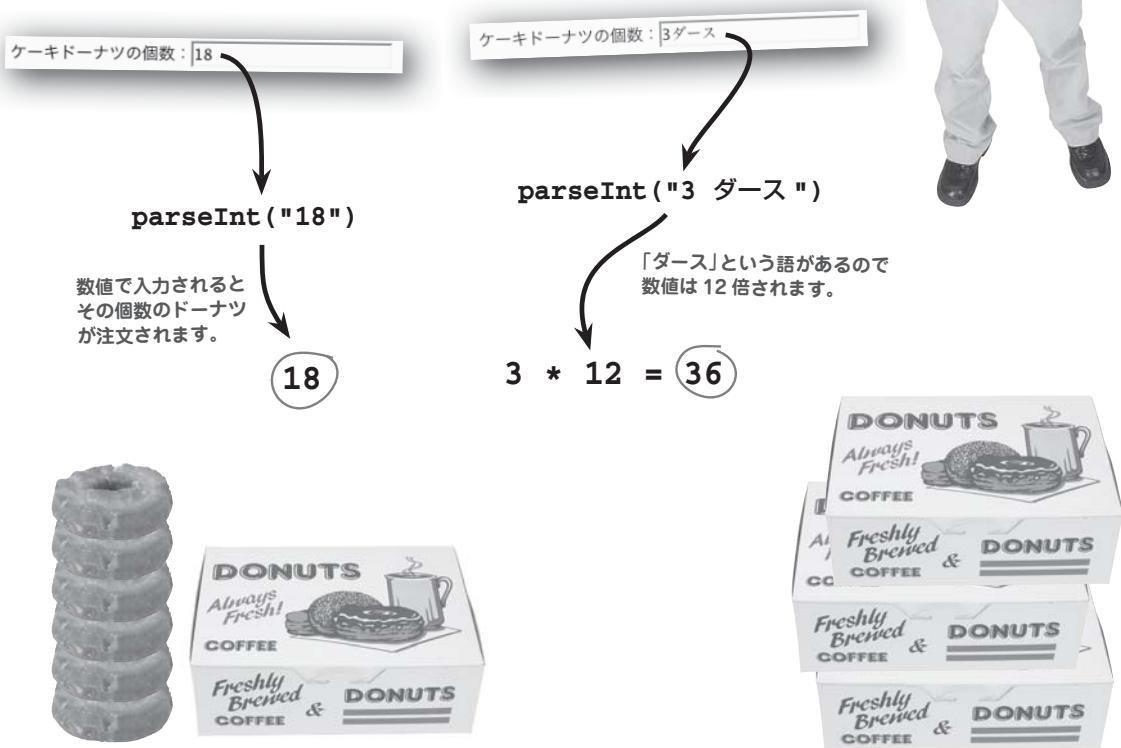
ドーナツスクリプトでユーザがダースで注文するとき、数値だけ入力するのも、数値と語「ダース」を入力するのも、どちらも許可するには、どうしたらしいでしょうか？

ユーザが入力したテキストに
「ダース」という語があるか
探せるかなあ？



ユーザがダースで注文したいときは 12 倍します。

ダースで注文できるオプションを実現するには、小計を計算する前にユーザの入力に「ダース」という語があるかチェックする機能をドーナツスクリプトに追加します。語「ダース」があれば 12 倍し、そうでなければ数値をそのまま使います。





後は焼くだけ JavaScript

カスタム関数parseDonuts()はドーナツの数量の入力データを処理します。データを数値に変換し、入力データに「ダース」という語があるかチェックします。「ダース」があればドーナツの数量を12倍します。

```
function parseDonuts(donutString) {
    numDonuts = parseInt(donutString);
    if (donutString.indexOf("ダース") != -1)
        numDonuts *= 12;
    return numDonuts;
}
```

ドーナツの個数を12倍しています。

入力データに「ダース」という語があるかチェックしています。

ダース単位のドーナツを処理する

parseDonuts()はupdateOrder()から呼び出されます。
updateOrder()はユーザが入力したデータから小計と合計を計算します。

```
function updateOrder() {
    const TAXRATE = 0.0925;
    const DONUTPRICE = 0.50;
    var numCakeDonuts = parseDonuts(document.getElementById("cakedonuts").value);
    var numGlazedDonuts = parseDonuts(document.getElementById("glazeddonuts").value);
    if (isNaN(numCakeDonuts))
        numCakeDonuts = 0;
    if (isNaN(numGlazedDonuts))
        numGlazedDonuts = 0;
    var subTotal = (numCakeDonuts + numGlazedDonuts) * DONUTPRICE;
    var tax = subTotal * TAXRATE;
    var total = subTotal + tax;
    document.getElementById("subtotal").value = "$" + subTotal.toFixed(2);
    document.getElementById("tax").value = "$" + tax.toFixed(2);
    document.getElementById("total").value = "$" + total.toFixed(2);
}
```

2つの定数を初期化しています。

フォームフィールドからドーナツの個数を取得しています。

ページに合計をドル単位で表示します。

合計を小数点以下2桁で四捨五入しています。

ジャストインタイムドーナツは大当たり！

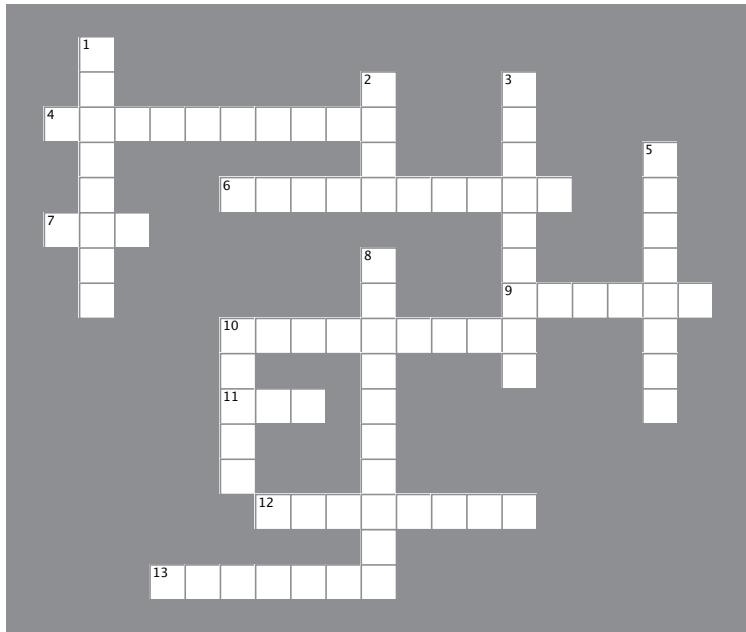
ダンカンはご機嫌です。ジャストインタイムドーナツはJavaScriptで強化されたページによって完全に実現されました。ユーザが入力した注文は注意深く検証されています。





JavaScriptクロスワード

JavaScriptコードにいつもデータが格納されているわけではありません。クロスワードパズルに格納されていることもあります。あなたが見つけてくれるのを待っています。



ヨコのカギ

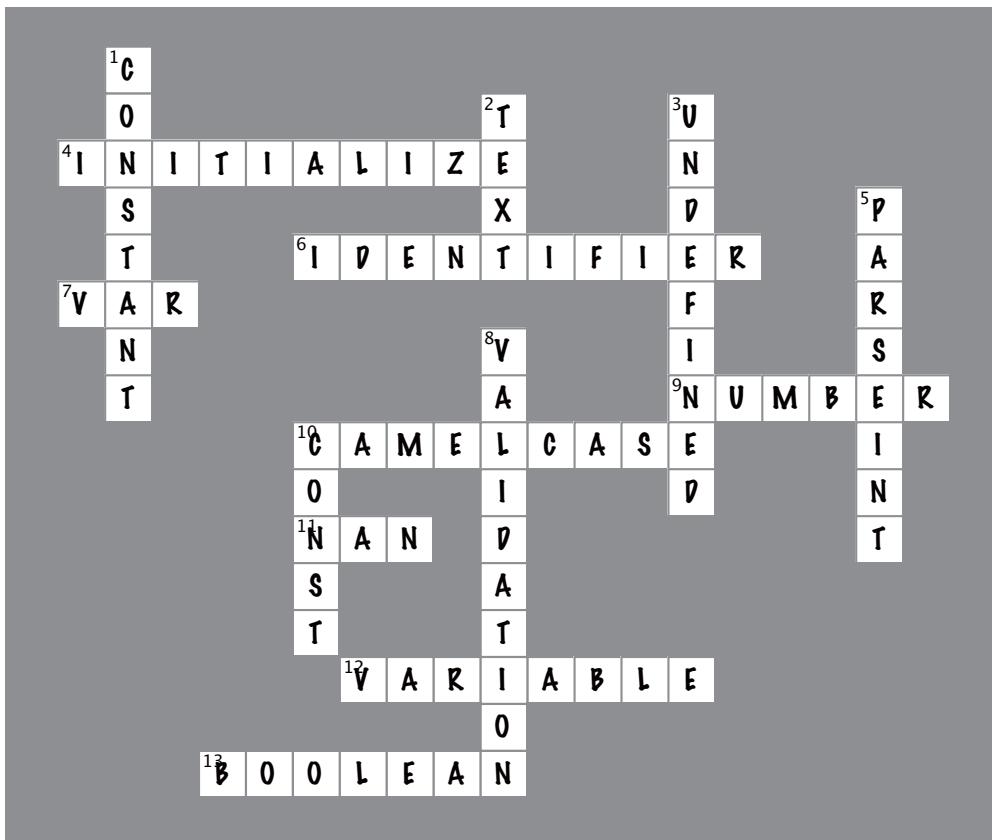
4. データ断片を作成するとき、その値を設定することをといいます。
6. データ断片を参照するときに使う一意の名前。
7. 変数を作成するときに使うJavaScriptのキーワード。
9. 3.14、11、5280 はすべてこのデータ型です。
10. ThisIsMyNameのように大文字と小文字が混在した識別子の名前に使うコーディング慣習です。
11. 数値ではないを示します。
12. 値を変更できる情報断片です。
13. このデータ型で格納されたデータ断片は値が on/off になります。

タテのカギ

1. このデータ断片の値は変更できません。
2. 文字、語、句などを格納するのに使うデータ型です。
3. 変数や定数で値が設定されなかった場合、データはと見なされます。
5. 文字列から数値に変換するのに使うJavaScriptの組み込み関数。
8. ユーザが入力したデータが正しいことをチェックする処理をといいます。
10. 定数を作成するときに使うJavaScriptのキーワード。



JavaScript クロスワードの答え



折り畳みページ

脳のところで
折り畳んでから
謎を解決しましょう。

スクリプトのデータに
何を求めていませんか？

左脳と右脳がご対面！



ユーザが入力したデータというのは、
そのまま信じてはいけない種類のデータです。ユーザがデータを
入力するとき、その内容を確認して入力していると前提するのは危険です。
JavaScriptを使うと、より安全なソリューションになります。

3章 ブラウザを調べる

* ブラウザを探検する *

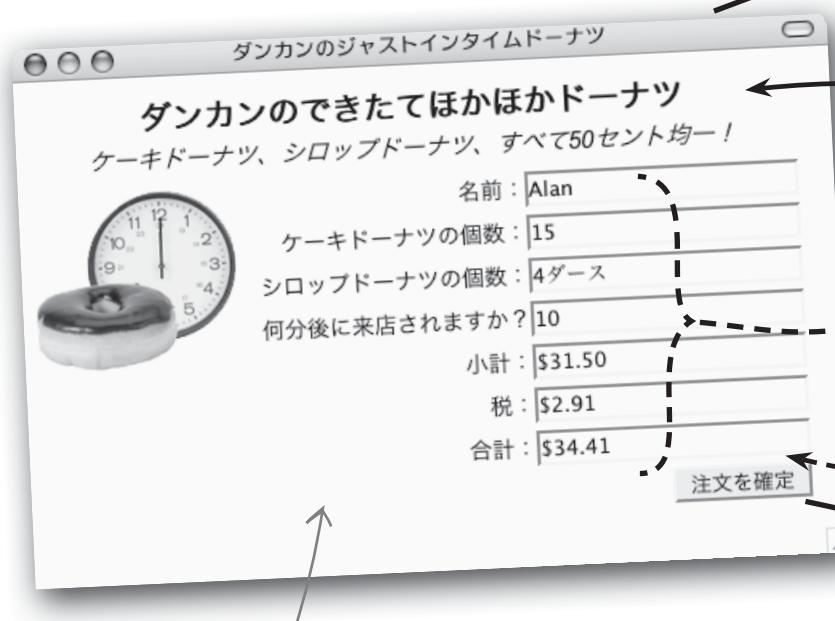


ときどき JavaScript は自分のまわりで何が起きているか知る必要があります。あなたのスクリプトはウェブページのコードとして開始されるかもしれません、最終的にはクライアントであるブラウザによって作成された世界で動作します。賢いスクリプトは自分が実行される環境について知る必要があるため、ブラウザとやり取りをして環境について情報を得ます。画面の大きさを調べたりブラウザのボタンにアクセスしたりといったブラウザとの関係を通して、スクリプトはおそらくたくさんのことを行えます。

クライアント、サーバ、JavaScript

ウェブブラウザでハイパーリンクをクリックしたりURLを入力すると、ブラウザはウェブサーバにページを要求します。サーバはウェブクライアントであるブラウザにページを配信します。ブラウザがページを表示する直前までJavaScriptは処理されません。ページの中のJavaScriptコードはウェブブラウザと力をあわせてユーザーの操作に応答したり、必要に応じてページを変更したりします。JavaScriptコードを実行するウェブブラウザの部分はJavaScriptインタープリタと呼ばれます。

① ブラウザはサーバにページをリクエストします。



③ ブラウザはページを表示します。

④ ユーザがデータを入力したとき、JavaScriptはフォームフィールドを検証します。

JavaScriptコードは
サーバには頼らず
すべてクライアント
実行されます。



⑤ この注文は...

ページがブラウザに配信されるとサーバはこの方程式からほとんど外れます。これ以降JavaScriptが実行するすべての処理はブラウザの枠の中で行われます。サーバが処理を行いデータを返すまで待つ必要がないので、ページの応答が機敏になります。JavaScriptがクライアント言語として知られているのはこのためです。

ページをリクエスト

```
GET / HTTP/1.1
Host: www.duncansdonuts.com
Connection: close
Accept-Encoding: gzip
Accept: image/gif, image/x-bitmap, image/jpeg, image/pjpeg, ...
Accept-Language: en-us
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X; en-US; rv:1.8.1.7) ...
```

HTMLはページの
内容を記述。

サーバのページ

```
<html>
<head>
  <title>ダンカンのジャストインタイムドーナツ </title>
  <link rel="stylesheet" type="text/css" href="donuts.css" />
  <script type="text/javascript">
    function updateOrder() {
      ...
    }

    function parseDonuts(donutString) {
      ...
    }

    function placeOrder() {
      ...
    }
  </script>
</head>

<body>
  <div id="frame">
    <div class="heading">ダンカンのできたてほかほかドーナツ </div>
    <div class="subheading">ケーキドーナツ、シロップドーナツ、すべて50セント均一！ </div>
    <div id="left">
      
    <div id="right">
      ...
    </div>
  </div>
</body>
</html>
```

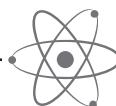
⑤ サーバに ...

CSSでページを綺麗に。

JavaScriptでインタラクティブな
ページに。ここではユーザが
入力したデータを検証。

② サーバは
ページを返します。

⑤ ...送信されます。



頭の体操

サーバではなくクライアントで実行する方が意味のあるタスクは他にありますか？

ブラウザはあなたのために何ができる？

クライアントであるウェブブラウザはJavaScriptコードを実行する責任があり、スクリプトがクライアント環境にアクセスするのを許可します。たとえばスクリプトからブラウザウィンドウの幅と高さやウェブページのアクセス履歴を取得することができます。JavaScriptに開放されているブラウザ機能には目覚まし時計のように働くタイミングの機構やクッキーへのアクセスがあります。クッキーを使ってデータを格納しておくと、ページから離れたりブラウザを閉じた後でもそのデータを復活できます。

ブラウザの計測

ブラウザウィンドウや表示されるウェブページの大きさに関する情報だけでなく、ブラウザのベンダーやバージョン番号に関する情報も含まれます。



ブラウザの履歴

ブラウザ履歴は最近訪問したページのリストです。JavaScriptを使ってこの履歴リストにアクセスし、その中のページにブラウザを飛ばすこともできます。ブラウザのナビゲーションを効率よく制御することができるのです。



クッキー

クッキーはブラウザによってユーザのハードディスクに格納される変数のようなものです。ページを離れてウェブセッションが終了した後もデータが残るので、ページに戻ったときはデータが復活します。



タイマー

タイマーを使うとJavaScriptコードの断片がある一定時間経過後に起動することができます。

こうした機能をすべてのクライアントがスクリプトに提供しているわけではありませんが、ページの中にある機能よりもっと多くの機能をJavaScriptによって実現できるわけです。実際、**ページの枠を越えて** ブラウザからちょっとした力を借りるのが便利な状況はたくさんあります。

素朴な疑問に

Q: JavaScriptはクライアントの一部なんですか？

A: はい。JavaScriptをサポートするウェブブラウザにはJavaScriptインターフェリタが内蔵されていて、ページからJavaScriptコードを読み取り、コードを実行します。

Q: JavaScriptコードはクライアント上で実行されるのであれば、サーバとはどういう関係なのでですか？

答えます

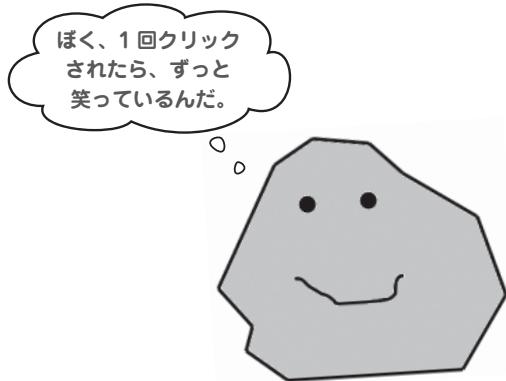
A: JavaScriptコードはクライアント上で実行されるので、ウェブサーバと直接の関係はありません。JavaScriptはサーバからブラウザに配達されるウェブデータを途中で捕捉するときによく使われます。サーバから情報を取得し、それを処理してページに情報を表示する、そのようなスクリプトを書くこともできます。こうしたAjaxと呼ばれるスクリプト手法の使い方については12章で説明します。

Q: JavaScriptを使ってクライアントを制御できますか？

A: イエスでもあります。ウェブブラウザはJavaScriptがクライアント環境の一部にアクセスするのを許可していますが、セキュリティの理由から全面的な自由を与えるわけではありません。たとえば、ほとんどのブラウザはユーザの了承なしにウインドウを開いたり閉じたりすることを許していません。

iRockは喜びすぎ

iRockを憶えますか？JavaScriptで成功したので、若い起業家のアランが買い取りましたが、いくつか問題があるので、再びあなたに声がかかりました。iRockがいつも喜んだ表情のまま変わらないので、ユーザはイライラしているのです。誰でもペットには喜んでいてほしいのですが、iRockの感情の幅はあまりに狭すぎるようです。



いつもニコニコしているペットを追求してたんだけど、ユーザはもうすこしリアルなのが好きなんだね。となると、きみにコードを見直してもらう必要がありそうだね。

iRockのオーナーのアランです。
お金持ちなので、あなたの
払いも良さそうですよ。



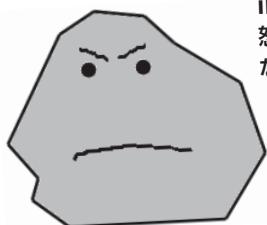
iRockの課題は**ユーザの期待**に応えることです。バーチャルペットというコンセプトは、できるだけ現実のペットに似せることです。iRockの**振る舞いを改善**して、もっとリアルにする方法を調べる必要があります。クライアントのウェブブラウザがiRockの課題を解決する鍵を握っているように思われます。

iRock はもっと反応する必要があります

iRockにどんな表情が増やせるか考えてみましょう。iRockの目標は、もっとインタラクティブにするのはもちろん、もっとリアルにしてユーザの気をひくことです。ユーザにもっと反応するようにして、反応するときのiRockの表情を増やす必要があります。

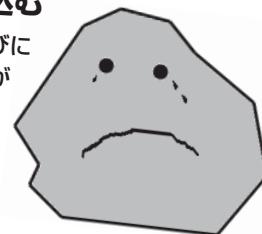
iRock 2.0 のコードは
<http://www.headfirstlabs.com/books/hfjs>
からダウンロードできます。

怒る



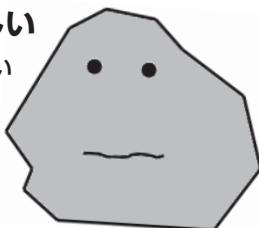
iRockはときどき理由もなく
怒ります。ユーザはiRockを
なだめる必要があります。

落ち込む

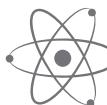


iRockはページが閉じられるたびに
泣き出します。ユーザはiRockが
落ち込まないようにブラウザを
開いたままにしておく必要が
あります。

寂しい



iRockは放っておかれる寂しい
表情になります。ユーザは
定期的にクリックして
iRockに気遣う必要があります。

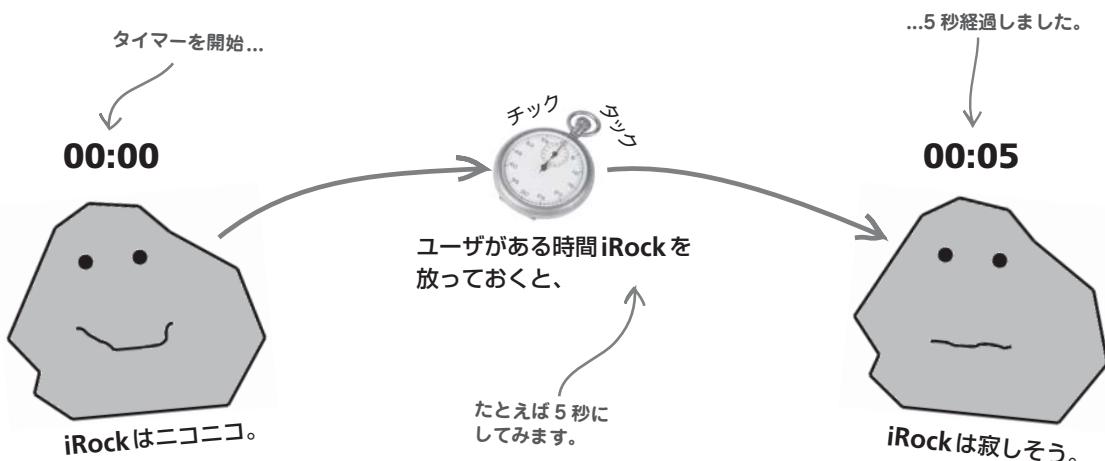


頭の体操

これらの表情のうちiRockにふさわしいのはどれでしょう？ iRockスクリプトにこれらの表情を実装するにはJavaScriptをどう使えばいいでしょう？

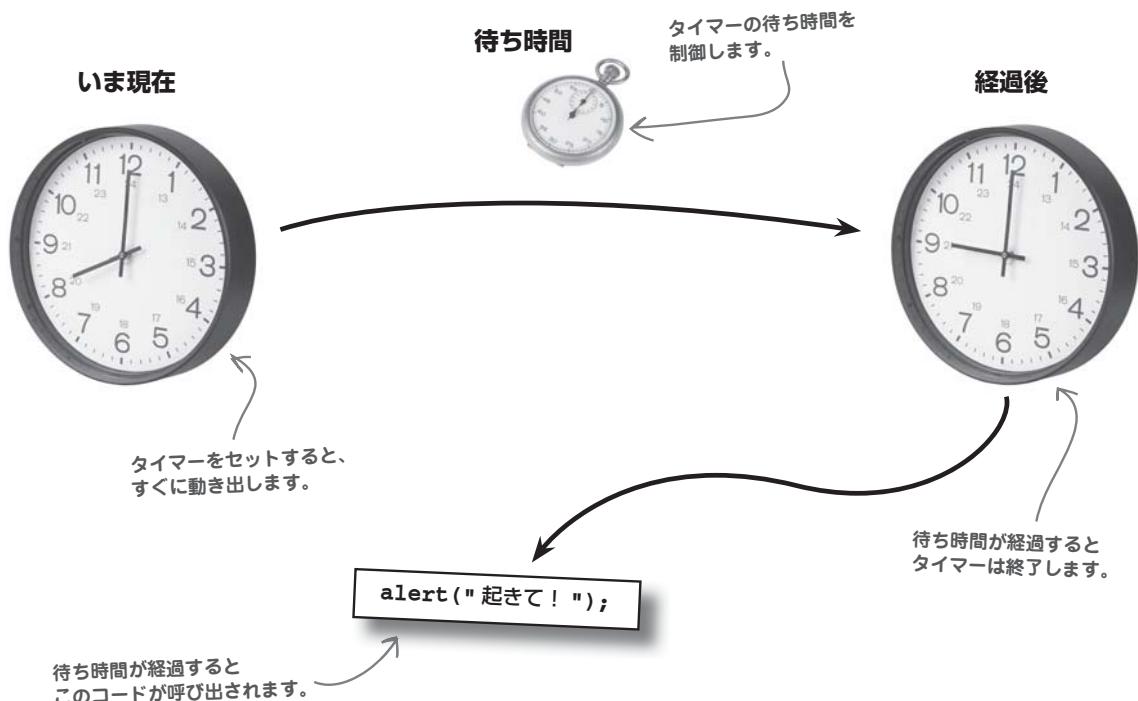
JavaScriptを使うとユーザがいつ操作したのか、いつ操作しなかったのかがわかります。

iRockを寂しがらせるのは面白いアイデアです。
ユーザは負い目を感じないようにiRockを定期的に
かまってやると、iRockはそれに反応して
明るい表情になるからです。



タイマーはアクションと経過時間を結びつけます

JavaScriptを使ってタイマーを設定できます。JavaScriptのタイマーは目覚まし時計のように動作します。待ち時間を指定すれば、その時間が経過したところでコードの断片が実行されます。ただし、目覚まし時計はある時刻になるとアクションが実行されますが、JavaScriptのタイマーはある時間が経過した後にアクションが実行されるところが異なります。つまり時刻を指定するかわりに経過時間を指定する必要があるので、この違いは問題ではありません。



JavaScript タイマーのいいところは、時間が経過した後に好きなコードを実行させられることです。タイマーの使い方はさまざまで、一定時間が経過するたびに定期的にデータを書き換えるウェブページもありますし、ユーザが一定時間ページを操作しなかったことを検出するページもあります。

タイマーを使うと
ある一定時間が
経過した後で
JavaScriptのコードを
実行させることができます。

タイマーを分解する

JavaScriptでタイマーを設定するときは、1) 経過時間を設定し、2) 時間が経過した後で実行するコードをタイマーに知らせる、この2つが鍵になります。タイマーは設定されたらすぐに時間を刻み始めます。



経過時間はミリ秒で設定します。1秒の1000分の1が1ミリ秒です。指定したい秒数を1000倍すればいいので、たとえば2秒は2000ミリ秒になります。

メッセージを表示

```
alert("起きて！");
```

指定した時間が経過した後タイマーに起動させる
JavaScriptコードは自由に設定できます。単文でも
複文(文をセミコロンで区切れます)でも、

組み込み関数でも
カスタム関数でもかまいません。

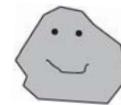
```
refresh(); setTimeout(refresh, 120000);
```

ページを更新。

次のタイマーを設定。

通常のJavaScriptタイマーは時間が経過してタイマーのコードを実行すると、タイマーの処理は終わりです。このタイプのタイマーはコード断片を1回しか起動しないので、1回かぎりのタイマーです。これとは別にインターバルタイマーも作成できます。インターバルタイマーは指定した時間間隔ごとに繰り返し何回もコードを起動します。あなたがそれを止めるまで処理は繰り返し実行されます。iRockのタイマーに適しているのは1回かぎりのタイマーですが、インターバルタイマーにもそれに適した作業はあります。

タイマーの
時間が過ぎると、
ぼくは寂しくなって、
それで終わり。



エクササイズ

ミリ秒で示した時間を右の時間と対応づけてください。

500ミリ秒

5分

300,000ミリ秒

5秒

5,000ミリ秒

1/2秒





エクササイズの 答え

ミリ秒で示した時間を右の時間と対応づけてください。



setTimeout()でタイマーを設定する

JavaScriptの組み込み関数 `setTimeout()` は 1 回かぎりのタイマーを実現します。経過時間と経過後に実行させるコードを指定します。引数の順番は以下の通りです。

`setTimeout()` は 1 回限りの
タイマーを設定します。

待ち時間 600,000 ミリ秒は
600 秒、つまり 10 分です。

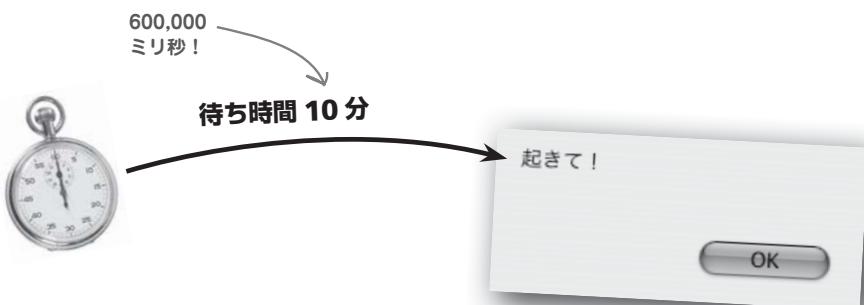
```
setTimeout("alert('起きて！');", 600000);
```

setTimeout()に
JavaScriptコードが
テキスト文字列で渡されます。
このため二重引用符で囲む
必要があります。

JavaScriptの数値には
たとえ大きな数であっても
カンマは書かないでください。

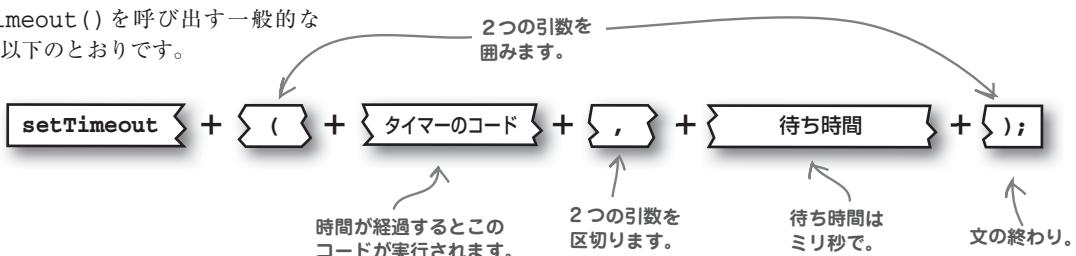
timerが終了すると
アラートボックスが
表示されます。

この `setTimeout()` を呼び出すと 10 分後にアラート
ボックスを表示するタイマーを作れます。



setTimeout() の詳細

setTimeout() を呼び出す一般的な形式は以下のとおりです。



インターバルタイマーの設定方法は1回かぎりのタイマーと似ていますが、`setTimeout()`のかわりに `setInterval()` を使います。インターバルタイマーを設定すると、指定した時間間隔おきにコードが何回も起動されます。

```

var timerID = setInterval("alert('起きて！');", 600000);
  ↑
  タイマーの
  IDを格納。
  ↓
  タイマーを設定。
  ↑
  ミリ秒で
  指定します。
  
```

インターバルタイマーは指定された時間間隔おきにタイマーコードを実行します。



要注意！

タイマーの経過時間はミリ秒で指定します。

ミリ秒は1秒の1000分の1なので、ミリ秒を使っていることを忘れるとなにかしらの瞬時に処理が起動されてしまいます。



次のページに進む
前に irock.html に
コードを追加して
試してみて
ください。

iRockの画像を5分たったら喜んだ表情から寂しい表情に変えるコードを書いてください。iRock画像要素のIDはrockImgで寂しい表情の画像はrock.pngです*。

* このファイルは <http://www.headfirstlabs.com/books/hfjs/> からダウンロードできます。

自分で考えてみよう の答え

関数のプロパティを入れ子にするには、二重引用符と単一引用符を使い分けます。

iRockの画像を5分たったら喜んだ表情から寂しい表情に変えるコードを書いてください。iRock画像要素のIDはrockImgで寂しい表情の画像はrock.pngです。

5分をミリ秒にするには、まず60倍して秒にして、1000倍してみミリ秒にします。

```
setTimeout("document.getElementById('rockImg').src = 'rock.png';",  
5 * 60 * 1000);
```

画像要素のsrc属性を変更するとiRockの画像が変わります。

画像要素のID。

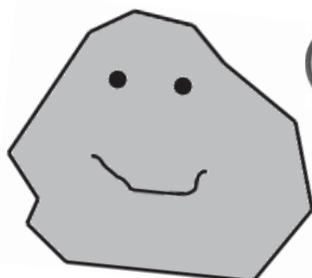
寂しそうなiRockの画像。

iRockが寂しがるようになりました！

上に示したコードを irock.html に反映して iRock を動かしてみましょう。iRock は 5 分間（クリックされずに）放ったらかしにされると寂しい表情をみせます。この間隔だと iRock は寂しがりすぎですが、ユーザの気をひくにはちょうどいいでしょう。甘えたがりくらいたがペットとしてはちょうどいいのです。



タイマーはiRockの「ご機嫌タイム」をカウントダウンします。

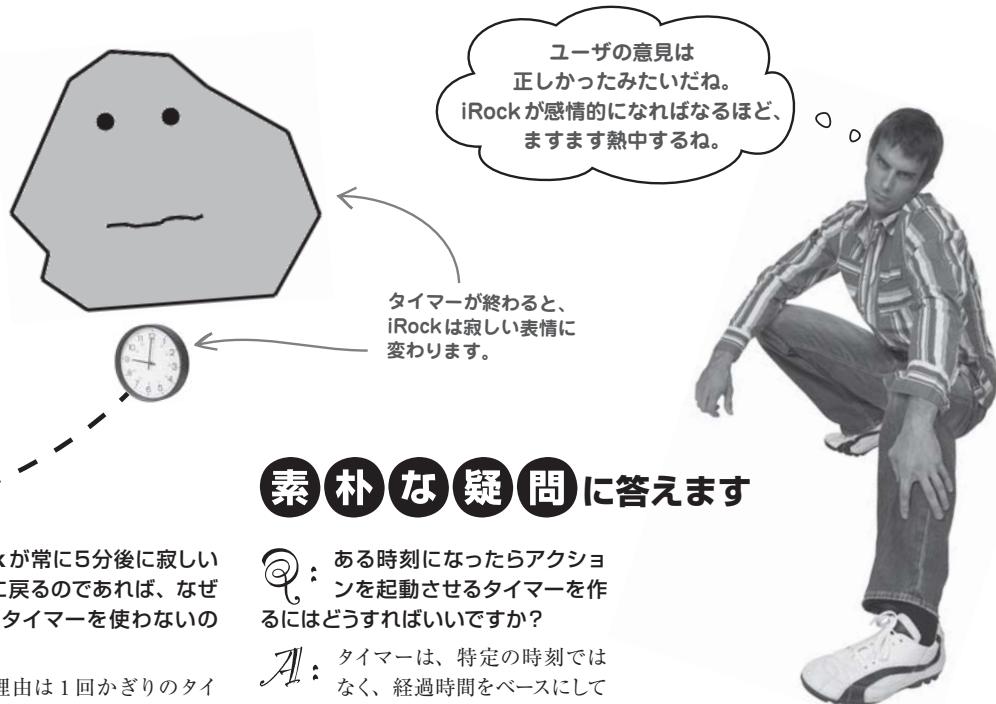


iRockの笑顔はいつまでも続くわけではなので、よりリアルになります。



マニア向け情報

setTimeout() を呼び出すとき経過時間を小さくすると iRock の表情が変わるまでの時間が短くなります。テストのときはあまり待っていられないで、時間を短くしておくといいでしょう。



素朴な疑問に答えます

Q: iRockが常に5分後に寂しい表情に戻るのであれば、なぜインターバルタイマーを使わないのですか？

A: その理由は1回かぎりのタイマーの使い方と関係しています。iRockは定期的に寂しい表情になりますが、その前に楽しそうな表情がしばらく続いています。最初のクリックでタイマーがセットされ、5分経過すると寂しい表情になり、もう一度クリックされるまで寂しい表情が続きます。これはインターバルタイマーの役割とは違っています。インターバルタイマーはユーザのアクションに関係なく5分毎にアクションを実行するきっかけを発生させます。

Q: タイマーが指定時間を経過する前にユーザがブラウザを閉じたらどうなりますか？

A: 何も起きません。JavaScriptインターフェリタはブラウザが閉じられるときシャットダウンされるので、タイマーの状態に関係なくJavaScriptコードは停止します。

Q: ある時刻になったらアクションを起動させるタイマーを作るのはどうすればいいですか？

A: タイマーは、特定の時刻ではなく、経過時間をベースにしているので、時刻を経過時間に変換する必要があります。この経過時間はターゲットとなる時刻から現在の時刻を引き算すれば得られます。この計算にはJavaScriptのDateオブジェクトの助けが必要です。Dateオブジェクトについては9章で説明します

Q: 変化するデータのあるページで、15分ごとにページを更新したいのですが、どうすればいいですか？

A: setInterval()を使ってインターバルタイマーを15分に設定します。900,000ミリ秒($15 \times 60 \times 1000$)です。ページを更新するタイマーのコードが必要なので、location.reload();と書いて、locationオブジェクトのreload()メソッドを呼びます。これでタイマーは15分ごとにページ更新を起動します。もちろん、ページを更新するかわりに、Ajaxを使つ

てデータを動的に読み込ませることもできます。

Q: インターバルタイマーは何度も繰り返し続くと理解していますが、インターバルタイマーを止めるにはどうしたらいいのでしょうか？

A: clearInterval()は setInterval()で設定されたインターバルタイマーをクリアするのに使います。clearInterval()には、タイマーを作成したとき setInterval()が返すインターバルタイマーのIDを渡す必要があります。そう、関数は情報を返すことができるのです。setInterval()が返す値をたとえばtimerIDに格納したら、 clearInterval(timerID)と書いて、clearInterval()に渡すだけで、タイマーを停止できます。



ブラウザの真実

今週のインタビュー：ウェブクライアントの告白

Head First：本日はお忙しい中おいでいただきまして、ありがとうございます。

ブラウザ：ほんとに忙しいですね。HTMLとCSSだけでも手一杯で、まったく性格の違うこのふたつでページを表示するときは頭痛がするのに、さらにJavaScriptという毛色の変わった獣まで相手にする必要があるんですから。

Head First：どういう意味でしょう？ JavaScriptは野生のままで手がつけられないんですか？

ブラウザ：「獣」と言ったのは、たとえですよ。JavaScriptには他にはない独特の問題があって、それにいつも悩まされると言いたかったわけです。JavaScriptコードを読んで、コードが間違ってないことを祈りつつ、HTMLとCSSに目を光らせながら同時にそのコードを実行する、まったく新しい仕事ですからね。

Head First：わかりました。この三つはうまくやっていけるのですか？

ブラウザ：少なくともそれは私の問題ですね。この三人者は仲良くやっています。JavaScriptが間違えてHTMLコードをめちゃめちゃにしてしまうこともあります、そうなっても私にはどうすることもできません。

Head First：つまりあなたは「イエスマン」なんですね？

ブラウザ：それは一方的だなあ。何よりも一貫性を重視しているというのが実情に即していますね。嘘じゃないです、神に誓って。サーバがくれたコードを書かれたとおりに正確に実行するのが私の仕事ですから。

Head First：それが間違いだとわかっていても、ですか？

ブラウザ：コードを読むときはベストを尽くしてそうした問題を解決するようにしていますが、なかなか大変な仕事ですね。この話は11章のテーマですね。ウェブクライアントとしての私の役割をお話ししましたね。

Head First：ああ、脱線してしまいましたね。クライアントであるとは、どういう意味でしょう？

ブラウザ：ウェブページの配達チャンネルの終端に立って、リクエストしたページをサーバから受け取ることですね。

Head First：JavaScriptを使う必要のある仕事って何でしょう？

ブラウザ：たくさんありますよ。ウェブページの表示やユーザ入力の処理といった泥臭い作業を処理するとき、JavaScriptは私の隣で鼻をくんくんわせながら変更作業をしています。悪いことばかりじゃないです。私は自分ではやりたくないけどJavaScriptにぴったりな仕事はたくさんありますから。

Head First：たとえばどんな仕事ですか？

ブラウザ：そうねえ、たとえば、ユーザが画像の上にマウスを置いたり、ウィンドウの大きさを変更したとき、私は自分では特別な仕事をしないんです。私のわりにJavaScriptが仕事をしてくれます。ページの外見を変えたり、クライアントの変化に対応してコンテンツを入れ替えたりする仕事は、スクリプトには朝飯前ですから。JavaScriptコードはページごとに実行されるのが基本ですから、影響があっても特定のページやウェブサイトに留まるので問題ないんです。

Head First：JavaScriptがなんだか他人みたいな口ぶりですね。ほんとはあなたがJavaScriptなのでは？

ブラウザ：どちらもありますね。JavaScriptは私の一部でもあり、私とは別の存在でもある。というのも、JavaScriptは限られたインターフェースを通してしかクライアントである私にアクセスできません。言い換えると、JavaScriptは私のすべてに無制限にアクセスできるわけではないのです。そんなことを許すのはちょっと無責任ですよね。誰が書いたかわからないスクリプトでも実行しなければいけないのでですから。

Head First：なるほど。おかげでクライアントの仕事がはっきりわかりましたよ。

ブラウザ：お役にたてて何よりです。

画面の大きさに不満が殺到

アランがやっと iRock の表情の作り変えを終わった頃、iRock に不満な飼い主の苦情が寄せられてきました。iRock の**大きさ**が一貫していないというのです。あるユーザは iRock が縮んでしまったと報告し、別のユーザは iRock の一部が巨大になって怖いと言っています。アランが信頼を寄せるあなたの手で iRock を修正してお金を稼ぐチャンスです。

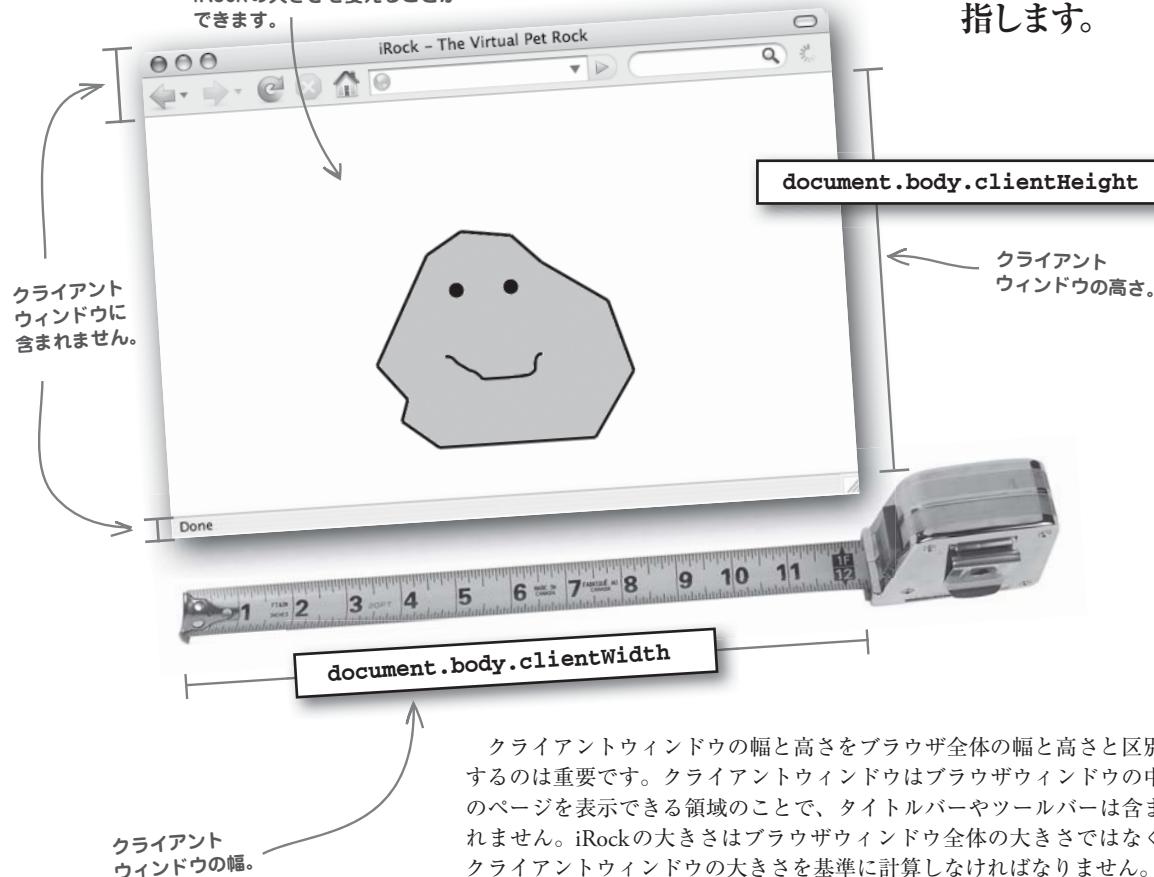


documentオブジェクトを使って クライアントウィンドウの幅を取得する

iRockの問題はブラウザウィンドウの大きさが変わってもiRockの大きさが変わらないことです。このこと自体は、ウェブをブラウズできるすべてのコンピュータ（小さな掌サイズのデバイスから巨大なモニタのデスクトップまで）のブラウザの大きさがあまりにも多様であることを考えなければ、問題ではないように思われます。必要なのはブラウザウィンドウの大きさを調べる方法です。これがわかれば、iRockの画像の大きさを変更する手段として使えます。

クライアントウィンドウにiRock
が表示されます。これを使って
iRockの大きさを変えることが
できます。

クライアント
ウィンドウはウェブ
ページを表示できる
ブラウザウィンドウの
領域だけを
指します。



documentオブジェクトのプロパティを使って クライアントウィンドウの幅を設定します

JavaScriptからdocumentオブジェクトを使ってアクセスできる領域であるクライアントウィンドウの大きさは、ウェブページと密接に関係しています。ページの要素にアクセスするためにgetElementById()を呼び出すのに使うオブジェクトもdocumentオブジェクトです。ドキュメントのプロパティであるbody.clientWidthとbody.clientHeightはクライアントウィンドウの幅と高さを保持しています。

```
<html>
<head>
<title>iRock - The Virtual Pet Rock</title>
<script type="text/javascript">
var userName;
function greetUser() {
    alert('こんにちは。私は iRock です。');
}
function touchRock() {
    if (userName) {
        alert(userName + "さん、声をかけてくれてありがとう。");
    }
    else {
        userName = prompt("あなたの名前は？");
        if (userName)
            alert("はじめまして、" + userName + "さん。");
    }
    document.getElementById("rockImg").src = "rock_happy.png";
    setTimeout("document.getElementById('rockImg').src = 'rock.png';",
               5 * 60 * 1000);
}
</script>
</head>

<body onload="greetUser();">
<div style="margin-top:100px; text-align:center">

</div>
</body>
</html>
```

document

documentオブジェクトはウェブページそのものを表現します。

document.body
documentのbodyはページの表示部分を表現します。クライアントの高さや幅などが含まれます。

素朴な疑問に答えます

Q: ウェブクライアント、ブラウザ、クライアントウィンドウ、ブラウザウィンドウ、これらの違いは何ですか？

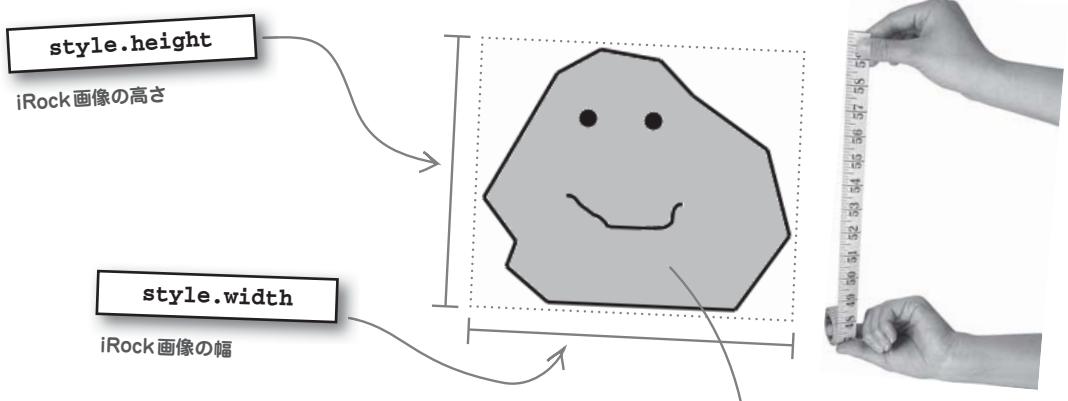
A: たしかにちょっと混乱しますよね。一般的なウェブの用語として、ブラウザはウェブページのサービスにおいてクライアント側にあるのでウェブクライアントになります。ブラウザの中では「クライアント」は別の意味になります。ページが表示されるブラウザウィンドウのある特定の領域を指します。このためクライアントウィンドウはブラウザウィンドウの中のある領域を指し、タイトルバー、ツールバー、スクロールバーなどの領域は含まれません。

Q: iRockの大きさを変更する手段としてクライアントウィンドウが使われるのはなぜですか？

A: iRockの画像の大きさを変更する手段をクライアントの領域の大きさに影響するからです。これによって、難しいアドオンツールバーの大きさや、プラットフォームやブラウザのベンダーによるブラウザウィンドウの違いなどを考慮せずにすみます。たとえば、MacのSafariのブラウザウィンドウの大きさはWindowsのFirefoxと違いますが、表示領域であるクライアントウィンドウの大きさは同じです。

iRock 画像の高さと幅を設定する

クライアントウィンドウの大きさがわかつただけでは iRock スクリプトで iRock の画像の大きさを変更できません。(CSS を使うのが嫌いだとしても) CSS の力を少し借りれば JavaScript で画像の大きさを調整できます。画像要素のプロパティである width と height を使えば、画像の最初の大きさを決められるだけでなく、必要に応じて動的に画像の大きさを変更することができます。



ウェブページのあらゆる要素に対して style オブジェクトがあるので、ページのあらゆる断片の幅と高さにアクセスできます。しかしスタイルにアクセスするには最初にウェブページの要素自体にアクセスする必要があります。この場合 iRock の画像になります。document オブジェクトの getElementById() を使えば、以下のように簡単に実現できます。

```
  
document.getElementById("rockImg").style.height
```

このコードは iRock
画像の高さに
アクセスします。

プロパティの width や height に値を設定するだけで iRock 画像の大きさを変更できます。どちらかのプロパティに値を設定すれば、画像の縦横比を保持したまま自動的に拡大縮小されます。

```
document.getElementById("rockImg").style.height = "100px";
```

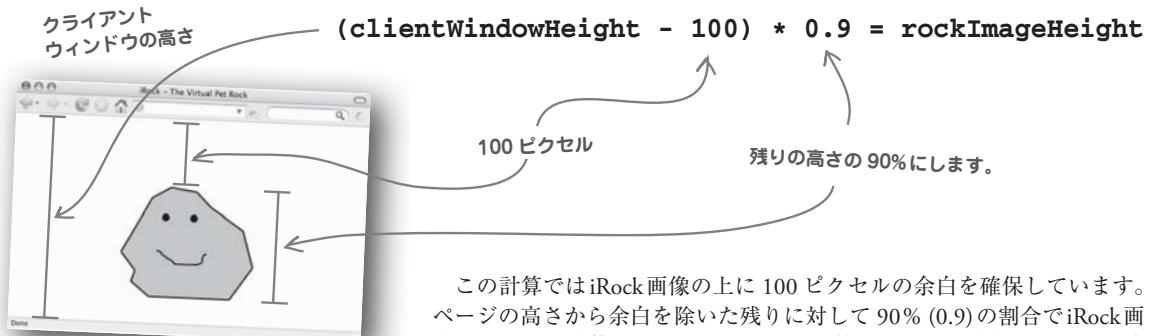
幅を設定する必要はありません。
高さにあわせて自動的に
拡大縮小されます。

iRock 画像の高さを
100 ピクセルに設定します。

ページに合わせて iRock の大きさを調整する

クライアントウィンドウの大きさをもとに iRock の画像の大きさを変更するための計算をまだ行っていません。iRock の大きさは **クライアントウィンドウの大きさに合わせて変更しなければならない**ので、クライアントウィンドウの大きさに対して iRock 画像の大きさが何パーセント占めるのか計算すればいいでしょう。

とはいって、クライアントウィンドウの幅と高さのどちらに対して iRock の大きさを比較すればいいのでしょうか？ ブラウザは表示する内容を縦方向に詰める傾向があるので、クライアントウィンドウの高さをもとに iRock 画像の大きさを比較した方が安全です。



この計算では iRock 画像の上に 100 ピクセルの余白を確保しています。ページの高さから余白を除いた残りに対して 90% (0.9) の割合で iRock 画像の大きさを計算しています。これらの値について試行錯誤して最適な大きさを見つけてください。iRock がどのように表示されるか試してみましょう。ともあれ、まずはコードを書いてください。

自分で考えてみよう

resizeRock() のコードを書いてください。クライアントウィンドウの大きさをもとに iRock の画像の大きさを変更します。また onload イベントハンドラーに resizeRock() を呼び出すコードを追加して、これに続けて greetUser() が呼び出されるようにしてください。

```
function resizeRock() {
    .....
}
...
<body onload= ...
</body>
```



`resizeRock()` のコードを書いてください。クライアントウィンドウの大きさをもとに iRock の画像の大きさを変更します。また `onload` イベントハンドラに `resizeRock()` を呼び出すコードを追加して、これに続けて `greetUser()` が呼び出されるようにしてください。

iRock 画像の大きさはクライアント
ウィンドウの高さをもとに計算されます。

```
function resizeRock() {
  document.getElementById("rockImg").style.height =
    (document.body.clientHeight - 100) * 0.9;
}
...
<body onload="resizeRock(); greetUser();"
  ...
</body>
```

iRock 画像の ID を使って
画像要素にアクセスします。

ページが最初に読み込まれたとき、
2つの関数が呼び出されます。
イベントには複数のコード断片を
結びつけることができます。

iRock の上にある余白を
考慮して 100 ピクセルだけ
引き算します。

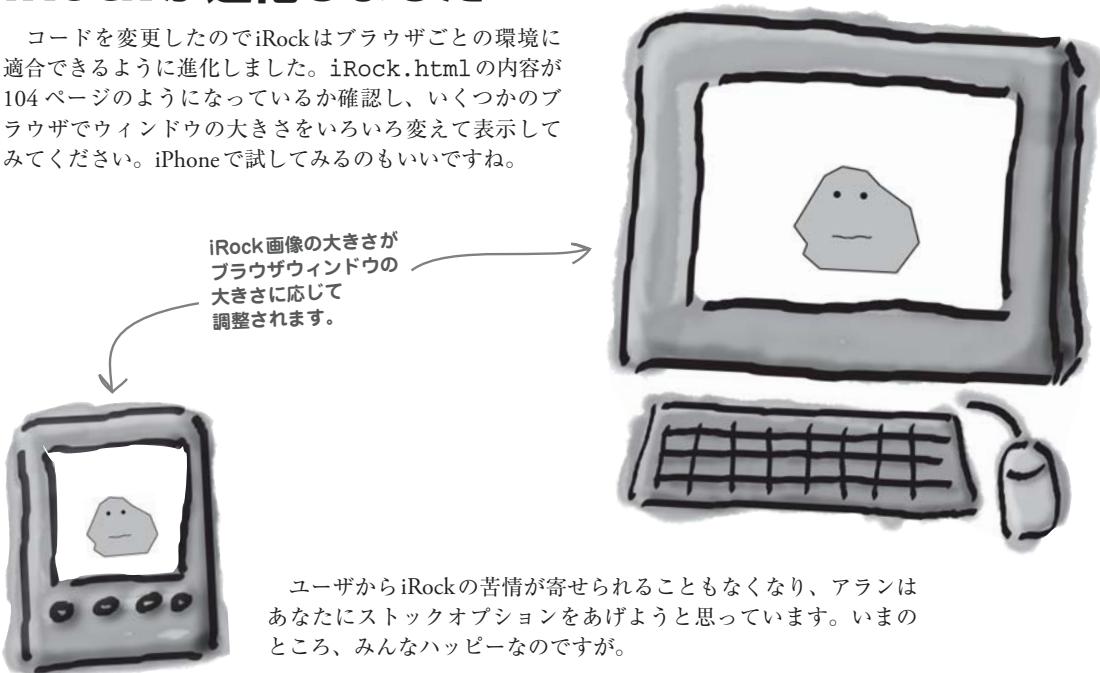
残りのウィンドウ
サイズの 90%。

重要ポイント

- `setTimeout()` を使うと 1 回かぎりのタイマーが作れます。一定時間が経過したら JavaScript コードが起動されます。
- 一定の時間間隔で処理を繰り返すタイマーを設定するには `setInterval()` を使ってインターバルタイマーを作成します。
- タイマーの経過時間は必ずミリ秒で指定します。ミリ秒は 1 秒の 1000 分の 1 です。
- ウェブページ要素には `width` や `height` などのスタイルプロパティを設定するのに使う `style` オブジェクトがあります。
- クライアントウィンドウはウェブページが表示されるブラウザウィンドウの領域です。
- `document` オブジェクトのプロパティである `body.clientWidth` と `body.clientHeight` を使えば、クライアントウィンドウの幅と高さにアクセスできます。

iRockが進化しました！

コードを変更したのでiRockはブラウザごとの環境に適合できるように進化しました。iRock.htmlの内容が104ページのようになっているか確認し、いくつかのブラウザでウィンドウの大きさをいろいろ変えて表示してみてください。iPhoneで試してみるのもいいですね。



ユーザからiRockの苦情が寄せられることもなくなり、アランはあなたにストックオプションをあげようと思っています。いまのところ、みんなハッピーなのですが。

素朴な疑問に答えます

Q: iRockの画像の大きさの計算で100の意味がわかりません。これは何ですか？

A: iRockページのHTML/CSSコードでは、クライアントウィンドウの上部にあるものとぶつからないように、画像の上に100ピクセル余白をとっています。このため、クライアントウィンドウの高さの90%で画像の高さを決定する前に、100ピクセル分オフセットとして引いています。100ピクセルには何の魔法でもなく、たまたまほとんどのブラウザでうまく配置されるだけです。

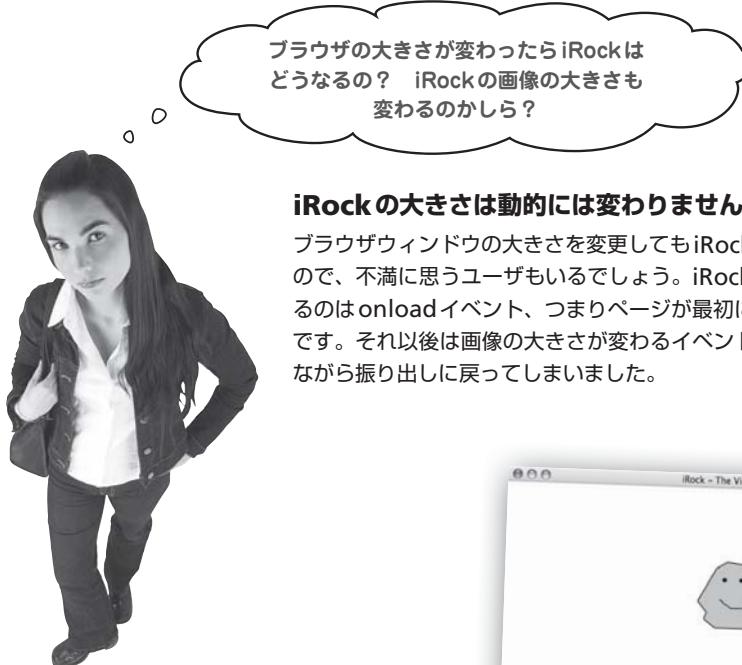
Q: CSSスタイルのプロパティであるwidthとheightを使えば、どんなものでも大きさを変更できますか？

A: ほとんど大丈夫です。ウェブページのコンテンツを操作するとき、JavaScriptがどれだけ強力であるか

を理解する手掛りになります。iRockスクリプトの場合、クライアントウィンドウに大きさを問い合わせ、その大きさを使って画像の大きさを変更することができます。

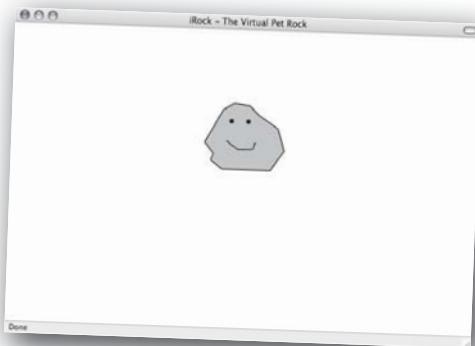
Q: onloadイベントを使うのではなく、なぜページのヘッド部にあるJavaScriptコードでiRock画像の大きさを変更しないのですか？

A: この問題はonloadイベントが発生するまでウェブページのコンテンツの読み込みが完了していないことと関係します。iRockコードがそうですが、JavaScriptコードがページの要素にアクセスする場合、onloadイベントが発生する前にコードを実行させることはできません。

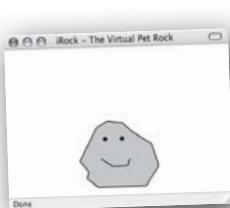


iRockの大きさは動的には変わりません。

ブラウザウィンドウの大きさを変更してもiRockの大きさは変わらないので、不満に思うユーザもいるでしょう。iRockの画像の大きさが変わるのは**onload**イベント、つまりページが最初に読み込まれたときだけです。それ以後は画像の大きさが変わるイベントは発生しません。残念ながら振り出しに戻ってしまいました。

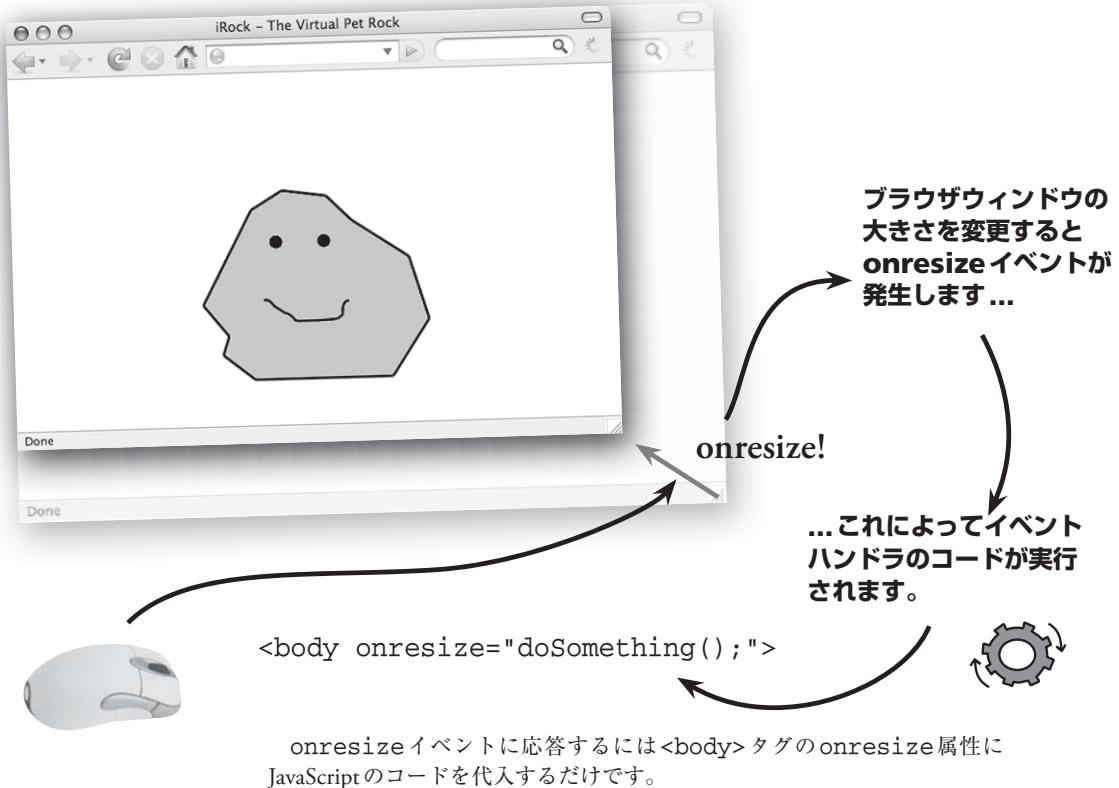


iRock画像の大きさは
ブラウザウィンドウの大きさ
が変わっても同じままです。



onresizeはブラウザの大きさが変更されたとき発生します

ブラウザのクライアントウィンドウに合わせてiRock画像の大きさを維持するには、ユーザがブラウザウィンドウの大きさを変更したタイミングをスクリプトが知る必要があります。ブラウザウィンドウの大きさが変更されたときonresizeイベントが発生するので、iRock画像の大きさを変更するのに必要なのはこのイベントを捕捉することです。



自分で考えてみよう

この中にひとつだけ他と違う性質のものがあります。それはどれでしょう？
どこが違うのでしょうか？

.onload

onresize

.onclick



この中にひとつだけ他と違う性質のものがあります。それはどれでしょう？
どこが違うのでしょうか？

onload

onresize

onclick

onresize と onclick はユーザの操作で発生するイベントですが、 onload はそうではありません。

onresize イベントを使って iRock の大きさを変更します

それでは iRock 画像の大きさを変更する関数を作成することにしましょう。ブラウザ ウィンドウの大きさが変更されたとき、それに応答して iRock 画像の大きさを変更するには、onresize イベントで resizeRock() を呼び出す必要があります。

```
<body onload="resizeRock(); greetUser();"  
          onresize="resizeRock(); ">
```

ページが最初に読み込まれたとき
発生します。

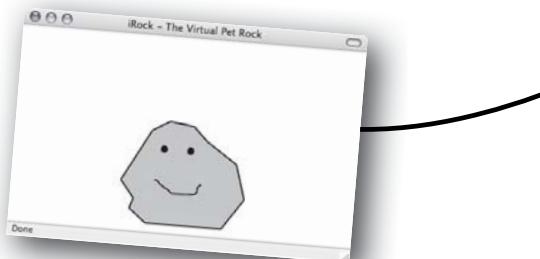
resizeRock() はページが最初に
読み込まれたときに呼び出され、
iRock 画像の大きさを初期設定
します。

このイベントに対して複数の
コード断片を呼び出すことが
できます。

resizeRock() はブラウザウィンドウの
大きさが変わると、
このイベントが発生します。

ユーザがブラウザ ウィンドウの大きさを変更すると、
iRock の画像の大きさは自動的に調整されます。

onresize イベントを使うとブラウザ
ウィンドウの大きさが変更されたのを検出し
応答することができます。

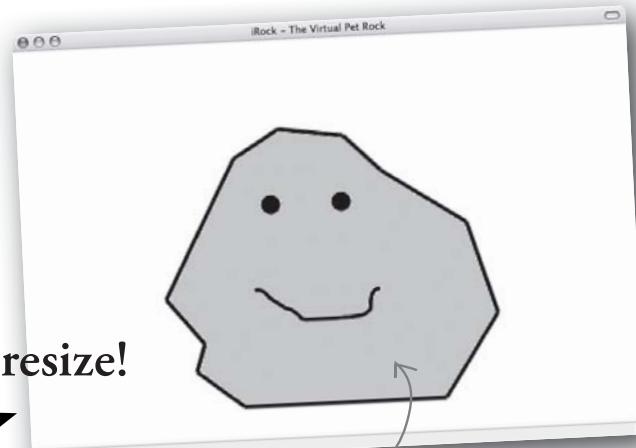




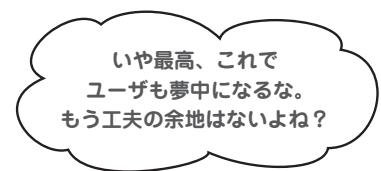
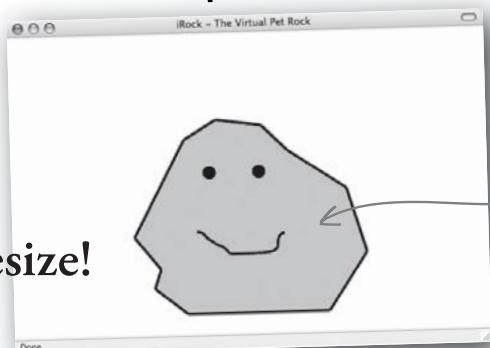
要注意！

とくに小さな画像を大きくするときは、
画像の質が悪くなることがあります。

JavaScriptで画像の
大きさを変えるときは
注意しましょう。



JavaScriptはクライアントの変化を
検知し、変化に応じてウェブページの
コンテンツを動的に変更します。



ブラウザの大きさにあわせてiRockが表示されるようになったので、
ユーザからの苦情もなくなりました。ブラウザのクライアントウィンドウ
に合わせるだけでなく、ブラウザの大きさが変更されたときもiRockの
大きさは動的に調整されます。

お会いしたことありました？

iRockの大きさの問題はもう過去のことになりましたが、ユーザがiRockを寂しがらせないように何回もクリックしたらどうなるでしょう？ コンピュータを再起動してからiRockを表示するとどうなるでしょうか？



iRockに最初に会ったとき
ユーザは名前を入力します。

あなたの名前は?

ポール

キャンセル OK

iRockは名前を呼んで
挨拶するので
親近感がわきます。

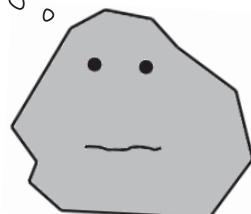
はじめまして、ポールさん。

OK

タップ
時間が経過して
何かが変わりました…

ポールって
誰だろう？ 会った
ことあるかな？

…iRockは名前を
憶えていません。



あなたの名前は?

ポール

キャンセル OK

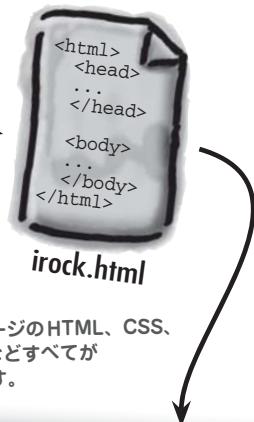
iRockは飼い主に何度も会っているのですが、
名前を忘れてしまうようです。

スクリプトにはライフサイクルがある

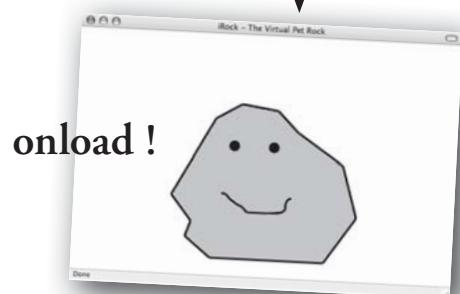
iRockの記憶喪失はスクリプトのライフサイクルと関係しています。ライフサイクルはスクリプトの変数に格納されたデータに影響します。



ブラウザが起動します。
ページはまだ読み込まれて
いないので、ユーザは
URLを入力する必要があります。



サーバからページのHTML、CSS、
JavaScriptなどすべてが
読み込まれます。



ブラウザが閉じたり
ページが再読み込み
されたとき、JavaScriptは
すべての変数を壊します。

スクリプトは停止し、
JavaScriptの変数は
消えて、ページが
閉じます。



ユーザーがブラウザを
閉じたりページを
再読み込みました。

onloadイベントが
発生するとJavaScript
変数が作成され
初期化されます。




頭の体操

iRockがユーザの名前を忘れてしまう問題を解決するにはコードを
どのように修正すればいいでしょう？

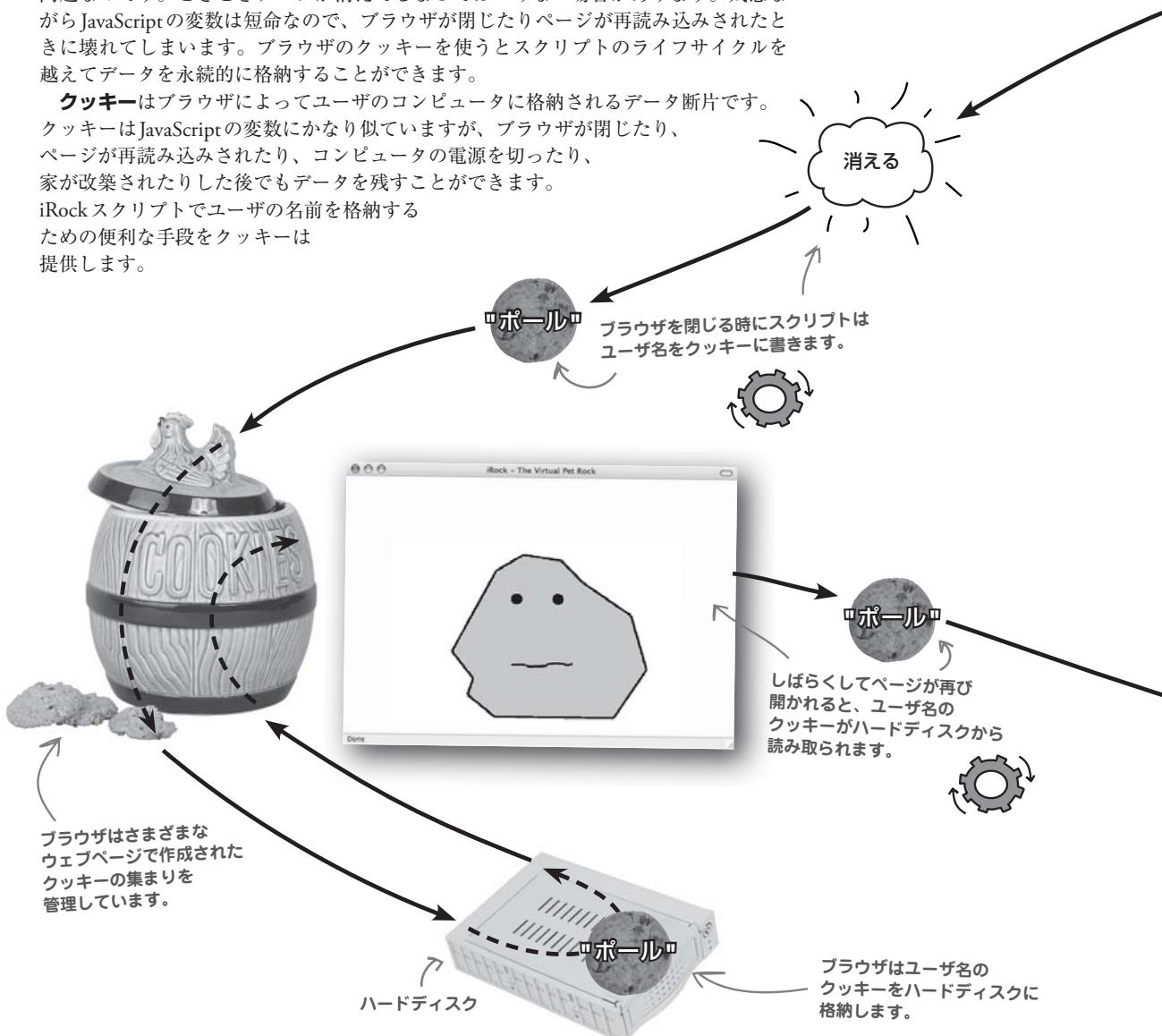
クッキーはスクリプトのライフサイクルを越える

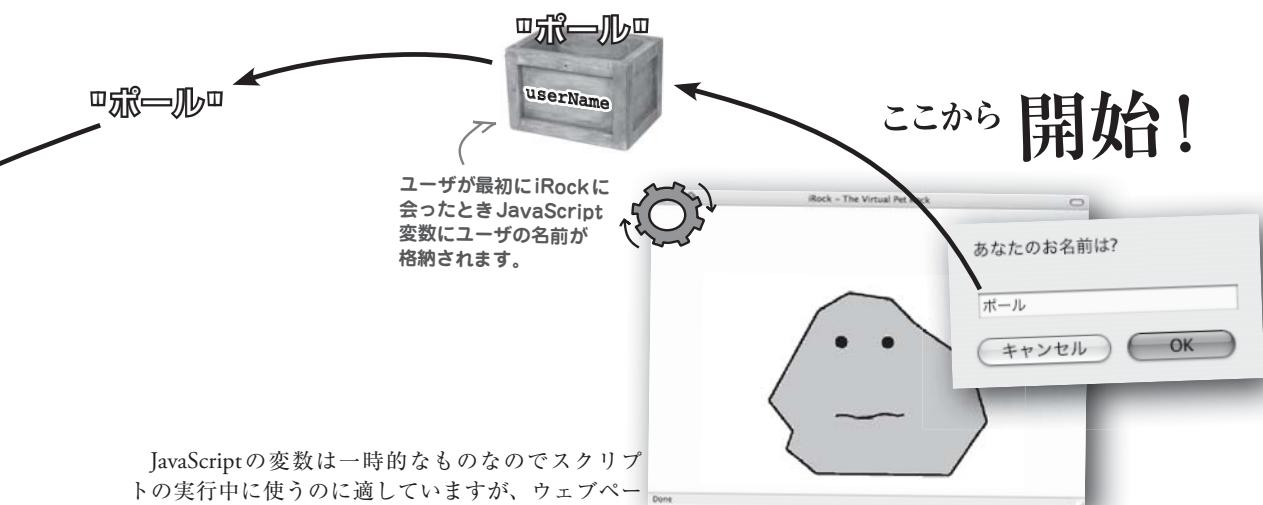
iRockが名前を忘れる問題は永続性の問題です。いや、正確には**永続性**がないのが問題なのです。ときどきデータが消えてしまってはいけない場合があります。残念ながらJavaScriptの変数は短命なので、ブラウザが閉じたりページが再読み込みされたときに壊れてしまいます。ブラウザのクッキーを使うとスクリプトのライフサイクルを越えてデータを永続的に格納することができます。

クッキーはブラウザによってユーザのコンピュータに格納されるデータ断片です。

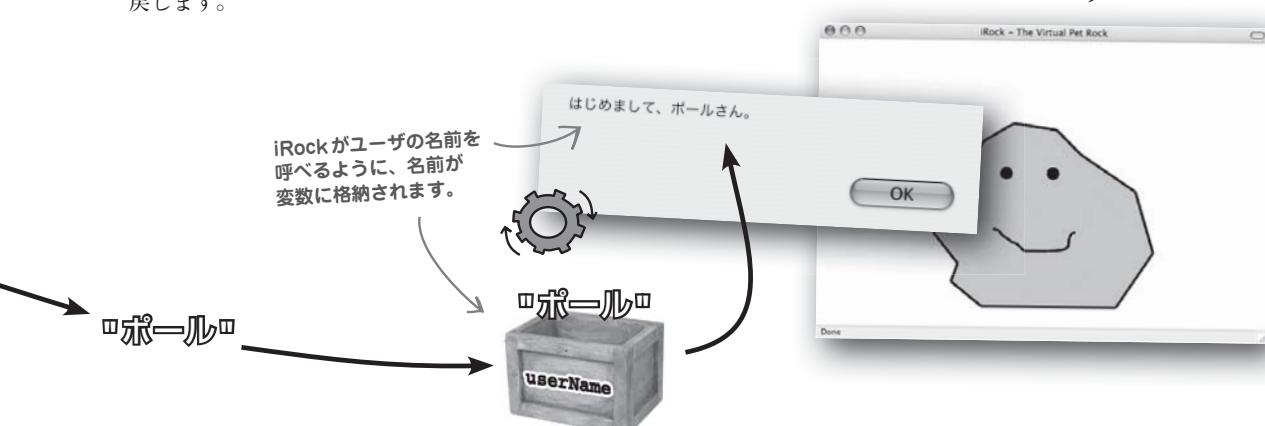
クッキーはJavaScriptの変数にかなり似ていますが、ブラウザが閉じたり、ページが再読み込みされたり、コンピュータの電源を切ったり、家が改築されたりした後でもデータを残すことができます。

iRockスクリプトでユーザの名前を格納するための便利な手段をクッキーは提供します。





終わり！



自分で考えてみよう

クッキーを使って永続的に格納しておくと便利なウェブページのデータにはどんなものがあるか書き出してください。

特別座談会



今夜の対談：
変数とクッキーが永続的データの重要性について
話し合います

変数：

なんできみと話さなきゃいけないのか、わからないなあ。
きみはJavaScriptと何の関係もないよね。

きみの得意分野はわかったよ。ブラウザが閉じたり、ページが再読み込みされたら、ぼくは意識もうろうになってしまふから、ぼくにはデータ記憶のオプションが足りないと君は思っているんだろうね。でも他のひとたちとは違って、ぼくはとてもアクセスしやすい奴なんだぜ。

そうかもしれないけど、他のクッキーたちの集団の中で窮屈な思いをしているんじゃなかった？

噂がほんとうなら、ひとつのクッキーを探すのにかなりの手間がかかるそうだね。きみたちは全員がひとつの大きなリストの中に格納されているだけだから。痛々しいなあ。
アクセスしにくいって、そういうことだと思うね。

クッキー：

まあ半分はそうですけどね。JavaScriptの力を借りなくとも、私は仕事ができますけど、でもJavaScriptと重要なつながりもあるんですね。私はデータを永続的に格納する手段を提供するわけなんです。たぶんあなたも理解されていると思いますが、JavaScriptは永続化に関しては得意じゃないですからね。

アクセスしにくいの間違いじゃなくて？ 私はいつもブラウザの中にいて、いつ呼び出されてもいいように待機していますよ。

そうですけど、それがどうかしました？

いやあ、まったく、私たちクッキーは大きなリストに格納されていますが、全員が一意の名前を持っているので、私たちをつかまえるのはそんなに難しくありませんよ。リストを分解して特定の名前を探す方法を勉強なさいですよ。

自分で考えてみよう の答え

クッキーを使って永続的に格納しておくと便利なウェブページのデータには
どんなものがあるか書き出してください。

ユーザID、買い物かご、住所、言語

変数：

その通りだけど、そこが問題なんだな。ぼくにアクセスするのにリストなんて要らないからね。名前を呼ばれただけで、♪アイル・ビー・ゼアってわけさ。

永遠ってのはすごいけど、日常の問題を解決できないよね。よく考えてみれば、永遠に続くデータなんてそんなに必要ないんだよ。実際のところ、データを一時的に格納して、そのデータが不要になったら捨ててしまうのがもっと効率的だよね。そこでぼくの出番となるわけ。スクリプトのデータのための究極の一時的記憶媒体、それがぼくさ。

それは面白いけど、そうした商品を買い物かごに入れるときのことはどう思うのかな？ 買い物かごのほとんどは、一時的なデータをぼくに格納しているんだよ。ぼくも同じくらい、いやそれ以上に重要なはずさ。たとえばぼくの物忘れがひどいとしてもね。

きみの言うとおりかもしれない。お互い抱える問題が違うから、ほんとは競うこともないのにね。そうだとしても、永続的に格納できるきみより、アクセスしやすいぼくの方が優れていると思うね。

ひどい話だね。

クッキー：

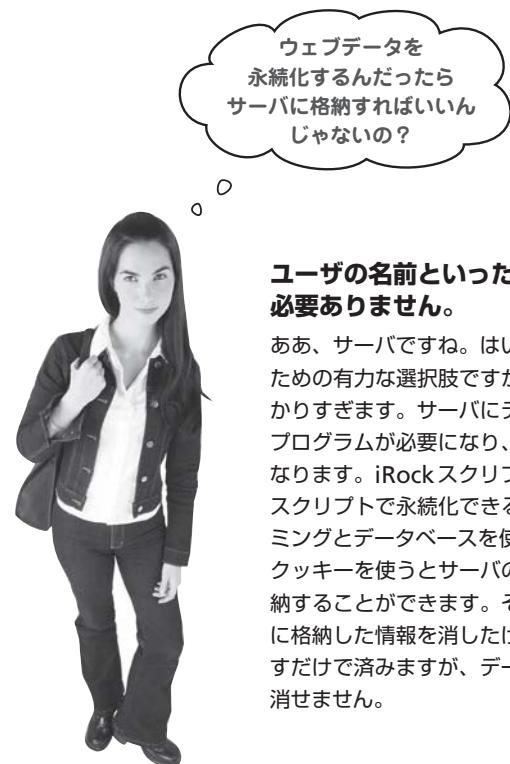
歌わなくてもいいのに。おっしゃりたいことはわかりました。でも問題はこういうことですよ。あなたがクッキーに何かを格納すると、私はそれを記憶します。ブラウザが閉じようが、ページが再読み込みされようが、そんなことは関係ない。それこそユーザがクッキーをすべて消してもしないかぎり、私は永遠ですから。でもそれはまた別の問題ですね。

ともあれ、永続的データ記憶がどれだけ重要か、あなたは理解されているようですね。オンラインショップをブラウズして数日後に戻ってみたら、買い物かごにデータが残っている魔法にびっくりしませんでしたか？ そうした魔法を私は可能にしているわけです。

なんだかまるで、お互に足りないところを補い合ってるみたいですね。因果応報ですかね。

でも分業には抵抗できないでしょ？ 私は王道を歩いているけど、あなたは次のページになったら、ここでの会話を忘れているのが容易に想像できますよ。

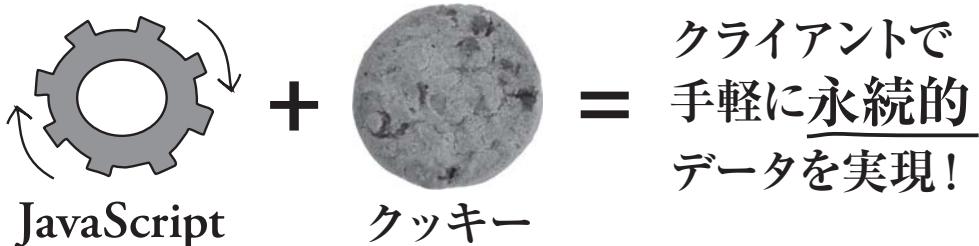
まったく。



ユーザの名前といった小さな情報断片ならばサーバは必要ありません。

ああ、サーバですね。はい、サーバはデータを永続的に格納するための有力な選択肢ですが、小さな情報断片を格納するのは大掛かりすぎます。サーバにデータを格納するにはサーバで動作するプログラムが必要になり、データベースなどの記憶媒体も必要になります。iRockスクリプトのユーザ名など簡単なクライアントスクリプトで永続化できるデータ断片であれば、サーバプログラミングとデータベースを使うまでもありません。

クッキーを使うとサーバの力を借りなくとも永続的にデータを格納することができます。それだけでなく、ウェブページが永続的に格納した情報を消したければ、ブラウザのクッキーデータを消すだけで済みますが、データをサーバに格納してしまうと簡単に消せません。



クッキーには名前と値と期限があります。

クッキーはデータの断片を変数のように一意の名前と対応させて格納します。ただし変数と違ってクッキーには有効期限があります。この期限を過ぎるとクッキーは破棄されます。そのため実際のところクッキーには完全な永続性ではなく、変数より寿命が長いだけです。有効期限なしでクッキーを作成することはできますが、その場合JavaScriptの変数と同じようにブラウザが閉じるとクッキーも消えます。

クッキーはユーザのコンピュータに格納されるとき、ウェブサイト（またはドメイン）に関連付けられたひとつの大きなテキスト文字列になります。テキスト内のクッキーはそれぞれセミコロン（;）で区切られています。セミコロンはクッキーのリストを読み、その中からひとつのクッキーを取り出すための鍵になります。





後は焼くだけ JavaScript

わからないところがあつても
大丈夫。本書を読み進むと
理解できます。

```
function writeCookie(name, value, days) {
    // デフォルトでは有効期限がないので一時的クッキーになります。
    var expires = "";

    // クッキーの有効期限を日数で指定。
    if (days) {
        var date = new Date();
        date.setTime(date.getTime() + (days * 24 * 60 * 60 * 1000));
        expires = "; expires=" + date.toGMTString();
    }

    // クッキーに名前、値、有効期限を設定。
    document.cookie = name + "=" + value + expires + "; path=/";
}

function readCookie(name) {
    // 指定されたクッキーを探し、値を返す。
    var searchName = name + "=";
    var cookies = document.cookie.split(';");
    for(var i=0; i < cookies.length; i++) {
        var c = cookies[i];
        while (c.charAt(0) == ' ')
            c = c.substring(1, c.length);
        if (c.indexOf(searchName) == 0)
            return c.substring(searchName.length, c.length);
    }
    return null;
}

function eraseCookie(name) {
    // 指定されたクッキーを消去。
    writeCookie(name, "", -1);
}
```

値を空にして有効期限を 1 日
前にすれば、クッキーは消去
されます。

クッキー利用をサポートする 3 つの関数のコードを以下に示します。
これを使うと、クッキーの書き出し、読み込み、消去が簡単に行えます。他人の成果をうまく利用するのは賢いやり方ですね。このレシピを使えば自家製のクッキー関数が最大限に活用できます。

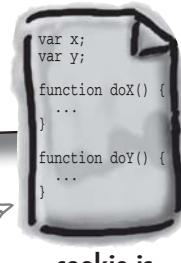
クッキーの有効期限
を日数で指定します。

日数をミリ秒に換算し、
現在の時刻を加算します。

クッキーのリストを
セミコロンで分割して
個々のクッキーに
分解します。

JavaScriptコード
だけを含むファイルは
拡張子を .js にします。

cookie.js という
空ファイルを作成し、
このコードを追加します。



JavaScriptはウェブページの外に置いてもかまいません

JavaScriptコードを別ファイルに格納するときは、そのコードを使うウェブページに取り込む必要があります。たとえばクッキー関数をまとめたcookie.jsの場合、iRock.htmlに取り込む必要があります。以下の<script>タグを使えば実現できます。

</script>タグで閉じるのを忘れないように。

```
<script type="text/javascript" src="cookie.js"></script>
```

JavaScriptコードの場合
typeをtext/javascriptに
します。

通常、スクリプトコードの
ファイル名は.jsで終わります。

このように書きます。
iRock.htmlとcookie.jsは
同じディレクトリに
配置してください。

ページに外部ファイルのスクリプトコードを取り込むとき、そのスクリプトファイルの中のすべてのJavaScriptコードがHTMLコードの<script>タグの中に挿入されるので、ちょうどウェブページに直接コードを書いたのと同じ状態になります。複数のページで使えるコードがあるときは、そのコードを外部ファイルにしてページの中に取り込むようにするのがいいでしょう。

```
<html>
<head>
<title>iRock - The Virtual Pet Rock</title>
<script type="text/javascript" src="cookie.js"></script>
<script type="text/javascript">
var userName;

function resizeRock() {
    document.getElementById("rockImg").style.height =
        (document.body.clientHeight - 100) * 0.9;
}

function greetUser() {
}
```

ページが読み込まれるとスクリプトコードがここに取り込まれます。



エクササイズ

再利用可能なコードを外部ファイルにするメリットを書いてください。



エクササイズの
答え

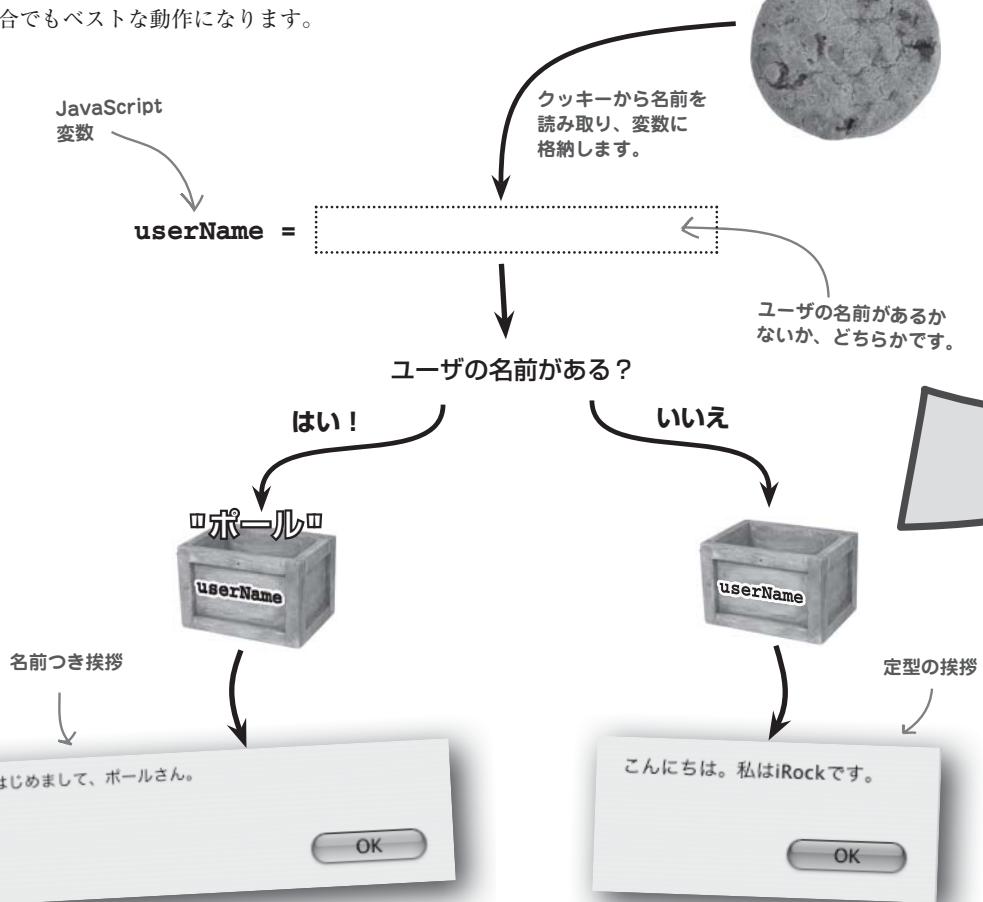
再利用可能なコードを外部ファイルにするメリットを書いてください。

コードが再利用できて維持管理が楽になります。

クッキーを使ってユーザに挨拶しましょう

iRockスクリプトのクッキー対応バージョンでは、ユーザが入力した名前がすでにクッキーに格納されていると前提して、挨拶文にユーザの名前を含めて表示する必要があります。クッキーに名前が格納されていない場合、定型の挨拶文が表示されるので、どちらの場合でもベストな動作になります。

userName = ポール
Expires 3/9/2009



コード詳細

```

function greetUser() {
  userName = readCookie("irock_username");
  if (userName)
    alert("こんにちは " + userName + "さん、会えなくてさみしかったです。");
  else
    alert('こんにちは。私は iRock です。');
}

```

greetUser() はページが最初に読み込まれたときユーザに挨拶するための関数です。

スクリプトに他のクッキーを追加するときに間違わないように iRock のユーザの名前のクッキーにはわかりやすいタイトルを付けます。

これは加算ではなく文字列の連結です！

クッキーから読み取られたユーザの名前は変数userNameに格納されます。

クッキーにユーザの名前がなかったので、名前つきの挨拶を表示します。

ユーザの名前が空なので、クッキーがありません。定型の挨拶を表示します。

greetUser() が クッキー対応になりました

greetUser() での処理の内容は、変数とクッキーによるデータのデュエットと言えます。クッキーから読まれたユーザの名前が変数に格納されます。ではクッキーが名前を保持しているとは前提できない状況、たとえばスクリプトが最初に実行されてユーザがまだ名前を入力していない場合にはどうなるでしょう？ このような状況に備えて、コードではクッキーから変数に値が設定されているか確認しています。これによって挨拶文に名前が含まれるかどうかが決まります。

クッキーを設定するのもお忘れなく

iRockのクッキーはうまく読み込めますが、そのためにはあらかじめクッキーを設定しておく必要があります。クッキーの書き出しはtouchRock()で実行されますが、この関数はユーザがiRockの画像をクリックしたときに呼び出されます。touchRock()ではすでにユーザに名前を入力するように促しているので、名前が入力された後でクッキーに名前を書き出せばいいのです。



**コード
詳解**

最初にユーザの名前があるか調べます。

ユーザの名前があったら、名前をつけて返事します。

ユーザの名前がなかったときはユーザに入力してもらう必要があります。

```
function touchRock() {
    if (userName) {
        alert(userName + " さん、声をかけてくれてありがとう。");
    }
    else {
        userName = prompt("あなたの名前は ?");
        if (userName) {
            alert("はじめまして、" + userName + " さん。");
            writeCookie("irock_username", userName, 5 * 365);
        }
    }
    document.getElementById("rockImg").src = "rock_happy.png";
    setTimeout("document.getElementById('rockImg').src = 'rock.png';",
        5 * 60 * 1000);
}
```

ユーザが名前を入力したかチェックします。

名前があるので、名前つきで挨拶し、名前をクッキーに書きます。

クッキーの読み取りに使ったのと同じ名前です。クッキーにはキャメルケースは使えません。HTMLのIDに似ています。

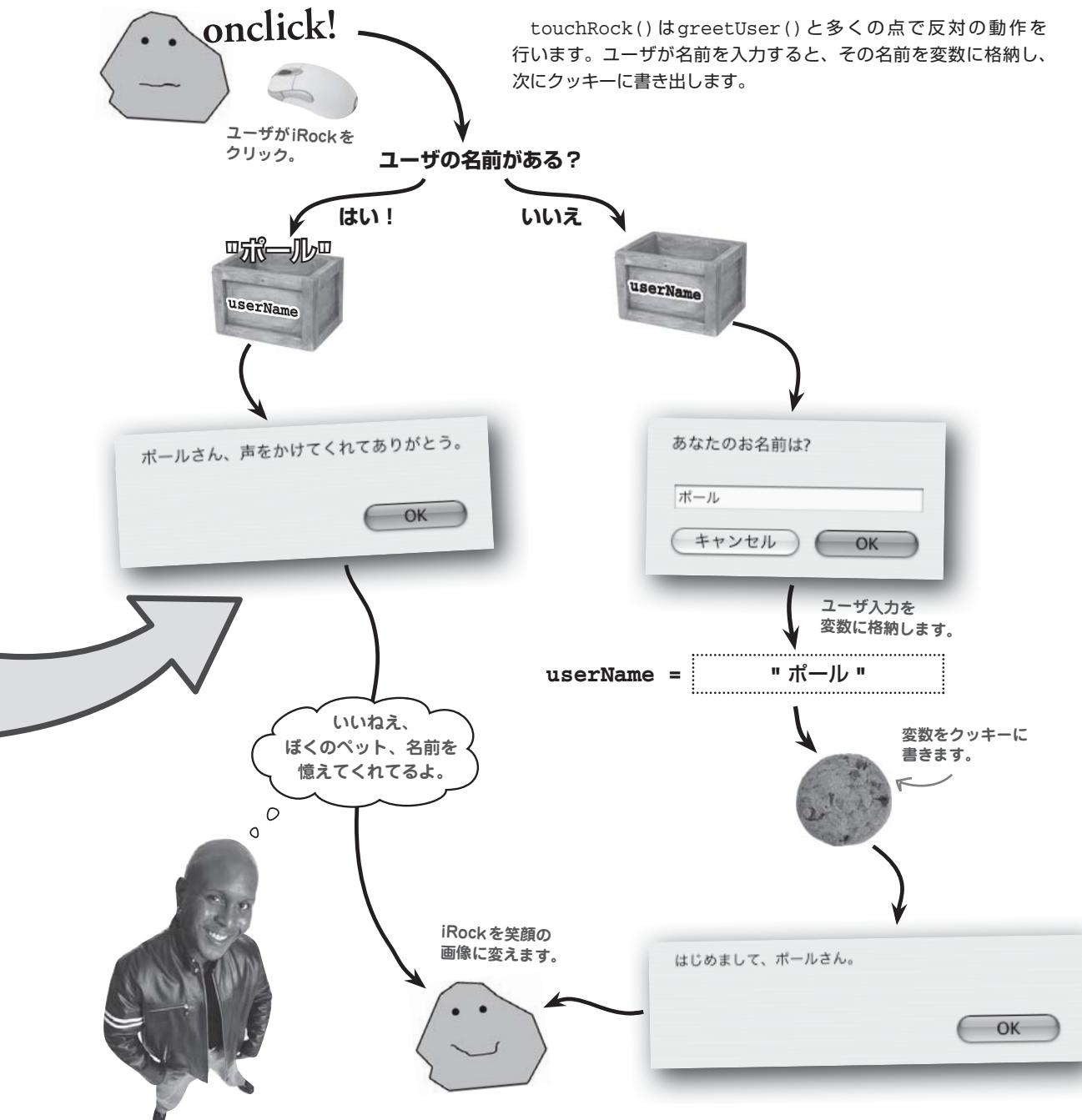
変数の値でユーザの名前がクッキーに書かれます。

ユーザの名前のクッキーを5年くらい保存します。

iRock画像がクリックされたときこの関数が呼び出されます。

```

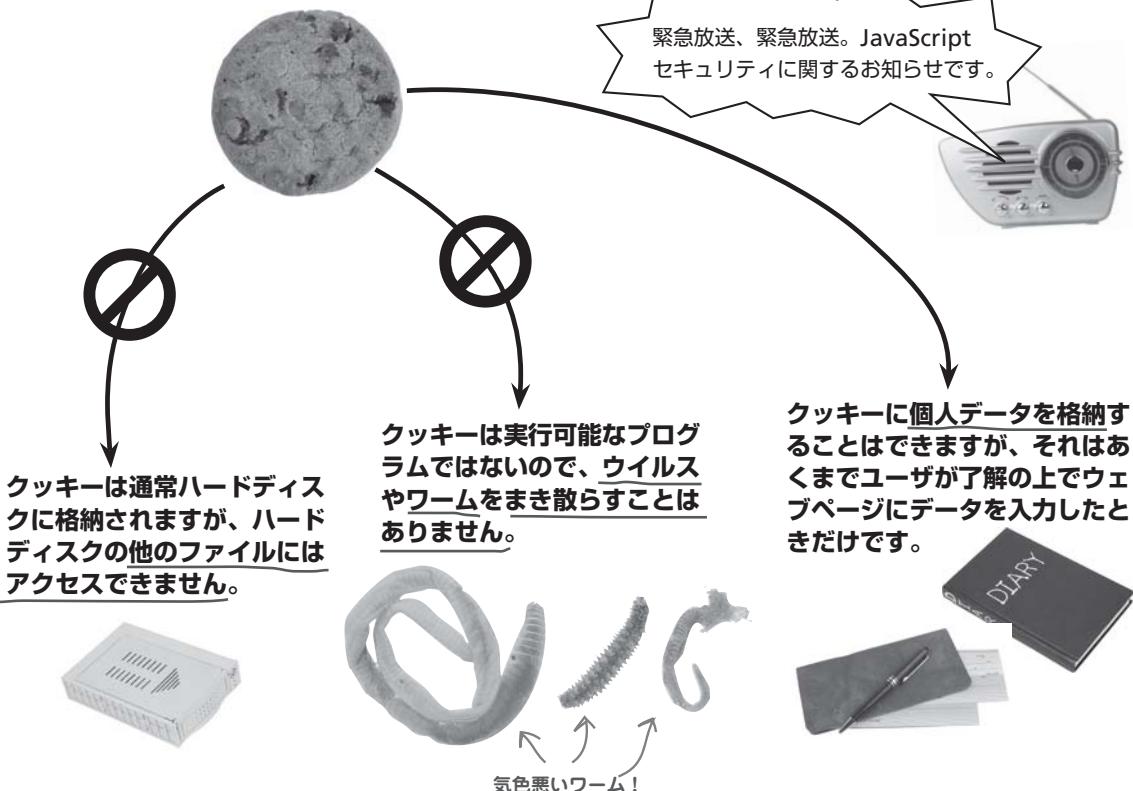
```



クッキーはブラウザのセキュリティに影響します

iRockユーザのほとんどはiRockの記憶喪失がクッキーで治ったことを喜んでいますが、クッキーによってセキュリティが脅かされる心配をしているユーザも少なくありません。クッキーには個人データが格納されることもあるので、この心配は偏ったものではありませんが、クッキーによってセキュリティのリスクが生じることは現実にはありません。すくなくとも、コンピュータに格納されている機密データがアクセスされることはありません。とはいえクッキー自体は安全な記憶領域とは見なされていないので、クッキーに機密データを格納するのは避けるべきです。

クッキーはあなたのコンピュータのファイルに格納されるテキストデータの断片です。



素朴な疑問に

Q: クッキーは必ずユーザのハードディスクに格納されるのですか?

A: いいえ。ブラウザのほとんどはクッキーをハードディスクに格納しますが、すべてのブラウザがハードディスクにアクセスできるわけではありません。たとえば、モバイル機器はハードディスクのかわりに特別なメモリを永続的データ領域として使います。この場合、ブラウザは永続的メモリを使ってクッキーを格納します。その場合でも、ブラウザ(およびスクリプト)から見ると、画面の裏でどのようにクッキーが格納されているかに關係なく、クッキーには値が記憶されています。

Q: クッキーの名前が一意かどうか、どうしたらわかりますか?

A: クッキーの名前が一意なのは、与えられたウェブページの中だけにおいてです。これは、クッキーが作成されたページ(そのページのウェブサイトを含む)に対応して格納されるからです。これは一意性の観点からみれば、ページがクッキーの名前の一部であることを意味します。つまり、クッキーは与えられたページまたはサイトの中で名前が一意であれば十分ということです。



答えます

Q: クッキーのデータを別々のブラウザ間で共有できますか?

A: いいえ。ブラウザはそれぞれがクッキーのデータベースを維持管理します。なので、Internet Explorerで設定されたクッキーはFirefoxやOperaからは見えません。

Q: クッキーがそんなに手軽だったら、なぜデータをサーバに格納する必要があるのでしょうか?

A: まず何よりも、クッキーは比較的小さなテキスト(4KB以下)にしか適していません。これはクッキーの重要な制約のひとつです。もっと問題なのは、クッキーがそれほど効率的でないことです。大量のデータを絶えず読み書きするのに、クッキーは適していません。それはサーバで稼働するデータベースの役割です。つまり、サーバ上のデータベースを必要としない小さなデータ断片の格納にはクッキーが向いていますが、ウェブデータのすべてのニーズを解決するものではありません。またセキュリティを念頭において設計されてはいないので、機密データの格納にも適してはいません。

Q: 永続的なクッキーを作成する方法はありますか?

A: いいえ。クッキーにはすべて有効期限があります。クッキーは、長期的なデータ記憶ではなく、中期的なデータ記憶のための手段なのです。言い換えると、クッキーは、日、週、月などの単位でデータを格納するのに適しています。長期間にわたって保管したいデータを扱うのであれば、クッキーではなくサーバに格納したくなるでしょう。数年にわたって保管したいデータをクッキーに格納できないわけではありませんが、コンピュータの入れ替えやブラウザの再インストールなどによって、クッキーのデータが消去されてしまうでしょう。

Q: クッキーのことはおよそわかりました。JavaScriptコードを外部ファイルに格納したときのデメリットはありますか?

A: 実際デメリットはありませんが、コードを外部化する目的は、コードを複数のウェブページで使う必要がある場合に、コードの共有と維持管理を容易にすることにあります。あるひとつページにだけ現れるコードを扱うのなら、それを外部ファイルに移してもメリットはありません。ページがとりわけ乱雑で、正気を保つためにコードを分離させたい場合は別ですが。

重要ポイント



- クッキーはブラウザによってユーザのコンピュータに格納されるテキストデータ断片です。
- クッキーを使うとスクリプトはひとつのウェブセッションを越えてデータを保持することができます。
- クッキーには有効期限があり、有効期限を過ぎるとブラウザによってクッキーは破棄されます。

- スクリプトコードを外部ファイルに移すのは、コードを再利用可能で維持管理しやすくする手軽な方法です。
- クッキーはユーザのハードディスクにアクセスしたりウイルスをまき散らすことはできませんが、ウェブページに入力された個人データを格納することはできます。

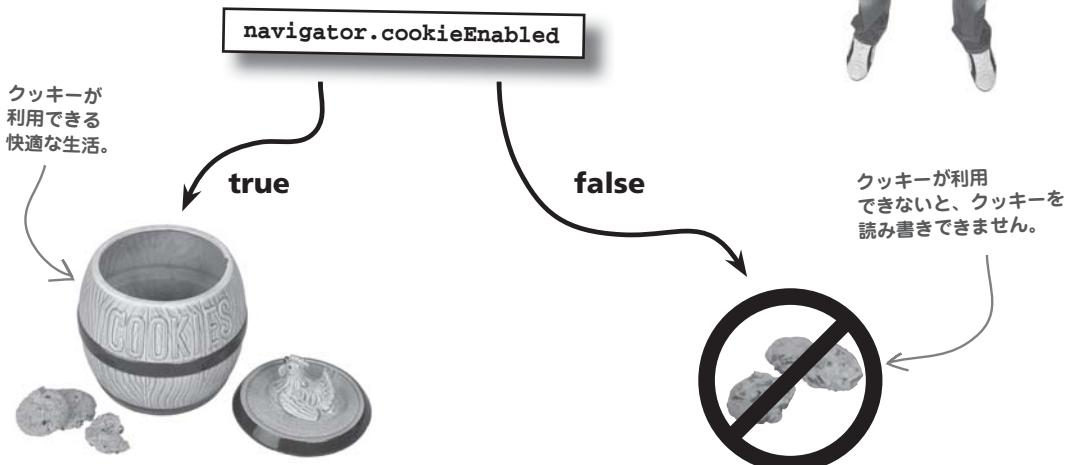
クッキーのない世界

セキュリティ対策やブラウザの機能が制限されているなどの理由から、iRockのユーザの何人かはクッキーがブラウザで利用できないので、クッキー対応iRockのメリットを享受できません。iRockスクリプトは誰でもクッキーを使えると前提しているので、これは大問題です。iRockが記憶をクッキーに頼っているのが原因ですが、少なくともクッキーが使えないユーザにそれが原因でiRockのフル機能を使えていないことを知らせないでおくのは良くありません。

ユーザを全員
放置すると顧客が一人も
いなくなるから、これは
受け入れられないな。



クッキーが利用可能かどうかを調べることができます。論理値プロパティを
ブラウザは持っています。navigatorオブジェクトのcookieEnabled
プロパティがそれです。このオブジェクトはブラウザに関する情報を
JavaScriptに提供します。





自分で考えてみよう

`greetUser()`と`touchRock()`の中にクッキー対応をチェックするコードを書いてください。`touchRock()`にはクッキーが利用できないことをユーザーに知らせるコードも追加してください。

```
function greetUser() {  
  
    .....  
    userName = readCookie("irock_username");  
    if (userName)  
        alert("こんにちは " + userName + "さん、会えなくてさみしかったです。");  
    else  
        alert('こんにちは。私は iRock です。');  
}  
  
function touchRock() {  
    if (userName) {  
        alert(userName + "さん、声をかけてくれてありがとう。");  
    }  
    else {  
        userName = prompt("あなたの名前は？");  
        if (userName) {  
            alert("はじまして、" + userName + "さん。");  
  
            .....  
            writeCookie("irock_username", userName, 5 * 365);  
        }  
    }  
    .....  
}  
document.getElementById("rockImg").src = "rock_happy.png";  
setTimeout("document.getElementById('rockImg').src = 'rock.png';",  
5 * 60 * 1000);  
}
```



自分で考えてみよう の答え

`greetUser()`と`touchRock()`の中にクッキー対応をチェックするコードを書いてください。`touchRock()`にはクッキーが利用できることをユーザに知らせるコードも追加してください。

クッキーが利用できるならiRockの
クッキーからユーザの名前を読みます。

```
function greetUser() {
    if (navigator.cookieEnabled) ←
        userName = readCookie("irock_username");
    if (userName)
        alert("こんにちは " + userName + "さん、会えなくてさみしかったです。");
    else
        alert('こんにちは。私はiRockです。');
}

function touchRock() {
    if (userName) {
        alert(userName + "さん、声をかけてくれてありがとうございます。");
    }
    else {
        userName = prompt("あなたの名前は？");
        if (userName) {
            alert("はじまして、" + userName + "さん。");
            if (navigator.cookieEnabled) ←
                writeCookie("irock_username", userName, 5 * 365); ←
                クッキーが有効なので
                ユーザの名前を
                クッキーに書きます。
            else
                alert("このブラウザではクッキーがサポートされていないので、あなたのことを憶えられません。ごめんなさい。");
        }
    }
    document.getElementById("rockImg").src = "rock_happy.png";
    setTimeout("document.getElementById('rockImg').src = 'rock.png';",
              5 * 60 * 1000);
}
```

↑ irock.htmlにこの関数を書いて、テスト
してみてください。

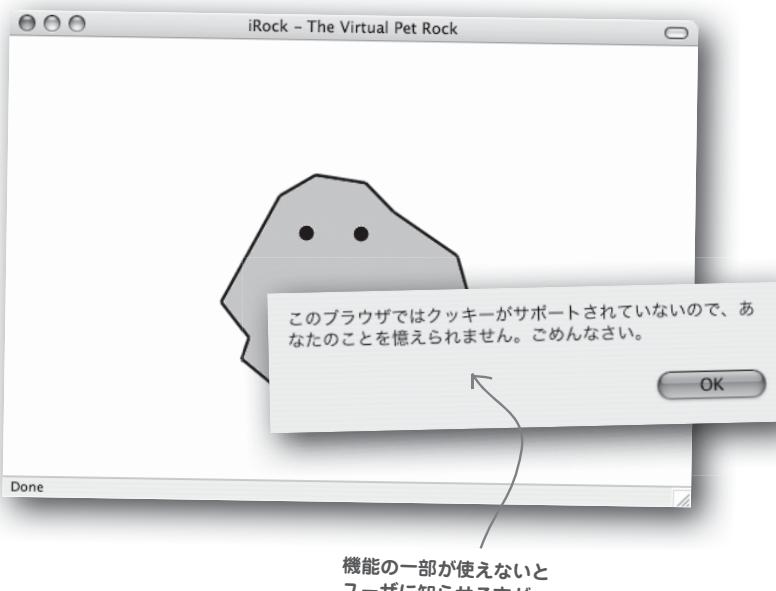
素朴な疑問に 答えます

Q: クライアントがクッキーをサポートしているか、ブラウザのタイプやバージョンをもとに調べることはできますか？

A: ブラウザのタイプをスクリプトで検出する場合、信頼できない結果になる傾向があります。ブラウザが自分で報告するバージョン情報は実は信頼できません。クッキーをサポートしているかチェックする唯一の信頼できる方法はnavigator.cookieEnabledプロパティを調べることです。

何もしないより、ユーザに話しかける方がベターです

クッキーが利用できないとき、かわりにクッキーをシミュレートする手段はありませんが、ユーザ満足の観点から考えると、ユーザにクッキーが利用できることを知らせるのは価値があります。

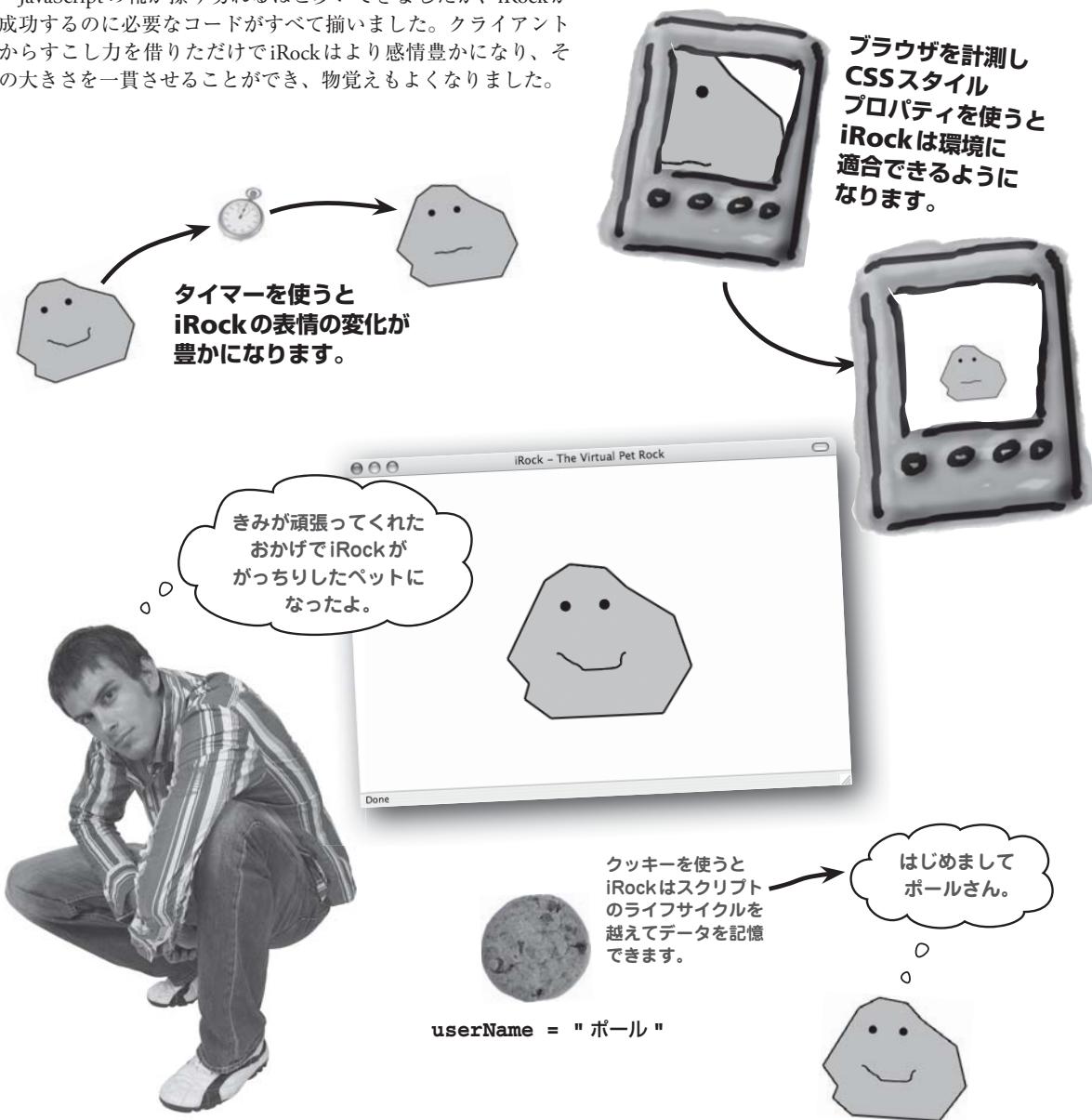


なるほどね、クッキーが
使えないブラウザには
こう対応すればいいのか。



iRock は JavaScript の王様です

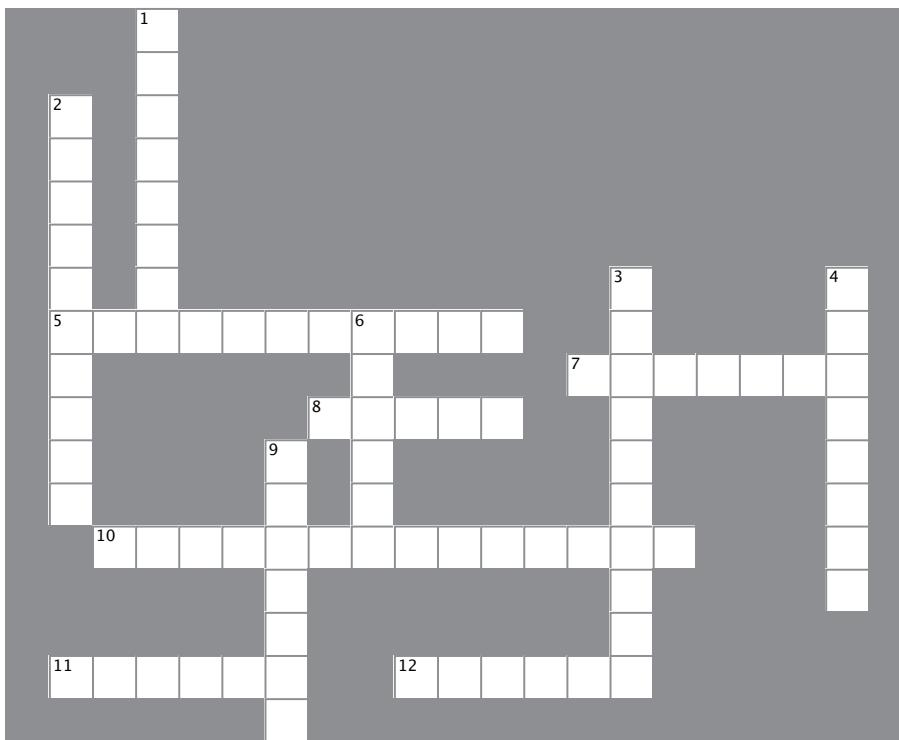
JavaScript の靴が擦り切れるほど歩いてきましたが、iRock が成功するのに必要なコードがすべて揃いました。クライアントからすこし力を借りただけで iRock はより感情豊かになり、その大きさを一貫させることができ、物覚えもよくなりました。





JavaScript クロスワード

すこし休憩して右脳で遊んでみましょう。よくあるクロスワードパズルです。
答えはすべて本章にあります。



ヨコのカギ

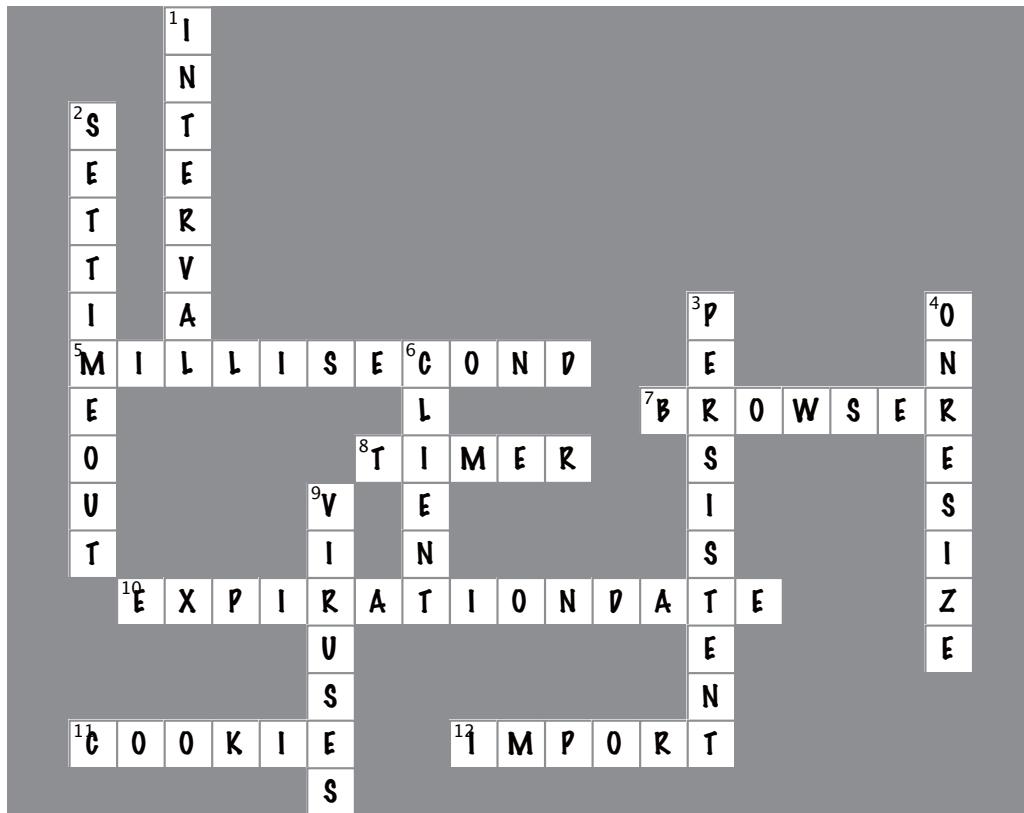
5. 1秒の1000分の1。
 6. 私はクッキーのリストを管理する責任があります。
 7. ある一定時間が経過したらコードを実行させるためのJavaScriptの機構。
 8. クッキーには名前と値と……………があります。
 9. 後で必要になるかもしれない情報断片をクライアントに格納するのに使います。
 10. ウェブページの外にあるJavaScriptコードを参照するときに行うこと。

タテのカギ

- あるコード断片を繰り返し実行するときに使うタイマーの種類。
 - この関数を使うと1回かぎりのタイマーを作成できます。
 - スクリプトが実行を完了した後でもデータが残るのであれば.....と見なされます。
 - ブラウザウィンドウの大きさが変更されると.....イベントが発生します。
 - ウェブブラウザはこのようにも言われます。
 - クッキーがこれをまき散らすことはありません。



JavaScript クロスワードの答え



脳のところで
折り畳んでから
謎を解決しましょう。

折り畳みページ

なぜJavaScriptはクライアントの
ことを気にかけるのでしょうか？

左脳と右脳がご対面！



クライアントはJavaScriptコードが実行される場所です。

これはJavaScriptがブラウザで活動することを意味します。

これはいいことです。たとえばクッキーがクライアントに格納されるので、
サーバは心配する必要がありません。

4章 意思決定

道が分かれていたら、
どっちに進むか決めなさい

制服姿の男って
魅力的よね、でもどっちを
選べばいいの？



人生は決断の連続です。止まるか進むか、煮るか焼くか、司法取引に応じるか裁判にかけられるか。決断を下すことができなければ、何も達成することができません。JavaScriptも同様です。決定によってスクリプトはさまざまな可能性からひとつを選びます。意思決定によってスクリプトの「物語」は進行するので、どんなにありふれたスクリプトであってもある種の物語があります。ユーザが入力したものを信用して雪男探検のツアーの予約を受け付けてもいいでしょうか？

それともほんとはバスに乗りたいだけなのか確認した方がいいのでしょうか？ その選択はあなた次第です。

当選者の方、こちらへどうぞ！

今日のスリリングな新企画「取引したいでショー」の当選者を発表
いたします…



選ぶことは意思決定です

そんなの彼のカードに書いてあったからに決まってます。司会者はカードに書いてあったエリックの名前をみて決定を下すのはあたり前のことです。人間は情報処理と意思決定に優れています。司会者がスクリプトだったら、そう簡単ではありません。

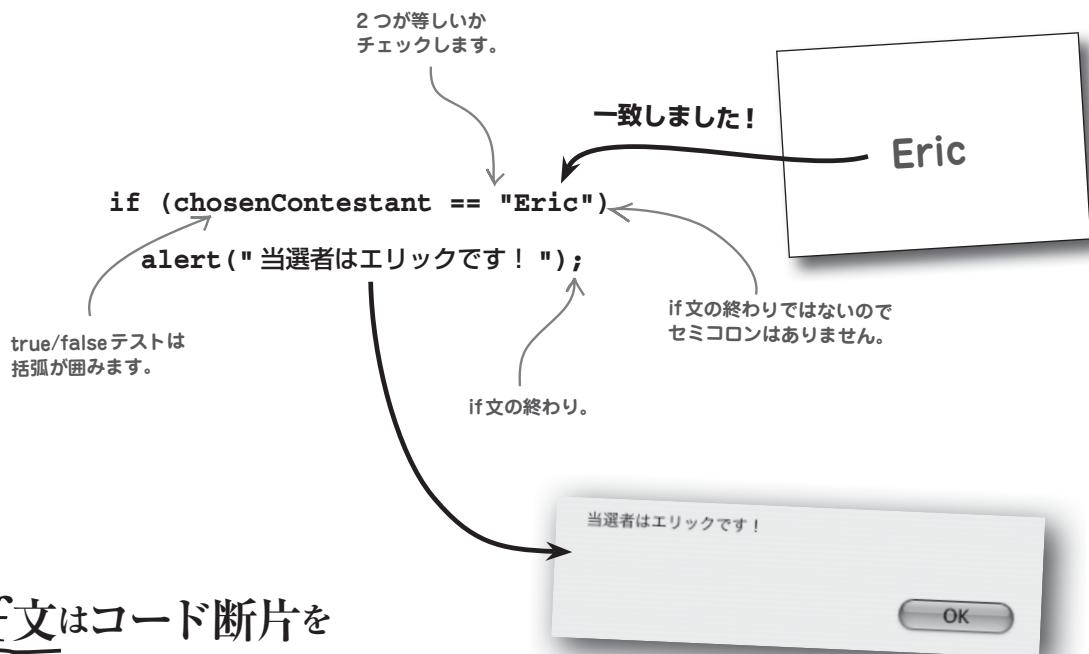


"if" これが真ならば... 何かを実行する

JavaScriptは情報処理と意思決定を得意としています。if文を使うのは得意技のひとつです。if文を使うと簡単な決定を下すことができます。テスト結果のtrue/falseをもとにJavaScriptのコード断片の実行を制御します。



ゲームショーをJavaScriptのif文の眼鏡で眺めると、
以下のようなコードになります。



if文はコード断片を
条件に応じて実行するための
優れた手段です。

if文は条件を評価し、結果に応じてアクションを実行します

if文はどれも同じ形式になります。この形式はiRockにクッキーを追加したときにすでに見ていますね。if文を分解すると以下のようになります。

このコードはtrueかfalseか評価されなければなりません。

if { + () + テスト条件 } +)

文 ;

読みやすくするために
インデントします。
インデントされた文は
ifの一部です。

if文の形式について注意することはそれほどありません。コード断片を1行だけしか実行できないこと、ifとテスト条件の次の行をインデントすることなどです。これは厳しく要求されているわけではなく、インデントしておけばif文の一部であることが一目でわかるメリットがあるということです。if文を書くための手順を以下に示します。

- ❶ true/false テスト条件を括弧で囲む。
- ❷ 次の行のコードを空白でインデントする。
- ❸ テスト条件がtrueのとき実行するコードを書く。



エクササイズ

以下のif文とそれにふさわしいアクションを対応づけてください。

```
if (hungry)
if (countDown == 0)
if (donutString.indexOf("ダース") != -1)
if (testScore > 90)
if (!guilty)
if (winner)
if (navigator.cookieEnabled)
```

```
numDonuts *= 12;
userName = readCookie("irock_username");
awardPrize();
goEat();
alert("ヒューストン、離陸しました。");
alert("無罪です！");
grade = "A";
```



エクササイズの 答え

以下のif文とそれにふさわしいアクションを対応づけてください。

if (hungry)

if (countDown == 0)

if (donutString.indexOf("ダース") != -1)

if (testScore > 90)

if (!guilty)

if (winner)

if (navigator.cookieEnabled)

文字列に「ダース」が
含まれていればtrue。

numDonuts *= 12;

userName = readCookie("irock_username");

awardPrize();

goEat();

alert("ヒューストン、離陸しました。");

alert("無罪です！");

grade = "A";

!guilty は guilty でないを
意味するので、guilty が
falseだと true。

ブラウザでクッキーが
有効なら true。

あれかこれか、どちらか
選びたいんだけど、
どうしたらしいの？

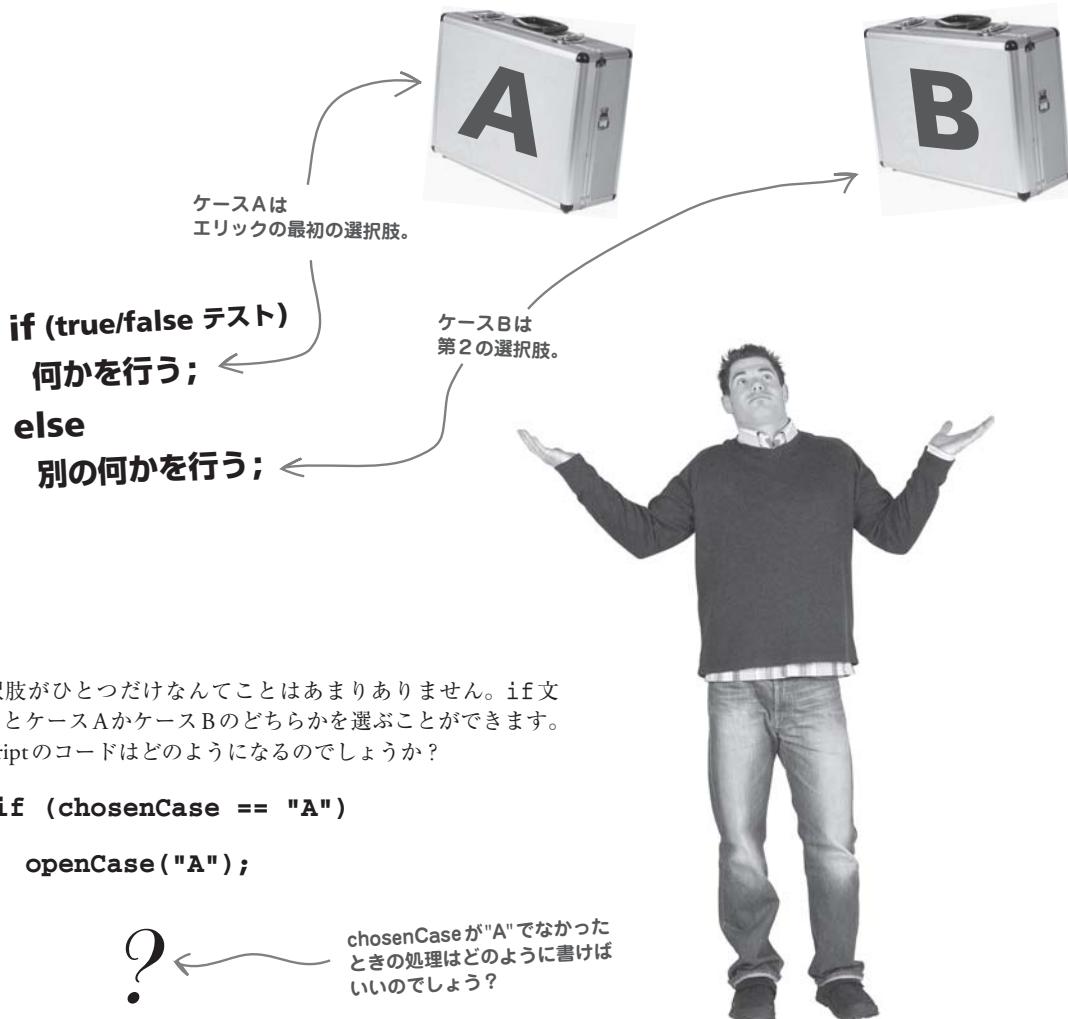


これを行う ... さもなければ。

JavaScriptは何でもできると考えていたら、普通でないと
思われるかもしれません。実際には、選択肢が複数あることは
けして特別ではありません。チョコレートかバニラか、
カフェイン抜きかレギュラーか、あれかこれかの選択は
たくさんあるように思えます。このため if 文は調整できるのです。
ある決定を行い、結果に従って 2 つの選択肢のどちらかを
実行します。あるいはそれ以上のことも可能なのです。

ifを使って2つのどちらかを選ぶ

`if`文を調整すると2つの選択肢から選ぶことができます。 「取引したいでショー」に戻ると、エリックはこれと似たような状況に置かれていました。2つの選択肢のどちらかを選ばなければいけないのです。



複数の if を使って複数の決定ができます

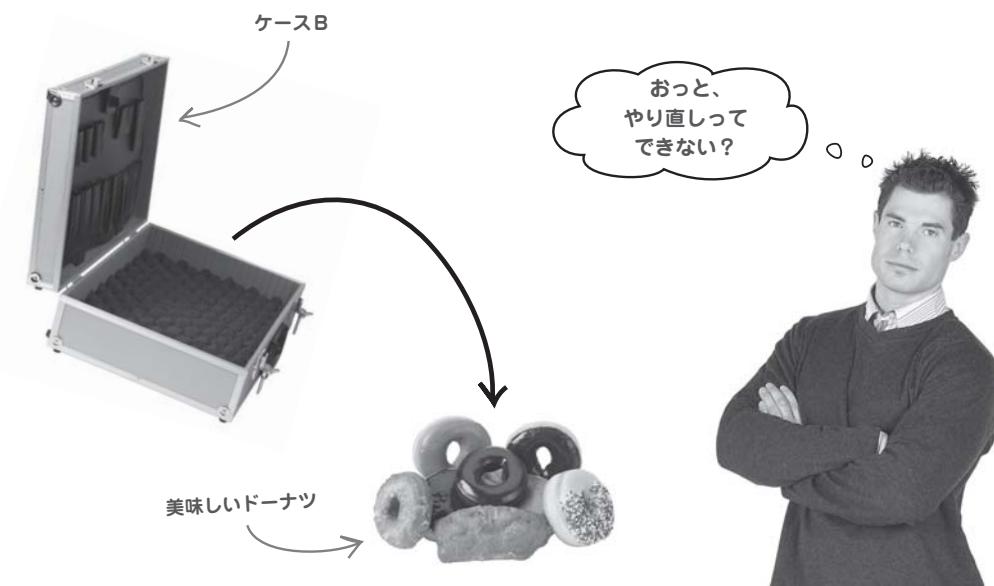
`if` 文を使って複数のアクションから選ぶことは `if/else` 文を使うことを意味します。`true/false` テストでもうひとつのコード断片を実行できるようになります。つまり、テスト結果が `true` のとき最初のコード断片が実行され、そうでないとき (`else`) はもうひとつのコード断片が実行されるということです。

```
if (chosenCase == "A")
    openCase("A");
else
    openCase("B");
```

if/else 文は 2 つの選択肢で構成されます。それぞれテスト条件の値に対応しています。

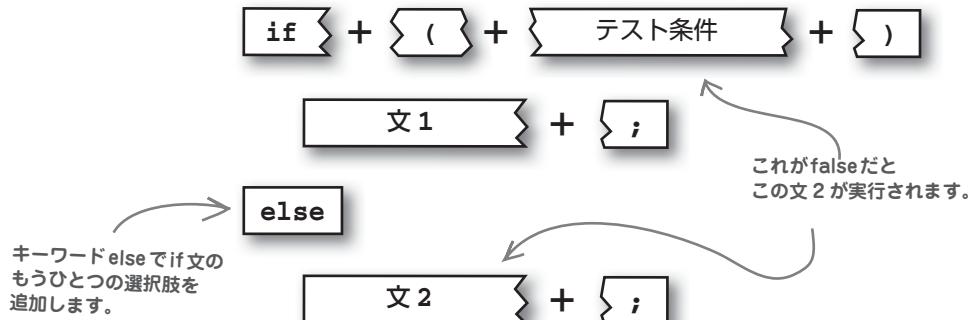
True
False

エリックはケース B を選びました。これは変数 `chosenCase` が "B" になることを意味します。最初のテスト条件は `false` なので `if/else` 文の `else` のコードが実行されます。残念ながらエリックが選んだケース B には彼が期待していた札束ではなくドーナツが入っていました。



if文にelseを追加する

if/else文の形式はif文にとても似ています。テスト条件がfalseの場合キーワードelseの後に続く別のコード断片が実行されます。



以下の3ステップでif文に2番目のアクションを追加できます。

- ① 最初のアクション文の後にキーワードelseを追加する。
- ② 次の行をスペースでインデントして読みやすくする。
- ③ テスト条件がfalseのとき実行されるコードを書く。

素朴な疑問に

答えます

Q: if文は括弧の後にセミコロンがないのはなぜですか？

A: JavaScriptの文法では、すべての文はセミコロンで終わる必要があります。if文も例外ではありません。しかしif文はif(テスト条件)だけでなく、テスト条件がtrueの場合に実行されるコードも含まれます。これらのコードはセミコロンで終わります。このようにif文の構成をよく理解すれば、if文がセミコロンで終わると見えるわけです。

Q: else節がないif文でテスト条件がfalseのとき、どうなりますか？

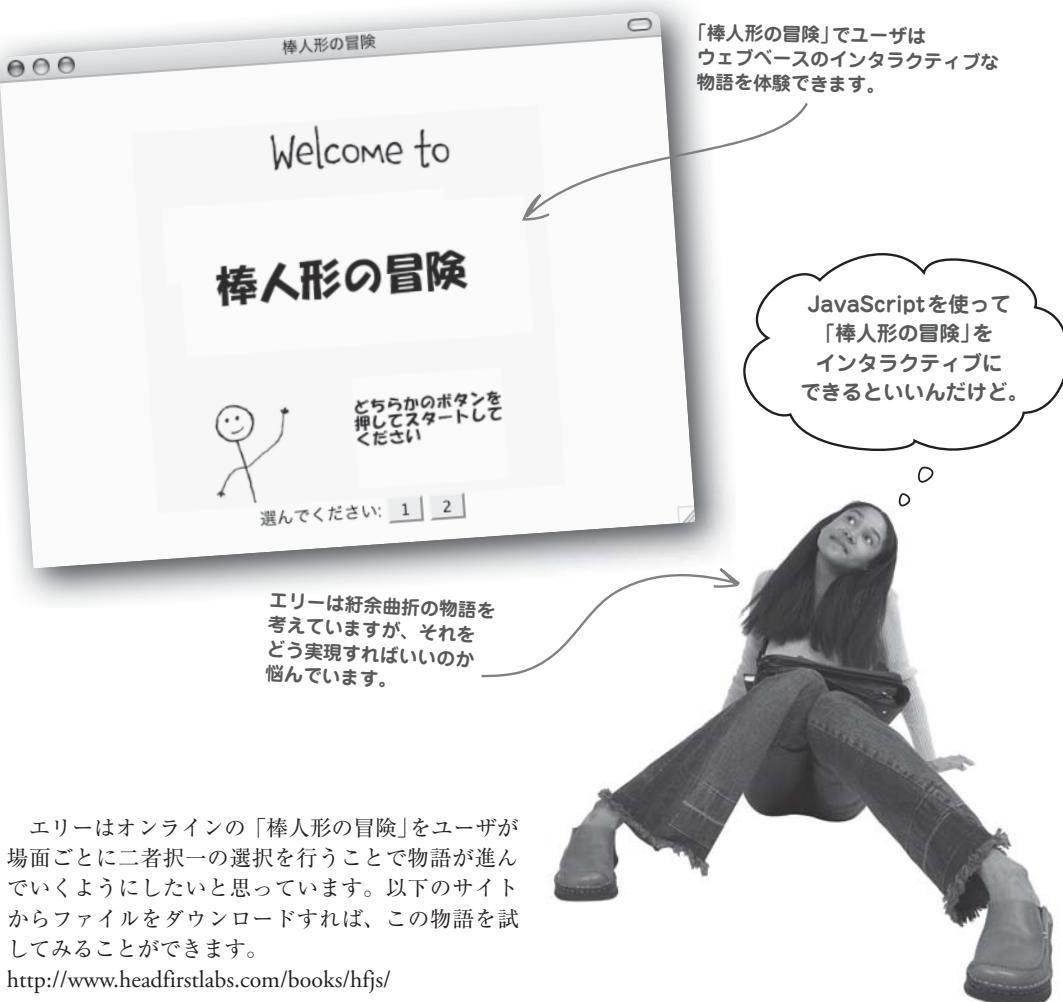
A: 何も起きません。この場合、テスト条件の値は文字通り解釈され、何もアクションは実行されません。

Q: 選択肢が3個以上ある場合、elseを複数使うことはできますか？

A: はい。if/else文で3個以上の選択肢をサポートするように構成することは可能ですが、else節を追加するときほど簡単ではありません。こうした状況の場合、if/elseのアプローチは悪くはないのですが、JavaScriptではよりよい構文としてswitch/case文を提供しています。これについては後ほど説明します。

壮大な冒険物語

エリーは「棒人形の冒険」というインターラクティブな冒険物語を作っています。彼女のプロジェクトでは物語の場面ごとに意思決定を行うことになるので、JavaScriptを使って意思決定ができないか、オンラインの冒險をみんなで楽しめるようにするための課題を解決できないかと思っています。

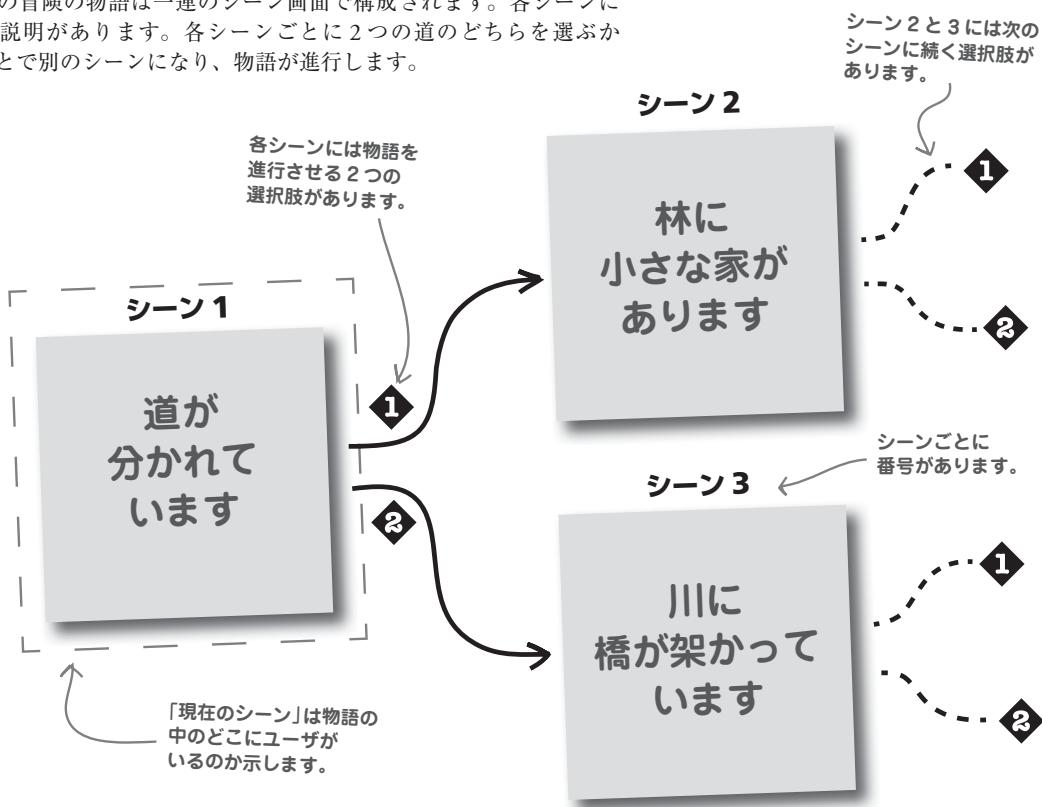


エリーはオンラインの「棒人形の冒険」をユーザが場面ごとに二者択一の選択を行うことで物語が進んでいくようにしたいと思っています。以下のサイトからファイルをダウンロードすれば、この物語を試してみることができます。

<http://www.headfirstlabs.com/books/hfjs/>

冒険をセットアップする

棒人形の冒険の物語は一連のシーン画面で構成されます。各シーンには画像と説明があります。各シーンごとに2つの道のどちらを選ぶか決めてることで別のシーンになり、物語が進行します。



自分で考えてみよう

「棒人形の冒険」の冒頭の3つのシーンで使う意思決定のためのコードをif/else文で書いてください。変数decisionにはユーザの選択がすでに格納されています。変数curSceneに選ばれたシーンが保持されるようにします。

自分で考えてみよう の答え

「棒人形の冒険」の冒頭の3つのシーンで使う意思決定のためのコードをif/else文で書いてください。変数decisionにはユーザーの選択がすでに格納されています。変数curSceneに選ばれたシーンが保持されるようにします。

変数decisionにはシーンごとにユーザーが選択した決定が格納されます。値は1か2になります。

```
if (decision == 1)
```

```
    curScene = 2;
```

```
else
```

```
    curScene = 3;
```

変数curSceneには現在のシーンが保持されます。ユーザーの選択をもとに次のシーンに進みます。

シーン2に進む。

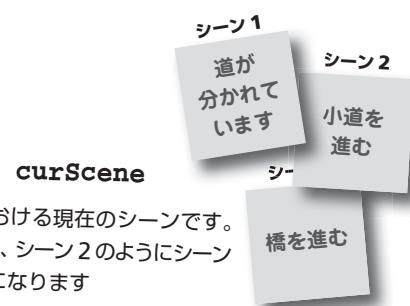
シーン3に進む。

変数を使って物語を進行させる

「棒人形の冒険」で使われている2つの変数を詳しく見ていきましょう。これらの変数はユーザーの決定に応答して物語を進行させる重要な役割を果たしています。



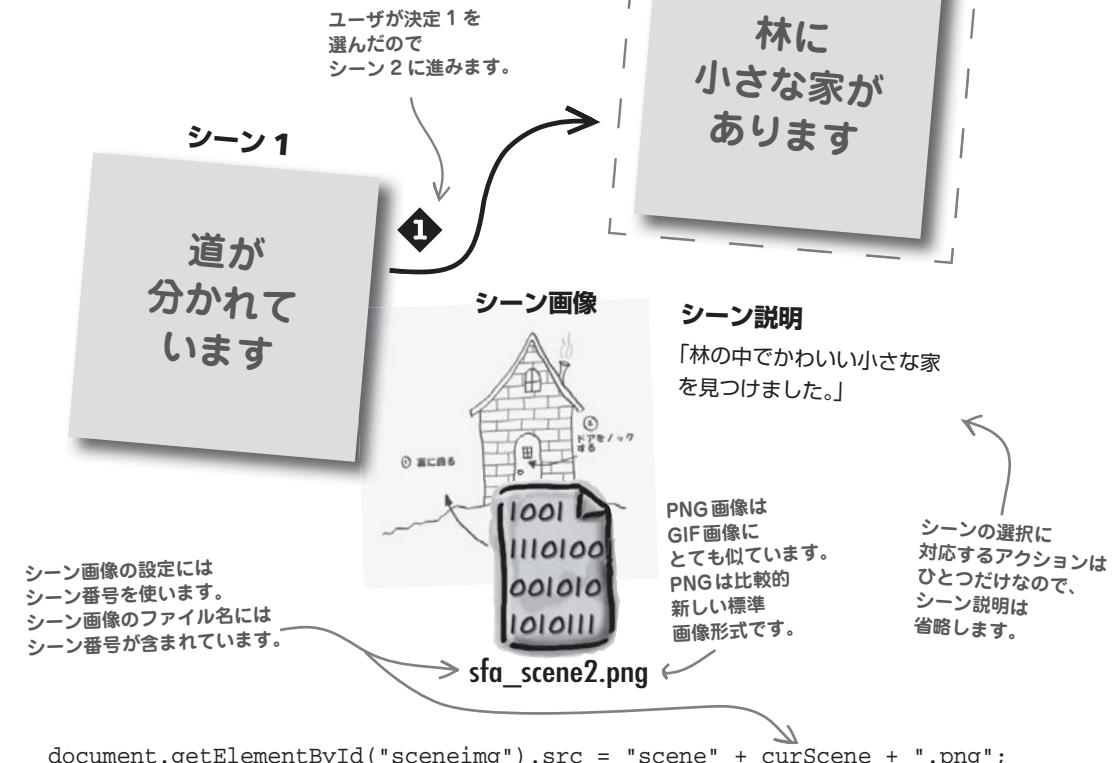
ユーザーが選んだ選択は常に1か2になります。この選択にしたがって物語の次のシーンが決まります。



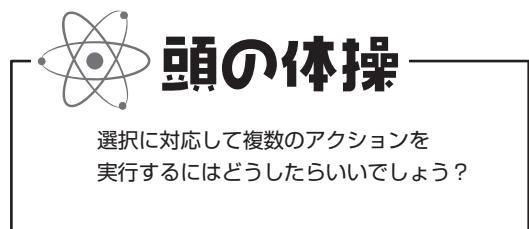
変数のdecisionとcurSceneはユーザーの選択を格納し物語を進行させる判断基準になります。この処理はif/else文のおかげで物語が続く限りシーンごと繰り返されます。

物語の一部が見つかりません

`if/else`文は「棒人形の冒険」の意思決定部分のエンジンとして働きますが、物語全体を語る語り手ではありません。シーンごとに画像と説明テキストがあり、物語の進行とともに内容が変わります。そのためシーンの画像を変更するには現在のシーン番号を変更するだけで十分ですが、シーンの説明テキストには役に立ちません。



`if/else`文の各部分に対応するコード断片しか実行できないと、ひとつのアクションしか実行できません。つまり画像の表示と説明テキストの表示を一度に実行することができないということです。



JavaScriptのアクションを複数にする

エリーはif/elseコードのそれぞれの部分で複数のアクションを実行させる必要があります。以下の2行でシーン番号とシーンの説明テキストの両方を変更しないと、物語を進行させられません。

```
document.getElementById("scenelimg").src = "scene" + curScene + ".png";  
alert(message);
```

選択されたシーンに対応する
シーン説明テキストを表示します。

JavaScriptはひとつのコード断片しか実行できないのですが、ここで複数のアクションを実行させるにはどうすればいいでしょう。複文を使えばコードのかたまりをあたかもひとつのコード断片のようにスクリプトの中で扱うことができます。複文を作るには一連の文を波括弧({})で囲むだけです。

```
doThis();  
doThat();  
doSomethingElse();
```

= 3つの文

コードを囲む
波括弧を対応
させます。

```
{  
    doThis();  
    doThat();  
    doSomethingElse();  
}
```

= 1つの文

複文になると、if/else文のそれぞれのアクションで複数の文を実行させることができます。

```
if (chosenDoor == "A") {  
    prize = "donuts;"  
    alert("賞品はドーナツです！");  
}  
else {  
    prize = "pet rock;"  
    alert("賞品はiRockです！");  
}
```

if/else文のアクションでは
複数の文を実行できます。

選択されたシーンにあわせて
シーン画像を変更します。



なるほど。
複文を使えば
コードのかたまりをひとつの
コード断片のように
扱えるね。



素朴な疑問に 答えます

：「棒人形の冒険」では物語を展開するために変数をどのように使っているのでしょうか？

A: どのシーンにおいても、`curScene`に現在のシーン番号が保持されています。各シーンでは画像と説明が表示され、ユーザが次のシーンを2つの選択肢から選べるようになっています。`decision`にはユーザが選んだ選択肢の値(1または2)が保持されます。選択の結果、`decision`の値と`curScene`の値をもとに次の新しいシーンが決まります。具体的には、シーン画像は`curScene`の値を使って変更され、シーンの説明メッセージはアラートボックスを使って表示されます。

：なぜ複文によって複数の文をひとつになることが重要なのでしょうか？

A: JavaScript言語の部品の多くは单文という概念を中心に構成されています。これは飛行機に搭乗する際手荷物が2つに制限されているようなものです。2つの手荷物の中に何を詰め込んでも、手荷物が2つであることには変わりありませんよね。複文はちょうどこの手荷物のようなものです。複文というコンテナの中に複数の文を詰め込んでも、スクリプトの他の部分はこれをひとつ文として認識します。

：なぜ複文はセミコロンで終わらないのですか？

A: セミコロンはあくまで個々の文のために取られるのであって、複文のためではありません。複文の中に現れる単文はセミコロンで終わる必要がありますが、複文自体には必要がありません。

： 関数は複文ですか？

A1: いい質問です。答えはイエスです。関数の中のコードは波括弧で囲まれます。今のところ、関数をデータのやり取りができる大きな複文と考えてもかまいません。



自分で考えてみよう

「棒人形の冒険」の最初の if/else のためのコードを書いてください。
複文を使ってシーン番号とシーンの説明テキストの両方を設定します。

自分で考えてみよう の答え

「棒人形の冒険」の最初の if/else のためのコードを書いてください。
複文を使ってシーン番号とシーンの説明テキストの両方を設定します。

ユーザの選択をもとに
現在のシーンの番号を
調整します。

新しいシーンに
対応したシーン
説明メッセージを
設定します。

波括弧を開いて
複文を開始
します。

複文のコードは
インデントすると
読みやすくなります。

```
if (decision == 1) {  
    curScene = 2;  
    message = "林の中でかわいい小さな家を見つけました。";  
}  
  
else {  
    curScene = 3;  
    message = "橋に立ち、穏やかな流れを眺めます。";  
}  
}  
} // シーン 3 のシーン説明  
// メッセージを設定します。
```

重要ポイント

- if文を使うとJavaScriptのひとつのコード断片を条件に応じて実行できます。
- if文のテスト条件は常に結果がtrueかfalseになる必要があります。
- if/else文を使って2つあるJavaScriptコード断片のどちらかを条件に応じて実行させることができます。
- 複文を使うとJavaScriptの複数のコード断片をひとつのコード断片のかわりに使うことができます。
- 複文を作るには複数の文を波括弧({})で囲みます。
- 複文を使うとif文やif/else文のアクション部分で複数の文を実行できます。

冒険のはじまり

`if/else`の中を複文にしたので「棒人形の冒険」をインタラクティブなオンライン物語としてはじめることができます。オンライン冒険の全機能が動くようになったのです。

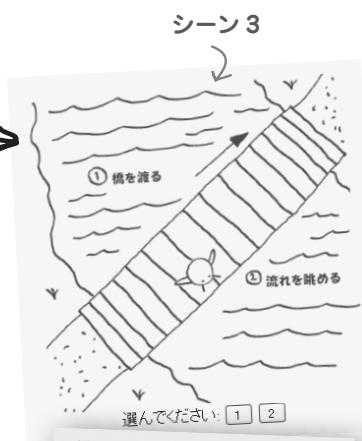
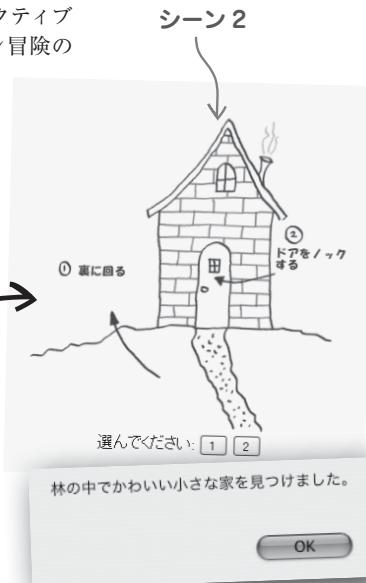


ボタン1を選択すると
シーン2に進みます。

ボタン2を選択すると
シーン3に進みます。

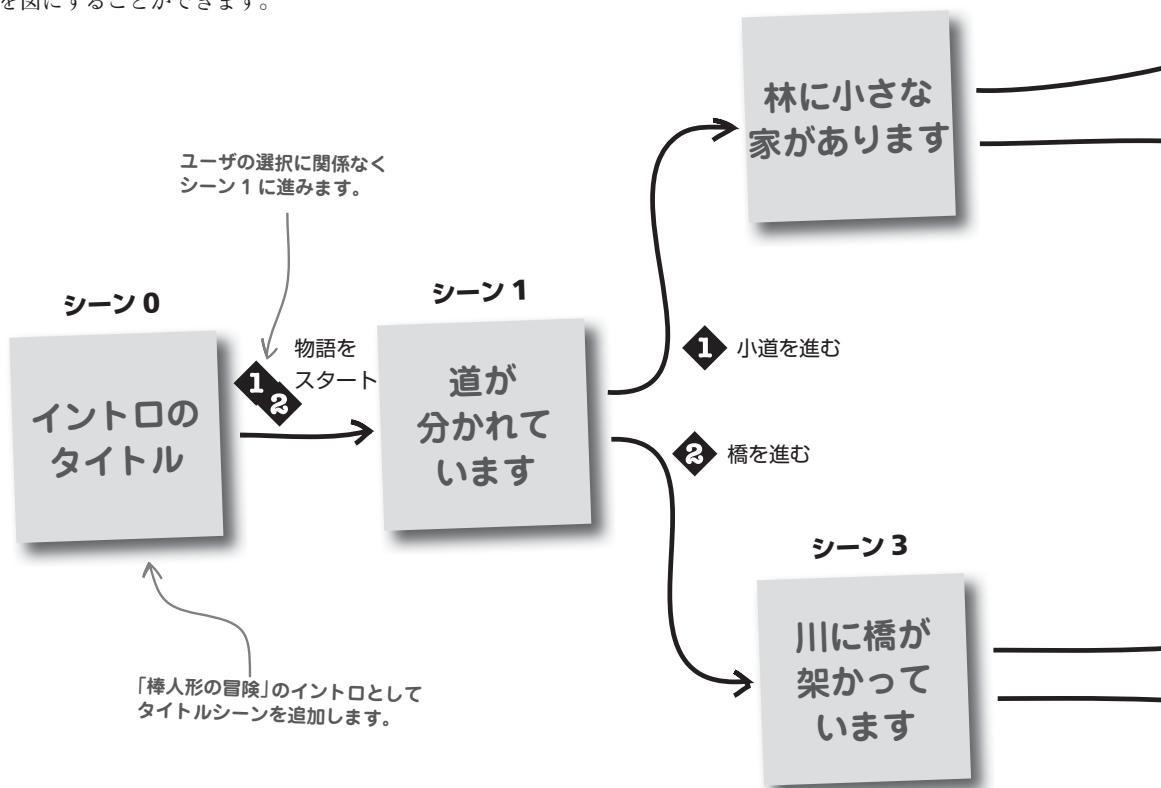
シーン説明テキストは
アラートボックスに
表示されます。

やったあ、
物語の最初のシーンは
いい感じね。

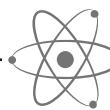
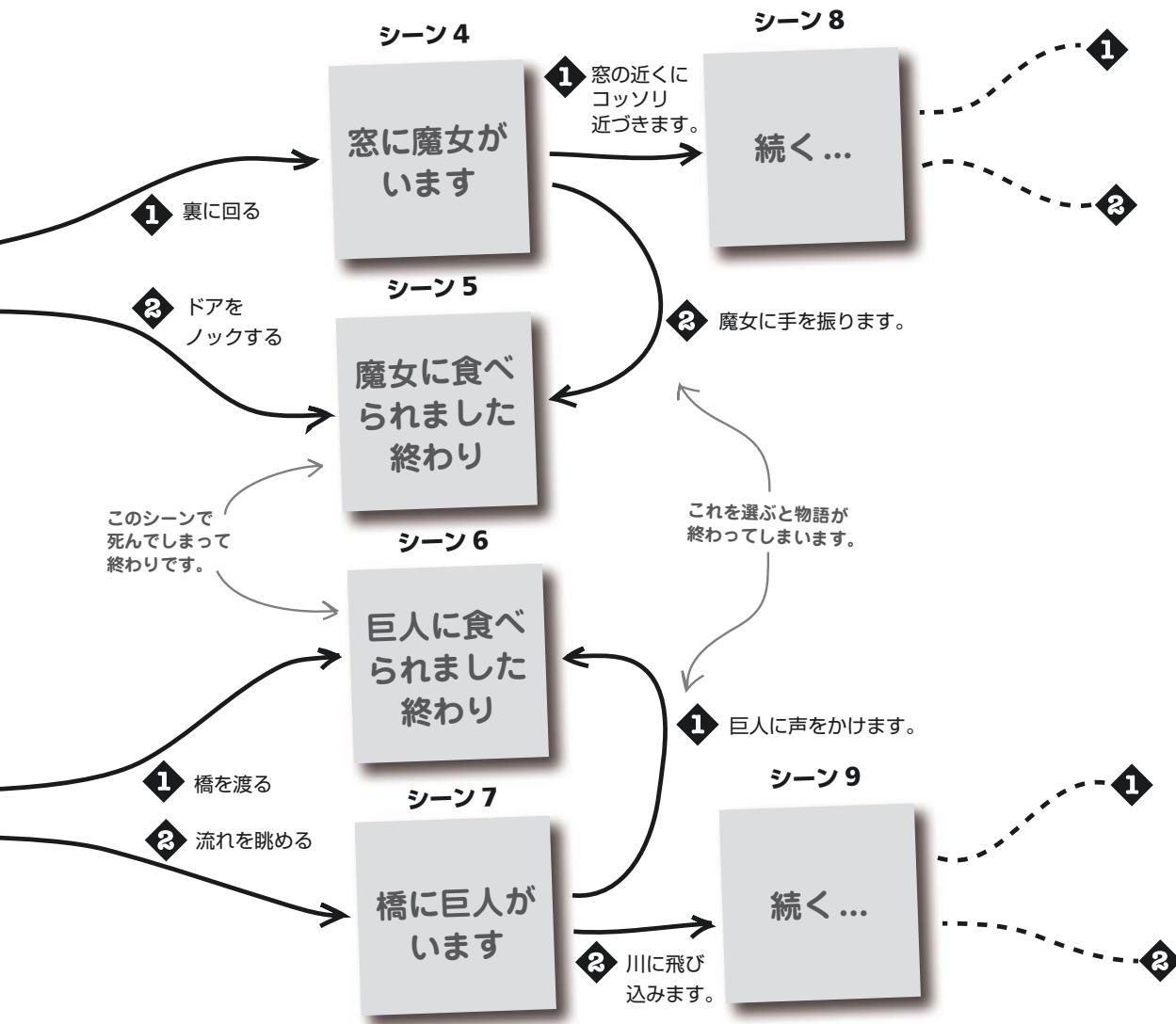


冒険全体の構成を見てみましょう

選択がひとつだけではインタラクティブな物語として面白くありません。エリーは「棒人形の冒険」にいくつかのシーンを追加してもっと好奇心をそそるようにするつもりです。これらのシーンは決定ツリーになるので、物語の構成を図にすることができます。



エリーはシーンを追加して物語を波瀾万丈にするだけでなく、物語の最初のシーン（シーン 1）の前にタイトルシーンを追加しました。このタイトルシーン（シーン 0）は他のシーンと違って選択 1 と選択 2 のどちらを選んでもシーン 1 に移ります。つまりシーン 0 は物語の枝ではなくオープニングクレジットのようなものなのです。



頭の体操

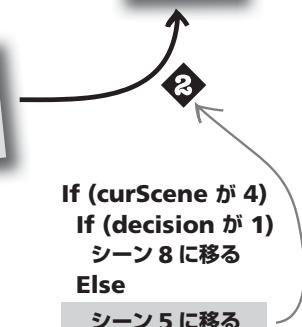
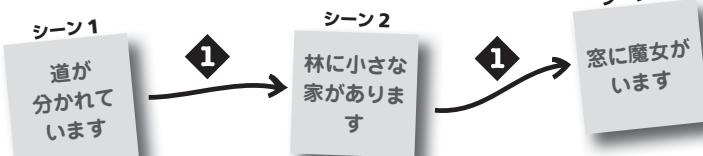
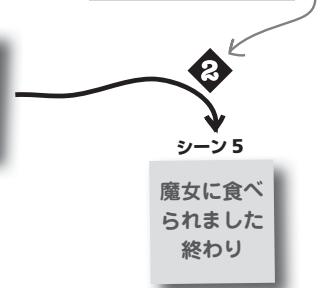
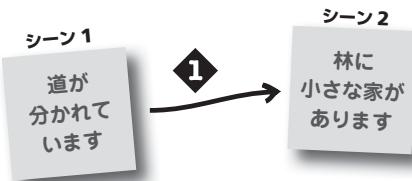
「棒人形の冒険」の決定ツリーはif/else文を使っているように見えませんか？

if/else を使って段階的決定にする

「棒人形の冒険」の決定ツリーにあるそれぞれの決定には選択肢が2つしかありませんが、後になってからの決定はその前の決定に依存していることにエラーは気づきました。たとえば、物語のはじまりからシーン5にたどり着くにはユーザは以下の経路をたどる必要があります。

```
If (curScene が 2)
If (decision が 1)
    シーン 4 に移る
Else
```

シーン 5 に移る



if/else 文の中に
別の if/else 文を書いても
大丈夫かしら？



ユーザが選んだ選択肢の情報だけでは次のシーンがどれなのか決定するのに十分ではありません。ひとつの解決策としては、最初に現在のシーンをチェックするやり方と同じように複数の if/else 文を使ってユーザの選択をもとにアクションを決定する方法です。しかしこの段階的な意思決定のアプローチでは if の中に if ができるので、奇怪な構成になってしまいそうです。

よくよく考えてみれば、私たちはいつも段階的決定を行っていることに気づくでしょう。「ポテトをお付けしますか？」と訊かれて返事したことがありますよね？ この質問はサラダを注文した後に訊かれることは滅多にありません。通常は「チーズバーガーひとつ」と注文した後に訊かれるわけです。これは段階的決定になっています。後の質問（ポテト？）は前の質問に対する返事（チーズバーガー、サラダ？）に依存しているからです。

ifの中に別のifを書くことができます

JavaScriptではifの中にifを書くのは文法的に何の問題もありません。if文はただの文なので、if文のアクション部分に別のif文を使えるのです。つまりひとつの質問の後に別の質問を続けられるということです。ifの中にifがあるのを入れ子になったif文といいます。

```
if (order == "チーズバーガー") {  
    if (wantFries)  
        order += " ポテト";  
    }  
    else if (order == "サラダ") {  
        if (wantFruit)  
            order += " フルーツ";  
    }  
}  
  
order = order + ... を使って  
注文を追加します。出力はサラダ フルーツか  
チーズバーガー ポテトになります。
```



自分で考えてみよう

「棒人形の冒険」のシーン 0 とシーン 1 の意思決定のコードを書いてください。
必要であれば入れ子になった if 文、if/else 文を使ってください。

自分で考えてみよう の答え

「棒人形の冒険」のシーン 0 とシーン 1 の意思決定のコードを書いてください。
必要であれば入れ子になった if 文、if/else 文を使ってください。

現在のシーンが
シーン 0 で
なければ、シーン 1
でないかチェック
します。

シーン 0 はかならず
シーン 1 に進むので
if 文は必要ありません。

シーン 1 の説明メッセージを
設定します。

```
if (curScene == 0) {  
    curScene = 1; ←  
    message = "旅は分かれ道から始まります。";  
}  
  
else if (curScene == 1) {  
    if (decision == 1) {  
        curScene = 2;  
        message = "林の中でかわいい小さな家を見つけました。";  
    }  
    else { ←  
        curScene = 3;  
        message = "橋に立ち、穏やかな流れを眺めます。";  
    } ←  
}  
}
```

入れ子になった
if/else 文を使って
シーン 1 での
ユーザの選択を
処理します。

ある文が別の文に入れ子になっている
とき、インデントすると読みやすく
なります。

波括弧の開くと閉じるを
対応させる必要があります。

ページを関数で制御する

「棒人形の冒険」のウェブページにある2つのボタン（1と2）を使ってユーザは物語を進行させます。ユーザが選んだボタンをクリックするとchangeScene()が呼び出され、クリックされたボタンをもとにシーンが変わります。

```

...
    function changeScene(option) {
        ...
    }
</script>
</head>
<body>
    <div style="margin-top:100px; text-align:center">
        <br />
        Please choose:
        <input type="button" id="decision1" value="1" onclick="changeScene(1)" />
        <input type="button" id="decision2" value="2" onclick="changeScene(2)" />
    </div>
</body>
</html>

```

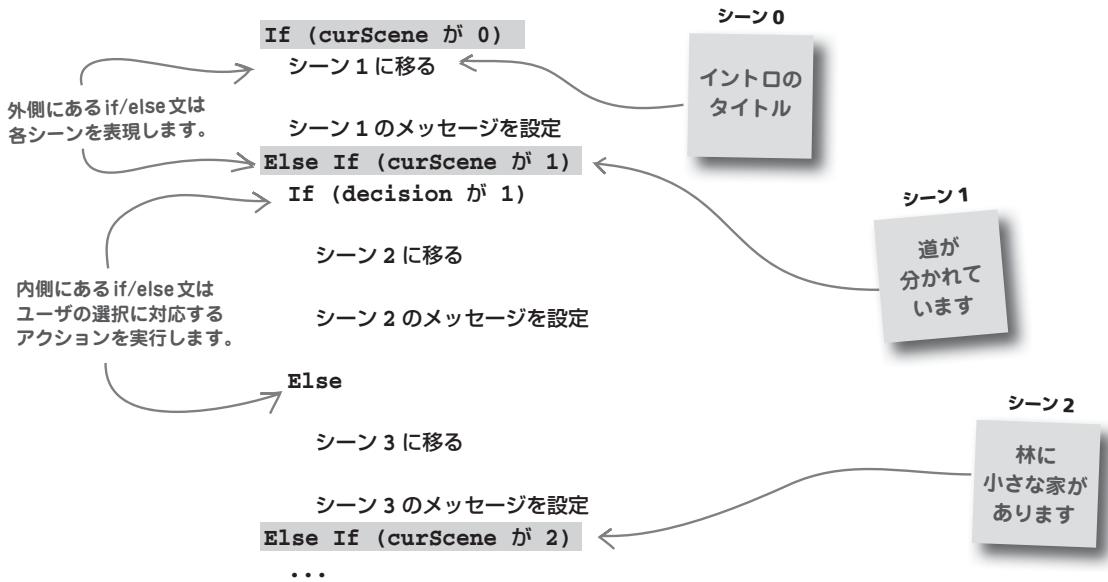
changeScene()はユーザが選んだボタンの値（1か2）を引数で受け取ります。この関数がシーンを変えるのに必要とするのはこの情報断片だけです。changeScene()は以下の3つの処理を行います。

ウェブページにある2つのボタンで次のシーンをどちらにするか決めます。

- ❶ 変数curSceneに新しいシーン番号を設定する。
- ❷ 変数messageに新しいシーン説明テキストを設定する。
- ❸ curSceneの値をもとにシーン画像を変更し、シーン説明テキストのメッセージを表示する。

疑似コードを使って冒険の綿密な計画をたてる

エリーは「棒人形の冒険」の決定ツリーをJavaScriptコードで実装するためにchangeScene()を組み立てるうまい方法をみつけました。しかし非常に多くの決定があるのでコーディングをはじめると混乱してしまうでしょう。そこでコーディングの前に決定ツリーを疑似コードで書くのをお勧めします。疑似コードはスクリプトのコードを簡潔で読みやすい方法で表現します。いったん疑似コードを作つておくとJavaScriptのコードを書くのがより明確で間違いにくくなります。



素朴な疑問に答えます

Q: 疑似コードはかなりJavaScriptコードと似ていますが、なぜわざわざ書くのですか？

A: わざわざ書く必要はありませんが、ロジックの複雑な木をJavaScriptコードに変換する作業を簡潔にするためにあり、エラーが発生する危険性を最小限にする効果があります。疑似コードはJavaScriptコードほど詳細なわけではないので、波括弧やセミコロンが正しい位置にあるか気にするのではなく、あるシーンから別のシーンへ続くロジックに集中すればよいのです。疑似コードに慣れたら、これをJavaScriptコードに翻訳するのはかなり自動的にできます。

： if文が入れ子のときは、波括弧を使う必要がありますか？

A: いいえ。if 文の中に別の if 文だけがあって他のコードがない場合、複文にする必要がないので、波括弧で囲まずにそのままですかいません。しかし、if 文が複雑に入れ子になっている場合、厳密にはそうする必要はありませんが、あえて波括弧を使った方が、入れ子が明確になるのでプラスになります。



JavaScript マグネット

「棒人形の冒険」のchangeScene()でコード断片の一部が消えてしまいました。152ページの図をもとにマグネットを使ってコードを完成させてください。シーンを決定するコードのすべてが書かれているわけではありません。いくつかのシーンは意図的に省略されています。

```
function changeScene(option) {
    var message = "";

    (curScene == 0) {
        curScene =      ;
        message = "旅は分かれ道から始まります。";
    }

    ...
    (curScene == 3) {
        (option == 1) {
            curScene =      ;
            message = "残念ですが、橋の向こう側には巨人がいました。" + "お屋ご飯になりました。";
        }
        {
            curScene =      ;
            message = "流れを眺めていると巨人がやってきました。";
        }
    }
    (curScene == 4) {
        if (option == 1) {
            curScene =      ;
        }
        {
            curScene =      ;
            message = "残念ですが、あなたはシチューの材料になりました。";
        }
    }
    ...
    document.getElementById("scenearrow").src = "scene" + curScene + ".png";
    alert(message);
}
```





JavaScriptマグネットの答え

「棒人形の冒険」のchangeScene()でコード断片の一部が消えてしまいました。152ページの図をもとにマグネットを使ってコードを完成させてください。シーンを決定するコードのすべてが書かれているわけではありません。いくつかのシーンは意図的に省略されています。

```

function changeScene(option) {
    var message = "";

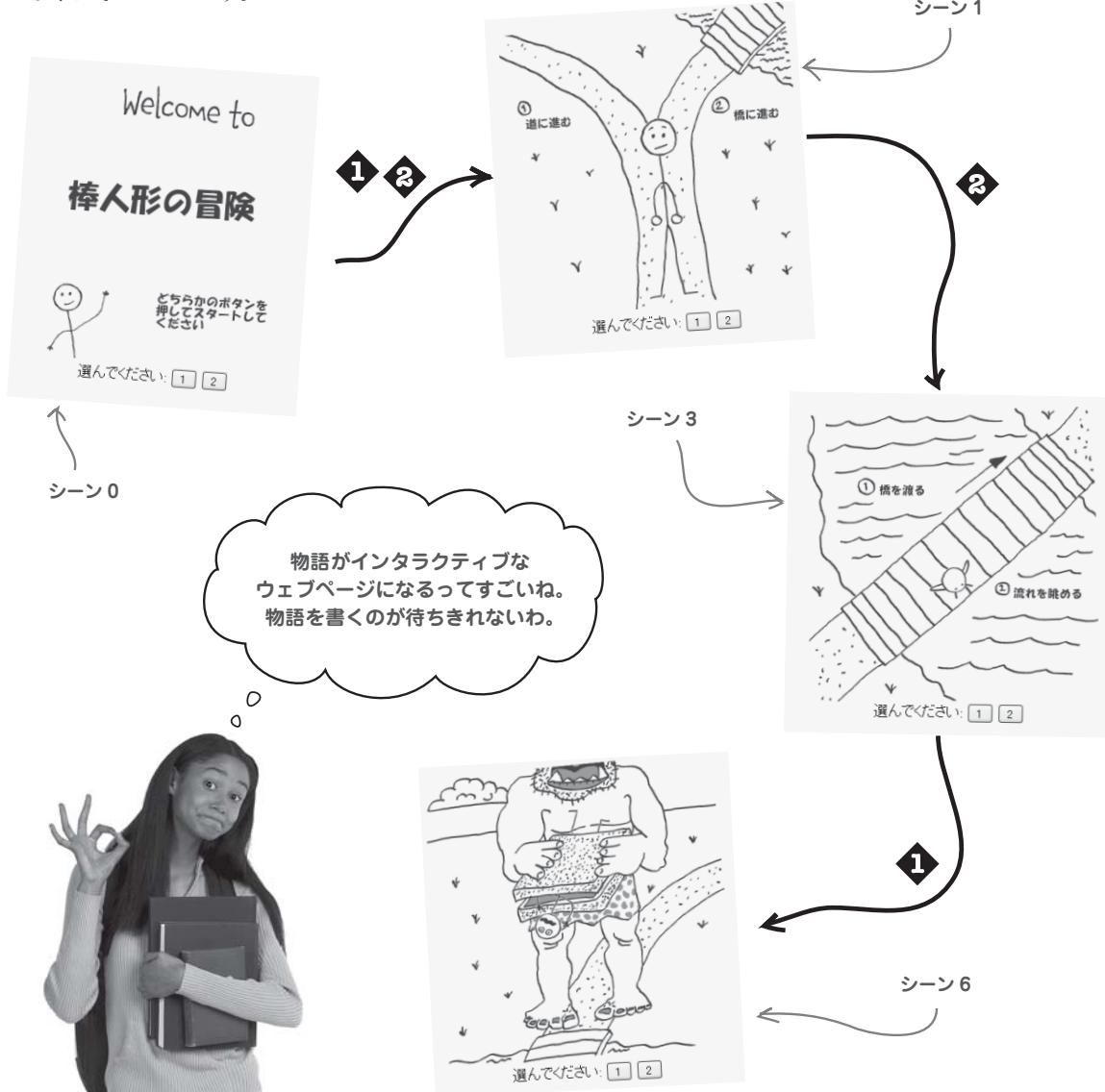
    if (curScene == 0) {
        curScene = 1;
        message = "旅は分かれ道から始まります。";
    }

    ...
    else if (curScene == 3) {
        (option == 1) {
            if curScene == 6;
            message = "残念ですが、橋の向こう側には巨人がいました。" + "お昼ご飯になりました。";
        }
        else {
            curScene = 7;
            message = "流れを眺めていると巨人がやってきました。";
        }
    }
    else if (curScene == 4) {
        if (option == 1) {
            curScene = 8;
        } else {
            curScene = 5;
            message = "残念ですが、あなたはシチューの材料になりました。";
        }
    }
    ...
    document.getElementById("sceneimg").src = "scene" + curScene + ".png";
    alert(message);
}

```

「棒人形の冒険」を続ける

「棒人形の冒険」のスクリプトは決定ツリー全体を反映しています。いくつかの経路で物語全体をたどることができます。以下はそのひとつです。



棒人形の不具合

残念ながらエリーは「棒人形の冒険」の不具合を見つけました。ページを友達何人に見てももらったところ、空のメッセージのウィンドウが表示されるという報告がいくつかあったからです。この「お化けウインドウ」は、冒険が終わった後、新しく冒険を始めるときに表示されます。どうやら他のシーンからシーン0に移るときに何か問題がありそうです。



シーン0に戻る経路があるシーンはシーン5とシーン6の2つしかないことわかりました。この2つのシーンは物語の最後を表現しているのでシーン0に戻ります。物語の最後になってから最初に戻れるのは意味があります。そこでエリーはchangeScene()の中で2つのシーンからシーン0に変える処理を行っているコードを調べてみました。

```
else if (curScene == 5) {  
    curScene = 0; <  
}  
else if (curScene == 6) {  
    curScene = 0; <  
}
```

シーン5と6ではcurSceneを設定していますが、messageには何も設定していません。

この簡単なコードには何も問題がないので、changeScene()の終わり近くにあるコードを見てみましょう。シーン画像を変更しシーン説明テキストを表示している箇所です。

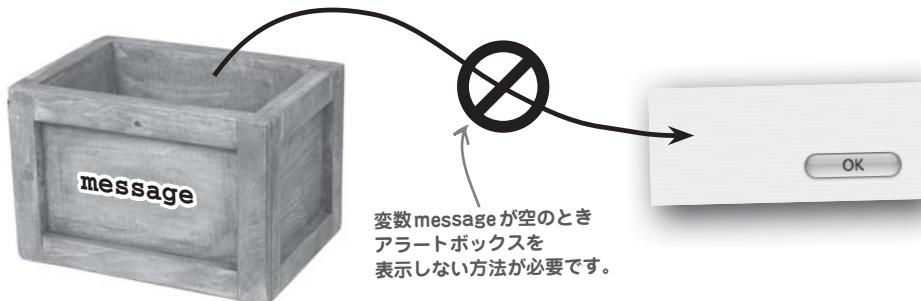
```
document.getElementById("sceneimg").src = "scene" + curScene + ".png";  
alert(message);
```

messageに格納された
シーン説明テキストを表示します。



!= あのさ、言いたいことは何もないんだけど

「棒人形の冒険」のコードの問題は、シーン0のように表示すべきメッセージがないときでも常にシーン説明のメッセージがアラートボックスで表示される点に問題があります。変数messageが実際にシーン説明テキストを含んでいるかチェックするにはどうすればいいでしょうか？



シーン6以外だったら
trueになります。

この問題は、アラートボックスを表示する前に変数messageが空テキスト ("") でないかチェックすれば解決します。変数messageが空テキストと等しくないときだけアラートボックスを表示するのです。なんだか後ろ向きの解決策みたいですが、アラートボックスを表示していいかどうかを true/false テストで判断するわけです。

等号演算子 (==) は2つの項が等しいかどうかチェックするのに使いますが、これと同じように不等号演算子 (!=) は2つの項が異なるかどうかチェックするのに使います。

```
if (curScene != 6)
    alert("幸運にも、巨人に食べられずにすみました。");
```



自分で考えてみよう

「棒人形の冒険」の説明メッセージをアラートボックスに表示するコードを書き直してください。メッセージに実際にテキストデータがあるときだけアラートボックスを表示するようにします。



自分で考えてみよう の答え

messageに空でない
テキスト文字列が
含まれているときだけ
trueになります。

```
if (message != "")  
    alert(message);
```

「棒人形の冒険」の説明メッセージをアラートボックスに表示するコードを書き直してください。メッセージに実際にテキストデータがあるときだけアラートボックスを表示するようにします。

比較演算子を使って決定を作り込む

JavaScript コードのテスト条件で意思決定をするときに使える
比較演算子は等号と不等号だけではありません。

$x > y$ より大きければ true。
それ以外は false。

等しい

$x == y$

x と y が等しいとき true。
それ以外は false。

等しくない

$x != y$

x と y が等しくないとき true。
それ以外は false。

より小さい

$x < y$

x が y より小さければ true。
それ以外は false。

より大きい

$x > y$

否定

$!x$

x が true なら false。
 x が false なら true。

以下

$x <= y$

x が y 以下なら true。
それ以外は false。

以上

$x >= y$

x が y 以上なら true。
それ以外は false。

JavaScript の比較演算子などは式を作るため
に使われます。式は JavaScript のコードの断片
を組み合わせたもので、評価されてひとつの
値になります。比較演算子をもとに作られた式
は論理値 (true/false) を返します。これは
if/else 文を使った意思決定ロジックを作る
のに便利です。

= と == の動作は異なります。

2つの値が等しいか比較するときには
= ではなく == を使います。=を使うと
代入が実行されるので、思わぬバグに
なってしまいます。



要注意！

素朴な疑問に

答えます

Q: 否定演算子はなぜひとつの値しか取らないのですか？

A: ほとんどの比較演算子は項を2つ取りますが、否定演算子の取る項はひとつだけです。仕事の内容は非常に簡単で、項の値を反転させるだけです。つまり、trueはfalseに、falseはtrueになります。

Q: 否定演算子が比較ではなく値に対して使われているのを見ました。これはどのように動作するのですか？

A: 比較でない値に対して否定演算子を使うコードは、JavaScriptが値の真偽を決定する仕掛けを利用して

います。比較が期待されている状況で、比較でない値を使うとき、null、0、""以外の値は自動的にtrueとみなされます。言い換えると、データが存在すれば、比較の文脈においては値がtrueとみなされるのです。そのため、否定演算子が比較でない値に対して使われると、null、0、""はtrueになり、それ以外の値はすべてfalseになります。

Q: 待ってください、nullって何ですか？

A: nullはJavaScriptの特別な値で、データの不在を示します。9章と10章で取り上げますが、オブジェクトの文脈では重要になります。



エクササイズ

このコードは「棒人形の冒険」に関するポジティブなメッセージを表示することができます。メッセージを完成させるには4つの変数a、b、c、dの値をそれぞれ何にすればいいでしょう？

```
var quote = "";

if (a != 10)
    quote += "Some guy";
else
    quote += "I";
if (b == (a * 3)) {
    if (c < (b / 6))
        quote += " don't care for";
    else if (c >= (b / 5))
        quote += " can't remember";
    else
        quote += " love";
}
else {
    quote += " really hates";
}
if (!d) {
    quote += " Stick Figure";
}
else {
    quote += " Rock, Paper, Scissors";
}

alert(quote + " Adventure!");
```

a =

b =

c =

d =



エクササイズの 答え

このコードは「棒人形の冒険」に関するポジティブなメッセージを表示することができます。メッセージを完成させるには 4 つの変数 a、b、c、d の値をそれぞれ何にすればいいでしょう？

```
var quote = "";
    aは 10 と等しい。
if (a != 10)
    quote += "Some guy";
else
    quote += "I";
if (b == (a * 3)) {
    if (c < (b / 6))
        quote += " don't care for";
    else if (c >= (b / 5))
        quote += " can't remember";
    else
        quote += " love";
}
else {
    quote += " really hates";
}
if (!d) {
    quote += " Stick Figure";
}
else {
    quote += " Rock, Paper, Scissors";
}
alert(quote + " Adventure!");
```

a = 10
b = 30
c = 5
d = false

!dがtrueなので、
dはfalse。

探していたポジティブな
メッセージがこれ。

I love Stick Figure Adventure!

OK

コメントとドキュメンテーション

「棒人形の冒険」の物語は開発中なのでスクリプトのコードは未完成です。たとえばシーン 8 とシーン 9 はどちらも「続く……」のシーンになっていて、エラーのクリエイティブな作業を必要としています。そこで未完成なコードの部分をマークして後で詳細を作り込むことを忘れないようにしておくと便利です。JavaScript のコメントを使うと、コードの実行され方に影響を与えずにコードに注記を加えることができます。

コメントは 2 つの
スラッシュで開始します。

// { + {コメント}

コメントのテキストは
JavaScript インタープリタ
によってすべて無視されます。

//で始まるJavaScriptのコメント

//で始まるコメントはスラッシュから行末までがコメントになります。
コードを追加する必要がある部分にこのコメントを使うといいでしょう。

```
else if (curScene == 8) {  
    // 続く  
}  
else if (curScene == 9) {  
    // 続く  
}
```

このコードの行は無視されます。

シーン8

続く...

シーン9

続く...

コメントは単なる埋め草ではありません。コードの構成を向上させ理解しやすくするコードのドキュメンテーションとして使われます。コード断片がどう動くか今は理解していても、将来も同じように記憶しているとは限りません。誰か他の人があなたのコードを引き継ぐこともあるので、コードの動作を注記しておくのはプラスになります。

```
// 現在のシーンをシーン0(イントロ)で初期化  
var curScene = 0;
```

このコメントで変数の初期化を説明します。

「棒人形の冒険」にある変数curSceneの初期化の処理は詳しいコメントのおかげで内容がわかります。同様のコメントは変数messageの初期化にも適用できます。

```
// シーンメッセージをクリア  
var message = "";
```

ここでもコメントでコードの動作を説明しています。

複数行にまたがるコメントが必要な場合は、以下のように複数行コメントを作ることができます。

複数行コメントは
/*で始まります。

/* + {コメントの始まり}

コメントの続き

複数行コメントは
*/で終わります。

{コメントの終わり} + */

1行コメントは

//で始まります。

複数行コメントは

/*と*/で囲みます。

複数行のコメントにしたいときは/*と*/で囲みます。
囲まれた部分がコメントになります。

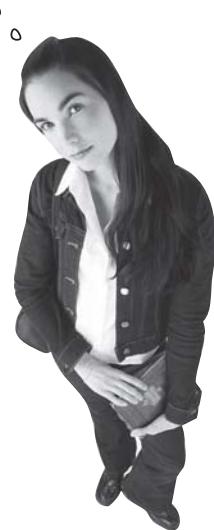
/* この3行コードはコメントです。からかっているのではありません、
真面目です。冗談ぬきで、これはコメントの一部なのです。 */

ちょっと待って。
コメントの意味はわかるけど、変数の
curScene と message がそれぞれ別の
場所で作成されている理由が
わからないわ。

curSceneは
changeScene() の外で
作成されています。

```
<script type="text/javascript">  
    // 現在のシーンをシーン 0 ( イントロ ) で初期化  
    var curScene = 0;  
  
    function changeScene(decision) {  
        // シーンメッセージをクリア  
        var message = "";  
        ...  
    }  
</script>
```

messageは
changeScene() の中に
作成されています。

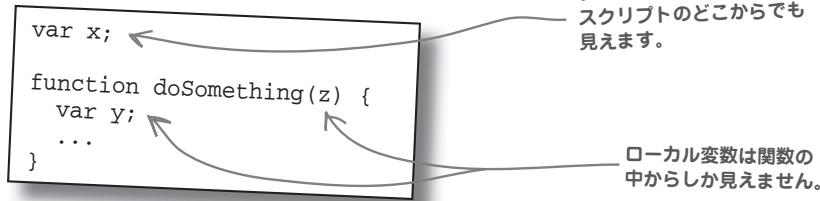


変数の場所

不動産の場合と同じように JavaScript でも場所 (location) は重要です。この例の場合「棒人形の冒險」の変数が作成される場所が重要です。message は関数の中で作成されていますが、curScene は changeScene() の外で作成されています。これは間違いではなく、変数のライフサイクルがスコープによって制御されるためです。スコープによってどのコードにアクセスできるか決まるのと同様です。

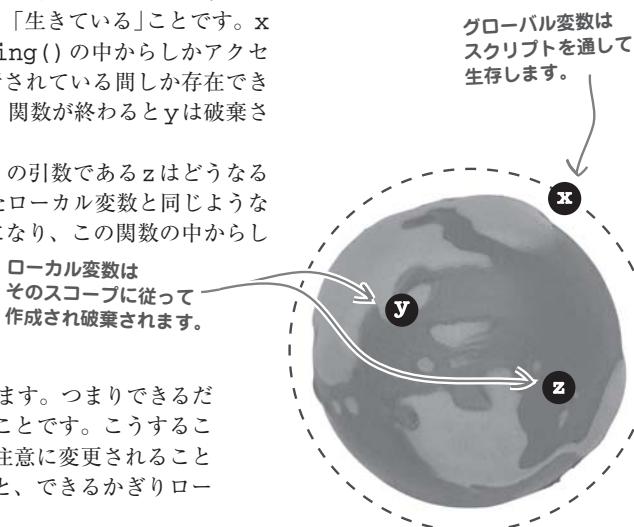
スコープとコンテキストでデータのライフサイクルが決まる

JavaScriptではスコープはデータのコンテキストのことを指します。スコープの中でデータは有効になりデータへのアクセスが可能になります。スクリプトのどこからでもデータにアクセスできる場合もあれば関数のようにある特定のコードブロックの中からしかアクセスできない場合もありますが、その違いはスコープの違いによるものです。以下に示す2つの変数はスコープが異なっています。

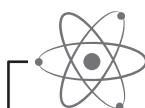


このコードではxは関数あるいはコードのブロックの外で作成されているのでグローバル変数になるのでスクリプトのどこからでもアクセスできます。重要なのはxはスクリプトが実行されている間ずっと「生きている」ことです。xと異なりyはローカル変数ですので、doSomething()の中からしかアクセスできません。またyはdoSomething()が実行されている間しか存在できません。つまり関数が開始されるとyが作成され、関数が終わるとyは破棄されるのです。

ここまで良しとして、ではdoSomething()の引数であるzはどうなるのでしょうか？ 関数の引数はすでに初期化されたローカル変数と同じような動作になります。そのためzはyと同じスコープになり、この関数の中からしかアクセスできません。



データの可視性は「知る必要性」を基準に決まります。つまりできるだけアクセスが制限できるなら制限すべき、ということです。こうすることでデータがまったく関係ないコードによって不注意に変更されることを避けることができます。実践的な言い方をすると、できるかぎりローカル変数を優先して使うべきということです。

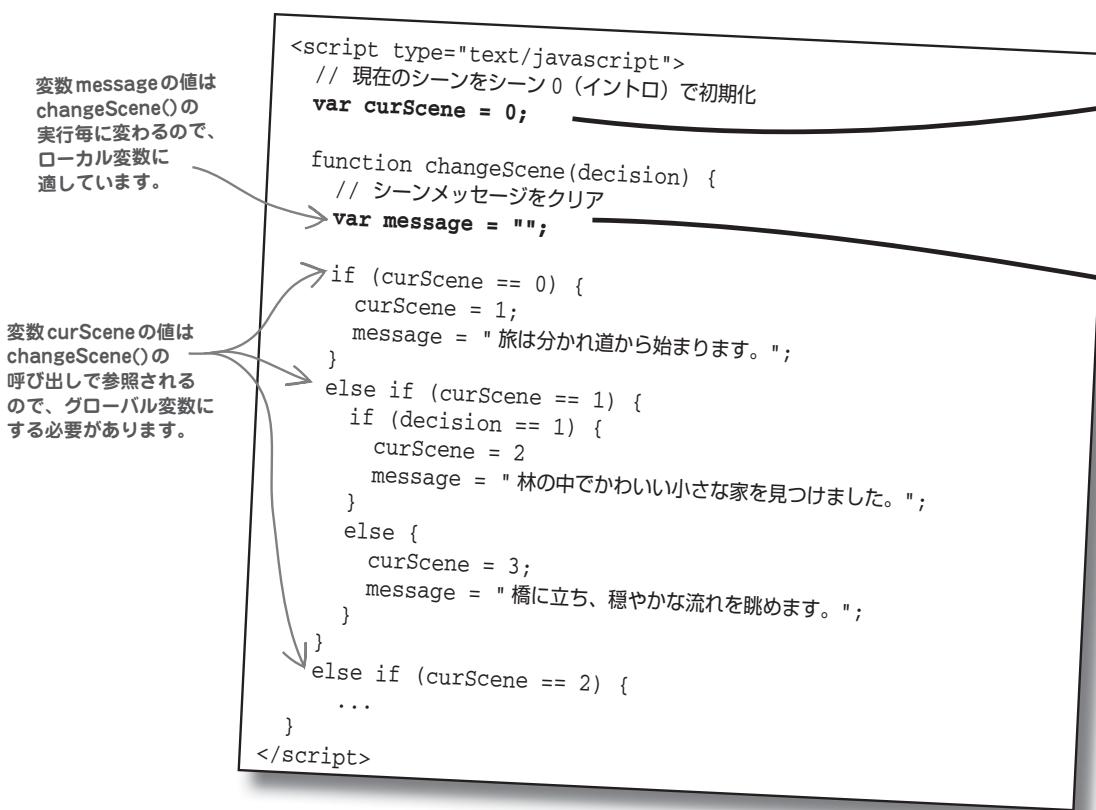


頭の体操

「棒人形の冒険」のコードでグローバル変数とローカル変数をどのように使い分ければいいでしょう？

変数のスコープをチェックする

ローカル変数とグローバル変数の違いをふまえて「棒人形の冒険」の変数を見直すと、変数が作成される場所が違っているのは理由があることがわかるでしょう。

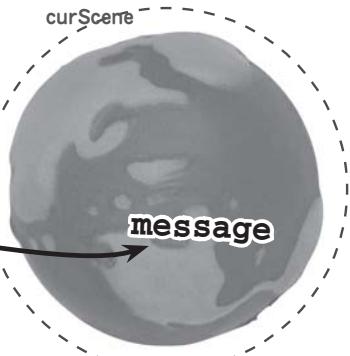


`changeScene()` のスコープの外にある変数の値を保持する必要があります。変数 `message` は関数の開始とともにクリアされるので関数の外で値を保持することはできません。一方 `curScene` は `if/else` のテスト条件でチェックされていますが、その値は関数の呼び出しが何回行われようと保持されています。つまり `message` はローカル変数に、`curScene` はグローバル変数にするのが妥当ということです。

データはどこに生存している？

スコープのことですこし頭がこんがらがったかもしれません、スクリプトのどの部分にあるかによってデータが生存する領域が違っていると考えるといいかもしれません。たとえば「棒人形の冒険」のスクリプトにはデータを格納するのに使えるスコープがいくつかあります。

グローバル変数

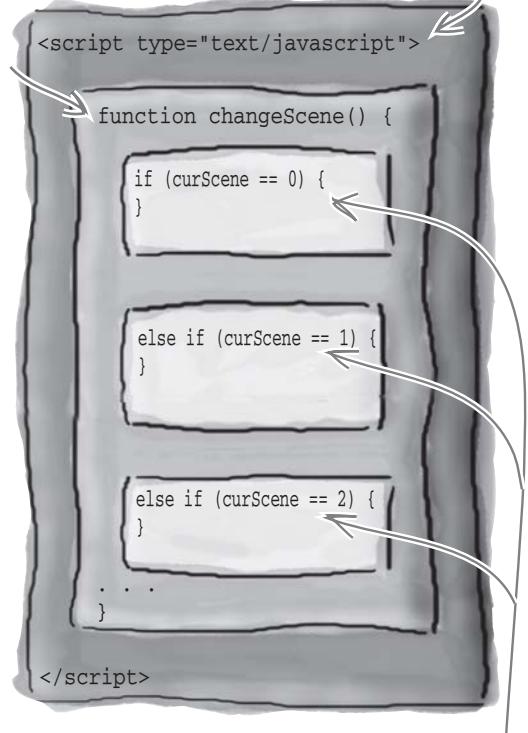


ローカル
変数

グローバル変数を作成できるのはグローバルスコープだけです。それ以外はすべてローカルになります。グローバルレベルで作成された変数はスクリプトのどこからでもアクセスできますが、ローカルなデータは限定されたスコープの中でしかアクセスできません。

changeScene() の
ローカルスコープ。

グローバルスコープ
(スクリプトレベル)。



複文ごとの
ローカルスコープ。

素朴な疑問に 答えます



： いくつかデータがあります。ローカル変数かグローバル変数、どっちを使って格納すればいいですか？



： どちらを使うべきかはそのデータの使い方によって決まります。一般的にはまずローカル変数で試してみて、それでうまくいかないときだけグローバル変数にしてみるといいでしょう。

特別座談会



今夜の対談：

ローカル変数とグローバル変数がデータにとって居心地のいい場所の重要性について討論します。

ローカル変数：

自分のまわりで起きていることだけに集中することで役に立っていると思いますね。実際、私の隣近所の外で何が起きているか知らないけれど、そういう生き方が好きなんです。

たしかに魅力的ですけど、自分のまわりが快適で安全なのが好きなのです。私の小さな世界の外から誰も私に干渉できないと思うと、ゆっくり寛げますね。

いやー、まいったなあ。私は靈魂の再生みたいなことを信じているわけではないんです。でも、あなたと同じように、データを格納するのに便利なのは、あなたと同じですよ。ただ外にいる誰からでも見られる場所に自分を置いていないだけでね。

そしてスクリプトがある情報をコードのあるセクションにだけ引きこもらせる必要があるときは、用心深い私の出番となるわけです。

だから人々は私たちそれぞれの有用さをわかっているんですよ。

グローバル変数：

おいおい、もっと広い世界に目をむける必要があるなあ。スクリプトの宇宙を知るためにも、ちょっと旅に出てみたら。

そうなんだろうね。でも、この大きなスクリプトの枠組みの中では、きみのささやかな生活なんて意味がないのは事実だよ。きみの小さな世界が現れては消えるたびに、きみの生活も作成され破壊される、その繰り返しだから。一方私はここにずっといる。ここにスクリプトがあれば、私もいるわけだ。

公平な意見だね。私も一度か二度誤って使われたことがあったのを認めるけど、いつも変わらずに値を保つのが私の良いところだし、そのおかげで問題を十分相殺してきたんだ。どこからでも値を見ることができるデータ断片をスクリプトが必要としているときは、私の出番ってわけさ。

そいつは素晴らしい。だけど、私はプライバシーよりアクセスしやすさと永続性を取るね。

素朴な疑問に答えます

 Q: JavaScriptのコードがコメントの中にあるとどうなりますか？

A: 何も起きません。JavaScript インタープリタはコメントを完全に無視するので、コメントの中に書かれたものは、インタープリタがスクリプトコードを実行開始したとき無視されます。このことがわかれれば、問題の原因を探したり、別のアプローチでコーディングを試みるときなどに、コードの断片を一時的に無効にするための手段としてコメントが使えます。

 Q: JavaScriptコードの1行の終わりに1行コメントを付けることはできますか？

A: はい。この場合、そのコードはコメントの一部ではないので実行されます。1行コメントを使う場合、その行全体をコメントにする必要はありません。//から行末までがコメントになります。//の後にコード断片が続く場合でも、コメントの前にあるコードは正しく動きます。

 Q: なぜコメントはセミコロンで終わらないのですか？

A: コメントはJavaScriptの文ではないからです。本の脚注のようにコメントはコードに関する付加情報を記述するラベルにすぎません。JavaScript インタープリタはあらゆるコメントを無視します。コメントは人間のためにあり、JavaScriptのためにあるのではありません。

 Q: グローバル変数の作成にかかる「スクリプトレベル」とはどういう意味ですか？

A: 「スクリプトレベル」とはスクリプトコードのトップレベル、つまり<script>タグの中のことです。「スクリプトレベル」の意味は、関数やコードのブロックの外側にあることなので、「スクリプトレベル」で作成されたものはすべてグローバルとみなされます。このため「スクリプトレベル」で作成されたものはスクリプトと生存をともにするので、ページの中のどのコードからもアクセスできます。

 Q: 複文の中で変数を作成した場合、ローカル変数になるのですか？

A: はい。複文は新しいスコープレベルを確立するので、複文の中で作成されたものは文の中のローカルスコープになります。こうした変数は、スクリプトの処理フローが複文に入ると同時に作成され、出るときに破棄されるので、一時変数を考えることができます。

 Q: スコープとかフローとか実行とか、このローカル変数とグローバル変数に関してはなにかと複雑ですね。そんなに難しいのですか？

A: そんなことはないですよ。重要なのは、ローカル変数は関数などのコードの塊の外で記憶しておく必要のない一時的な情報を格納するに適していることです。データをスクリプトのライフサイクル全体で生存させる必要がある場合、これをグローバル変数にするべきです。面白いことに、ほとんどのスクリプトデータは最初に想定していた以上に一時的な扱いになることが多いので、グローバル変数よりもローカル変数を使うことが多くなるでしょう。ローカル変数には一時的情報を格納し、グローバル変数にはスクリプトのライフサイクル全体で使えるデータを格納します。

ローカル変数には一時的情報を格納します。

グローバル変数はスクリプトと同じ生存期間になります。

重要ポイント

- コメントは後で追加すべきコードを忘れないようにメモするのに便利です。
- コードの動きを説明するコメントをたくさん書くと後でコードが理解しやすくなります。
- 1行コメントを開始するときはスラッシュの組(//)を使います。
- 複数行コメントは /* で開始し */ で終わります。
- グローバル変数はスクリプトレベルで作成され

ます。つまり関数やコードのブロックの外で作成され、スクリプトのライフサイクルを通して値が保持されます。

- ローカル変数はコードのブロックの中で作成される（そして破棄される）ので、そのブロックの中からしかアクセスできません。
- ローカル変数はグローバル変数よりアクセスの制限を厳しくできるので優先的に使われます。

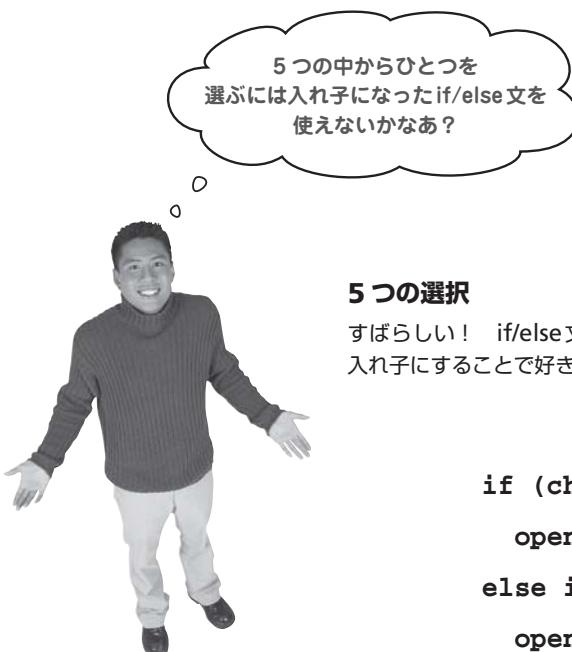
5つの選択

この章のはじめの方に出てきたゲームショーの当選者のエリックはどうしているでしょう？ ドーナツを平らげたエリックは「取引したいでショー」の後半戦に進んでいるようです。彼はとても難しい選択に直面しています。5つの選択肢からひとつを選ばなければならぬのです。



頭の体操

選択肢が5つある中から選ぶ処理のコードはどうなるでしょう？



5つの選択

すばらしい！ if/else文は二者択一の決定を下すために作られていますが、入れ子にすることで好きなだけ選択肢を増やすことができます。

```
if (chosenCase == "A")
    openCase("A");
else if (chosenCase == "B")
    openCase("B");
else if (chosenCase == "C")
    openCase("C");
else if (chosenCase == "D")
    openCase("D");
else if (chosenCase == "E")
    openCase("E");
```

このコードは動くには動きますが、最後の条件になるまですべてのテスト条件を評価するので、効率がよくありません。

入れ子になったif/elseは複雑です

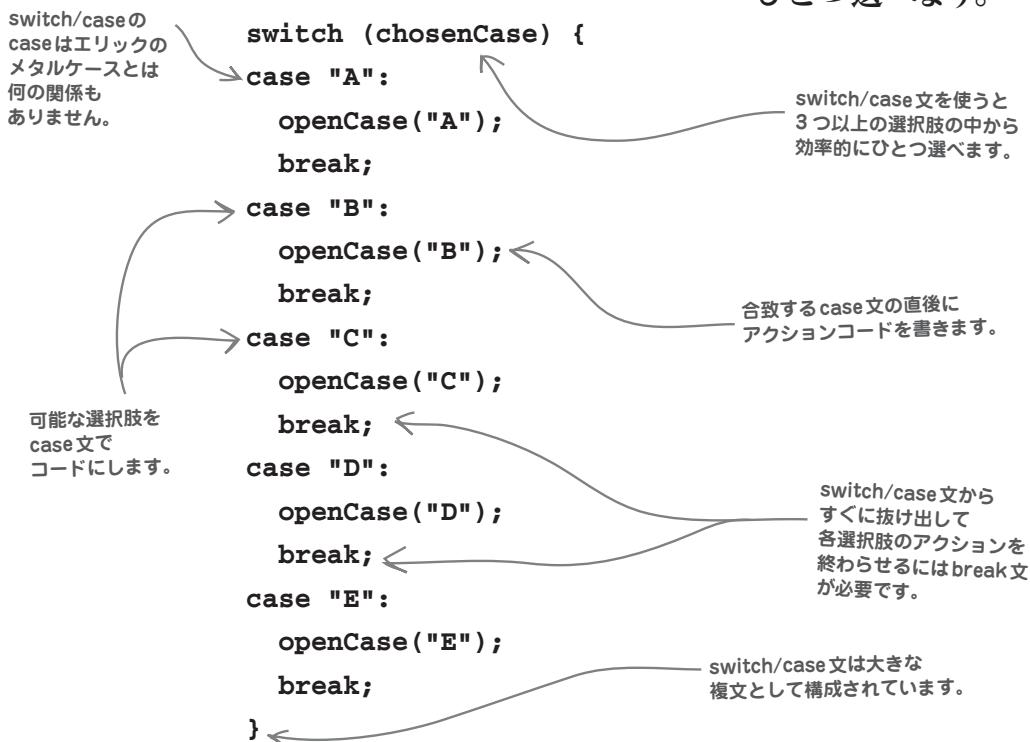
入れ子になったif/else文は動くには動きますが、あまり効率的ではありません。もともと二者択一のために設計されているので、それより選択肢が多くなると効率がわるくなります。たとえばケースEを選ぶまでに論理値のテストが何回実行されているか考えてみましょう。5つあるテスト条件がすべて評価されるので、かなり効率が悪いわけです。



switch文には複数の選択肢があります

JavaScriptには複数の選択肢に対応した意思決定のための文があります。if/else文は二者択一に最適ですが、switch/case文はたくさんある選択肢の中から効率的に実現します。エリックのジレンマをswitch/case文で見直してみましょう。

switch/case文を使うと3つ以上の選択肢の中から効率的にひとつ選べます。



エクササイズ

switch/case文はif/else文ができるることは何でもできる。ホント、ウソ、どっち？

ホント

ウソ



エクササイズの
答え

switch/case文はif/else文ができることは何でもできる。ホント、ウソ、どっち？

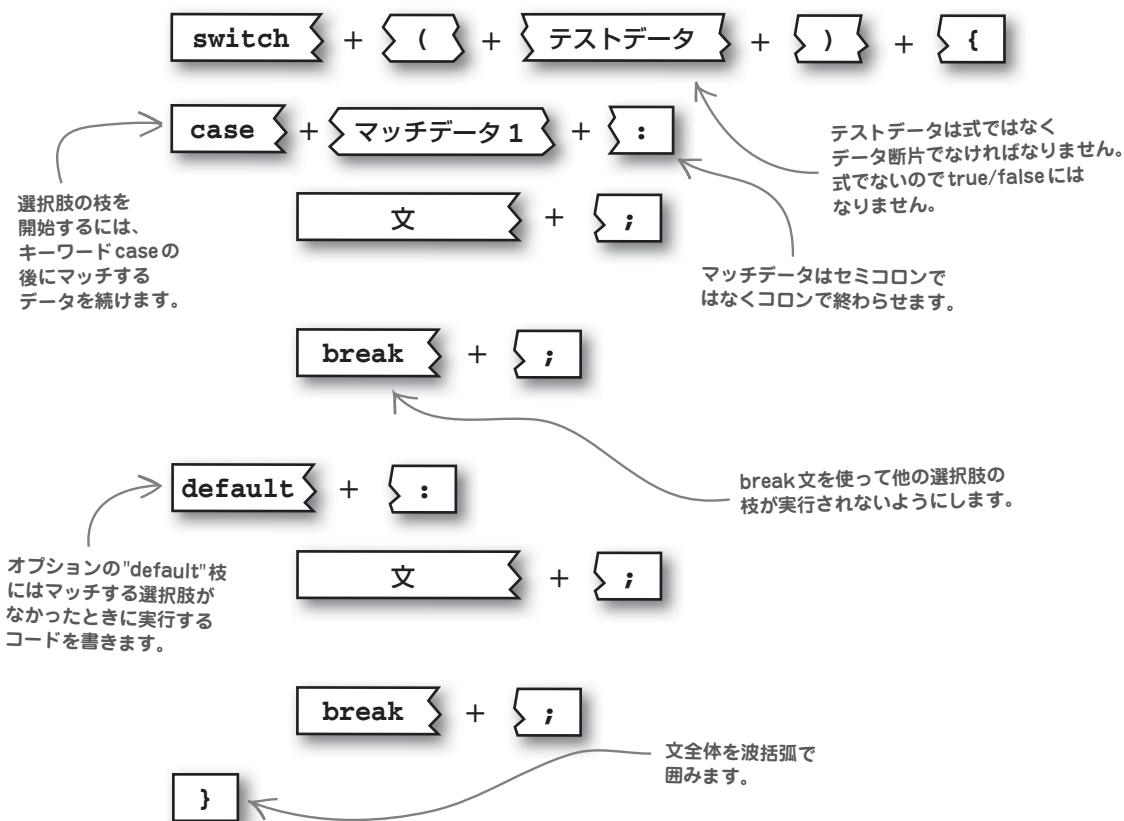
ホント

ウソ

if/elseと違ってswitch/case文を制御する
テストデータは式にすることができません、
データ断片にしなければなりません。

switch文の内部構造

switch/case文の使用例をみてきたので、この文を分解して一般的な形式をみてみましょう。



switch/case文を書いてみましょう

switch/case文を作るのはif/else文を作るときよりも確かにすこし厄介ですが、3つ以上の選択肢を扱うときは処理が効率よくなります。以下のような手順になります。

- ❶ テストデータを括弧で囲み、複文を開始します({ })。
- ❷ caseに対応する選択肢の値を書いて、最後にコロンを付けます(:)。
- ❸ 条件にマッチしたとき実行するコードを書きます。ここには複数行のコードを書けるので複文にする必要はありません。
- ❹ break文を使います。セミコロン(;)を忘れないように。
- ❺ マッチするものがなかったときに実行するコードをdefaultで書くこともできます。
- ❻ 複文を閉じます(})。

switch/case文を使ったら
「棒人形の冒険」は効率的に
なるかなあ。

素朴な疑問に

答えます

Q: switch/case文はtrue/false式を使って判断を行うわけではないですか？

A: ご指摘の通りです。if文やif/else文と違って、switch/caseはテストデータの断片を使って判断を行います。このため複数の選択肢をサポートできるのです。



要注意！

安全のためにbreakを使いましょう。

それぞれのswitch-caseとbreak文を対応させて、他のコードが誤って実行されないようにしましょう。

Q: それぞれのcaseはそのテストデータにマッチするのですか？

A: はい。変数をテストデータとして使うという発想です。リテラル値を使ってどの条件にマッチするか判断します。

Q: switch/caseの中からbreak文をすべて取り除いたらどうなりますか？

A: 結果は予測できません。break文はswitch/case文にあるアクションコードをセクションに区切るために使います。breakがないと、アクションコードはひとつの大きな塊として実行されるので、それぞれ異なる判断をするという目的に反してしまいます。switch/case文でマッチするものがあった場合、該当するcaseに続くコードが実行され、break文になったところで終了します。switch/case文の途中で脱出できるのはbreak文だけなのです。





switch の真実

今週のインタビュー：Switchは意思決定のキーマン

Head First: ようこそお越しくださいました。さっそくですが、あなた自身を一言で表すとしたら？

Switch: 選び屋かな。

Head First: 詳しく話していただけますか？

Switch: たくさんの異なるものを選べるようにするのです。もちろん白か黒かの簡単な選択の場合もありますが、もっと微妙な違いが要求される状況もたくさんあります。そこで私の出番となるわけです。

Head First: 同じことはifでもできるし、その方がコードが小さくなることもある、と言う人もいますよ。

Switch: そうかもしれませんね。材木に十分な長さがあれば、ハンマーでも切れますよね。でも私はノコギリを使う方がいいんです。誰しもそれぞれ専門があって、私の専門はいくつかの異なるものの中から効率的に選ぶことなんです。ifに不満はありませんが、彼の専門は私とは別なのです。

Head First: 効率的とおっしゃいました。あなたの仕事と効率の関係を教えてください。

Switch: ええっと、私はデータ断片の値にもとづいて判断を下すように構成されています。データ断片をいくつかの可能性と比較して、どのコードを実行するか決めるのです。式を評価しようとして悩むこともありませんし、複数の選択肢から選ぶときに入れ子みたいな気の利いた仕掛けも使いません。データの断片をもとにすばやく判断したいときは、私がうってつけというわけです。

Head First: あなたの友人であるbreakについて聞かせてください。彼なしでは一日たりとも過ごせないと聞いています。

Switch: そうです。breakがいないと、アクションコードを区切る方法がわからないので、大きなトラブルになってしまいます。breakはコードのセクションがどこで終わるか教えてくれるので、誤って他のコード

を実行することなく終了できるのです。

Head First: わかりました。caseとも親しい仲なんですね？

Switch: その通りです。caseと私はとても親密な仲でして、caseは与えられたテストデータに対する可能な選択肢をすべて私に教えてくれます。caseがないなければ、判断を下す基準がなくなってしまうのです。

Head First: caseがさまざまな選択肢を並べてくれることはわかりました。それらの選択肢を使って何を実行するか決めるのですね。でもテストデータにマッチするものがなかったら、どうなりますか？

Switch: 場合によりけりです。もし「該当なし」のシナリオを扱うコードが追加されていなければ、何も起きません。しかし、親友のdefaultは、該当なしのイベントのときだけ特定のコードを実行できるようにしてくれます。

Head First: それは知りませんでした。caseとdefaultの仲はどうなんですか？

Switch: いい関係ですよ。互いに競って気を引こうとはしないので、お互いの邪魔をすることもありません。caseはマッチするものを処理し、defaultはマッチするものがないときの処理を行います。defaultがいてくれるおかげでcaseは該当なしの場合の心配をしなくてすむので、caseはすこし安心できていると思います。

Head First: わかりました。そろそろお別れの時間ですね。最後に一言お願いします。

Switch: わかりました、優柔不断ほど悪いものはありません。曖昧さは好まれません。可能性がたくさんあるからといって、あきらめて逃げてはいけません。私に声をかけてくだされば、あなたのスクリプトが最高の状態で意思決定できるように私は全力をつくしてお手伝いします。



自分で考えてみよう

「棒人形の冒険」のコードの最初の 2 つのシーンのコードを if/else の代わりに switch/case 文を使って書き換えてください。

```
if (curScene == 0) {
    curScene = 1;
    message = "旅は分かれ道から始まります。";
}
else if (curScene == 1) {
    if (decision == 1) {
        curScene = 2;
        message = "林の中でかわいい小さな家を見つけました。";
    }
    else {
        curScene = 3;
        message = "橋に立ち、穏やかな流れを眺めます。";
    }
}
```

if/else文で
コーディングした
もとのバージョン
です。

自分で考えてみよう の答え

棒人形の冒険」のコードの最初の 2 つのシーンのコードを if/else の代わりに switch/case 文を使って書き換えてください。

```
...  
if (curScene == 0) {  
    curScene = 1;  
    message = "旅は分かれ道から始まります。";  
}  
else if (curScene == 1) {  
    if (decision == 1) {  
        curScene = 2;  
        message = "林の中でかわいい小さな家を見つけました。";  
    }  
    else {  
        curScene = 3;  
        message = "橋に立ち、穏やかな流れを眺めます。";  
    }  
}  
...  
...
```

if/else 文で
コーディング
したもの
のバージョンです。

各 case の
マッチデータは
シーン番号に
対応しています。

選択肢ごとに if/else 文
を使ってユーザの選択
を処理しています。

残りのシーンも
似たような構造に
なります。

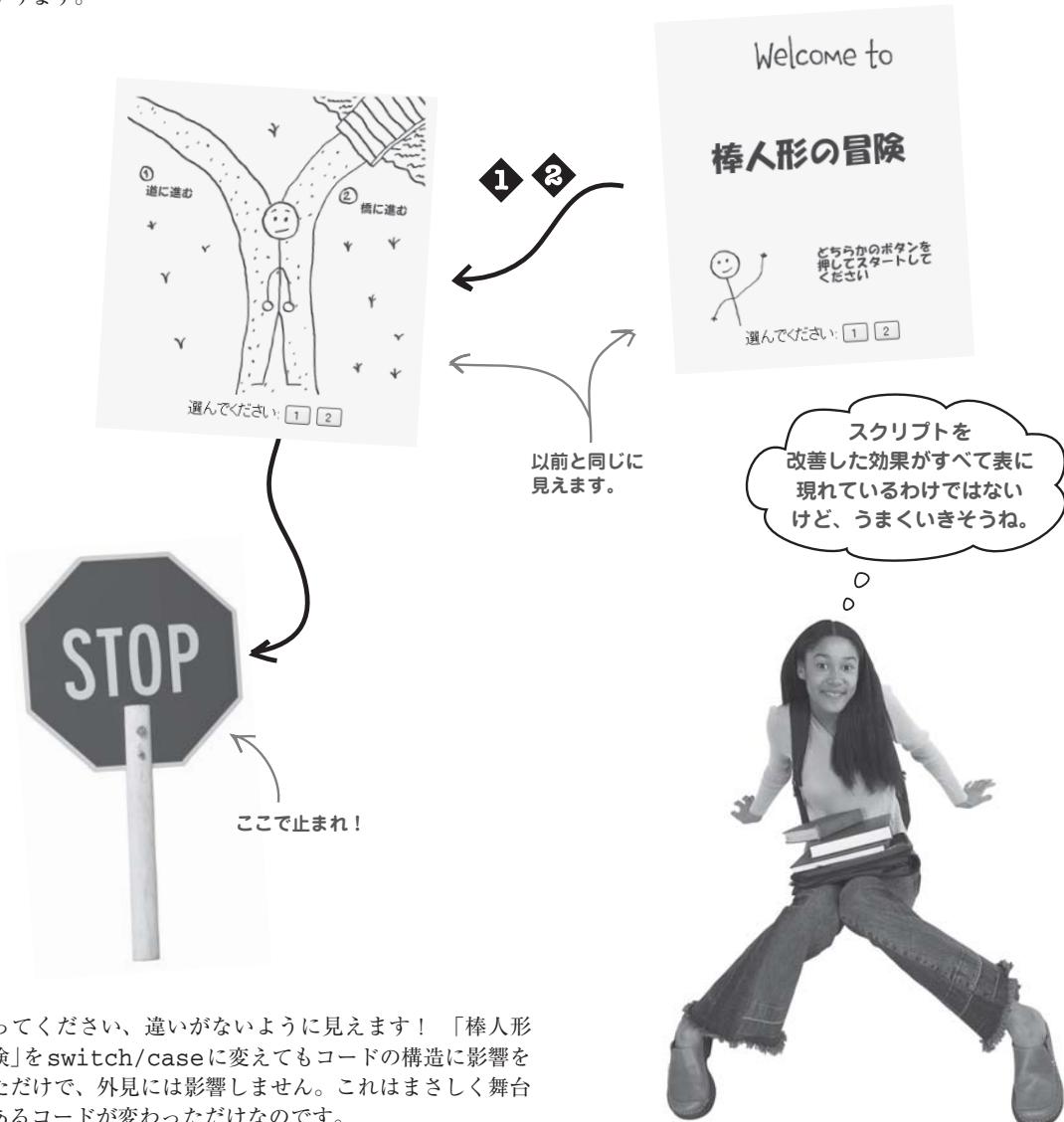
switch/case 文を
} で閉じます。

```
switch (curScene) {  
case 0:  
    curScene = 1;  
    message = "旅は分かれ道から始まります。";  
    break;  
case 1:  
    if (decision == 1) {  
        curScene = 2;  
        message = "林の中でかわいい小さな家を見つけました。";  
    }  
    else {  
        curScene = 3;  
        message = "橋に立ち、穏やかな流れを眺めます。";  
    }  
    break;  
...  
}
```

if/else 文と同じように
シーン番号とシーン説明
メッセージを設定します。

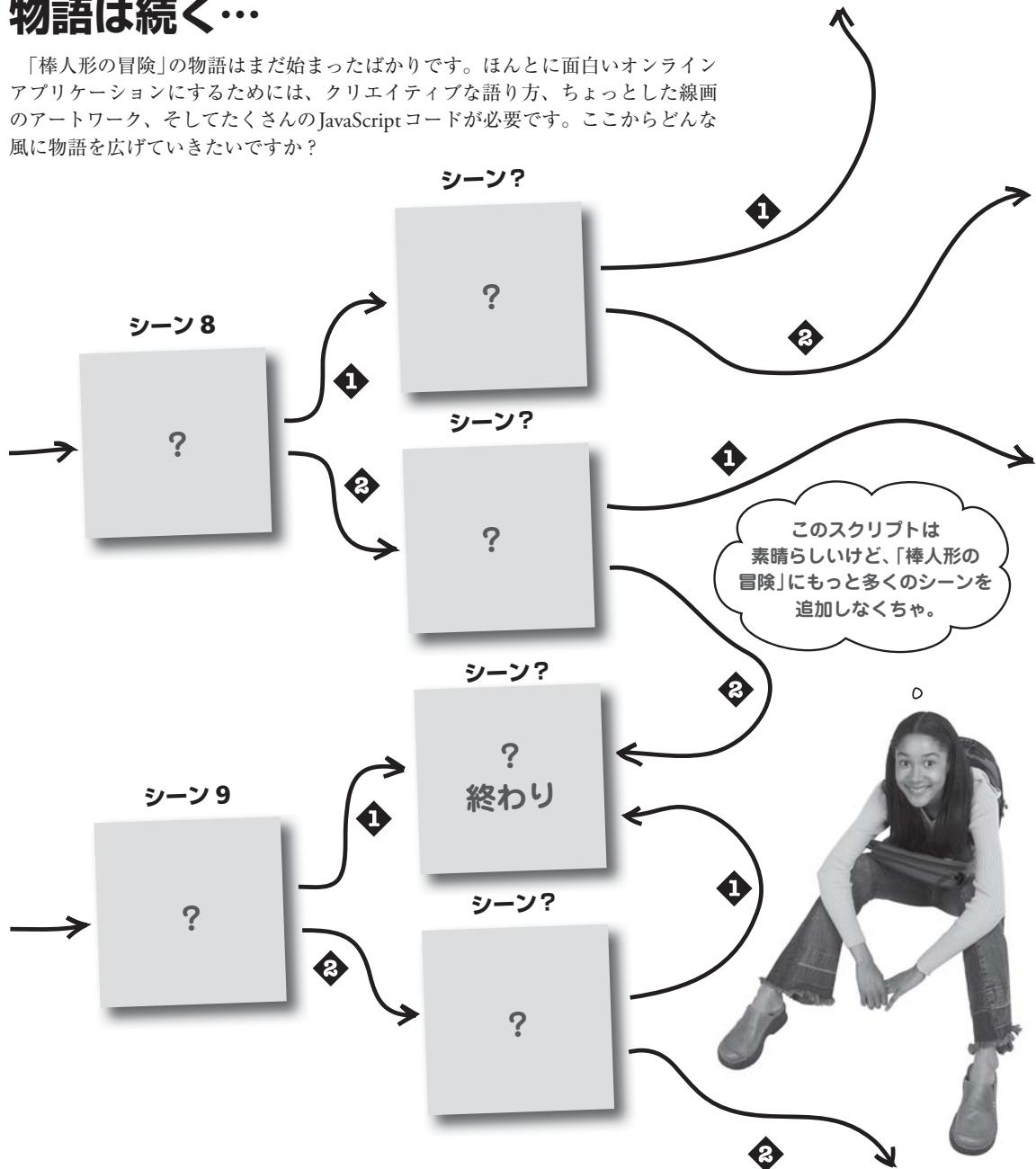
「棒人形の冒険」のswitchバージョンをテスト運転する

「棒人形の冒険」の意思決定ロジックを完全に書き直したので、エラーははやく結果を確認したくてうずうずしています。物語をたどってみると変更の効果がすぐわかります。



物語は続く…

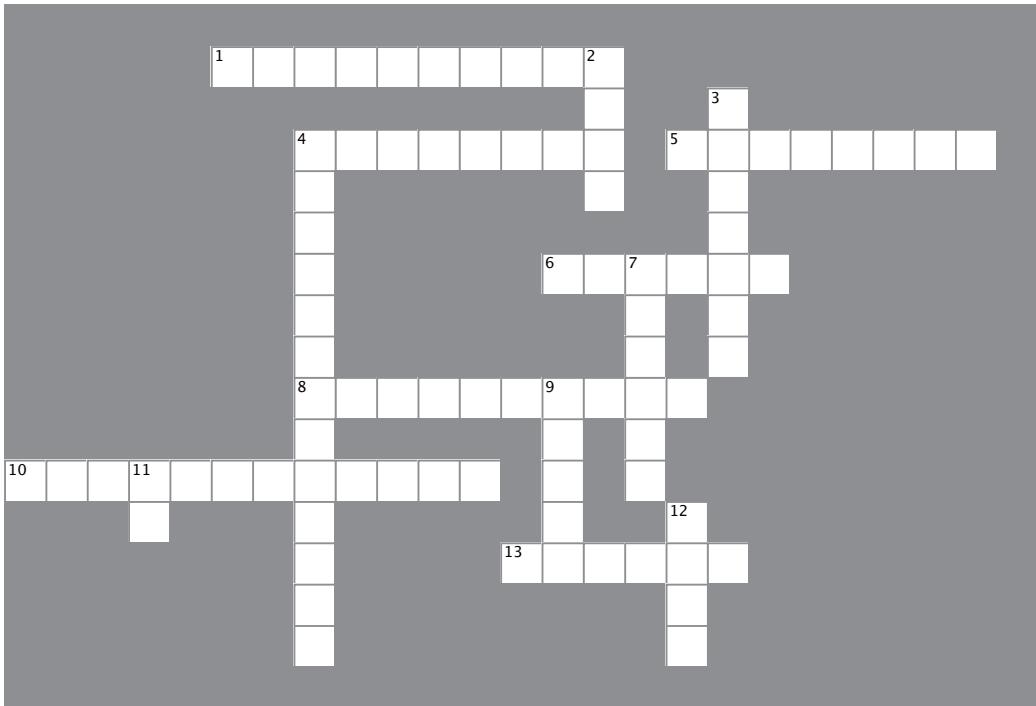
「棒人形の冒険」の物語はまだ始まったばかりです。ほんとに面白いオンラインアプリケーションにするためには、クリエイティブな語り方、ちょっとした線画のアートワーク、そしてたくさんのJavaScriptコードが必要です。ここからどんな風に物語を広げていきたいですか？





JavaScript クロスワード

さあ、簡単な選択です。休憩してちょっとしたクロスワードパズルをやってみませんか？



ヨコのカギ

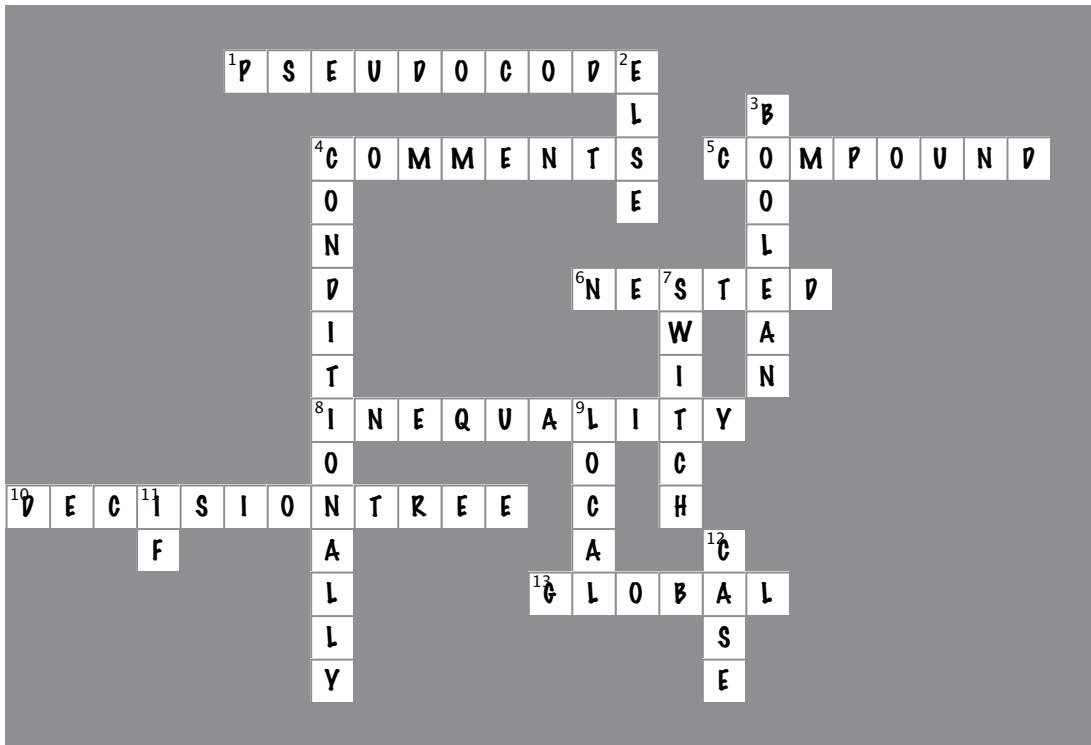
1. 最初にこれを書いておくと複雑なJavaScriptコードを書くのもかなり簡単になります。
4. これを使ってコードの説明を書きます。
5. この種類の文は実際には複数の文で作られます。
6. if文の中にif文があるとき と言われます。
8. !=演算子はこれをテストします。
10. これを使うことで複雑な決定を視覚化することができます。
13. この種類の変数はスクリプト全体からアクセスできます。

タテのカギ

2. これを実行するか あれを実行する。
3. この種類の演算子は結果がtrue/falseになります。
4. if/else文の一部になったときコードはこのように実行されます。
7. この文を使うとデータ断片の値をもとに決定を行うことができます。
9. この変数のスコープは制限されています。
11. この文を使うとコードの断片を条件に応じて実行できます。
12. switch文の各枝にはそれぞれこれがひとつあります。



JavaScript クロスワードの答え



折り畳みページ

脳のところで
折り畳んでから
謎を解決しましょう。

if/elseでは十分でない
ときには…

左脳と右脳がご対面！



シーン0

イントロ
の
タイトル

シーン7

橋に
巨人が
います

シーン?

沼地

グローバル

ローカル

シーン?

溝

12

1

2

シーン1

道が
分かれて
います

1

2

シーン8

川に橋が
架かって
います

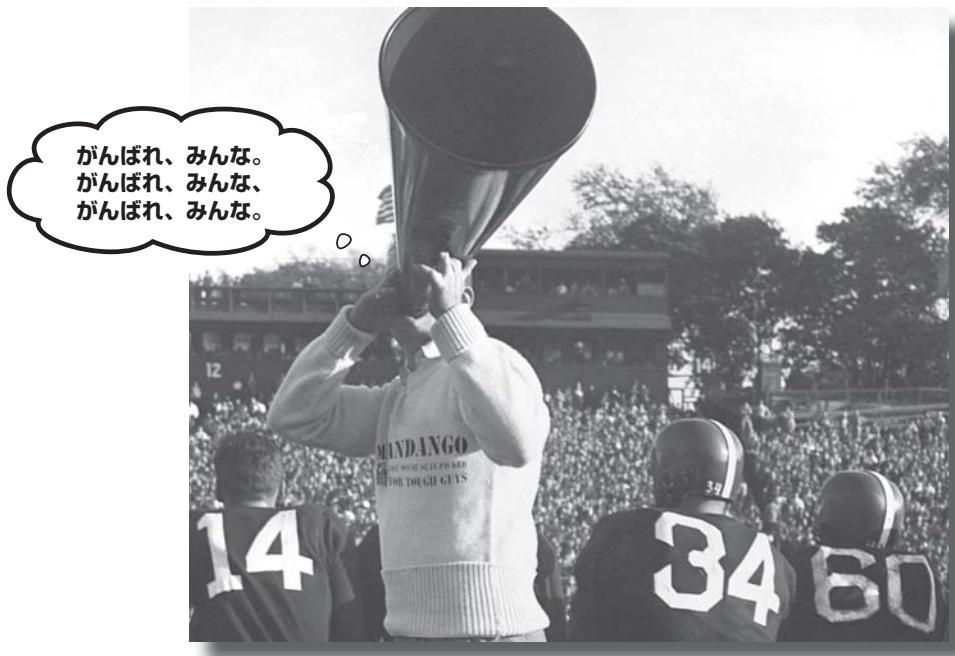
if/else文は信じられないくらい手軽ですが、限界もあります。

たとえば3つ以上の選択肢を扱えません。

嘘だと思うなら自分で試してみましょう。

5章 ループ

同じことが
重複するのは危険

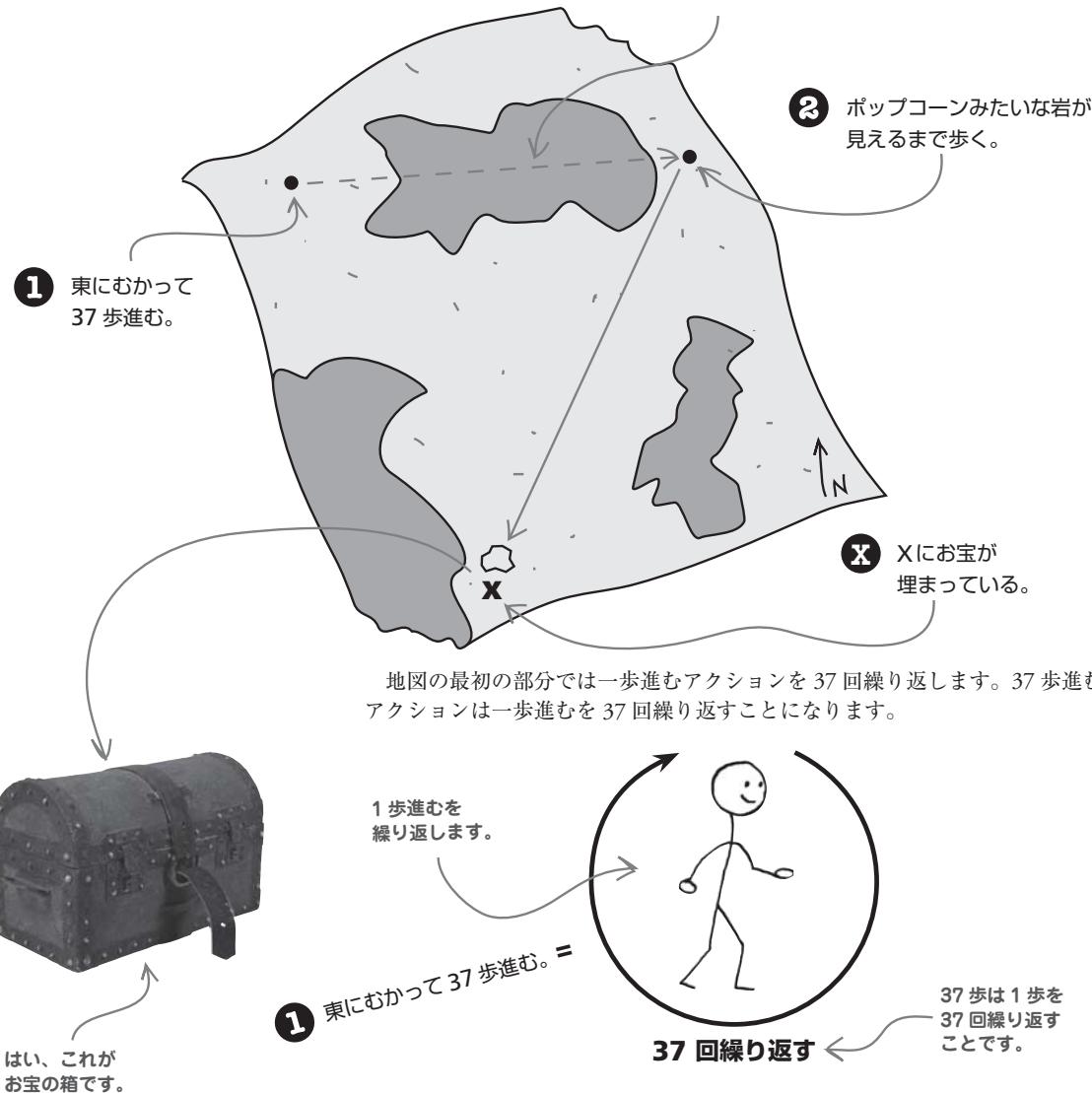


繰り返しに人生の趣がある、という人もいます。新しくて面白いことはたしかに刺激的ですが、毎日の生活を支えているのは些細なことの繰り返しです。でも、強迫神経症的に手を洗うこと、緊張したときの癖、受け取った異常なメッセージで「全員に返信」をクリックすることなど、現実には繰り返しがいつもいいとは限らないようです。しかし、JavaScriptの世界では繰り返しは最高にお手軽です。スクリプトでは驚くほどコード断片を何度も繰り返し実行する必要があります。ループの力が発揮されるのはそのときです。ループがなければ多くの時間を浪費して無駄なコードのかたまりをカット&ペーストしたことでしょう。

お宝はXに隠されている

埋蔵された宝ものには抗しがたいロマンがあります。JavaScriptで動くお宝地図をみてみましょう。

37 歩



そこで質問です。JavaScriptで繰り返しを実現するにはどうしたらいいでしょう？

デジャヴュのように何度も繰り返すならforループ

JavaScriptではループを使ってあるコードの繰り返しを実行します。とくにforループは何かをある回数だけ繰り返すのに適しています。たとえばゼロになるまでカウントダウンしたり、ある値までカウントアップする、といった処理にforループは向いています。forループは4つの部品でできます。

① 初期化

forループの最初に1度だけ初期化が実行されます。

② テスト条件

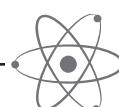
テスト条件ではループをもう1度繰り返すかチェックされます。

③ アクション

ループのアクション部は、サイクルごとに実際に繰り返されるコードです。

④ 更新

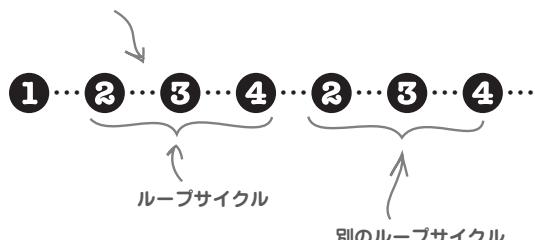
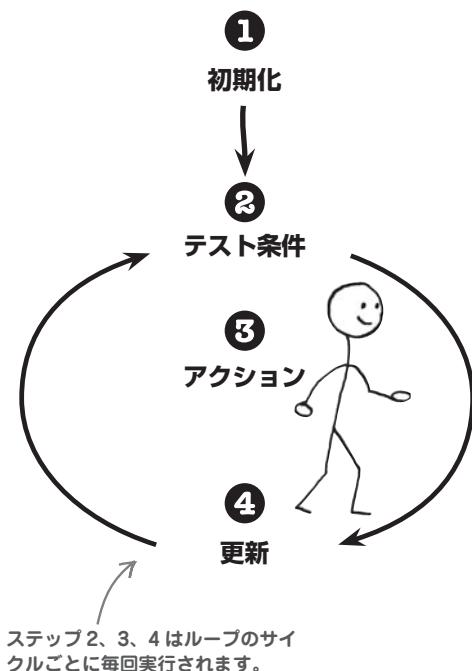
ループの更新部ではひとつのサイクルの終わりでループ変数を更新します。



頭の体操

forループの4つの部品をお宝地図に応用するにはどうすればいいでしょう？

forループを使うとコードがある指定した回数繰り返すことができます。



for ループを使ってお宝探し

for ループを使ってお宝地図をたどる理由は、ある回数だけ進むというアクションがあるからです。お宝地図の最初の部分に for ループを適用すると、以下のようにになります。

```

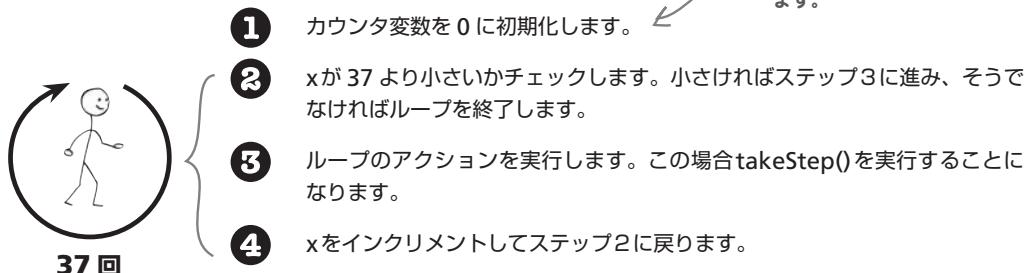
1   2   4
for (var x = 0; x < 37; x++)
      takeStep();
3

```

x のインクリメントは
 $x = x + 1$ と同じです。

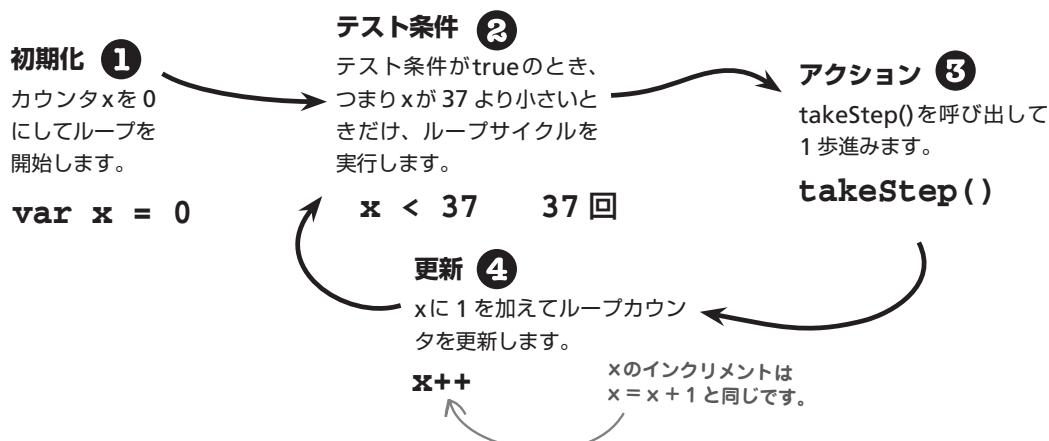


for ループのコードを分解すると次のようになります。



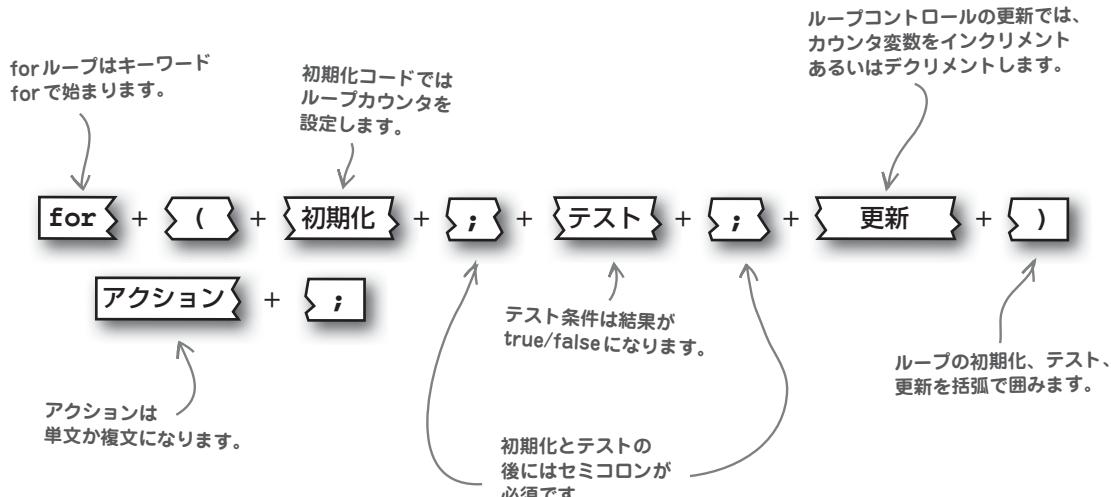
JavaScript のループは通常 0 から開始しますが、簡単な変更で 1 から開始させることもできます。

37 回ループを繰り返したら、 x が 37 になったところでループが終わります。for ループのパズルを構成する 4 つの部品がかみ合って動くことで JavaScript の繰り返しが実行されます。



forループを解剖する

forループには一貫した形式があり、4つの部品を特定の位置に配置する必要があります。この形式にはとても柔軟性があるので、ループを作るのは難しくありません。



エクササイズ

このコードを完成させてください。まずユーザに0より大きな数値の入力を促し、次のその数値をカウントの開始値として使ってforループを実行します。映画のカウントダウンのようなものです(4、3、2、1、スタート)。カウントダウンの実行前に数値が0より大きいか検証するのを忘れずに。

このカウント変数に
数の値を格納します。

ユーザに数の入力
を促します。

```
var count = prompt("0より大きな数字を入力してください。", "10");
```

.....

.....

.....

.....

.....

.....



エクササイズの 答え

このカウント変数に
数の値を格納します。

ユーザに数の
入力を促します。

1まで
カウント
ダウン。

カウントが
0より大きい
のを確認します。

```
var count = prompt("0より大きな数字を入力してください。", "10");
```

ループカウンタ
xを初期化
します。

```
if (count > 0) {  
    for (var x = count; x > 0; x--)
```

ループごとに
毎回カウントを
デクリメント
します。

```
    alert("Starting in..." + x);
```

```
    alert("映画スタート！");
```

現在の
カウントを
表示します。

```
}
```

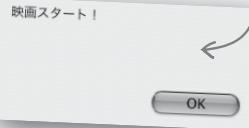
```
else
```

```
    alert("数字が0より大きくないです。映画はお見せできません。");
```

不正なデータ。

数字が0より大きくないです。映画はお見せできません。

カウントが完了！



お前のことキレイ
じゃなんだけど、席は
空けてくれな。

わかってるよ

Mandango: マッチョのための映画館座席探し

JavaScriptのループを映画に応用するとなったら、カウントダウンの映像だけではありません。マッチョな男たちは映画館で映画を観るとき、左右の席が空いた真ん中の席を陣取りたがります。セスとジェイソンは、こうしたマッチョな男たちがふんぞり返って占有できる席を探せるようにMandangoという検索機能を作ろうとしています。マッチョな男にまとめて3席分買わせるのが目的ですが、セスとジェイソンはまだこの検索機能の実現方法がわからず困っています。



最初に空席をチェックします

二人が直面している課題は、1列に並んだ座席を順番にみていくて3席連続した空席を探すことです。



だめだね。

3席空いていないので、
マッチョな映画鑑賞が
できません。



これだ！

3席連続して空いているので、
マッチョな映画鑑賞ができます。



自分で考えてみよう

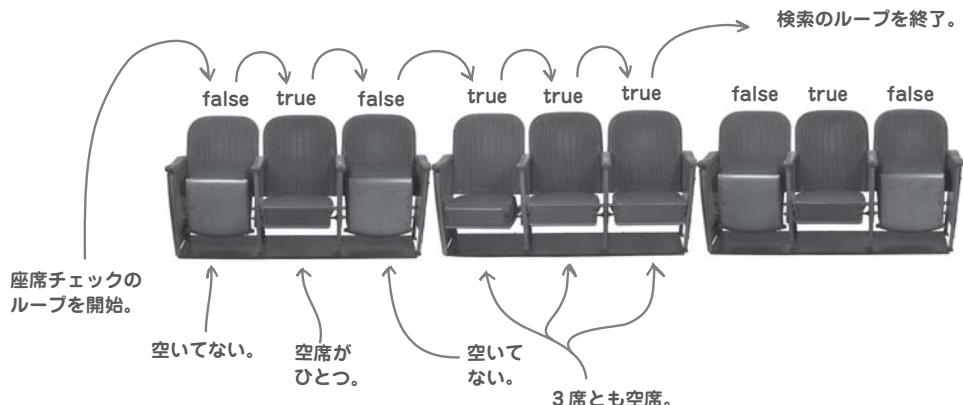
以下に示した映画館の座席の列に対して、forループを使って連続した空席3席を探す方法を書いてください。座席に対してループをどのように使えばいいか、正確に書いてください。



自分で考えてみよう の答え

以下に示した映画館の座席の列に対して、forループを使って連続した空席 3 席を探す方法を書いてください。座席に対してループをどのように使えばいいか、正確に書いてください。

それぞれの座席が空いているかを論理値変数で表現されていれば、ループで順に 3 席空いているか調べることができます。



ループと HTML と座席

Mandango のデザインはおおよそ見当がつきますが、各座席の状態をどのように HTML コードに翻訳すればいいのか、はっきりしていません。

このサンプルの
HTML と画像は
<http://www.headfirstlabs.com/books/hfjs/> からダウンロードで
きます。

Mandango のページでは
座席を画像で表示します。

```









```

HTML 画像要素をループする必要があるだけでなく、JavaScript コードの論理値変数として座席の状態を格納する方法も必要です。

座席を変数にする

ループで順に空席を調べる前に、各シートの状態を JavaScript コードで表現する必要があります。9席の状態は9個の論理値変数で表現できます。

```
var seat1 = false;
var seat2 = true;
var seat3 = false; ←
var seat4 = true;
var seat5 = true; ←
var seat6 = true;
var seat7 = false;
var seat8 = false; ←
var seat9 = false;
```

各座席の状態は論理値で格納されます。

trueは空席を意味します。

falseは座席が空いてないことを意味します。

複数のデータ断片をループできるように同じ変数名を使う必要があるんだな。
楽しそうだって？
ちょっとも楽しくねえや。

ここで9席のうち空席が3席あるか調べる処理を for ループで作ってみましょう。

```
for (var i = 0; i < 10; i++) {
  if (seat1)
    ...
}
```

ループの中で変数名を変えることができません。



ここで問題が生じました。for ループでは毎回座席の変数の値をチェックする必要がありますが、各座席の変数名が異なるので、うまく処理できません。

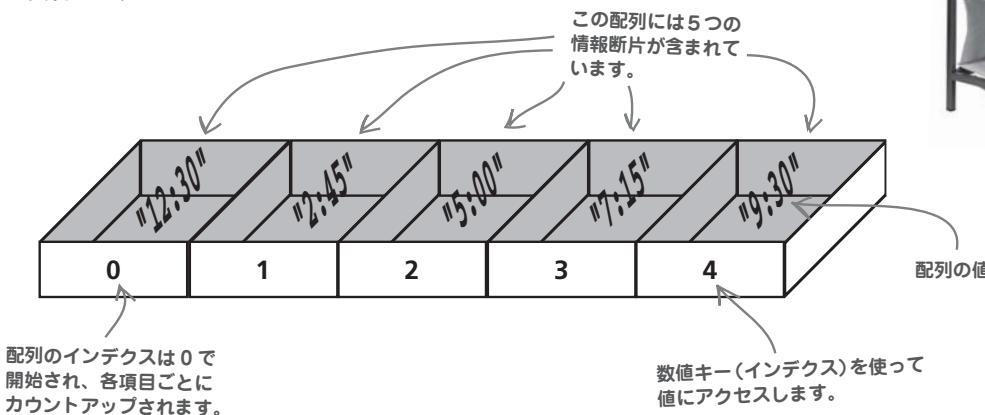


ループの中で変数が別々であるためうまくいきません。ループできるように情報を格納するにはどうしたらいいでしょう？

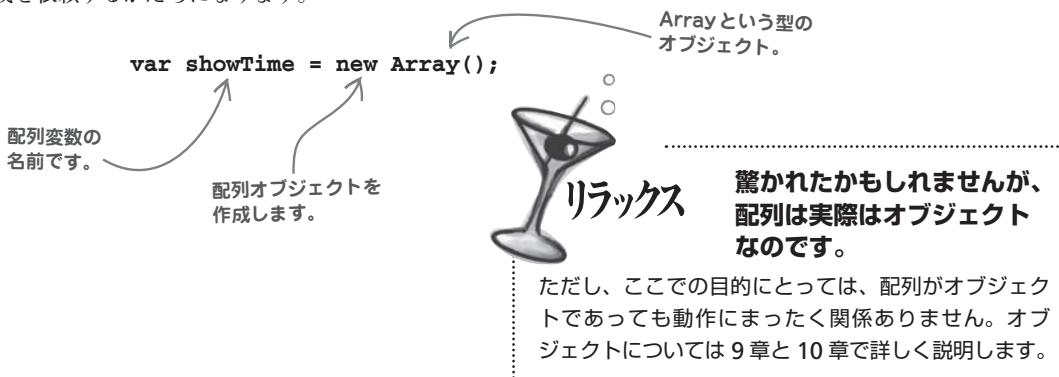
配列は複数のデータ断片を集めたもの

JavaScriptには配列と呼ばれる特別なデータ型があり、複数のデータ断片をひとつの変数に格納することができます。配列変数は通常の変数と同様、名前はひとつですが、記憶に使う場所が複数あります。それはちょうど家の中にある収納棚のようなものです。家具としてみればひとつですが、収納スペースは複数ありますよね。

配列の各アイテムは、値とその値にアクセスするために使う一意的なキー、この2つの情報断片で構成されます。キーは通常0からはじまり、順にアイテムを数え上げていきます。数値キーはインデクスとも呼ばれ、インデクス配列を実現します。



配列の作成方法は変数の作成方法と似ていますが、記憶領域をひとつだけ使うのではなく、配列にしたい意志をJavaScriptに知らせる必要があります。このため実際にはJavaScriptにオブジェクトの作成を依頼するかたちになります。



配列の値はキーとともに格納される

オブジェクトかどうかはともかく、いったん配列を作成したら、配列にデータを格納して、データにアクセスすることができます。配列に格納されたデータを取得するときはキーが鍵になります。データ断片に関連づけられた一意的キーを使ってデータにアクセスします。インデックス配列の場合、アクセスしたい配列要素のインデクスを使います。

```
showTime[0] = "12:30";
```

配列変数の名前。

配列のインデクスは角括弧で囲みます。

配列に格納された値。

このコードは配列 showTime の最初の値に時刻を設定しています。配列の値をひとつずつ設定たくないときは、配列を最初に作成するときに配列全体を初期化することもできます。

```
var showTime = [ "12:30", "2:45", "5:00", "7:15", "9:30" ];
```

配列作成の前半は変数の作成と同じです。

配列の値のリストを角括弧で囲みます。

セミコロンを忘れないように。

配列の値のリストはカンマで区切れます。

ちょっと待ってください。このコードにはオブジェクトに関する記述がどこにもありません。空のオブジェクトを作成せずに、いきなり配列に含まれる値をもとに配列（オブジェクト）を作成しています。角括弧で囲まれた要素のリストがそのまま配列になっています。データをもとにして作った配列をさっそく使ってみましょう。

```
alert("レイトショーは " + showTime[4] + " からです。");
```

配列は複数のデータ断片を
ひとつの場所に格納します。

配列の最後の値を取得します。

レイトショーは9:30からです。

OK



配列の真実

今週のインタビュー：データ格納屋さんの頭の中

Head First：はじめまして、配列さん。複数のデータ断片を格納するのがお得意だそうですね。

配列：はい、そうです。50 個の文字列や 300 個の数値を格納する場所が必要なときは、私が力になりますよ。

Head First：面白そうですね。でもすでに通常の変数でも多数のデータを格納できていませんか？

配列：まあ、裸足で外に出ても平気な人もいますから。やり方はひとつとは限りませんしね。でも、私だったら通常の変数よりもっといいやり方で多数の情報を格納できますね。

Head First：たしかに、私も靴を履くほうがいいですね。それはともかく、どれくらい違いがあるんですか？

配列：日記に喻えて説明しましょう。毎日書いている日記を数年後に読み返すとき、どうしますか？

Head First：日記を開けば、当時のことが書かれていますよね。それがどうかしましたか？

配列：日記を開くとき、月日の経過にそって連続して書かれていることを前提にしてますよね。もし日記のページがでたらめな順番だったなら、読み返すのが大変ですね。

Head First：なるほど、日記はそうですね。では配列にはどのようにデータが格納されるのですか？

配列：日記と同じように配列も簡単にアクセスできるようになっています。たとえば去年の 6 月 6 日の日記を読み返すとき、日記の 124 ページを開いたとします。配列にも日記のページ番号に相当するものがあって、キーと呼ばれています。

Head First：配列のインデクスは聞いたことがありますけど、キーは初耳ですね。

配列：そうでしたか、失礼しました。キーはデータを参照するときに使う情報です。インデクスはキーの種類のひとつで、数値がキーになります。キーはインデクスより汎用的な用語です。日記のページはキーであり、インデクスもあります。ある数値をもとにデータを参照するとき、キーとインデクスはまったく同じ意味です。

Head First：なるほど。でもまだわからないことがあります。ループ処理のときは何を使うのでしょうか？

配列：うーん、特に何も使いませんね。データを格納するときは、ループを使わなくても、すごく簡単に処理できますし、もちろん、ループでデータを順番に処理することもできますよ。

Head First：それはどうやって？

配列：ループを制御するときは数値カウンタを使うことが多いですよね。配列に格納されたデータを順番に処理するときは、配列のインデクスを数値カウンタとして使えばいいんです。

Head First：配列インデクスはループカウンタとして使える、ということですか？

配列：まさにそのとおりです。

Head First：すごいですね。

配列：はい、データをループ処理するスクリプトが私を必要とするのは、そういうわけなんです。配列のデータ全体をループ処理するコードが、ほんの数行で書けてしましますから。すごいでしょ。

Head First：目に浮かびます。今日はループさんとの関係もお話しいただいて、ありがとうございました。

配列：どういたしまして。またいつでもいらしてください。



自分で考えてみよう

Mandangoで使うコードを書いてください。処理の内容は、まず座席の配列を作り、次に配列から座席をループで順に調べて、各座席が空いているかアラートで表示します。

素朴な疑問に 答えます

Q: 終わりのないループをforループで実現できますか?

A: できますね。無限ループというやつです。脱出できないループ、空間と時間があるかぎり、別のページが読み込まれるまで、いつまでも繰り返される無限ループ。スクリプトで他の処理が何もできなくなるので、無限ループは悪とされています。アプリケーションが固まってしまうのと同じです。Windowsがブルースクリーンになる前兆とまでは言いませんが。無限ループに陥るのは、ループカウンタが適切に更新されていないか、あるいはループのテスト結果が `false`にならないためです。このことを理解して、ループでのテスト条件と更新のロジックは注意深く二重三重に見直しましょう。もしスクリプトが固まって何もしていなないように思えたら、無限ループに陥っているとみて間違ひありません。

Q: forループのアクション部で複文を使えますか?

A: えます。最も簡単なループのシナリオでは、複文を使うことになるでしょう。現実的なほとんどのループ処理では、複数の文を実行するからです。

：ループの条件テストがfalseのとき、最後の回のアクション部は実行されますか？

A: 実行されません。ループのアクション部はテスト条件の結果がtrueのときだけ実行されます。`false`になると、他のコードはいっさい実行されずにループから抜けます。

 : インデックス配列のインデクスはかならず 0 から始まるので
すか?

A: デフォルトは0から始まります。この動作を無効にして0以外の数値はじめることも、やろうと思えばできます。あえてそうする理由がなければ、デフォルトのままにした方が余計な混乱がなくてすみます。

Q: 配列に格納されるデータは、どれもすべて同じ型でなければならないのですか？

A: そうでなくてかまいません。配列をループで使うそもそも目的は、同じ種類のデータの集まりを順に処理することにあるので、そのため、配列のデータが同じ型であることは重要です。たとえば平均値を計算するとき、数値以外に論理値があっても意味がありません。もちろん、配列のデータはすべて同じ型でなければならない、というわけではありませんが、同じ種類のデータの集まりを格納するときは、同じ型のデータを配列に格納するのが一般的です。

自分で考えてみよう の答え

配列インデックスは 0 から始まるので、ループカウンタも 0 から開始します。

ここでは数値の 9 を使うより配列オブジェクトの length プロパティを使った方がベターです。そうすれば配列の座席の数が変わってもコードを変える必要がありません。

ループカウンタを配列インデックスとして使って配列の値を順に処理します。

座席が空席(true)かそうでないか(false)によってアラートの表示を変えます。

Mandango で使うコードを書いてください。処理の内容は、まず座席の配列を作り、次に配列から座席をループで順に調べて、各座席が空いているかアラートで表示します。

配列の値はカンマで区切れます。

ループカウンタをインクリメントします。

座席配列を作成し論理値で初期化します。

```
var seats = [ false, true, false, true, true, true, true, false, true, false ];
```

```
for (var i = 0; i < seats.length; i++) {
```

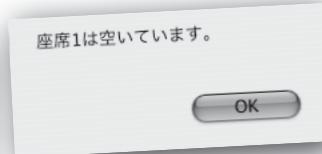
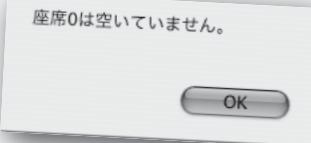
if (seats[i])

```
    alert("座席 " + i + " は空いています。");
```

else

```
    alert("座席 " + i + " は空いていません。");
```

}



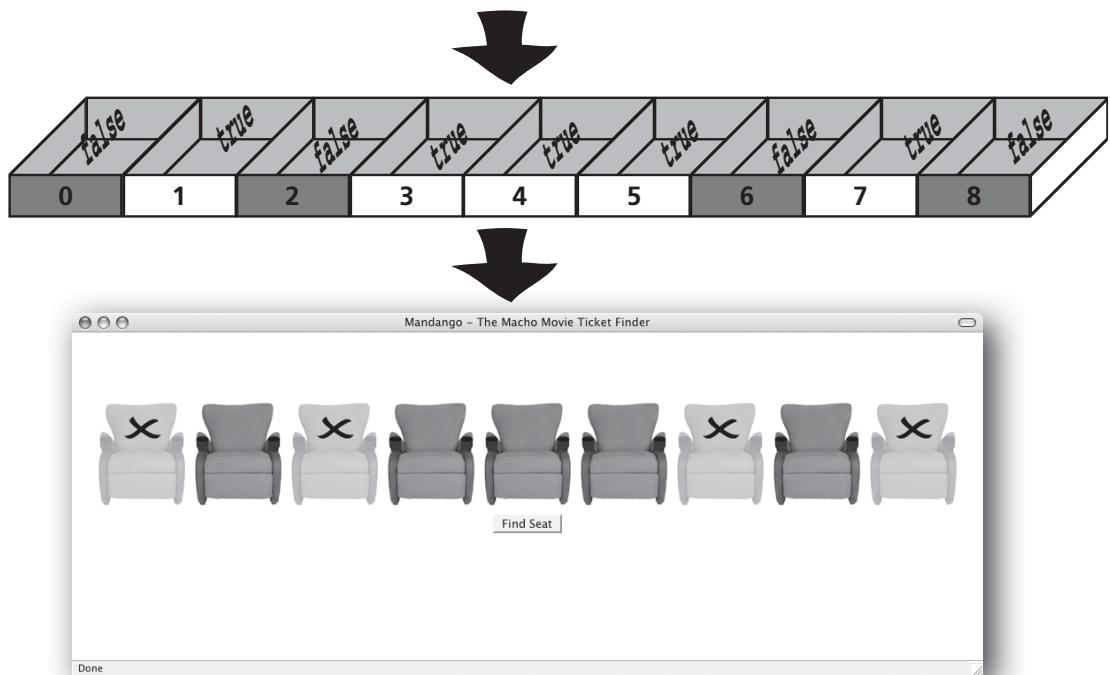
重要ポイント

- for ループでは JavaScript コードの断片を指定された回数繰り返します。
- インクリメント(++)とデクリメント(--)はループカウンタを更新する手段を提供する演算子です。
- 配列を使うと複数のデータ断片をひとつの場所に格納できます。
- 配列には複数の情報断片が保持されていますが、変数名はひとつになります。
- インデックス配列はインデクスと呼ばれる数値キーを使ってアクセスできます。
- インデックス配列はループ処理に適しています。配列データを順にループ処理するときループカウンタが使えるからです。

JavaScriptからHTMLを操作する

Mandangoの座席の状態は論理値配列で表現されています。そこで座席の状態が Mandangoのウェブページに反映されるように、この配列をHTML画像に変換する必要があります。

```
var seats = [ false, true, false, true, true, true, false, true, false ];
```



一見問題なさそうですが、論理値配列をウェブページの座席画像に対応させるコードがありません。どのようなコードを書けばいいのでしょうか。



JavaScriptの座席配列とMandangoページの座席画像を結びつけるにはどうしたらいいでしょう？

Mandangoの座席を表示する

JavaScriptの配列をHTML画像に結びつけるには、画像が簡単に配置できるようにして、席の状態に対応する画像を決める必要があります。まず後者から着手してみましょう。

この座席画像はMandangoの座席検索で使用されます。
選択された座席はハイライト表示されます。



席の状態に対応するこれらの画像は、ページに表示される席の画像のsrc属性に代入されます。

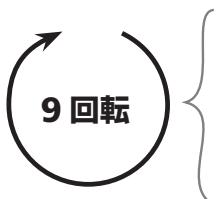
```

```

このIDは配列を座席の
画像に対応づける際に
重要です。配列インデクス
と同じように、0から
8までの値になります。

論理値配列をループでまわして、ページの各タグに座席の画像を設定します。このとき座席配列をループでまわしたのとほとんど同じ作業を行います。ループの中で行われるアクションが違うだけです。

- 1 カウンタ変数*i*を0で初期化します。
- 2 *i*が配列の長さ(9)より小さいかチェックし、そうであればステップ3に進みループのサイクルを続けます。そうでなければループを終了します。
- 3 ループのアクションコードを実行します。この例では座席の画像を設定します。
- 4 *i*をインクリメントしてステップ2に戻ります。



initSeats () の詳細

Mandangoの座席の初期化は initSeats() で実行されます。座席を初期化するループを使って JavaScript の配列を HTML の座席画像に対応付けます。

インデックス配列は 0 から開始されるので、ループカウンタは 0 から開始します。

テスト条件ではループで座席をすべて調べたかチェックします。

この initSeats() を呼び出すと初期化が実行されます。ページの座席の画像が設定されます。
for ループの初期化設定と混同しないように。

```
function initSeats() {
    // すべての座席を初期化します
    for (var i = 0; i < seats.length; i++) {
        if (seats[i]) {
            // 座席を空席に設定
            document.getElementById("seat" + i).src = "seat_avail.png";
            document.getElementById("seat" + i).alt = "Available seat";
        } else {
            // 座席を空席でないに設定
            document.getElementById("seat" + i).src = "seat_unavail.png";
            document.getElementById("seat" + i).alt = "Unavailable seat";
        }
    }
}
```

座席の値が true なら、座席を空席に設定します。

毎回のループごとにループカウンタから座席画像の ID を作成します。

ループカウンタをインクリメントします。

```
<body onload=initSeats();>
<div style="margin-top:75px; text-align:center">
    <img id=seat0 src="" alt="" />
    <img id=seat1 src="" alt="" />
    <img id=seat2 src="" alt="" />
    <img id=seat3 src="" alt="" />
    <img id=seat4 src="" alt="" />
    <img id=seat5 src="" alt="" />
    <img id=seat6 src="" alt="" /> 座席画像ごとに
    <img id=seat7 src="" alt="" /> src 属性と alt 属性が
    <img id=seat8 src="" alt="" /> 動的に調整されます。
</div>
</body>
</html>
```

この座席画像の ID は “seat6” です。



それほどマッショでない座席の探し方

座席が初期化できたら、Mandangoの要となる座席検索に着手できます。セスとジェイソンは、いきなり3席探すのではなく、まず個々の座席が空席かどうかを調べることにしました。まず簡単な処理からはじめて、徐々にアプリケーションを構築する方がいいと考えたからです。空席を探したいので、まず最初にスクリプトに必要なのは、選択した座席を把握する変数です。

この変数には選択された座席が格納されますが、スクリプトの実行後も値を保持しておく必要があるので、グローバル変数にします。`findSeat()`は、ユーザが座席を探す処理を扱います。選択された座席のインデックスが変数`selSeat`に格納されます。

selSeatはわかったけど、
選択されない席を示す
値はどうなる？



セスがいい質問をしました。`selSeat`は選択された座席のインデックスを格納するので、値の範囲は0から8の間になります。しかし、ユーザがまだ座席を選んでいない状態を区別する必要があります。そこで、座席が選択されていない状態を示すために値を-1にします。このため`selSeat`は-1で初期化される必要があります。

座席が選択されていない状態に
するため、変数`selSeat`を
-1で初期化します。

`var selSeat = -1;`

選択された座席のための変数もできたので、`findSeat()`を作りましょう。この関数は座席の配列の各座席を順に空席かどうか調べて、空席であればユーザにその座席を予約するかどうかを尋ねます。この最初のバージョンのMandangoは、マッショな男を喜ばせるものではありませんが、開発の方向は間違っていません。





JavaScript マグネット

MandangoのfindSeat()では、空席を探し、見つかったら、予約するかしないかユーザに訊ねます。マグネットでコードを完成させてセスとジェイソンを助けてあげましょう。

```
function findSeat() {
    // 座席がすでに選択されていたら、座席を再初期化します
    if ( ..... >= 0) {
        ..... = -1;
        ....();
    }

    // 座席が空席かどうか順に調べます
    for (var i = 0; i < seats.length; i++) {
        // 現在の座席が空席か調べます

        if ( ..... ) {
            // 座席を選択された座席に設定して座席の画像を更新します
            = i;
            document.getElementById("seat" + i). ..... = "seat_select.png";
            document.getElementById("seat" + i). ..... = "Your seat";

            // 座席を予約するかユーザに訊ねます
            var ..... = confirm("座席 " + (i + 1) + " が空いています。予約しますか? ");

            if ( ..... ) {
                // ユーザがこの座席を断ったので、座席を選択されない状態にして検索を続けます
                ..... = -1;
                document.getElementById("seat" + i). ..... = "seat_avail.png";
                document.getElementById("seat" + i). ..... = "Available seat";
            }
        }
    }
}
```





JavaScript マグネットの答え

MandangoのfindSeat()では、空席を探し、見つかったら、予約するかしないかユーザに訊ねます。マグネットでコードを完成させてセスとジェイソンを助けてあげましょう。

```

function findSeat() {
    // 座席がすでに選択されていたら、座席を再初期化します
    if (selSeat >= 0) {
        selSeat = -1;      ← selSeat が-1以外だったら
                            次の検索を開始し、
                            座席をリセットします。
        initSeats();
    }

    // 座席が空席かどうか順に調べます
    for (var i = 0; i < seats.length; i++) {
        // 現在の座席が空席か調べます
        if (seats[i]) {   ← 座席が空席なら seats[i] は
                           trueになります。
            // 座席を選択された座席に設定して座席の画像を更新します

            selSeat = i;
            document.getElementById("seat" + i).src = "seat_select.png";
            document.getElementById("seat" + i).alt = "Your seat";

            // 座席を予約するかユーザに訊ねます
            var accept = confirm("座席 " + (i + 1) + " が空いています。予約しますか？");
            if (!accept) {   ← ユーザがこの座席を断ったので、座席を選択されない状態にして検索を続けます
                // ユーザがこの座席を断ったので、座席を選択されない状態にして検索を続けます
                selSeat = -1;
                document.getElementById("seat" + i).src = "seat_avail.png";
                document.getElementById("seat" + i).alt = "Available seat";
            }
        }
    }
}

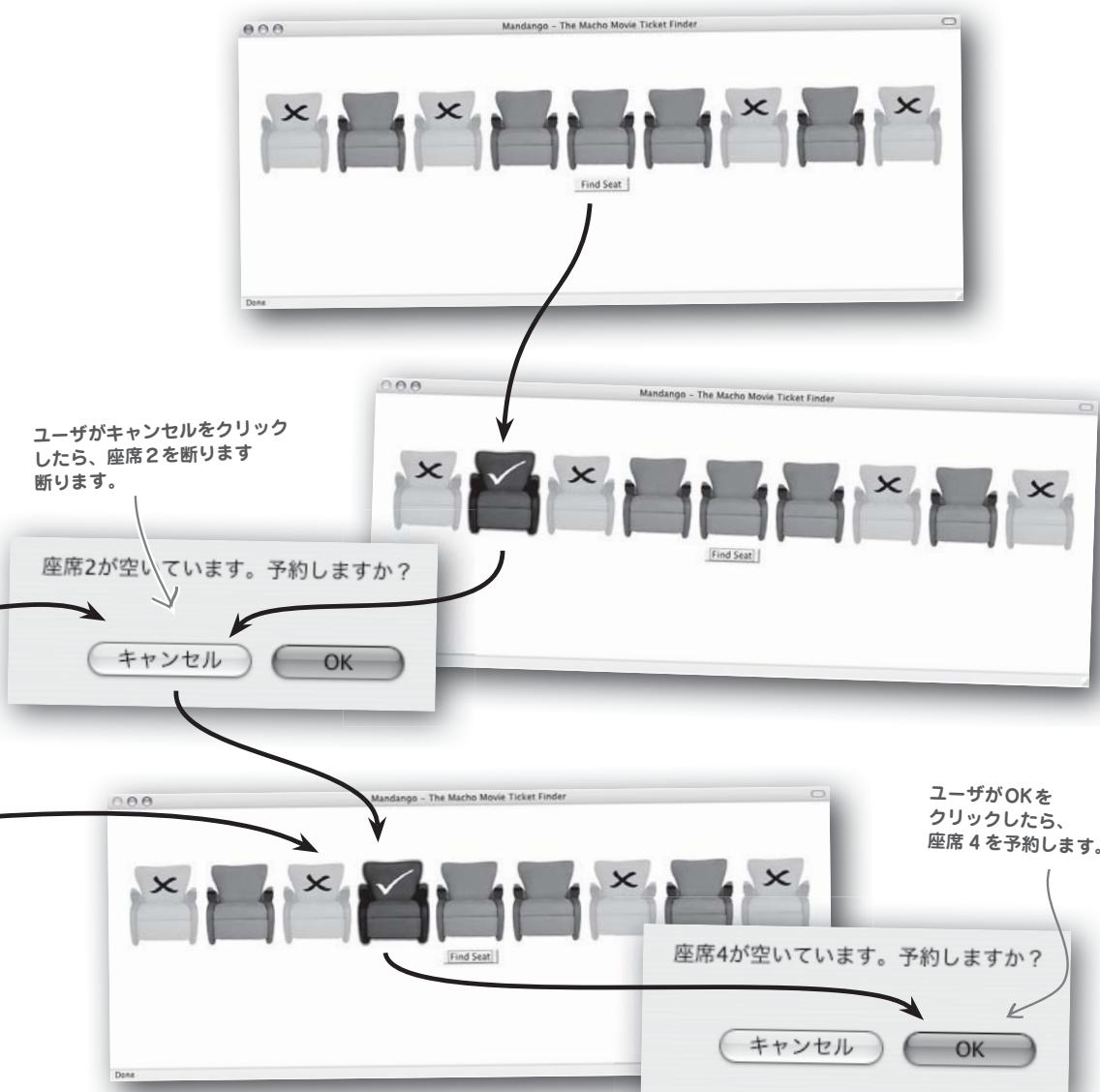
```

ほとんどのユーザは
0ではなく1から
数えるので、ユーザに
表示する座席番号は
1だけ増やします。

ユーザが空席を
予約したか
チェックします。

座席検索をテスト運転してみる

座席をひとつ探すバージョンのMandangoは、`for`ループと配列を使って空席を探します。マッチョ仕様にはなっていませんが、とりあえず機能します。



ループが終わらないのは問題です

Mandangoが空席を探す処理は、動くには動いていますが、ループがどこで終わったらいいかわからぬところに問題があります。ユーザがOKをクリックして座席を予約した後でも、スクリプトは残りの空席でループし続けてしまいます。



ループには脱出条件が必要

座席の検索があまりに熱心すぎてループが終わらないので、ジェイソンは `findSeat()` の `for` ループを順に詳しく調べることにしました。

```

for (var i = 0; i < seats.length; i++) {
    // 現在の座席が空席か調べます
    if (seats[i]) {
        // 座席を選択された座席に設定して座席の画像を更新します
        selSeat = i;
        document.getElementById(seat + i).src = "seat_select.png";
        document.getElementById(seat + i).alt = "Your seat";

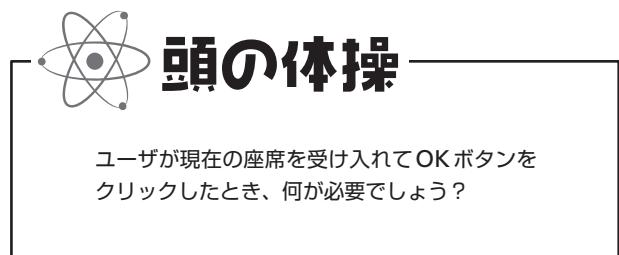
        // 座席を予約するかユーザーに訊ねます
        var accept = confirm("座席" + (i + 1) + "が空いています。予約しますか？");
        if (!accept) {
            // ユーザがこの座席を断ったので、座席を選択されない状態にして検索を続けます
            selSeat = -1;
            document.getElementById("seat" + i).src = "seat_avail.png";
            document.getElementById("seat" + i).alt = "Available seat";
        }
    }
}

```

ユーザが空席を予約しても、何も起こらずループが続きます。

`confirm()` はユーザにイエスかノーを訊ねます。結果は `true` (イエス) か `false` (ノー) で返ります。

ユーザがキャンセルをクリックしたとき、`selSeat` は `-1` に設定され、ループは続いますが、ユーザが座席を予約したときのコードがありません。`selSeat` が現在の座席を記憶しているのはいいのですが、ループを止めるコードがないのです。



アクションの中の break

Mandangoのコードは、ユーザが席を予約したときループから抜けられない点に問題があります。これを修正するのなら、配列の長さより大きな値をカウンタに設定してforループから抜けるように仕掛けるといいでしょう。

```
i = seats.length + 1;
```

強制的にテスト条件がfalseになるようにしてループを終わらせていますが、ループを終わらせるにはもっと良い方法があります。

このコードのちょっとした仕掛けはちゃんと動きますが、ループ条件をだすためにループカウンタをいじるよりもっといい方法があります。break文は、ループのコードをはじめとして、コードのセクションから抜け出るために設計されています。

```
break;
```

ループを即座に
脱出します。

ループでbreak文になると、テスト条件をまったく無視して、即座にループから抜け出ます。break文は問答無用でループから抜け出るための手軽な手段を提供します。

continue文はbreak文と密接な関係にあります。continue文も現在のループサイクルをスキップしますが、ループそのものからは抜けません。言い換えると、continueを使うと次のサイクルにループを強制的にジャンプさせることができます。

```
continue;
```

現在のループサイクルから
抜け出し、次のサイクルに
続きます。

break文とcontinue文はループの制御を調整するときに非常に便利です。セスとジェイソンが探しているMandangoのループ問題はbreak文で解決します。

break文の使い方が
わかったよ。これで不要な
ループから抜けられるね。



素朴な疑問に答えます

Q : break文を使う場合、forループのアクション部の残りは実行されますか？

A: 実行されません。break文はループを強制終了するので、ループの通常のフローは完全に停止します。

Q: ループカウンタをいじってループを強制終了させるのはよくないですか?

A: ループカウンタを本来の目的で使用しない場合、思わぬバグになる危険があります。期待通りに配列の要素をカウントするのではなく、カ

ウンタを配列の有効範囲外の値に強制的に変更してループを終わらせるこ
ともできますが、一般には、ループカウンタが変更される場所をループの更新部だけに限定した方が危険が少ないで
しょう。ちょっとした仕掛けが必要にな
ることもありますが、このケースは該当しません。`break`文を使えば、よけいな混乱を心配することなく、ループから
きれいに抜けられます。

自分で考えてみよう

MandangoのfindSeat()のforループでは、ユーザが座席を予約したとき、ループから抜けるための助けが必要です。空欄になっているコードを埋めて、ループから抜ける処理を書いてください。コードの動作を説明するコメントも忘れずに。

自分で考えてみよう の答え

MandangoのfindSeat()のforループでは、ユーザが座席を予約したとき、ループから抜けるための助けが必要です。空欄になっているコードを埋めて、ループから抜ける処理を書いてください。コードの動作を説明するコメントも忘れずに。

```
// 座席が空席かどうか順に調べます
for (var i = 0; i < seats.length; i++) {
    // 現在の座席が空席か調べます
    if (seats[i]) {
        // 座席を選択された座席に設定して座席の画像を更新します
        selSeat = i;
        document.getElementById("seat" + i).src = "seat_select.png";
        document.getElementById("seat" + i).alt = "Your seat";

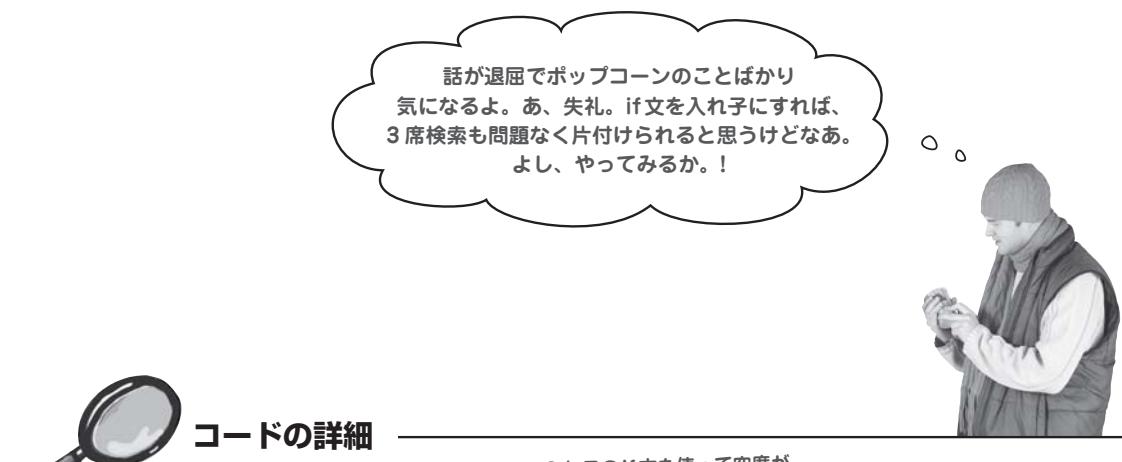
        // 座席を予約するかユーザに訊ねます
        var accept = confirm("座席" + (i + 1) + "が空いています。予約しますか？");
        if (accept) {
            // ユーザが座席を予約しました
            break;
        }
    } else {
        // ユーザがこの座席を断ったので、座席を選択されない状態にして検索を続けます
        selSeat = -1;
        document.getElementById("seat" + i).src = "seat_avail.png";
        document.getElementById("seat" + i).alt = "Available seat";
    }
}
```

Mandangoをマッチョ対応にする

Mandangoのそもそもの意図は、ユーザが空いている3席を探せるようにすることでした。席をひとつ探す機能は動いているので、セスとジェイソンはこれをマッチョのための座席検索に変えようとしています。そこで空いている3席を探す方法が必要になります。

3席とも空席なので
余裕たっぷりです。





コードの詳細

for (var i = 0; i < seats.length; i++) {
 // 現在の座席と次の2席が空席か調べます
 if (seats[i]) {
 if (seats[i + 1]) {
 if (seats[i + 2]) {
 // 座席を選択された座席に設定して座席の画像を更新します
 selSeat = i;
 document.getElementById("seat" + i).src = "seat_select.png";
 document.getElementById("seat" + i).alt = "Your seat";
 document.getElementById("seat" + (i + 1)).src = "seat_select.png";
 document.getElementById("seat" + (i + 1)).alt = "Your seat";
 document.getElementById("seat" + (i + 2)).src = "seat_select.png";
 document.getElementById("seat" + (i + 2)).alt = "Your seat";

 // 座席を予約するかユーザーに訊ねます
 var accept = confirm("座席 " + (i + 1) + " から " + (i + 3) + " が空いています。予約しますか？");
 if (accept) {
 // ユーザが座席を予約しました
 break;
 }
 else {
 // ユーザがこの座席を断ったので、座席を選択されない状態にして検索を続けます
 selSeat = -1;
 document.getElementById("seat" + i).src = "seat_avail.png";
 document.getElementById("seat" + i).alt = "Available seat";
 document.getElementById("seat" + (i + 1)).src = "seat_avail.png";
 document.getElementById("seat" + (i + 1)).alt = "Available seat";
 document.getElementById("seat" + (i + 2)).src = "seat_avail.png";
 document.getElementById("seat" + (i + 2)).alt = "Available seat";
 }
 }
 }
 }
}

入れ子のif文を使って空席が
続けて3席あるかチェック
しています。

空席3席見つかったので、
最初の座席を選択します。

ユーザにどの席が空席か
わかるように3席を
「選択された座席」の
画像に変更します。

ユーザが座席を断ったら、
画像を「空席」に
戻します。

* Mandangoのコードと画像は <http://www.headfirstlabs.com/books/hfjs/> からダウンロードできます。



入れ子になったif文を
もう少しエレガントにできたら
いいのになあ。

&&を使うと論理的でエレガントな良い設計になります

Mandangoで3席を探すための良い方法があります。if文の入れ子を使うバージョンは、とりあえず動くものの、改善の余地があります。このコードをもっとエレガントにできる変更方法があるのです。

エレガント、
だと。おい、からかって
るのか？ でも、
それいいかも！



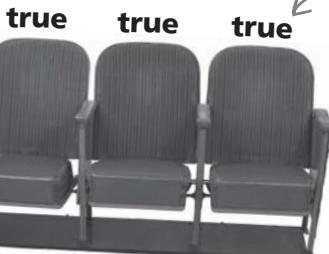
セスの物言いはともかくとして、エレガントにすることは、きれいで、効率的で、理解しやすく、維持管理しやすいコードにすることです。if文の入れ子は、以下の方法によって、ひとつのif文にエレガントにまとめることができます。

論理値を返すAND演算子は
2つの論理値がどちらも
trueか調べます。

```
if (seats[i] && seats[i + 1] && seats[i + 2]) {
```

}

...

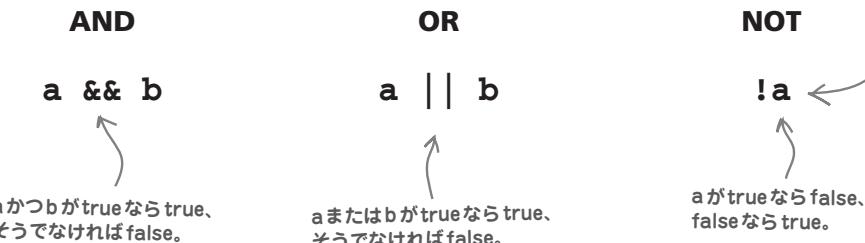


論理演算子のAND (&&) は2つの論理値を比較して、両者がどちらもtrueかどうかを調べます。Mandangoのコードでは、AND演算子を2つ使うことで、3席がどれもtrueかどうかを調べることができます。この条件が成立すれば、3席がすべて空席であることがわかります。これで問題が解決し、コードもすこしえレガントになりました。

論理値を扱う論理演算子について

すでにこれまで、`==`や`<`といった比較演算子をいくつか見てきました。これらの比較演算子のほとんどは、2つの値を比較し、結果を論理値で返します。論理値を返す論理演算子も結果を論理値で返しますが、この演算子は論理値に対してしか使えません。つまり論理値に対する論理比較を実行するわけです。

これはすでに見た
ことがありますね！



論理値を返す論理演算子を互いに組み合わせることで、複雑な判断ができる興味深い論理比較を実行させることができます。

括弧で囲んで論理値を返す
論理式をまとめます。

```
if ((largeDrink && largePopcorn) || coupon)
    freeCandy();
```

コンボセットを買うか、クーポンを持っていると、キャンディのおまけつきです。

論理値を返す

論理演算子を

組み合わせれば複雑な
判断を実行できます。

この例では、AND演算子を使ってラージドリンクとラージポップコーンの組み合わせを調べています。この組み合わせには無料のキャンディがついてきます。このキャンディを手に入れるもうひとつの手段がクーポン券です。つまり、ラージドリンクとラージポップコーンを注文するか、あるいはポップコーンを見せればいいのです。こうした判断は、論理値を返す論理演算子がなければ実行するのが極端に難しくなります。



素朴な疑問に 答えます

Q: 論理値を返す通常の演算子と論理演算子の違いが、いまだにわかりません。

A: どちらも論理値を返す演算子です。両者の違いは演算子が扱うデータ型にあります。「等しい」「等しくない」「より大きい」などの比較演算子は、さまざまなデータ型を扱いますが、AND、OR、NOTの論理演算子は論理値データしか扱いません。論理演算子はtrue/falseしか扱いませんが、論理値を返す通常の演算子はあらゆる

Q: 論理値を返す演算子と一緒に括弧を使うとどうなりますか？

A: 括弧を使うと、演算子が評価される通常の優先順序を変えることができます。括弧の内にある演算子が括弧の外にある演算子よりも優先されます。`largeDrink && largePopcorn`は`()`の中にあるので、`||`よりも先に評価されます。

種類のデータを扱います。

Q: NOT演算子は論理演算子ですか？

A: はい、これは論理値だけを扱うので、論理演算子に該当します。扱う項がひとつなので単項演算子でもあります。

自分で考えてみよう

Mangandoのforループの6番目のサイクル(`i = 5`)の状態を以下に示します。各座席の空きを調べて、空席3席を調べる論理値の論理がどうなるか検証してください。



```
for (var i = 0; i < seats.length; i++) {
    // 現在の座席と次の2席が空席か調べます

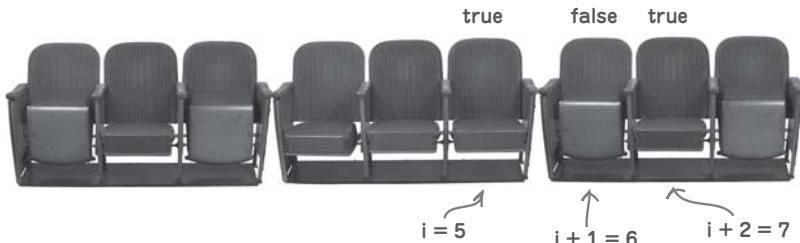
    if (seats[i] && seats[i + 1] && seats[i + 2]) {
        ...
    }
    ...
}
```

..... && && =



自分で考えてみよう の答え

Mangandoのforループの6番目のサイクル($i = 5$)の状態を以下に示します。各座席の空きを調べて、空席3席を調べる論理値の論理がどうなるか検証してください。



```
for (var i = 0; i < seats.length; i++) {
    // 現在の座席と次の2席が空席か調べます

    if (seats[i] && seats[i + 1] && seats[i + 2]) {
        ...
    }
    ...
}
```

..... true $\&\&$ false $\&\&$ true $=$ false

マッチョな座席検索がついに完成

ついにMandangoは空席3席を正しく探せるようになりました。これでどんなマッチョでも満足させられる映画のチケットサービスが実現できます。

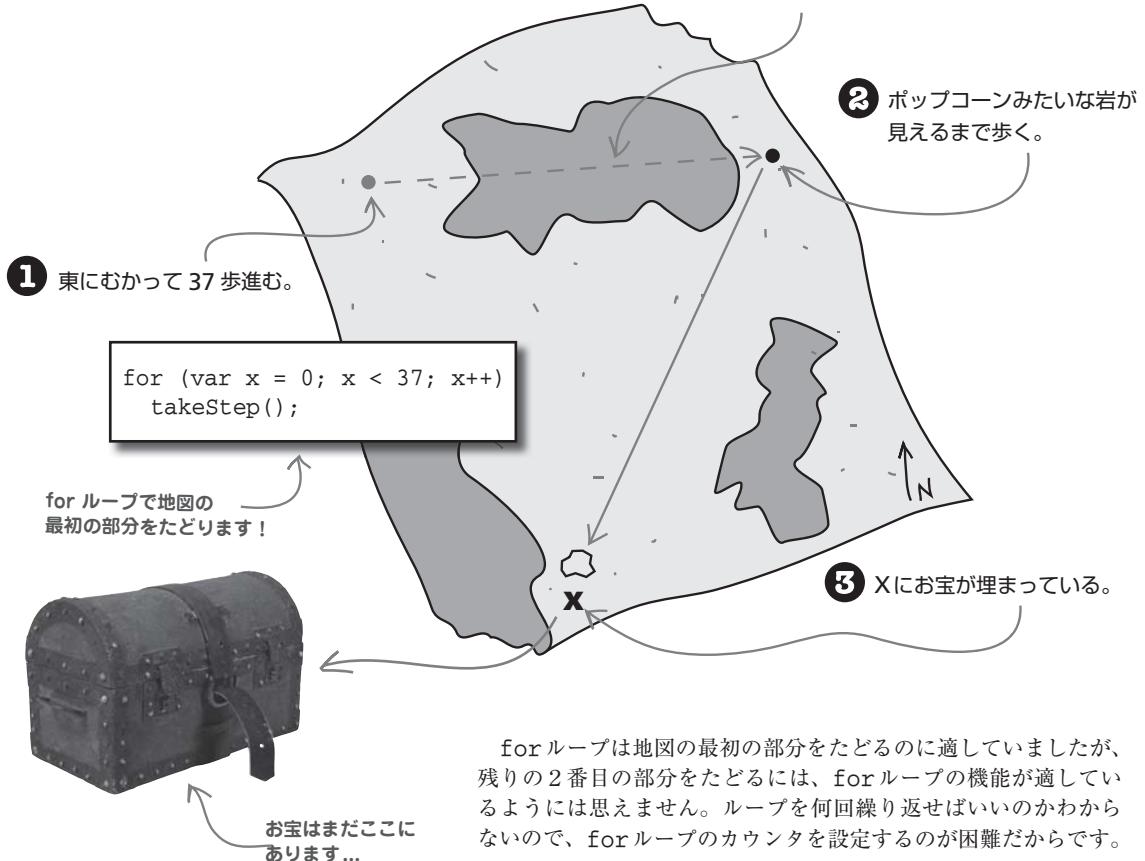
空席3席を予約するか、
ユーザに訊ねます。



お宝地図に戻ると

Mandangoは順調なので、お宝探しに戻ってみましょう。お宝地図です、憶えてますか？

37 歩



頭の体操

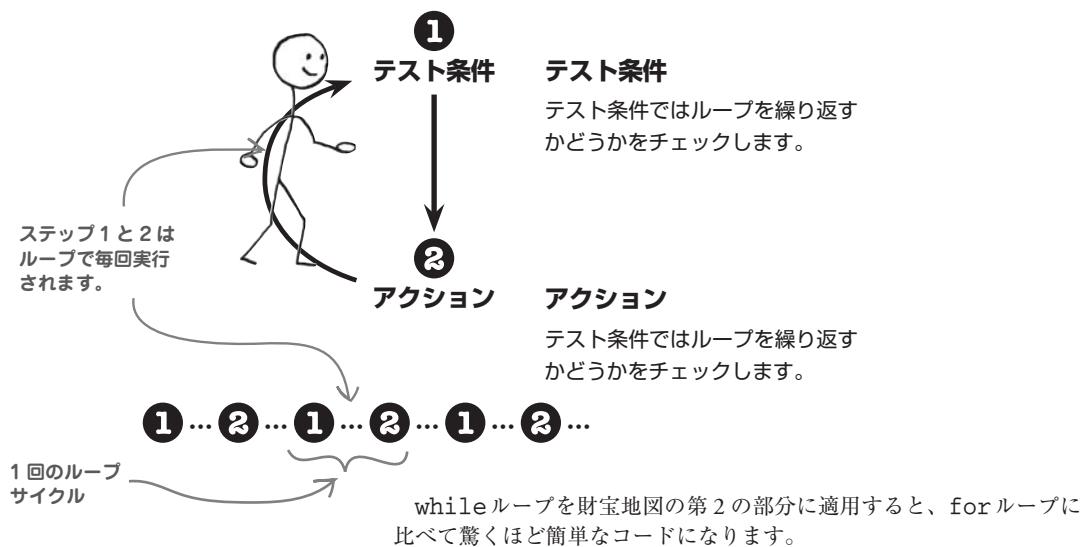
お宝地図の2つの部分の違いは何でしょう？ 地図の2番目の部分をたどるループを作るにはどうしたらいいでしょう？

whileを使って、ある条件が満たされる間だけループする

forループを使ってお宝地図の第2の部分をたどることは可能ですが、もっと良いやり方があります。forループはループカウンタを中心に構成されているのに対して、whileループはある条件が満たされる間だけループを続けます。この条件では必ずしもループカウンタを使う必要はありません。

whileループは2つの部分で構成されます。

whileループは
ある条件がtrueの間
コードを
繰り返します。



テスト条件 ①

テスト結果がtrueのときだけループサイクルを実行します。つまり岩が見えない間だけです。



① while (!rockVisible)

takeStep();

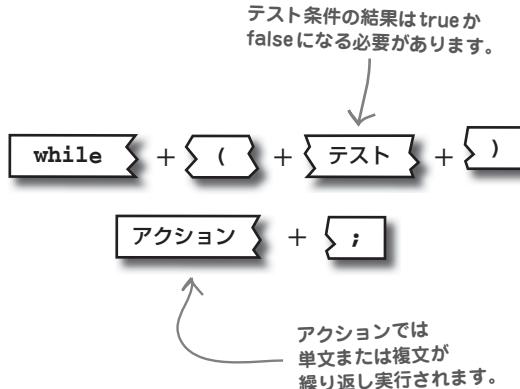
②

このwhileループの動作を以下に示します。

- ①** 岩が見えないことをチェックします。見えなければステップ2に移動します。見えればループを終了します。
- ②** ループのアクションコードを実行します。この場合、`takeStep()`を実行します。

whileループを分解する

whileループはforループよりも構造が簡単ですが、以下の公式に従う必要があります。



要注意！

while ループの テスト条件には 注意しましょう。

whileループにはループを更新するコード断片は組み込まれていないので、テスト条件に影響するコードをループの中に書く必要があります。そうしないと無限ループになる危険性があります。



エクササイズ

映画のカウントダウンの例題を使ったループのコードを書き直してください。ユーザに0より大きな数値の入力を促し、その数値をループの開始値として使って古い映画によくあるカウントダウン(4、3、2、1、スタート)を実行します。forループのかわりにwhileループを使ってください。

```
var count = prompt("0より大きな数字を入力してください。", "10");

if (count > 0) {
```

1

else

```
alert("数字が0より大きくないです。映画はお見せできません。");
```


**エクササイズ
答え**

映画のカウントダウンの例題で使ったループのコードを書き直してください。ユーザに 0 より大きな数値の入力を促し、その数値をループの開始値として使って古い映画によくあるカウントダウン(4、3、2、1、スタート)を実行します。for ループのかわりに while ループを使ってください。

```

変数 count に
値を格納します。
↓
カウントが
0 より大きい
ことを確認
します。
↓
このカウンタは
while ループの
外で作成されて
います。
↓
このループの
アクションは
複文です。
↓
不正データ
↓
var count = prompt("0 より大きな数字を入力してください。", "10");
if (count > 0) {
    var x = count;
    while (x > 0) {
        alert("Starting in..." + x);
        x--;
    }
    alert("映画スタート！");
} else
    alert("数字が 0 より大きくないです。映画はお見せできません。");
}
↓
ユーザに数値の
入力を促します。
↓
0 までカウント
ダウンします。
↓
ループのアクションの
中でカウンタを
デクリメントします。
x=x-1 と同じです。
↓
カウントダウンが
終了しました。
↓
OK
↓
OK

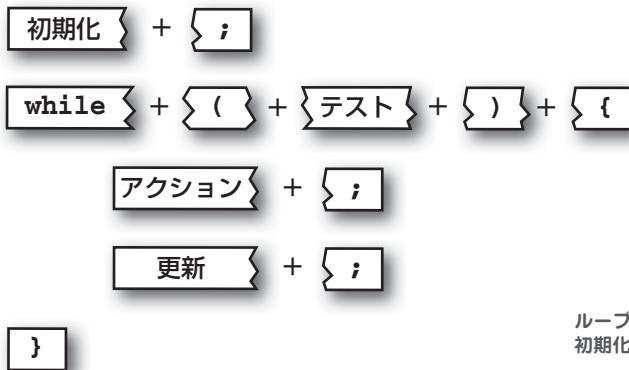
```

重要ポイント

- break 文はループの残りのコードをスキップして即座にループから抜け出ます。
- 論理値を返す論理演算子は意思決定を行うための強力な true/false ロジックを作ります。
- while ループはテスト条件が true である間コード断片を実行します。
- while ループが無限ループにならないように、テスト条件に影響するコードを書きましょう。

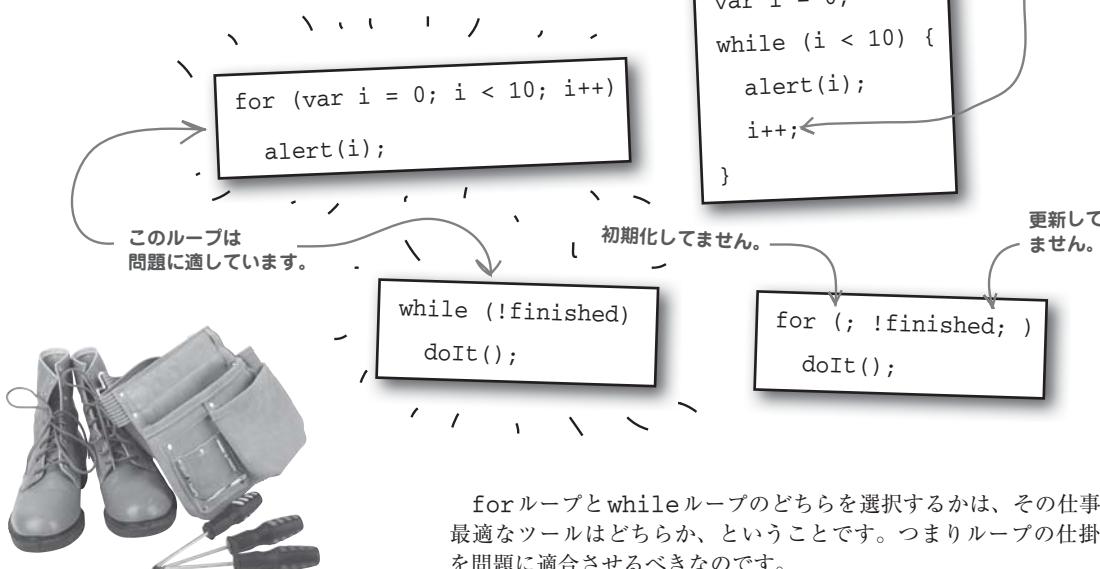
問題に適したループを使う

映画のカウントダウンの例で明らかなように、`for`ループと`while`ループはどちらも同じ問題を解決できる場合があります。実際`for`ループは、`while`ループを使って以下のように再構成することができます。



`while`ループで
実現できることは
`for`ループでも
実現できます。

つまり技術的には`for`と`while`のどちらを使っても同じループを実現できます。とはいえコードを作る観点から判断すると、どちらかが適している場合がほとんどです。それはエレガンスと関係あるのかかもしれません。



特別座談会



今夜の激論：
ゲストは **for** ループと **while** ループです。

for ループ：

あーあ、いつもくどいと言われてるオレたちが揃って出演なわけね。

ちっとも混みいってないけどな。ある種類のループを作るときに、ちょっとした構造を入れてるだけさ。ループのとき数値カウンタを使いたがる人がいるけど、オレがカウンタの初期化と更新を制御しているのを見たら、なんて楽なやり方だろうと思うさ。

違いがわからないな。それだったらオレにもできるぜ。でもオレは、カウントの刻みにぴったり合ってる方がノリがいいんだな。ループをはじめる前にしっかり自分を初期化するのは、そういう理由なんだ。ループの1回ごとに、自分を更新して次に続いているのも、自分に期待されてる動きができるようにするためだし。まあちょっと神経質なところはあるけどな。

ループを構造化する方法がいろいろあるのは、オレもわかっているさ。オレはすみずみまでコントロールされた動きが好きなんだ。

while ループ：

やーね、はじめに断っておくけど、あなたのダンスのステップにすっかりはまっている、なんてことはないわよ。あんな混みいったステップ、ごめんだわ。

そうよね。でもカウンタがループのすべてじゃないでしょ。カウンタを使わないかっこいいループだって、たくさんあるのよ。たまにはあなただって、「はい、これをしばらく続けて」とだけ言うこともあるでしょ。それがわたしのループよ。

あなたの熱心さに拍手。でも決まりきった初期化と更新がなくても、ループを間違なく正確に繰り返すことができるの、あなたもわかってるでしょ。わたしは初期化が必要ない状況でコードを繰り返すことがあるけど、そのとき更新はアクションコードの中で片付けているの。形式にとらわれず、ループに集中できるから、とってもいい気分。

forループ：

そのとおり。二人ともやり方は違うけど、それぞれのやり方でうまくいってるから。まあ、ループに簡単な論理制御があるときは、オレのスタイルの方が若干有利だけだな。

何だと、この野郎！

気持ち悪いこと言うなよな。じゃあな。

whileループ：

まあスタイルの違いよね。それにループはループなんだし。あなたはループを決められた形式にそってコントロールするのが好きで、一方私はもっとくだけたやり方が好きというわけよね。

なんだかんだで、あなたとはうまくヤレると思うわ。

あら、ごめんなさい、うっかり失言しちゃったみたい。

素朴な疑問に**答えます**

Q：whileループはとても単純に見えますが、何か見落としていますか？

A：かならずしもそうではあります。せん。単純だからといって、弱点や制約があるわけではないです。むしろwhileループが強力なので、驚かれているのでは。whileループはテスト条件とアクションコードだけで構成されますが、ほんとにこのツルッとしたループだけで事足りることが多いのです。テスト条件で論理演算子を使っている場合はなおさらです。またアクション部で複文を使えば、もっとたくさんのができます。

Q：while(true)と書いたwhile文は動きますか？

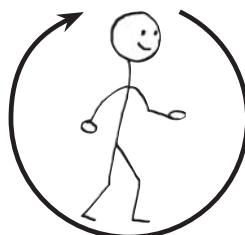
A：はい、動きます。ただし、テスト条件がずっとtrue

Q：ループのアクション部を1回も実行せずに済ますことはできますか？

A：はい、forループもwhileループも、アクションコードを実行するには、その前にテスト条件がtrueでなければなりません。なので、最初にこれがtrueにならないとアクションコードは実行されずループが終了します。

Q：ループの中にループを入れ子にすることはできますか？

A：できますよ。入れ子のループを使うと、二重、三重の繰り返しが実現できます。いまはまだ不思議でしょうが、これはすごいですよ。後で説明しますが、入れ子のループを使うとMandangoが映画館全体の席を探せるようになります。



ループが終わってお宝が見つかる

for ループの後に while ループを続けると、お宝地図を完全にたどることができ、Xでマークされたスポットにあるお宝が見つかります。



これは何かの暗示でしょうか？ while ループの知識と映画のチケットから導きだされるのは、Mandango しかありませんね。



自分で考えてみよう

MandangoのfindSeats()の中のループを、forループではなくwhileループに書き直してください。break文を使うかわりに、新しいループコントロール変数を追加してテスト条件によってループを抜けるようにします。

```
// 現在の座席と次の2席が空席か調べます
if (seats[i] && seats[i + 1] && seats[i + 2]) {
    // 座席を選択された座席に設定して座席の画像を更新します
    ...
    // 座席を予約するかユーザに訊ねます
    var accept = confirm("座席 " + (i + 1) + " から " + (i + 3) +
        " が空いています。予約しますか？");
    ...
    else {
        // ユーザがこの座席を断ったので、座席を選択されない状態にして検索を続けます
        ...
    }
}
// ループカウンタをインクリメント
}
```





自分で考えてみよう の答え

MandangoのfindSeats()の中のループを、forループではなくwhileループに書き直してください。break文を使うかわりに、新しいループコントロール変数を追加してテスト条件によってループを抜けるようにします。

```

var i = 0, finished = false;
while ((i < seats.length) && !finished) {
    // 現在の座席と次の2席が空席か調べます
    if (seats[i] && seats[i + 1] && seats[i + 2]) {
        // 座席を選択された座席に設定して座席の画像を更新します
        ...
        // 座席を予約するかユーザに訊ねます
        var accept = confirm("座席 " + (i + 1) + " から " + (i + 3) +
            " が空いています。予約しますか？");
        if (accept) {
            // ユーザが座席を予約したので完了です
            finished = true;
        } else {
            // ユーザがこの座席を断ったので、座席を選択されない状態にして検索を続けます
            ...
        }
    }
    // ループカウンタをインクリメント
    i++;
}

```

ループカウンタとfinishedを初期化します。

ループカウンタが座席数より小さく、かつfinishedがtrueでない間、ループします。

このループは、これまで見てきたカウンタと論理式のハイブリッドです。whileを使ってハイブリッドなループにするとコードが簡単になります。

finishedをtrueに設定するとテスト条件に影響してループから抜け出します。そのためbreakを使う必要はありません。

Mandangoのループはなかなかいい感じだけど、席が1列しかない映画館なんて現実にはないから。たくさんの列を処理できる方法を調べる必要があるなあ。

映画館の座席データをモデル化する

ジェイソンの言う通りでした。Mandangoを実用化するにはもっと多くの列を扱える必要があります。これまででは1列でも空席を表現する論理値の配列にきれいに対応づけできるので意味がありました。このアイデアを複数の列に拡張するには、配列の次元を追加する必要があります。そうです、2次元配列のことです。



この映画館には9席の
列が4列あります。
窓がそうですね。

この2次元配列の
各項目も論理値です。

実際の座席は9席並んだ列が4列あるので、大きさが 9×4 の配列にする必要があります。

配列インデックスに
別次元を追加します。

	false	true	false	true	false	true	false	true	false
0	0	1	2	3	4	5	6	7	8
1	0	1	2	3	4	5	6	7	8
2	0	1	2	3	4	5	6	7	8
3	0	1	2	3	4	5	6	7	8

2次元配列は配列の配列

2次元配列を作るのに特別な眼鏡なんか必要ありません。実際のところ、2次元配列を作るのは、配列の要素を配列にする点を除けば、通常の1次元配列を作るのと同じです。配列の中の配列（部分配列）を追加すれば2次元になるので、行と列のある表になります。

```
var seats = new Array(new Array(9), new Array(9), new Array(9), new Array(9));
```

最初に部分配列のための配列を作成します。これは1次元です。

4つの部分配列を作ると
4行の配列になります。

次に外側の配列の要素になる
部分配列を作成します。
これで2次元になります。

Mandangoの場合、配列要素の初期値がすでにわかっているので、別のやり方で2次元配列を作成する意味があります。つまり配列リテラルを使うのです。これによって配列の作成と初期化が一度に行えるので一石二鳥です。

```
var seats = [[ false, true, false, true, true, true, false, true, false ],
             [ false, true, false, false, true, false, true, true, true ],
             [ true, true, true, true, true, true, false, true, false ],
             [ true, true, true, false, true, false, false, true, false ]];
```

角括弧が二重にあるので
2次元配列を示します。

部分配列は配列
インデクスで
特定できます。
この場合 0 から
3 になります。

	false	true	false	true	true	false	true	false
0	0	1	2	3	4	5	6	7
1	0	1	2	3	4	5	6	7
2	0	1	2	3	4	5	6	7
3	0	1	2	3	4	5	6	7

最初の論理値リストは、
2次元配列の最初の行を
示します。

座席が埋まって
いたら false。

空席なら true。

2 次元データにアクセスするには キーが 2 つ必要

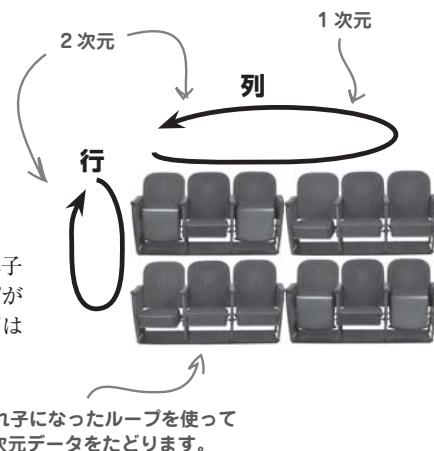
2次元配列のデータへのアクセスは1次元配列へのアクセスと違いはありませんが、追加された配列のインデックスを追加情報として指定する必要があります。具体的には、配列にあるデータの行と列のインデックスを指定する必要があるということです。たとえば、2行目にある4番目の席の値を調べるには、以下のコードを使います。

配列の2行目の
インデクスは1です。

alert(seats[1][3]);

行の4番目の要素の
インデクスは3に
なります(0から開始)。

2次元以上の配列をループでたどるには、各次元ごとにループを入れ子にする必要があります。2次元配列のループでは、全部で2つのループが必要で、ループの中にループがあるかたちになります。外側のループは配列データの行をたどり、内側のループはその行の列をたどります。

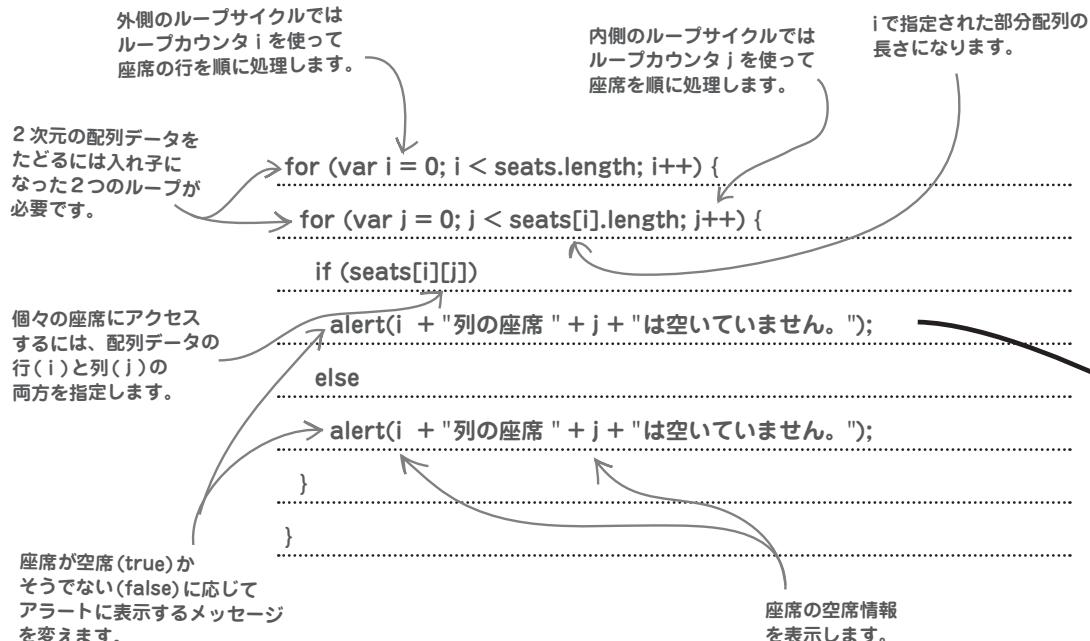


自分で考えてみよう

2 次元の座席配列をループで順にたどり、空席だったらユーザにアラートするコードを書いてください。

自分で考えてみよう の答え

2次元の座席配列をループで順にたどり、空席だったらユーザにアラートするコードを書いてください。



Q: 2次元より大きな配列は作れますか？

A: はい。ただし次元が増えるとデータを視覚化するのに仕掛けが必要になりますが。3次元は現実世界のデータのモデリングでよく使われます。空間の点がx-y-z座標で表現されます。これより大きな次元になるのは、かなり特別な状況でしょう。次元を追加するときは、個々の配列要素を部分配列に置き換えて考えるといいです。

Q: 配列をデータで初期化した後、配列にさらにデータを追加することはできますか？

素朴な疑問に 答えます

A: できますよ。未使用的配列要素に新しいデータを代入すると、配列にデータが追加されます。Mandangoの例では、5番目の行(配列のインデックス4に対応)に新しい部分配列を追加すると、座席の行がひとつ追加されます。seats[4]に部分配列を代入するだけです。Arrayオブジェクトのpush()メソッドを呼んで、配列の終わりに新しい項目を追加することもできます。

Q: 2次元配列の行は大きさを同じにする必要がありますか？

A: いいえ、そうしなくても大丈夫です。ただし、行の大きさ(要素の数)が同じでないと、ループで思わぬ災難に見舞われることもあるので、注意しましょう。というのは、入れ子のループは、部分配列の大きさが一定である前提で組まれていることが多いからです。2次元配列の行の大きさを同じにしないことは可能ですが、かなり危険をともなうので、避けた方が得策です。



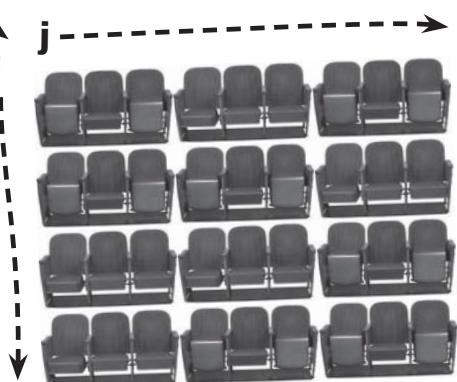
重要ポイント

- 2次元配列を使うとデータの行と列を表構造で格納できます。
- 2次元配列の個々のデータ断片にアクセスするときは、行と列のインデクスを指定する必要がります。
- 2次元配列のデータを順に処理するときは入れ子のループが使えます。
- 通常の配列と同じように、2次元配列も配列オブジェクトリテラルで作成し初期化できます。

2次元の Mandango

すでにコードの一部を動かしてきましたが、Mandangoを1列座席から館内の全座席に対応させるには、2次元データを考慮したスクリプトコードに書き換える必要があります。

2次元の座席配列をたどるにはループカウンタを2つ使います。



Mandangoを1次元から2次元にするには、かなり大幅なコード変更になります。

行を増やすとなると、すごいことになるな。
よし、コードを作り直すとするか。



頭の体操

映画館全体の座席データでMandangoが動くように、2次元配列を使うとMandangoにどんな影響があるでしょう？ どんなスクリプトコードになると思いますか？



2 次元 Mandango 詳細

```

<html>
  <head>
    <title>Mandango - The Macho Movie Ticket Finder</title>

    <script type="text/javascript">
      var seats = [[ false, true, false, true, true, true, false, true, false ],
                   [ false, true, false, false, true, false, true, true, true ],
                   [ true, true, true, true, true, true, false, true, false ],
                   [ true, true, true, false, true, false, false, true, false ]];
      var selSeat = -1;

      function initSeats() {
        // すべての座席を初期化します
        for (var i = 0; i < seats.length; i++) {
          for (var j = 0; j < seats[i].length; j++) {
            if (seats[i][j]) {
              // 座席を空席に設定
              document.getElementById("seat" + (i * seats[i].length + j)).src = "seat_avail.png";
              document.getElementById("seat" + (i * seats[i].length + j)).alt = "Available seat";
            } else {
              // 座席を空席でないに設定
              document.getElementById("seat" + (i * seats[i].length + j)).src = "seat_unavail.png";
              document.getElementById("seat" + (i * seats[i].length + j)).alt = "Unavailable seat";
            }
          }
        }
      }

      function findSeats() {
        // 座席がすでに選択されていたら、座席を再初期化します
        if (selSeat >= 0) {
          selSeat = -1;
          initSeats();
        }

        // 全座席から空席を探します
        var i = 0, finished = false;
        while (i < seats.length && !finished) {
          for (var j = 0; j < seats[i].length; j++) {
            // 現在の座席と次の 2 席が空席か調べます
            if (seats[i][j] && seats[i][j + 1] && seats[i][j + 2]) {
              // 座席を選択された座席に設定して座席の画像を更新します
              selSeat = i * seats[i].length + j;
              document.getElementById("seat" + (i * seats[i].length + j)).src = "seat_select.png";
              document.getElementById("seat" + (i * seats[i].length + j)).alt = "Your seat";
              document.getElementById("seat" + (i * seats[i].length + j + 1)).src = "seat_select.png";
              document.getElementById("seat" + (i * seats[i].length + j + 1)).alt = "Your seat";
              document.getElementById("seat" + (i * seats[i].length + j + 2)).src = "seat_select.png";
              document.getElementById("seat" + (i * seats[i].length + j + 2)).alt = "Your seat";

              // 座席を予約するかユーザに訊ねます
              var accept = confirm((i + 1) + " 列の座席 " + (j + 1) + " から " + (j + 3) +
                " が空いています。予約しますか? ");
              if (accept) {
                // ユーザが座席を予約したので完了です
                finished = true;
                break;
              } else {
            }
          }
        }
      }
    </script>
  </head>
  <body>
    ...
  </body>
</html>

```

mandango.html

2 次元の Mandango の完全なコードです。

座席の空席状態を示す論理値の 2 次元配列を作成します。

ユーザがFind Seatsボタンをクリックして検索を開始したら、座席を初期化し直します。

while ループで各行を順にまわし、for ループで行の座席を順にまわしています。両方のループのいいところです。

```

// ユーザがこの座席を断ったので、座席を選択されない状態にして検索を続けます
selSeat = -1;
document.getElementById("seat" + (i * seats[i].length + j)).src = "seat_avail.png";
document.getElementById("seat" + (i * seats[i].length + j)).alt = "Available seat";
document.getElementById("seat" + (i * seats[i].length + j + 1)).src = "seat_avail.png";
document.getElementById("seat" + (i * seats[i].length + j + 1)).alt = "Available seat";
document.getElementById("seat" + (i * seats[i].length + j + 2)).src = "seat_avail.png";
document.getElementById("seat" + (i * seats[i].length + j + 2)).alt = "Available seat";
}
}

// ループカウンタをインクリメント
i++;
}
}

</script>
</head>

```

ページが最初に読み込まれたとき initSeats() が呼び出されます。



コードの量がかなり多いですが怖がる必要はありません。

Mandangoの文脈にあわせて 2 次元配列の手法を使っています。HTMLコードと画像は <http://www.headfirstlabs.com/books/hfjs/> からダウンロードできます。

9 席の行が 4 行あるので、36 席分の HTML 画像が必要です。ひえー！

ユーザーが Find Seats ボタンをクリックすると findSeats() が呼び出されます。

```

<body onload="initSeats();">
<<div style="margin-top:25px; text-align:center">
<img id="seat0" src="" alt="" />
<img id="seat1" src="" alt="" />
<img id="seat2" src="" alt="" />
<img id="seat3" src="" alt="" />
<img id="seat4" src="" alt="" />
<img id="seat5" src="" alt="" />
<img id="seat6" src="" alt="" />
<img id="seat7" src="" alt="" />
<img id="seat8" src="" alt="" /><br />
<img id="seat9" src="" alt="" />
<img id="seat10" src="" alt="" />
<img id="seat11" src="" alt="" />
<img id="seat12" src="" alt="" />
<img id="seat13" src="" alt="" />
<img id="seat14" src="" alt="" />
<img id="seat15" src="" alt="" />
<img id="seat16" src="" alt="" />
<img id="seat17" src="" alt="" /><br />
<img id="seat18" src="" alt="" />
<img id="seat19" src="" alt="" />
<img id="seat20" src="" alt="" />
<img id="seat21" src="" alt="" />
<img id="seat22" src="" alt="" />
<img id="seat23" src="" alt="" />
<img id="seat24" src="" alt="" />
<img id="seat25" src="" alt="" />
<img id="seat26" src="" alt="" /><br />
<img id="seat27" src="" alt="" />
<img id="seat28" src="" alt="" />
<img id="seat29" src="" alt="" />
<img id="seat30" src="" alt="" />
<img id="seat31" src="" alt="" />
<img id="seat32" src="" alt="" />
<img id="seat33" src="" alt="" />
<img id="seat34" src="" alt="" />
<img id="seat35" src="" alt="" /><br />
<input type="button" id="findseats" value="Find Seats" onclick="findSeats();"/>
</div>
</body>
</html>

```

全座席からマッチョのための席を探す

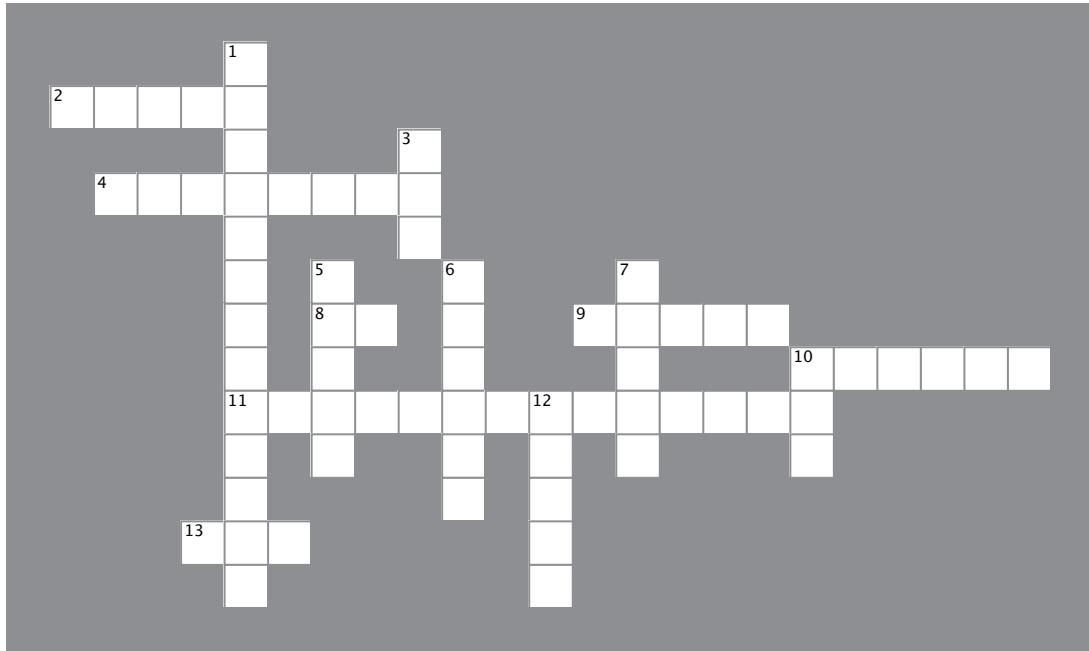
2次元対応ができたので、セスとジェイソンはMandangoを映画館の全座席からマッチョのための席を探せるようにしました。二人とも大喜びです。





JavaScript クロスワード

ずっと座席の話をしてきたので、映画を観たくなられたかもしれませんね。
その前にクロスワードパズルでちょっと気分転換してみませんか？



ヨコのカギ

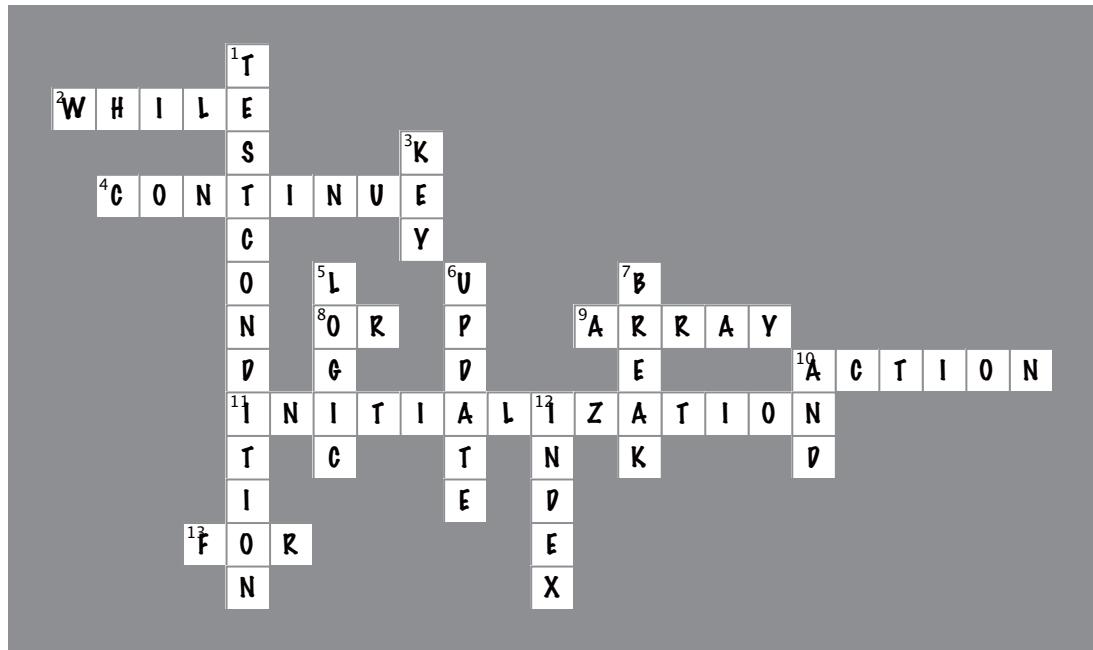
2. テスト条件がtrueの間コードを実行し続けるタイプのループ。
4. 現在のループサイクルから抜けるけれども、ループは続けるときに使う文。
8. aがtrueまたはbがtrueならば a .. b はtrue、そうでなければ a .. b はfalse。この..とは？
9. ひとつの変数に複数のデータ断片を格納できるデータ型。
10. ループの部品のうち、繰り返し実行されるコードに相当するところ。
11. ループの部品のうち、ループの開始時に実行されるところ。
13. カウントに最適なタイプのループ。

タテのカギ

1. ループの部品のうち、論理値の結果にならなければならないところ。
3. 配列の値にアクセスするときに ... が使われます。
5. Boolean operators は論理値に対する演算子であり、結果を論理値で返します。
6. ループの部品のうち、ループコントロールの状態を変更する責任があるところ。
7. ループを即座に終了したいときは、この文を使います。
10. aがtrueかつbがtrueならば a ... b はtrue。そうでなければ a ... b はfalse。この...とは？
12. 配列の値に数値を使ってアクセスするとき必要なのは 。



JavaScript クロスワードの答え

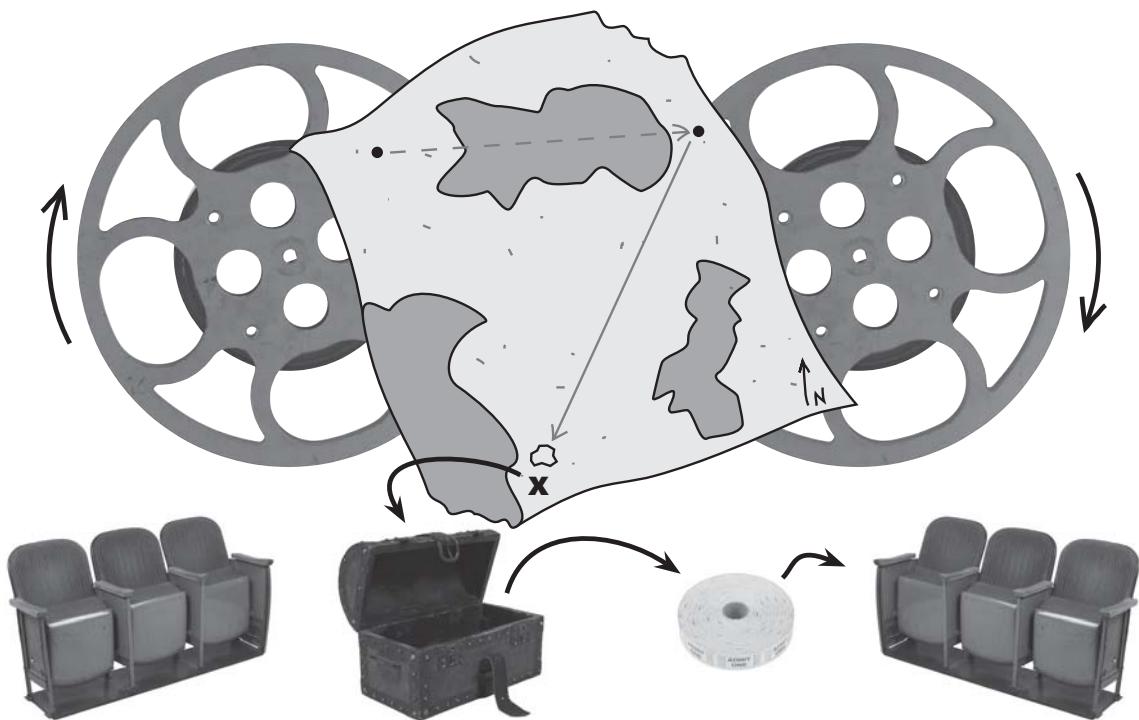


折り畳みページ

脳のところで
折り畳んでから
謎を解決しましょう。

ループと映画の
共通点は何でしょう？

左脳と右脳がご対面！



プロットが循環して物語のあらすじを追うのが難しい映画もあります。
動きとアクションで観客をひきつける映画もあります。
とはいえ、映画は映画にすぎません。

6 章 関数

✳ 節約、再利用、リサイクル ✳



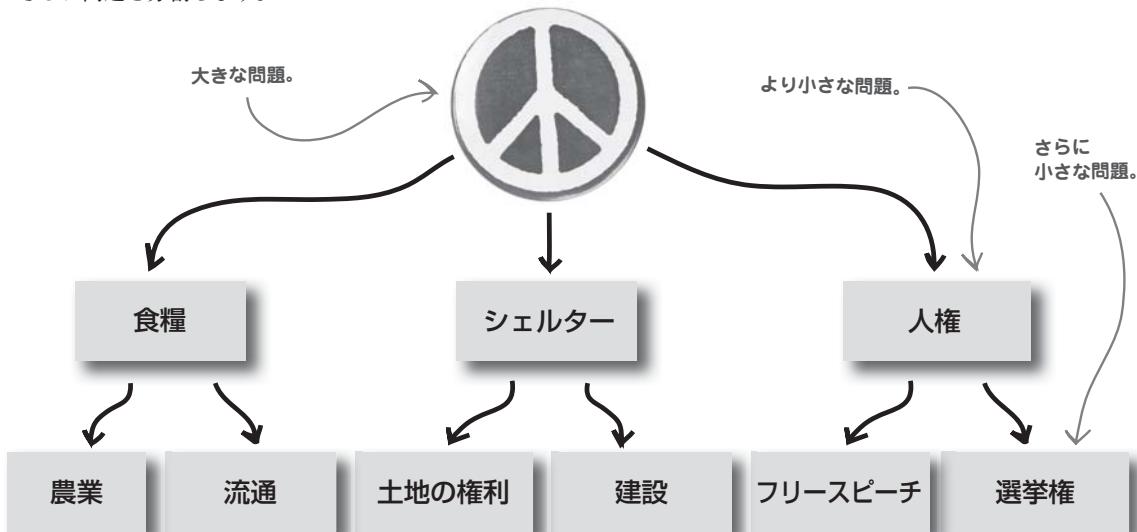
もし JavaScript の世界に環境運動があるとしたら、運動のリーダーは関数です。関数は JavaScript のコードをより効率的に、そしてここが肝心なのですが、再利用できるものにします。関数のアピールポイントは、タスク指向であること、コードの編成に優れていること、問題解決の達人であることです。よくできた履歴書みたいですね。実際、単純なスクリプトを除けば、ほとんどのスクリプトは関数で再編成されることで恩恵がもたらされます。平均関数の二酸化炭素排出量を数字で出すのは難しいですが、スクリプトをできるだけ環境に優しく作ることで、自分の役割を果たすことにしましょう。

すべての問題の根源

ウェブスクリプティングの本質を突き詰めていくと、その目的は問題解決にあります。問題がどれほど大きくても、思慮深く計画をたてれば必ず解はあります。でもほんとに手に負えない問題はどうでしょう？



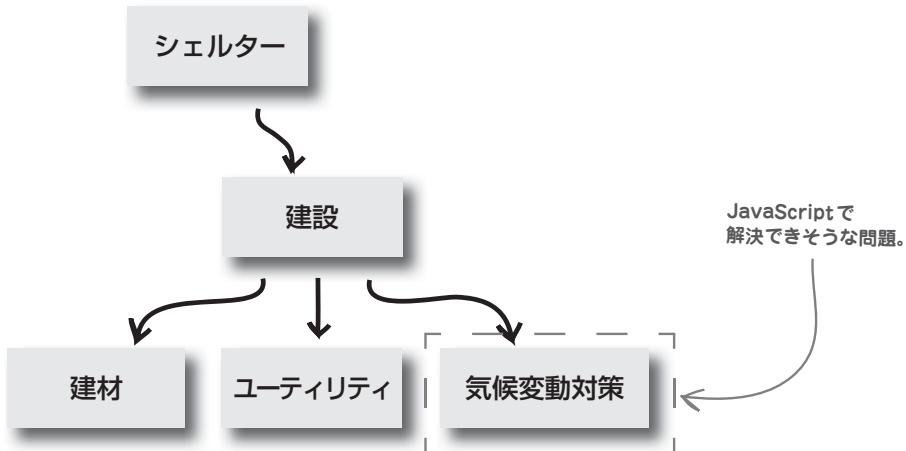
大きな問題を解決するコツは、問題を分割して、取り組みやすい小さな問題にすることです。それでもまだ問題が大きいときは、さらに問題を分割します。



この分割を、どんどんどんどん、続けてください。

大きな問題を小さな問題で解決する

世界平和の問題を小さな問題にどんどん分割していくと、最後にはJavaScriptで扱えるほど小さな問題にたどり着きます。



JavaScriptで扱える気候変動対策があるとすれば、サーモスタッフのスクリプトがそうかもしれません。環境の温度を調整するスクリプトです。最も基本的なサーモスタッフには「HEAT」ボタンがあるでしょう。



このサーモスタッフの暖房のしくみについては何もわかりません。ただ「HEAT」ボタンを押すと暖かくなります。これで気候変動対策は解決ですね。

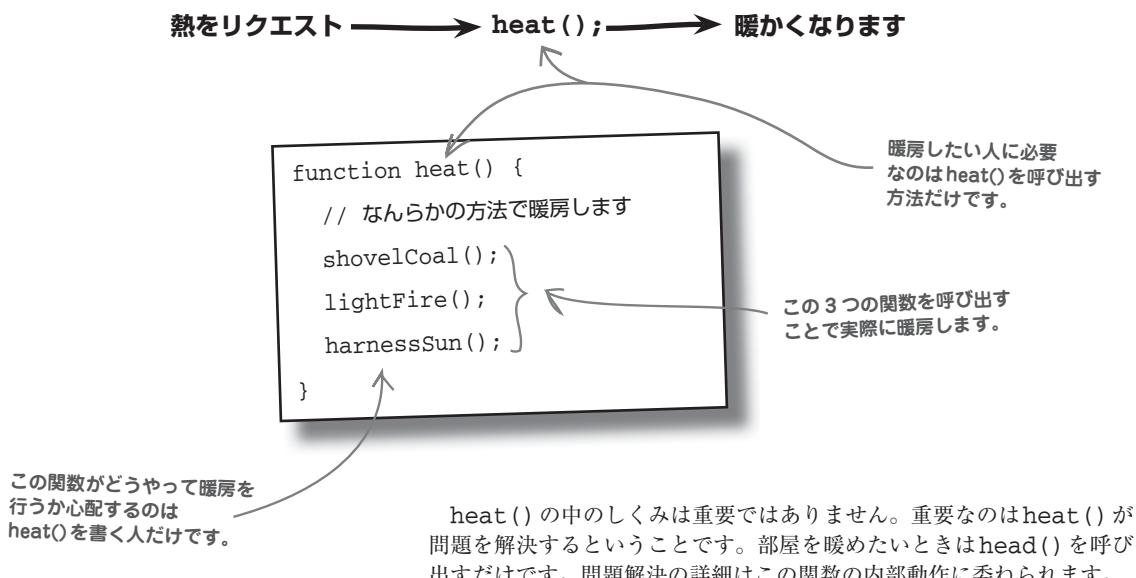
問題解決のための関数

サーモスタットの「HEAT」ボタンはJavaScriptの**関数**に相当します。関数の概念は現実世界のサーモスタットと同じです。誰かが暖かさを要求すると、関数が暖かさを提供します。暖房のしくみの詳細は関数の中にあるので、この関数を呼び出すコードにとっては、詳細は重要ではありません。このように関数を「ブラックボックス」とみなせば、情報の入力と出力がありますが、黒箱の中で起きることは黒箱の責任であり、箱の外にあるコードにとっては重要ではありません。

関数は大きな問題を
小さな問題に変えます。

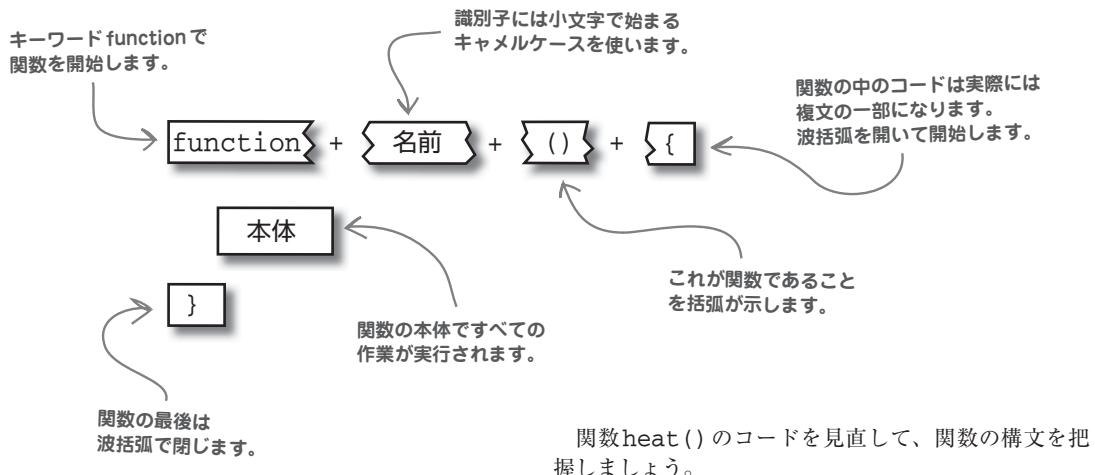


「HEAT」ボタンをJavaScriptのコードに置き換えると、heat()という関数を呼び出すことになります。



関数の仕組み

関数を作りたくなったら、もう問題は解決されたも同然です。関数を作るのに必要なことは、構文規則に従って関数の名前をその関数が実行するコードと結びつけることです。



関数の本体が実際に暖房します。

```
function heat() {  
    // なんらかの方法で暖房します  
    shovelCoal();  
    lightFire();  
    harnessSun();  
}
```

関数の本体を波括弧で囲みます。実際には複文になります。

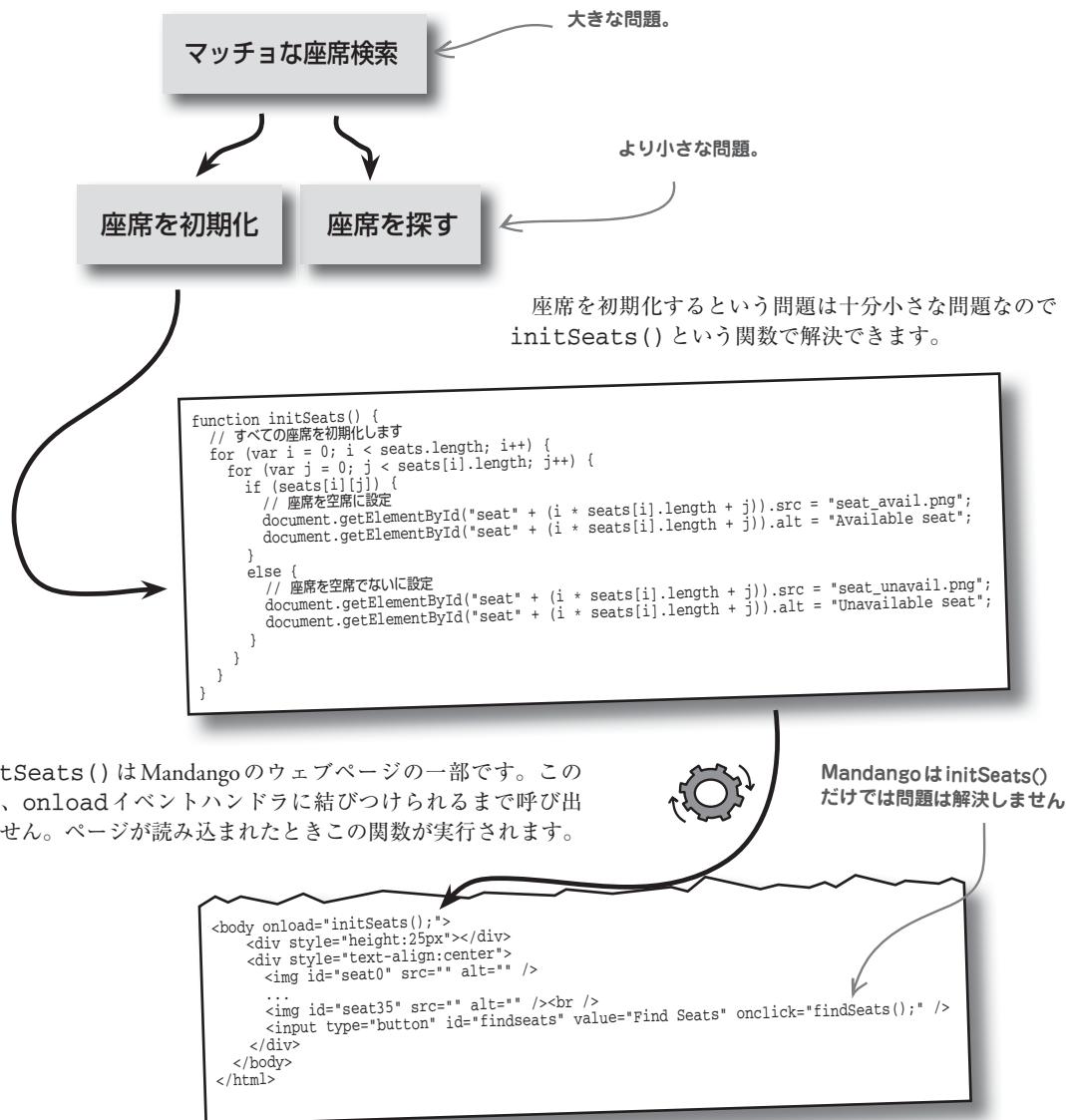


頭の体操

これまで問題を解決するために、いつ関数が使われていたでしょう？

これまで見てきた関数

問題を解決する関数のいい例なので Mandango を見てみましょう。問題は映画館の座席データを初期化することでした。Mandango の問題は以下のように分解されました。



素朴な疑問に 答えます

Q: 関数の命名規則はどうなっていますか?

A: JavaScriptの識別子は小文字で始まるキャメルケース(lowerCamelCase)を使うのが慣習になっています。識別子の最初の単語は小文字で始まりますが、その後の単語は大文字で始まります。なので、たとえば映画のレートをつける関数はrateMovie()になります。映画の上映中でも携帯電話でしゃべり続ける礼儀知らずをつまみ出す関数はremoveInappropriateGuy()になります。

Q: 関数は大きな問題を小さな問題に分割するために存在するのですか?

A: 必ずしもそうではありません。関数を使ってコードを分割するだけでも、コーディングが楽になることもあります。ひとつの問題を解決する

ために、これをいくつかの関数に分割するわけです。この場合、コードを関数に分割するのが理にかなっているのは、それぞれの関数がひとつの仕事を専念できるからです。仕事を何人かで分業するとき、各人がある特定の作業に専念できるようにしますよね。それと同じです。分割された関数は、ひとつの問題だけを解決するケースもありますが、そうでないケースもありますが、仕事を分割することでスクリプトの構造は間違いなく改善されます。

Q: コードの塊をひとつの関数にするかどうか、いつ、どうやって判断したらいいですか?

A: コードを関数にするタイミングを知る魔法は残念ながらありません。でも手掛りはいくつかあります

す。ひとつは重複しているコードがないか探すことです。あちこちに重複したコードがあると、変更などがあったとき、それをすべて同じように修正する必要があるので、まったくいいことがありません。なので、重複するコードはひとつの関数にする候補になります。もうひとつはコードが大きくなりすぎて手に負えなくなったときです。これを論理的なまとまりごとに分割するのです。「分業」の考え方をコードに適用して、仕事をいくつかの関数に分割することを検討しましょう。

Q: 関数は引数を受け取りデータを返すと理解したつもりなんですが、何か見落としてませんか?

A: いいえ、何も見落としていませんよ。多くの関数はデータを受け取りデータを返します。後で説明しますがheat()もデータを受け取るようになります。



エクササイズ

関数がどんな処理を行うのかすぐに理解できるためには、関数の名前はとても重要です。以下の関数に名前をつけてください。小文字で始まるキャメルケースにするのをお忘れなく。

通路側座席(aisle seat)を → リクエスト → 通路側座席のチケットを受け取る



払戻し(refund)を要求 → → 払戻しを受け取る



ポップコーン(popcorn)を → 投げる → ポップコーンが他人に投げつけられる





エクササイズの
答え

関数がどんな処理を行うのかすぐに理解できるためには、関数の名前はとても重要です。
以下の関数に名前をつけてください。小文字で始まるキャメルケースにするのをお忘れなく。

通路側座席(aisle seat)を →
リクエスト → 通路側座席のチケットを受け取る



requestAisleSeat()

払い戻し(refund)を要求 → → 払い戻しを受け取る



getRefund()

ポップコーン(popcorn)を →
投げる → ポップコーンが他人に投げつけられる



throwPopcorn()

この他にも有効な関数名はたくさんあります。
小文字で始まるキャメルケースを使います。

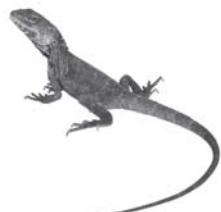
暑くて死にそう。ねえ、暖房を落としてくれない。それとも温暖化の影響かしら？

あーあ、
ぼく快適なのに。



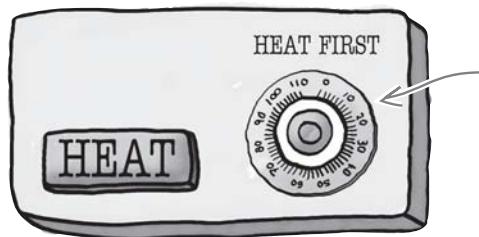
暑すぎて処理できないよ

世界平和を気候変動対策に分割したにも関わらず、思わぬ問題が浮上してきました。「HEAT」ボタンが効きすぎたようです。heat()にもっとデータが必要なのでしょうか。いずれにせよ、修正が必要なようです。



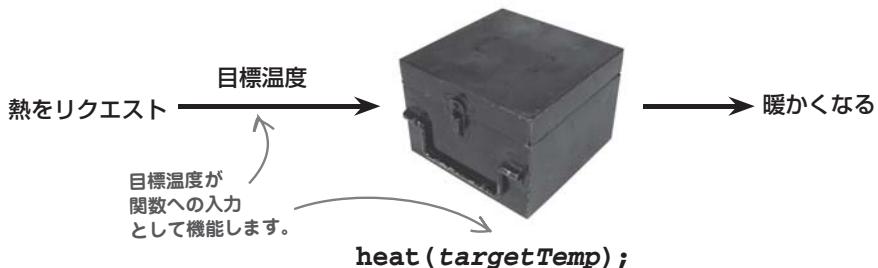
データを増やしてサーモmostatを改良する

サーモmostatに戻りましょう。目標温度を設定していないので、いつ暖房を止めたらいいのか、わからないのが問題です。問題を効果的に解決するのに必要な情報が不足していたわけです。「HEAT」ボタンを押すと、そのままずっと暖房されたままになります。



このノブを使って目標温度を設定することができます。

そこでサーモmostatを改良して、目標温度を入力として受けとれるようにします。これを使って暖房の処理を改良しましょう。



`heat()`を改良するコードを書いてみましょう。目標温度を受け取り、現在の温度が目標温度より低い間だけ発熱するようにします。

ヒント：現在の温度を取得する`getTemp()`という関数を想定して、これを呼び出します。

最初のコードはこうなります。

```
function heat(targetTemp){
```



エクササイズの 答え

`heat()`を改良するコードを書いてみましょう。目標温度を受け取り、現在の温度が目標温度より低い間だけ発熱するようにします。

ヒント：現在の温度を取得する`getTemp()`という関数を想定して、これを呼び出します。

目標温度は引数として
関数に渡されます。

現在の温度が目標温度
より低かったときだけ
暖房します。

```
function heat(targetTemp){
    while (getTemp() < targetTemp) {
        // なんらかの方法で暖房します
        shovelCoal();
        lightFire();
        harnessSun()
    }
}
```

目標温度はwhileループの
テスト条件の一部として
使われます。

関数に情報を渡す

JavaScriptの関数にデータを渡すときは**引数**を使います。引数はいわば入力です。関数の構文を見てください。関数を作ると、括弧の中に引数があるのがわかりますね。

括弧の中には引数を
複数書けます。

function <名前> <(> <引数> <) > < { >

本体

関数の本体の中では引数は
初期化されたローカル変数と
同じようにアクセスできます。

関数に渡す引数の個数にはとくに制限はありませんが、2、3個に留めておくのが現実的です。どんなデータも引数として関数に渡すことができます。定数(`Math.PI`)、変数(`temp`)、リテラル(72)、なんでもOKです。

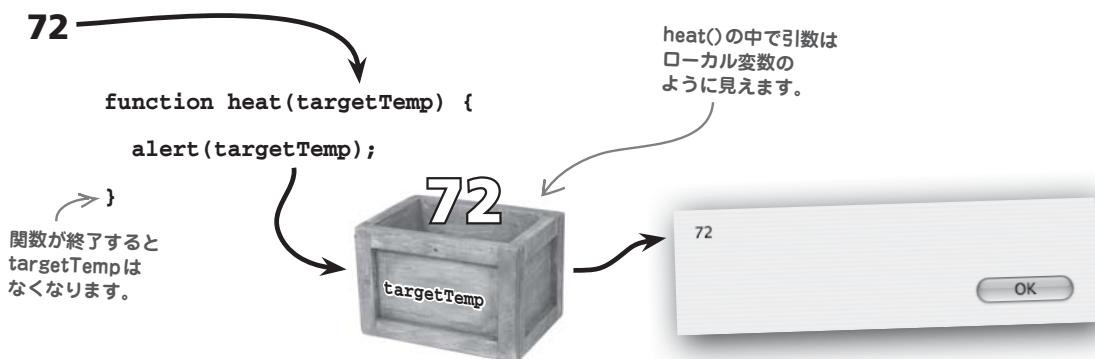
データとしての引数

関数に引数で渡されたデータは、関数の中では初期化されたローカル変数と同じような扱いになります。以下に示す例では、heat()に目標温度が引数として渡されています。

`heat(72);`

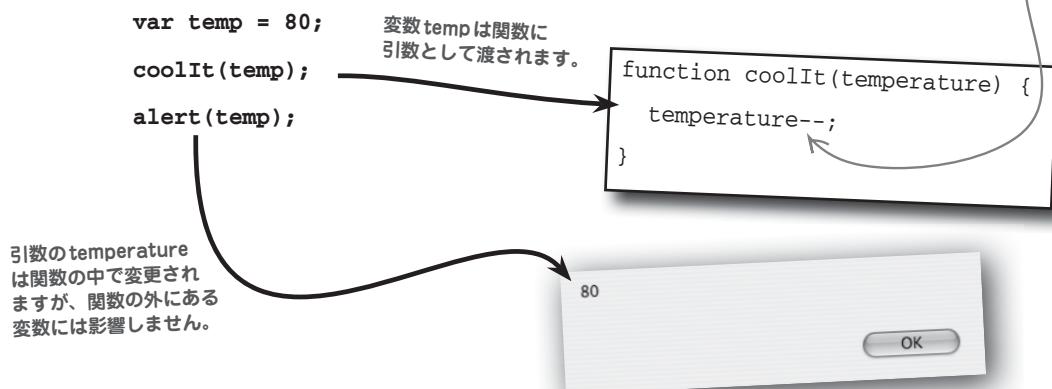
目標温度はリテラル数値
として関数に渡されます。

heat()の中では72で初期化されたローカル変数と同じように引数targetTempにアクセスできます。heat()のコードを書き換えて、引数の値をアラートで表示させてみましょう。



引数は関数内のローカル変数のように動作します。関数の内部で引数の値を変更しても、関数の外部では何の影響もありません。ただし引数としてオブジェクトが渡された場合、このルールは適用されません。オブジェクトについて詳しく説明します。

temperatureから1を引きます。



関数を使ってコードの重複をなくす

関数は、大きな問題を比較的容易に解決できる小さな問題に分割するためだけでなく、コードの重複をなくし一般化された作業に変える上でも役に立ちます。一般化された作業にすることで、あちこちにある類似したコードをなくすことができます。コードが全く同一ではないとしても、多くの場合それを関数に置き換えることで一般化できます。そうすれば重複していたコードのかわりに関数を呼び出すだけになります。

以下に示す3つのコード断片は、いずれも似たような作業を行っているので、ひとつの再利用可能な作業に一般化することができます。

割引の計算を重複させる
必要はありません。

// 昼間のチケットは 10% オフ

```
matineeTicket = adultTicket * (1 - 0.10);
```

// 老人のチケットは 15% オフ

```
seniorTicket = adultTicket * (1 - 0.15);
```

// 子供のチケットは 20% オフ

```
childTicket = adultTicket * (1 - 0.20);
```



3つのコード断片は、それぞれ異なる割引率で映画のチケットの価格を計算しています。これらの作業は、割引率をもとにチケットの価格を計算する作業に一般化できます。

```
function discountPrice(price, percentage) {  
    return (price * (1 - (percentage / 100)));  
}
```

関数はデータを
返すことも
できます。

一般化されたチケット割引関数があれば、3つのコード断片をより効率的なコードに書き換えることができます。

// 昼間のチケットは 10% オフ

```
matineeTicket = discountPrice(adultTicket, 10);
```

// 老人のチケットは 15% オフ

```
seniorTicket = discountPrice(adultTicket, 15);
```

// 子供のチケットは 20% オフ

```
childTicket = discountPrice(adultTicket, 20);
```



効率化のエキスパートになつてみよう

以下のfindSeats()はMandangoで座席検索に使われる関数ですが、まだ改善の余地があります。

これまで学んだ効率化の知識を応用して、

一般化された再利用可能な関数に書き換えられる類似したコードを丸で囲んでください。



```

function findSeats() {
    // 座席がすでに選択されていたら、座席を再初期化します
    if (selSeat >= 0) {
        selSeat = -1;
        initSeats();
    }

    // 全座席から空席を探します
    var i = 0, finished = false;
    while (i < seats.length && !finished) {
        for (var j = 0; j < seats[i].length; j++) {
            // 現在の座席と次の2席が空席か調べます
            if (seats[i][j] && seats[i][j + 1] && seats[i][j + 2]) {
                // 座席を選択された座席に設定して座席の画像を更新します
                selSeat = i * seats[i].length + j;
                document.getElementById("seat" + (i * seats[i].length + j)).src = "seat_select.png";
                document.getElementById("seat" + (i * seats[i].length + j)).alt = "Your seat";
                document.getElementById("seat" + (i * seats[i].length + j + 1)).src = "seat_select.png";
                document.getElementById("seat" + (i * seats[i].length + j + 1)).alt = "Your seat";
                document.getElementById("seat" + (i * seats[i].length + j + 2)).src = "seat_select.png";
                document.getElementById("seat" + (i * seats[i].length + j + 2)).alt = "Your seat";

                // 座席を予約するかユーザに訊ねます
                var accept = confirm((i + 1) + "列の座席 " + (j + 1) + "から" + (j + 3) +
                    "が空いています。予約しますか？");
                if (accept) {
                    // ユーザが座席を予約したので完了です
                    finished = true;
                    break;
                }
                else {
                    // ユーザがこの座席を断ったので、座席を選択されない状態にして検索を続けます
                    selSeat = -1;
                    document.getElementById("seat" + (i * seats[i].length + j)).src = "seat_avail.png";
                    document.getElementById("seat" + (i * seats[i].length + j)).alt = "Available seat";
                    document.getElementById("seat" + (i * seats[i].length + j + 1)).src = "seat_avail.png";
                    document.getElementById("seat" + (i * seats[i].length + j + 1)).alt = "Available seat";
                    document.getElementById("seat" + (i * seats[i].length + j + 2)).src = "seat_avail.png";
                    document.getElementById("seat" + (i * seats[i].length + j + 2)).alt = "Available seat";
                }
            }
        }

        // ループカウンタをインクリメント
        i++;
    }
}

```



効率化のエキスパートになつてみよう の答え

以下のfindSeats()は Mandango で座席検索に使われる関数ですが、まだ改善の余地があります。

これまで学んだ効率化の知識を応用して、一般化された再利用可能な関数に書き換えられる類似したコードを丸で囲んでください。



```

function findSeats() {
    // 座席がすでに選択されていたら、座席を再初期化します
    if (selSeat >= 0) {
        selSeat = -1;
        initSeats();
    }

    // 全座席から空席を探します
    var i = 0, finished = false;
    while (i < seats.length && !finished) {
        for (var j = 0; j < seats[i].length; j++) {
            // 現在の座席と次の2席が空席か調べます
            if (seats[i][j] && seats[i][j + 1] && seats[i][j + 2]) {
                // 座席を選択された座席に設定して座席の画像を更新します
                selSeat = i * seats[i].length + j;
                document.getElementById("seat" + (i * seats[i].length + j)).src = "seat_select.png";
                document.getElementById("seat" + (i * seats[i].length + j)).alt = "Your seat";
                document.getElementById("seat" + (i * seats[i].length + j + 1)).src = "seat_select.png";
                document.getElementById("seat" + (i * seats[i].length + j + 1)).alt = "Your seat";
                document.getElementById("seat" + (i * seats[i].length + j + 2)).src = "seat_select.png";
                document.getElementById("seat" + (i * seats[i].length + j + 2)).alt = "Your seat";

                // 座席を予約するかユーザに訊ねます
                var accept = confirm((i + 1) + "列の座席 " + (j + 1) + "から" + (j + 3) +
                    "が空いています。予約しますか？");
                if (accept) {
                    // ユーザが座席を予約したので完了です
                    finished = true;
                    break;
                }
                else {
                    // ユーザがこの座席を断ったので、座席を選択されない状態にして検索を続けます
                    selSeat = -1;
                    document.getElementById("seat" + (i * seats[i].length + j)).src = "seat_avail.png";
                    document.getElementById("seat" + (i * seats[i].length + j)).alt = "Available seat";
                    document.getElementById("seat" + (i * seats[i].length + j + 1)).src = "seat_avail.png";
                    document.getElementById("seat" + (i * seats[i].length + j + 1)).alt = "Available seat";
                    document.getElementById("seat" + (i * seats[i].length + j + 2)).src = "seat_avail.png";
                    document.getElementById("seat" + (i * seats[i].length + j + 2)).alt = "Available seat";
                }
            }
        }
        // ループカウンタをインクリメント
        i++;
    }
}

```

この6つのコードは同じタスクを実行しているので、ひとつの関数にまとめることができます。

部分配列の要素の数もlengthプロパティで参照できます。

コードが重複しています。
ここから同じ属性を抽出できます…

座席を設定する関数を作る

Mandangoの2人は関数が効果的であることを聞きつけ、関数を導入して座席予約のコードをより効率的にしようと盛り上がっています。しかし、`setSeat()`を書くには、必要となる引数を調べる必要があります。重複したコードのどこが異なっているか調べれば、必要な引数を特定できます。`findSeats()`のどこが重複しているかよく調べれば、必要な引数が明らかになります。

座席番号

予約する座席の番号。これは配列のインデックスではなく座席の番号です。座席を左から右に上から下に数えるときに使い、値は0から始めます。

重複しているコードから
`findSeats()`の属性を
抽出しました。

状態

座席の状態には空き、予約済み、予約があります。状態をもとに表示する座席画像を決めます。

説明

座席状態の説明です。座席画像の
`alt`属性に設定します。



自分で考えてみよう

Mandangoの`setSeat()`のコードを書いてください。



Mandango の setSeat() のコードを書いてください。

もとのコードで
使われていた
データを引数で
置き換えます。

```
function setSeat(seatNum, status, description) {
    document.getElementById("seat"+ seatNum).src = "seat_" + status + ".png";
    document.getElementById("seat" + seatNum).alt = description;
}
```

この 3 つの引数は
カンマで区切れます。

Mandango を関数ダイエット

似たような重複したコードを取り出して setSeat() にすれば、findSeats() のコードは見違えるほど簡潔になります。setSeat() の呼び出しが 6 カ所になりましたが、コードの再利用という点では大幅に改善されています。

setSeat() の呼び出し
で座席番号、状態状態、
説明が渡されます。

setSeat() は
6 回呼ばれます。

```
function findSeats() {
    ...
    // 全座席から空席を探します
    var i = 0, finished = false;
    while (i < seats.length && !finished) {
        for (var j = 0; j < seats[i].length; j++) {
            // 現在の座席と次の2席が空席か調べます
            if (seats[i][j] && seats[i][j + 1] && seats[i][j + 2]) {
                // 座席を選択された座席に設定して座席の画像を更新します
                selSeat = i * seats[i].length + j;
                setSeat(i * seats[i].length + j, "select", "Your seat");
                setSeat(i * seats[i].length + j + 1, "select", "Your seat");
                setSeat(i * seats[i].length + j + 2, "select", "Your seat");

                // 座席を予約するかユーザーに訊ねます
                var accept = confirm((i + 1) + " 列の座席 " + (j + 1) + " から " + (j + 3) +
                    " が空いています。予約しますか? ");
                if (accept) {
                    // ユーザが座席を予約したので完了です
                    finished = true;
                    break;
                } else {
                    // ユーザがこの座席を断つたので、座席を選択されない状態にして検索を続けます
                    selSeat = -1;
                    setSeat(i * seats[i].length + j, "avail", "Available seat");
                    setSeat(i * seats[i].length + j + 1, "avail", "Available seat");
                    setSeat(i * seats[i].length + j + 2, "avail", "Available seat");
                }
            }
            // ループカウンタをインクリメント
            i++;
        }
    }
}
```

setSeat()でMandangoを改良します

`setSeat()`の改善効果によって`findSeats()`だけではなく`initSeats()`も効率的になります。これらの関数にも座席予約のコードがあるからです。

```

function initSeats() {
    // 座席の画像を初期化します
    for (var i = 0; i < seats.length; i++) {
        for (var j = 0; j < seats[i].length; j++) {
            if (seats[i][j]) {
                // 座席を空席にします
                document.getElementById("seat" + (i * seats[i].length + j)).src = "seat_avail.png";
                document.getElementById("seat" + (i * seats[i].length + j)).alt = "Available seat";
            } else {
                // 座席をふさぎます
                document.getElementById("seat" + (i * seats[i].length + j)).src = "seat_unavail.png";
                document.getElementById("seat" + (i * seats[i].length + j)).alt = "Unavailable seat";
            }
        }
    }
}

function initSeats() {
    // 座席の画像を初期化します
    for (var i = 0; i < seats.length; i++) {
        for (var j = 0; j < seats[i].length; j++) {
            if (seats[i][j]) {
                // 座席を空席にします
                setSeat(i * seats[i].length + j, "avail", "Available seat");
            } else {
                // 座席をふさぎます
                setSeat(i * seats[i].length + j, "unavail", "Unavailable seat");
            }
        }
    }
}

```

たった2行の簡単な関数がMandangoのスクリプトでもう8カ所も使われています。関数にすることで、スクリプトが簡潔になるだけでなく、スクリプトの維持管理も楽になります。もし座席予約の方法を変更する必要がある場合、8カ所のコード断片をひとつひとつ修正するのではなく、`setSeat()`のコードだけを変更すればいいからです。気が確かなJavaScriptプログラマなら、あちこちにあるコードを変更したいとは思わないものです。維持管理が楽になるのは重要なことです。

重要ポイント

- 関数を使って大きな問題を小さな問題に変えると、問題を解決するのが容易になります。
- 関数はスクリプトのタスクを分割し再利用可能なコードのかたまりにして実行する仕掛けを提供します。
- 関数の中のコードは必要なときに何度も再利用できるので、重複したコードをなくす優れた手段にもなります。
- 引数を使うとタスクの入力として関数にデータを渡すことができます。

素朴な疑問に 答えます

Q: 関数に渡す引数の数に上限はありますか?

A: 論理的上限はありませんが物理的上限はあります。コンピュータのメモリの制約があるので、あまりに多くの引数を渡すときはメモリの制約が問題になることがあります。そのような場合は、そんなに引数が必要なのか考えなおしましょう。良い設計にするには、引数の数をある程度制限するのが現実的です。途方もなく引数が多いと理解しにくくなるので、そうならないように引数の数は多くても10個くらいに留めるといいでしょう。

Q: 関数は大きな問題を小さな問題に分割するもの、スクリプトをつくる作業を楽にするもの、コードの重複をなくすもの、と習いましたが、どれが正しいのでしょうか?

Q: 以前同じ質問をしましたが、関数はウェブページのヘッド部とボディ部のどちらに書くのでしょうか?

A: 関数はウェブページのヘッド部の<script>タグの中に書くべきです。あるいはJavaScriptファイルにして、そのファイルをページのヘッド部に取り込みます。

Q: 引数の値を変更する関数が必要なんですが、どうすればいいですか?

A: 関数の引数を直接変更することはできません。また関数の中で引数の値を変更しても、関数の外ではその値はもとのままで。引数で渡されたデータを変更したければ、関数の中で変更した値を返す必要があります。関数から値を返す方法については、この後説明します。



このサーモスタッフ、
調子悪いね。凍えそうだよ。

快適なんだ
けどなあ。



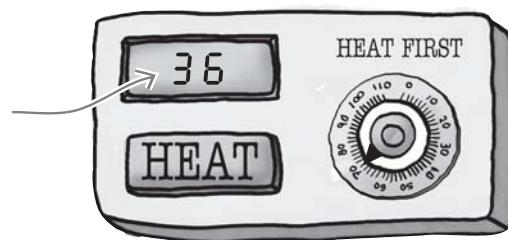
7月の冬：関数でフィードバック

関数のおかげでMandangoは何歩も前進しましたが、気候変動対策の方は芳しくありません。JavaScriptのサーモスタッフには、まだうまく動いていないようです。寒さに震えるユーザが暖房の止まらない古い「HEAT」ボタンを要望しています。

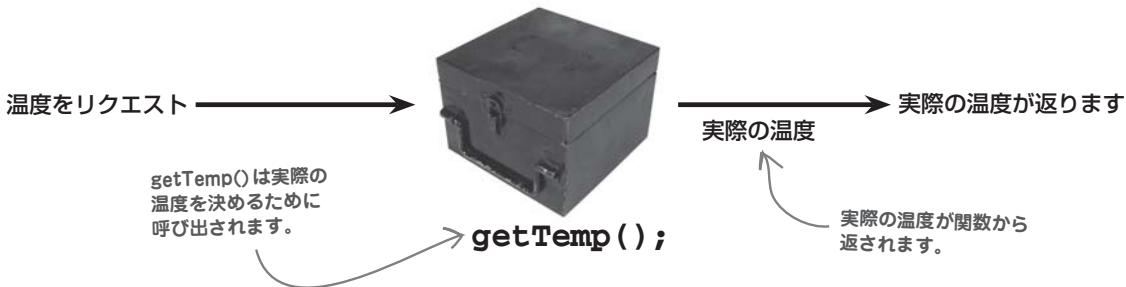
フィードバックの意義

現行のサーモmostatは引数のおかげで温度を設定できますが、現在の温度を表示しません。現在の温度は、目標温度を決める際の基準になるので重要です。同じ部屋にあるのに、別のサーモmostatでは別の温度が表示されることがあります。つまりフィードバックが必要ということです。意味のある目標温度を設定するには、現在の温度を知る必要があります。

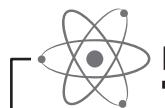
現在の温度がディスプレイに表示されるのでユーザはより正確に暖房を調整できます。



サーモmostatに定期的に現在の温度が表示されれば最適温度を決めるときに役立ちます。



ここで必要になるのは、関数を呼び出したコードに情報を返す機能です。



頭の体操

関数がデータを返すようにするにはどうすればいいと思いますか？

関数からデータを返す

関数から情報を返すにはキーワード `return` を使い、これに返すべきデータを続けます。このデータが関数を呼び出したコードに返されます。

`return` + 値 + ;

キーワード `return` で関数が返す
値を指定します。

どんな値も返す
ことができます。

`return` 文は関数の中のどこででも使えます。関数は `return` になったところで即終了するので注意してください。`return` 文はデータを返すだけでなく、関数を終了させます。たとえば、`getTemp()` はセンサーから取得した実際の温度を返して終了します。

関数からデータ断片を
返値として返す
ことができます。



```
function getTemp() {  
    // 実際の温度を読み取り、変換します  
    var rawTemp = readSensor(); ←  
    var actualTemp = convertTemp(rawTemp);  
    return actualTemp;  
}
```

センサーの温度が
奇妙な形式なので変換する
必要があります。

`return` 文を使って実際の
温度を関数から返します。

注意してサーモスタットのコードを見直すと、`getTemp()` はすでに使われています。

```
function heat(targetTemp) {  
    while ( getTemp() < targetTemp ) { ←  
        // なんらかの方法で暖房します  
        ...  
    }  
}
```

getTemp() は heat() の
while ループの条件テストで
使う値を返します。

`getTemp()` の関数呼び出しが `getTemp()` の返値で 置き換え られて、
while ループのテスト条件の一部になります。

関数の呼出しは
関数の返値で
置き換えられます。

返す値を複数もたせる

`return`文は関数を即終了させますが、関数のフロー制御と`return`文を組み合わせて使うことができます。それだけでなく、返値を使って関数の実行が成功したことを知らせることもよくあります。`heat()`ではこの両方が使われています。

```
function heat(targetTemp) {
  if (getTemp() >= targetTemp) ← 変数actualTempを憶えてますか？
    return false; ← getTemp()の返値がそれです。
  while (getTemp() < targetTemp)
  {
    // なんらかの方法で暖房する このコードで実際に暖房します。
    ...
  }
  return true; ← 暖房を終わらせます。trueを
}                                返して成功したことを探せます。
```

`heat()`をみると、論理値の返値を使って関数のフロー制御を行っています。成功すれば`true`、失敗すれば`false`を返します。純粋なフロー制御であれば、返値をとらない`return`文を使って関数から脱出することができます。`heat()`の別バージョンでは、成功か失敗を示す返値を使っていません。

```
function heat(targetTemp) {
  if (getTemp() >= targetTemp) ← 暖房する必要がないのでreturn文を
    return; ← 使って関数を終了させます。
  while (getTemp() < targetTemp) {
    // なんらかの方法で暖房する
    ...
  }
}
```

ここで関数が終わるので
return文を使う必要はありません。

**return文はそこで
関数を終了させるときに
使われます。**



Return の真実

今週のインタビュー：関数脱出の極意

Head First：とても捕まえにくい人だそうですね。どんなところからでも脱出できるという噂ですが。

Return：そうですね。私を関数の中に入れると、すぐにそこから脱出します。しかもデータを手にしてね。

Head First：関数から出たら、どこへ行くんですか？

Return：うーん、そもそも関数は他のコードから呼ばれます。なので関数から出るということは、関数を呼んだコードに戻ることなんです。データを返す場合、そのデータは関数を呼んだコードに返されるのです。

Head First：具体的にどんな動作になるんですか？

Return：関数呼び出しを式と考えてみましょう。データを返さない関数は、結果がない式と同じです。データを返す関数、まあ関数の多くはデータを返すのですが、その返されたデータが式の結果になります。

Head First：関数が単なる式だとすると、関数が返す値を変数に代入できるんですか？

Return：できないとも、できるとも言えますね。関数そのものは式ではありません。関数の呼び出しが式になります。関数はその結果を変数に代入するために呼び出されることが多いのです。ここで式の出番です。関数の呼び出しが評価されると、関数が返す値が式として扱われるのです。

Head First：わかりました。でも関数が何も返さないとき、それを式として扱うとどうなりますか？

Return：データを返さない関数を使うと、式は空ですね。

Head First：それって問題ではありませんか？

Return：実はそれほど問題ではないんですね。問題になるのは、関数が返すデータを使って何かを行うと

きだけです。何も返さない関数を使うときは、そもそも扱うべき値がないわけですから、心配しなくていいんです。

Head First：そういうことですか。脱出の話に戻りますが、関数の本来の実行が途中で終わるのは、よくないんじゃないでしょうか？

Return：そんなことはありません。というのも、どんなときでも関数のコードを最初から最後まですべて実行する必要がある、ということではないですから。関数に始まりと終わりがあると考えるのは、むしろ危険です。熟練した開発者の手にかかると、関数の本来の終わりがコードの途中になることもありますから。

Head First：よくわかりません。関数のコードの一部がけして実行されなくとも、それが正常だって言うんですか？

Return：けしてなんて言ってませんよ。ひとつの関数には普通は複数の実行経路があるんです。で、こうした複数の経路をつくるのに私が一役買っているんですね。関数の実行がそれ以上続けられない状態になつたら、私が出口を提供しています。もちろん私に会うことなく最後の行までたどり着く関数もありますが、それ以外の場合、関数の最後で私がデータを返しています。

Head First：そうだったんですか、わかりました。あなたはデータを返す場合と関数の実行フローを制御する場合の両方の選択肢を提供しているんですね。

Return：やっと理解が追いつきましたね。

Head First：ほんと、駆け足で話を追ってましたから。お時間ありがとうございました。

Return：ご心配なく。話はここまで、ということで。



JavaScriptは気候変動問題の渦中にまきこまれたようです。WARM (We're Against Rapid Warming)の人たちが温暖化の危機を訴えるメッセージを伝えるスクリプトを作成しました。一方ARCTIC (Annoyed but Responsible Citizens Tired of Increasing Cold)の人たちはWARMのメッセージを抑えようとしてスクリプトコードを妨害しています。そこで以下のスクリプトに含まれる誤りを訂正して、WARMの意図した通りのメッセージが表示されるようにしてください。

```

function showClimateMsg() {
    return;
    alert(constructMessage());
}

function constructClimateMsg() {
    var msg = "";
    msg += "Global ";// "Local ";

    if (getTemp() > 80)
        msg += "温暖化";
    else
        msg += "寒冷化";

    if (true)
        msg += "ではありません";
    else
        msg += "です";

    if (getTemp() <= 70)
        return msg + "はウソ!";
    else
        return msg + "は本当!";

    return "信じられない。";
}

function getTemp() {
    // 実際の温度を読み取ります
    var actualTemp = readSensor();
    return 64;
}

```



WARM 支援者



ARCTIC 工作員



エクササイズ 答え

このreturnがあるとアラートが表示されません。

実際の温度によってメッセージが変わるので、これは問題ありません。

if/else文で処理が終わり、このコードは実行されないので、意味がありません。

It seems JavaScriptは気候変動問題の渦中にまきこまれたようです。WARM (We're Against Rapid Warming)の人たちが温暖化の危機を訴えるメッセージを伝えるスクリプトを作成しました。一方ARCTIC (Annoyed but Responsible Citizens Tired of Increasing Cold)の人たちはWARMのメッセージを抑えようとしてスクリプトコードを妨害しています。そこで以下のスクリプトに含まれる誤りを訂正して、WARMの意図した通りのメッセージが表示されるようにしてください。

```
function showClimateMsg() {
    return;
    alert(constructMessage());
}

function constructClimateMsg() {
    var msg = "";
    msg += "Global" // "Local ";
    if (getTemp() > 80)
        msg += " 温暖化 ";
    else
        msg += " 寒冷化 ";
    if (true)
        msg += " ではありません ";
    else
        msg += " です ";

    if (getTemp() <= 70)
        return msg + " はウソ !";
    else
        return msg + " は本当 !";
    return " 信じられない。";
}

function getTemp() {
    // 実際の温度を読み取ります
    var actualTemp = readSensor();
    return 64, actualTemp;
}
```

表示したい語がコメントになっています。

このif文は常にtrueになるので意味がありません。

センサーから取得した実際の温度を返します。

ありがとう
JavaScript!

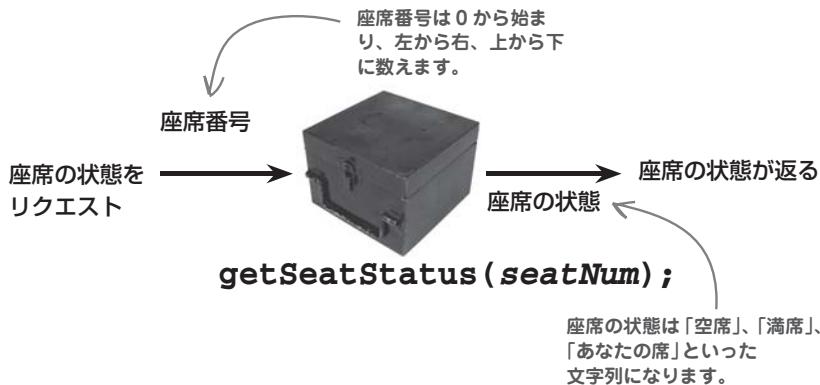


この格好じゃ暑すぎるよ、助けて！

温暖化は切実な問題です！

座席の状態を取得する

Mandangoに戻ってみたら、セスとジェイソンはサーモスタッフが改善された話を聞いて落ち込んでいます。自分たちのコードをもっと改善しようとしています。座席の色の違いがわかりにくいので、座席が開いているかどうかクリックして調べられるようにしてほしいというユーザがいました。Mandangoには新機能が必要なようです。



getSeatStatus()のマグネット

MandangoのgetSeatStatus()から座席の状態を調べるために重要なコードが消えています。この関数はまず指定された座席が選択された空席3席に含まれるかチェックします。含まれない場合、座席配列から座席を見て、その席が空席かどうかを調べます。下のマグネットを使ってコードを完成させてください。

```
function getSeatStatus(seatNum) {
  if ( ..... != -1 &&
      ( ..... == ..... || ..... == ( ..... + 1) || ..... == ( ..... + 2)))
    return "あなたの席";
  else if ( ..... [Math.floor( ..... / ..... [0].length)][ ..... % ..... [0].length])
    return "空席";
  else
    return "満席";
}
```

seats

selSeat

seatNum



getSeatStatus() マグネットの解答

MandangoのgetSeatStatus()から座席の状態を調べるために重要なコードが消えています。この関数はまず指定された座席が選択された空席3席の中に含まれるかチェックします。含まれない場合、座席配列から座席を見て、その席が空席かどうかを調べます。下のマグネットを使ってコードを完成させてください。

座席が選択されていないと
グローバル変数selSeatは
-1になるので、まずこれを
チェックします。

3席分を扱うので、この座席と次の2席を
チェックする必要があります。

```
function getSeatStatus(seatNum) {
    if ( selSeat != -1 &&
        ( seatNum == selSeat || seatNum == ( selSeat + 1 ) || seatNum == ( selSeat + 2 ) ) )
        return "あなたの席";
    else if ( seats[ Math.floor( seatNum / seats[ 0 ].length ) ][ seatNum % seats[ 0 ].length ] )
        return "空席";
    else
        return "満席";
}
```

座席番号を行の座席数で割った商で行の
インデクスを求めて、その行の座席の
配列を調べます。

座席番号を行の座席数で割った
余りで列のインデクスを求めて、
その座席を調べます。

ここを9にすることもできます
が、もし座席配列の大きさを
変えた場合、動作しなくなってしまいます。

座席の状態を表示する

座席の状態は簡単に取得できますが、座席の状態をユーザが調べられるよう
にするには、ユーザが座席をクリックしたとき座席の状態を表示する手段が
必要です。showSeatStatus()はこの問題を解決します。泥臭い仕事は
getSeatStatus()に任せています。

getSeatStatus()に
座席番号を渡して
座席の状態を取得します。

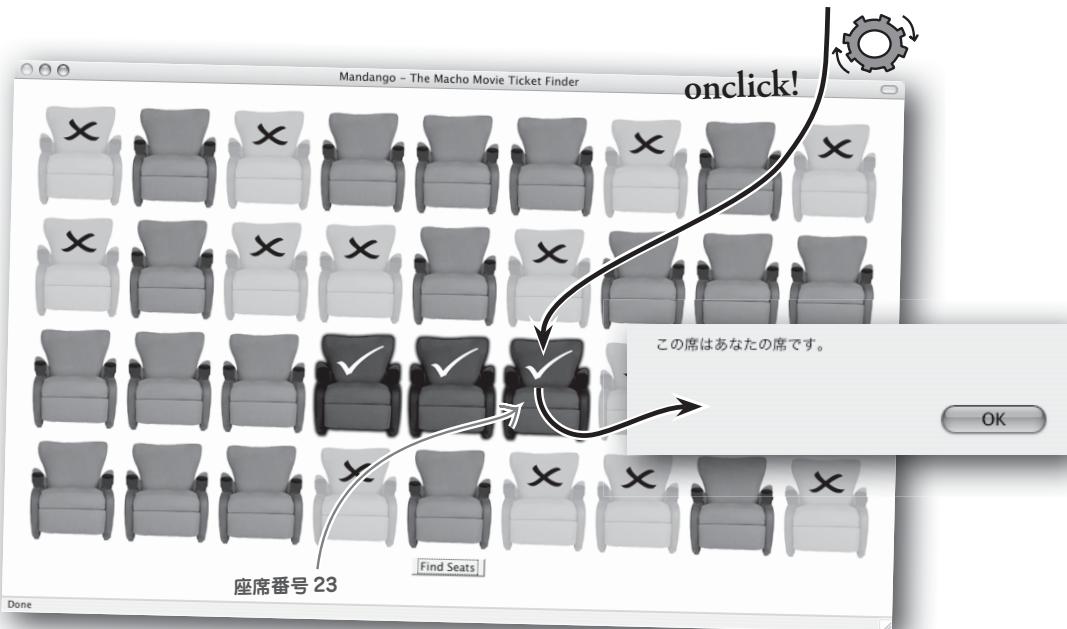
```
function showSeatStatus(seatNum) {
    alert("この席は " + getSeatStatus(seatNum) + " です。");
}
```

文字列を連結して
状態に関する
メッセージを作ります。

画像と関数を結びつける

Mandangoのページにある座席の画像にこの関数を書くと、ユーザが画像をクリックしたときその状態が表示されます。画像のonclickイベントをshowSeatStatus()に結びつける必要があります。

```
<img id="seat23" src="" alt="" onclick="showSeatStatus(23);"/>
```



クリックするだけで、アラートボックスに座席の状態が表示されます。
これなら座席の画像の色がわからないユーザでも、簡単に座席の状態を調べられます。

重要ポイント

- return文を使うと関数を呼び出したコードに関数からデータを返すことができます。
- 関数から返されるデータ断片は関数を呼び出したコードに置き換わります。
- 関数が返せるのはひとつのデータ断片だけです。
- return文は関数を途中で終わらせるためにデータなしで使うこともできます。

繰り返しの多いコードは良くない

Mandangoのスクリプトはうまく動いていますが、二人は長期にわたる維持管理について心配はじめました。ジェイソンはいろいろ調査して、最新のウェブアプリケーションはHTML、JavaScript、CSSから恩恵を受けていることを知りました



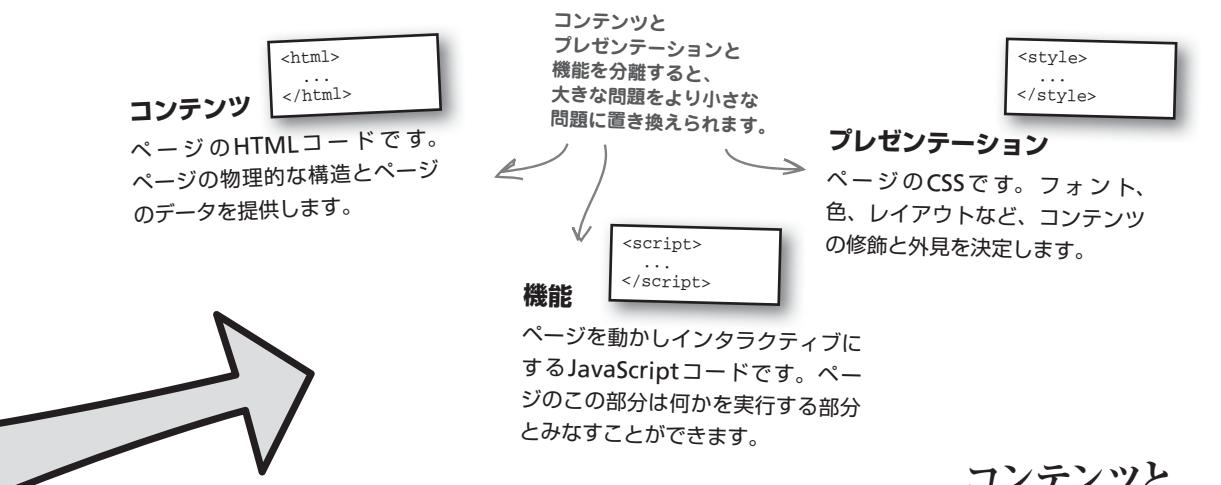
```
<html>
  <head>
    <title>Mandango - The Macho Movie Ticket Finder</title>
    <script type="text/javascript">
      ...
      function initSeats() {
        ...
      }
      function getSeatStatus(seatNum) {
        ...
      }
      function showSeatStatus(seatNum) {
        alert("この席は" + getSeatStatus(seatNum) + "です。");
      }
      function setSeat(seatNum, status, description) {
        document.getElementById("seat" + seatNum).src = "seat_" + status + ".png";
        document.getElementById("seat" + seatNum).alt = description;
      }
      function findSeats() {
        ...
      }
    </script>
  </head>
  <body onload="initSeats();">
    <div style="margin-top:25px; text-align:center">
      <img id="seat0" src="" alt="" onclick="showSeatStatus(0);"/>
      <img id="seat1" src="" alt="" onclick="showSeatStatus(1);"/>
      <img id="seat2" src="" alt="" onclick="showSeatStatus(2);"/>
      <img id="seat3" src="" alt="" onclick="showSeatStatus(3);"/>
      <img id="seat4" src="" alt="" onclick="showSeatStatus(4);"/>
      <img id="seat5" src="" alt="" onclick="showSeatStatus(5);"/>
      <img id="seat6" src="" alt="" onclick="showSeatStatus(6);"/>
      <img id="seat7" src="" alt="" onclick="showSeatStatus(7);"/>
      <img id="seat8" src="" alt="" onclick="showSeatStatus(8);"/><br />
      <img id="seat9" src="" alt="" onclick="showSeatStatus(9);"/>
      <img id="seat10" src="" alt="" onclick="showSeatStatus(10);"/>
      <img id="seat11" src="" alt="" onclick="showSeatStatus(11);"/>
      <img id="seat12" src="" alt="" onclick="showSeatStatus(12);"/>
      <img id="seat13" src="" alt="" onclick="showSeatStatus(13);"/>
      <img id="seat14" src="" alt="" onclick="showSeatStatus(14);"/>
      <img id="seat15" src="" alt="" onclick="showSeatStatus(15);"/>
      <img id="seat16" src="" alt="" onclick="showSeatStatus(16);"/>
      <img id="seat17" src="" alt="" onclick="showSeatStatus(17);"/><br />
      <img id="seat18" src="" alt="" onclick="showSeatStatus(18);"/>
      <img id="seat19" src="" alt="" onclick="showSeatStatus(19);"/>
      <img id="seat20" src="" alt="" onclick="showSeatStatus(20);"/>
      <img id="seat21" src="" alt="" onclick="showSeatStatus(21);"/>
      <img id="seat22" src="" alt="" onclick="showSeatStatus(22);"/>
      <img id="seat23" src="" alt="" onclick="showSeatStatus(23);"/>
      <img id="seat24" src="" alt="" onclick="showSeatStatus(24);"/>
      <img id="seat25" src="" alt="" onclick="showSeatStatus(25);"/>
      <img id="seat26" src="" alt="" onclick="showSeatStatus(26);"/><br />
      <img id="seat27" src="" alt="" onclick="showSeatStatus(27);"/>
      <img id="seat28" src="" alt="" onclick="showSeatStatus(28);"/>
      <img id="seat29" src="" alt="" onclick="showSeatStatus(29);"/>
      <img id="seat30" src="" alt="" onclick="showSeatStatus(30);"/>
      <img id="seat31" src="" alt="" onclick="showSeatStatus(31);"/>
      <img id="seat32" src="" alt="" onclick="showSeatStatus(32);"/>
      <img id="seat33" src="" alt="" onclick="showSeatStatus(33);"/>
      <img id="seat34" src="" alt="" onclick="showSeatStatus(34);"/>
      <img id="seat35" src="" alt="" onclick="showSeatStatus(35);"/><br />
    </div>
    <input type="button" id="findseats" value="Find Seats" onclick="findSeats();"/>
  </body>
</html>
```

JavaScriptとHTMLが混在した
コードは、HTMLのイベントハンドラ
属性で分離することができます。

Mandangoの
ウェブページでは
JavaScriptと
HTMLのコードが
混在しています。

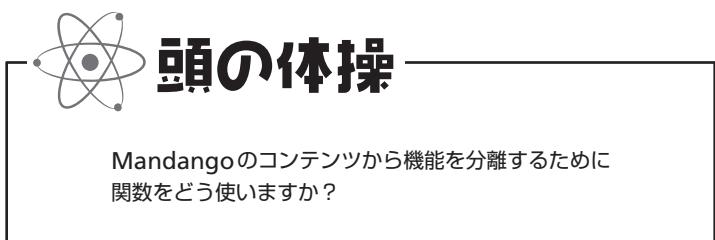
コンテンツから機能を分離する

コードを混在させるのは何が問題なのでしょうか？ 動いているから問題ないのでは？ JavaScriptを駆使したページをページとしてではなく**アプリケーション**として眺めると、問題がいくつもあります。良いアプリケーションがそうであるように、JavaScriptアプリケーションも長期にわたって成功をおさめるには、綿密な計画と設計が必要です。重要なのは、コンテンツとプレゼンテーションと機能を分離することによって、バグの少ない簡単に維持管理できるアプリケーションになることです。この観点からMandangoをみると、これら3つがすべて曖昧に混ざっています。



コードの分離について考えてみましょう。セスとジェイソンは自分たちが作ったコードのかわりになる素晴らしいスクリプトを見つけました。この座席管理スクリプトを使うにはMandangoを改修する必要がありますが、JavaScriptコードがHTMLコードに密着しているので、ページの構造がめちゃめちゃになる危険があります。HTMLコードを分離すれば、JavaScriptとHTMLのつなぎはJavaScriptの側だけで発生するので、ページの構造がこわれる心配はありません。

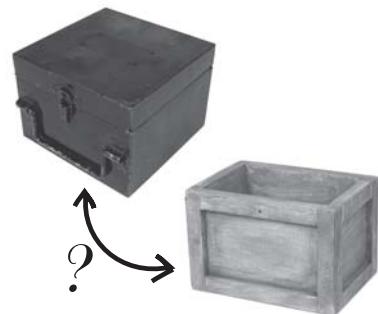
コンテンツと
機能を分離すると
ウェブアプリケーション
の作成と維持管理が
楽になります。



関数なんてただのデータ

コードを効率的に分離するには、関数とイベントのつながりを理解する必要があります。ここまでではHTMLの属性を使ってつながりを実現してきましたが、JavaScriptとHTMLを混在させるよりも優れたやり方があります。このやり方では関数の別な側面を使ってイベントハンドラにつなぎます。

きっと驚かれるでしょうが、関数は実はただの変数なのです。関数の名前は変数名ですが、値は関数の本体になります。以下に示すのは、これまで慣れ親しんできた関数の書き方です。



```
function showSeatStatus(seatNum) {
    alert("この席は " + getSeatStatus(seatNum) + " です。");
}
```

いつものように関数を作成するところになります。

```
var showSeatStatus = function(seatNum) {
    alert("この席は " + getSeatStatus(seatNum) + " です。");
};
```

変数名が関数の名前になります。

関数の本体が変数の値になります。
これは関数リテラルといいます。

このコードは変数と同じ構文を使って関数を作る方法を示しています。使う道具も同じで、一意の識別子が関数名、値が関数の本体になります。関数の本体には関数の名前がありません。これは関数リテラルと呼ばれています。

たいへん興味深いことに、このように関数をとらえると関数を変数のように操作できることがわかります。たとえば以下のコードはどうなるでしょう？

```
var myShowSeatStatus = showSeatStatus;
```

↑
showSeatStatus() を変数
myShowSeatStatus に
代入します。

関数の呼び出しと参照

関数の名前を別の変数に代入すると、変数から関数の本体へのアクセスが可能になります。つまり以下のように書くことができます。

```
alert(myShowSeatStatus(23));
```



同じ関数を変数
myShowSeatStatus から
呼び出します。

`myShowSeatStatus()` を呼び出した結果は `showSeatStatus()` の呼び出し結果と同じです。どちらの関数も同じコードを参照しているからです。このため関数名は **関数参照** と呼ばれることがあります。

```
showSeatStatus → function() {  
    ...  
    myShowSeatStatus  
};
```



関数の **参照** と関数の **呼び出し** の違いは、関数名の後に括弧 () を続けるかどうかの違いになります。関数呼び出しの場合、関数名の後に括弧を続けますが、関数参照には括弧を付けません。

```
myShowSeatStatus()  
を実行するのは  
showSeatStatus()を  
実行するのと同じです。
```



var myShowSeatStatus = showSeatStatus;
→ myShowSeatStatus(23);



関数参照を
myShowSeatStatus に
代入します。



エクササイズ

以下のコードを分析して、アラートボックスに表示される値を書いてください。

```
function doThis(num) {  
    num++;  
    return num;  
}  
  
function doThat(num) {  
    num--;  
    return num;  
}  
  
var x = doThis(11);  
var y = doThat;  
var z = doThat(x);  
x = y(z);  
y = x;  
alert(doThat(z - y));
```





エクササイズの 答え

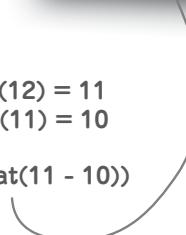
以下のコードを分析して、アラートボックスに表示される値を書いてください。

```
function doThis(num) {
    num++;
    return num;
}

function doThat(num) {
    num--;
    return num;
}

var x = doThis(11);
var y = doThat;
var z = doThat(x);
x = y(z);
y = x;
alert(doThat(z - y));
```

$x = 12$
 $y = \text{doThat}$
 $\rightarrow z = \text{doThat}(12) = 11$
 $x = \text{doThat}(11) = 10$
 $y = 10$
 $\text{alert}(\text{doThat}(11 - 10))$



素朴な疑問に 答えます

Q: コンテンツを分離することは
そんなに重要なんでしょうか?

A: 場合によりけりです。簡単な
アプリケーションだったら、
HTMLとCSSとJavaScriptをひとつ
にまとめておくのも悪くありません。ア
プリケーションが複雑になればなるほど
コードも増えていくので、そういう
とコードを分離するメリットが効いてき
ます。大きなアプリケーションになるほど、全体像を理解するのが困難にな
り、コードを変更するときトラブルにな
りやすいのです。種類の違うコードが
混在していればなおさらです。コードを
きれいに分離しておけば、ページの
構造や外見を壊さずにその機能だけを
安全に変更できます。またひとつのブ
ロジェクトで領域が異なる専門家どう
しが分業することもできます。たとえば

JavaScriptのコードを理解していない
ウェブデザイナはページの構造と外見だけに専念でき、JavaScriptのコードにエ
ラーをもちこむ危険もありません。

Q: 関数がデータにすぎないとし
たら、関数と通常の変数はどうやって区別できるのでしょうか?

A: 関数と通常の変数の違いは
データを使って何をするの違
いです。関数に関連づけられたデータ
(コード)は実行することができます。
関数を実行したいことを示すために関
数名の後に括弧を付けるのです。関数
がデータを必要とするときは括弧の中
に引数を指定するわけです。

Q: 関数参照の要点は何でしょ
う?

A: 通常の変数はメモリ領域にそ
のデータが値として格納されま
す。一方、関数はそのコードへの参照
が格納されます。つまり変数としての関
数の値はコードそのものではなく、コー
ドが格納されているメモリの所在地を
指す参照なのです。たとえば郵便の宛
先はあなたの家のへの参照であって、あ
なたの家そのものではありませんよね。
これと同様、関数は実際の値ではなく
参照として使われます。この方が関数
のコードのコピーを毎回格納するよりも
はるかに効率的だからです。関数をイ
ベントハンドラに代入するとき、実際に
は関数コードへの参照を代入している
だけであって、コードそのものを代入し
ているわけではありません。



そうね、関数参照は
すごく良さそうなんだけど、コンテンツと
機能を分離するためにどう扱えば
いいのかな？

コールバックのための関数

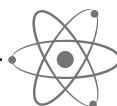
関数参照は、ある特別な関数呼び出し方法と密接な関係にあります。この方法を使うと、コンテンツと機能の分離をさらに押し進めることができます。Mandango のコードでの関数呼び出しが以下のように書かれています。

```
setSeat(i * seats[i].length + j, "select", "Your seat");
```



```
function setSeat(seatNum, status, description) {  
    ...  
}
```

スクリプトから関数を呼び出す方法はこれ以外にもあります。コールバック関数と呼ばれる別の種類の関数があり、これを使えば関数の呼び出しを直接コードに書かなくても関数を呼び出すことができます。



頭の体操

Mandangoでコールバック関数をどう使うとメリットがあると思いますか？

特別座談会



今夜の対談：
ノーマル関数とコールバック関数が真正面からぶつかります。

ノーマル関数：

聞くところによると、あなたはローカルな呼び出しを受け付けないそうですね。それはあなたのポリシーなんですか？

まるでブラウザみたいですね。スクリプトコードと通常のやり取りをする関数をちょっと見下しているように思うんですが。

おお、それは一大事ですね。いや、待ってください。そもそもスクリプトの外で起きていることなんて、誰が気にすると言うんですか？

それもそうですね。私だって、いつページが読み込まれたのか、いつユーザがクリックしたりタイプしたのか、そりや知りたいですよ。あなたがいなければ、私はそれを知ることができない、ということですか？

なるほど、良かった。私たちはそれほど違っているわけではないんですね。

あなたからではなく、私から呼び出しますから。

コールバック関数：

いいえ、ポリシーではなく、使命が違うだけです。遠く離れた場所からの呼び出しだけを受け付けるのです。

いいですか、優劣を競っているのではないんです。2人ともスクリプトコードなのは同じなんですから。ただ私はスクリプトコードにアクセスする手段を外部のひとたちに提供しているだけなんです。もし私がいなければ、スクリプトの外で何かが起きたときも、それを知りようがないですよ。

みんな気にしてますよ。そもそもスクリプトの核心は、ユーザによりよい体験をしてもらうことにあるわけです。もしスクリプトがその外で起きた事を検知する手段を持っていなかったら、ユーザの体験の質を向上することなんて、恐ろしく困難になるでしょうね。

ええ、そのとおり。ブラウザが私を呼び出すと、私は多くの場合、あなたを呼び出します。というのも、外で起きた事に応答するには、いくつかの関数が必要になるからです。

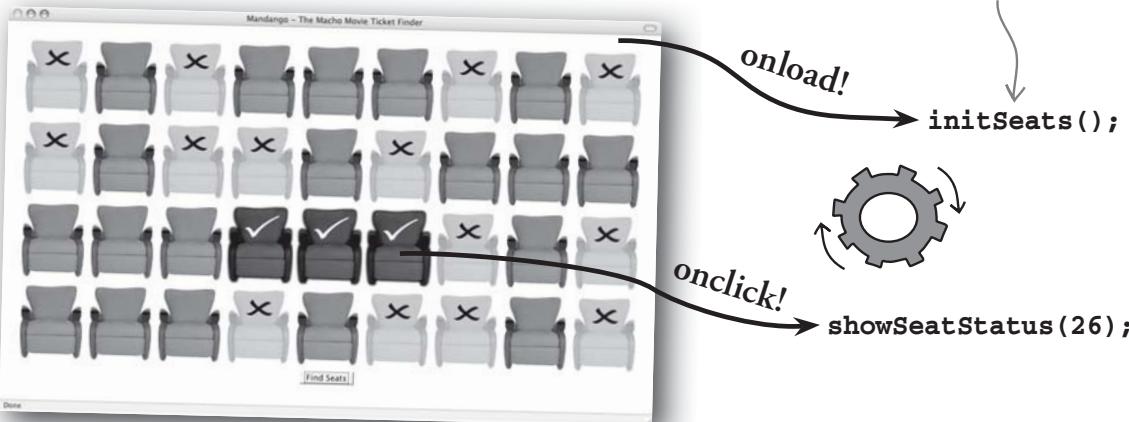
そうですね。また近々お会いしましょう。

それがかなうといいですね。

イベント、コールバック、HTML属性

実はこれまでコールバック関数を使っているのですが、コードからではなくブラウザから呼び出されていました。コールバック関数の一般的な用途はイベント処理です。すでに Mandango はコールバック関数に深く依存しています。HTML と JavaScript のコードが混在してしまう問題の原因はイベント処理の関数にあります。

ページが読み込まれるとブラウザは initSeats() を呼び出します。



これらのコールバック関数は Mandango のページの HTML コードの中でイベントにつなげられています。

```
<body onload="initSeats();>
```

HTML の `onload` 属性を使うと `initSeats()` を `onload` イベントにつなぐことができます。

```
<img id="seat26" src="" alt="" onclick="showSeatStatus(26);"/>
```

HTML の `onclick` 属性を使うと `showSeatStatus()` を `onclick` イベントにつなぐことができます。

この手法のようにイベント処理の関数を HTML の属性を経由してつなぐ場合、動作に問題はないのですが、JavaScript と HTML のコードが混在してしまう欠点があります。関数参照を使うと HTML と JavaScript を分離することができます。



関数参照を使ってイベントとつなぐ

HTML 属性を使ってコールバック関数をイベントハンドラとしてイベントにつなぐかわりに、JavaScript コードで関数参照を直接代入することができます。HTML コードをまったくいじる必要がありません。関数参照を使ってコールバック関数を設定するだけです。これはすべて JavaScript の中に実現できます。

`window.onload = initSeats;`

onload イベントは
window オブジェクトの
プロパティです。

initSeats() への参照を
onload イベントプロパティに
代入します。

関数を実行するのではなく、
関数を参照したいだけなので、
関数名の後に括弧は
つけません。

イベントハンドラの設定を JavaScript コードだけで実現するには、オブジェクトのイベントプロパティに関数参照を代入する必要があります。このコードの場合、`onload` イベントが発生したとき `initSeats()` が呼び出されます。この呼び出しはイベントが発生したとき自動的に実行されます。

`onload!` → `window.onload();` → `initSeats();`

onload イベントが window.onload
プロパティによってイベントハンドラを
起動します…

…このプロパティは関数参照に設定
されているので、イベントが処理される
`initSeats()` が呼び出されます。



関数参照を使ってイベントハンドラ関数をイベントに代入する利点は、JavaScript と HTML のコードを明確に分離できる点にあります。HTML のイベント属性に JavaScript のコードを埋め込む必要はありません。

`<body onload="initSeats();">` → `</body>`

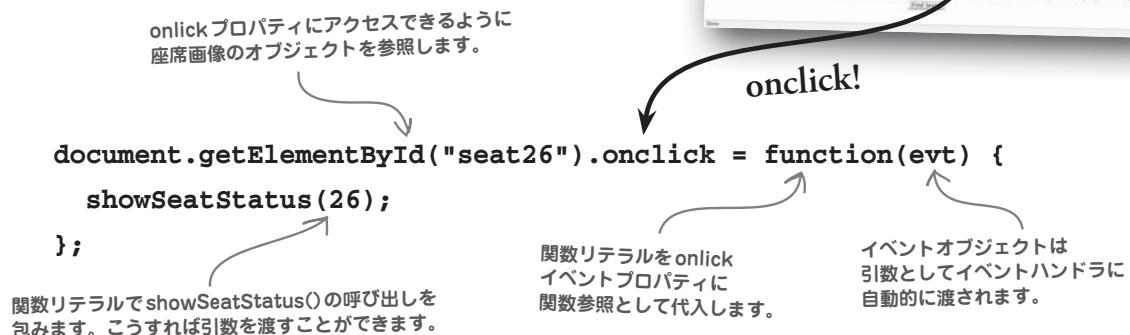
この関数ハンドラは JavaScript コードだけでつながれているので、`<body>` タグに `onload` 属性は必要ありません。イベント代入のコードはできるだけ早く実行される必要があるので、通常はページのヘッド部にこのコードを書きます。

まだ問題があります。イベントハンドラに引数を渡す必要がある場合、どうすればいいでしょうか？ これは Mandango の場合、`onload` ではなく `onclick` で問題になります。`onclick` イベントではクリックされた座席の番号を渡す必要があるからです。関数参照には引数を渡す手段がありません。そのため別の方法が必要になります。

関数参照は
イベントハンドラ
関数とイベントを
つなぐ便利な
手段を提供します。

関数リテラルで解決

Mandangoの座席画像にあるonclickイベントは、クリックされた座席がわかるようにshowSeatStatus()を引数(座席番号)付きで呼んでいます。関数参照を代入するだけでは引数が渡せないので問題になりましたが、別の方法があります。関数リテラルを関数参照として使い、この関数リテラルの中でshowSeatStatus()を呼び出せばいいのです。



関数リテラルはshowSeatStatus()の呼び出しを包むのに使われていますが、この関数に座席番号を渡すという重要な役割を果たしています。関数リテラルはイベントを処理する名前のない関数と考えることができます。このため関数リテラルは匿名関数とも呼ばれます。

このコードはJavaScriptを使ってイベントオブジェクトをイベントハンドラに渡す方法を示しています。ここでは引数evtでイベントオブジェクトを渡しています。イベントオブジェクトにはイベントごとに異なる特定の情報が含まれています。この例ではイベントの詳細情報を知る必要はないので、引数evtを使っていません。

関数リテラルを使うと匿名のイベントハンドラ関数を作成できます。



エクササイズ

initSeats()と onload イベントハンドラをつないでください。今回は関数参照ではなく関数リテラルを使ってください。



エクササイズの 答え

initSeats() と onload イベントハンドラをつないでください。今回は関数参照ではなく関数リテラルを使ってください。

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

initSeats() は onload
イベントハンドラの
関数リテラルの中で
呼び出されます。

この onload イベントハンドラに
イベントオブジェクトは必要ないので、
引数 evt は無視されます。

どこでつながっている？

関数リテラルを使ってイベントにつなぐ場合、まだいくつか問題があります。
onload イベントハンドラはページのヘッダ部にある <script> タグの中で
代入できますが、 onload に結びついたコードはページ読み込みが完了する
(onload イベントが発生する)まで実行されません。このため <body> タグの
onload 属性に initSeats() を書いていました。では関数リテラルを使う場合、イベントハンドラをどこでつなげばいいでしょうか？

onload イベントハンドラのコールバック関数に戻ってみましょう。この関数はページで発生するすべてのイベントをつなぐのに最適な場所です。

onload イベント
ハンドラはすべての
イベントを初期化
するのに最適な
場所です。

```
window.onload = function() {  
    // ここで他のイベントをつなぎます  
    ...  
    // 座席の画像を初期化します  
    initSeats();  
};
```

onload イベントハンドラの
中でページの他のイベントを
つなぐことができます。

onload イベントに固有の
コードもイベントハンドラの
中で実行されます。

このコードの要点は、 onload イベントハンドラがイベント初期化の関数になっていることです。この初期化関数ではページで発生するすべてのイベントが設定されています。このため onload イベントハンドラは座席の初期化などページ読み込み時の通常処理を実行するだけでなく、他のすべてのイベントハンドラコールバック関数の設定も実行しています。

素朴な疑問に

答えます

Q: コールバック関数はなぜ重要なのでしょうか?

A: コールバック関数が重要なのはコードの外側で起きた事に反応できるからです。コールバック関数を作ると、コードから関数を呼び出すかわりに何かが起きるのを待って、何かが起きたら即座にアクションに飛ばすことができます。何かが起きたとき、それをコールバック関数に知らせるのはブラウザの責任です。あなたの仕事は舞台を設定するだけです。通常はイベントですが引き金になるとコールバック関数を結ぶ作業になります。

Q: イベントハンドラ以外にコールバック関数はありますか?

A: はい。イベントハンドラ以外のコールバック関数の用途については12章で説明します。Ajaxを使ってサーバから送信されたデータを処理するときコールバック変数を使います。

Q: 関数リテラルについてまだ混乱しています。それは何ですか、なぜ重要なんですか?

A: 関数リテラルは名前のない関数本体にすぎません。数値や文字列などのデータ断片はリテラルですが、関数リテラルもこれと同じです。関数リテラルが重要である理由は、

ちょっとした仕事をこなすコールバック関数が必要なとき、関数リテラルがぴったりだからです。この関数は1回しか呼ばれませんし、あなたのコードから呼ばれるかもしれません。なので、名前つき関数を作つて、その参照を代入するのではなく、関数リテラルを作つて、これをイベントプロパティに直接代入しています。名前つき関数が必要な状況であれば、コーディングの効率がよくなるので、関数リテラルを使うメリットがあります。関数リテラルがほんとうに必要になるのは、引数のある関数のように関数の参照だけではできない処理が必要な場合です。このことは忘れないようにしましょう。

自分で考えてみよう

Mandangoの新しいonloadイベントハンドラ関数を完成させてください。

```
window.onload = function() {
    // Find Seat ボタンのイベントをつなぎます
    document.getElementById("findseats"). ..... = ....;

    // 座席画像のイベントをつなぎます
    document.getElementById("seat0"). ..... = function(evt) { ..... };
    document.getElementById("seat1"). ..... = function(evt) { ..... };
    document.getElementById("seat2"). ..... = function(evt) { ..... };

    ...
    // 座席画像を初期化します
    ....;
};
```

自分で考えてみよう の答え

Mandangoの新しい onload イベントハンドラ関数を完成させてください。

この onload イベント
ハンドラ全体が
関数リテラルです。

```
window.onload = function() {  
    // Find Seat ボタンのイベントをつなぎます  
    document.getElementById("findseats").onclick = findSeats;  
  
    // 座席画像のイベントをつなぎます  
    document.getElementById("seat0").onclick = function(evt) {showSeatStatus(0)};  
    document.getElementById("seat1").onclick = function(evt) {showSeatStatus(1)};  
    document.getElementById("seat2").onclick = function(evt) {showSeatStatus(2)};  
    ...  
    // 座席画像を初期化します  
    initSeats();  
};
```

onload のタスクの
最後で initSeats() を
呼び出します。

findSeats() は関数参照を
使って onclick イベントに
つながれています。

各座席画像の onclick プロパティ
は onclick イベントハンドラを
設定するためアクセスされます

showSeatStatus() は
引数が受け取れるように
関数リテラルの中で
呼び出されます。

重要ポイント

- コールバック関数はスクリプトの外で起きたことへの対応としてブラウザから呼び出されます。
- 関数参照は関数をあたかも変数であるかのように代入するときに使うことができます。
- 関数参照を使うと HTML コードを変更することなく JavaScript コードの中でイベントハンドラ関数をつなぐことができます。
- 関数リテラルは名前のない関数であり、名前つき関数が必要ない状況で便利です

素朴な疑問に答えます

Q: Mandangoの onload イベントハンドラはなぜ関数リテラルで作成されているのですか？

A: 名前つき関数を作る必要がないからです。この関数は1回しか呼ばれません。onloadイベントに応答するだけです。もちろん名前つき関数を作り、この参照を window.onload に代入するのは簡単にできますが、関数リテラルを使って直接イベントと結びつけた方がコードバック関数とイベントの接続関係が明確になります。

Q: 他のコールバック関数の場合でも onload イベントハンドラに結びつける必要がありますか？

A: はい。ページのヘッダ部にある <script> タグの中直結できると思われているかもしれません。ページのコンテンツの読み込みが完了していないと getElementById() の呼び出しあはすべて失敗してしまい、イベントハンドラにつながりません。onload ハンドラはページの読み込みが完了したことを確実に保証します。

HTML ページの殻

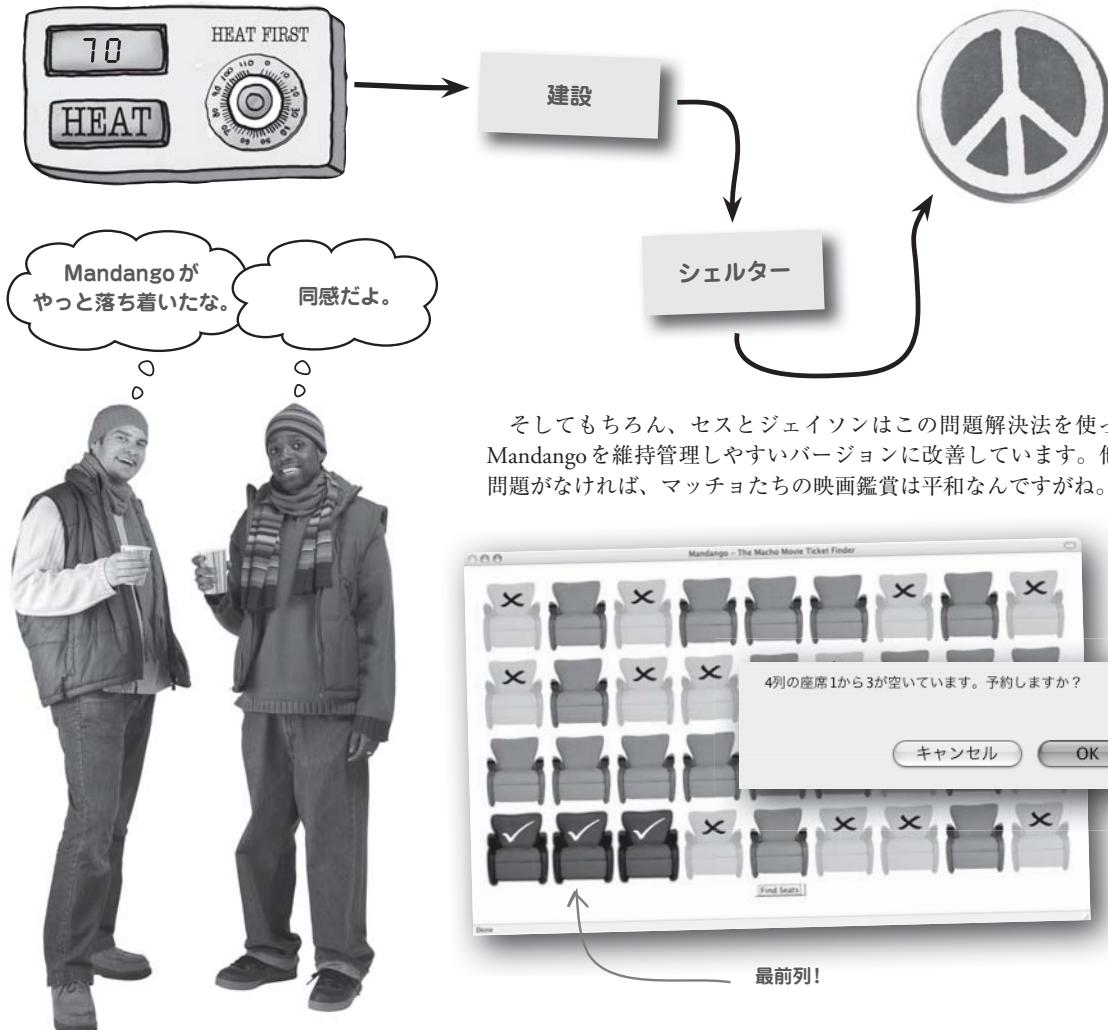
Mandango は JavaScript と HTML のコードを分離したので、ページの構造が極めて簡潔になりました。これで HTML コードの維持管理が容易になり、JavaScript コードをいじってアプリケーションを壊す心配もなくなりました。



```
<body>
<div style="margin-top:75px; text-align:center">
  <img id="seat0" src="" alt="" />
  <img id="seat1" src="" alt="" />
  <img id="seat2" src="" alt="" />
  <img id="seat3" src="" alt="" />
  <img id="seat4" src="" alt="" />
  <img id="seat5" src="" alt="" />
  <img id="seat6" src="" alt="" />
  <img id="seat7" src="" alt="" />
  <img id="seat8" src="" alt="" />
  <img id="seat9" src="" alt="" />
  <img id="seat10" src="" alt="" /><br />
  <img id="seat11" src="" alt="" />
  <img id="seat12" src="" alt="" />
  <img id="seat13" src="" alt="" />
  <img id="seat14" src="" alt="" />
  <img id="seat15" src="" alt="" />
  <img id="seat16" src="" alt="" />
  <img id="seat17" src="" alt="" />
  <img id="seat18" src="" alt="" />
  <img id="seat19" src="" alt="" /><br />
  <img id="seat20" src="" alt="" />
  <img id="seat21" src="" alt="" />
  <img id="seat22" src="" alt="" />
  <img id="seat23" src="" alt="" />
  <img id="seat24" src="" alt="" />
  <img id="seat25" src="" alt="" />
  <img id="seat26" src="" alt="" />
  <img id="seat27" src="" alt="" />
  <img id="seat28" src="" alt="" />
  <img id="seat29" src="" alt="" /><br />
  <img id="seat30" src="" alt="" />
  <img id="seat31" src="" alt="" />
  <img id="seat32" src="" alt="" />
  <img id="seat33" src="" alt="" />
  <img id="seat34" src="" alt="" />
  <img id="seat35" src="" alt="" />
  <img id="seat36" src="" alt="" />
  <img id="seat37" src="" alt="" />
  <img id="seat38" src="" alt="" />
  <img id="seat39" src="" alt="" /><br />
  <input type="button" id="findseats"
        value="Find Seats" />
</div>
</body>
```

JavaScriptのもう一歩

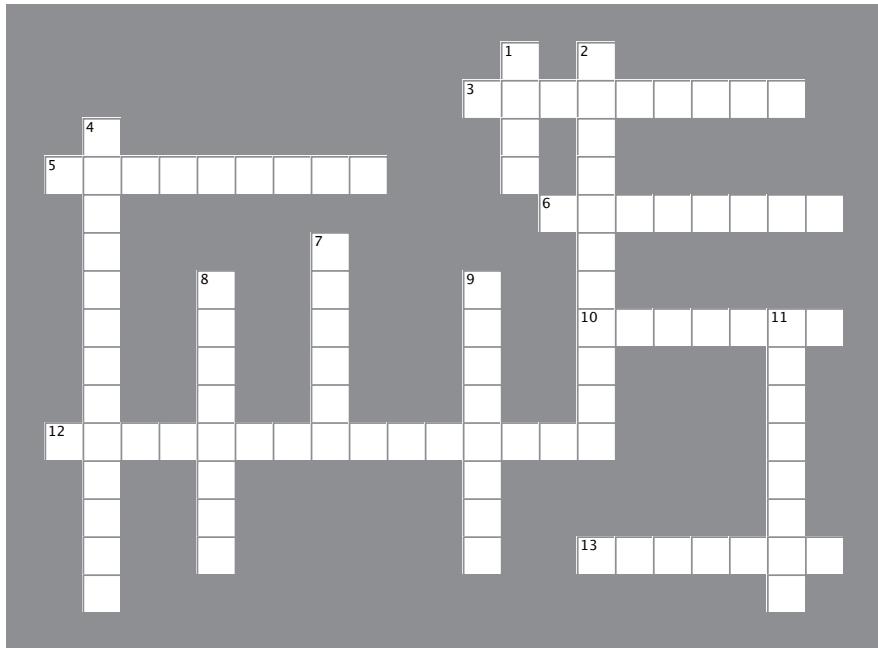
世界平和を解決する方法は取り上げませんでしたが、JavaScript を気候変動対策に役立てるための一歩は踏み出せました。大きな問題を小さな問題に変えて、それぞれに固有な目的に焦点をあわせ、コードの再利用を促進すること、これはすべて関数によって実現され、スクリプトは改善されます。





JavaScriptクロスワード

さあ、休憩の時間ですよ。気分を楽にして、クロスワードパズルで頭をひねってみましょう。



ヨコのカギ

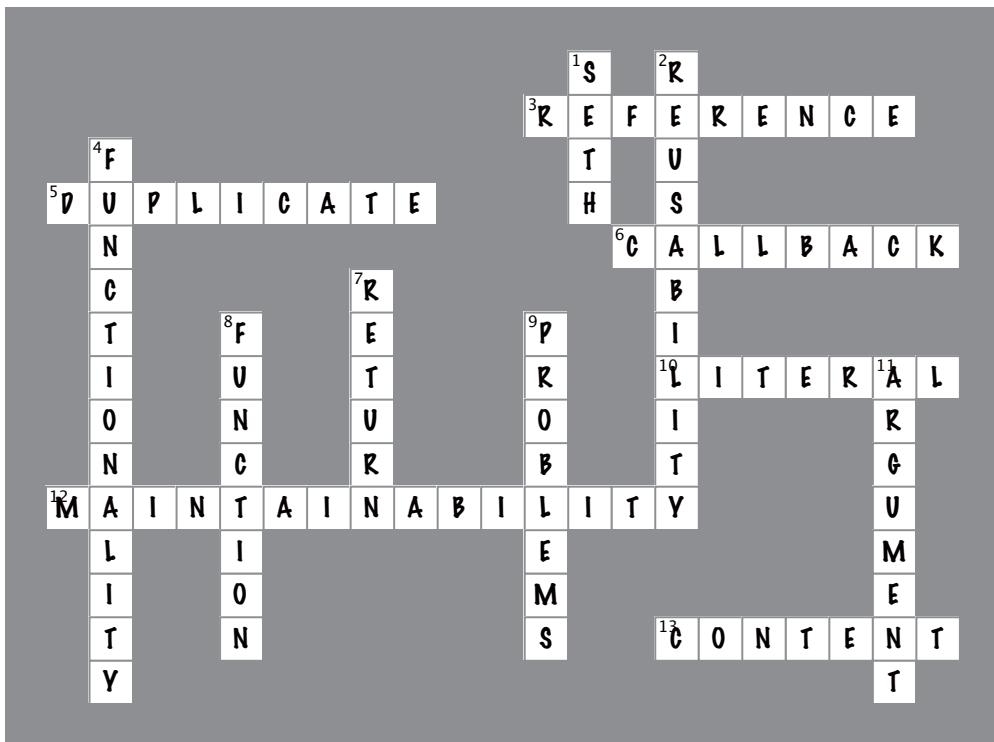
3. 関数を変数に代入するとき、関数 を使います。
5. 関数を使うとコードの がなくせます。
6. この種類の関数は直接呼び出せません。
10. 名前のない関数の本体。
12. コードが比較的簡単に変更できるとき がいいと見なされます。
13. HTMLコードはウェブページのこの部分を表現します。

タテのカギ

1.はMandangoに満足しています。
2. 関数を使うと不必要的重複がなくなるのでコードの が向上します。
4. JavaScriptはウェブページのこの部分を表現します。
7. 関数からデータを返すときは、データを で返します。
8. 再利用可能なJavaScriptコードの断片。
9. 関数はこれを分割するのに適しています。
11. 関数にデータを渡すときこれを使います。



JavaScript クロスワードの答え

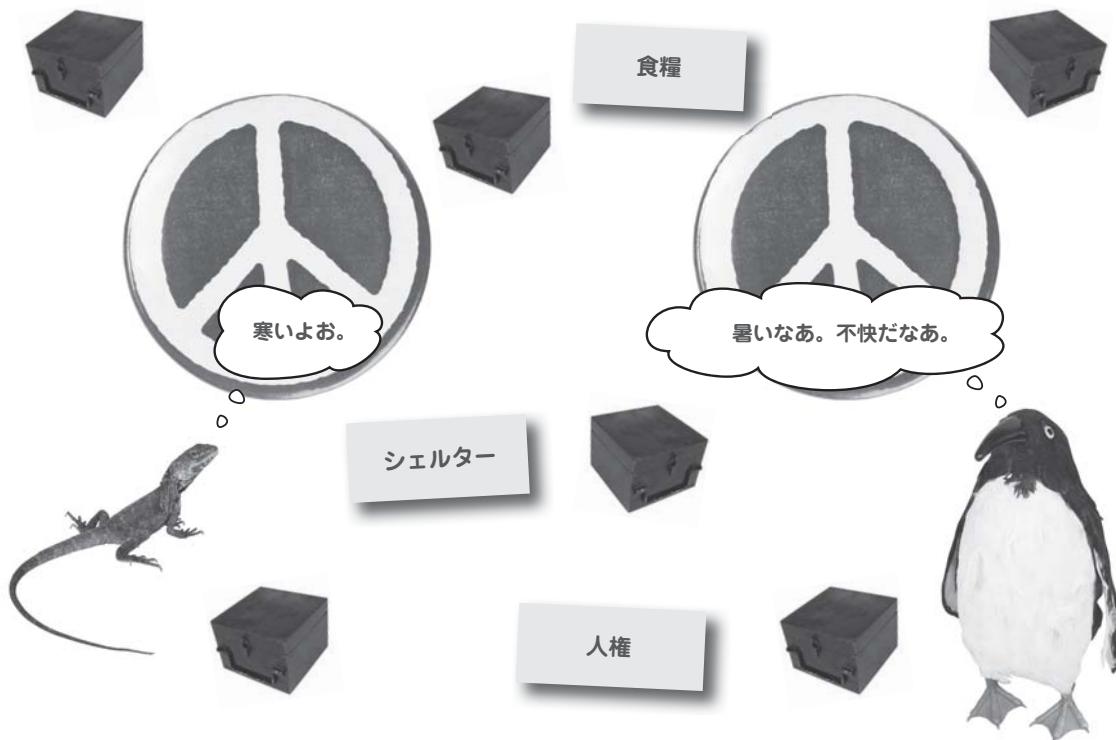
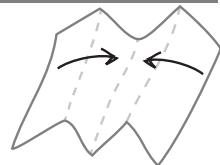


折り畳みページ

脳のところで
折り畳んでから
謎を解決しましょう。

関数はJavaScriptの生命に
何を加えるのでしょうか？

左脳と右脳がご対面！



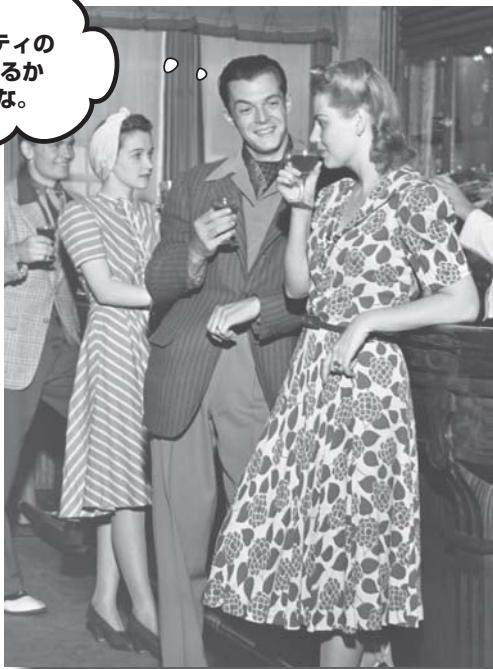
平和はいつも巧妙な提案です。

JavaScriptのコードでさえ、最もよく組織化されたコードだけが
安定と平穏に導かれます。少なくともJavaScriptに関しては、
快適な生活にたどりつくのはそう簡単ではありません。

7章 フォームと検証

ユーザに洗いざらい 話してもらう

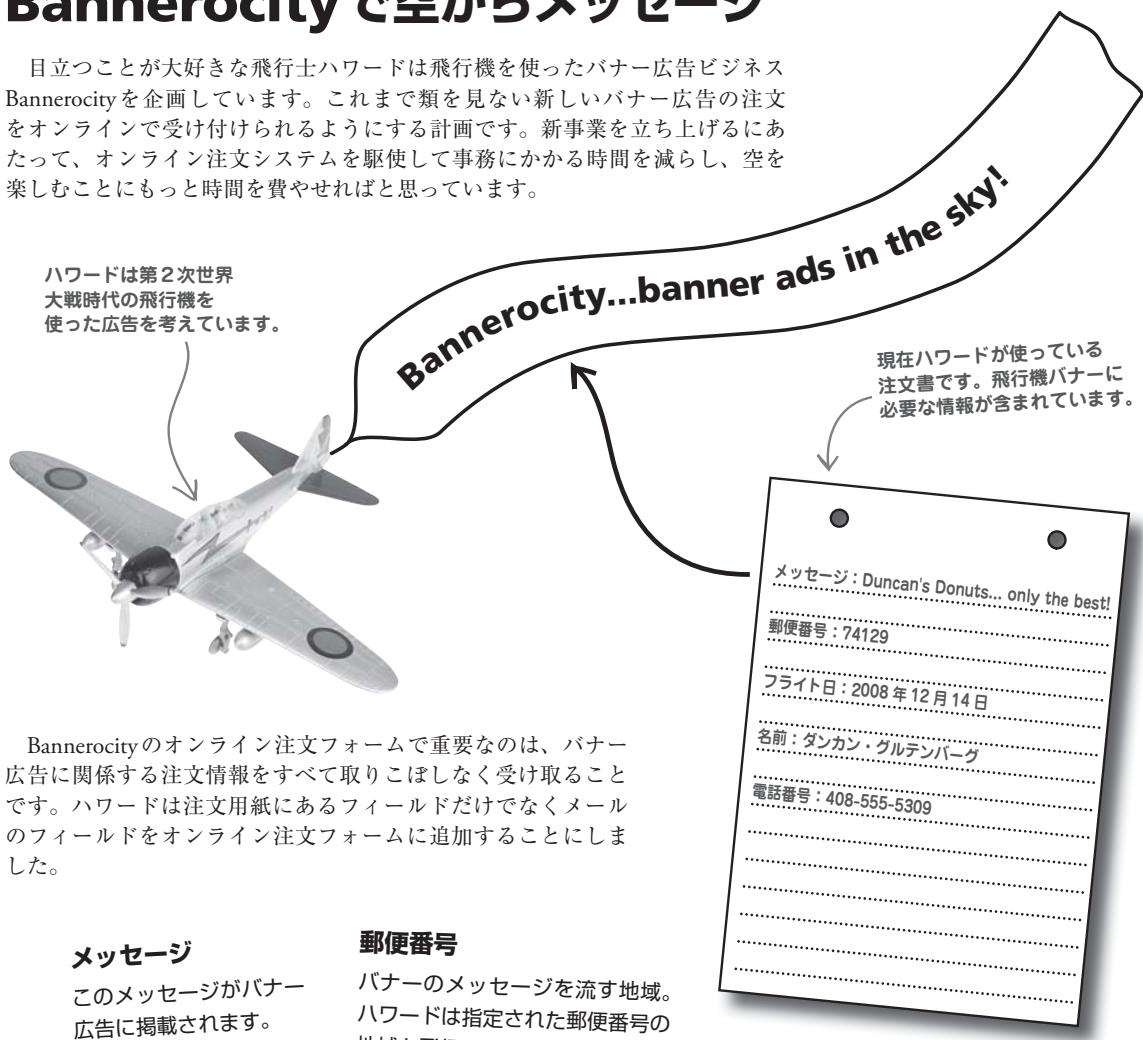
ぼくの上品で
礼儀正しい人柄でベティの
電話番号を聞き出せるか
ためしてみようかな。



JavaScriptを使ってユーザの情報をうまく聞き出すには、人あたり良く振る舞ったり、あるいはコソコソする必要はありませんが、**慎重さ**は必要です。オンラインのフォームで提供されたデータが正確かつ有効であると前提できないのに、それをそのまま受け取ってしまい失敗することがあります。フォームにデータが入力されたとき **JavaScript** のコードを経由してフォームデータを送信すれば、ウェブアプリケーションの信頼性はもっと高くなりサーバの負荷も軽減されます。人目をひくビデオやかわいいペットの写真などの重要なコンテンツのために、**貴重な帯域を節約**する必要があります。

Bannerocityで空からメッセージ

目立つことが大好きな飛行士ハワードは飛行機を使ったバナー広告ビジネス Bannerocityを企画しています。これまで類を見ない新しいバナー広告の注文をオンラインで受け付けられるようにする計画です。新事業を立ち上げるにあたって、オンライン注文システムを駆使して事務にかかる時間を減らし、空を楽しむことにもっと時間を費やせればと思っています。



Bannerocityのオンライン注文フォームで重要なのは、バナー広告に関する注文情報をすべて取りこぼしなく受け取ることです。ハワードは注文用紙にあるフィールドだけでなくメールのフィールドをオンライン注文フォームに追加することにしました。

メッセージ

このメッセージがバナー広告に掲載されます。

郵便番号

バナーのメッセージを流す地域。
ハワードは指定された郵便番号の地域を飛行します。

フライト日

バナー広告を流す日付。

氏名

顧客の名前。

電話番号

顧客の電話番号。

電子メール

顧客の電子メールアドレス。

BannerocityのHTMLフォーム

ハワードはHTMLを使ってBannerocityのオンライン注文フォームを作ってみました。なかなかのできです。

Bannerocity – Personalized Online Sky Banners

メッセージを入力してください:

飛行地域の郵便番号を入力してください:

バナー広告を流す日付を入力してください:

お客様の名前を入力してください:

電話番号を入力してください:

メールアドレスを入力してください:

完了

このオンライン
注文フォームは
なかなかのできだね。

Bannerocity注文フォームには必要なフィールドがすべて含まれています。JavaScriptコードを使わなくても注文を受け取れるようになっています。何か落とし穴はないでしょうか？

ハワードはここ
しばらく民間企業で
働いていますが、制服を
脱ぐことはなさそうです。



自分で考えてみよう

ハワードのHTML注文フォームを使って注文してみてください。
ご心配なく、このバナー広告に料金はかかりません。

メッセージを入力してください:

飛行地域の郵便番号を入力してください:

バナー広告を流す日付を入力してください:

お客様の名前を入力してください:

電話番号を入力してください:

メールアドレスを入力してください:



自分で考えてみよう の答え

ハワードのHTML注文フォームを使って注文してみてください。
ご心配なく、このバナー広告に料金はかかりません。

実は 32 文字までしかバナーに
表示できません。そのため
これは長すぎます。

郵便番号が長すぎます。
5 衔にしてください。

メッセージを入力してください：

飛行地域の郵便番号を入力してください：

バナー広告を流す日付を入力してください：

お客様の名前を入力してください：

電話番号を入力してください：

メールアドレスを入力してください：

氏名は
OKですね。

ドメイン名に .biz が
欠けているので、この
メールアドレスは不正です。

電話番号は #####-#####-#####
の形式にします。
括弧は使えません。

この日付形式は
MM/DD/YYYY に
なっていません。

HTMLではできないこと

ハワードはJavaScriptを使ってフォームのデータが有効なのかチェックする
必要があることに気づきました。オンラインのフォームはデータが入力され
るだけで、それが有効であるとは限りません。ユーザが正しいデータを入力
したか確かめる手段が必要です。バナー広告のメッセージは32文字までと
か、日付はMM/DD/YYYYで入力してください、といったヒントがユーザに
示されなければ、ユーザは間違いに気づきません。

ごめんなさい、
バナー広告テキストは
32 文字までなんです。

メッセージを入力してください：

JavaScriptの力を
借りればこの不正
データを防げます。

HTML フォームは
まともに話ができません。

ちょっとした問題があります。JavaScriptコードがどんなに賢く
データを操作できるとしても、JavaScriptからフォームデータにアク
セスする方法がわからないかぎり、ハワードの役に立ちません。

フォームデータにアクセスする

フォームに入力されたデータにアクセスするには、まずフォームの各フィールドを一意に識別する必要があります。HTMLコードでは、そのための属性が2つあります。

`id` 属性はページの任意の要素を一意に識別するのに使います。

<input `id="zipcode"`

`name` 属性はフォーム内のフィールドを一意に識別するのに使います。

飛行地域の郵便番号を入力してください :

どちらの属性も入力フィールドの識別子として使えます。

フォームフィールドを識別する方法が2つあるのは、それぞれのアクセス方法が異なるためです。最初の`getElementById()`を使う手法は、ページの要素にアクセスするときに使います。フォームフィールドにアクセスする場合、これよりもっと簡単な手法があります。

フォームフィールドはすべて`form`オブジェクトを持っています。データを検証するとき、このオブジェクトを関数に渡すことができます。

```
<input id="zipcode" name="zipcode" type="text" size="5" onclick="showIt(this.form)" />
```

`form`オブジェクトのいいところは、フォームにあるすべてのフィールドを配列として保持している点です。配列の要素は数値インデックスではなく`name`属性に設定された一意の識別子を使って格納されています。`form`オブジェクトが`theForm`という名前の引数で関数に渡されると、郵便番号フィールドに入力された値は以下のようにしてアクセスできます。

`form`オブジェクトである引数の名前。

```
function showIt(theForm) {
    alert(theForm["zipcode"].value);
}
```

フォームフィールドの名前。
<input>タグの`name`属性で設定されています。

フィールドそのものではなく
フィールドの値を取り出します。

郵便番号フィールドの値を表示します。

100012

OK

このアプローチは`getElementById()`を使うアプローチよりも優れています。ただし使う道具が少ないのでコードは読みやすくなります。`form`オブジェクトは手っ取り早い手段なのです。

素朴な疑問に

答えます

Q：個々のフォームフィールドごとにformオブジェクトにアクセスできるのはなぜですか？

A：例外もありますが、フォームフィールドは検証の関数を呼び出すことができます。呼び出された関数は、呼び出したフィールド以外のフィールドのデータにアクセスする必要があります。この場合、各フィールドから利用できるformオブジェクトが他のフィールドにアクセスするための便利な手段になります。通常formオブジェクトは検証の関数に渡されるので、検証の関数は必要に応じてフィールドをつかむことができます。Bannerocityは注文フォームのフィールドにアクセスするため、formオブジェクトにかなり依存します。

Q：valueはフォームフィールドのプロパティですか？ということは、各フォームフィールドはオブジェクトなのですか？

A：どちらもその通りです。各フォームフィールドはオブジェクトで表現されます。formオブジェクトは、フォームにあるフィールドをオブジェクトとしてアクセスする手段を提供します。フォームフィールドのオブジェクトをform["オブジェクト名"]という書き方で取得しておけば、そのオブジェクトのvalueプロパティにアクセスすれば値が取得できます。オブジェクトに関しては9章と10章で説明します。

JavaScriptでフォームデータに
アクセスするのが重要なのは、データが
有効か確認するためね。それはわかったけど、
いつデータを検証したらいいの？

フォームデータをチェックするタイミングは、
処理に選んだユーザ入力イベントによって
異なります。

データ検証はイベントによって開始されます。ユーザが
特定のフィールドにデータを入力したとき、どのイベントが
発生するか知る必要があります。データが入力された直後に
発生するイベントに反応するのが課題です。では、どの
イベントが発生するのでしょうか？

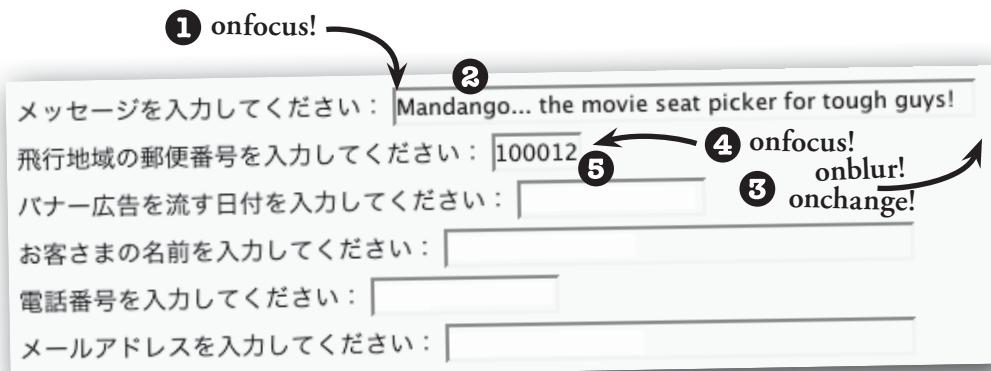


フォームフィールドで発生するイベントの連鎖

フォームにデータが入力されるとき、立て続けにイベントが発生します。これらのイベントを使うと、フィールドごとにデータ検証の入り口を設定することができます。そのためには、典型的な入力シーケンスにおいてどのイベントがいつ発生するのか理解しておく必要があります。

- ① 入力フィールドを選択する(onfocus)。
- ② フィールドにデータを入力する。
- ③ 入力フィールドから離れ、次のフィールドに移る
(onblur/onchange)
- ④ 次の入力フィールドを選択する(onfocus)。
- ⑤ フィールドにデータを入力する。

フォームにデータを
入力すると、一連の
興味深いJavaScript
イベントが発生します。



あるフィールドが選択されたとき、onfocusイベントが発生します。フィールドから入力カーソルがなくなったとき、onblurイベントが発生します。onchangeイベントはonblurに似ていますが、入力カーソルがなくなっただけでなくデータが変更されたときだけ発生します。



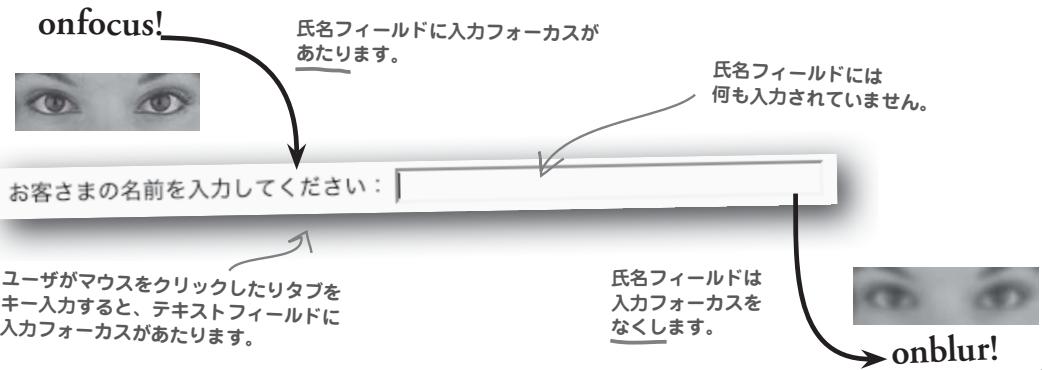
頭の体操

フォームのフィールドのデータを検証する際に
最も意味のあるイベントはどれでしょう？

フォーカスがなくなったときはonblur

データ検証に onchange イベントを使う方法はすでに説明しましたが、空のフィールドを検証するときには問題があつて使えません。フォームが最初に読み込まれたときフィールドに何もデータがないので、ユーザが空のフィールドを移動していっても onchange は発生しません。この問題を解決するのが onblur イベントです。フィールドからフォーカスがなくなったときこのイベントが発生します。

onblur イベントは
データ検証に
最適です。



onchange と違って onblur はフィールドのフォーカスがなくなったときに発生します。データが変更されたかどうかは関係ありません。onblur はとても強力なのですが、データ検証の結果をユーザに知らせるときには注意する必要があります。たとえば検証結果をアラートボックスで知らせるのは簡単ですが危険もあります。

素朴な疑問に答えます

Q： ユーザが実際にフォームデータを入力したとき、いくつかイベントが発生しませんか？

A： はい、onkeypress、onkeyup、onkeydown といったイベントが、キーボード操作に反応して発生します。これらのイベントに反応しなければならない状況もありますが、通常のデータ検証にはあまり適していません。イベントが発生したとき、ユーザはまだ情報を入力している最中だからです。データ検証でこれらのイベントを使うと、ユーザがデータを入力するたびに、途中で何度もイベントが発生してしまいます。なので、データの入力が完了しユーザがそのフィールドから離れるまで待った方がいいです。onblur イベントに反応すればこれが実現できます。

Q： onblur はなんとも変なイベントの名前ですね。どういう意味ですか？

A： onblur は onfocus と対になっています。onfocus は要素やフォームフィールドに入力フォーカスがあつたときに発生するイベントです。一方、onblur はフィールドのフォーカスがなくなったときに発生します。この文脈での「focus」という単語は、視覚の焦点という本来の意味ではありませんが、「blur」という単語は焦点がぼけた状態を意味します。JavaScript の言葉遊びは、少しあわざりにくいですね。ともかく、フィールドのフォーカスがなくなったとき onblur イベントが発生することだけ覚えておいてください。

アラートボックスを使って検証メッセージを表示できます

ユーザーに情報をすばやく表示するための手軽な手段がアラートボックスです。最も簡単な形式でフォームデータに問題があることをユーザーに知らせます。onblurイベントの処理でフォームデータに問題があることを検出したら、`alert()`を呼び出すだけです。

フォームフィールドが空か調べます。

検証関数を呼び出して
氏名のデータを検証します。

ユーザーに問題への対応の
仕方を知らせます。

値を入力してください。

OK

```
function validateNonEmpty(inputField)
{
    // 入力値にテキストが含まれているか調べます
    if (inputField.value.length == 0) {
        // データが不正なのでユーザーに知らせます
        alert("値を入力してください。");
        return false;
    }

    return true;
}
```

氏名フィールドが
空なのでアラートを
表示します。



自分で考えてみよう

以下の入力シーケンスでは`onblur`イベントが何回発生しますか？
`onchange`イベントは何回発生しますか？ `onfocus`は心配いりません。

お客様の名前を入力してください :

電話番号を入力してください :

メールアドレスを入力してください :

`onblur` イベントの数
.....

`onchange` イベントの数
.....



以下の入力シーケンスでは onblur イベントが何回発生しますか？
onchange イベントは何回発生しますか？ onfocus は心配いりません。

お客様の名前を入力してください : onblur!
onchange!

電話番号を入力してください : onblur!

メールアドレスを入力してください : onblur!
onchange!

onblur イベントの数 3
onchange イベントの数 2

特別座談会



今晚の対談のゲストは onblur と onchange です。
間違ったフォームデータに反応するときの問題を議論します。

onblur:

最近、スクリプトはユーザが何をしようとしているのか、ずっと気にしているみたいですね。それで私たちがこの席に呼ばれたんでしょう。

そのことについてお話しするつもりでした。空のデータがあるって噂が広まっています。あなたのやり方に問題があるんじゃないかな、と指摘する声もあります。

そうですよね。データが変更されたときのあなたの信頼性が問題にされているわけじゃない。けして変更されない空のデータがフォームにあったとき、どうなるのかが問題なんです。

onchange:

まったくです。要素やフォームフィールドのフォーカスがなくなったり、データが変更されたとき、それを知らせるのが、私たちの仕事というわけですね。

率直なところ、そうした非難の声に、少しショックを受けてます。データが変更されたことをスクリプトに知らせるとき、タイミングを外したことはないんですけどね。

onblur:

おっしゃる通り、意味がないです。でも、どうなるか試してみようってユーザがいるんですよ。

まあまあ、落ち着いてください。あなたのせいではありません。変更されていないデータについては、あなたに責任はありません。そもそも、名前が**onchange**なんですから。

冷静になりました。私が言ったように、それはあなたの責任ではありません。フォームのフィールドが空でないか、スクリプトが検証するとき、その検証コードの引き金として、あなたを使うべきではないですから。

まあまあ、落ち着いて。そう過剰に反応しないことです。検査コードの引き金として、あなたが完璧でないとしても、データが変更されたかどうか、スクリプトはいっさい関心がない、ということではないですから。ユーザにフォームのデータを編集させる場合を考えてみましょう。データがほんとうに変更されたときだけデータを保存できるようにするには、まさにあなたの力が必要です。

なくてはならない存在ですよ。だから自分を責める必要はないんです。

どういたしまして。もっとお話を続けたいところですが、検証データがありますので。

onchange:

空のフィールドがあるフォームをユーザが送信することがある、ということですか？ それって意味あるんですかね。

空のフィールドのあるフォームがあって、ユーザは何もデータを入力せずに、そのままフォームを送信するわけですか。ああ、不安で発作が起きそうです。

でも、いま話したように、空データのせいでスクリプトを呼びそこなってしまい、穴があき始めているとしたら、問題ですよね。

それを聞いてほっとしました。たとえ私がもはや誰の役にも立っていないとしても。ああ、また発作が…。

そうなんです。まだ私の存在価値はありますよね。

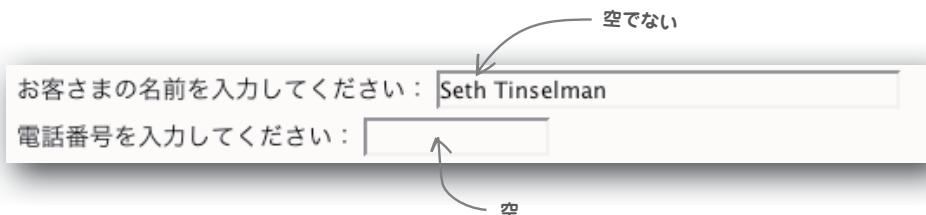
ありがとう。おかげで元気が出てきました。

データがあるかどうかをチェックする

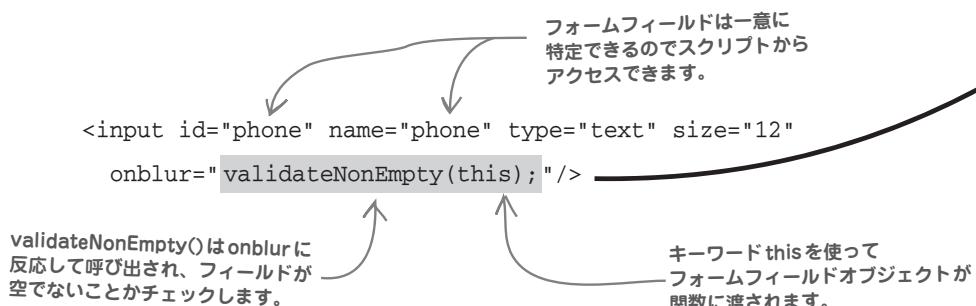
Bannerocityに戻ります。ハワードは最低限すべてのフィールドにデータがあることを検証する必要があると思っていますが、JavaScriptの立場にたつと別の見方でとらえ直す必要があります。つまり、フィールドにデータがあるか調べるかわりに、フィールドにデータがないことを調べるのです。「何かある」を「空ではない」と解釈するわけです。

何かある = 空でない

このように直感に反した捉え方をする理由は、フォームフィールドにデータがあるか調べるよりも空であるか調べる方が容易だからです。データ検証の最初の防衛線はフィールドが空でないかチェックすることです。



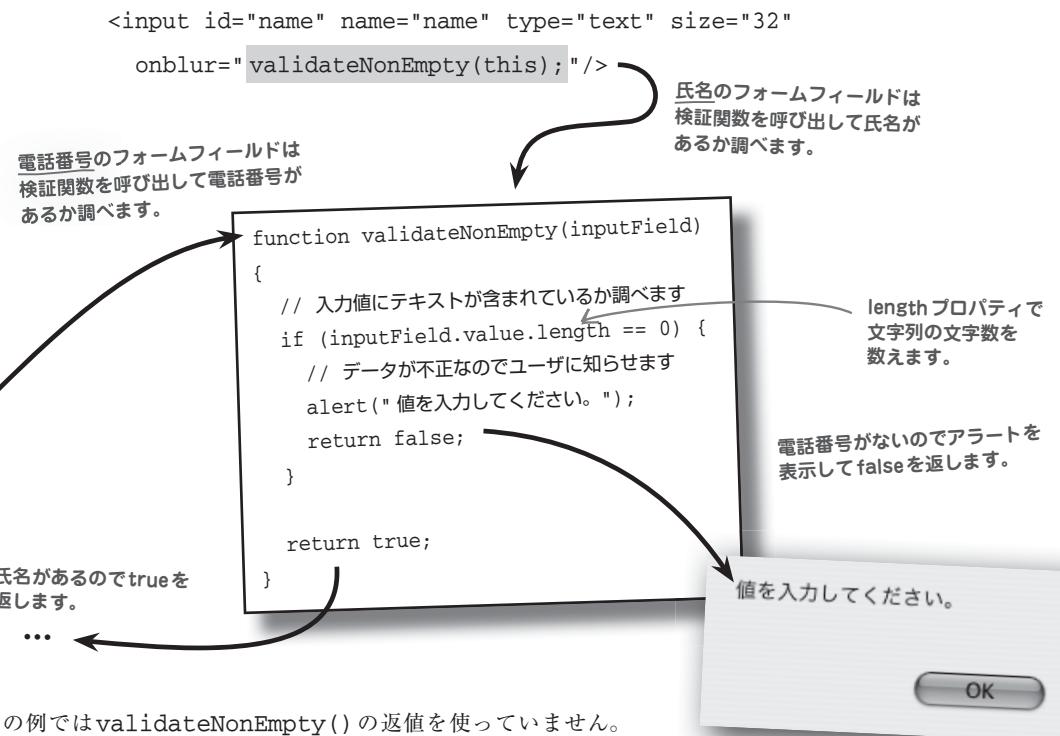
ハワードの書く検証関数は、空でない検証を実行するためにフィールドごとのonblurイベントに反応する必要があります。



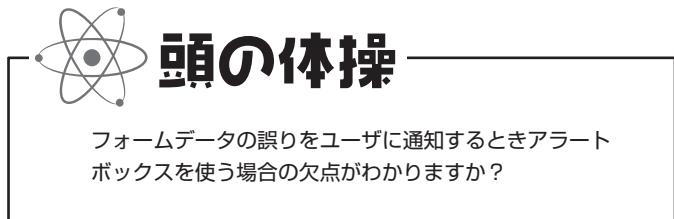
このコードではキーワードthisを使ってフォームフィールド自体を参照しています。フォームフィールドをオブジェクトとして検証関数に渡すことで、検証関数はフォームフィールドの値だけでなく、すべてのフォームフィールドを含むformオブジェクトにもアクセスできます。

フィールドが空でないか検証する

各フォームフィールドではどれも同様にonblurイベントをvalidateNonEmpty()につないでいます。各フィールドのonblurイベントをこの関数につなぐことで、フォームのすべてのデータを検証できます



この検証関数は、
フォームフィールドが
空でないかチェックします。



うるさいアラートボックスを使わない

ほどなくハワードは、ユーザにデータが無効であるのを知らせる手段としてアラートボックスが最善ではないことに気がつきました。ユーザがBannerocityの注文を入力しようとすると、アラートボックスが表示されるので、苦情が寄せられています。ポップアップウィンドウが表示されると、多くの人はそれをうるさく感じるようになりました。データ検証も例外ではありません。たとえアラートボックスの情報が役に立つとしてもです。

ハワードは「控えめなヘルプシステム」で問題を解決することにしました。アラートボックスを使わないので、データ入力のフローを中断しません。このアプローチでユーザに検証結果を知らせるには、フォームに新しいHTML要素を追加する必要があります。



この新しいHTML要素で検証のヘルプメッセージを表示します。

電話番号を入力してください： 値を入力してください。

新しく追加するHTMLヘルプ要素は、アラートボックスよりも優れています。ユーザの操作を邪魔することなく、今までと同じ情報を表示できます。必要な作業は、フォームフィールドの横にタグを追加し、その名前をフォームフィールドの名前と対応付けることです。この新しく追加するタグのコードは、入力フィールドのすぐ後にあります。

validateNonEmpty() の
第2引数にはヘルプテキスト
要素を渡します。

```
<input id="phone" name="phone" type="text" size="12"  
onblur="validateNonEmpty(this, document.getElementById('phone_help'));" />
```


 タグは最初は空ですが、
電話番号フォームフィールドに
関連したIDがあります。

スタイルクラスを使って
ヘルプテキストを赤の
イタリックフォントに
整形します。

この2つのIDは同じになります。
入力フィールドに対応する
ヘルプテキストを表示します。

ヘルプテキストの表示場所である要素を追加したら、後は実際にヘルプメッセージを表示するコードを書くだけです。validateNonEmpty()に追加した第2引数を利用すれば、ユーザにヘルプテキストが表示されたかどうか確認することができます。

空でない検証をさらに改善する

ハワードは控えめなヘルプメッセージを実現するために、`validateNonEmpty()`を改良しました。この関数はフォームフィールドに対応するヘルプメッセージの設定と消去を処理します。

ヘルプテキストのオブジェクトは第2引数として関数に渡されます。

```
function validateNonEmpty(inputField, helpText) {
  // 入力値にテキストが含まれているか調べます
  if (inputField.value.length == 0) {
    // データが不正なのでヘルプメッセージを設定します
    if (helpText != null) {
      helpText.innerHTML = "値を入力してください。";
      return false;
    }
  } else {
    // データに問題ないのでヘルプメッセージをクリアします
    if (helpText != null)
      helpText.innerHTML = "";
    return true;
  }
}
```

ヘルプテキスト要素が存在するか確認して(`helpText != null`)、`innerHTML`プロパティにヘルプメッセージを設定します。

フォームフィールドにデータが入力されたらヘルプテキストをクリアします。

煩わしいアラートボックスがなくなって、かなりよくなったね。

控えめなヘルプのアプローチのおかげで Bannerocity でのデータ検証は大きく改善されました。JavaScript という薬を適切に使ってユーザの体験を中断しないようにしています。

飛行地域の郵便番号を入力してください：	<input type="text"/> 値を入力してください。
バナー広告を流す日付を入力してください：	<input type="text"/> 値を入力してください。
お客様の名前を入力してください：	Seth Tinselman
電話番号を入力してください：	<input type="text"/> 値を入力してください。
メールアドレスを入力してください：	<input type="text"/> 値を入力してください。

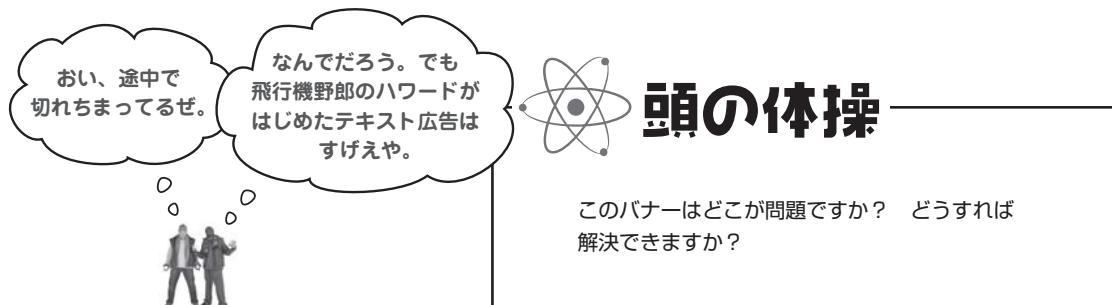
データが空なので、控え目なヘルプメッセージが表示されます。

氏名のデータはあるので、ヘルプメッセージは表示されません。



多すぎるのは問題です

空でない検証はうまく動いていますが、データが多すぎるのも問題のようです。ハワードの最新バナーをみるとわかるように、Bannerosity の注文フォームには新しい問題が生じています。



サイズの問題

Bannerocityで問題なのは、バナーの文字数は32文字までなのに、注文フォームのメッセージフィールドには上限がないことです。ユーザがメッセージを入力しなかつたときは警告が表示されるので問題ないのですが、メッセージが制限字数を超えていても問題なく通過してしまいます。これはハワードにとっては大問題なのです。

ユーザが入力したテキストは長すぎますが、Bannerocityは問題を知らせません。

メッセージを入力してください：

Mandango... the movie seat picker fo

バナーの制限文字数を越えているので、広告テキストが尻切れになっています。
これではダメですね。

字数に制限があるスペースに無制限にテキストを表示しようとしてもうまくいきません。セスやジェイソンのように顧客を怒らせるだけです。

このバナーテキストは32文字以内に収まっています。

メッセージを入力してください：

Mandango... macho movie tickets!

制限文字数以内に収まっているので、バナーのテキストに問題はありません。

自分で考えてみよう

Bannerocityの文字の長さを検証する方法を疑似コードで書いてください。
長さの下限と上限の両方で検証します。



自分で考えてみよう の答え

Bannerocityの文字の長さを検証する方法を疑似コードで書いてください。
長さの下限と上限の両方で検証します。

Bannerocityの
バナーテキストの
場合、関数の引数
minLengthは1に、
maxLengthは32に
設定します。

```
If (fieldValue が minLength よりも小さいか、または、fieldValue が maxLength より大きい)
    ヘルプテキストを表示
Else
    ヘルプテキストをクリア
```

データの長さを検証する

新しいvalidateLength() の役割はフォームフィールドのテキストの文字数が下限と上限の間に収まっているか確認することです。Bannerocityの場合、この関数はバナーテキストフィールドの長さに上限を設定するために使っています。たとえばLという1文字だけを飛行機で飛ばしたい顧客はいないでしょうが、主な目的は1文字以上32文字以下であるか確認することです。

validateLength() は引数を4個となります。文字数の下限と上限、この2つに加えて、検証の対象となる入力フィールド、ヘルプメッセージの表示に使うヘルプテキスト要素です。



```
validateLength(minLength,
  maxLength, inputField,
  helpText);
```

maxLength

フィールドに入力できる
テキストの最大文字数。

メッセージを入力してください : 1字以上32字以下の文字を入力してください。

minLength
フィールドに入力できる
テキストの最小文字数。

inputField
この入力フィールドの
テキストの長さを検証します。

helpText
この要素にヘルプテキストが
表示されます。

```
<input id="message" name="message" type="text" size="32"
  onblur="validateLength(1, 32, this, document.getElementById('message_help'))" />
<span id="message_help" class="help"></span>
```

バナーメッセージの
入力フィールドのオブジェクト。

validateLength() は引数inputFieldの値をチェックして長さがminLengthとmaxLengthの間であることを確認します。値が短すぎたり長すぎたりした場合、ページのhelpText要素にメッセージが表示されます。



重要ポイント

- フォームフィールドはJavaScriptオブジェクトとしてアクセスできます。
 - フォームフィールドオブジェクトの中にはformというプロパティがあり、フォーム全体をフィールドの配列で表現します。
 - onblurイベントはフォームフィールドから入力フォーカスが離れるときに発生します。データ検証関数を起動するのに最適です。
 - アラートボックスはユーザにデータ検証での不具合を知らせる手段としては、あまり魅力がなく気に障ることが多いです。
 - 控えめなアプローチで検証を行なう方がより直感的なのでユーザをいらいらさせません。
 - 文字列のlengthプロパティは文字列の文字数を知らせます。



自分で考えてみよう

`validateLength()`のコードを完成させましょう。渡された引数には細心の注意を払ってください。



validateLength() のコードを完成させましょう。渡された引数には細心の注意を払ってください。

フォーム
フィールドの
値が下限と
上限の範囲内に
あるか確認
します。

フィールドの
長さの問題を
示すヘルプ
メッセージを
設定します。

フィールドの
長さがOKなら
ヘルプテキスト
をクリアします。}

```
function validateLength(minLength, maxLength, inputField, helpText) {  
    // 入力値の文字数がminLength以上でmaxLength以下か調べます。  
    if (inputField.value.length < minLength || inputField.value.length > maxLength) {  
        // データが不正なのでヘルプメッセージを設定します。  
        if (helpText != null)  
            helpText.innerHTML = minLength + "字以上" + maxLength +  
                "字以下の文字を入力してください。";  
        return false;  
    }  
    else {  
        // データがOKなのでヘルプメッセージをクリアします。  
        if (helpText != null)  
            helpText.innerHTML = "";  
        return true;  
    }  
}
```

メッセージの問題が解決

バナーの文字数の問題が解決して、ハワードはホッとしています。長いバナーを買う以外にいい解決策が思いつかないので、JavaScript のレベルで問題を取り組むのがいいとわかりました。そうすれば少なくともユーザは注文する前にバナーに文字数制限があることがわかります。

バナーメッセージが文字数
上限を越えていることを
ヘルプテキストで知らせます。

メッセージを入力してください : 1字以上32字以下の文字を入力してください。

素朴な疑問に答えます

Q: アラートボックスを使ったデータ検証だと、何が良くないのでしょう？ アラートボックスとポップアップ広告の区別がつかない人がほとんど、というわけでもないですよね。

A: たしかにほとんどの人はJavaScriptのアラートボックスとポップアップ広告の違いがわかっています。でもアラートボックスはかなり出しゃばりであることは事実です。有無を言わせずユーザの作業を中断し別のウインドウで何かをクリックさせてるので、ユーザにとても強引な印象を与えます。JavaScriptプログラミングでアラートボックスが効果的なケースはありますか、データ検証に関してはそうではありません。

Q: フォームフィールドのonblurのコードでのthisの使い方がよくわかりません。フォームフィールドがオブジェクトなんですか、それともフォームそのものがオブジェクトなんですか？

A: どちらもオブジェクトです。HTML要素の文脈では、thisはその要素をオブジェクトとして参照します。フォームフィールドの場合、thisはフォームフィールドオブジェクトを参照します。フォームフィールドオブジェクトにはformというプロパティがあり、フォーム全体をオブジェクトとしてアクセスする手段になります。つまりフォームフィールドのonblurコードにあるthis.formは、フォームそのものをオブジェクトとして参照しているのです。Bannerocityのコードにあるthis.formの目的は、特定の入力フィールドに関連付けられたヘルプテキスト要素へのアクセスを提供することです。this.formはformオブジェクトへの参照であり、すべてのフォームフィールドを含んでいる連想配列でもあります。そのためmy_fieldというフィールドにアクセスする場合、配列の記法を使ってthis.form["my_field"]と書くことができます。getElementById()を使うこともできますが、フォームを使うアプローチの方がちょっとだけ簡潔になります。

Q: ヘルプテキスト要素が入力フィールドの中で関連付けられていますが、それぞれのnameとidの意味はどうなるのでしょうか？

A: ヘルプテキスト要素のidは、これに関連付けられた入力フィールドのid/nameをもとにしていますが、必ずしも同じではありません。ヘルプテキストのIDには、入力フィールドのIDに_helpを付加したものを使っています。この命名規則の要点は、入力フィールドとヘルプテキストが表示される要素の間の関連を明確で一貫性のあるものにすることです。ヘルプテキスト要素のIDは他と重複しないように好きなだけ長くできます。このIDが検証関数に渡されます。

Q: データ検証が成功したとき、検証関数でヘルプメッセージをクリアする必要があるのはなぜですか？

A: ヘルプテキストの要点は、問題があるときにユーザを助けることです。フォームフィールドに入力されたデータに問題がなければヘルプメッセージを表示する理由はありません。以前の検証の結果ヘルプメッセージが表示されたまま残っているかもしれないで、データ検証に成功したときは毎回安全のためヘルプメッセージをクリアしています。

Q: 検証関数の引数としてヘルプテキスト要素が渡されなかったらどうなりますか？

A: スクリプトがその要素を探し続けるうちにページが加热してきてブラウザが黒こげになってしまいます。なんてことはありません。Bannerocityの控えめなヘルプシステムは、検証関数でテキストヘルプの引数が使われない場合、静かに消える設計になっています。入力フィールドに対応するヘルプテキストが表示されないだけです。つまりヘルプテキストシステム全体がオプションとして設計されているということです。このアプローチの利点は、ヘルプテキストをどこまで使うか、好きなように制御できる点です。フォームのすべてのフィールドに対してヘルプテキスト要素を追加しなくてもいいのです。検証コードでは、引数htmlTextがnullでないことをチェックしているので、ヘルプテキスト要素はオプション扱いになります。ヘルプテキスト要素がnullでない場合、この要素が存在することを意味するので、ヘルプテキストを表示できます。nullの場合その要素がないので何も表示されません。

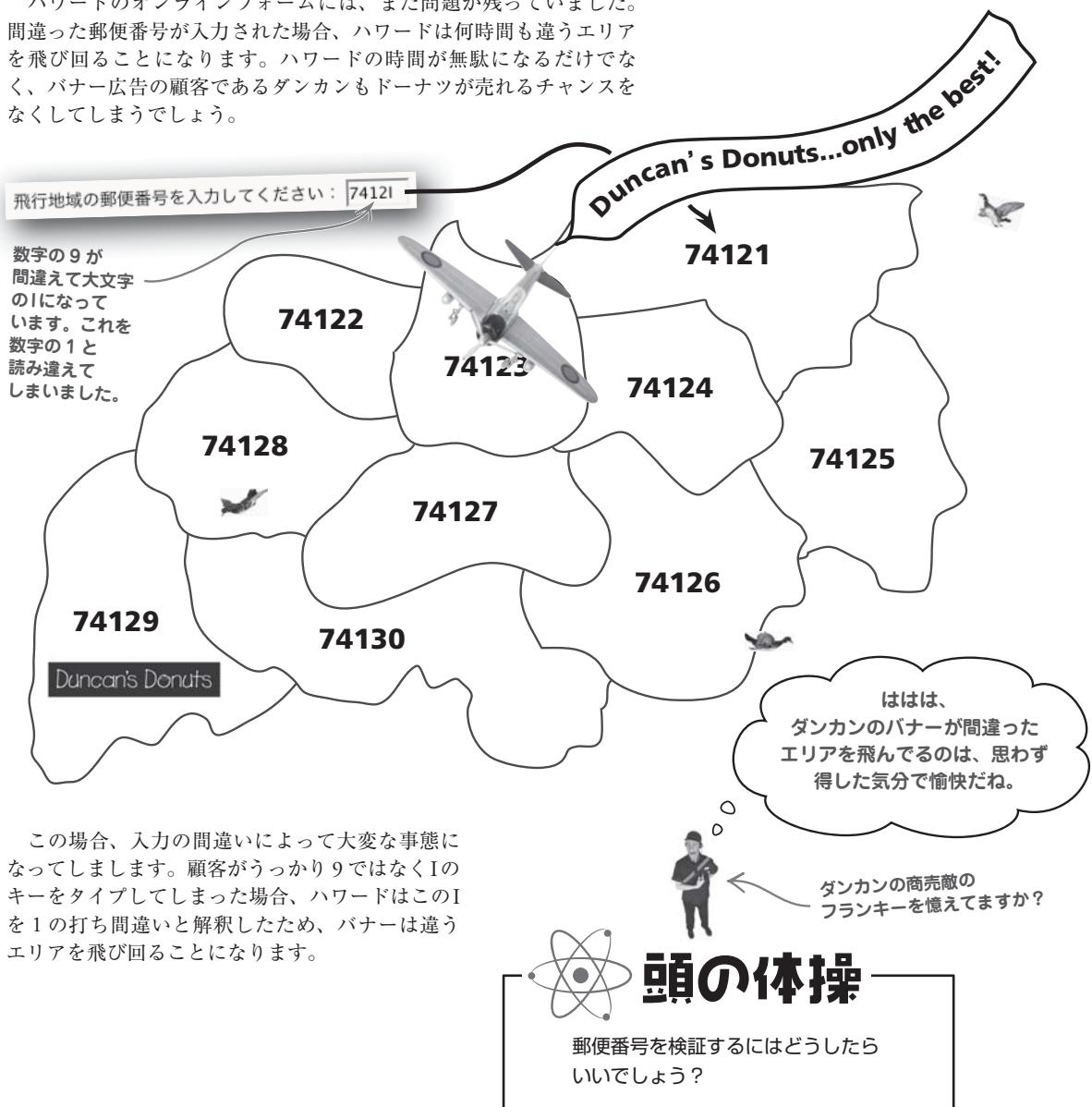


Q: HTMLフォームフィールドにはsize属性があります。これでフィールドの長さを制限できないのですか？

A: HTMLのsize属性はページのフォームフィールドの物理的な大きさを制限するだけであり、入力されるデータの文字数を制限するものではありません。Bannerocityにある郵便番号フィールドのsize属性は5に設定されていますが、これはページ上のフィールドの大きさがおよそ5文字分のテキストになるようにするためのものです。maxlength属性を使えばテキストの実際の文字数の上限を設定できますが、minlengthに相当するものはありません。検証関数を使えば、フィールドに入力されるデータの文字数をできるだけ柔軟に制御することができます。郵便番号の場合、テキストが5文字になっているか確認するだけでなく、5文字がいずれも数字であることも確認するといいでしょう。これはハンドがBannerocityに追加しようと思っている機能かもしれません。

バナーはいいけど、エリアが違う

ハワードのオンラインフォームには、まだ問題が残っていました。間違った郵便番号が入力された場合、ハワードは何時間も違うエリアを飛び回ることになります。ハワードの時間が無駄になるだけでなく、バナー広告の顧客であるダンカンもドーナツが売れるチャンスをなくしてしまうでしょう。



郵便番号を検証する

ハワードがかかえる問題は、入力された郵便番号が正しくないことが原因です。簡単な郵便番号は5桁の数字でできています。そのため郵便番号を検証するには、ユーザが入力したデータが5桁の数字であることを確認すればいいでしょう。

きっちり5桁の
数字



自分で考えてみよう

validateZIPCode() のコードを完成させてください。郵便番号が数字5文字になっているか検証します。

```
function validateZIPCode(inputField, helpText) {
    // まず入力値の長さが5以下であることを調べます

    if ( ..... ) {
        // inputStr を検証してOKか調べます
        if (helpText != null)
            helpText.innerHTML = "5桁の数字を入力してください";

        .....
    }
    // 次に入力値が数値であるか調べます
    else if ( ..... ) {
        // データが不正ならヘルプメッセージを表示し false を返します
        if (helpText != null)
            helpText.innerHTML = "数字を入力してください。";

        .....
    }
    else {
        // データがOKならヘルプメッセージをクリアし true を返します
        if (helpText != null)
            helpText.innerHTML = "";

        .....
    }
}
```



自分で考えてみよう の答え

validateZIPCode()のコードを完成させてください。郵便番号が数字5文字になっているか検証します。

郵便番号の文字列の長さが5以外でないか調べます。

```

function validateZIPCode(inputField, helpText) {
    // まず入力値の長さが5以下であることを調べます

    if ( inputField.value.length != 5 ) {
        // inputStr を検証してOKか調べます
        if (helpText != null)
            helpText.innerHTML = "5桁の数字を入力してください";

        return false; ← 郵便番号の長さが5でなかったのでfalseを返します。
    }
    // 次に入力値が数値であるか調べます ← isNaN()を使って値が数値でないことをチェックします。
    else if ( isNaN(inputField.value) ) {
        // データが不正ならヘルプメッセージを表示しfalseを返します
        if (helpText != null)
            helpText.innerHTML = "数字を入力してください。";

        return false; ← 郵便番号が数値でなかったのでfalseを返します。
    }
    else {
        // データがOKならヘルプメッセージをクリアしtrueを返します
        if (helpText != null)
            helpText.innerHTML = "";

        return true; ← 郵便番号の検証がOKだったのでtrueを返します。
    }
}

```



要注意!

郵便番号は数字だけで構成されているとは限りません。

ウェブのフォームで米国以外の郵便コードを受け取る場合、数字であるか検証するのは問題があるでしょう。国によっては、文字と数字が混在している郵便コードもあるからです。また米国の郵便コードも実際には、#####-####の形式なので、9桁の数字以外にハイフンが含まれています。



Bannerocity の検証関数はきちんとできているけど、
ユーザがヘルプメッセージを無視して間違ったデータのまま
「注文」ボタンをクリックしたらどうなるの?
フォームがサーバに送信されてしまわないかな?

間違ったデータを絶対にサーバに送らない

ユーザが間違いに気づかずにボタンをクリックしてフォームを送信してしまうと、もうデータ検証コードにはどうすることもできません。Bannerocity の致命的な欠陥は、フォームを送信する前にフォームフィールドの検証がされていない点です。このため誤ったフォームデータがサーバに送信されてしまいます。

ユーザはメッセージを無視したまま問題のあるフォームを送信できてしまいます。

Message input field: メッセージを入力してください: [] 1字以上32字以下の文字を入力してください。
 Flight zone input field: 飛行地域の郵便番号を入力してください: [] 値を入力してください。
 Banner delivery date input field: バナー広告を流す日付を入力してください: [] 値を入力してください。
 Customer name input field: お客様の名前を入力してください: [] 値を入力してください。
 Phone number input field: 電話番号を入力してください: [] 値を入力してください。
 Email address input field: メールアドレスを入力してください: [] 値を入力してください。

**Order Banner ボタンで
フォームデータを送信する
前に検証が必要です。**

ほんとうに堅牢な
アプリケーションになると
サーバ上でもデータ検証を行ないます。

Bannerocity に必要なのは、フォームをサーバに送信する前にすべてのフォームフィールドを検証する関数です。「注文」ボタンに placeOrder() を結びつけ、注文を完了する前に検証の最終ラウンドを実行するようにします。

```
<input type="button" value="注文" onclick="placeOrder(this.form); " />
```



placeOrder() の詳細 -

```
function placeOrder(form) {
    if (validateLength(1, 32, form["message"], form["message_help"])) &&
        validateZIPCode(form["zipcode"], form["zipcode_help"]) &&
        validateNonEmpty(form["date"], form["date_help"]) &&
        validateNonEmpty(form["name"], form["name_help"]) &&
        validateNonEmpty(form["phone"], form["phone_help"]) &&
        validateNonEmpty(form["email"], form["email_help"])) {
        // 注文をサーバに送信
        form.submit();
    } else {
        alert("申し訳ありませんが、入力情報に誤りがあります。");
    }
}
```

個々のフォームフィールドに
アクセスできるように、この関数には
formオブジェクトが渡されます。

フォームフィールドとテキスト
要素はどちらもformオブジェクト
から配列要素としてアクセス
できます。

if/else 文の中で各フォーム
フィールドの検証関数を
呼び出します。

フォームフィールドを検証してOK
だったら、submit()を呼び出して
フォームをサーバに送信します。

注文処理の検証で問題があった場合は
アラートボックスで表示するのがいいでしょう。

素朴な疑問に答えます

Q: placeOrder() は、サーバにフォームを送信する
かどうか、どのように制御しているのでしょうか？

All: この関数の中のif/else文はその条件部でフォームの各フィールドが検証されるように構成されています。フォームに無効なデータがあるとelse節が実行されます。else節にはalert()があるだけなので何も送信されません。一方、データがすべて有効だった場合、formオブジェクトのsubmit()が呼ばれます。これはサーバにフォームを送信します。つまり、サーバにフォームが送信するかどうかは、submit()を呼ぶか呼ばないかで制御されています。このメソッドはHTMLのsubmitボタンに相当します。

：データ検証でアラートボックスを使うのは良いな
いと思います。

A: 多くの場合そうなのですが、いくつか例外があります。実際のところ重要なのはページの流れを中断するタイミングです。中断してポップアップ(アラート)を表示しユーザにメッセージを読んでOKをクリックしてもらうことになります。注文ボタンはユーザが注文を送信したいときだけクリックされるので、データに問題があることを知らせる価値があります。このような場合、ユーザにとって重要な問題なので、アラートが適しています。一方、ユーザが訂正するのに役立つ情報は、今まで通り控えめなヘルプメッセージとして表示されます。

データ検証はタイミングがすべて

残念ながらハワードが修正した郵便番号とフォーム送信の検証は、一時的な気休めにしかなりませんでした。まったく新しい問題が生じたのです。郵便番号の検証コードを修正したおかげで、間違ったエリアを飛ぶことはなくなりましたが、もっと悪いことに今度は間違った日にバナーを飛ばしてしまったのです。

入力された日付に間違いがありました。

日付がタイプミスで9ではなく0になっています。9を正しくタイプできる人はいないのでしょうか？

バナー広告を流す日付を入力してください：05/10/2008

ハワードは0をゼロだと解釈して19日ではなく10日に飛行機を飛ばしました。

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
				1	2	3
4	5	6	7	8		10
11	12		14	15	16	
18	19	20	21	22	23	24
25	26			30	31	

今日は月曜日
なのに私のバナーが
飛んでないじゃない！

頭の体操

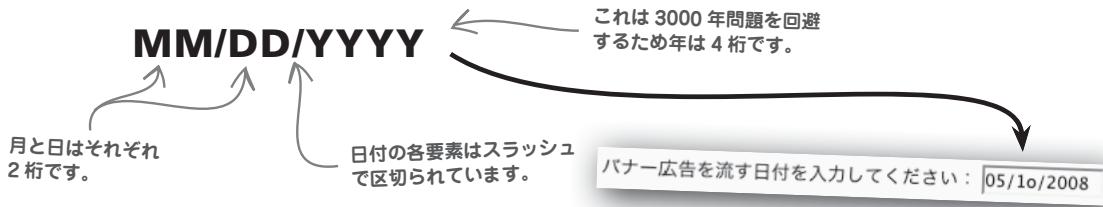
エリーは不満です。

フォームフィールドの日付を検証するとき、データがMM/DD/YYYYのような特定のパターンになっているか確認するにはどうしたらいいでしょう？

Go on a Stick Figure Adventure!

日付を検証する

どうやらハワードはユーザが日付フィールドにデータを入力したか確認するだけで済ませるつもりはないようです。有効な日付が入力されたか実際にチェックしようとしています。日付の検証の鍵になるのは、日付の形式を決めることと、その形式に従っているか判定することです。一般的な日付の形式は、2桁の月、2桁の日、4桁の年で、これらをスラッシュで区切ります。

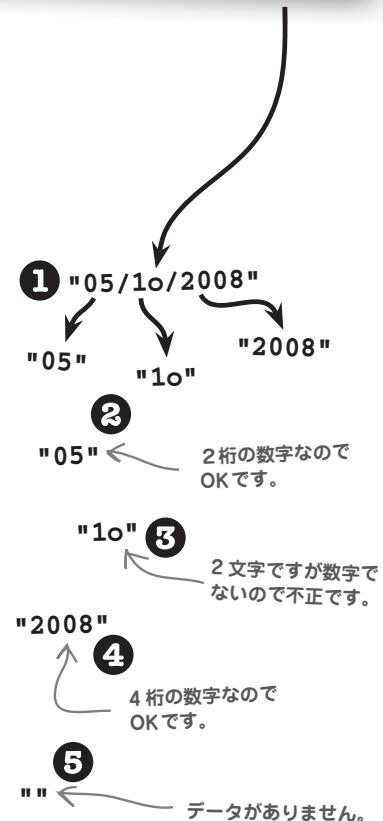


日付の形式を決めるのは簡単ですが、形式に従ってデータを検証するのは厄介です。強力な文字列関数を使えば、文字列をある文字で分解することができます。この場合、スラッシュで分解すればいいでしょう。しかし、分解した個々の断片がある桁数の数字で構成されているか確認する作業はかなり複雑になります。郵便番号の検証をさらに押し進めた検証になります。

データ検証関数の動作を順に追ってみましょう。

- 1 フォームフィールドの値をスラッシュで区切って、文字列の組に分解します。
- 2 月の部分文字列が2桁の数字になっているか確認します。
- 3 日の部分文字列が2桁の数字になっているか確認します。
- 4 年の部分文字列が4桁の数字になっているか確認します。
- 5 年の後に続くデータがあっても無視します。

この一連の手順をコーディングするのは、悪夢というほどではありませんが、かなり簡単なパターンの検証なのに恐ろしくたくさんの作業が必要に思えます。





正規表現は普通の表現ではありません

JavaScriptには正規表現と呼ばれるとても強力なツールが組み込まれています。テキストのパターンマッチングのために特別に設計されたツールです。正規表現を使うと、あるパターンを作成し、それをテキストの文字列に適用して、パターンに一致する部分を探すことができます。ちょうど行列の中から容疑者を探すようなものです。



正規表現はマッチさせるパターンを定義します

容疑者のパターンが、身長、髪型といったその人物の身体の特徴だったのと同様、テキストのパターンは、ある一連の文字、たとえば5桁の数字だけでできた行、といった記述を使います。ちょっと待ってください、これは郵便番号のパターンかもしれません。

英字が含まれています。

桁が少ないです。

"A3492"

"5280"

マッチしました！

このパターンは
5桁の数値を記述
しています。

"37205"

英字が含まれています。

"007JB"

"741265"

桁が多すぎます。

残念ながら、郵便番号に適用する5桁の数字というパターンの正規表現は、直感的に理解できるもではありません。これは正規表現の構文がとてもコンパクトであり、どこか暗号めいた記号を使ってテキストのパターンを記述するためです。5桁の数字に一致させるための正規表現を右に示しますが、すぐに理解するのは難しいでしょう。

ご静粛に。パターンを
調べてみましょう。



文字列は数字で開始
されなければなりません。

パターン = `/^\d{5}$/`

1桁の数字が
5回繰り返されます。

正規表現をスラッシュで
囲みます。

1桁の数字。

文字列は数字で
終わらなければ
なりません。

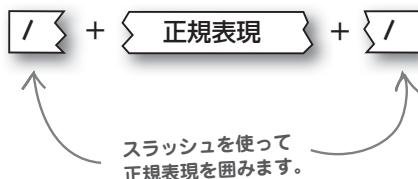


正規表現がちょっと難解に思えて
心配いりません。

いくつか実践的な検証の例をみていくうちに、
だんだんわかるようになります。

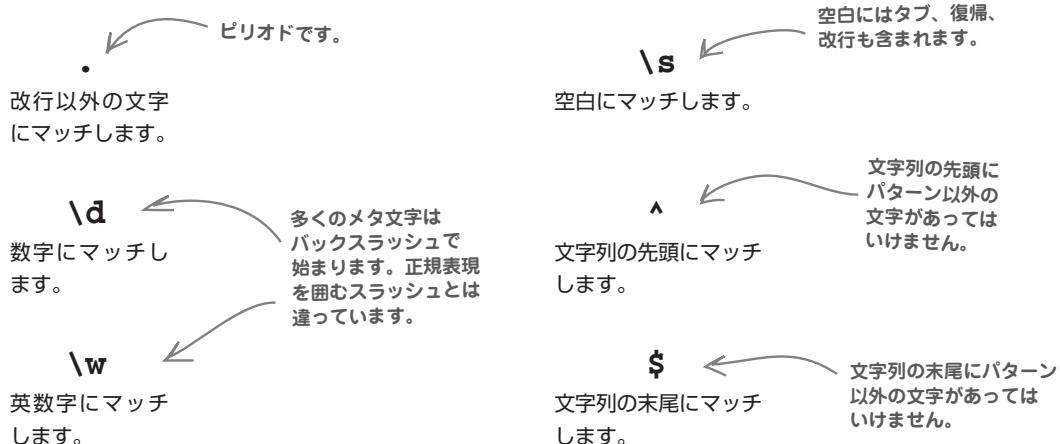
正規表現の真実

正規表現の作り方は文字列リテラルに似ていますが、二重引用符や單一引用符ではなくスラッシュ (/) で囲みます。



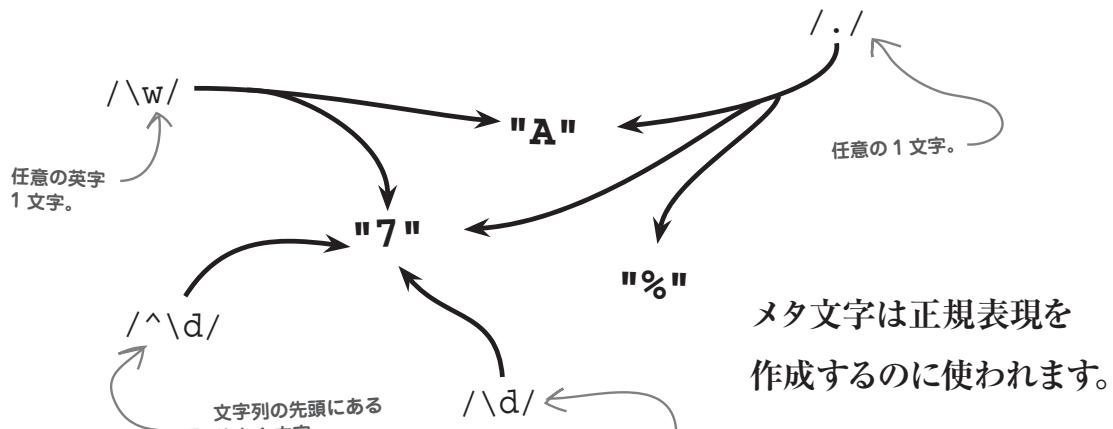
正規表現はかならず
スラッシュで前後を
囲みます。

正規表現の中では、文字や数字と一緒に**メタ文字**と呼ばれる特殊文字の集まりを使って記述性の高いテキストパターンを作成します。実践的な正規表現を作るためには、必ずしも正規表現の言語の詳細を理解しておく必要はありません。一般によく使われる正規表現のメタ文字は以下のとおりです。

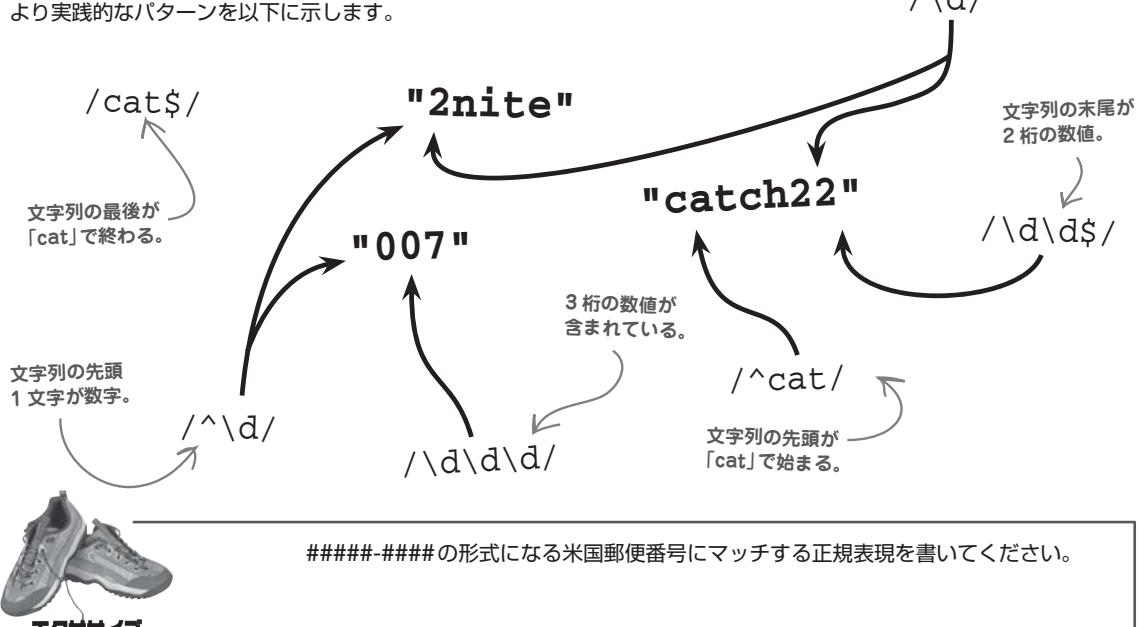


正規表現のメタ文字に関する上の説明は正確ではありませんが、実際にパターンが使われる文脈を調べると理解しやすいでしょう。

メタ文字は正規表現を作るときに使う記号です。



あるひとつの文字に一致する正規表現は何通りものやり方で記述できます。では複数の文字を含む文字列の場合はどうでしょう？より実践的なパターンを以下に示します。




**エクササイズの
答え**

文字列の先頭はこのパターンでなければなりません。

スラッシュで正規表現を開始します。

5桁の数字と一致します。

文字列の最後はこのパターンでなければなりません。

スラッシュで正規表現を閉じます。

4桁の数字と一致します。

このハイフンに特別な意味はありません。
郵便番号の数字を区切るハイフンです。

量化子を使って正規表現を強化する

正規表現では、メタ文字でない文字はその文字通りに一致します。
`/howard/` は文字列中の「howard」に一致します。この他に量化子(quantifier)と呼ばれる記号があり、より洗練されたパターンを作れます。量化子は正規表現の中で量化子の前にある部分パターンに対して適用され、部分パターンが繰り返される回数を制御するのに使います。

***** ←
直前にある部分パターンが0回以上繰り返されている。

部分パターンが0回にも一致します。

{n} ←
直前にある部分パターンがn回繰り返されている。

部分パターンが繰り返される回数を指定します。

+ ←
直前にある部分パターンが1回以上繰り返されている。

この部分パターンは1回以上でないと一致しません。

これは量化子ではありません。
部分パターンをグループにまとめるときに使います。

? ←
直前にある部分パターンが0回か1回繰り返されている。

この部分パターンは0回にも一致します。

() ←
部分パターンの中で文字やメタ文字をグループにします。

パターンの回数を制御する

量化子を使うと、メタ文字だけで書くより簡潔に正規表現を書けます。明示的に部分パターンを繰り返すのではなく、量化子を使うことで、その部分パターンが何回繰り返されるかを指定できます。以下の例は、量化子を使って郵便番号に一致するパターンになります。

`/^\d{5}-\d{4}$/`

{ }を使うと部分パターンを繰り返し記述する必要がありません。

メタ文字と量化子を使うと、文字列に出現するほぼすべてのパターンに一致する強力な正規表現を作ることができます。

英字の繰り返しに一致します。
空の文字列を含みます。

なんらかの文字が 1 回以上現れる。つまり空でない文字列に一致します。

`/(\w*)/`

「Donuts」か「Hot Donuts」に一致します。

誰が何をする？

文字列の最後でパターンが終わります。

\w

部分パターンが 0 回以上繰り返されます。

\$

英数字に一致します。

\d

改行以外の文字に一致します。

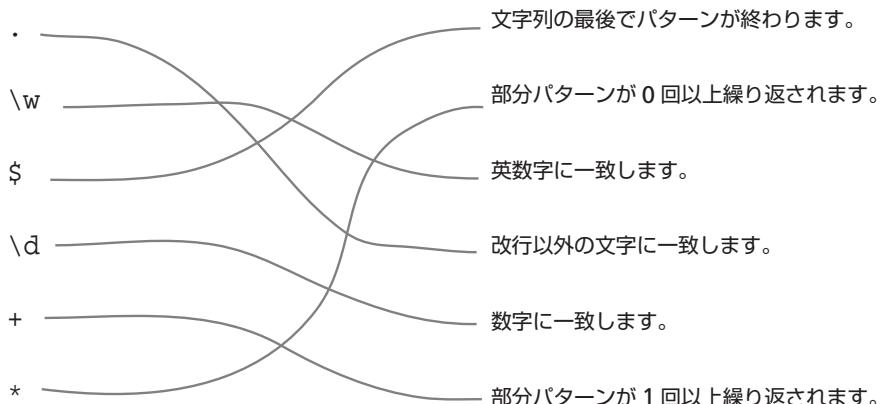
+

数字に一致します。

*

部分パターンが 1 回以上繰り返されます。

誰が何をする？の答え



Q：正規表現は文字列ですか？

A：いいえ。正規表現は文字列に関する記述と考えることができます。少なくとも文字列の一部に関する記述であると言えます。正規表現は文字列と密接な関係があり、文字列に出現するテキストのパターンマッチングに使いますが、それ自体は文字列ではありません。

Q：正規表現を文字列以外のデータに適用することはできますか？

A：いいえ、正規表現はテキスト内の文字列に対してだけパターンマッチングを行うように設計されています。なので文字列だけにしか適用できません。複雑なテキストマッチング処理の場合、文字列しか使えないとは極端に難しくなりますが、正規表現を

素朴な疑問に 答えます

Q： 難なく実行できるので圧倒的に便利です。

Q： ドル記号などのメタ文字そのものに一致させたいときは、どうすればいいでしょう？

A： JavaScriptの文字列と同様、正規表現において特別な意味をもつメタ文字はバックスラッシュでエスケープすることができます。ドル記号に一致させたい場合\\$と書きます。このルールは、^、*、+などの他のメタ文字についても同様です。メタ文字以外の文字はすべて正規表現にそのまま書くことができます。

Q： 正規表現にはデータ検証のための機能がありますか？

A： もうしばらくしたら、お望み通り正規表現を使いますよ。日付やメールアドレスなど、複雑な形式のデータを検証する方法を見つけるために、あえて正規表現への道をすこし遠回りしているのです。Bannerocityにはまだデータ検証すべき項目がたくさん残っているので、正規表現を適用する機会もたくさんあります。もうしばらくお待ちください。

Q： JavaScriptでは正規表現をどのように使いますか？

A： これからその話をしますから。JavaScriptでは正規表現はオブジェクトで表現されます。このオブジェクトでは、正規表現を使って文字列のパターンマッチを行うためのメソッドが提供されています。



パターンの達人

今週のインタビュー：
謎の力をもつ正規表現さんです

Head First：あなたは文字列を調べてパターンを見出すことができるそうですけど、ほんとうですか？

正規表現：暗号破りというのは名ばかりで、文字列をみて、そこからパターンを取り出すことができます。CIAは私みたいな男を使うんでしょうね。お声がかかったことはないですね。

Head First：スパイ行為に关心があるんですか？

正規表現：いいえ、ただテキストからパターンを探するのが好きなんです。パターン探し屋ですよ。あなたが探してるものに関するパラメータを私に教えていただければ、私はそれを探します。少なくとも文字列の中にそれがあるかどうかをお知らせできます。

Head First：それはすごい。でも文字列検索だったら、Stringオブジェクトの`indexOf()`メソッドでもできますよね。

正規表現：これだから素人はパターンのいろはがわかつてないね。たとえば文字列の中にlameって単語があるか探すだけだったら簡単な話で、`indexOf()`で十分ですよ。でも、文字列を分析しようとすると`indexOf()`には本格的な機能がないってすぐにわかるでしょう。

Head First：でも文字列検索ってパターンマッチングのひとつですよね？

正規表現：あのね、郵便ポストまで歩くのも、運動といえば運動ですが、それとオリンピックの競技とは違いますよね。文字列検索はパターンマッチングを極めて単純化したものなんです。固定された語句をパターンにしているだけです。日付やウェブサイトのURLを考えてください。厳格な形式がありますよね。これこそパターンです。探しものの詳細は固定ではないんですね。

Head First：誤解はしてないと思いますよ。パターンとは、文字列に出現するテキストに関する記述で

あって、必ずしもテキストそれ自体とは限らない、と。

正規表現：まさにそう。たとえば、背が高くて、短髪で、眼鏡をかけてない人が通ったら、私に教えてください、と頼んだとします。これはその人に関する記述であって、その人自身ではないですよね。この記述に合致するアランって男が歩いていたら、パターンと一致したと言えるわけです。でもこのパターンに一致する人は他にもたくさんいるはずです。もしパターンがなかつたら、記述をもとに人を探すことはできないでしょう。ある一人の人物を探すしかない。`indexOf()`を使って特定のテキスト断片を探すのと、私を使ってパターンマッチングするのは、アランを探すのか、それとも背が高く、短髪で、眼鏡をかけてない人を探すのか、その違いと同じです。

Head First：ごもっとも。でもパターンマッチングをデータ検証に応用するには、どうすればいいんですか？

正規表現：データ検証は、データがある形式、つまりパターンに一致しているか、確かめることです。私の仕事は、与えられたパターンに文字列が一致しているか確認することです。一致していれば、そのデータは有効だと判断し、そうでなければ無効というわけです。

Head First：それでは、別の種類のデータを対象とする別の種類の正規表現もあるんですか？

正規表現：そうです。そうしたデータ検証こそ、私の出番です。データの形式にうまくあてはまる正規表現を考えていただければね。

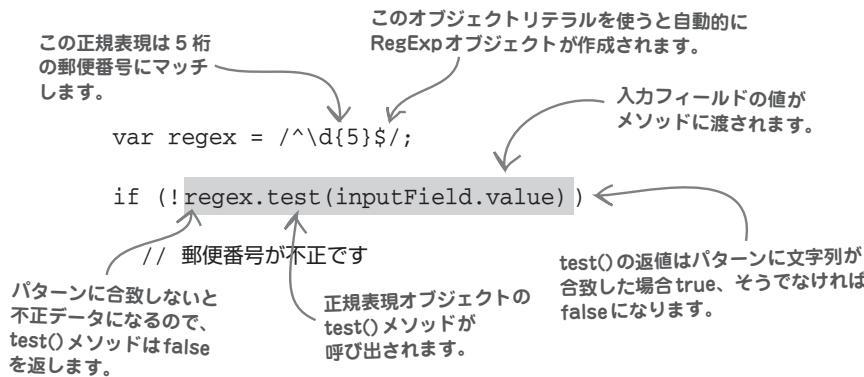
Head First：とても興味深いお話ですね。データ検証であなたが果たす役割を説明していただいて、感謝しています。

正規表現：どういたしまして。私の行動パターンは、自分自身を説明することなんでしょうね。

正規表現を使ってデータ検証する

正規表現は、テキストにあるパターンを探すためだけではなく、Bannerocityの日付フィールドを検証するためにも使うことができます。JavaScriptの正規表現はRegExpオブジェクトを使って表現します。このオブジェクトにはtest()というメソッドがあり、正規表現を使ったデータ検証を実行します。test()は文字列の中に指定したパターンがあるかチェックします。

RegExpオブジェクトの
test()は文字列に対して
正規表現パターンで
テストするときに
使います。



個々の検証関数の中でtest()メソッドを呼び出すこともできますが、正規表現をベースにした汎用的な検証関数を作ることにしましょう。この汎用の検証関数は個々の検証関数から呼び出されます。汎用の関数validateRegEx()は以下の処理を順に実行する必要があります。

- ❶ 引数で渡された正規表現と入力文字列を使って、正規表現のテストを実行する。
- ❷ パターンに一致した場合、引数で渡されたヘルプメッセージをヘルプテキストに設定し、falseを返す。
- ❸ パターンに一致しなかった場合、ヘルプメッセージを消去し、trueを返す。

後はこの関数のコードを書くだけです。この関数はなかなかのものです。このコードのほとんどの部分がすでに他の検証関数に出現しているので、汎用の正規表現データ検証関数であるvalidateRegEx()はコードの再利用の点からも優れています。



```

validateRegEx(regex,
  inputStr, helpText,
  helpMessage);
  
```

```

function validateRegEx(regex, inputStr, helpText, helpMessage) {
    // inputStrを検証してOKか調べます
    if (!regex.test(inputStr)) {
        // データが不正ならヘルプメッセージを表示しfalseを返します
        if (helpText != null)
            helpText.innerHTML = helpMessage;
        return false;
    }
    else {
        // データがOKならヘルプメッセージをクリアしtrueを返します
        if (helpText != null)
            helpText.innerHTML = "";
        return true;
    }
}

```

正規表現、入力文字列、ヘルプメッセージテキスト、ヘルプメッセージ要素を引数で渡します。

正規表現で入力文字列をテストします。

テストでマッチしなかったら、データが不正なのでヘルプメッセージテキストを表示します。

日付をチェックしてOKだったらヘルプメッセージを表示します。

自分で考えてみよう



validateDate()のコードを書いてください。validateNonEmpty()とvalidateRegEx()を呼び出してBannerocityのデータフォームフィールドを検証します。

ヒント：この関数は日付入力フィールドとそれに関連するヘルプテキスト要素を引数として受け取ります。

自分で考えてみよう の答え

validateNonEmpty()を呼び出してフィールドが空でないか調べます。

validateDate()のコードを書いてください。validateNonEmpty()とvalidateRegEx()を呼び出してBannerocityのデータフォームフィールドを検証します。

ヒント：この関数は日付入力フィールドとそれに関連するヘルプテキスト要素を引数として受け取ります。

```
function validateDate(inputField, helpText) {  
    .....  
    // まず入力値にデータが含まれているか調べます  
    if (!validateNonEmpty(inputField, helpText))  
        return false;  
    .....  
    // 次に入力値が日付であるか調べます  
    return validateRegEx(/^\d{2}\/\d{2}\/\d{4}$/, inputField.value, helpText,  
        "日付を入力してください(例: 01/14/1975)");  
}
```

日付の正規表現をvalidateRegEx()に渡します。

日付の正規表現ではメタ文字と量化子を使ってMM/DD/YYYY形式になっているか調べます。

スラッシュは正規表現で特別な意味があるので、バックスラッシュでエスケープする必要があります。



スクリプトが2100年になっても使えるか、議論がわかれるところです。

ユーザが年を2桁の数字で入力できるようにするのはいいアイデアだよね。

Y2100は遠い未来

次の世紀になるまで、まだたっぷり時間があるので、年を4桁ではなく2桁でユーザに入力してもらっても大丈夫でしょう。今日書いたJavaScriptコードが90年以上も問題なく使われ続けることは、まずありません。ハワードはしばらく考えた末、Bannerocityが末永く使えるように年を4桁にすることにしました。次の世紀になるまでこのコードが使われていたら、そのときに修正すればいい、という考えたわけです。



素朴な疑問に答えます

Q: validateDate()の中でvalidateNonEmpty()を呼び出すのはなぜですか？空データだったら正規表現でチェックできますよね。

A: はい、この正規表現は日付が空でないことを検証できます。空でない検証を削除しても、日付の検証は正しく動作します。しかし、最初に空でない検証をしておけばデータ入力時に表示するヘルプメッセージがより詳細になるので、ユーザにとってよりわかりやすくなります。データが入力されなかった場合、入力されたデータが無効であるときは異なるメッセージを表示できます。その結果、控えめだったヘルプシステムがユーザにフォームの記入漏れを指摘できるガイドのように感じられます。ほんのちょっとしたコードを追加するだけで使いやすさが向上するわけです。

Q: 変更なしでずっと使えるスクリプトコードにしたのですが、どうすればいいですか？それとも、こうした発想に問題がありますか？

A: そんなことはありませんよ。将来生じるであろうニーズを見越して、それに対応できるようにコードを書くのは、問題でもなんでもありません。Bannerocityの場合、年のフィールドが4桁なので2桁の場合よりも長持ちします。もっと本格的に作り込むとしたら、ユーザには2桁だけ入力させて画面の裏で世纪に相当する数を先頭に追加することもできます。そうすればフォーム上では2桁なのに実際の日付は年が4桁で格納されます。

繰り返し回数の最小と最大を指定する

量化子{}は、文字列の中で部分パターンが繰り返される回数を指定できますが、この量化子には回数を2個とするバージョンもあり、それぞれ部分パターンが出現する最小回数と最大回数になります。これを使えば、部分パターンの出現回数が簡単に微調整できます。

{ 最小 , 最大 } ←

部分パターンの出現回数は最小と
最大の範囲で指定できます。

直前の部分パターンが最小の回数
以上、最大の回数以下で出現して
いる必要があります。

/^\w{5,8}\$/



多くのパスワードは5文字以上から
8文字以下の英数字になるので、最小/
最大の{}量化子を使うのに最適です。



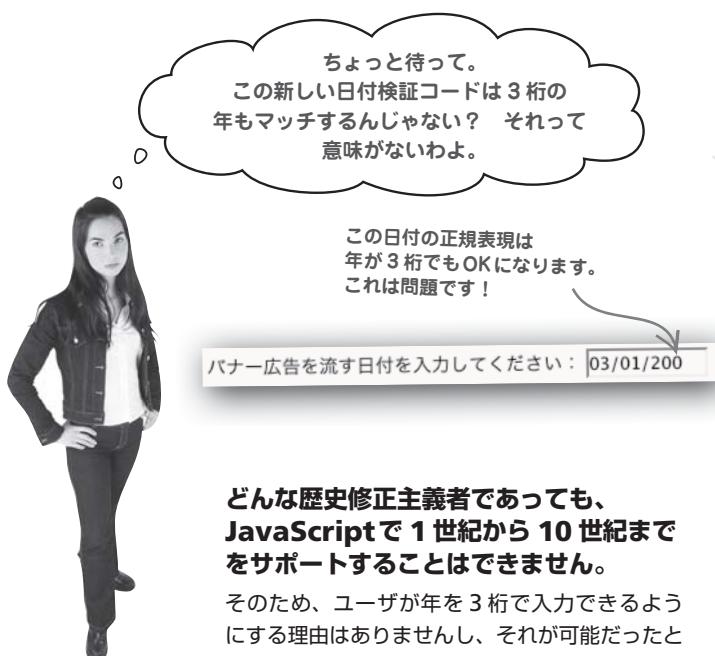
validateDate()で使われている正規表現を書き換えて、2桁の年も4桁の年もどちらもマッチできるようにしてください。

自分で考えてみよう の答え

最小／最大を指定した {} 量化子
を使って日付の年の部分の
出現回数を設定します。

validateDate() で使われている正規表現を書き換えて、2 衝の年も 4 衝の
年もどちらもマッチできるようにしてください。

/^\d{2}\/\d{2}\/\d{2,4}\$/



どんな歴史修正主義者であっても、 JavaScript で 1 世紀から 10 世紀まで をサポートすることはできません。

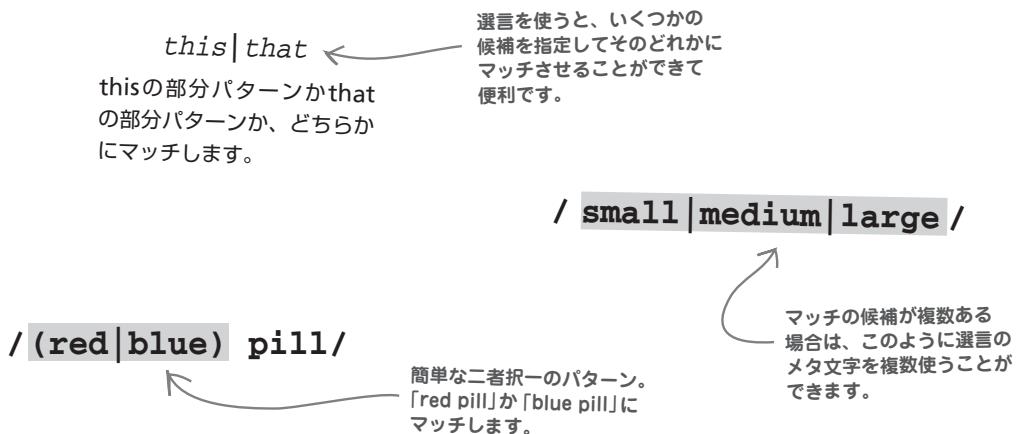
そのため、ユーザが年を 3 衝で入力できるよう
にする理由はありませんし、それが可能だったと
しても、バナーの注文で過去の日付を受け付ける
必要もありません。そのため年が 3 衝の数字の
場合を受け付けないようにする修正は重要です。
そうしておけば、新しい日付入力の問題でハワー
ドが忙殺されることもないでしょう。

重要ポイント

- 正規表現は前後をスラッシュで囲みます。文字列に対してテキストのパターンをマッチさせます。
- 正規表現は通常のテキストの他にメタ文字と量化子で作成されます。量化子を使うとテキストパターンがマッチする回数を制御することができます。
- JavaScript では 正規表現は組み込みオブジェクト RegExp がサポートします。正規表現はリテラルで作成されることが多いので、このオブジェクトはあまり見かけません。
- RegExp オブジェクトの test() メソッドはテキスト文字列に対して正規表現パターンがあるかテストするのに使われます。

これか、あれかのパターンで 3 行数字をなくす

正規表現にはもうひとつ便利なメタ文字があります。選言というメタ文字を使うと、JavaScript の論理演算子 OR とかなり似た働きを実現することができます。JavaScript の OR 演算子と違って、選言では縦棒|を 1 個だけを使います。選言ではサブパターンのリストを指定することができます。つまり、選言のサブパターンのどれかに一致すれば、パターンに一致したとみなされるのです。これは「これか、それともこれか、それともこれか、…」と言っているのと同じなので、論理演算子 OR とかなり似ています。



自分で考えてみよう

`validateDate()` で使われている正規表現をもう一度書き換えてください。今度は 2 行の年か 4 行の年だけにマッチし、それ以外にはマッチしないようにします。

この番号で合ってますか？

自分で考えてみよう の答え

validateDate()で使われている正規表現をもう一度書き換えてください。今度は2桁の年か4桁の年だけにマッチし、それ以外にはマッチしないようにします。

選択のメタ文字(|)で2桁か4桁の年にマッチします。

/^\d{2}|\d{4}\/(\d{2}|\d{4})\$/

他のフィールドも見逃さない

正規表現を使った正確なパターンマッチングによる新しい堅牢なデータ検証をハワードはとても気に入っています。彼は正規表現を使った検証を、Bannerocityの残りの2つのフィールド、電話番号とメールアドレスにも適用しようとしています。

いいねえ。
他のフィールドにも
使おう！

メッセージを入力してください： Mandango... macho movie tickets!

飛行地域の郵便番号を入力してください： 10012

バナー広告を流す日付を入力してください： 03/11/200 日付を入力してください (例：01/14/1975)

お客様の名前を入力してください： [input]

電話番号を入力してください： [input]

メールアドレスを入力してください： [input]

注文

完了

日付のフィールドは正規表現を使ったおかげで日付形式のチェックが厳密にできるようになりました。

Bannerocityの注文フォームにある電話番号とメールアドレスを検証するには、これらのデータ形式に適用できる正規表現を新しく作る必要があります。

電話番号を検証する

電話番号を検証の観点からながめると、厳格な形式に従っているので、正規表現にするのはさほど難しくありません。もちろん、正規表現を使わなければ文字列を分解したものを扱うことになりますが、正規表現を使えば電話番号の検証はいとも簡単に実現できます。米国では電話番号は以下のパターンに従います。



パターン = ####-###-####

このパターンは、ダッシュをスラッシュに置き換え、数字の数をすこし調整すると、日付のパターンに非常に似ていることがわかります。

ハワードは自分のエリア以外を飛行する計画はないので、合衆国の電話番号の形式を想定しておけば安全です。

/^\d{2}\/\d{2}\/\d{2,4}\$/

この日付パターンは MM/DD/YYYY か MM/DD/YY になります。 \d は数字のメタ文字で {} は量化子です。

電話番号のパターンは日付のパターンに似ています。ただし区切り文字はハイフンになります。

/^\d{3}-\d{3}-\d{4}\$/

validatePhone() は、電話番号の正規表現と validateRegEx() のおかげで、かなりわかりやすくなりました。

```
function validatePhone(inputField, helpText) {
    // まず入力値にデータが含まれているか調べます
    if (!validateNonEmpty(inputField, helpText))
        return false;

    // 次に入力値が電話番号か調べます
    return validateRegEx(/^\d{3}-\d{3}-\d{4}$/,
        inputField.value, helpText,
        "電話番号を入力してください (例: 123-456-7890)");
}
```

メールアドレスを検証する

電話番号の検証作業は固まつたので、ハワードは Bannerocity の最後のフォームフィールドであるメールアドレスの検証に挑戦しています。他のデータと同様、メールアドレスの検証に必要なのは、形式を分解して正規表現でモデル化できるパターンにすることです。

2 文字か 3 文字の英数字。

パターン = ローカル名 @ セカンドドメイン . トップドメイン

メールアドレスは、アットマーク (@) とピリオドで区切られた 3 個の英字テキスト断片になります。この見立てはそれほど悪くありません。

英数字

これらの電子メールアドレスはパターンに従っています。これで電子メールのパターンは完璧でしょうか？



このメールのパターンに一致する正規表現は簡単に作れます。

メールアドレスは 1 文字以上の英数字で開始される必要があります。

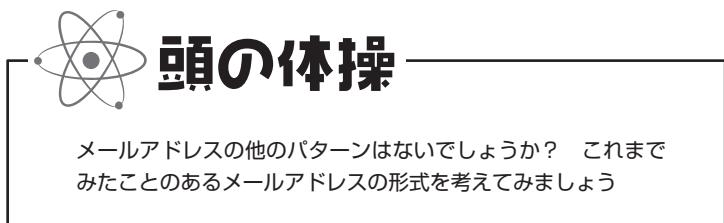
ピリオドは正規表現の中で特別な意味を持つのでエスケープする必要があります。

/ ^ \w+ @ \w+ \. \w{2,3} \$ /

電子メールアドレスは 2 文字か 3 文字の英数字で終わらなければなりません。

このパターンでとりあえず動きますが、見落としがあります。はたしてすべてのメールアドレスがこの形式に従っているでしょうか？

@ の後には 1 文字以上の英数字が続けます。



例外に学ぶ

メールアドレスは、最初にみた形式よりも実際はもっと複雑です。データ検証を行うためのパターンをもっと信頼できるものにするには、メールアドレスの基本形式のいくつかのバリエーションを考慮する必要があります。以下に示すメールアドレスはどれも有効です。



`cube_lovers@youcube.ca`

ローカル名に下線があります。

`rocky@i-rock.mobi`

セカンドドメインにハイフンがあります。
トップドメインが4文字です。

`aviator.howard@bannerocity.com`

ローカル名にピリオドがあります。

`i-love-donuts@duncansdonuts.com`

ローカル名にハイフンがあります。

`seth+jason@mandango.us`

ローカル名にプラスがあります。

`ruby@youcube.com.nz`

トップドメインの上にさらにドメインがあります。

メールアドレスを検証するにはオプション文字とマッチさせる必要があるなあ。



ここに示したメールアドレスからわかるように、いくつかのオプション文字をパターンに含める必要があります。

これまで英数字だけで処理してきましたが、メールアドレスのあちこちで、いくつかのオプション文字が使われています。これらのオプション文字をパターンに組み込む手段が必要です。

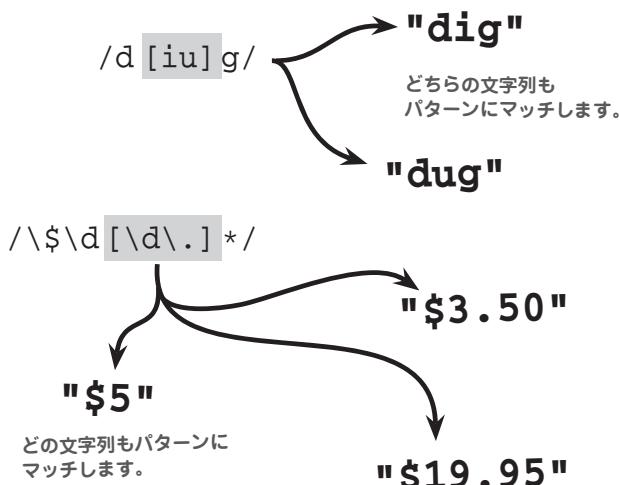
オプション文字を集合にする

正規表現の便利な機能として文字クラスがあります。メールアドレスのパターンにはこの機能が適用できます。きつく制御された部分パターンをパターンの中に作ることができます。オプション文字が部分パターンの中で重要な役割を担う場合には、マッチングルールの確立が簡単に記述できる文字クラスの方が優れています。文字クラスは、ひとつの文字に対するマッチングのルールを集めたものと考えることができます。



文字クラスはある文字にマッチする
文字の集合を表す正規表現です。

文字クラスの中で指定した文字はいずれも文字のマッチングの候補になります。これは選言を使って部分パターンのリストを作ったときの動作に似ています。ただし、文字クラスの後に量化子を続けないかぎり、文字クラスは常に1文字に対してマッチングが行われます。文字クラスの使い方をいくつかの例で紹介します。



文字クラスを使うと正規表現パターンの中でオプション文字が効率よくコントロールできます。

文字クラスは
Bannerocityで必要と
されるメールアドレスの
検証に追加すべき
機能そのものです。



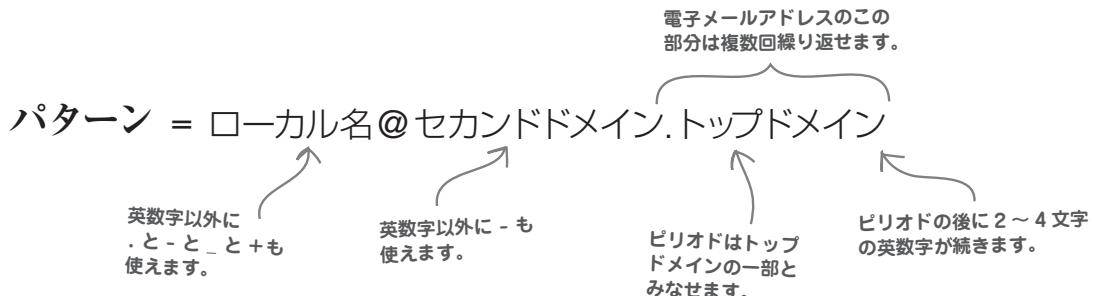
要注意!

正規表現の中の特殊
文字はエスケープする
のを忘れずに。

正規表現の中で特別な意味をもつ文字を実際の文字として使う場合にはその文字をエスケープする必要があります。以下の文字はその前にバックスラッシュ(\)を付けてエスケープします。
[\^\$.!?*+()]

メールアドレスの検証関数を作る

メールアドレスのためのパターンをより堅牢にするために、ローカル名とドメイン名に出現するオプション文字をすべて組み込みます。



パターンを作るアプローチはさまざまであり、メールアドレスのパターンも例外ではありません。ある特定のデータ形式における細かな例外にうまく対応できるパターンを作るのは、驚くほど厳しい作業になります。汎用パターンの設計がうまくいくことはすでに経験したとおりですので、この設計を実際の正規表現で実装してみましょう。

自分で考えてみよう

Bannerocityのメールアドレス検証で使われるvalidateEmail()のコードを完成させてください。

```
function validateEmail(inputField, helpText) {
    // まず入力値にデータが含まれているか調べます
    if (! .....(inputField, helpText))
        return false;

    // 次に入力値が電子メールアドレスか調べます
    return validateRegEx(.....,
        inputField.value, helpText,
        .....);
}
```



Bannerocityのメールアドレス検証で使われるvalidateEmail()のコードを完成させてください。

```
function validateEmail(inputField, helpText) {
    // まず入力値にデータが含まれているか調べます
    if (!validateNonEmpty (inputField, helpText))
        return false;
    // 次に入力値が電子メールアドレスか調べます
    return validateRegEx (/^[\w\.-_+]+@[\\w-]+(\\.\\w{2,4})+$/, inputField.value, helpText,
        "メールアドレスを入力してください(例:johndoe@acme.com)");
}
```

ローカル名には英数字以外に . と - と _ と + が使えます。文字列の先頭でなければなりません。

電子メールを検証するために、これまで学んできたほとんどの正規表現を使って います。

トップドメインは 2 から 4 文字の英数字で終わります。

メールアドレスの検証に問題があればヘルプメッセージを表示して入力フォーマットの例を示します。

Bannerocity の防弾フォーム

ハワードの注文データは、厳しい検証処理の結果、完璧なものになりました。ハワードは、彼にしかできないバナー広告を飛ばすことができて、とても興奮しています。

ハワードは大好きな飛行に戻れて興奮しています。

データ検証はいいことだ!

メッセージを入力してください:	<input type="text" value="Mandango...macho movie ti"/>
飛行地域の郵便番号を入力してください:	<input type="text" value="10012"/>
バナー広告を流す日付を入力してください:	<input type="text" value="03/11/2009"/>
お客様の名前を入力してください:	<input type="text" value="Seth Tinselman"/>
電話番号を入力してください:	<input type="text" value="212 555-5339"/>
メールアドレスを入力してください:	<input type="text" value="setht@mandango"/>

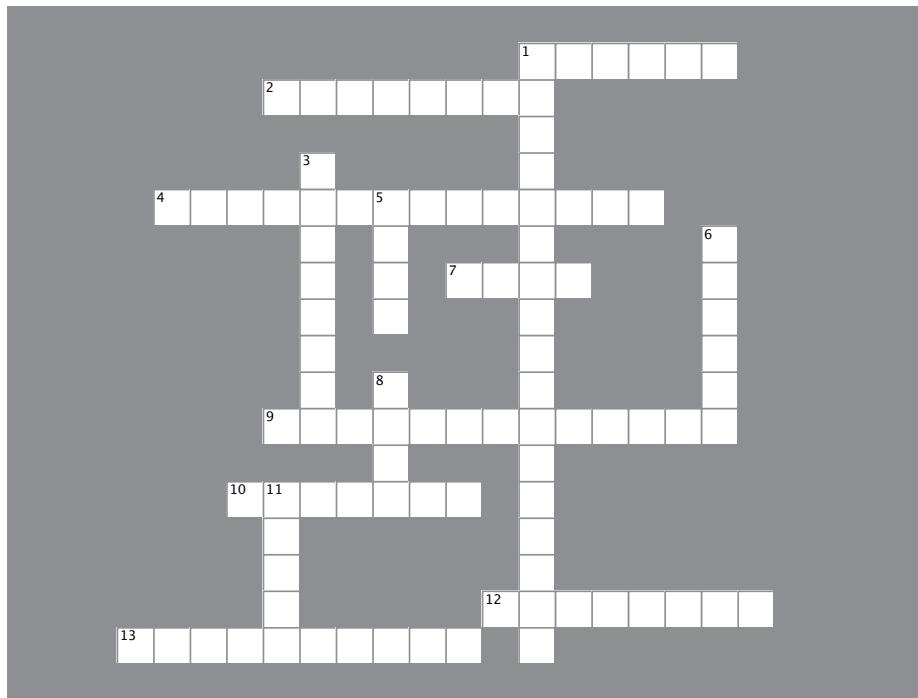
電話番号と電子メールアドレスのフィールドは厳密なデータ形式で検証されるようになりました。

メールアドレスを入力してください (例: johndoe@acme.com)



JavaScriptクロスワード

このパターンは何だと思いますか？ そうです、クロスワードパズルです。
検証は必要ありません、答を探すだけです。



ヨコのカギ

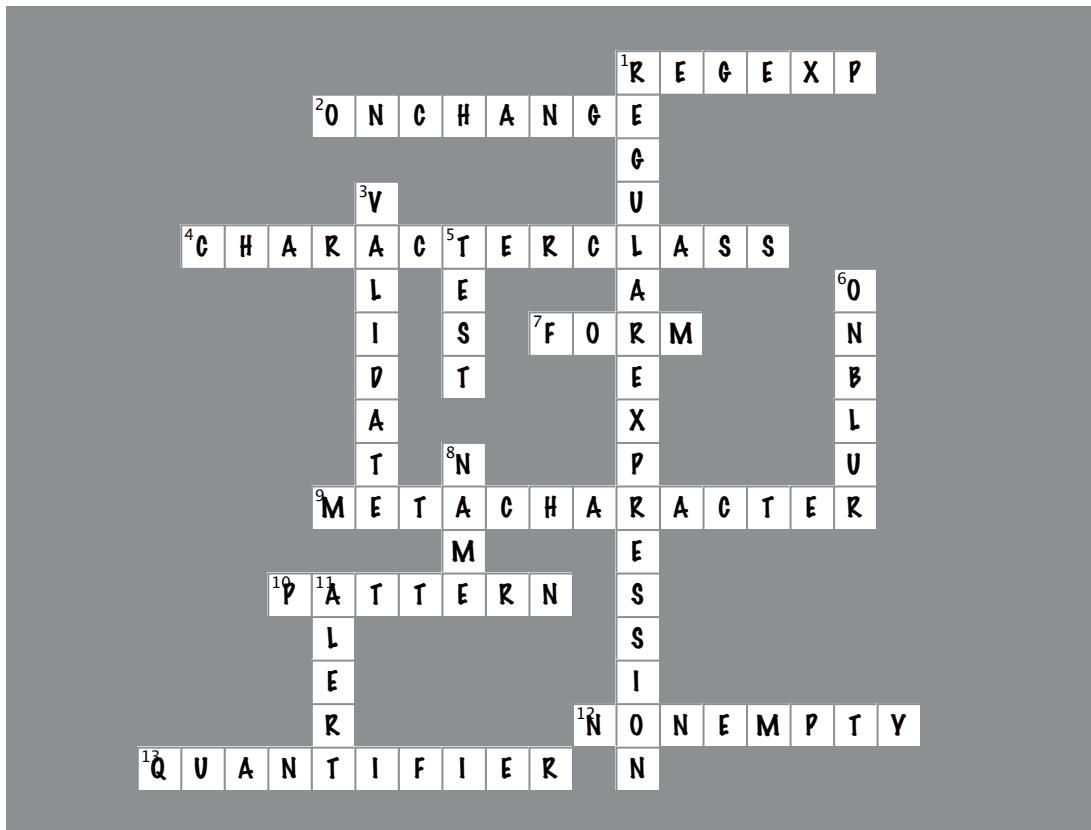
- 正規表現をサポートするJavaScriptオブジェクト。
- フォームフィールドのデータが変更されたとき発生するイベント。
- 正規表現の中でオプション文字を指定する簡単な方法。
- このオブジェクトにはフォームの個々のフィールドがすべて含まれています。
- 正規表現の中の特殊文字。
- データ形式を記述したもの。
- この種類の検証ではフォームフィールドにデータがあるかチェックします。
- これを使って正規表現の中で部分パターンが何回出現するかコントロールします。

タテのカギ

- テキストのパターンにマッチさせるのに使います。
- フォームデータがルールに従っているか確認すること。
- 正規表現を使って文字列をマッチさせるとき使うメソッド。
- ユーザがフォームフィールドを離れたとき発生するイベント。
- フォームのフィールドを一意に特定するHTML属性。
- 妥当でないデータをユーザに通知するのに手軽ではあってもベストではない方法。



JavaScript クロスワードの答え



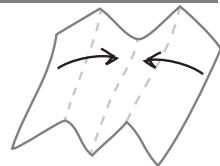
折り畳みページ

脳のところで
折り畳んでから
謎を解決しましょう。

JavaScriptはウェブのフォームに
何をもたらしますか？



左脳と右脳がご対面！



Mandango...the movie seat picker for tough guys!

105012
03/11/200
212-555-5339
setht@mandango

...macho movie seats!

100012
March 11, 2009
(212) 555-5339
seth%t@mandango.us



/^val(ley|ue|krie)/

/name|id\$/

JavaScriptがウェブのフォームにもたらすものはたくさんあるので、
その中からひとつを選ぶのは難しいです。
この答にはおそらくあるレベルのデータが関係しています。
でも具体的にどう関係しているでしょう？

8章 ページの部品をかき集める

HTMLをDOMで切る

いい材料と
私の包丁さばきがあれば、
何でもすぐに料理できるんだけど。
このパイはなかなかうまく
できないのよ。

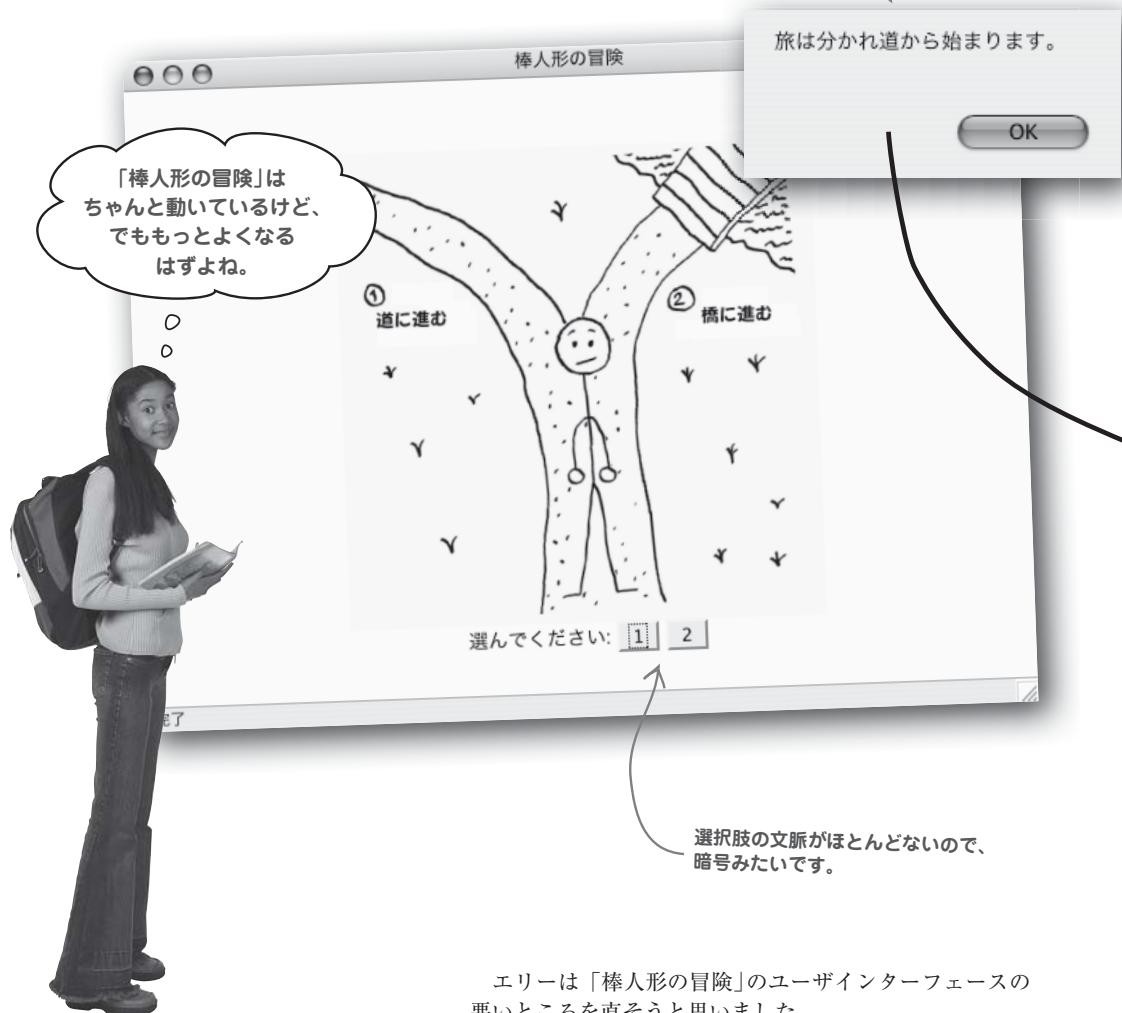


JavaScriptを使ってウェブページのコンテンツを制御するのは、オーブンで焼く料理に近いところがあります。でも食堂がないのはもちろん、残念なことに食べられるご褒美もありません。とはいえウェブページの材料であるHTMLにアクセスして、ここが重要なんですが、ページのレシピを替えることができます。JavaScriptを使えばウェブページの中にあるHTMLコードをあなたの思いのままに操作できるのです。DOM (Document Object Model) と呼ばれる標準オブジェクトの集まりをJavaScriptで操作すれば、ありとあらゆる興味深い操作が可能になります。

機能的だけど退屈なインターフェース

4章で紹介した「棒人形の冒険」のスクリプトはJavaScriptを使ったインタラクティブな意思決定のサンプルでしたが、現代のウェブ標準の基準で評価するとユーザインターフェースがすこし退屈です。アラートボックスでナビゲートしているとだんだん退屈に感じてきますし、選択肢のボタンも1や2のラベルがあるだけなので、ちょっと暗号みたいで直感的ではありません。

アラートはうるさいだけでなく、アプリケーションの流れを中断してしまいます。

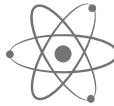


アラートボックスを使わずにシーンを説明する

シーンの説明をアラートボックスを使って表示すると、ユーザがOKをクリックしたときのテキストがどこにも残りません。ページに直接表示できれば、アラートにイライラすることもなく、ウェブページの本体に物語が綴られていきます。エリーは「棒人形の冒険」をこんな風にしたいと思っています。

最新の「棒人形の冒険」のファイルは <http://www.headfirstlabs.com/books/hfjs/> からダウンロードできます。



 **頭の体操**

新しいシーン説明の機能をJavaScriptで実現するには
どうするのが一番いいでしょう？

divを使ってページの余白をつくる

ページにシーンの説明を表示するには、JavaScriptコードについてあれこれ考える前に、まずページ上にHTMLの要素としてエリアを確保する必要があります。シーンの説明は段落として表示されるので、`<div>`タグを使ってエリアを確保することにします。

`<div>`タグにはシーン説明テキストを保持する要素を一意に識別するIDがあります。

```
<body>
  <div style="margin-top:100px; text-align:center">
    <br />
    <div id="scenetext"></div><br />
    選んでください：
    <input type="button" id="decision1" value="1" onclick="changeScene(1)" />
    <input type="button" id="decision2" value="2" onclick="changeScene(2)" />
```

`<div>`タグにid属性があるのはわかった。このIDを使ってシーンの説明にアクセスできるのかな？



IDを使えば、シーン説明の`<div>`はもちろん、ページの要素に正確にアクセスできます。

そうです、JavaScriptコードからページの要素にアクセスするための土台として`<div>`タグのid属性が使えるのです。もうすでにそれを使ってましたよね。



要注意！

ページ上の要素のIDは一意でなければなりません。

id属性の要点はページ上の要素を一意に識別することです。このためひとつのページ内で一意でなければなりません。

HTML要素にアクセスする

標準ドキュメントオブジェクトの`getElementById()`メソッドは、すでにたくさん使ってきました。ページ内のHTML要素に一意のIDがあれば、要素にアクセスすることができます。

```
var sceneDesc = document.getElementById("scenetext");
```

`div`要素には`id`属性を使ってアクセスできます。

HTML要素の`id`属性、この場合`div`の`id`属性を指定します。

```

<div id="scenetext"></div><br />
```

選んでください:

シーン説明の要素が取得できたら、そこに格納された内容の操作に一歩近づきます。先に調べておくといいメソッドがもうひとつあります。`getElementsByName()`は、ページ内の`div`や`img`など、ある特定の要素をすべてつかみます。このメソッドはページにある要素のすべてを含む配列を返します。配列要素の順番はHTMLで出現した順番になります。

```
var divs = document.getElementsByTagName("div");
```

タグの名前を`<>`なしで指定します。



エクササイズ

HTMLのボディ部にあるオレンジ画像にアクセスするJavaScriptコードを書いてください。最初に`getElementById()`を使うコードを、次に`getElementsByTagName()`を使うコードを書いてください。

```
<body>
  <p>ゲーム開始前に、冒険のレベルを選んでください:</p>
  <br />
  <br />
  <br />
  <br />
  
</body>
```

Using `getElementById()`:

Using `getElementsByTagName()`:



エクササイズの

答え

HTMLのボディ部にあるオレンジ画像にアクセスするJavaScriptコードを書いてください。
最初にgetElementById()を使うコードを、次にgetElementsByTagName()を使うコードを書いてください。

```
<body>
  <p> ゲーム開始前に、冒険のレベルを選んでください : </p>
  <br />
  <br />
  <br />
  <br />
  
</body>
```

オレンジ画像は
配列の 4 番目の
要素です。
インデクスは
3 になります。

Using getElementById(): `document.getElementById("orange")`

Using getElementsByTagName(): `document.getElementsByTagName("img")[3]`

HTMLの内部にアクセスする

このHTML要素にアクセスする目的は、要素に格納されたコンテンツを取得するためです。divやpなど、テキストを格納できるHTML要素にアクセスするには、innerHTMLプロパティを使います。

innerHTML
プロパティを使うと、
ある要素に格納された
コンテンツのすべてに
アクセスできます。

あなたは林で
1人で佇んでいます。

HTMLコンテンツは
innerHTMLプロパティ
にも格納されています。

<p id="story">

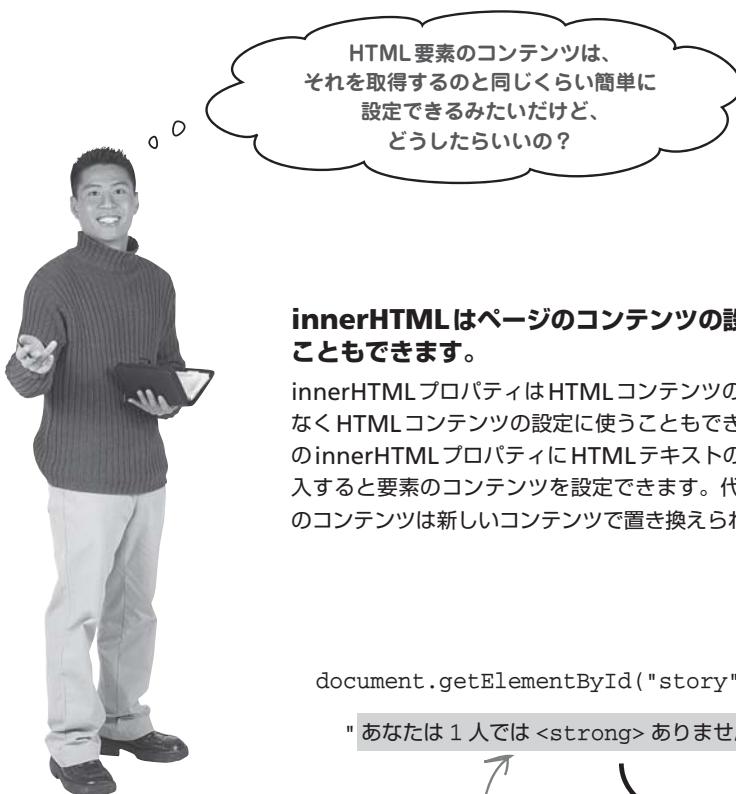
あなたは林で

1人で佇んでいます。

</p>

`document.getElementById("story").innerHTML`

innerHTMLにはHTMLタグを
含むすべてのコンテンツが含まれています。



innerHTMLはページのコンテンツの設定に使うこともできます。

innerHTMLプロパティはHTMLコンテンツの取得だけでなくHTMLコンテンツの設定に使うこともできます。要素のinnerHTMLプロパティにHTMLテキストの文字列を代入すると要素のコンテンツを設定できます。代入前の要素のコンテンツは新しいコンテンツで置き換えられます。

```
document.getElementById("story").innerHTML =
  "あなたは1人では<strong>ありません</strong>！";
```

文字列をinnerHTML
プロパティに代入して要素の
コンテンツを設定します。

あなたは1人では**ありません**！

自分で考えてみよう



ユーザが選んだ決定をもとにシーン説明メッセージがすでに正しく設定されていると想定して、「棒人形の冒険」のシーン説明要素にinnerHTMLを使ってメッセージテキストを設定するコードの行を書いてください。

自分で考えてみよう の答え

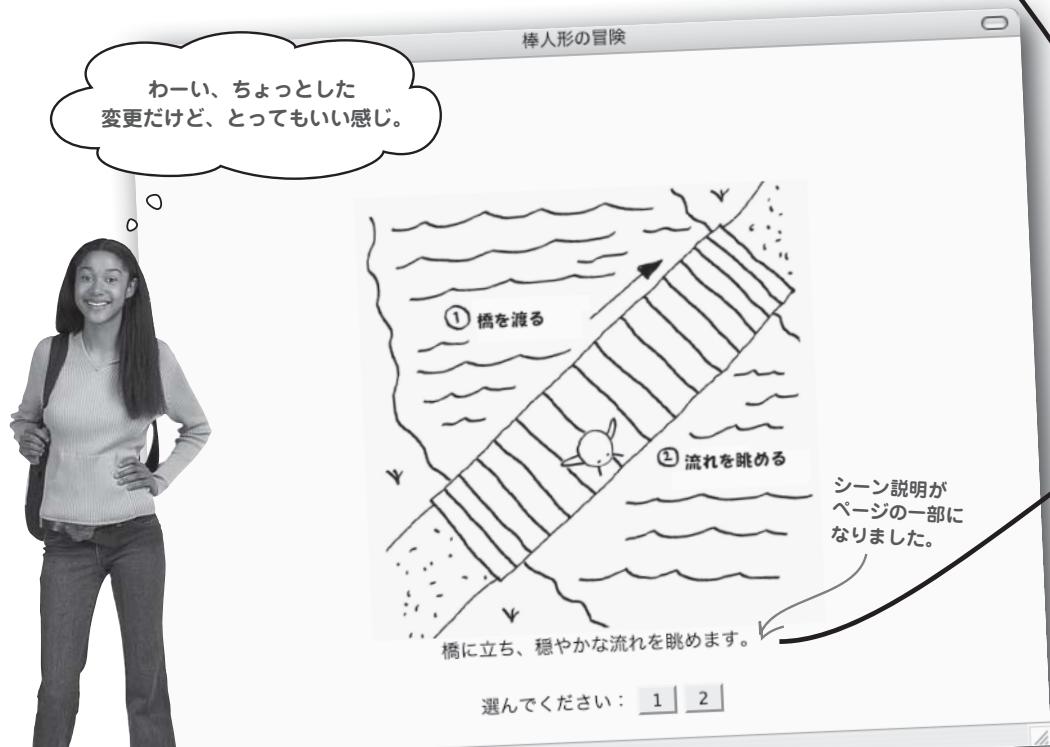
シーンメッセージの
<div>のIDは
“scenetext”です。

ユーザが選んだ決定をもとにシーン説明メッセージがすでに正しく設定されていると想定して、「棒人形の冒険」のシーン説明要素にinnerHTMLを使ってメッセージテキストを設定するコードの行を書いてください。

```
document.getElementById("scenetext").innerHTML = message;
```

アラートによる中断をなくす

シーン説明のエリアが動的に更新されるようになり、うるさいアラートがなくなったので、「棒人形の冒険」のユーザ体験はより滑らかで楽しめるものになりました。



シーン説明（メッセージ）エリア用に<div>を追加し、innerHTMLプロパティを設定するコードを追加したので、あとは「棒人形の冒険」のコードに対する変更は、変数messageを追加し、各シーンごとのmessageを設定するだけです。



「棒人形の冒険」コードの詳細

```

<html>
  <head>
    <title>Stick Figure Adventure</title>

    <script type="text/javascript">
      // 現在のシーンをシーン 0 (イントロ) で初期化
      var curScene = 0;

      function changeScene(decision) {
        // シーンメッセージをクリア
        var message = "";
        switch (curScene) {
          case 0:
            curScene = 1;
            message = "旅は分かれ道から始まります。";
            break;
          case 1:
            if (decision == 1) {
              curScene = 2
              message = "林の中でかわいい小さな家を見つけました。";
            }
            else {
              curScene = 3;
              message = "橋に立ち、穏やかな流れを眺めます。";
            }
            break;
          ...
        }

        // シーンの画像を更新
        document.getElementById("scenemimg").src = "scene" + curScene + ".png";

        // シーンの説明文を更新
        document.getElementById("scenetext").innerHTML = message;
      }
    </script>
  </head>

  <body>
    <div style="margin-top:100px; text-align:center">
      <br />
      <div id="scenetext"></div><br />
      選んでください：
      <input type="button" id="decision1" value="1" onclick="changeScene(1)" />
      <input type="button" id="decision2" value="2" onclick="changeScene(2)" />
    </div>
  </body>
</html>

```

ローカル変数 message を作成して次のシーンの説明メッセージを格納します。

変数 message には各シーンごとに説明テキストが設定されます。

innerHTML プロパティを使ってシーン説明テキストに変数 message を代入します。

素朴な疑問に答えます

Q: getElementById()を使ってページのどの要素でもアクセスできますか？

A: はい、ただし要素のid属性に一意の値が設定されている必要があります。getElementById()を使うときは、このid属性が決定的に重要です。

Q: innerHTMLを使えばどのHTML要素のコンテンツでも設定することができますか？

A: いいえ、要素の「内部HTML」コンテンツを設定するには、その要素がHTMLコンテンツを含むことができる必要があります。実際のところinnerHTMLは、div、span、pなど、コンテンツのコンテナとして機能する要素のコンテンツを設定するためにあります。

Q: innerHTMLを使って要素のコンテンツを設定すると、そのコンテンツに何が起きるのですか？

A: innerHTMLプロパティを設定すると、それ以前のコンテンツは完全に上書きされます。innerHTMLにコンテンツを追加するという概念はありません。これを実現するには、古いコンテンツに新しいコンテンツを追加した結果をinnerHTMLに代入する必要があります。こんな感じです：

```
elem.innerHTML += この文が追加される。
```

ちょっと待って。
innerHTMLはウェブの標準ではないと聞いたんだけど、ほんと？



はい、そうです。でもウェブ標準かどうかって、そんなに気になりますか？

innerHTMLはもともとMicrosoftがInternet Explorerのために作成した独自仕様の機能でしたが、その後、他のブラウザもinnerHTMLを採用したので、ウェブページの要素のコンテンツをすばやく簡単に変更するための非公式な標準になりました。

ただしinnerHTMLがいまだに標準でないのは事実です。このことは大問題ではありませんが、標準が存在する理由は、ウェブページとアプリケーションができるだけ多くのブラウザとプラットフォームで動作できるようにするためにです。innerHTMLと同じ仕事を標準に準拠した方法で実現することもできます。その方法はけして簡単ではないのですが、より柔軟で強力なアプローチです。DOM(Document Object Model)と呼ばれるオブジェクトの集まりを使うアプローチの場合、JavaScriptはDOMを使ってウェブページの構造とコンテンツを完全に制御することができます。

木に林を見る：DOM (Document Object Model)

DOMはスクリプトに適したウェブページの構造とコンテンツのビューを提供します。JavaScriptを使ってページを動的に更新する場合、DOMは重要です。DOMのレンズを通して見るページは、要素の階層が木を形しているように見えます。木のそれぞれの葉はノードであり、ページにあるそれぞれの要素に直接関係しています。木においてあるノードの下に現れた別のノードは、子ノードと見なされます。

奇妙な木に見えますが、
ページのノードが木を組み立てます。

```
<html>
<head></head>

<body>
<p id="story">
    あなたは林で<strong>1人で</strong>佇んでいます。
</p>
</body>
</html>
```

DOMは
ウェブページをノードの
階層木に見立てます。

<p>タグの前後にある空白は
空テキストとして解釈されます。

タグのノードの
下に強調表示される
テキスト "alone" があります。

DOMツリーのトップ
にはDocumentノードが
あり、この下にHTML
要素があります。

これらはすべて
ノードです。

Document

html

body

head

p

strong

"佇んでいます。"

"あなたは林で "

"1人で "

ページはDOMノードの集まり

DOMツリーにあるノードはそのタイプに応じて分類されます。主なノードのタイプはページの構造に対応していて、主に要素ノードとテキストノードで構成されます。

DOCUMENT

DOMツリーのトップノード。
ドキュメント自体を表現し、
html要素の上に位置します。

TEXT

要素のテキストコンテンツ。要素の
下の子ノードとして格納されます。

ELEMENT

HTMLコードのタグに対応する
HTML要素。

ATTRIBUTE

要素の属性。要素ノードからアクセスできますが、
DOMツリーには直接現れません

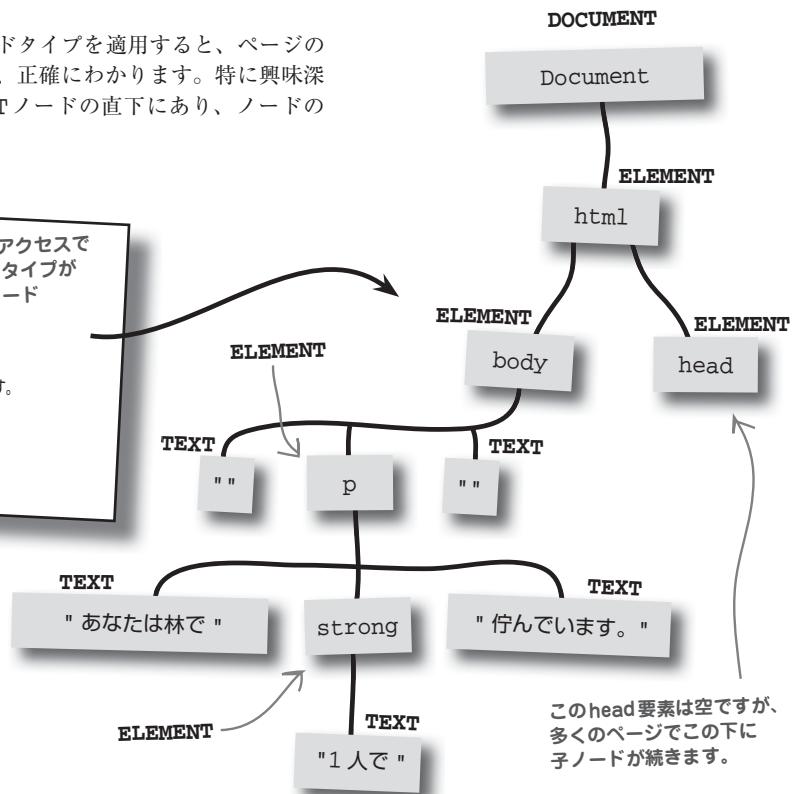
DOMノードは
ノードのタイプに
応じて分類
されます。

ウェブページのDOMツリーにノードタイプを適用すると、ページの各断片がDOMのどこに位置するのか、正確にわかります。特に興味深いのは、TEXTノードは常にELEMENTノードの直下にあり、ノードのコンテンツになっていることです。

```
<html>
  <head></head>
  <body>
    <p id="story">
      あなたは林で<strong>1人で</strong>何んでいます。
    </p>
  </body>
</html>
```

属性にはDOMを使ってアクセスできます。属性にはノードタイプがありますが、ページのノードツリーには現れません。

あなたは林で1人で何んでいます。

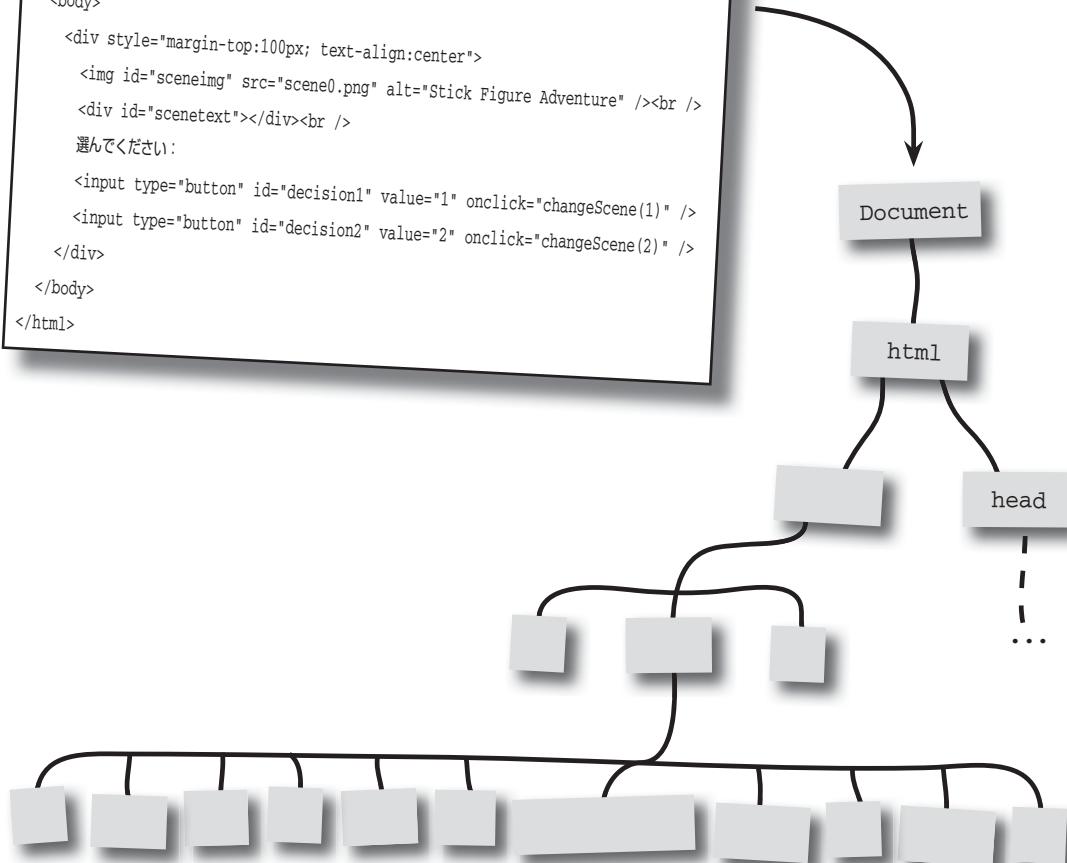


自分で考えてみよう

「棒人形の冒険」のHTMLコードのDOMツリー表現を各ノードの名前を書いて完成させてください。また各ノードのタイプも書いてください。

```
<html>
<head>
...
</head>

<body>
<div style="margin-top:100px; text-align:center">
<br />
<div id="scenetext"></div><br />
選んでください:
<input type="button" id="decision1" value="1" onclick="changeScene(1)" />
<input type="button" id="decision2" value="2" onclick="changeScene(2)" />
</div>
</body>
</html>
```



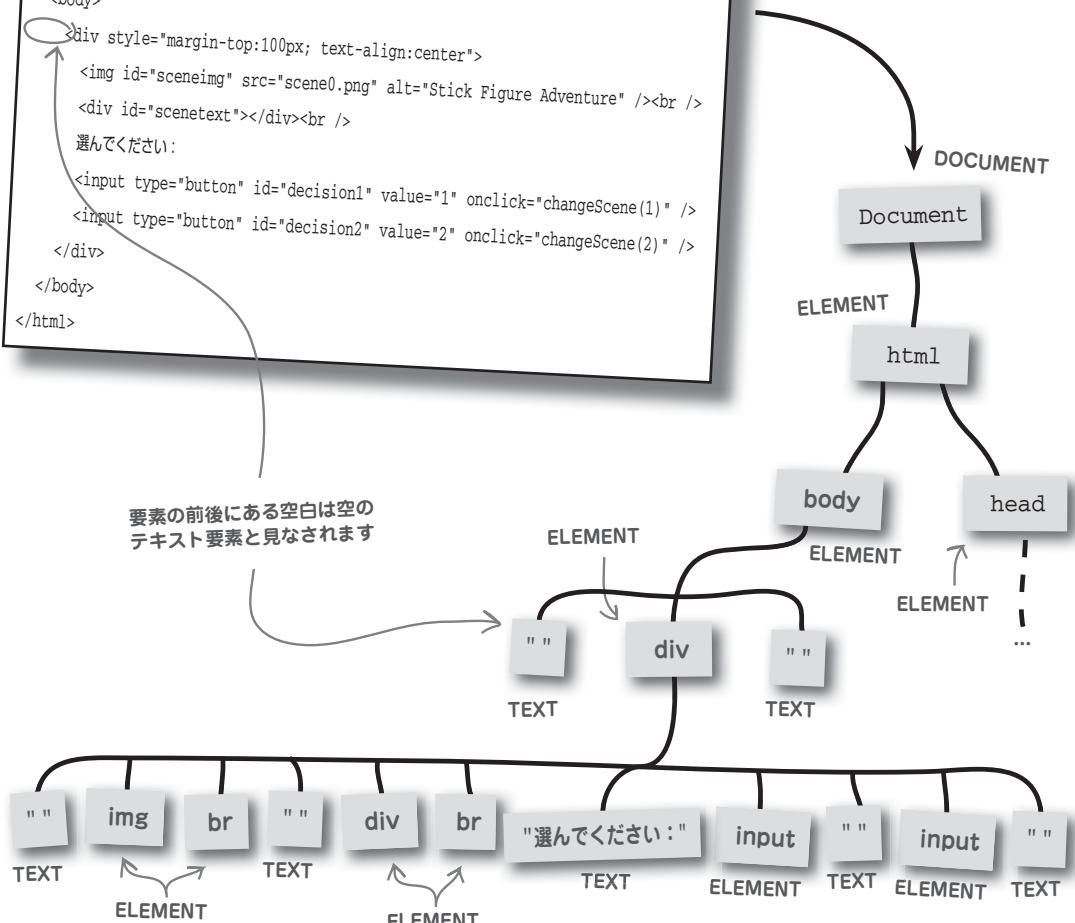
自分で考えてみよう の答え

「棒人形の冒険」のHTMLコードのDOMツリー表現を各ノードの名前を書いて完成させてください。また各ノードのタイプも書いてください。

```
<html>
  <head>
    ...
  </head>

  <body>
    <div style="margin-top:100px; text-align:center">
      <br />
      <div id="scenetext"></div><br />
      選んでください:
      <input type="button" id="decision1" value="1" onclick="changeScene(1)" />
      <input type="button" id="decision2" value="2" onclick="changeScene(2)" />
    </div>
  </body>
</html>
```

要素の前後にある空白は空の
テキスト要素と見なされます



DOMツリーをプロパティでたどる

DOMとのやり取りはドキュメントのノードツリーの最上位にある `document` オブジェクトから始まります。このオブジェクトは `getElementById()` や `getElementsByName()` などの便利なメソッドといくつかのプロパティを提供します。プロパティの多くはツリーの各ノードから利用できます。これらのオブジェクトのいくつかは、他のノードへの移動に使うことができます。つまりノードプロパティはノードツリーのナビゲーションに使えるということです。

`nodeValue`

あるノードに格納されている値。
テキストと属性ノード（要素ではない）だけが持ります。

`childNodes`

あるノードの下にある子ノードをすべて含む配列です。HTMLコードに現れる順番で格納されています。

`nodeType`

ノードのタイプです。DOCUMENTか TEXTですが、数値で表現されます。

`firstChild`

あるノードの下にある最初の子ノード。

`lastChild`

あるノードの下にある最後のノード。

これらのプロパティを使うと、ドキュメントツリーをたどってある特定のノードのデータにアクセスすることができます。たとえば、`getElementById()` でノードにアクセスし、そのノードのプロパティを使って特定のノードを取り出すことができます。

```
alert(document.getElementById("scenetext").nodeValue);
```

nodeValue プロパティを使うと
ノードに格納されたテキストコンテンツに
アクセスできます。

ノードプロパティは
DOMツリーの
ノードをたどるのに
便利です。

nodeValue プロパティには
HTMLの書式を含まない
純粋なテキストが
格納されています。

「棒人形の冒険」のシーン
説明テキストは最初は
空白で始まります。

あまりいいサンプルではありませんが、「棒人形の冒険」でシーン説明テキストの `div` は最初は空になります。物語の進行とともに、読み手を引きつけるテキストが設定され、より賢いコードになります。



以下のコードは 356 ページにあるツリーのノードを参照しています。コードをよく読み、
参照されているノードを丸で囲んでください。

```
document.getElementsByTagName("body")[0].childNodes[1].lastChild
```



以下のコードは 356 ページにあるツリーのノードを参照しています。コードをよく読み、参照されているノードを丸で囲んでください。

エクササイズ 答え

```
document.getElementsByTagName("body")[0].childNodes[1].lastChild
```

<body> タグはひとつだけなので、
getElementsByTagName() が返す配列の最初の要素を参照します。

body 要素の 2 番目の子ノードは
div 要素です。

Document

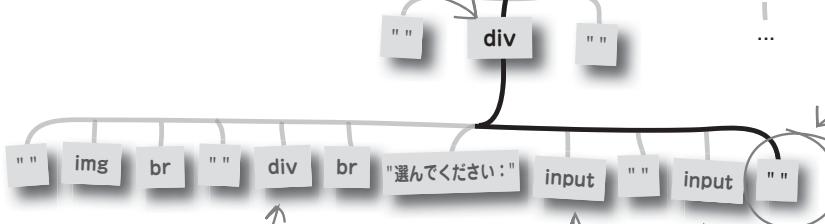
html

body

head

div 要素の最後の
子ノードは
空テキスト要素
です。

...



getElementsById() は特定の ID に
設定されたひとつの要素を取得します。

getElementsByTagName() は
ページ全体からあるタグ名の
要素 (たとえば <input>) を
すべてかき集めます。

素朴な疑問に答えます

Q: getElementById() と getElementsByTagName() の違いは何ですか？ どちらを使つたらいいんでしょう？

A: この 2 つのメソッドの違いは、ひとつの要素を取り出すのか、それとも同様の要素のグループを取り出すのか、そうしたアプローチの違いです。ひとつの要素を取り出すときは迷わず getElementById() ですね。要素の ID を持つて直行しましょう。一方、ノードのグループを対象にするときは getElementsByTagName() の方がいいでしょう。たとえば JavaScript を使ってページの画像をすべて隠

したいときには、まず getElementsByTagName() に "img" を渡して呼び出します。するとページにあるすべてのイメージノードが取得できます。次にそれぞれのイメージ要素のもつ CSS スタイルプロパティ visibility を変更して画像を隠します。おっと、すこし先走ってしまいましたね。CSS は後ほど説明しますので、DOM に話を戻します。ここでは getElementsByTagName() は getElementById() ほどの人気はないけど、ある状況では役に立つということだけ理解しておいてください。

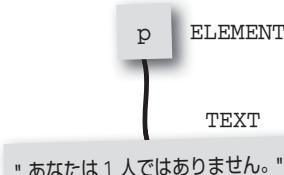


DOMプロパティを使えば、ウェブ標準に準拠したままウェブページのコンテンツを変更できます。

DOMではウェブページにあるすべてがノードになるので、ページの変更是ノードの変更になります。テキストコンテンツの場合、div、span、pなどの要素のテキストはその要素（ノード）の直下にある子ノードとして現れます。テキストが他のHTML要素を含まない単一のテキストノードであれば、そのノードは最初の子ノードになります。

```
document.getElementById("story").firstChild.nodeValue
```

```
<p id="story">  
あなたは1人ではありません。  
</p>
```



あなたは1人ではありません。



頭の体操

DOMを使ってノードのテキストを変更するにはどうすればいいでしょう？

DOMを使ってノードのテキストを変更する

あるノードのテキストコンテンツはそのノードの子ノードに保持されている、そして子ノードはひとつしかない、と前提すれば、`nodeValue`プロパティを使って子ノードに新しいテキストコンテンツを代入することができます。このアプローチは、あくまで子ノードがひとつの場合だけ有効です。

```
document.getElementById("story").firstChild.nodeValue = "OK、多分あなたは1人です。";
```

新しいテキストコンテンツで
子ノードの既存のコンテンツ
を置き換えます。

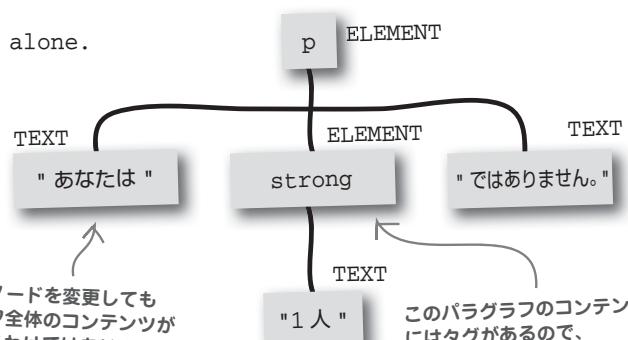


"OK、多分あなたは1人です。"

とはいっても、いつもこの前提が成立するとは限りません。子ノードがひとつでなく複数あったらどうなるでしょう？

```
<p id="story">  
  You are <strong>not</strong> alone.  
</p>
```

このパラグラフは
複数の子ノードに
分解されます。



最初の子ノードを変更しても
パラグラフ全体のコンテンツが
変更されるわけではありません。

最初の子ノードだけ置き換えてしまうと、残りの子ノード
はそのままなので、奇妙な結果になってしまいます。

```
document.getElementById("story").firstChild.nodeValue = "OK、多分あなたは1人です。";
```

最初の子ノードしか変更されて
いないので残りのコンテンツは
そのままです。混乱したメッセージになります。

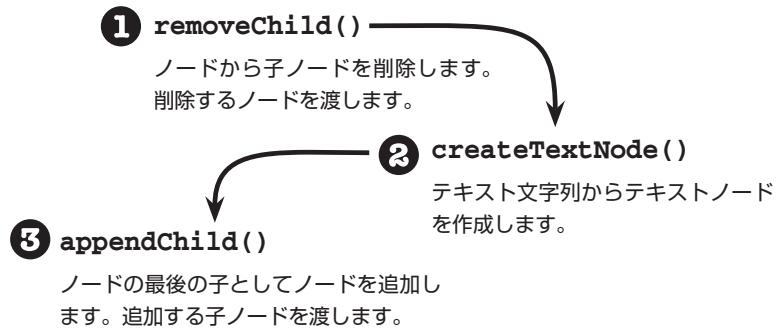
OK、多分あなたは1人です。1人ではありません。

ノードテキストを変更するための安全な手順

ノードのコンテンツを変更するときの問題は、最初の子ノードを変更しただけでは残りの子ノードは変更されない点です。そのためノードのコンテンツを変更するには、いったん子ノードをすべて消去してから新しいコンテンツを含む新しい子ノードを追加する必要があります。

- ① 子ノードをすべて削除する。
- ② 新しいコンテンツをもとに新しいテキストノードを作成する。
- ③ 新規作成したテキストノードを子ノードとして追加する。

この手順は、DOMの3つのメソッドを使って実現できます。



"あなたは1人ではありません。"のテキストコンテンツを変更するには、上記の3手順に従って、まず子ノードをすべて削除し、次に新しいテキストノードを作成し、最後にテキストノードをパラグラフに追加します。

var node = document.getElementById("story");
while (node.firstChild) **1** 子ノードがなくなるまで最初の子ノードを削除します。

```
node.removeChild(node.firstChild);  
node.appendChild(document.createTextNode("OK、多分あなたは1人です。"));  
2 テキストノードを新規作成します。
```

子ノードをすべて削除したら、親ノードにテキストノードを追加します。

3 オーバーレイ: OK、多分あなたは1人です。



DOMの積み木

今週のインタビュー：DOMツリーの知恵についてノードが議論します

Head First：あなたはDOMツリーの中の最小単位、いわばHTMLコンテンツの原子みたいなものだそうですね。

ノード：極小かどうかはわかりませんが、まあそんなもんです。DOMツリーの不連続な情報を表現するのが私です。DOMはウェブページを一口サイズの情報に分解したものだと考えてください。私は那一口サイズの情報というわけです。

Head First：ウェブページをぶつ切りされたデータに分解できることが、どうして重要なんでしょう？

ノード：それはウェブページの情報にアクセスしたり情報を変更するときだけです。多くのスクリプトにとって、まさにこのことが大切なことです。実際のところDOMが重要なのは、ウェブページを小さな断片や部品に分解することができる点です。

Head First：ページをバラバラにしてしまうと、どこかが失われる危険はありませんか？ 分解してはみたものの、そのまま放り出して、結局壊しただけって人が多いですよね。

ノード：いいえ、DOMに関しては問題になりません。ウェブページをノードのツリーに文字通り分解する必要はないんです。実際にウェブデータの形を変えたり一部を削ったりするかどうかに関係なく、DOMはツリーのビューを提供します。

Head First：それなら安心です。でもウェブページの一部を削ったりしたいときは、あなたの出番なんですね？

ノード：そうです。一部を削るだけじゃなく、ウェブデータのツリーに追加することもできますよ。

Head First：おお、素晴らしい。どうやって実現するんですか？

ノード：ページ上のすべての情報は、ツリーにおけるノードとしてモデル化されていることを思い出してください。なので、私を経由することでページ内のどん

な情報にでもアクセスできるのです。私を使って新しいウェブデータを作成し、それをツリーに追加することもできます。DOMはほんとうに変幻自在なんです。

Head First：いいなあ。それでもうひとつ分からるのは、あなたと要素の関係です。ほんとに同一人物なんですか？

ノード：その通りです。でも私が一步先を行ってます。要素は見方をええれば

やといったタグです。ページのすべての要素はドキュメントツリーにおけるノードとして表現できます。その意味では私と要素は同じです。しかし、私は要素の中に格納されたコンテンツを表現することもできます。

の中に格納されたテキストもまたノードでありdivノードの下にそのノードが格納されているのです。

Head First：ちょっと混乱してきました。要素とそのコンテンツの違いを教えてください。

ノード：まず第一に、要素ノードの中に格納されたコンテンツはDOMツリーではそのノードの子ノードになります。第二に、すべてのノードはタイプによって区別できます。要素ノードはELEMENTノードタイプをもち、テキストノードはTEXTノードタイプをもちます。

Head First：つまり、ある要素のテキストコンテンツにアクセスしたければ、TEXTノードタイプを探すだけでいいと？

ノード：そうです。nodeTypeプロパティは実際にはノードタイプを数値で返します。TEXTノードタイプは3、ELEMENTは1になります。でもコンテンツにアクセスするのが目的ですから、要素ノードの子ノードを探せばいいわけ、この情報は必要ないでしょう。

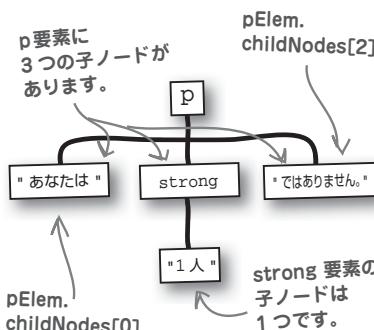
Head First：わかりました。お時間いただき、ありがとうございました。

Node：どういたしまして。ツリーを外科手術したい気分のときは、お立ち寄りください。

素朴な疑問に 答えます

Q: 子ノードがどう編成されているのか、よくわかりません。`childNodes`プロパティはどのように働きをするのですか？

A: ノードの中にデータが含まれる場合、そのノードは親ノードと見なされ、DOMでたどれるデータは子ノードと見なされます。データが生のテキストデータ以外で構成される場合、さらにいくつかの子ノードに分解されます。親ノードの下の子ノードは、親ノードの`childNodes`配列プロパティとして現れます。配列内の順序はHTMLコードに現れた順序と同じです。このため`childNodes`配列の最初の子ノードは`childNods[0]`でアクセスできます。配列をループでたどれば、それぞれの子ノードにアクセスできます。



Q: あるノードからすべての子ノードを削除するコードを書きたいのですが、whileループのテスト条件はどうすればいいですか？

A: ループのテスト条件は以下のようになります：

```
while(node.firstChild)
```

このテストでは、ノードに最初の子ノードが含まれているかチェックしています。最初の子ノードがあれば、whileループの文脈では値が`true`になるので、ループは次の反復を続けます。最初の子ノードがなければ、もう子ノードがないことを意味します。この場合、`node.firstChild`の結果は`null`になり、whileループの文脈では`false`に変換されます。つまり、whileループは最初の子ノードが`null`かどうかを調べています。`null`の場合、もう子ノードが残っていない決定的証拠というわけです。



JavaScriptマグネット

DOM準拠の「棒人形の冒険」で重要なコードのいくつかの断片がなくなりました。下のマグネットを使ってシーンテキスト要素のノードテキストを変更するコードを完成させてください。マグネットは何回でも使えます。

```
// シーンの説明文を更新
var ..... = document.getElementById(".....");
while (.....)
    .....(.....);
    .....(document.createTextNode(.....));
.....;
```

`firstChild`

`appendChild`

`scenetext`

`removeChild`

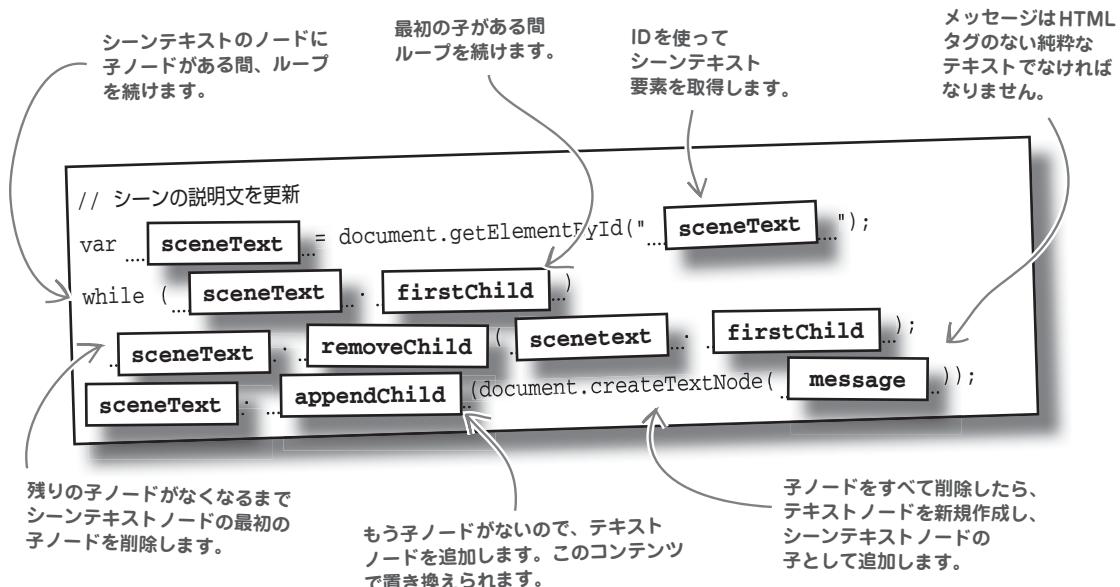
`scenetext`

`message`



JavaScript マグネットの答え

DOM 準拠の「棒人形の冒険」で重要なコードのいくつかの断片がなくなりました。下のマグネットを使ってシーンテキスト要素のノードテキストを変更するコードを完成させてください。マグネットは何回でも使えます。

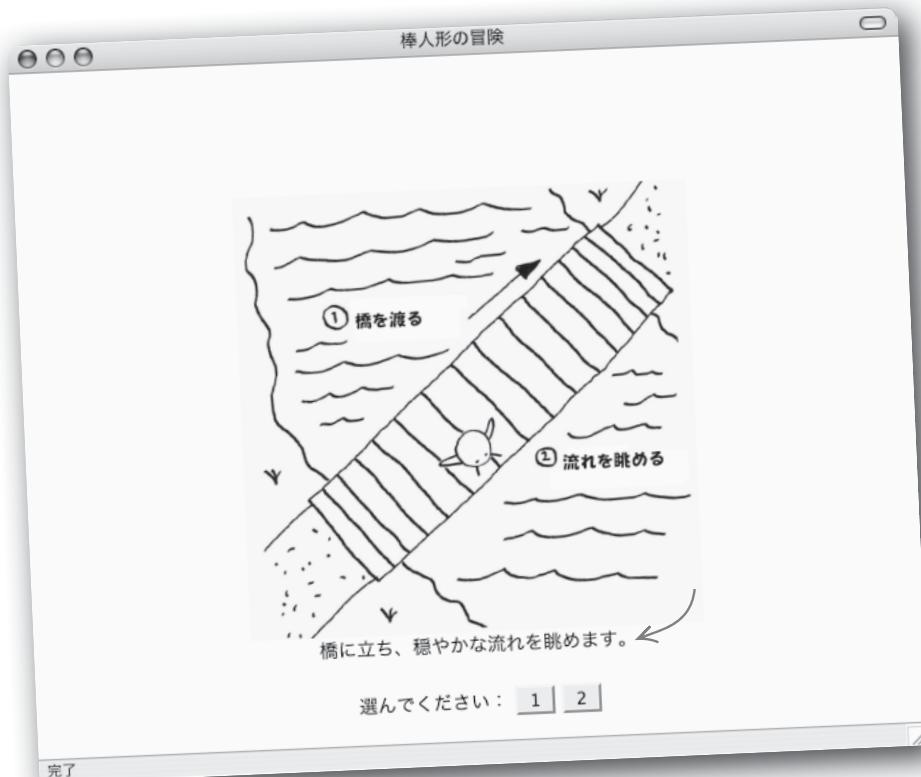


重要ポイント

- innerHTML プロパティはウェブ標準ではありませんが、要素に格納されたコンテンツのすべてにアクセスできます。
- DOM (Document Object Model) はウェブページデータのアクセスと変更に使う標準メカニズムを提供します。
- DOM はウェブページをノードの階層ツリーと見なします。
- The DOM のかわりにウェブページのコンテンツをinnerHTMLを使って変更すると、要素の子ノードがすべて削除されてから、新しいコンテンツを含む新しい子ノードが作成されて追加されます。

「冒険」を標準に準拠させる

なんだか面白そうですね。標準に準拠しているのが良い冒険の印かどうかは別にして、現代のウェブアプリという文脈では、標準に準拠しているのは良いことです。さらに重要なのは、シーン説明テキストをDOMのアプローチによって変えることで、「棒人形の冒険」に劇的な変化がもたらされることです。



ページの外見は変わっていないのですが、シーンの裏側はDOMを使った最新のウェブ標準になっています。JavaScriptコードのすべてが外見に貢献しているわけではないので、「棒人形の冒険」のDOM対応を完成させるには、まだやるべきことがあります。

DOMを使って
HTMLを操作すると
innerHTMLプロパティ
を使うときより細かな
コントロールができます。

よりよい選択肢を探す

「棒人形の冒険」の動的なシーン説明テキストを二度作り直しましたが、暗号のような選択肢ボタンはもとのままで。1と2のどちらかを選んで物語を進行させるのではなく、もっと魅力的で直感的にする必要があります。



選択肢ボタンは数字のままで。
これではユーザが直面する選択の
内容がわかりません。

変更はうまくいったと
思うけど、選択肢ボタンには
がっかりさせられるわ。もっと内容が
わかるボタンにしたらしいのに。



かなりいいですね！ 新しい
選択肢は内容がわかります。

そもそも考えてみると、フォームボタンを使わなければならない理由はありません。テキストを含むことができるHTML要素であれば、うまく動作します。CSSスタイルを使ってこれをドレスアップすれば、入力コントロールのように見せることができます。

頭の体操

「棒人形の冒険」で各シーンにおける選択肢をテキスト表示するにはどう実装しますか？

きれいな選択肢ボタンを設計する

「棒人形の冒険」の新しい選択肢ボタンはテキストを含むHTML要素になったので、各シーンにあるボタンのテキストをDOMを使って動的に置き換えることができます。各シーンには説明と一緒にボタンのテキストが設定されます。changeScene()は、2つの新しい変数decision1とdecision2にボタンのテキストを格納する必要があります。

changeScene()ではシーン1のボタンのテキストを設定してシーン3に移動するようにしています。



```
curScene = 3;  
  
message = "橋に立ち、穏やかな流れを眺めます。";  
  
decision1 = "橋を渡る";  
decision2 = "流れを見つめる";
```

変数のdecision1とdecision2を使って、あるシーンのボタンのテキストを格納します。

自分で考えてみよう



「棒人形の冒険」の選択肢を動的にするには、HTMLコードで選択肢を表現する新しいアプローチが必要です。既存の<input>ボタンを置き換える新しいテキスト要素のコードを書いてください。

ヒント：新しい要素のCSSスタイルクラスの名前は"decision"です。最初の要素のコンテンツは"ゲームスタート"に初期設定されます。

選んでください：

```
<input type="button" id="decision1" value="1" onclick="changeScene(1)" />  
<input type="button" id="decision2" value="2" onclick="changeScene(2)" />
```

動的な選択肢になるコードを書き換えてください。

ゲームスタート

自分で考えてみよう の答え

「棒人形の冒険」の選択肢を動的にするには、HTMLコードで選択肢を表現する新しいアプローチが必要です。既存の<input>ボタンを置き換える新しいテキスト要素のコードを書いてください。

ヒント：新しい要素のCSSスタイルクラスの名前は "decision" です。最初の要素のコンテンツは "ゲームスタート" に初期設定されます。

選んでください： 1 2

```
<input type="button" id="decision1" value="1" onclick="changeScene(1)" />  
<input type="button" id="decision2" value="2" onclick="changeScene(2)" />
```

動的な選択肢になるコードを書き換えてください。

ゲームスタート

```
<span id="decision1" class="decision" onclick="changeScene(1)">ゲームスタート</span>  
<span id="decision2" class="decision" onclick="changeScene(2)"></span>
```

ノードテキストを
置き換えるタスクは関数に
まとめた方が便利そうね。

ノードテキストの置き換えについて再考する

「棒人形の冒険」に新しい動的な決定テキストを導入する上で、まだ要素にテキストを設定するコードが書かれていません。このコードはシーン説明テキストを動的に変更するために書いたDOMのコードとまったく同じ処理を行います。実際には、シーン説明と2つの決定、これら3つの異なる要素に対して同じ作業を実行する必要があります。



関数を使ってノードのテキストを置き換える

汎用のノードテキスト置換関数があると「棒人形の冒険」以外でも便利です。このタイプの関数はすでに作成しているスクリーン説明テキストの置換コードと似ていますが、今回は引数を取るところが違っています。

```
function replaceNodeText(id, newText) {
  ...
}
```

IDで指定されたノードのコンテンツが置き換えられます。

この新しいテキストコンテンツがノードに置かれます。

`replaceNodeText()` は、コンテンツを置換するノードの ID、そしてノードに格納する新しいテキスト、この 2つを引数に取ります。この関数はページにあるテキスト保持可能な要素のコンテンツを変更します。「棒人形の冒険」では、この関数によってスクリーン説明テキストおよび 2つの決定のテキスト、これらを一度に変更できるようになります。

同じコードを3回繰り返す代わりに、関数を3回呼び出します。

シーン説明テキストを新しいメッセージで置き換えます。

2つのボタンのテキストを変更します。

```
replaceNodeText ("scenetext", message);
replaceNodeText ("decision1", decision1);
replaceNodeText ("decision2", decision2);
```



自分で考えてみよう

`replaceNodeText()` のコードを書いてください。ID で参照されるノードの中にあるテキストを置き換える汎用の関数です。`id` と `newText`、この 2つの引数をとります。

.....

.....

.....

.....

.....

.....

自分で考えてみよう の答え

IDを使って要素を
取得します。

```
function replaceNodeText(id, newText) {
```

```
    var node = document.getElementById(id);
```

ノードから子ノード
をすべて
削除します。

```
    while (node.firstChild)
```

```
        node.removeChild(node.firstChild);
```

関数にテキストを
渡して、子の
テキストノードを
新規作成します。

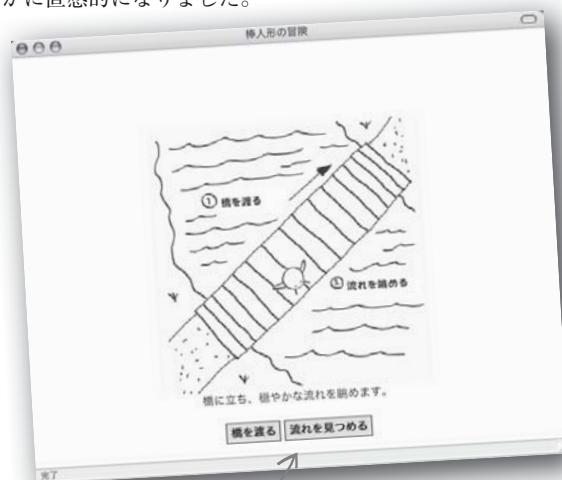
```
    node.appendChild(document.createTextNode(newText));
```

```
}
```

createTextNode()はdocument
オブジェクトで呼び出します。特定の
ノードとの直接的な関係はありません。

選択肢が動的に変わるのは良いことです

「棒人形の冒険」は、暗号めいたボタンを使っていましたときより、
はるかに直感的になりました。



ダイナミックで
わかりやすくて
嬉しいなあ。



ボタンが動的に変わるので、
ユーザはシーンごとの選択肢の
内容がわかります。

素朴な疑問に答えます

Q: 「棒人形の冒険」のボタンは、div要素ではなくspan要素が使われているのは、なぜですか？

A: ボタン要素を横に並べる必要があります。つまり改行して始まるブロック要素は使えないということです。divはブロック要素ですが、spanはインラインです。decisionはインラインにする必要があるので、spanを使います。

Q: createTextNode()を使って新しいノードを作成すると、そのノードはどこに行きますか？

A: どこにも行きません。新しいテキストノードが最初に作成されたとき、それはまだページのDOMツリーのどこにも存在していません。別のノードの子ノードとして追加してはじめてツリーに実際に追加され、その結果ページに追加されるのです。

Q: createTextNode()で作成されたテキストノードのコンテンツはテキストにする必要がありますか？

A: はい。タグ混じりのテキストを代入できるinnerHTMLのようにはDOMは動作しません。DOMが扱う「テキストノード」は、他のタグなどを含まない純粋なテキストです。

選択肢をインタラクティブにする

「棒人形の冒険」の選択肢は動的なテキストになり、以前の暗号めいたものより改善されました。たとえば、選択肢の上にマウスポインターが置かれたとき、クリックできることを知らせるためにハイライト表示するといいでしょう。

ユーザがマウスポインタをボタンの上に置くと、ボタンのテキスト要素がハイライト表示されます。



ハイライト表示にはCSSを使いますが、DOMも直接関係します。

ウェブページのコンテンツをハイライト表示するには、要素の背景色をいじる必要があるので、CSSを使います。しかし、DOMは要素のCSSスタイルにプログラムからアクセスする手段を提供するので、DOMもハイライト表示に関係します。



CSSとDOMでスタイルを変更する

CSSのスタイルはHTML要素に結びついています。DOMは要素(ノード)からスタイルにアクセスするための手段を提供します。DOMを使ってCSSスタイルを調整すると、現在の表示を動的に操作することができます。DOM経由で取得されるCSSスタイルは要素のstyleクラスにあります。これは要素に適用されるスタイルのグループ(クラス)です。

```
<span id="decision1" class="decision" onclick="changeScene(1)">ゲームスタート</span>
```

```
<span id="decision2" class="decision" onclick="changeScene(2)"></span>
```

```
<style type="text/css">
  span.decision {
    font-weight:bold;
    border:thin solid #000000;
    padding:5px;
    background-color:#DDDDDD;
  }
</style>
```

decisionスタイルクラスでボタンのビジュアル効果を定義します。

ゲームスタート

DOMを使ってノードオブジェクトのclassNameプロパティにアクセスすれば、要素のstyleクラスが取得できます。

```
alert(document.getElementById("decision1").className);
```

classNameプロパティを使って要素のスタイルクラスにアクセスします。



ノードのclassNameプロパティを使うとスタイルクラスにアクセスできます。



要注意!

**CSSスタイル
クラスとJava
Scriptのクラス
を混同しないよう
にしましょう。**

CSSスタイルクラスとJavaScriptのクラスは別種の概念です。CSSスタイルクラスはページの要素に適用できるスタイルの集まりです。一方JavaScriptのクラスはJavaScriptオブジェクトを作成するためのテンプレートです。JavaScriptのクラスとオブジェクトについては10章で詳しく説明します。

スタイルクラスを入れ替える

要素の外見をまったく別のスタイルクラスに変更するには、スタイルクラスの名前を別のCSSスタイルクラスに変えるだけです。

```
document.getElementById("decision1").className = "decisioninverse";
```

同じボタンの要素ですが
スタイルクラスを
変えています！

classNameを使って新しい
スタイルクラスがボタン要素に
適用されます。

ゲームスタート

decisioninverseスタイル
クラスはボタンのテキスト
の配色を定義します。

```
<style type="text/css">  
span.decisioninverse {  
    font-weight:bold;  
    font-color:#FFFFFF;  
    border:thin solid #AAAAAA;  
    padding:5px;  
    background-color:#000000;  
}  
</style>
```

classNameプロパティを使って要素のスタイルクラスを変更すると、要素の外見は即座に新しいスタイルクラスに変更されます。この手法を使うと、あまりコードを書かなくても、ページの要素の外見を劇的に変えることができます。



自分で考えてみよう

マウスによるハイライト効果を実現するために、onmouseoverとonmouseout、この2つのマウスイベントを使ってスタイルクラスを変更するコードを「棒人形の冒険」にある要素に追加してください。

ヒント：ハイライトで使うスタイルクラスはdecisionhoverという名前です。

```
<span id="decision1" class="decision" onclick="changeScene(1)">  
.....> ゲームスタート </span>  
<span id="decision2" class="decision" onclick="changeScene(2)">  
.....></span>
```

自分で考えてみよう の答え

decisionhoverスタイル
クラスはonmouseover
イベントに反応して
設定されます。

マウスによるハイライト効果を実現するために、onmouseoverとonmouseout、この2つのマウスイベントを使ってスタイルクラスを変更するコードを「棒人形の冒険」にある要素に追加してください。

ヒント：ハイライトで使うスタイルクラスはdecisionhoverという名前です。

```

<span id="decision1" class="decision" onclick="changeScene(1)">
  onmouseover="this.className = 'decisionhover'" ..... > ゲームスタート </span>
<span id="decision2" class="decision" onclick="changeScene(2)">
  onmouseover="this.className = 'decisionhover'" ..... ></span>

```

マウスポインタが要素の上にあるとき
イベントが発生します。

onmouseoutイベントに反応してハイライトでないスタイルが設定されます。

マウスポインタが要素から外れるとこのイベントが発生します。

クラス対応の選択肢

「棒人形の冒険」にスタイルクラスを適用すると、決定要素の外見を通常とハイライトの2つに切り替えることができます。

```
<style type="text/css">
  span.decision {
    font-weight:bold;
    border:thin solid #000000;
    padding:5px;
    background-color:#DDDDDD;
  }
</style>
```

ノーマル

裏に回る ドアをノック

この2つのスタイルクラスは背景色だけが違います。

```
<style type="text/css">
  span.decisionhover {
    font-weight:bold;
    border:thin solid #000000;
    padding:5px;
    background-color:#EEEEEE;
  }
</style>
```

ハイライト表示

裏に回る ドアをノック



素朴な疑問に答えます

Q: マウスが移動したときのボタンをハイライト表示をCSSを使って実現できませんか？

A: 実現できますよ。こうしたホバーボタンを作成する場合、CSSはJavaScriptよりも広くサポートされているので、CSSを使う方がいいでしょう。モバイル機器でもサポートされています。「棒人形の冒険」はJavaScriptアプリケーションなので、CSSだけでは実現できない多くの機能が可能になります。なので、画面決定ボタンにJavaScriptを使うのはマイナスにはなりません。

スタイル対応の選択肢を テスト運転する

「棒人形の冒険」のユーザインターフェースは、DOMを使って要素のスタイルクラスを動的に変更できるようになったおかげで改善されました。エリーはスクリプトに満足しています。

この新しい
ハイライト効果、
ばっちりね。



マウスポインタがボタンの上にあると、
ボタン要素がハイライト表示されます。



素朴な疑問に 答えます

Q: onmouseoverと
onmouseoutは標準イベントですか？

A: はい。まだ紹介していないJavaScriptの標準イベントはたくさんあります。重要なのは、イベントに反応する方法であって、イベントについて必ずしもすべて知っている必要はありません。この2つのマウスイベントの場合、名前からわかるように、一方は要素の上にマウスポインタがきたとき発生し、もう一方は要素からマウスポイン

タが離れたときに発生することを理解しておけば十分です。

Q: decision要素のスタイルクラスを設定するコードでgetElementById()を使う必要がないのはなぜですか？

A: JavaScriptではすべての要素はオブジェクトです。ある要素のHTMLコードの中では、thisというキーワードを使ってそのオブジェクトにアクセスします。「棒人形の冒険」のコードでは、thisはspan要素に対応するノードオブジェクトを参照します。このオブジェクトはスタイルクラスにアクセス

できるclassNameプロパティを持っています。そのためスタイルクラスを変更する場合、this.classNameを設定するだけですみます。

Q: スタイルクラスはかっこいいんですけど、ひとつのスタイルプロパティだけを変えたいのです。これは可能ですか？

A: とても鋭いですね。エリーが「棒人形の冒険」でなかなか解決できなかった問題が、まさにそれです。そして嬉しいことに、JavaScriptとDOMを使えばスタイルプロパティを個別に操作できるのです。

空の選択肢ボタンが表示されてしまう

この不具合はもともとあったのですが、エリーは空の選択肢に関する不具合を解決することにしました。いくつかのシーンでは、可能な選択肢がひとつしかないのに、このスクリーンショットのように、両方の選択肢の要素が表示されてしまいます。何も情報がない選択肢の要素が表示されると、ユーザはすこし不安になります。

空の選択肢が表示されてしまうシーンがいくつかあって、困ったなあ。何の意味もないし、ユーザを混乱させるだけよね。

棒人形の冒険

Welcome to

どちらかのボタンを押してスタートしてください

ゲームスタート

空の選択肢の要素があると、混乱してしまいます。

完了

頭の体操

空の選択肢が表示される問題が起きるシーンは他にどれがありますか？ この問題をなくすにはどうしたらいいでしょうか？

スタイル調整のアラカルト

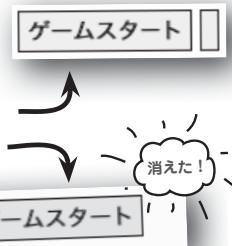
要素のスタイルクラス全体を変更してしまうと、やりすぎになることがあります。もっと細かな変更が要求されるときのために、styleオブジェクトがあります。styleオブジェクトは、ノードオブジェクトのプロパティであり、個々のスタイルをプロパティとしてアクセスするための手段を提供します。visibilityスタイルプロパティは、要素の表示と非表示の切り替えに使うことができます。「棒人形の冒険」のHTMLでは、以下のコードを使って2番目のオプション要素を非表示で初期化することができます。

```
<span id="decision2" class="decision" onclick="changeScene(2)">
  onmouseover="this.className = 'decisionhover'"
  onmouseout="this.className = 'decision'"
  style="visibility:hidden"></span>
```

それ以後は、このvisibilityスタイルプロパティをvisibleとhiddenのどちらに設定するだけで、要素の表示と非表示が切り替えられます。

```
document.getElementById("decision2").style.visibility = "visible";
document.getElementById("decision2").style.visibility = "hidden";
```

あるノードのstyle
プロパティを使って
個々のstyleプロパティ
にアクセスします。



自分で考えてみよう

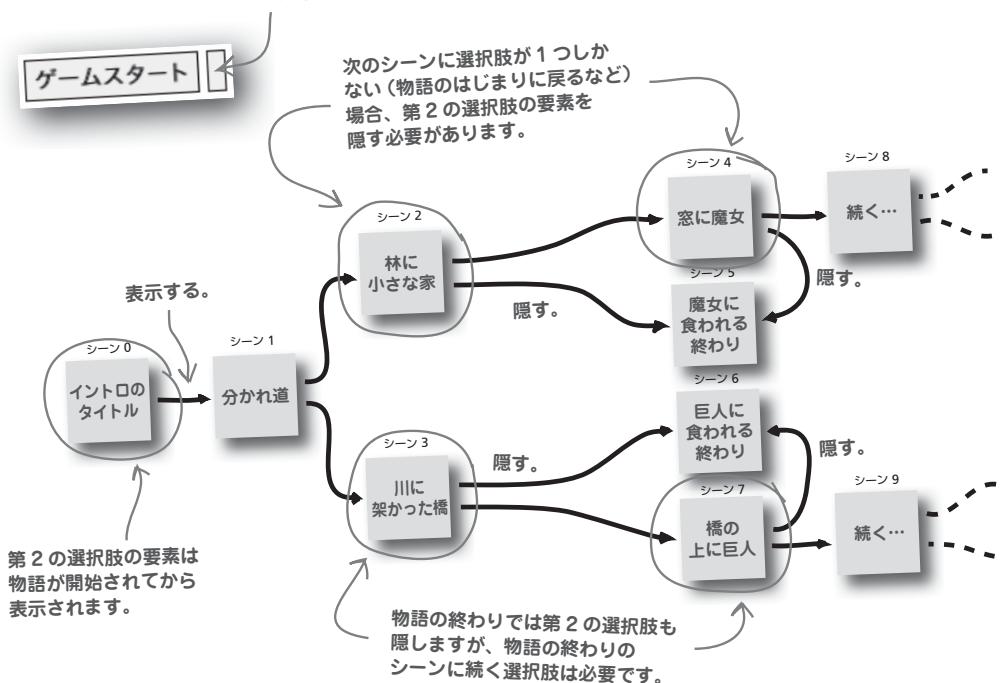
2番目の選択肢。

「棒人形の冒険」のいくつかのシーンでは、新しいシーンに変更するとき2番目の選択肢要素を非表示にする必要があります。該当するシーンを丸で囲み、どの選択肢を隠せばいいか記入してください。

自分で考えてみよう の答え

「棒人形の冒険」のいくつかのシーンでは、新しいシーンに変更するとき 2 番目の選択肢要素を非表示にする必要があります。該当するシーンを丸で囲み、どの選択肢を隠せばいいか記入してください。

2 番目の選択肢。



各シーンの第 2 の選択肢は style オブジェクトの visibility プロパティを使って表示するかしないかを制御します。

```
...
case 7:
    if (decision == 1) {
        curScene = 6;
        message = 残念ながら、あなたは巨人のおいしいお屋ご飯になりました。;
        decision1 = Start Over;
        decision2 = "";
    }
    // 第 2 の選択肢を隠します
    document.getElementById(decision2).style.visibility = hidden;
}
else {
    curScene = 9;
    decision1 = ?;
    decision2 = ?;
}
break;
...
```



重要ポイント

- ノードの className プロパティを変えると、そのノードのスタイルクラスが変わるので、大きなスタイルの変更に適しています。
 - ノードの style プロパティを変えると、そのノードの個々の style プロパティが変わるので、ちょっとしたスタイルの変更に適しています。
 - CSS スタイルクラスは JavaScript のクラスとは何の関係もありません。この 2 つはまったく別のものです。
 - ページの要素はその要素オブジェクトの visibility プロパティによって表示と非表示を動的に切り替えることができます。
- ↑
- display プロパティを使って表示/非表示を切り替えることもできます。display:none (非表示) または display: block (表示) にします。

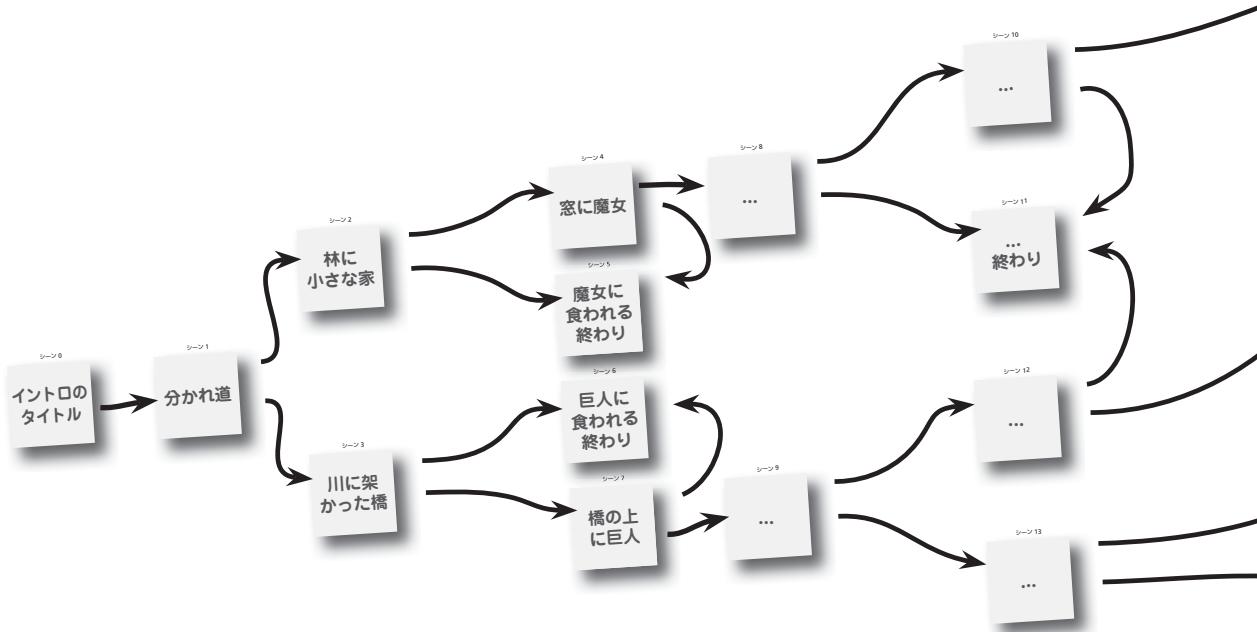
空の選択肢はもう表示しない

DOM を使えば個々のスタイルを操作できるので、2 番目の選択肢の表示と非表示を切り替えることができます。その結果、空だった選択肢が消え、ユーザインターフェースがさらに改善されました。



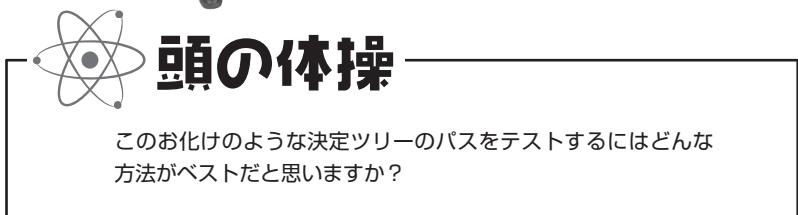
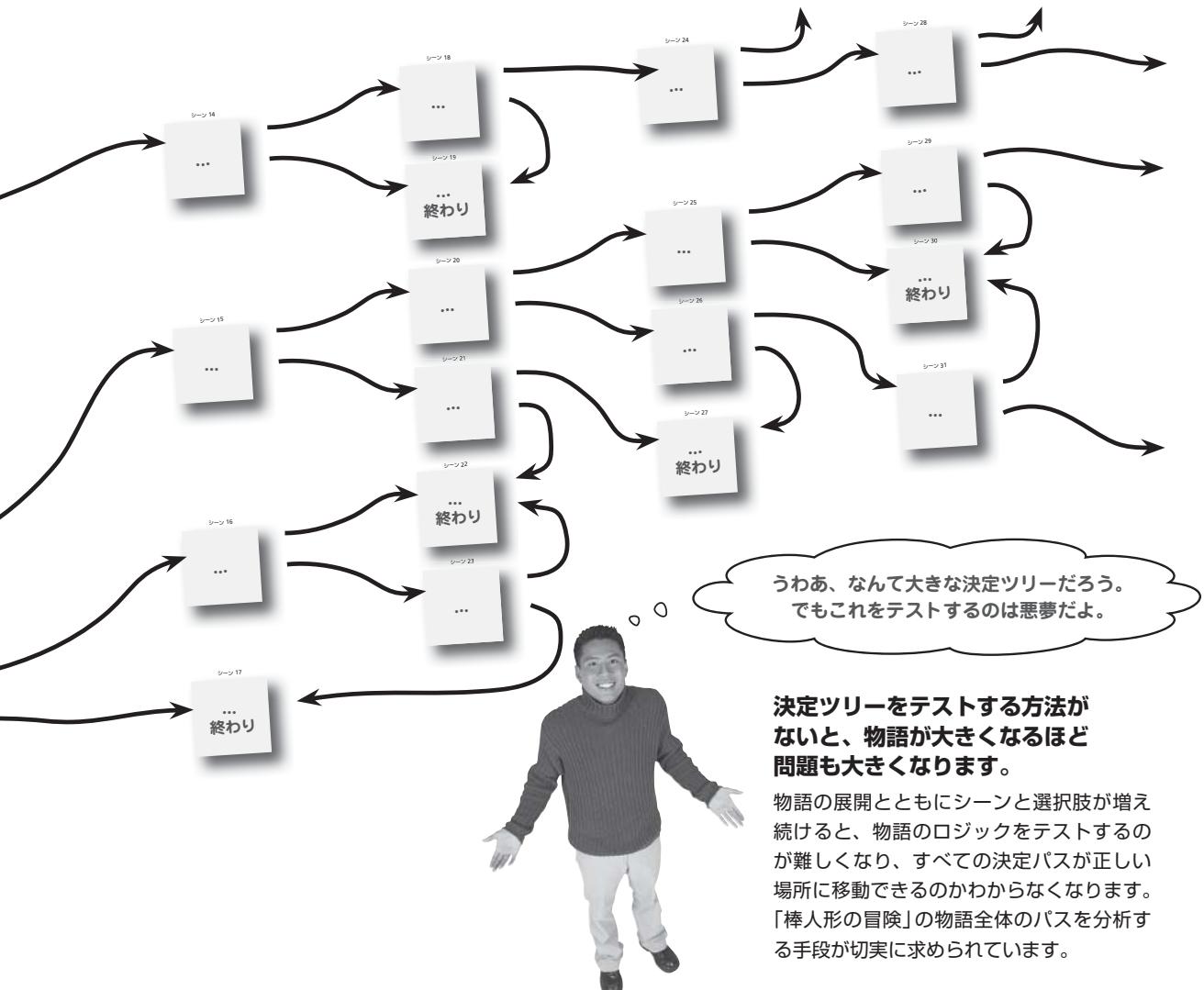
選択肢を増やして、より複雑にする

エリーの頭の中で「棒人形の冒険」のストーリー展開がどんどん膨らんで、新しいシーンと選択肢が浮かんできました。DOMを使えば複雑な構成を支えられるので、「棒人形の冒険」の物語をもっと掘り下げることができます。



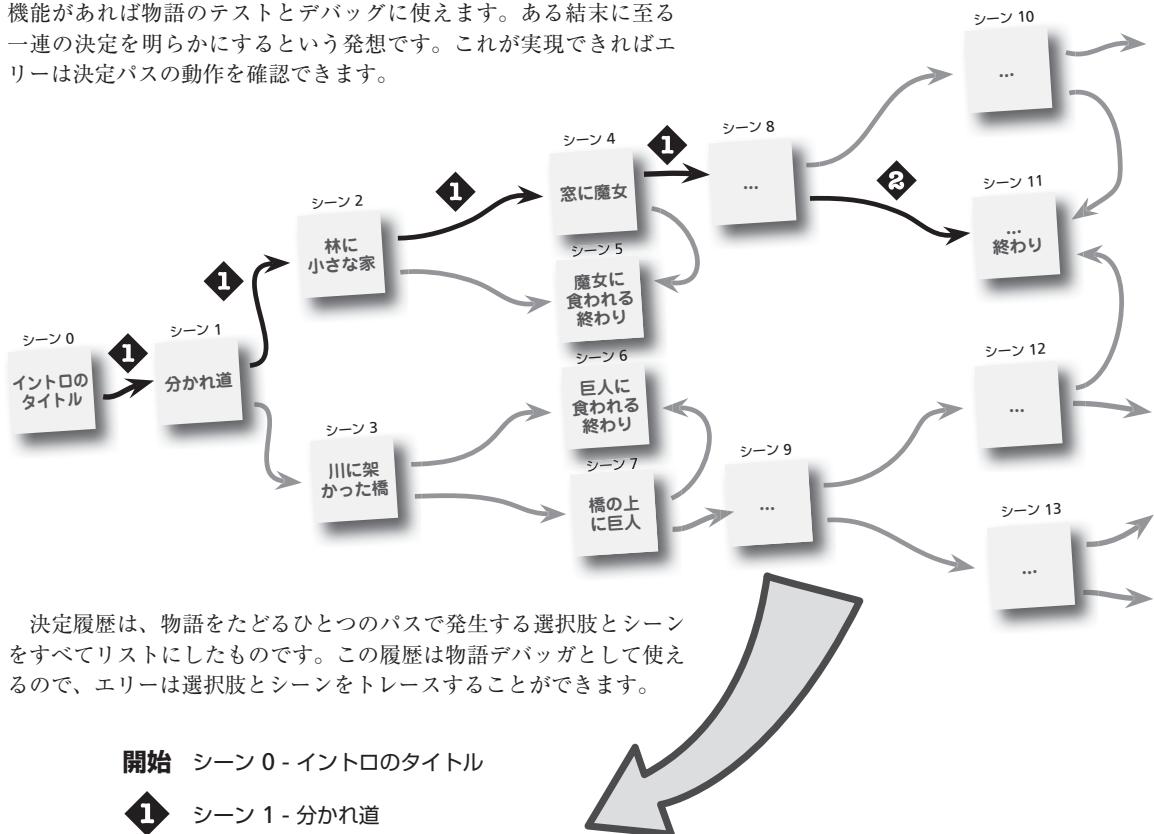
冒険に深入りする = 決定ツリーが大きくなる!

* 「棒人形の冒陶」の最新バージョンは <http://www.headfirstlabs.com/books/hfjs/> からダウンロードできます。



決定ツリーをたどる

ウェブブラウザの履歴機能にはあなたが開いた一連のページが保持されていますが、これと同じように「棒人形の冒険」に決定の履歴機能があれば物語のテストとデバッグに使えます。ある結末に至る一連の決定を明らかにするという発想です。これが実現できればエリーは決定バスの動作を確認できます。



開始 シーン 0 - イントロのタイトル

1 シーン 1 - 分かれ道

ボタン 1 を
選択 ...

1 シーン 2 - 林に小さな家

1 シーン 4 - 窓に魔女

... シーン 4 に
続く

1 シーン 8 - ...

2 シーン 11 - ...

終わり



頭の体操

「棒人形の冒険」のウェブページで決定の履歴機能をサポートするにはどんな変更をすればいいでしょう？

決定履歴をHTMLにする

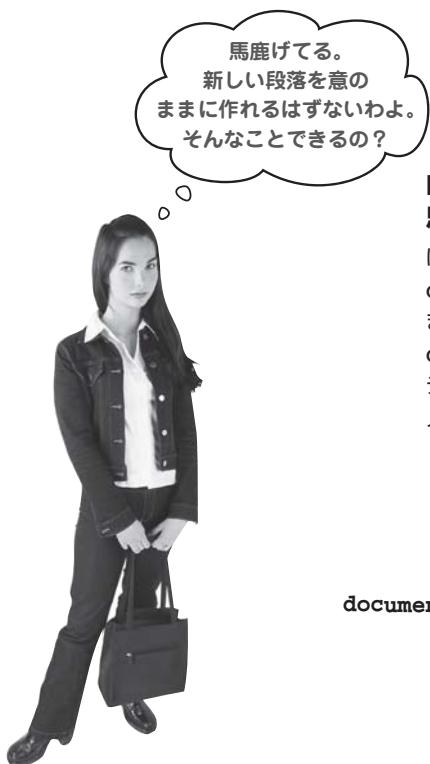
HTMLからみると、決定履歴のコードはそれほど極端に複雑ではありません。`div`要素とテキスト段落がそれぞれの決定ごとにあれば十分です。

```
<div id=history>
  <p>決定 1 -> シーン 1 : 分かれ道</p>
  <p>決定 1 -> シーン 2 : 林に小さな家</p>
  <p>決定 1 -> シーン 4 : 窓に魔女</p>
  ...
</div>
```

あとはDOMを使って決定履歴をノードの集まりとして生成するJavaScriptを書くだけです。

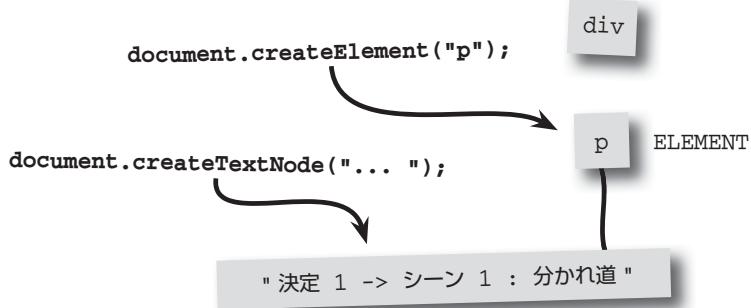
`p`要素には決定履歴にある選択が含まれています。

「棒人形の冒険」に決定履歴機能があれば、とても便利な物語デバッグツールになります。



DOMを使うと、テキスト段落を含むHTML要素を思いのままに作成できます。

ほんとうですよ。ドキュメントオブジェクトのもうひとつのメソッド`createElement()`を呼び出すと、HTML要素を作成することができます。`createElement()`を使って新しいコンテナ要素を作成し、次に`createTextNode()`を使って子テキストノードを作成してテキストコンテナを追加します。この結果、ページのノードツリーにまったく新しいノードの枝が接ぎ木されます。



HTML コードを組み立てる

`createElement()` を使って新しい要素を作成する場合に必要なのはタグの名前だけです。パラグラフ (`p`) 要素を作成するには、引数 "`p`" でこのメソッドを呼び出し、作成された要素を保持します。

```
var decisionElem = document.createElement("p");
```

新しい `p` 要素で開始します。

この時点ではまだ新しいパラグラフ要素にコンテンツがないので、ページの一部にはなっていません。要素にテキストコンテンツを追加するには、テキストノードを作成し、それを `p` ノードの子ノードとして追加します。

```
decisionElem.appendChild("決定 1 -> シーン 1 : 分かれ道");
```

`p` 要素はまだページに割り当てられていません。子のテキストノードを追加したので、テキストが含まれています。

"決定 1 -> シーン 1 : 分かれ道"

パラグラフ要素を `div` 要素である `history` の子ノードとしてページに追加します。

```
document.getElementById("history").appendChild(decisionElem);
```

`p` 要素は `div` 要素の子ノードとして追加されます。これはウェブページのパラグラフに追加されます。

```
...  
<div id="history">  
  <p>決定 1 -> シーン 1 : 分かれ道</p>  
</div>  
...
```

"決定 1 -> シーン 1 : 分かれ道"

「棒人形の冒険」で各シーンをたどるごとに、これらの手順が繰り返され、決定履歴が動的に作成されます。



重要ポイント

- `document`オブジェクトの`createElement()`を使うとHTML要素が作成できます。
- 要素にテキストコンテンツを追加するには、まず子ノードのテキストを作成し、それを要素に追加します。
- DOMツリーにノードを追加したり削除すれば、ウェブページを意のままに分解したり組み立て直すことができます。



自分で考えてみよう

`changeScene()`にコードを追加して「棒人形の冒険」の決定履歴機能を実現しましょう。

ヒント：現在のシーンがシーン0でないとき、決定履歴要素に子ノードのテキストを使って新しい段落を追加する必要があります。シーンが0のときは決定履歴をクリアします。

```
function changeScene(decision) {
    ...
    // 決定履歴を更新
    .....
}
```

自分で考えてみよう の答え

新しいテキストノードが
新しいパラグラフ要素に
追加されます。

`changeScene()` にコードを追加して「棒人形の冒険」の決定履歴機能を実現しましょう。

ヒント：現在のシーンがシーン 0 でないとき、決定履歴要素に子ノードのテキストを使って新しい段落を追加する必要があります。シーンが 0 のときは決定履歴をクリアします。

```
function changeScene(decision) {
  ...
  // 決定履歴を更新

  var history = document.getElementById("history");

  if (curScene != 0) {
    // 履歴に最新の決定を追加
    var decisionElem = document.createElement("p");
    decisionElem.appendChild(document.createTextNode("決定 " + decision +
      " -> シーン " + curScene + ":" + message));
    history.appendChild(decisionElem);
  } else {
    // 決定履歴をクリア
    while (history.firstChild)
      history.removeChild(history.firstChild);
  }
}
```

パラグラフ要素を
div に追加して
ページに
追加します。

`changeScene()` には `decision` と
いう名前のローカル変数があるので、
この変数には別の名前をつけます。

履歴の div を ID で
取得します。

決定履歴情報を使って新しい
テキストノードを作成します。

子ノードをすべて削除して
履歴 div をクリアします。

冒険物語をトレースする

「棒人形の冒険」の決定履歴の機能によって、物語の進行にあわせてロジックを注意深く追うことができます。



旅の続きはあなた次第

せっかく決定履歴デッキがあるんですから、機能が無駄にならないように「棒人形の冒険」の物語を膨らませましょう。あなたの物語を待っている友達がいますよ。



エクササイズ

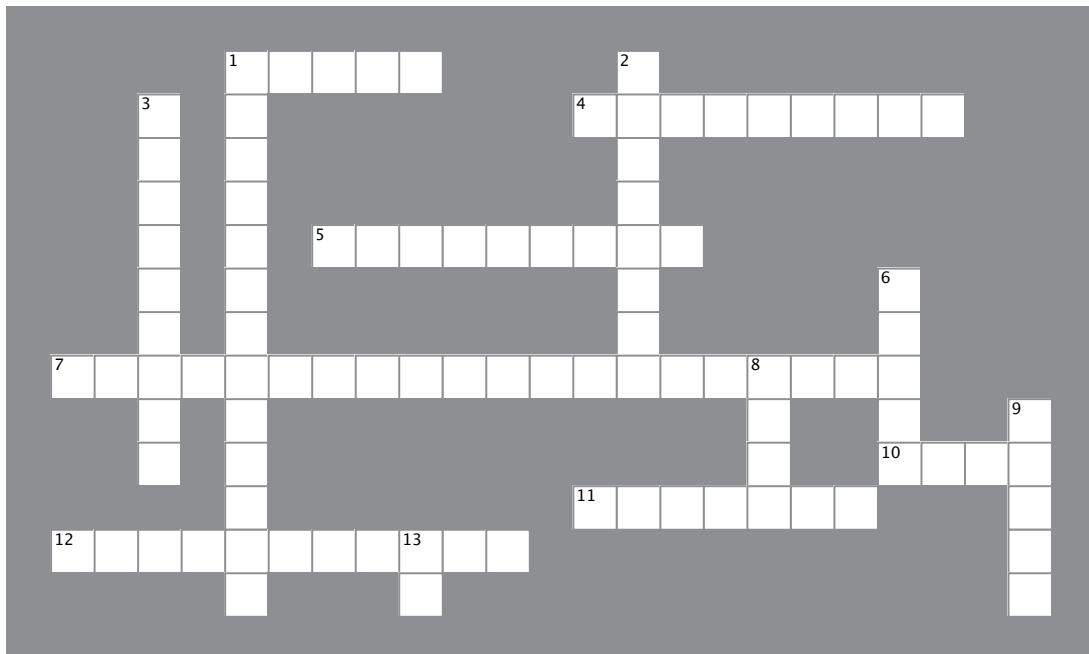
「棒人形の冒険」の物語の続きを思い描いて、インターラクティブなアプリケーションをオンラインで共有できるようにコードを追加してみましょう。

この課題に解答はありません。夢の冒険を楽しんでください。



JavaScriptクロスワード

「棒人形の冒険」の続きの制作に入り込む前に、ちょっとしたクロスワードパズルで休憩してみましょう。



ヨコのカギ

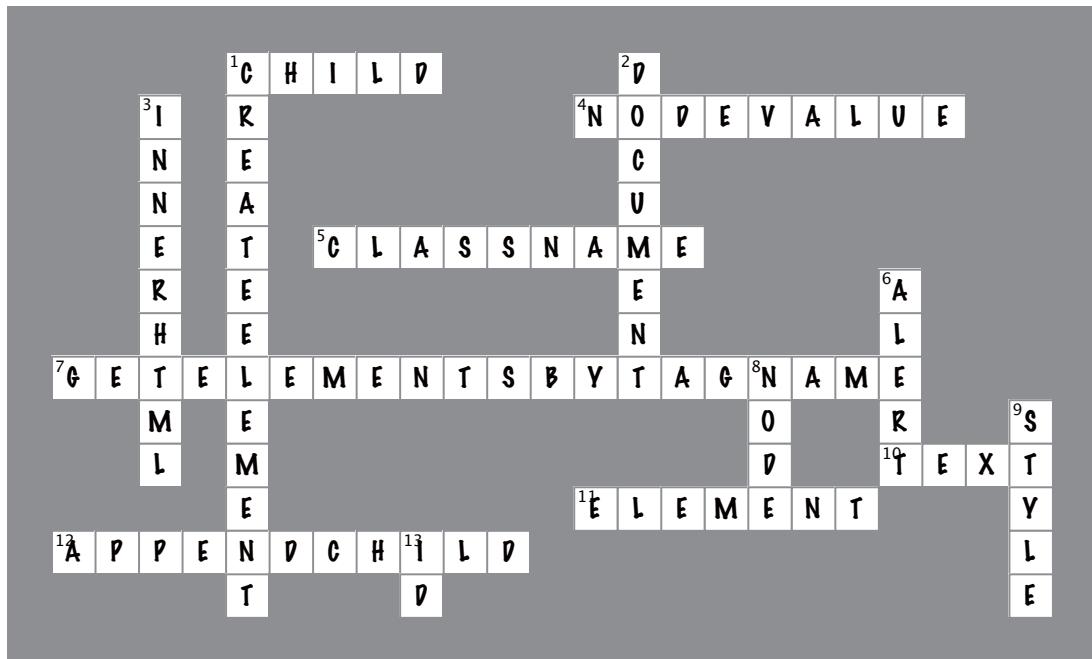
1. DOMツリーにあるノードの下にあるノードは.....と呼ばれています。
4. ノードオブジェクトの値を取得するのに使われるプロパティ。
5. 要素のスタイルクラスを設定するのに使われます。
7. divなどのあるタイプの要素をすべて取得するときにこのメソッドを呼び出します。
10. このタイプのノードはテキストコンテンツを保持します。
11. HTMLタグと一致するDOMノードのタイプです。
12. このメソッドを使ってノードに子ノードを追加します。

タテのカギ

1. このメソッドを呼び出してHTML要素を作成します。
2. DOMツリーの最上位にあるノードです。
3. HTML要素のコンテンツを変更する標準的でない方法です。
6. オンラインで物語を語るための出来の悪い方法。
8. ウェブページのコンテンツのDOMツリーにおける葉の部分。
9. このプロパティを使って要素の個々のスタイルプロパティにアクセスします。
13. JavaScriptからアクセスできるようにHTMLタグにこの属性を設定します。



JavaScript クロスワードの答え

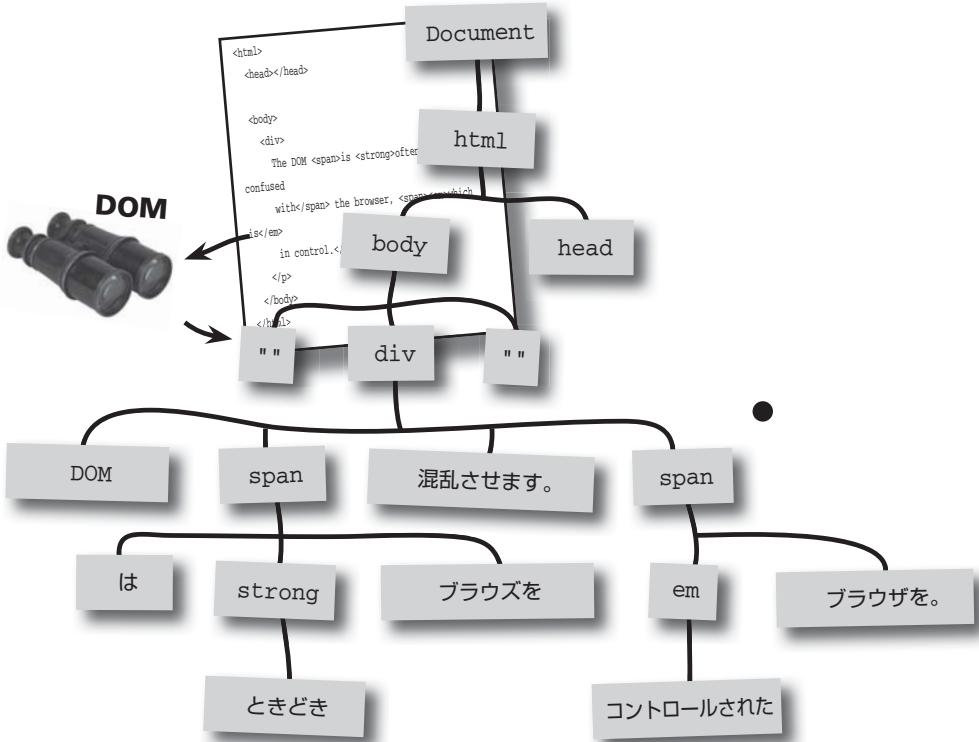
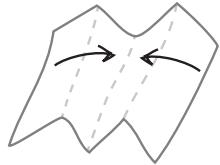


折り畳みページ

脳のところで
折り畳んでから
謎を解決しましょう。

DOMというのは、
ほんとは何なのでしょう？

左脳と右脳がご対面！

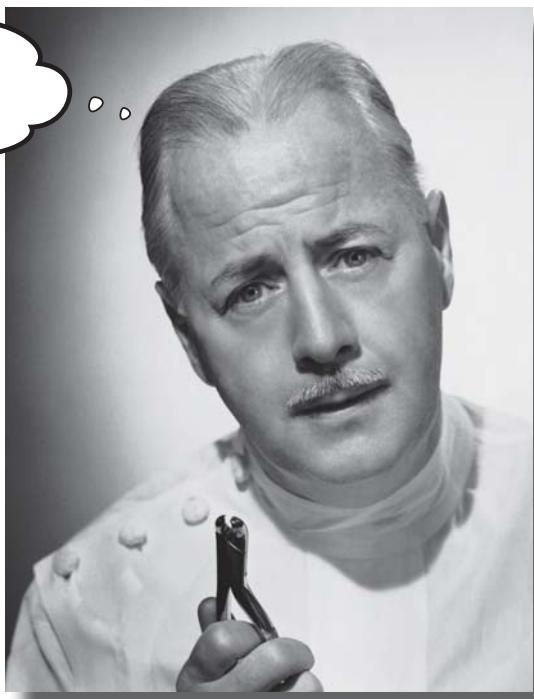


JavaScriptプログラマはDOMに夢中になりすぎないように
注意する必要があります。HTMLタグにアクセスする
手段としては便利ですが、あまり操作しすぎると、
ノードを消してしまうかもしれません。

9章 データを活氣づける

オブジェクトは
フランケンデータ

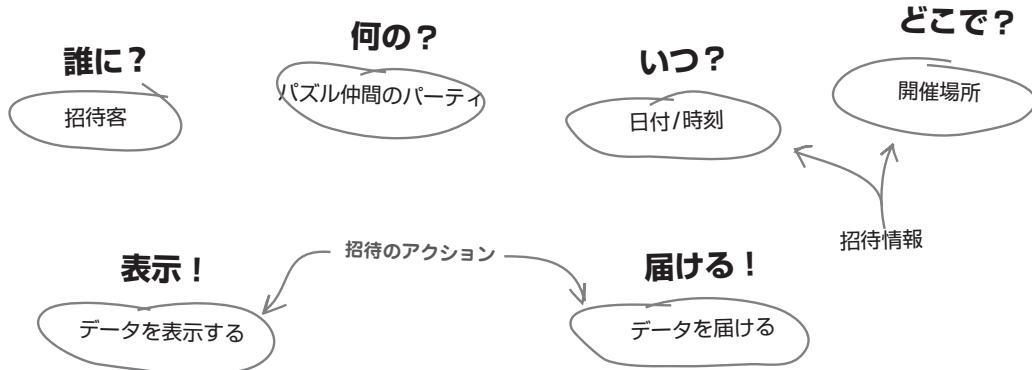
ばらばらにした
身体のあちこちを、後で
つなぎ合わせるのです。彼に
きいてみてください。



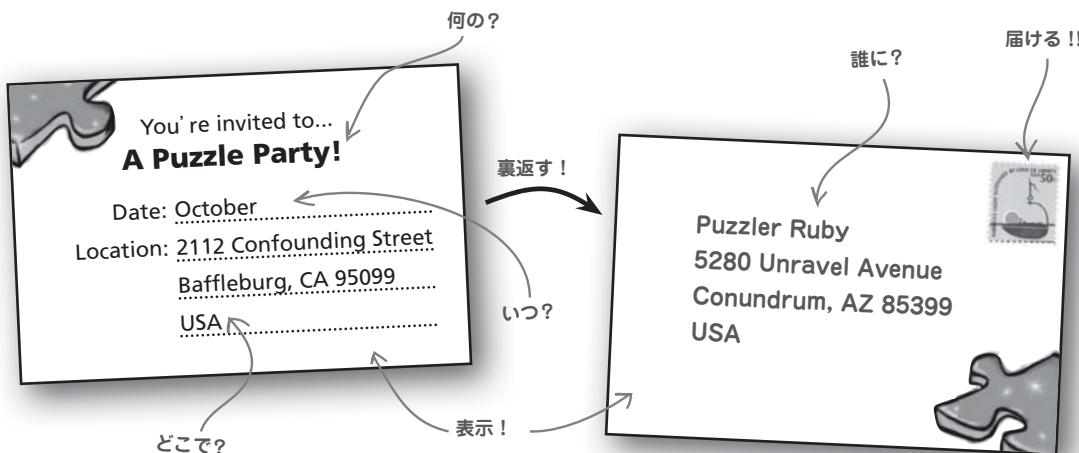
JavaScriptオブジェクトはフランケンシュタイン博士が作った人造人間みたいにおぞましくはありませんが、JavaScript言語の断片や部品をつなぎ合わせて、より強力なものになります。オブジェクトはデータとアクションを組み合わせた新しいデータ型になります。これまでみてきたデータ型より活動的です。自分自身をソートできる配列、自分自身を検索できる文字列。毛がはえて月に吠える狼男スクリプトなんてどうですか？

JavaScriptでパーティの招待状を作る

パーティを開催することになり、あなたは招待状を作ることになりました。そこで質問です。招待状を完璧なものにするには、どんな情報が必要でしょう？



JavaScriptで作る招待状は、データを変数で、アクションを関数でモデル化します。現実世界では、データとアクションを区別することができないところに問題があります。



現実世界では招待状はデータとアクションが組み合わさったひとつのモノ、つまりオブジェクトになっています。

データ+アクション=オブジェクト

JavaScriptではデータとアクションをそれぞれ別々に扱う必要はかならずしもありません。実際、JavaScript **オブジェクト**はこの2つを組み合わせて、データの**格納**とデータにもとづく**動作**の両方を行うデータ構造についています。この機能のおかげでJavaScriptでは現実世界における考え方をスクリプトに適用することができます。データとアクションを分けるのではなく、ひとつになったモノとして考えてみましょう。招待状をJavaScriptオブジェクトの観点でながめると、次のように整理されます。

オブジェクト

データ アクション

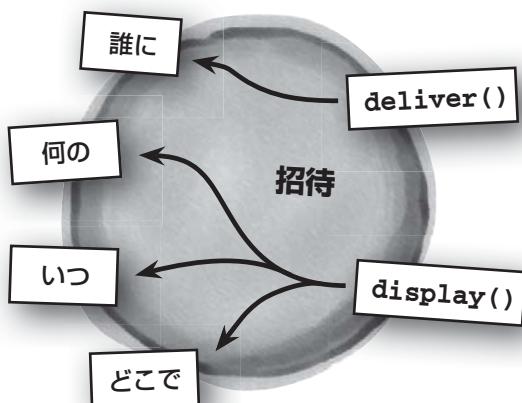
```
var who;
var what; + function display(what, when, where) {
var when; } var deliver(who) {
var where; } ...
```

オブジェクトの外からデータを
引数として関数に渡す必要があります。

=

```
function display() {
...
}
function deliver() {
...
}
```

招待状オブジェクトの中でデータと関数が共存し、それらがオブジェクトの外にある場合よりも互いに密接な関係にあります。さらに、オブジェクトの中にある関数は、関数の引数として変数を渡さなくてもオブジェクトの中の変数にアクセスすることができます。

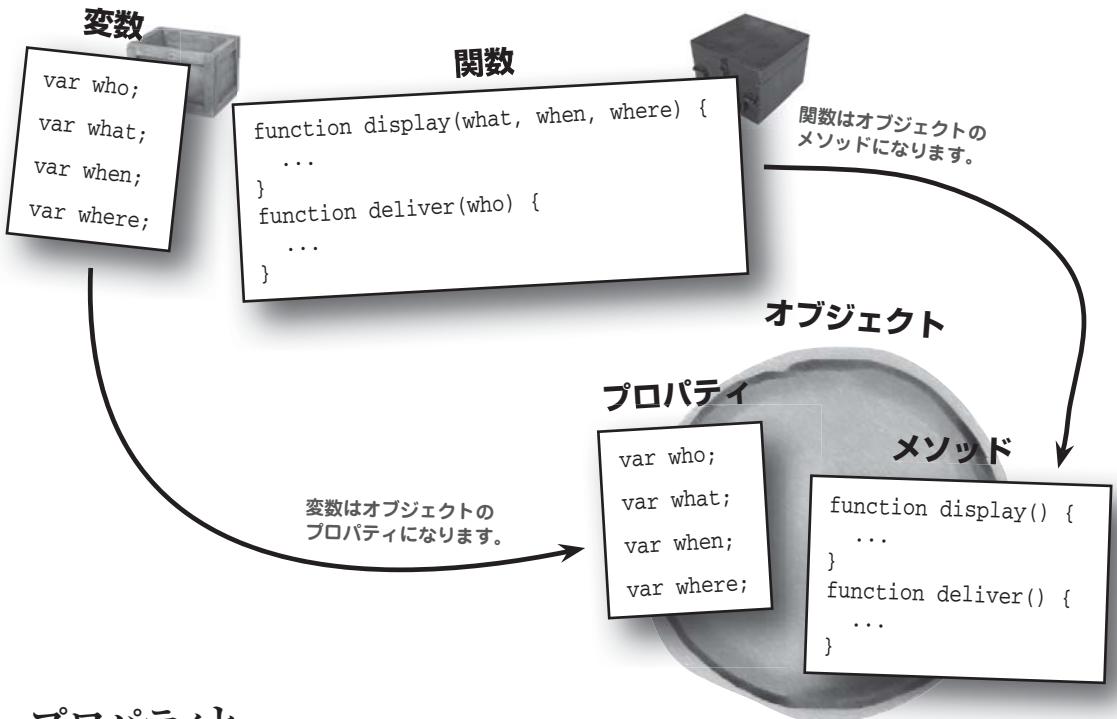


オブジェクトは
変数と関数を
ひとつの
コンテナに
まとめます。

招待状の中のデータは関数からアクセスできますが、オブジェクトの外からは見えないように隠されています。つまりオブジェクトはデータを格納するコンテナとして働き、それをもとに動作するコードと結びついています。

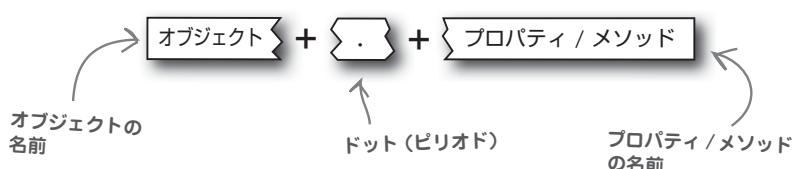
オブジェクトには固有のデータがある

オブジェクトの中にある変数と関数は、そのオブジェクトに属するメンバーです。専門的な言い方をすると、変数はオブジェクトプロパティであり、関数はオブジェクトメソッドです。オブジェクトにはデータが格納され、データに対するアクションが実行されますが、それはオブジェクトの文脈の中で行われます。



プロパティと
メソッドは
オブジェクトに
おける変数と
関数に
相当します。

プロパティとメソッドはオブジェクトが所有します。これは配列にデータが格納されるのと似たような意味で、オブジェクトの中に格納されるということです。しかし配列と違って、オブジェクトのプロパティとメソッドにアクセスするときは、**ドット演算子**と呼ばれる特別な演算子を使います。

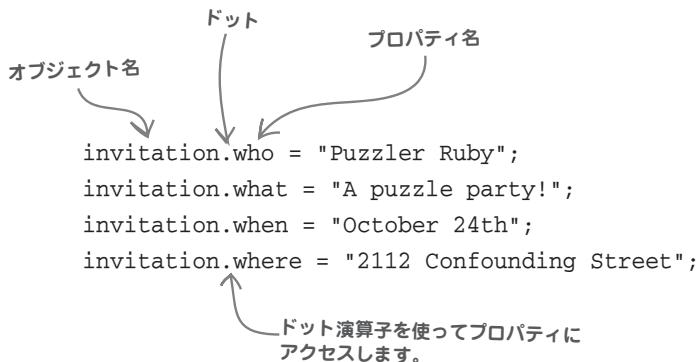


オブジェクトのメンバーをドットで参照

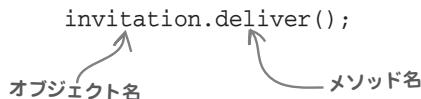
ドット演算子はオブジェクトに属するプロパティやメソッドをオブジェクトから参照するためにあります。これは姓があなたの属する家族を示し、名がその人を示すのに似ています。オブジェクトも同様で、プロパティ名はプロパティを示し、**オブジェクト名**はそのプロパティが属するオブジェクトを示します。ドット演算子はこの2つを結びつけるのです。

プロパティとドット演算子を使ってJavaScript招待状オブジェクトに実際にデータを代入することができます。

ドット演算子は
オブジェクトの
プロパティや
メソッドを
参照します。



データとアクションはどちらも同じオブジェクトの中にあるので、メソッドがオブジェクトのデータを使うときメソッドに何かを渡す必要はありません。招待状オブジェクトのアクションは次のように簡単に実行できます。



エクササイズ

招待状にはパーティの出欠の返事を示すRSVPプロパティがありません。パズル愛好家のためのパーティを計画しているルビーの招待状にrsvpプロパティを追加し、sendRSVP()を呼び出して返事を送信するコードを書いてください。

.....
.....



エクササイズ

答え

招待状にはパーティの出欠の返事を示す RSVP プロパティがありません。パズル愛好家のためのパーティを計画しているルビーの招待状に rsvp プロパティを追加し、sendRSVP() を呼び出して返事を送信するコードを書いてください。

```
invitation.rsvp = "attending";  
invitation.sendRSVP();
```

ドット演算子で
invitationオブジェクトの
プロパティやメソッドを
参照します。

これを論理値プロパティにすることも
できます。trueの場合その人は出席、
falseの場合欠席になります。

Q: オブジェクトって何なんでしょう？ データ型はあるんですか？

A: オブジェクトにデータ型はあります。オブジェクトはプロパティとメソッドの集まりに名前をついたものです。もっと正確に説明すると、オブジェクトはデータ型そのものです。これまで説明したデータ型には、数値、テキスト、論理値がありました。これらは情報のひとつ断片を表現するので、プリミティブなデータ型と呼ばれています。オブジェクトはデータの断片を複数包んでいるので、複雑なデータ型とみなされます。すでに知っているプリミティブのリスト（数値、文字列、論理値）に第四のデータ型として「オブジェクト」を追加することができます。組み込みのJavaScriptオブジェクトやあなたが作成したオブジェクトは、オブジェクトのデータ型になります。

Q: プロパティとメソッドのかわりにグローバル変数と関数を使うことはできませんか？ 関数はグローバル変数にアクセスできるので、問題ないですよね？

A: できるにはできますが、問題あります。というのは、関数以外の他のコードもグローバル変数にアクセスできてしまうからです。データをほんとうに必要とするコードからだけそのデータが見えるように制限したいですね。そうすれば他のコードが誤ってデータを変更してしまうことも防げます。

素朴な疑問に

答えます

残念ながら、JavaScriptにはオブジェクトの外にあるコードからプロパティにアクセスできないようにする機能は、いまのところありません。またオブジェクトのプロパティに直接アクセスしたくなる状況もあります。しかしオブジェクトにデータを置いておけば、そのデータはオブジェクトに論理的に関連づけられていることになります。データの断片をオブジェクトに結びつけておく方が、スクリプトの中で浮遊するデータの断片（グローバル変数）にしておくよりも、その文脈が分かるので有益です。

Q: これまで何度かオブジェクトの記法でドット演算子が使われていました。オブジェクトはいつでも使えるのですか？

A: はい。オブジェクトを使わずにJavaScriptを使うことは、現実にはほとんどありえないことがわかるでしょう。JavaScriptそのものがたくさんのおbjectをひとつに集めたものだからです。たとえば、alert()はwindowオブジェクトのメソッドです。つまりwindow.alert()で呼び出すことができます。windowオブジェクトはブラウザウインドウを表現します。ただし明示的にオブジェクトを参照する必要がないので、alert()だけで呼び出せるのです。

Q: うーん、ほんとに混乱してきました。つまり関数は実際にはメソッドということですか？

A: はい、そうです。でもこのように関数を考えると混乱しますね。関数とは、他のコードから名前で呼ぶことができるコードのかたまりです。メソッドはオブジェクトの中に置かれた関数にすぎません。すべての関数が実際にはひとつのオブジェクトに属していると考えると、混乱しますね。alert()は関数でありメソッドでもあります。そのため関数としても呼び出せるし、メソッドとしても呼び出せます。しかし、ほとんどのメソッドは、オブジェクト記法を使って呼び出す必要があります。実際、JavaScriptのほとんどの関数はひとつのオブジェクトに属しているので、メソッドでもあるわけです。そのオブジェクトは多くの場合ブラウザのwindowオブジェクトです。このオブジェクトは、メソッド呼び出しにおいてオブジェクトが指定されなかつた場合、デフォルトのオブジェクトとして使われます。このためalert()などのメソッドは関数と見なすことができます。これらのメソッドはたまたまwindowオブジェクトに属しているだけであって、そのオブジェクトとの論理的なつながりはありません。



重要ポイント

- オブジェクトはデータとそのデータを扱うコードを組み合わせた特別なデータ構造です。
- 実際のところオブジェクトは変数と関数をひとつずつ構造に組み合わせた変数にすぎません。
- 変数はオブジェクトの中に置かれるとプロパティになります。関数はメソッドになります。
- プロパティやメソッドを参照するときは、オブジェクトの名前の後にドットを続けて、さらにプロパティやメソッドの名前を続けます。

キューブパズル愛好家のためのブログ

招待状の送り先の一人がルビーです。キューブパズルに夢中な彼女は、仲間に会える日を待ちきれません。ルビーはパーティに行くだけでなく、ブログを作ってキューブパズルの楽しさをみんなと共有したいと考えています。そこで彼女はYouCubeというブログを始めることにしました。



オブジェクトを使うと、
コードを変更する必要があった
ときでも、簡単に管理できるそうね。
その分、キューブパズルに時間を
さけて嬉しいわ。



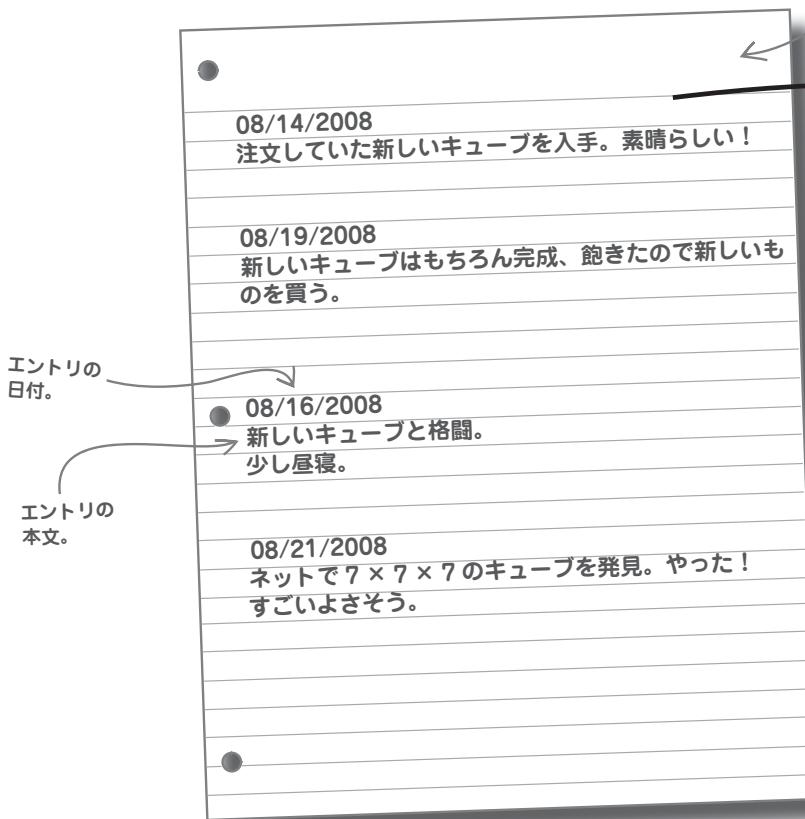
JavaScriptではより堅牢で圧倒的に管理しやすいコードを作るための手段としてカスタムオブジェクトがサポートされていることを、ルビーは聞きつけました。また、ブログの多くが飽きられてしまうのは、プロガーがブログの管理に疲れてしまうから、という話も聞きました。そこでルビーは、YouCubeをオブジェクト指向スクリプトとして作って、カスタムオブジェクトを使えるようにすることで、長く続けられるブログにしようと思いました。

**オブジェクト指向の = キューブで遊べる
YouCube 時間が増える！**

YouCubeを脱構築する

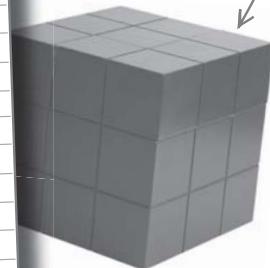
ルビーは手書きの日記をみて、彼女のブログは日付とテキストで構成する必要があることに気づきましたが、JavaScriptを使ってそれらを格納する方法がわかりません。でもキューブ日記を手書きするのは、もううんざりです。

手書きの
YouCube。



} エントリは日付とテキスト
文字列の組み合わせです。

ルビーのお気に入り
キューブパズル。



ルビーが必要としているのは、情報の組（日付とテキスト）の格納とアクセスを行うための手段です。これはまさにJavaScriptオブジェクトが提供すべきものと非常に似ています。これをを使って複数の情報をひとつにまとめるのです。

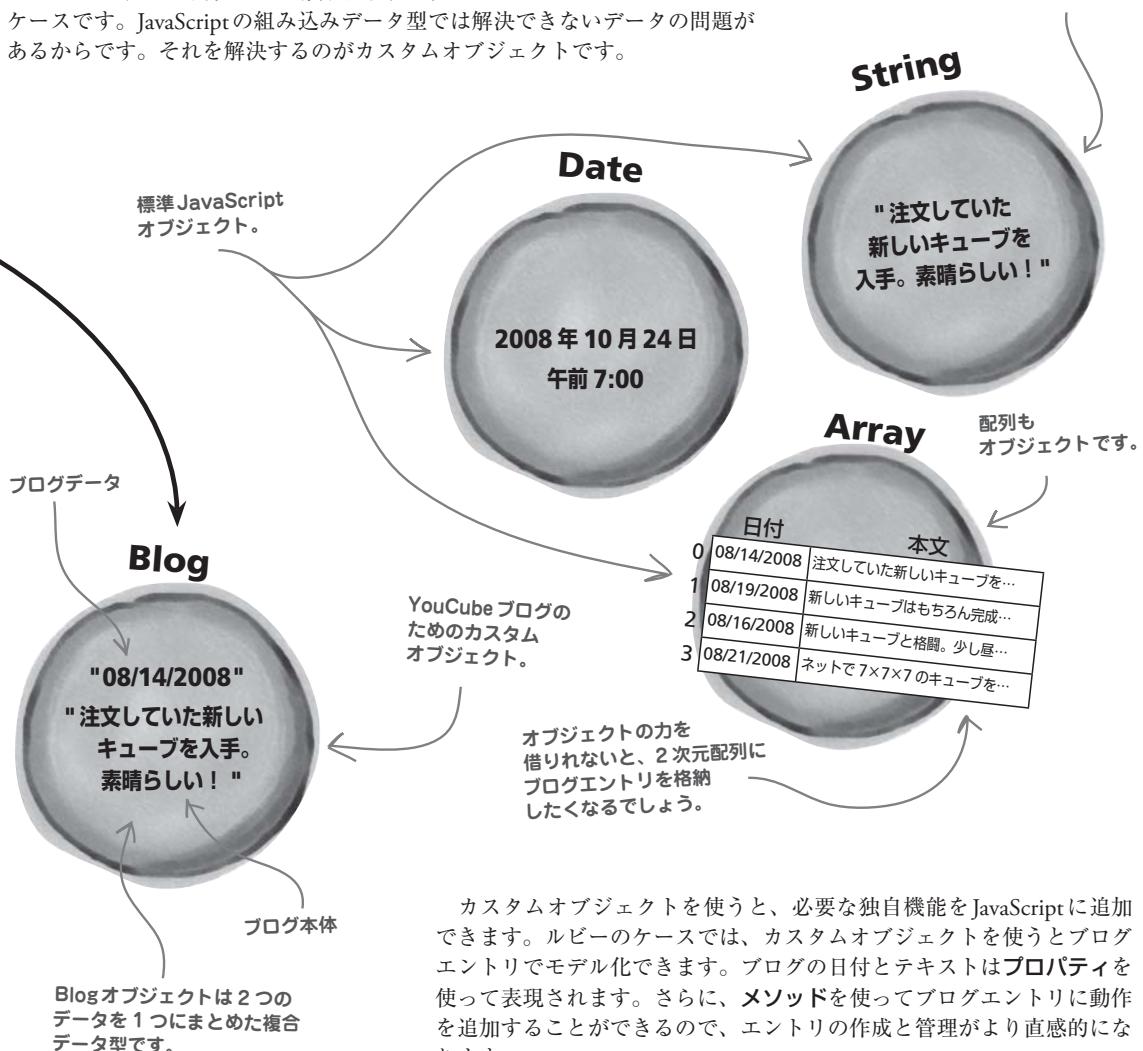
ブログデータ+ブログ本体=ブログオブジェクト

カスタムオブジェクトを作ると、ブログデータの
2つの断片をひとつに
まとめることができます。

カスタムオブジェクトでJavaScriptを拡張する

JavaScript言語には、たくさんの標準オブジェクトがあり、そのいくつかについてこの章の後半で説明します。これらのオブジェクトは便利なですが、これだけでは十分でない場合もあります。YouCubeブログはまさにそのケースです。JavaScriptの組み込みデータ型では解決できないデータの問題があるからです。それを解決するのがカスタムオブジェクトです。

文字列は実は
オブジェクトなんです。



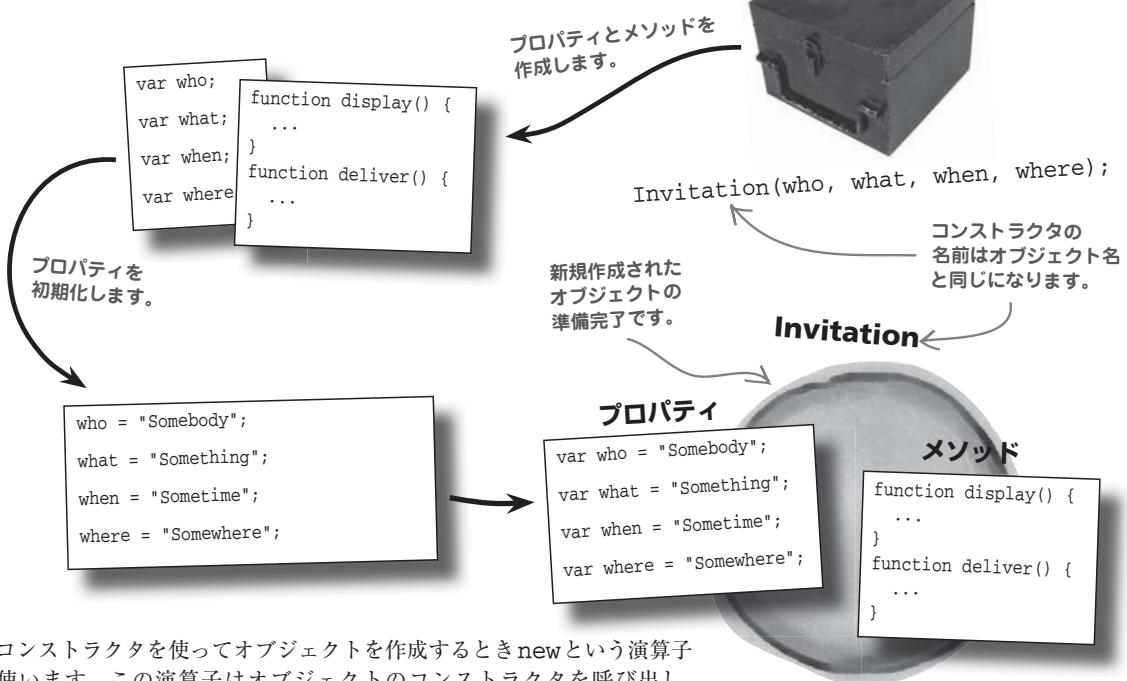
カスタムオブジェクトを使うと、必要な独自機能をJavaScriptに追加できます。ルピーのケースでは、カスタムオブジェクトを使うとブログエントリでモデル化できます。ブログの日付とテキストはプロパティを使って表現されます。さらに、メソッドを使ってブログエントリに動作を追加することができるので、エントリの作成と管理がより直感的になります。

しかし、カスタムオブジェクトを実現するには、その作り方を理解する必要があります。

カスタムオブジェクトを構築する

オブジェクトを作成するとき、オブジェクトに関連づけられているデータを初期化する必要があります。それを実現するのが**コンストラクタ**と呼ばれる特別なメソッドです。カスタムオブジェクトにはそれ固有のコンストラクタがあり、メソッドの名前はオブジェクトの名前と同じです。コンストラクタはオブジェクトが作成されるとき呼び出され、オブジェクトを初期化します。カスタムオブジェクトを作るとき、それに適したコンストラクタを書くのはあなたの仕事です。

コンストラクタは
オブジェクトを作成
する責任があります。



コンストラクタを使ってオブジェクトを作成するとき `new` という演算子を使います。この演算子はオブジェクトのコンストラクタを呼び出し、オブジェクトの作成プロセスを開始します。コンストラクタによるオブジェクト作成はメソッドの呼び出しに似ていますが、オブジェクトのコンストラクタを直接呼び出すのではなく必ず `new` 演算子を使ってオブジェクトの作成を開始する点が重要です。

```
var invitation = new Invitation("Somebody", "Something", "Sometime", "Somewhere");
```

Three annotations explain the code:

- An arrow from the `new` keyword points to the text "new演算子はオブジェクトを新規作成するときに使われます。" (The new operator is used when creating a new object).
- An arrow from the argument list points to the text "オブジェクトは変数に格納されます。" (The object is stored in a variable).
- An arrow from the constructor call points to the text "コンストラクタはメソッドのように呼び出されます。" (The constructor is called like a regular method).
- An arrow from the variable `invitation` points to the text "プロパティはコンストラクタに引数を渡して初期化します。" (Properties are initialized by passing arguments to the constructor).

コンストラクタの中はどうなっている？

コンストラクタの最大の仕事は、オブジェクトのプロパティの初期値を設定することです。コンストラクタの中でプロパティを作成するときは、JavaScriptのキーワードである `this` を使ってプロパティを設定します。この `this` はプロパティの持ち主をオブジェクトに設定し、同時に、その初期値を設定します。このキーワードは文字通りの意味をもたらします。コンストラクタの中にローカル変数を作成するのではなく、「このオブジェクト」に属しているプロパティを作成することになるわけです。

キーワード `this` でオブジェクトのプロパティを他の通常の変数と区別します。

```
function Invitation(who, what, when, where)
{
    this.who = who;
    this.what = what;
    this.when = when;
    this.where = where;
}
```

コンストラクタの名前は
オブジェクトと同様に
大文字で始めます。

コンストラクタの引数は
プロパティに代入されます。

コンストラクタの作り方は
他の関数と同じです。

`this` という
キーワードは
コンストラクタの
中でオブジェクト
のプロパティを
作成するための
鍵になります。

オブジェクトプロパティは、オブジェクトの記法（ドット演算子）とキーワードの `this` を使ってコンストラクタの中で初期化されます。キーワード `this` がなかったら、コンストラクタはオブジェクトプロパティを作成しなければならないことがわかりません。コンストラクタの処理では、4つのプロパティが作成され、コンストラクタの引数で渡された値をそれぞれのプロパティに代入しています。

自分で考えてみよう



Blogオブジェクトのコンストラクタを書いてください。ブログエントリの日付と本文テキストのプロパティを作成し初期化します。

.....
.....
.....
.....

自分で考えてみよう の答え

Blogオブジェクトのコンストラクタを書いてください。ブログエントリの日付と本文テキストのプロパティを作成し初期化します。

コンストラクタ
の名前は
オブジェクトと
同じにします。

```
function Blog(body, date) {  
    this.body = body;  
    this.date = date;  
}
```

キーワードthisを
使ってオブジェクト
のプロパティを参照します。

本文テキストと日付が
引数としてコンストラクタ
に渡されます。

コンストラクタの引数を使って
プロパティを初期化します。

ブログオブジェクトを実際に作成する

Blogオブジェクトの姿はかなりはっきりしてきましたが、まだ実際には作成されていません。理論的には正しいとしても、まだ証明されていない仮定の段階です。コンストラクタはオブジェクトの設計を決めているだけあって、new演算子を使ってはじめてコンストラクタが呼び出され、オブジェクトが実際に作成されます。それではBlogオブジェクトを作ってみましょう。

手書きの
ブログエントリ。



このサンプルは <http://www.headfirstlabs.com/books/hfjs> からダウンロードできます。

JavaScript Blog
オブジェクト。

```
var blogEntry = new Blog ("注文していた新しいキューブを…", "08/14/2008");
```

Blog

"注文していた
新しいキューブを入手。
素晴らしい！"

"08/14/2008"

Blog()
コンストラクタを
呼び出してオブジェクトを
作成します。

```
function Blog(body, date) {  
    ...  
}
```

オブジェクトが
作成されました。

Q: オブジェクトの作成について混乱しています。オブジェクトを作成するのは演算子newですか、それともコンストラクタですか？

A: 両方です。演算子newはオブジェクトの作成を設定するだけであり、オブジェクト作成の仕事の大部分はコンストラクタを呼び出すことで実現しています。演算子newを使わずに関数のようにコンストラクタを呼び出しても、オブジェクトは作成されません。またコンストラクタなしで演算子newを使っても意味がありません。

Q: オブジェクトをカスタム作成するときはコンストラクタが必要ですか？

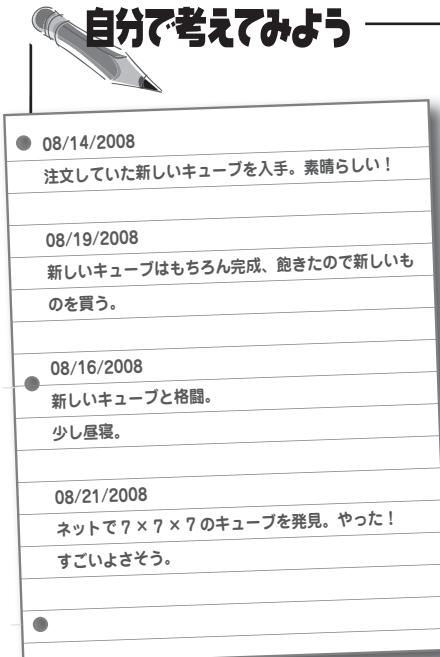
A: オブジェクトのプロパティを作成するのはコンストラクタ

素朴な疑問に 答えます

Q: thisとは何でしょうか？

A: thisはJavaScriptのキーワードで、オブジェクトを参照するのに使われます。thisはあるオブジェクトの中からそのオブジェクトを参照します。奇妙に聞こえるかもしれませんが、けして頭がおかしくなったわけではありません。たとえば、人だからのする部屋で腕時計を落としまい、それを誰かが見つけたとします。その腕時計を拾った人をみて、あなたは「それはわたしの腕時計です」と言うでしょう。重要なのは「わたしの」という言葉で時計の持ち主が自分であることを主張している点です。thisというキーワードの意味も同様で、オブジェクトの所有者を示しています。このためthis.dateは、このコードが現れるオブジェクトが持つdateプロパティを意味します。

自分で考えてみよう



blogという名前の変数をBlogオブジェクトの配列として作成してください。この配列はYouCubeブログのブログエントリで初期化されます。各エントリの本文テキストは先頭の数語だけ書いてください。

```
var blog =
[ .....  
.....  
.....  
];
```

自分で考えてみよう の答え

● 08/14/2008	注文していた新しいキューブを入手。素晴らしい！
08/19/2008	新しいキューブはもちろん完成、飽きたので新しいものを買う。
08/16/2008	新しいキューブと格闘。 少し昼寝。
08/21/2008	ネットで $7 \times 7 \times 7$ のキューブを見た。 やった！ すごいよさそう。
●	

blog という名前の変数を Blog オブジェクトの配列として作成してください。この配列は YouCube ブログのブログエントリで初期化されます。各エントリの本文テキストは先頭の数語だけ書いてください。

```
var blog =
  [ new Blog("注文していた新しいキューブを…", "08/14/2008"),
    new Blog("新しいキューブはもちろん完成…", "08/19/2008"),
    new Blog("新しいキューブと格闘。少し昼…", "08/16/2008"),
    new Blog("ネットで $7 \times 7 \times 7$ のキューブを…", "08/21/2008") ];
```

各ブログエントリは
本文テキストと日付を使って
Blog オブジェクトとして
作成されます。

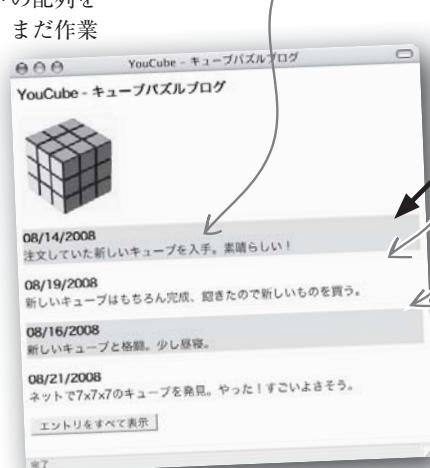
YouCube 1.0

ブログのデータを表示する JavaScript コードと Blog オブジェクトの配列を組み合わせれば、YouCube の最初のバージョンのできあがりです。まだ作業が終わったわけではありませんが、とりあえずブログの最初の一歩が踏み出せて、ルビーは喜んでいます。



Blog オブジェクトの良い
ところは、YouCube の
ブログの日付とテキストを
組み合わせている点ね。

Blog オブジェクトに
格納されている日付は
YouCube ページに
きれいに表示されます。



YouCube 1.0 を実現するのに必要な Blog オブジェクトを作成するコードをちょっと見てみましょう。

コード詳細



```

<html>
  <head>
    <title>YouCube - キューブパズルのブログ</title>

    <script type="text/javascript">
      // ブログオブジェクトコンストラクタ
      function Blog(body, date) {
        // プロパティを代入します
        this.body = body;
        this.date = date;
      }

      // ブログエントリのグローバル配列
      var blog = [ new Blog("注文していた新しいキューブを…", "08/14/2008"),
        new Blog("新しいキューブはもちろん完成…", "08/19/2008"),
        new Blog("新しいキューブと格闘。少し…", "08/16/2008"),
        new Blog("ネットで7×7×7のキューブを…", "08/21/2008") ];
    
```

showBlog() はブログエントリをページの "blog" という div に表示します。page.

```

      // ブログエントリのリストを表示します
      function showBlog(numEntries) {
        // 表示するエントリの件数を調整します
        if (!numEntries)
          numEntries = blog.length;

        // ブログエントリを表示します
        var i = 0, blogText = "";
        while (i < blog.length && i < numEntries) {
          // ブログエントリの背景色をグレイにします
          if (i % 2 == 0)
            blogText += "<p style='background-color:#EEEEEE' >";
          else
            blogText += "<p>";

          // ブログの HTML コードを生成します
          blogText += "<strong>" + blog[i].date + "</strong><br />" + blog[i].body + "</p>";

          i++;
        }
        // ページに HTML コードを設定します
        document.getElementById("blog").innerHTML = blogText;
      }
    </script>
  </head>
  <body onload="showBlog(5);">
    <h3>YouCube - キューブパズルのブログ</h3>
    
    <div id="blog"></div>
    <input type="button" id="showall" value="エントリをすべて表示" onclick="showBlog();"/>
  </body>
</html>

```

"blog" という div に整形されたブログエントリのコードを設定します。

"blog" という div は最初は空ですが、整形されたブログデータが挿入されます。

このボタンがクリックされたらブログエントリをすべて表示します。

Blog() コンストラクタはプロパティを 2 つ作成します。

Blog オブジェクトの配列。

表示するブログエントリの数が引数で渡されなかった場合、すべてのエントリを表示します。

ブログエントリの背景色を変えて読みやすくします。

Q: YouCubeで「エントリをすべて表示」ボタンが必要なのにはなぜですか？

A: このブログはエントリが4個しかないのが現状なので、からならずしもこのボタンは必要ではありません。しかしブログが成長すると、YouCubeのメインページに最初に表示されるエントリの数を制限して、ユーザをうんざりさせないことが重要になります。そのためブログのコードではデフォルトで最初の5個のエントリだけを表示するようにしています。「エントリをすべて表示」ボタンは、このデフォルト値を無効にしてすべてのブログエントリを表示します。

素朴な疑問に 答えます

Q: DOMメソッドのかわりにinnerHTMLを使ってブログエントリを表示するのはなぜですか？

A: DOMメソッドはウェブ標準準拠である点が好まれているのですが、複雑なHTMLコードを自動的に生成するには、あまり扱いやすくありません。コンテナタグの<p>やは、それらのコンテンツを子ノードとして持つ親ノードとして作

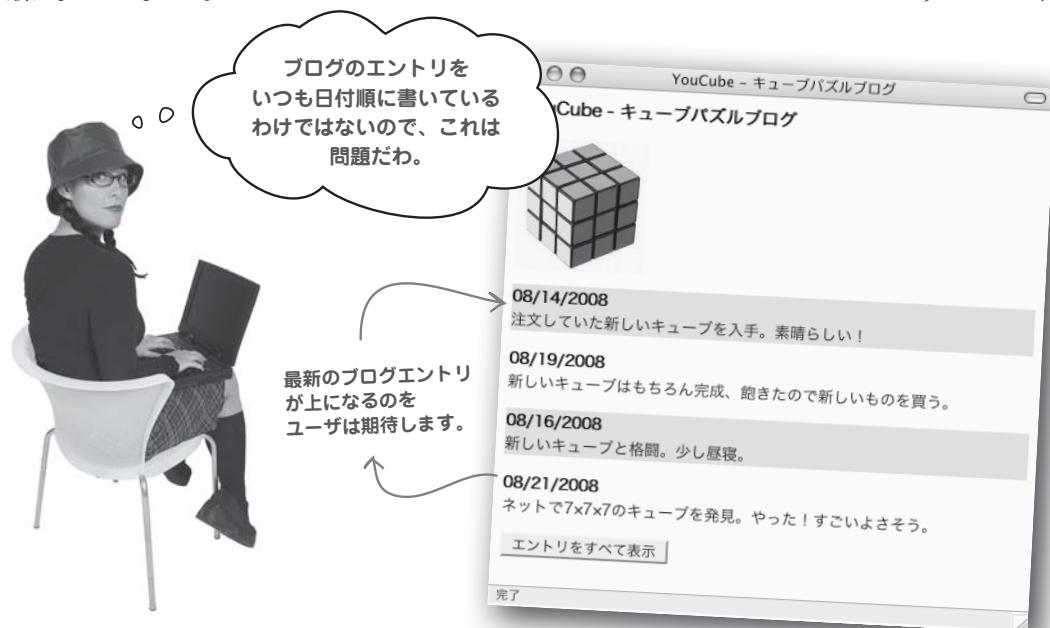
成される必要があるからです。この場合innerHTMLの方が便利ですし、YouCubeのコードが簡潔になります。

Q: どうしてBlogオブジェクトにinnerHTMLはメソッドがないのですか？

A: YouCubeを動かすためには、Blogオブジェクトのメソッドを作る前に片付けておかなければならないことがあります。でも心配しないでください、YouCubeの長期計画にはメソッドの作成も含まれています。よく設計されたオブジェクトではメソッドが重要な部分を占めています。Blogオブジェクトも例外ではありません。

ブログの順番がめちゃくちゃ

YouCube 1.0には不具合がないわけではありません。ルビーはブログエントリの順番が間違っていることに気づきました。一番最近のエントリが最初に表示されなければなりません。いまはエントリが格納されている順番で表示されていて、日付順になってしまっています。



最新の
ブログエントリが
先頭に表示される
ようにします。

ソートする必要がある

ブログの表示順の問題を解決するために、ルビーはブログの配列を日付順でソートしました。JavaScriptはループ処理と比較をサポートしているので、ブログエントリをループでまわして、日付を互いに比較し、日付の新しい順に並べ替えることができます

- ① ブログ配列をループする
- ② Blog オブジェクトの日付を互いに比較する
- ③ 現在のエントリより次のエントリの日付が新しければ、両者を入れ替える。

このブログのソート方法は、ブログの日付を比較する処理をうまく書ければ、問題なく動くように見えます。



ちょっと待ってよ。日付が文字列で格納されていたら、どうやって比較したらどっちが新しいかわかるの？

文字列に格納された日付は実は日付ではありません。

ルビーが考えたブログのソート方法には思ぬ問題が潜んでいます。文字列に格納された日付には時間という概念がないのです。言い換えると、"08/14/2008"と"08/19/2008"を比較したとき、どちらも文字列なので、どちらが新しいのかわからないのです。もちろん文字列を比較することはできますが、その場合日付の形式を理解せずに比較するので、日付を構成する月、日、年ごとに比較することができません。

ブログエントリを日付でソートする方法を考える前に、まずブログに日付を格納する方法について考え直す必要があります。



日付のためのJavaScriptオブジェクト

ルビーが必要としているのは、互いに日付を比較できるような日付の格納方法です。言い換えると、日付であることを理解していて、日付として振る舞えるものが必要です。そうです、これはオブジェクトにとても似ていますね。実はJavaScriptにはDateオブジェクトが組み込まれていて、まさにルビーが必要としている機能を提供してくれるのです。

最初のブログ
エントリの日付。

日付データを設定する
ためのメソッド。

getDate()

setMonth()

getDay()

setYear()

getFullYear()

2008年8月14日

日付データを
取得するための
メソッド。

組み込みオブジェクト
であるDateはある
時刻を表現します。

Dateオブジェクトは時間のある時点をミリ秒単位で表現するJavaScriptの標準オブジェクトです。Dateオブジェクトは内部でプロパティを使いますが、オブジェクトのユーザには見えません。Dateオブジェクトの操作では純粋にメソッドだけを使います。

Blogオブジェクトと同様、new演算子を使ってDateオブジェクトを作成します。現在の日付と時刻を表現するDateオブジェクトを作成する例を以下に示します。

新規作成された
Dateオブジェクトを
変数に格納します。

var now = new Date();

new演算子を使って
Date
オブジェクトを作成します。

このDateオブジェクトは
現在の日付/時刻を
表します。

Dateオブジェクトの
中の時刻はミリ秒
で表現されます。

このDateオブジェクトは現在の日付と時刻で作成され初期化されます。Dateオブジェクトを作成する構文は関数やメソッドの呼び出しつとんどり似ています。というのも実際にはDateオブジェクトのコンストラクタを呼び出しているからです。Date()コンストラクタには日付を指定する文字列を引数で渡すことができます。たとえば、次のDateオブジェクトは最初のブログエントリの日付を表現します。

日付はテキスト文字列と
してコンストラクタに
渡されます。

var blogDate = new Date("08/14/2008");

時間の計算

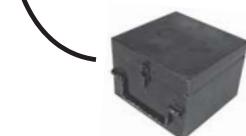
オブジェクトの最も強力な機能のひとつは、オブジェクト自体を操作する方法を継承できる点です。たとえば2つの日付の差の日数を計算する方法を考えてみましょう。閏年を考慮しながら日付を日数に変換する必要があります。Dateオブジェクトを使ってこの処理を行うことができます。2つのDateオブジェクトの差の日数を計算する関数を以下に示します。

```
この関数は2つの
Dateオブジェクトを
引数として受け取ります。
function getDaysBetween(date1, date2) {
    var daysBetween = (date2 - date1) / (1000 * 60 * 60 * 24);
    return Math.round(daysBetween);
}
```

ミリ秒を秒にして分にして
時間にして日にします。

シンプルですがこれだけで
すべての処理を行なう
強力なコードです。

round() は Math オブジェクトの
メソッドです。この章の後半で
説明します。



`getDaysBetween(date1, date2);`

この関数は日付の差を計算する簡単なコードで Date オブジェクトの力を示しています。日付の差の計算において複雑な処理は Date オブジェクトの中に埋まっています。必要な結果は日付の差だけですが、この差はミリ秒で表されます。そのためミリ秒と日付に変換して、結果を四捨五入します。この小さな関数を使えば、2つの日付の差が必要な場合、いつでも再利用することができます。



エクササイズ

YouCubeの最初の2つのブログエントリのDateオブジェクトを作成してください。この2つのDateオブジェクトを引数にして`getDaysBetween()`を呼び出し、アラートボックスに結果を表示してください。



エクササイズ

答え

YouCubeの最初の2つのブログエントリのDateオブジェクトを作成してください。この2つのDateオブジェクトを引数にしてgetDaysBetween()を呼び出し、アラートボックスに結果を表示してください。

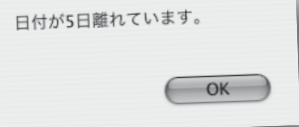
ブログエントリの日付を表す

Dateオブジェクトを
2つ作成します。

```
var blogDate1 = new Date("08/14/2008");
var blogDate2 = new Date("08/19/2008");
alert("日付が" + getDaysBetween(blogDate1, blogDate2) + "日離れています。");
```

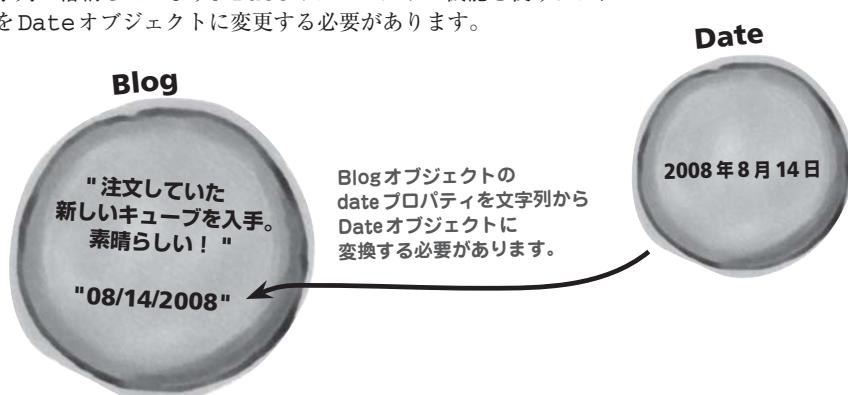
この関数は差を
返します。

2つのDateオブジェクト
を引数として渡します。



ブログの日付を再考する

JavaScriptが提供するDateオブジェクトを使うと日付を賢く操作することができますが、YouCubeのBlogオブジェクトでは日付をDateオブジェクトではなく文字列で格納しています。Dateオブジェクトの機能を使うには、ブログの日付をDateオブジェクトに変更する必要があります。



Blogオブジェクトのdateプロパティを、文字列ではなくDateオブジェクトで格納することはできるでしょうか？

オブジェクトの中にあるオブジェクト

Blog オブジェクトは他のオブジェクトを持つ必要があるオブジェクトの事例に適しています。Blog オブジェクトの 2つのプロパティは実際にはすでにオブジェクト、つまり String オブジェクトです。String オブジェクトは、文字列を囲っただけのオブジェクトリテラルで作成されているので、オブジェクトのようには見えません。Date オブジェクトは融通がきかないでの、new 演算子を使って作成する必要があります。

ブログの date プロパティを Date オブジェクトで作成するには、Blog オブジェクトを作成する際に new 演算子を使う必要があります。悪夢のように感じられるかもしれません、以下のコードをみれば、安心できるでしょう。

```
var blogEntry = new Blog("お天気以外はいつもどおり。",
```

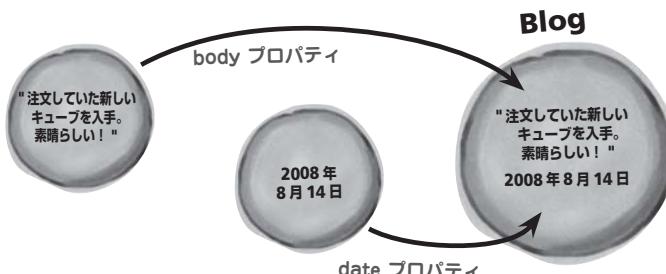
```
    new Date("10/31/2008") );
```

文字列リテラルは自動的に String オブジェクトを作成します。

Blog オブジェクトは
new 演算子を使って
作成されます。

new 演算子を使って
Date オブジェクトを作成し
Blog() コンストラクタに渡します。

このコードが示すように、YouCube ブログエントリは、2つのオブジェクト (String オブジェクトと Date オブジェクト) を含むオブジェクトとして作成されています。もちろん、YouCube のすべてのブログエントリをうまく表現するには、Blog オブジェクトの配列を作る必要があります。



new 演算子は
コンストラクタの力を
借りてオブジェクトを
作成します。

自分で考えてみよう

YouCube の Blog オブジェクトの配列を作成するコードを書き直してください。
日付を Date オブジェクトにします。本文テキストは短くしてかまいません。

自分で考えてみよう の答え

YouCubeのBlogオブジェクトの配列を作成するコードを書き直してください。
日付をDateオブジェクトにします。本文テキストは短くしてかまいません。

ここでもブログエントリは
Blogオブジェクトとして
作成します

```
var blog = [ new Blog("注文していた新しいキューブを…", new Date("08/14/2008")),  
            new Blog("新しいキューブはもちろん完成…", new Date("08/19/2008")),  
            new Blog("新しいキューブと格闘。少し昼…", new Date("08/16/2008")),  
            new Blog("ネットで7×7×7のキューブを…", new Date("08/21/2008")) ];
```

ブログエントリの本文テキストは
文字列リテラルで大丈夫です。

Blogオブジェクトの日付は
Dateオブジェクトとして作成します。

素朴な疑問に答えます

Q: Dateオブジェクトの日付がミリ秒で格納されているのはなぜですか？

A: まず最初に、Dateオブジェクトは時間におけるある瞬間を表現していることを理解してください。宇宙にポーズボタンがあったとして、これをクリックして宇宙を一時停止させたとします。しかし、何かを参照しないかぎり、時間を止めたその瞬間がいつなのか、人々に知らせる手段がありません。そこで1970年1月1日を基点に時刻を表すことになると、基点からの隔たりを計測することが必要になります。たとえばその隔たりが38年8月14日3時間29分11秒あったとして、このやり方で隔たりを把握するのは何かと面倒です。そこである極めて小さい時間を単位にし

て時刻を表せば、操作がより簡単になります。その単位をミリ秒にしたらどうでしょうか？ そうするとさきほどの時刻は1,218,702,551,000ミリ秒と表されます。このようにミリ秒を単位にすると、とても大きな数値になりますが、JavaScriptでは問題になりません。

Q: Dateオブジェクトを使うときは、ミリ秒への変換を気にする必要がありますか？

A: 場合によります。Dateオブジェクトには、ミリ秒を直接操作せずにすむように日付の部分を抽出するためのメソッドがあります。しかし、2つの日付の差を計算する必要がある場合、ミリ秒が必要になるでしょう。

重要ポイント

- JavaScriptの標準オブジェクトであるDateは時刻をミリ秒で表現します。
- Dateオブジェクトは日付と時刻の構成要素にアクセスするためのメソッドを持っています。
- Dateオブジェクトは互いに日付を比較するだけでなく、日付を数学的に操作する方法も提供します。
- String以外のほとんどのオブジェクトがそうであるようにDateオブジェクトもnew演算子を使って作成します。

Dateのままでは使いにくい

Blogオブジェクトの日付プロパティがDateオブジェクトに変換されたので、ルビーはブログエントリの日付順ソートの作業に戻りました。準備完了と思ったら、ブログエントリの日付の表示がなにか暗号みたいになっている問題が生じました。ユーザは投稿の時間帯など気にしないだろうけど、YouCubeの体験を損ねるのでは、とルビーは心配しています。YouCubeにDateオブジェクトを導入するには、もっと細かな検査が必要なのです。

ブログの日付がひどいことになっています。情報が多くすぎます。

The image shows a screenshot of a blog titled "YouCube - キューブパズルブログ". The blog entries are displayed in a list:

- Thu Aug 14 2008 00:00:00 GMT+0900 (JST)**
注文していた新しいキューブを入手。素晴らしい！
- Tue Aug 19 2008 00:00:00 GMT+0900 (JST)**
新しいキューブはもちろん完成、飽きたので新しいものを買う。
- Sat Aug 16 2008 00:00:00 GMT+0900 (JST)**
新しいキューブと格闘。少し寝覚。
- Thu Aug 21 2008 00:00:00 GMT+0900 (JST)**
ネットで7x7x7のキューブを発見。やった！すごいよさそう。

A button labeled "エントリをすべて表示" is visible at the bottom left. A callout bubble from a character's head contains the text:

Dateオブジェクトに
変えたのは意味があったけど、
でもブログの日付がひどいことに
なったわ。日付の形式を変える
コードの書き方がわからないなあ。

Another callout bubble points to the date format in the entries, stating:

日付の表示がひどいだけ
なく、ブログエントリの
表示順も揃っていません。

To the right, a woman with glasses and a pensive expression is shown holding a laptop. A thought bubble above her head contains the text:

ルビーはYouCubeの日付が暗号みたいになった理由で頭を悩
ませています。日付を表示するコードなどを書いた覚えがないか
らです。ただ文字列をDateオブジェクトに変換しただけです。
JavaScriptはどこで悪さをして日付の表示が不細工になったので
しょうか？

オブジェクトをテキストに変換する

幸いなことに、何かのせいで YouCube の日付が不細工になったのではなく、実際は JavaScript のオブジェクトが日付をこのように表示するのは自然なことだったのです。つまりこういうことです。JavaScript オブジェクトには `toString()` というメソッドがあり、オブジェクトをテキストで表現する機能を提供します。日付が暗号みたいになったのは、`Date` オブジェクトの **デフォルト** の `toString()` メソッドがそのように出力したからなのです。

```
var blogDate = new Date("08/14/2008");
alert(blogDate.toString());
```



文字列が期待される文脈でオブジェクトが使われるとき、自動的に `toString()` が実行されます。たとえばブログの日付のアラートコードを次のように書き換えれば、同じ結果が表示されます。

```
alert(blogDate);
```

`alert()` は文字列を受け取るので、この裏では `toString()` が呼ばれて `Date` オブジェクトを文字列に変換されています。



toString() メソッドを使うと `Date` オブジェクトが時刻をどのように保持しているのか明らかになります。

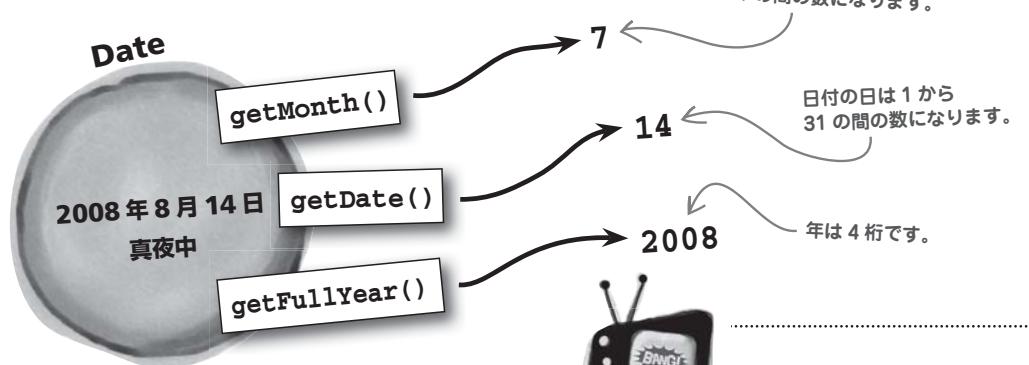
toString() は
オブジェクトの文字列
表現を提供します。

`alert()` は文字列を期待しているので、`Date` オブジェクトはその文脈を察知して文字列での表現を提供します。ここで呼ばれるのが `toString()` なのです。

この `toString()` の処理は、YouCube の日付の表示を MM/DD/YYYY のように読みやすい形式にしないのであれば問題になりません。しかし、日付が `Date` オブジェクトの `toString()` を使ったデフォルトの文字列表現のままでは、YouCube にはふさわしくありません。

日付の断片にアクセスする

ルピーが必要としているのは日付の表示形式を変更する方法です。Dateオブジェクトの表示形式を変更するには、日付の個々の断片、つまり、月、日、年にアクセスする必要があります。これを別の形式で組み立て直せばいいのです。幸いなことにDateオブジェクトは日付情報の断片にアクセスするためのメソッドを提供しています。



Dateオブジェクトでは、この3つのメソッド以外にもたくさんのメソッドが提供されています。いずれもDateオブジェクトの日付と時間をさまざまな方法でアクセスするためのメソッドです。ただし、YouCubeのブログ日付を変更するのに必要なのは、この3つのメソッドです。

Dateのメソッドが返す値には注意しましょう。
要注意！

getMonth()は月を0(1月)から11(12月)の間の数値で返します。getDate()は日を1から31の間の数値で返します。



YouCubeのブログの日付問題を解決するためにコードを書き直してください。ブログエントリを整形して、変数blogTextに格納します。ブログの日付の表示形式はMM/DD/YYYYにします。以下のコードはもとのバージョンです。

```
blogText += "<strong>" + blog[i].date + "</strong><br />" + blog[i].body + "</p>";
```

自分で考えてみよう の答え

YouCubeのブログの日付問題を解決するためにコードを書き直してください。ブログエントリを整形して、変数blogTextに格納します。ブログの日付の表示形式はMM/DD/YYYYにします。以下のコードはもとのバージョンです。

```
blogText += "<strong>" + blog[i].date + "</strong><br />" + blog[i].body + "</p>;
```

```
blogText += "<strong>" + (blog[i].date.getMonth() + 1) + "/" +
```

```
    blog[i].date.getDate() + "/" +
```

```
    blog[i].date.getFullYear() + "</strong><br />" +
```

```
    blog[i].body + "</p>;
```

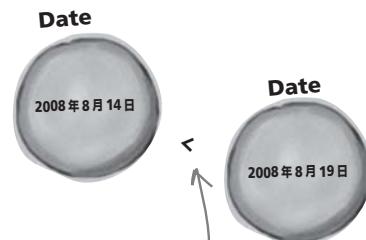
Dateオブジェクトの自動変換に頼らずに表示形式を細かくコントロールできます。

Dateオブジェクトのメソッドを呼び出すと日付の構成要素構成要素を抽出できます。

ブログの日付はMM/DD/YYYYのカスタム形式で表示されます。

Dateを使うとソートが楽になる

ブログの日付はうまくDateオブジェクトに変換できたので、文字列よりもはるかにソートしやすくなりました。ブログの表示順に戻りましょう。ブログのエントリはブログ配列に格納されているのと同じ順序で表示されているので、日付の逆順になっていません。ほとんどのブログは逆順で表示されるので、最も新しいエントリがブログエントリのリストの最初に表示されます。もとのブログのソート方法を思い出してください。



- ① ブログ配列をループでまわします。
- ② Blogオブジェクトの中にあるDateオブジェクトを次のものと比較します。
- ③ 次のブログエントリの日付が現在のエントリより新しければ、エントリを入れ替えます。

この方法の日付を比較する部分でDateオブジェクトを使えば、処理はかなり簡単になりますが、その他の部分のコードについては、それほど変わらないでしょう。一連のデータをソートする方法は一般的なプログラミングの問題なので、さまざまな解決法が存在します。車輪を再発明するのは嫌ですよね。



JavaScriptに組み込みの
ソート機能があったら、一連のデータを
ソートする作業が省略できるんじゃ
ないかしら？

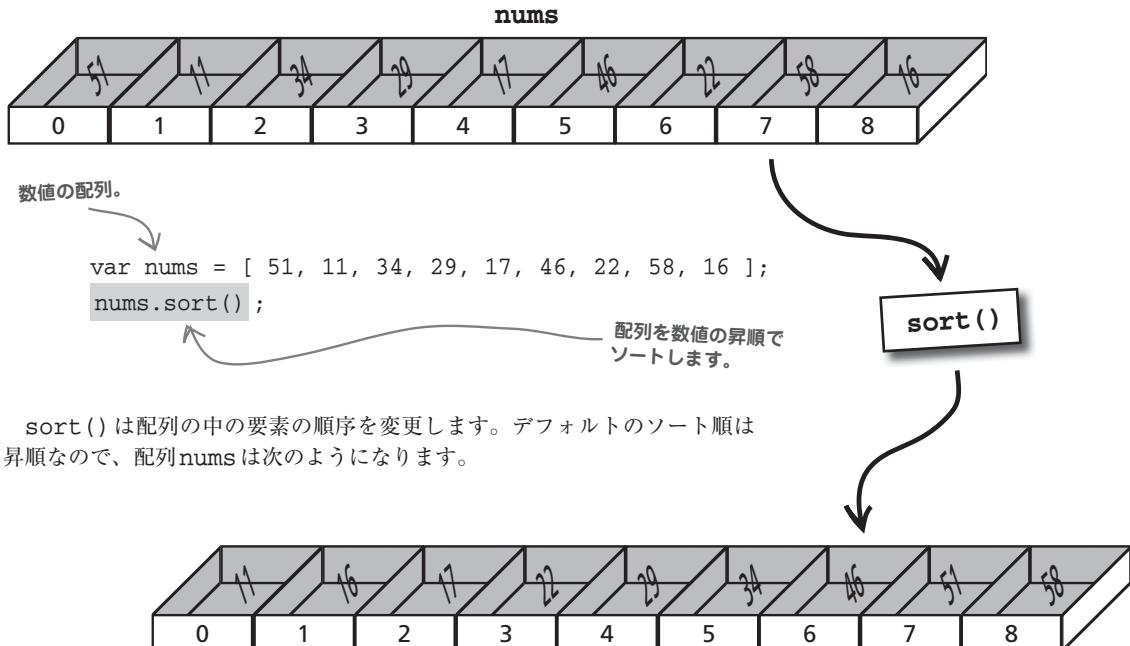
オブジェクトとしての配列

配列が自分で自分をソートできる、なんてことがあるんでしょうか？ 日付が自分で文字列にする方法を知っているとしたら、配列も自分で自分をソートすることができると考えても、それほど無理はありません。しかし、それを実現するにはソートを実現するメソッドが必要なので、配列はオブジェクトである必要があります。そして実際にそうなのです。Mandangoのスクリプトにあったコードを思い出してください。

配列は実は
オブジェクト
なのです。

```
for (var i = 0; i < seats.length ; i++) {  
    ...  
}  
  
変数seatsは  
配列です。  
  
lengthは配列オブジェクトの  
プロパティです。配列にある  
要素の数を返します。
```

配列は実はオブジェクトだったという秘密がわかったとはいえ、はたして配列は自分で自分をソートできるのでしょうか？ 配列はたとえばlengthプロパティを持っているだけでなく、配列のデータに対して働くメソッドももっています。そして配列のデータをソートするsort()というメソッドがあります。その動作は次のようにになります。



配列のソートを調整する

Arrayオブジェクトのsort()メソッドのデフォルトの動作では十分でないことがあります。ソートの動作はsort()が配列の要素を比較するときに使う**比較関数**によって決まります。この比較関数を独自に作れば、ソート順を調整することができます。この関数は一般には次のような作りになっています。

```
function compare(x, y) {
    return x - y;
}
```

2つの引数はソートしたい配列の要素になります。
この返値をもとに配列の中のxとyの位置を置き換えます。

compare()の返値は、xとyを比較した結果を示す数値になります。

compare(x, y)

<0 → xの後にyがきます。
 0 → ソートしません。
 x と y はそのままです。
 >0 → yの後にxがきます。



sort()を呼び出すとき、このcompare()関数が配列ソートの方程式に注入されます。compare()関数への参照をメソッドに渡すわけです。

nums.sort(compare);

配列のソートはカスタム
関数compare()を使って
実行されます。



自分で考えてみよう

YouCubeのブログ配列のエントリを降順（最新のものが最初になる）でソートするcompare()というカスタム比較関数のコードを書いてください。

ヒント：Blogオブジェクトはマイナス記号を使って互いに引き算することができます。



自分で考えてみよう の答え

YouCubeのブログ配列のエントリを降順（最新のものが最初になる）でソートする compare() というカスタム比較関数のコードを書いてください。

ヒント：Blogオブジェクトはマイナス記号を使って互いに引き算することができます。

配列にはBlogオブジェクトが格納されているので、この2つの引数はBlogオブジェクトです。

```
function compare(blog1, blog2) {
    return blog2.date - blog1.date;
}
```

2番目の日付から1番目の日付を引くので、降順のソート順になります。

2つの日付を数値（ミリ秒）で引き算します。

関数リテラルを使ってソートを簡潔にする

配列ソートの比較関数の役割を考えると、実際のところこれを使うのは sort() メソッドだけです。YouCubeのスクリプトにおいて、他では使われていないので、名前のある関数にする理由はありません。

6章のサーモスタッフで使った関数リテラルを憶えてますか？ compare() の使われ方を考えると、これは関数リテラルに適しています。実際、YouCubeのブログのソート処理は簡単なので、 compare() を関数リテラルにすれば sort() に直接渡せます。



```
blog.sort(function(blog1, blog2) {
    return blog2.date - blog1.date;
});
```

関数リテラルを配列の sort() メソッドに直接渡します。

熱心なパズルファンのルビーは効率を優先します。この場合、名前のある関数は sort() の脇役にすぎないので、これを関数リテラルにした方が効率的です。ルビーはなぜ比較関数が3行のコードである必要があるのか、疑問に思っています。この場合、関数リテラルにしても JavaScript コードの実行部分はまったく変わらないので、コードを1行にまとめて簡潔にする意味があります。

```
blog.sort(function(blog1, blog2) { return blog2.date - blog1.date; });
```

関数リテラルを 1行コードにしています。

素朴な疑問に答えます

Q: どんなオブジェクトでも`toString()`を持っていませんか?

A: はい。`toString()`をもたないカスタムオブジェクトを作成した場合、そのオブジェクトを文字列が期待される文脈で使うと、JavaScriptはこれがオブジェクトであると報告します。ここで報告される文字列はあまり意味のあるものではないので、もっと意味のあるものにしたいときは、カスタムオブジェクトに`toString()`を追加すればいいのです。

Q: Dateオブジェクトの場合、ソート用の比較はどうなりますか?

A: ソート用の比較関数の目的は、2つの引数のソートを制御するための値を数値で返すことです。日付を比較する場合、現在に近い日付ほど先頭にくるようにソートしたいとしましょう。現在に近いほど値が大きくなるので、第2引数の日付から第1引数の日付を引き算すれば、求めるソート結果が得られます。つまり、第2引数が第1引数より大きければ(結果は0より大きくなるので)、第2引数の日付の後に第1引数の日付が続きます。

Q: `Array.sort()`はカスタムの比較関数を使うか、それともデフォルトの比較を使うか、どうやって判断するのでしょうか?

A: `sort()`に引数があるかないかで判断します。引数がない場合、デフォルトのソート用比較が使われます。引数がある場合、関数参照として解釈され、これをもとに項目をソートします。つまり、比較関数の関数参照がオプション引数になっているわけです。

ルビーはご機嫌

ルビーの構想である全世界のキューブ愛好家のためのキューブパズルのブログにYouCubeのブログは近づきつつあります。

YouCube - キューブパズルブログ

8/21/2008 ネットで7x7x7のキューブを発見。やった！すごいよさそう。

8/19/2008 新しいキューブはもちろん完成、飽きたので新しいものを買う。

8/16/2008 新しいキューブと格闘。少し昼寝。

8/14/2008 注文していた新しいキューブを入手。素晴らしい！

[エントリをすべて表示](#)

完了

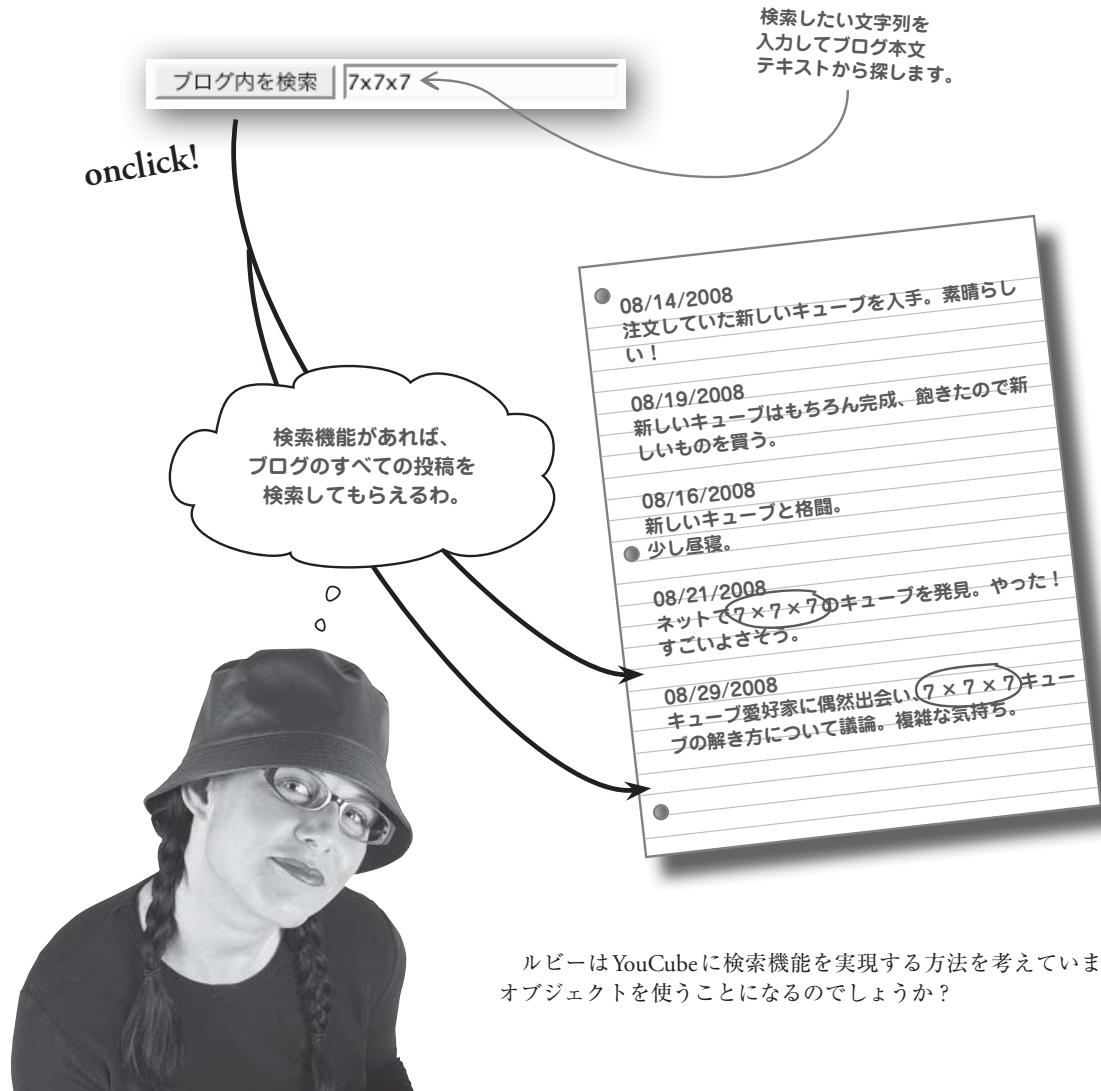
日付の表示がきれいになりました。

パズルと同じくらい
ブログも大好き。



検索できるといいね

YouCubeはとても順調に稼働しているのですが、ブログの投稿をすべて検索できる機能を要望するユーザが出てきました。ブログの投稿をこの先も増やす計画なので、長く続けていくためにも、便利な検索機能に賛成です。



ブログ配列を検索する

YouCubeで検索機能を実現するには、ブログ配列のエントリをループでまわして、投稿されたテキストに 対して検索をかける必要があります。

ユーザから検索テキストを受け取ります。

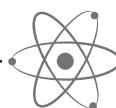
ブログエントリを順にループします。

各ブログエントリにマッチするテキストがあるかチェックします。

マッチするエントリがあったらループを抜け出します。

ブログ内を検索 7x7x7

設計はこれでいいと思うけど、
ブログエントリのテキストを検索するには
どうしたらいいのかしら？ 難しそうだけど、
きっとできるはずよね！



頭の体操

YouCubeのブログエントリを文字列検索できる
ようにするには、どうしたらいいでしょう？



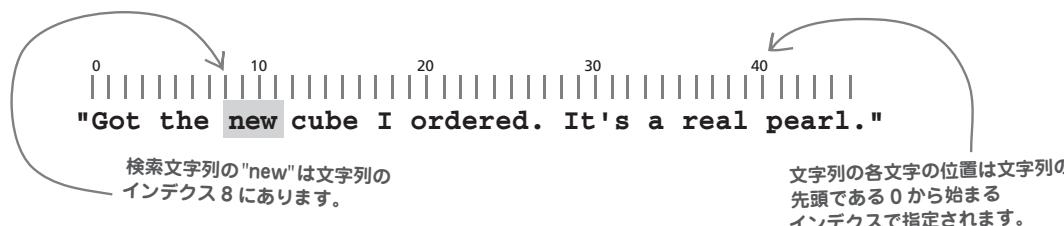
文字列内を検索するにはindexOf()

`indexOf()` は `String` オブジェクトの中の部分文字列を検索します。
`String` オブジェクトのメソッドであり、引数として部分文字列を渡します。
部分文字列が含まれていれば、その開始位置を返し、なければ-1を返します。

```
var str = "Got the new cube I ordered. It's a real pearl.";
alert( str.indexOf("new") );
```



この例では返値が 8 ですが、文字列を文字の配列のようなものと考えると、理解しやすいでしょう。



`indexOf()` を使って文字列を検索するとき、文字列に部分文字列が含まれないと結果は-1になります。

```
var searchIndex = str.indexOf("used");
```

Stringオブジェクトの中に検索文字列がないので結果は-1になります。



以下の文はルビーが好きななぞなぞです。この文字列の中で部分文字列 "cube" が現れるインデクスを記入してください。

"A cubist cubed two cubes and ended up with eight. Was she Cuban?"



エクササイズ
答え

以下の文はルビーが好きななぞなぞです。この文字列の中で部分文字列 "cube" が現れるインデックスを記入してください。

文字列の先頭の
インデックスは 0 です。

これは違
いますよね？

"A cubist cubed two cubes and ended up with eight. Was she Cuban?"

インデックスは
9 です。

インデックスは
19 です。

ブログ配列を検索する

StringオブジェクトのindexOf()のおかげで、文字列検索はさほど難しくなくなりましたが、ルビーはブログ全体を検索する必要があります。彼女が考えは、ブログエントリの配列をループでまわり、各ブログエントリの本文テキストに対してindexOf()を使って部分文字列を検索する、というものです。もし部分文字列が含まれていたら、そのブログエントリをアラートボックスで表示します。

実際のブログ検索を処理する関数を書く前に、YouCubeブログには検索テキストの入力フィールドと検索を開始するためのボタンが必要です。

```
<input type="button" id="search" value="Search the Blog" onclick="searchBlog(); " />
```

```
<input type="text" id="searchtext" name="searchtext" value=" " />
```

検索テキストはsearchtextという
IDでアクセスできます。

検索テキスト！

ブログ内を検索

7x7x7 ←



検索のためのHTML要素ができたら、あとはsearchBlog()のためのコードを書くだけです。この関数は検索結果をアラートを使って表示するので、値を返す必要はありません。また検索テキストをHTMLのテキストフィールドから直接読み取るので、引数も必要ありません。



JavaScript マグネット

YouCubeのsearchBlog()はブログエントリの配列をループでまわり、マッチするテキストをブログ本体から探します。マグネットで埋めて関数を完成させてください。ヒント：表示する検索結果は、日付をMM/DD/YYの形式にして角括弧で囲み、その後にブログの本文テキストを続けます。

```
function searchBlog() {
    var ..... = document.getElementById(".....").value;
    for (var i = 0; i < ..... ; i++) {
        // ブログエントリに検索テキストが含まれているか調べます
        if (blog[i]. ..... .toLowerCase().indexOf(searchText.toLowerCase()) != -1) {
            alert("[ " + (blog[i]. ..... . ..... + ..... ) + "/" +
                blog[i].date.getDate() + "/" + blog[i]. ..... .getFullYear() + " ] " +
                blog[i]. ..... );
            break;
        }
    }
    // 検索テキストが見つからないのでメッセージを表示します
    if (i == ..... )
        alert("検索テキストを含むエントリは見つかりません。");
}
```

searchText

blog.length

date

searchtext

body

1

getMonth()



JavaScriptマグネットの答え

YouCubeのsearchBlog()はブログエントリの配列をループでまわり、マッチするテキストをブログ本体から探します。マグネットを埋めて関数を完成させてください。ヒント：表示する検索結果は、日付をMM/DD/YYYYの形式にして角括弧で囲み、その後にブログの本文テキストを続けます。

HTMLテキストフィールドから検索テキストを取り出します。

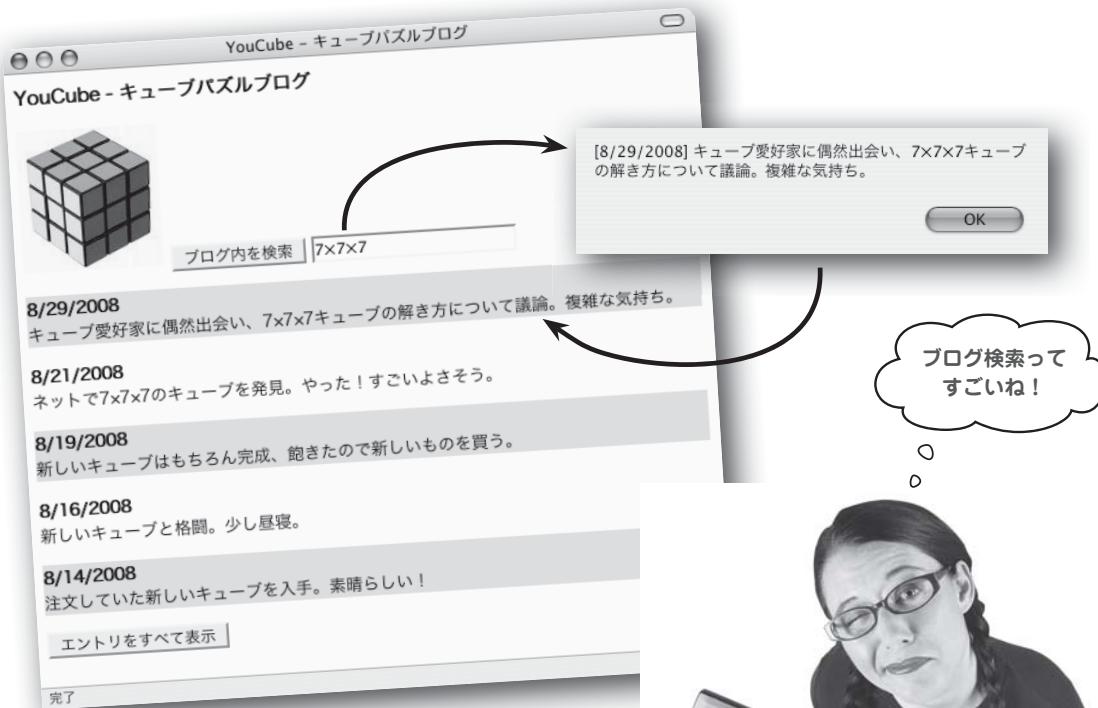
```
function searchBlog() {
    var searchText = document.getElementById("...").value;
    for (var i = 0; i < ...blog.length; i++) {
        // ブログエントリに検索テキストが含まれているか調べます
        if (blog[i]...body...toLowerCase().indexOf(searchText.toLowerCase()) != -1) {
            alert("[" + (blog[i]...date...getMonth() + 1) + "/" +
                blog[i].date.getDate() + "/" + blog[i]...date...getFullYear() + "] " +
                blog[i]...body...);
            break;
        }
    }
    // 検索テキストが見つからないのでメッセージを表示します
    if (i == ...blog.length)
        alert("検索テキストを含むエントリは見つかりません。");
}
```

iがblog配列の大きさと等しければ、
forループすべてのブログエントリを
調べたけれどもマッチするエントリが
なかったことを意味します。

マッチしたブログエントリは、
日付をMM/DD/YYYYの
形式にして角括弧で囲み、
その後に本文テキストが続けます。

検索も動くようになりました！

完成したYouCube 2.0 の検索機能は String オブジェクトに組み込まれている検索機能に深く依存しています。オブジェクトがデータをアクティブにすることを示す良い事例です。この場合、テキスト文字列（純粋なデータ）を動作をもつ（自分自身を検索できる）モノに変えています。なにより重要なのは、ルビーは自分で検索ルーチンを考えずにすむので、ブログの書き込みに専念できることです。



ルビーはブログの新機能にゾクゾクしていますが、成功に酔いしれているだけではありません。YouCube 3.0 を構想しています。



素朴な疑問に 答えます

Q：文字列が実はオブジェクトだ
というのが、さっぱり理解で
きません。ほんとにオブジェクトなん
ですか？

A：はい、そうなんです。Java
Scriptでは、あらゆるひとつの
文字列はオブジェクトです。JavaScript
コードでは、"Ruby"のように、あなたの
名前を引用符で囲むとオブジェクト
を作成したことになります。これはやり
すぎじゃないの、と思われるかもしれません
が、JavaScriptにおいて文字列
がオブジェクトとして扱われることは、
文字列の長さを調べる、部分文字列を
探す、といった、さまざまな便利な能
力が文字列にはあることを意味します。

Q：文字列がオブジェクトだとい
うのはわかりましたが、これ
は文字を要素とする配列と似ている
ように思います。文字列は配列では
ないのでしょうか？

A：それは違います。文字列はけ
して配列ではありません。し
かし、Stringのメソッドの多くは、
ちょうど文字を要素とする配列である
かのように、文字列データを操作しま
す。たとえば、文字列の中の文字はイ
ンデックス0から始まり、1文字ごとにイ
ンデックスも増えていきます。しかし、文
字列の中の文字を配列のように角括弧
([])でアクセスすることはできません。

前回の検索インデックスをindexOf()
に指定することで、文字列全体に対し
て検索を継続することができます。

Q：searchBlog()でtoLowerCase
Case()を2箇所で呼び出し
ている目的は何ですか？

A：いい質問ですね。この答は、
ブログにあるテキストを検索する
ときの大文字小文字の問題と関係して
います。誰かが"cube"という語でブ
ログを検索したら、"cube"、"Cube"、
"CUBE"など、大文字小文字の区別なく、
それらすべてと一致させたいと思う
でしょう。この問題は、探している部
分文字列とブログの本文の両方を小文
字か大文字のどちらかに変換すると簡
単に解決します。searchBlog()は
toLowerCase()を使っていますが、
toUpperCase()も使えます。検索の要
点は大文字小文字の区別をなくす
ことにあります。

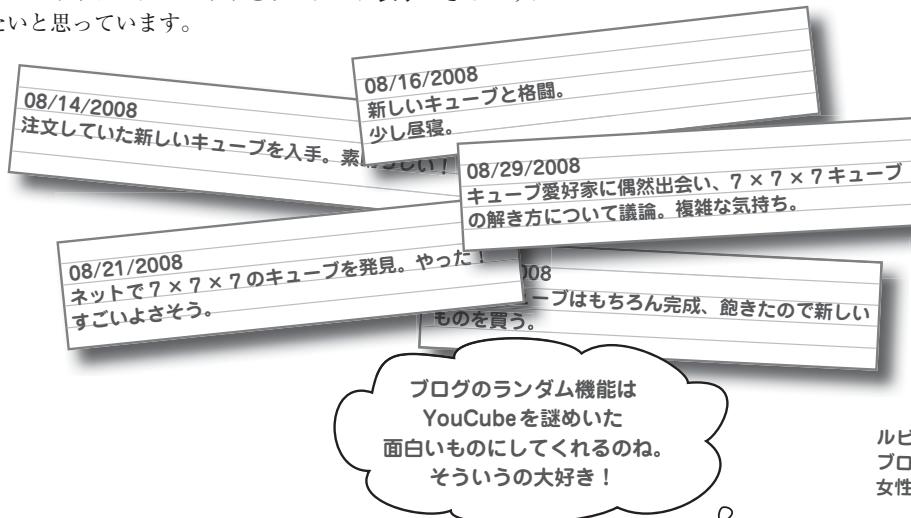


重要ポイント

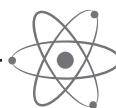
- `toString()`はオブジェクトをテキスト表現に変換するのに使います。
- 配列と文字列はどちらもオブジェクトです。`JavaScript`の標準オブジェクトである`Array`と`String`がメソッドとデータ記憶領域を提供します。
- `Array`オブジェクトの`sort()`は配列を思い通りの順序でソートするときに使います。
- `String`オブジェクトの`indexOf()`は文字列の中からあるテキスト文字列を探し、検索文字列の開始位置のインデックスを返します。

YouCubeをランダムにする

ユーザが彼女のブログに関心を持ち続けるように、パズル仲間が楽しめる機能をYouCubeに追加しようとルビーは思いました。「ランダム表示」ボタンを追加して、ユーザがブログエントリをランダムに表示できるようにしたいと思っています。



ルビーはキューブパズルのブロガー。謎が大好きな女性です。

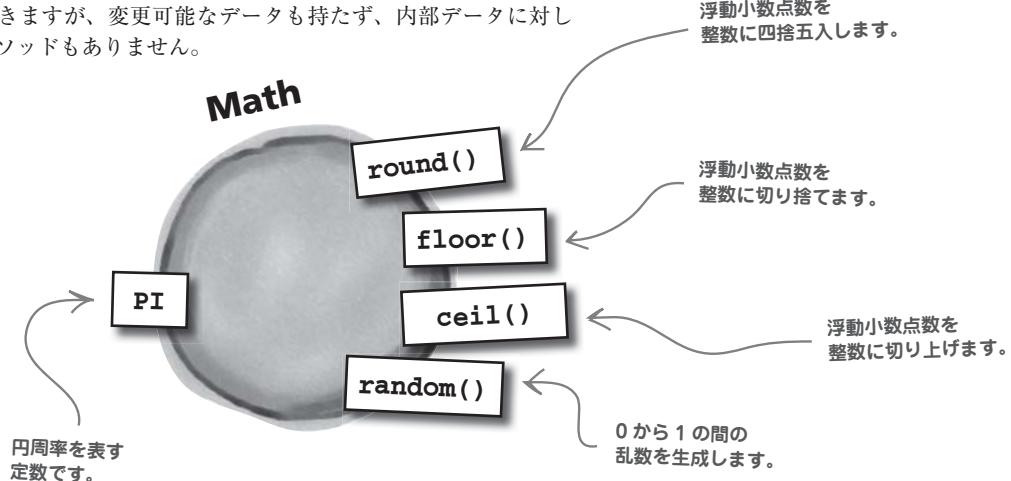


頭の体操

YouCubeのブログエントリをランダムに選択するにはどうすればいいでしょう？

Mathオブジェクトは編成されたオブジェクト

YouCubeにランダム機能を追加するには、乱数を生成する方法が必要です。そのためにはJavaScriptの組み込みオブジェクトを使うことになりますが、標準であるMathオブジェクトはこれまで使ってきた他のオブジェクトと違っています。Mathオブジェクトは、たとえば乱数を生成することができますが、変更可能なデータも持たず、内部データに対して行われるメソッドもありません。



Mathオブジェクトは数学関係のメソッドと定数を集めて編成されたオブジェクトです。変数がないので、Mathオブジェクトには状態がありません。データを格納することもできません。PI (3.14)などの定数がいくつか含まれているだけです。とはいえ、Mathオブジェクトのメソッドは、とても便利です。たとえば、random()は0から1の間の乱数を浮動小数点数で生成します。

Mathオブジェクトは
数学関係のメソッドと
定数を格納するように
編成されたオブジェクトです。



エクササイズ

以下のMathメソッドを呼び出したときの結果を書いてください。

```
Math.round(Math.PI)  
Math.ceil(Math.PI)  
Math.random()
```

.....
.....
.....

→ 解答は 436 ページ。



Mathオブジェクトの真実

今週のインタビュー：数学関数を直撃

Head First：ちょっと混乱しているんですが、あなたはオブジェクトなのに、実際はたくさんの数学メソッドとわずかな定数を持っているだけだそうですね。オブジェクトの要点は、データをアクティブにすることだと思っていたのですが。いくつかのデータを包み、それを使ってかっこいい処理を行うメソッドも持っているのがオブジェクトですよね。

Math：型にはまったJavaScriptはそうなんですが、しかしすべてのオブジェクトがデータに命を吹き込むわけではないんです。私みたいなまとめ役のオブジェクトだっているんですから。

Head First：これらの数学メソッドは標準関数としては作られていませんよね？

Math：はい、そうなんですが、JavaScript言語はオブジェクトで作られていることを忘れていませんか？ 実際には「標準関数」なんて存在ないんですから。

Head First：でもオブジェクトの外に関数を作ることができて、それはちゃんと動いているようにみえます。

Math：たしかに。でも実際には、すべての関数はなんらかのオブジェクトに属しているメソッドなんです。そのオブジェクトは隠れていますがね。「標準関数」が存在しないのは、そういう意味です。

Head First：そうなんですか。とすると、あなたが数学メソッドを持っている意味を知りたいですね。

Math：メソッドを使って操作する内部データを持っているないからといって、オブジェクトとして重要な役割を果たしていないのではないことを、忘れないでください。

Head First：どういう意味ですか？

Math：たとえばルービックキューブに対する関心を共有する人たちのグループを想像してみてください。多くの場合、そうした人々は自分たちの興味の対象について情報交換をするものです。数学メソッドはそ

うした人たちの輪と同じではありませんが、オブジェクトの中に集まっていることでプラスになることがあるんです。

Head First：つまり数学メソッドは関心を共有しているということですか？

Math：そうです。その関心とは、四捨五入、三角関数、乱数といった数学的な作業を実行するということです。

Head First：乱数について言及されましたか、あなたの乱数はほんとうの乱数ではないと聞いています。噂はほんとうですか？

Math：それについては認めざるをえませんね。コンピュータが生成する乱数は、ほとんどがそうなんですが。私の乱数は「疑似乱数」なんですが、でも実際にはそれで十分なんです。

Head First：疑似乱数って、疑似科学とか、疑似コードとかと同じですか？

Math：うーん、どっちにもとれますね。疑似科学ではありませんが、疑似コードにはすこし似ているところあります。疑似コードは、実際のコードは使わずにコードの背後にある考えを表現するためにあります。疑似乱数は、ほんとうの乱数ではないけれども、乱雑さを近似したものです。

Head First：つまり、ほとんどのJavaScriptアプリケーションで使うのに十分な乱雑さをもっていると思っていいんですね？

Math：はい、十分な乱雑さにするいい方法があります。国家機密レベルには使えませんが、日常のスクриプトで使うには十分ですよ。

Head First：了解しました。乱数のこと正直に答えていただいて、ありがとうございました。

Math：どういたしまして。嘘じゃないって、わかつたでしょう？



**エクササイズ
答え**

定数PIの値は
3.14です。

以下のMathメソッドを呼び出したときの結果を書いてください。

Math.round(Math.PI)

Math.ceil(Math.PI)

Math.random()

3.14が3になります。

3

4

3.14が切り上げで
4になります。

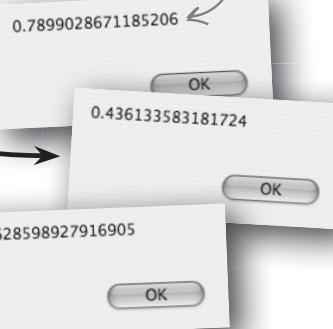
これはひっかけでした。
乱数なので結果は
わかりません。

Math.randomを使って乱数を生成する

疑似乱数かどうかの問題はともかくとして、Mathオブジェクトのrandom()が生成する乱数は、データの集まりからランダムにひとつ選択する機能を必要とするYouCubeのようなアプリケーションには最適です。問題は、random()が返す乱数は0から1の間の数なので、ルビーが必要とする0からブログ配列の最後までの間ではないことです。彼女が必要としているのは、ブログのランダムなインデックスなのです。

```
alert(Math.random());
alert(Math.random());
alert(Math.random());
```

これらの乱数は0から
1の間の数になります。



0と1の間ではない乱数を生成する場合、Mathオブジェクトのもうひとつのメソッドを組み合わせる必要があります。floor()は小数を切り捨てて整数にするメソッドです。これを使えば、与えられた範囲内の整数で乱数を生成することができます。



0 - 5
1 - 6

```
var oneToSix = Math.floor(Math.random() * 6) + 1;
```

Q: Mathオブジェクトを使う前に作成しておく必要がないのはなぜですか?

A: オブジェクトに関するとても重要なポイントをつく、鋭い質問ですね。Mathオブジェクトにはインスタンスデータと呼ばれる変更可能なデータが含まれていないので、オブジェクトを作成する必要はありません。Mathオブジェクトは静的メソッドと定数の集まりにすぎないので、Mathオブジェクトに必要なものはすべて揃っており、新たに作成すべきものは存在しません。オブジェクトのインスタンスとクラスに関しては10章で詳しく説明します。

Q: Mathオブジェクトの
round()とfloor()の違い
は何ですか?

素朴な疑問に 答えます

All: round()は小数部を四捨五入します。Math.round(11.375)は11、Math.round(11.625)は12になります。floor()は小数部の値に関係なく小数部を切り捨てます。

Q: Mathオブジェクトを使って
他に何ができますか?

A: たくさんありますが、まだ紹介していない便利なメソッドに `min()` と `max()` があります。これは 2 つの数値を比較して、前者は小さい方、後者は大きい方を返します。`abs()` も便利なメソッドです。受け取った数値を正数(絶対値)にして返します。



マニア向け情報

JavaScriptアプリケーションでどうしても本格的な乱数が必要な場合には、疑似乱数を越える方法を <http://random.org> で学習してください。



自分で考えてみよう

ブログエントリをランダムに選択し、結果をアラートボックスに表示する randomBlog() のコードを書いてください。

ヒント：アラートボックスのブログエントリはsearchBlog()の検索結果と同じ形式で表示します。

自分で考えてみよう の答え

乱数を使って
ブログエントリを
選択します。

ブログエントリをランダムに選択し、結果をアラートボックスに表示する randomBlog() のコードを書いてください。ヒント：アラートボックスの ブログエントリは searchBlog() の検索結果と同じ形式で表示します。

0 から blog 配列の
大きさまでの
間の乱数を生成します。

```
function randomBlog() {
    // 0 から blog.length - 1 の間の乱数を生成します
    var i = Math.floor(Math.random() * blog.length); ←
    alert("[" + (blog[i].date.getMonth() + 1) + "/" + blog[i].date.getDate() + "/" +
        blog[i].date.getFullYear() + "] " + blog[i].body); ←
}

```

ブログエントリは日付を MM/DD/YYYY 形式で
表示した後に本文が続きます。

ランダムだけでは十分じゃない

ルビーのブログにランダムな検索機能も加わり、彼女は満足しています。
ユーザはどのエントリが表示されるか予想がつかないので、いい意味で
好奇心をそそられています。

The screenshot shows a blog interface with a sidebar containing a list of posts:

- 8/29/2008 キューブ愛好家に偶然出会い、7x7x7キューブの解き方について議論。複雑な気持ち。
- 8/21/2008 ネットで7x7x7のキューブを発見。やった！すごいよ
- 8/19/2008 新しいキューブはもちろん完成、飽きたので新しいも
- 8/16/2008 新しいキューブと格闘。少し昼寝。
- 8/14/2008 注文していた新しいキューブを入手。素晴らしい！

At the bottom of the sidebar are two buttons: [エントリをすべて表示] and [エントリのランダム表示] (highlighted with a red box).

A modal dialog box is displayed, containing the text: [8/16/2008] 新しいキューブと格闘。少し昼寝。 and an OK button.

ランダムに選択された
ブログエントリ。

OK

ブログの新機能に興奮しつつも、ルビーは YouCube にまだ何か足りないものがある気がして仕方ありません。彼女の Blog オブジェクトは、いまのところプロパティの集まりにすぎず、ばらばらな関数の集まりに依存しています。これはあまり良いオブジェクト設計ではないように感じられます。

オブジェクトのアクションを追加

ルビーの YouCube オブジェクトに対する直感は大当たりです。オブジェクトの動作にかかわる部分が欠落しているので、ブログ特有の作業を行うメソッドを使うように、オブジェクトを再構成する余地があります。ルビーは Blog オブジェクトにいくつかのアクションを追加するメソッドを考えています。



自分で考えてみよう

```

function showBlog(numEntries) {
    // まずブログを降順(最新のものが先頭になるように)にソートします
    blog.sort(function(blog1, blog2) { return blog2.date - blog1.date; });

    // 表示するエントリの件数を調整します
    if (!numEntries)
        numEntries = blog.length;

    // ブログエントリを表示します
    var i = 0, blogText = "";
    while (i < blog.length && i < numEntries) {
        // ブログエントリの背景色をグレイにします
        if (i % 2 == 0)
            blogText += "<p style='background-color:#EEEEEE' >";
        else
            blogText += "<p>";

        // ブログのHTMLコードを生成します
        blogText += "<strong>" + (blog[i].date.getMonth() + 1) + "/" +
            blog[i].date.getDate() + "/" +
            blog[i].date.getFullYear() + "</strong><br />" +
            blog[i].body + "</p>";

        i++;
    }

    // ページにHTMLコードを設定します
    document.getElementById("blog").innerHTML = blogText;
}

function searchBlog() {
    var searchText = document.getElementById("searchtext").value;
    for (var i = 0; i < blog.length; i++) {
        // ブログエントリに検索テキストが含まれているか調べます
        if (blog[i].body.toLowerCase().indexOf(searchText.toLowerCase()) != -1) {
            alert("[ " + (blog[i].date.getMonth() + 1) + "/" + blog[i].date.getDate() + "/" +
                blog[i].date.getFullYear() + " ] " + blog[i].body);
            break;
        }
    }

    // 検索テキストにマッチしなかったらメッセージを表示します
    if (i == blog.length)
        alert("Sorry, there are no blog entries containing the search text.");
}

function randomBlog() {
    // 0からblog.length - 1 の間の乱数を生成します
    var i = Math.floor(Math.random() * blog.length);
    alert("[ " + (blog[i].date.getMonth() + 1) + "/" + blog[i].date.getDate() + "/" +
        blog[i].date.getFullYear() + " ] " + blog[i].body);
}

```

自分で考えてみよう の答え

```

function showBlog(numEntries) {
    // まずブログを降順（最新のものが先頭になるように）にソートします
    blog.sort(function(blog1, blog2) { return blog2.date - blog1.date; });

    // 表示するエントリの件数を調整します
    if (!numEntries)
        numEntries = blog.length;

    // ブログエントリを表示します
    var i = 0, blogText = "";
    while (i < blog.length && i < numEntries) {
        // ブログエントリの背景色をグレーにします
        if (i % 2 == 0)
            blogText += "<p style='background-color:#EEEEEE' >";
        else
            blogText += "<p>";

        // ブログのHTMLコードを生成します
        blogText += "<strong>" + (blog[i].date.getMonth() + 1) + "/" +
                    blog[i].date.getDate() + "/" +
                    blog[i].date.getFullYear() + "</strong><br />" +
                    blog[i].body + "</p>";

        i++;
    }

    // ページにHTMLコードを設定します
    document.getElementById("blog").innerHTML = blogText;
}

function searchBlog() {
    var searchText = document.getElementById("searchtext").value;
    for (var i = 0; i < blog.length; i++) {
        // ブログエントリに検索テキストが含まれているか調べます
        if (blog[i].body.toLowerCase().indexOf(searchText.toLowerCase()) != -1) {
            alert("[ " + (blog[i].date.getMonth() + 1) + "/" + blog[i].date.getDate() + "/" +
                  blog[i].date.getFullYear() + " ] " + blog[i].body);
            break;
        }
    }

    // 検索テキストにマッチしなかったらメッセージを表示します
    if (i == blog.length)
        alert("Sorry, there are no blog entries containing the search text.");
}

function randomBlog() {
    // 0から blog.length - 1 の間の乱数を生成します
    var i = Math.floor(Math.random() * blog.length);
    alert("[ " + (blog[i].date.getMonth() + 1) + "/" + blog[i].date.getDate() + "/" +
          blog[i].date.getFullYear() + " ] " + blog[i].body);
}

```

Blog.toHTML()
ブログエントリをHTMLコードに変換します。このメソッドがあればブログを整形して表示する必要のある他のコードで苦労せずにすみます。

Blog.containsText()
コードはあまりありませんが、ブログエントリが本文テキストを検索できるようにすべきなので、メソッドにする価値があります。

ブログエントリをHTMLに変換する処理をBlog.toHTML()にまとめます。

Blog.toString()
ブログエントリを文字列に変換します。日付が角括弧で囲まれた下に本文が続く場合にこれを使う意味があります。

素朴な疑問に 答えます

Q: スクリプトコードをメソッドにする手順はどうなるのでしょうか?

A: まず最初に、メソッドに何を実行させるのかを明らかにして、オブジェクトの状態(データ)をもとにアクションを考えます。ある程度までは、オブジェクトのメソッドを決めれば、オブジェクトが実際に何を行うのか、あるいは何を実行する必要があるのかわかります。次にオブジェクトに必要な処理が実行できるようにオブジェクトを強化します。

たとえば、Blogオブジェクトを文字列やHTMLコードに変換できるようにするのは意味があります。この2つのアクションは内部のブログデータにアクセスするからです。同様に、ブログエントリのテキスト検索も、Blogオブジェクトのメソッドにする意味があります。

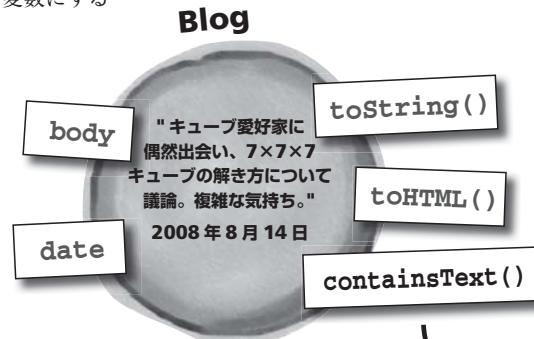
Q: では逆にBlogオブジェクトに持たせるべきでないアクションはありますか?

A: Blogオブジェクトのスコープの外にあるアクションとしてブログエントリのリストの表示や検索があげられます。Blogオブジェクトはひとつのブログエントリを表現するだけだからです。そのため複数のBlogオブジェクトで構成されるブログ配列があるわけです。個々のBlogオブジェクトは、Blogオブジェクトの集まりに関する情報を持つ必要がありません。個々のBlogオブジェクトは自分の仕事をするだけであり、エントリの日付と本文テキストをもとにアクションを実行します。

関数をメソッドに変える

YouCubeのコード断片のいくつかは孤立しているので、Blogオブジェクトのメソッドにするのに適しています。そのひとつをBlogのメソッドにする方法を詳しく見てみましょう。`containsText()`はブログエントリの本文に対して部分文字列を検索するメソッドです。検索関連のコードをメソッドにすると、Blogオブジェクトの`body`プロパティを直接扱うことができ、`searchBlog()`関数の中のローカル変数にする必要がありません。処理の手順は以下の通りです。

- ❶ `containsText()`メソッドを宣言し、検索テキストなど、必要となる引数のリストを追加する。
- ❷ 既存のコードを新しいメソッドに移す。
- ❸ `containsText()`の中の`this.body`のように、オブジェクトプロパティを使うコードに変更する。



自分で考えてみよう

Blogオブジェクトの`containsText()`のコードを書いてください。このメソッドは`Blog()`コンストラクタの中で`this.containsText`に関数リテラルを代入することで作成されます。

自分で考えてみよう の答え

このメソッドはメソッド参照に
関数リテラルを代入することで
作成されます。

```
this.containsText = function(text) {
    return (this.body.toLowerCase().indexOf(text.toLowerCase()) != -1);
};
```

キーワード `this` は、プロパティを作成するのに使われる
のと同様に、メソッドを作成するのに使われます。

Blog オブジェクトの `containsText()` のコードを書いてください。
このメソッドは `Blog()` コンストラクタの中で `this.containsText`
に関数リテラルを代入することで作成されます。

メソッドの中のコードはキーワード `this` を
使ってプロパティに直接アクセスします。

新しいブログオブジェクトのお披露目

`containsText()` の他に、さらに 2 つのメソッドが Blog オブジェクト
の新バージョンに含まれています。プロパティと動作の両方を兼ね備えてい
るわけです。

どうです、
綺麗なコード
になりましたよ！

```
function Blog(body, date) {
    // プロパティを代入します
    this.body = body;
    this.date = date;

    // ブログエントリの文字列表現を返します
    this.toString = function() {
        return "[" + (this.date.getMonth() + 1) + "/" + this.date.getDate() + "/" +
            this.date.getFullYear() + "] " + this.body;
    };

    // ブログエントリの HTML 表現を返します
    this.toHTML = function(highlight) {
        // 背景色をグレイにしてハイライト表示にします
        var blogHTML = "";
        blogHTML += highlight ? "<p style='background-color:#EEEEEE'>" : "<p>";
        blogHTML += "<strong>" + (this.date.getMonth() + 1) + "/" +
            this.date.getDate() + "/" + this.date.getFullYear() + "</strong><br />" +
            this.body + "</p>";
        return blogHTML;
    };

    // ブログ本文にテキスト文字列が含まれているか調べます
    this.containsText = function(text) {
        return ((this.body.toLowerCase()).indexOf(text.toLowerCase()) != -1);
    };
}
```

プロパティを作成し
初期化します。

`toString()` はブログエントリを
文字列テキストにして返します。

`toHTML()` はブログエントリを
HTML コードにして返します。

`containsText()` は本文テキストに検索テキスト
が含まれていたら `true` を返し、含まれ
ていなければ `false` を返します。

オブジェクトが YouCube にもたらしたもの

Blog オブジェクトの新バージョンが YouCube のスクリプトに導入されて、はじめてオブジェクト指向プログラミングの利点が明らかになりました。いくつかの重要なブログ特有の作業が Blog メソッドに移譲されたので、スクリプトのコードは見違えるほど簡潔になりました。

新しい Blog オブジェクトは YouCube スクリプトを簡潔にします。

```
// ブログエントリのリストを表示します
function showBlog(numEntries) {
    // まずブログを降順に（最新が先頭になるように）ソートします
    blog.sort(function(blog1, blog2) { return blog2.date - blog1.date; });

    // 表示するブログエントリの数を調整します
    if (!numEntries)
        numEntries = blog.length;

    // ブログエントリを表示します
    var i = 0, blogListHTML = "";
    while (i < blog.length && i < numEntries) {
        blogListHTML += blog[i].toHTML(i % 2 == 0); // toHTML() はブログエントリを
        i++;                                         // HTML コードに整形します。
    }

    // ブログの HTML コードをページに設定します
    document.getElementById("blog").innerHTML = blogListHTML;
}

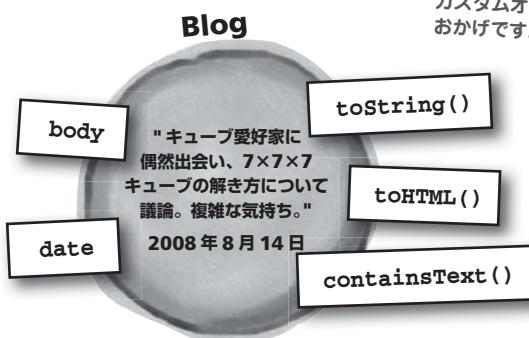
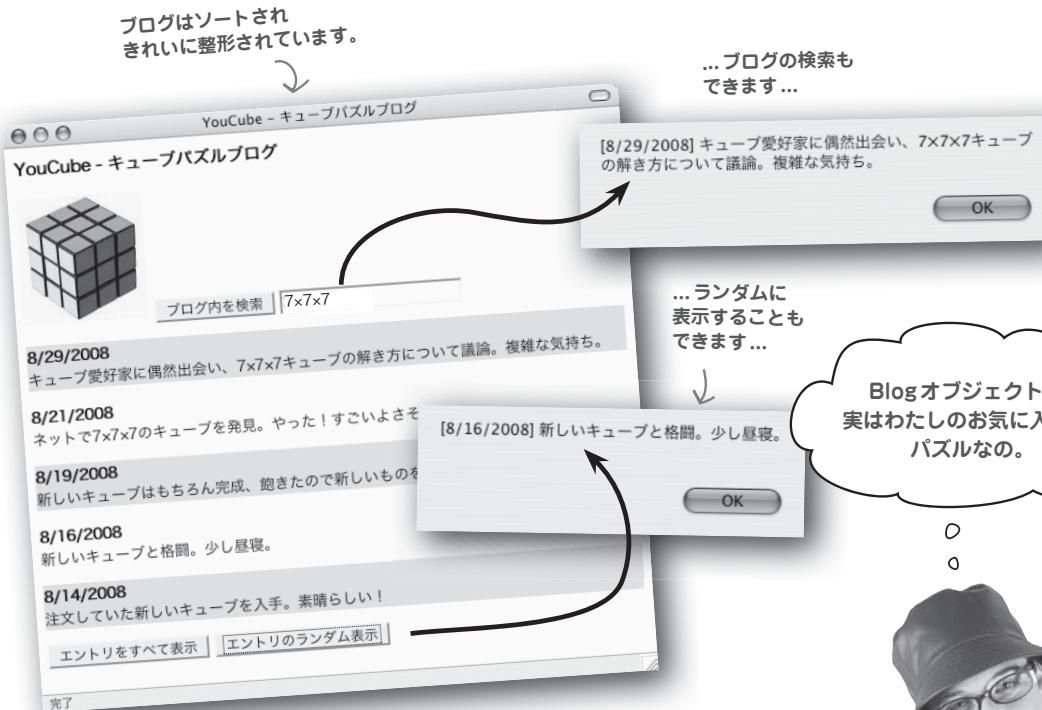
// ブログエントリのリストからテキスト断片を検索します
function searchBlog() {
    var searchText = document.getElementById("searchtext").value;
    for (var i = 0; i < blog.length; i++) {
        // ブログエントリに検索テキストが含まれているか調べます
        if (blog[i].containsText(searchText)) {
            alert(blog[i]); // containsText() は
            break;          // ブログエントリから
        }                  // 部分文字列を探します。
    }

    // 検索テキストが含まれていなかったらメッセージを表示します
    if (i == blog.length)
        alert("検索テキストを含むエントリは見つかりません。");
}

// ブログエントリをランダムに選んで表示します
function randomBlog() {
    // 0 から blog.length - 1 の間の乱数を生成します
    var i = Math.floor(Math.random() * blog.length);
    alert(blog[i]); // ブログエントリが文字列として扱われる
}                                // 文脈で使われるとき toString() が
                                 // 自動的に呼び出されます。
```

YouCube 3.0!

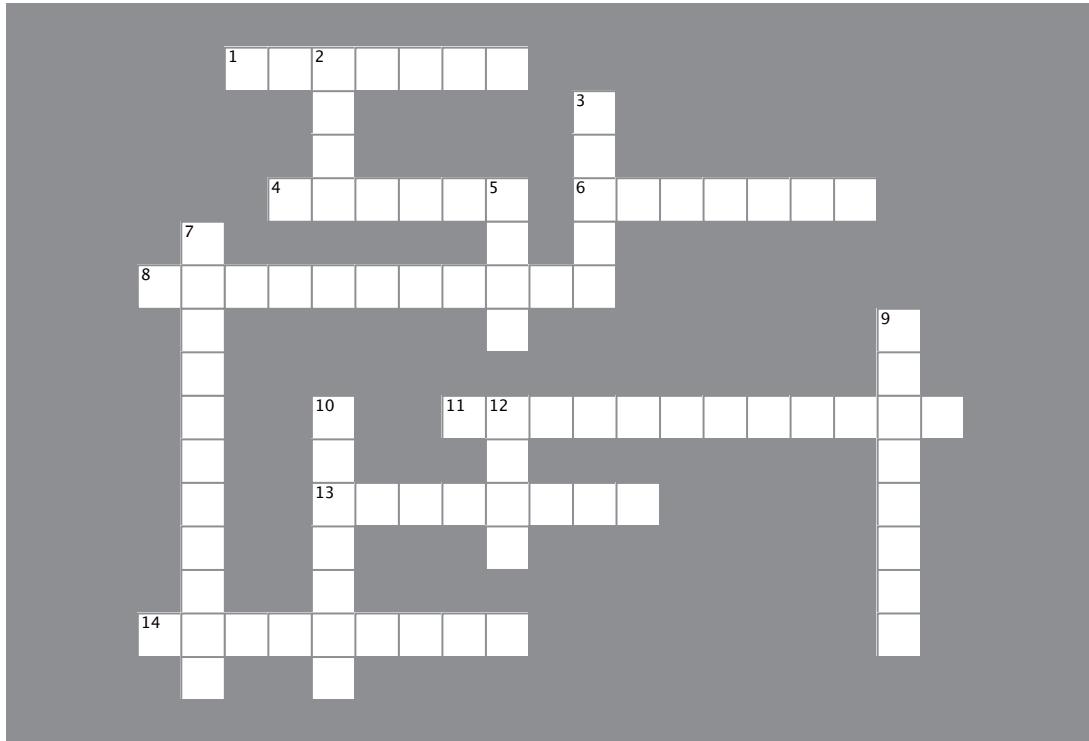
なかなかのプロジェクトでしたが、YouCube 3.0 のできばえに満足したルビーは、パズルを楽しむ時間をとって、招待されたパーティの準備に熱中しています。





JavaScriptクロスワード

ルビーが待ちに待ったパズルのページになりました。キューブパズルではなくクロスワードパズルですが、どうぞお楽しみください。



ヨコのかぎ

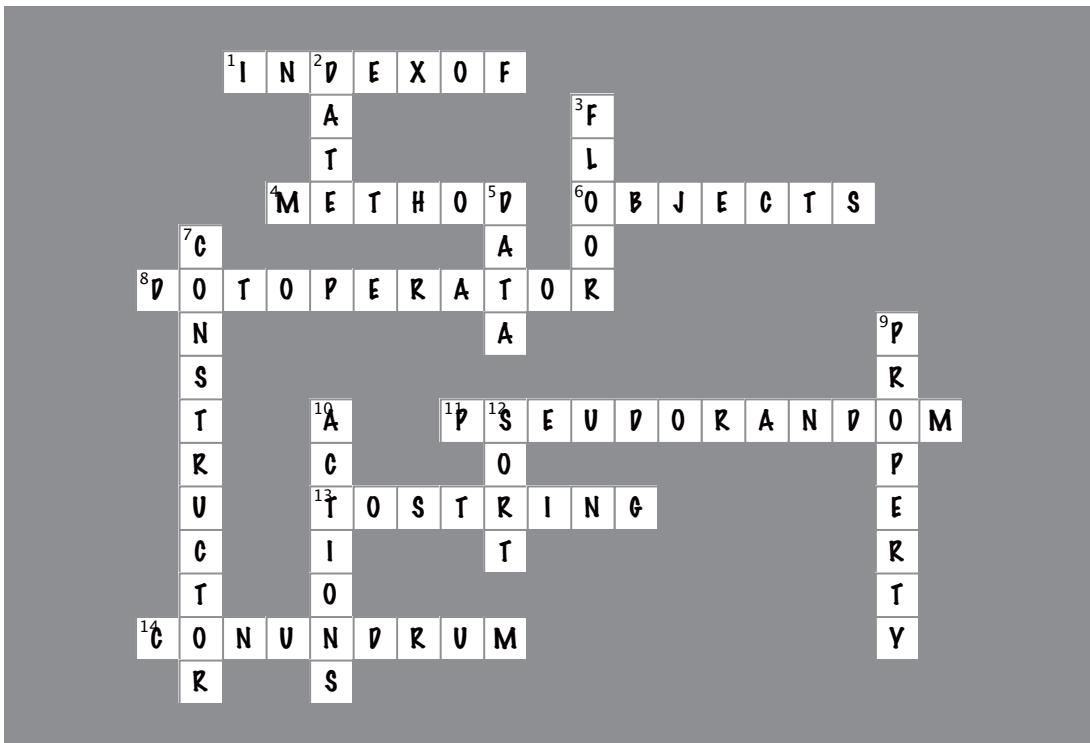
- テキスト文字列を検索するときこのStringメソッドを使います。
- オブジェクトに関数を置くと……になります。
- JavaScriptの配列と文字列は実は……です。
- オブジェクトのメンバーにアクセスするときこれをを使います。
- これはかなりランダムです。
- このメソッドはオブジェクトをテキスト文字列に変換します。
- ルビーのホームタウンといえば……。

タテのかぎ

- このオブジェクトを使って時間を処理します。
- このMathオブジェクトは数値を切り下げます。
- オブジェクトの中でプロパティはこれを格納します。
- オブジェクトのプロパティはここで作成されます。
- オブジェクトの中のデータ断片。
- メソッドはオブジェクトにこれをさせます。
- このメソッドを呼び出して配列の項目の順序を変えます。



JavaScript クロスワードの答え

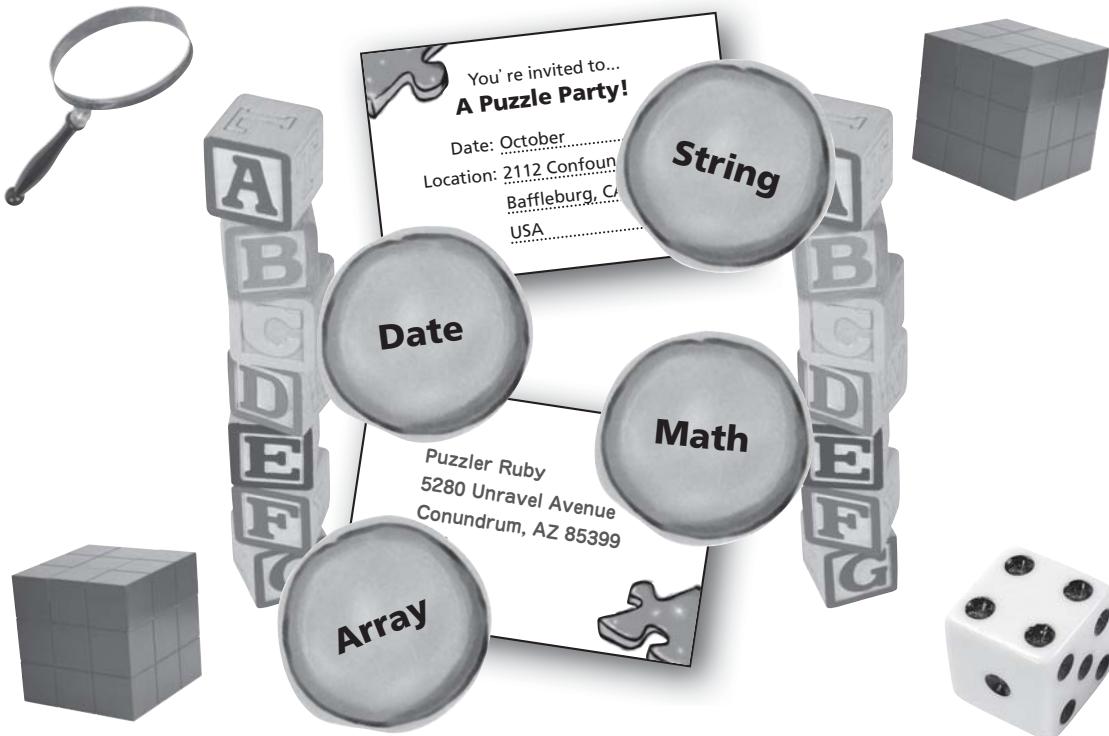


折り畳みページ

脳のところで
折り畳んでから
謎を解決しましょう。

JavaScriptのオブジェクトは
そのデータを使って何ができますか？

左脳と右脳がご対面！



あなたがしたいことを探しましょう。

データをソートしたり分析したりするのにJavaScriptのオブジェクトほど
適したものはないことがわかるでしょう。何のトラブルもなく
数値を乱数にすることだってできるのです。

10章 カスタムオブジェクトを作成する

カスタムオブジェクトを 思い通りに

さあ、いかがですか。いますぐ
直接注文いただければ、返金保証は
もちろん、たった1ドルです。
ご都合にあわせてご利用
いただけますよ。



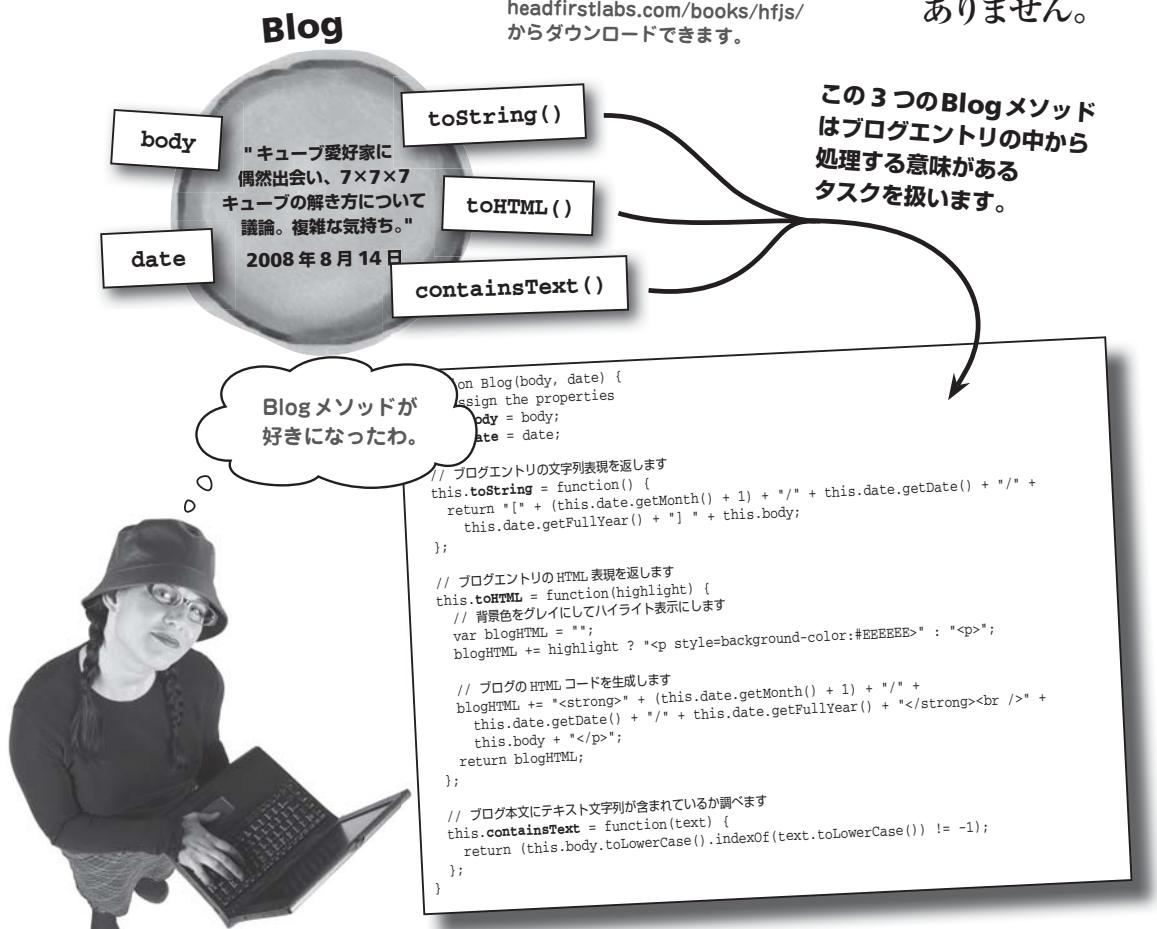
JavaScriptには返金保証はありませんが、好きなように利用できます。JavaScriptのカスタムオブジェクトは、スタバでコーヒーを注文するようなものです。キャラメルマキアートのホット、サイズはグランデ、バニラシロップとフォームミルク……。はい、カスタムコーヒーの出来上がりです。JavaScriptのカスタムオブジェクトを使えば、プロパティとメソッドの利点を生かしながら、あなたの思い通りの処理を行うコードを作ることができます。再利用可能なオブジェクト指向のコードによって、拡張されたJavaScript言語があなたのものになります。

YouCubeのBlogメソッドを再考する

前回ルビーに会ったとき、彼女はキューブパズルの楽しみについて書くためにオブジェクト対応のブログを作ることに熱中していました。YouCubeブログを動かすBlogオブジェクトをきれいに作ったのですが、YouCubeにオブジェクト指向の方式を適用する際に、いくつかの重要なチャンスを見落としてしまいました。Blogオブジェクトがもっと効率的で整理されたものになり将来の維持管理が楽になる、そんな方法があるのに十分に検討できなかったのです。

Blogオブジェクトを最後に調整したとき、ブログ特有の作業を処理する3つのメソッドを作りました。

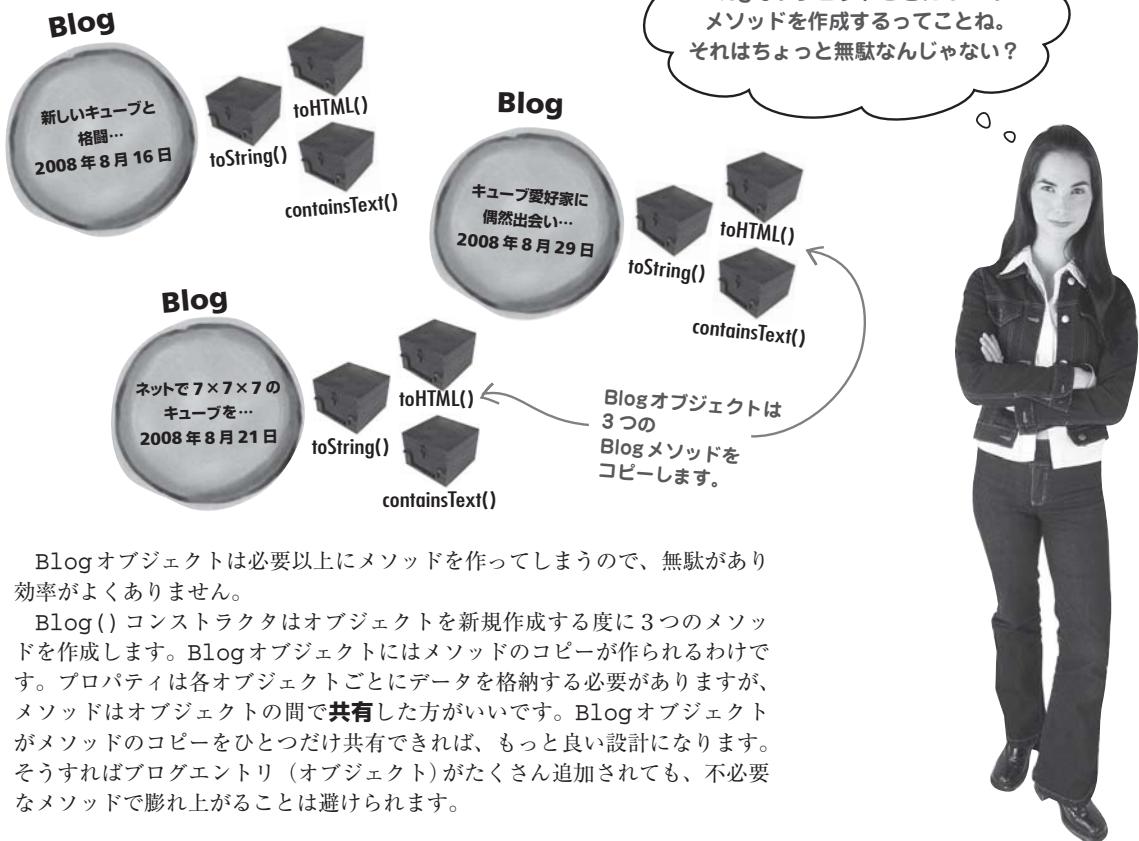
最新のファイルは <http://www.headfirstlabs.com/books/hfjs/> からダウンロードできます。



YouCubeのメソッドは良さそうに見えますが、ちょっとした問題があります。

メソッドのオーバーロード

Blogオブジェクトのメソッドは、ブログプロパティと同様、コンストラクタの中で `this`キーワードを使って作成されます。このアプローチだと、Blogオブジェクトが作成される度にメソッドの新しいコピーが作成されることになります。ブログに6つのエントリがあるときは、3つのBlogメソッドが6回コピーされるわけです。



Blogオブジェクトは必要以上にメソッドを作ってしまうので、無駄があり効率がよくありません。

Blog()コンストラクタはオブジェクトを新規作成する度に3つのメソッドを作成します。Blogオブジェクトにはメソッドのコピーが作られるわけです。プロパティは各オブジェクトごとにデータを格納する必要がありますが、メソッドはオブジェクトの間で**共有**した方がいいです。Blogオブジェクトがメソッドのコピーをひとつだけ共有できれば、もっと良い設計になります。そうすればブログエントリ（オブジェクト）がたくさん追加されても、不必要的メソッドで膨れ上がることは避けられます。

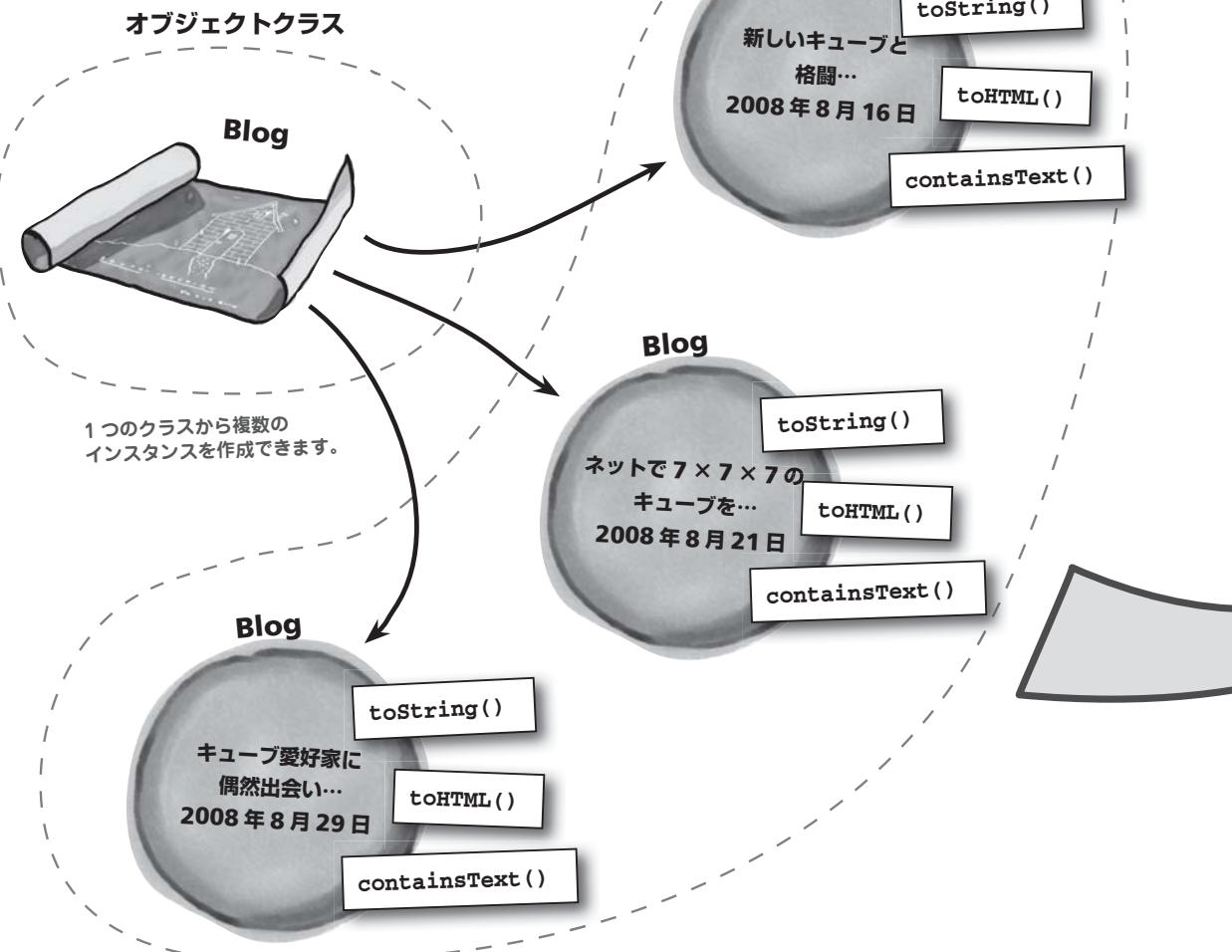


頭の体操

新しいオブジェクトでメソッドのコードが重複しないようにBlogオブジェクトを再設計できますか？

クラス×インスタンス

メソッドが重複する問題は、JavaScriptオブジェクトに関する重要な概念であるオブジェクトクラスとオブジェクトインスタンスの違いに関わっています。クラスはオブジェクトの記述であり、オブジェクトの構成要素を示すテンプレートです。インスタンスは実際のオブジェクトであり、クラスから作成されます。現実世界で喻えると、クラスは家の設計図であり、オブジェクトは家そのものです。JavaScriptのオブジェクトと同様に、ひとつのクラス（設計図）からたくさんの家のインスタンスを作られます。

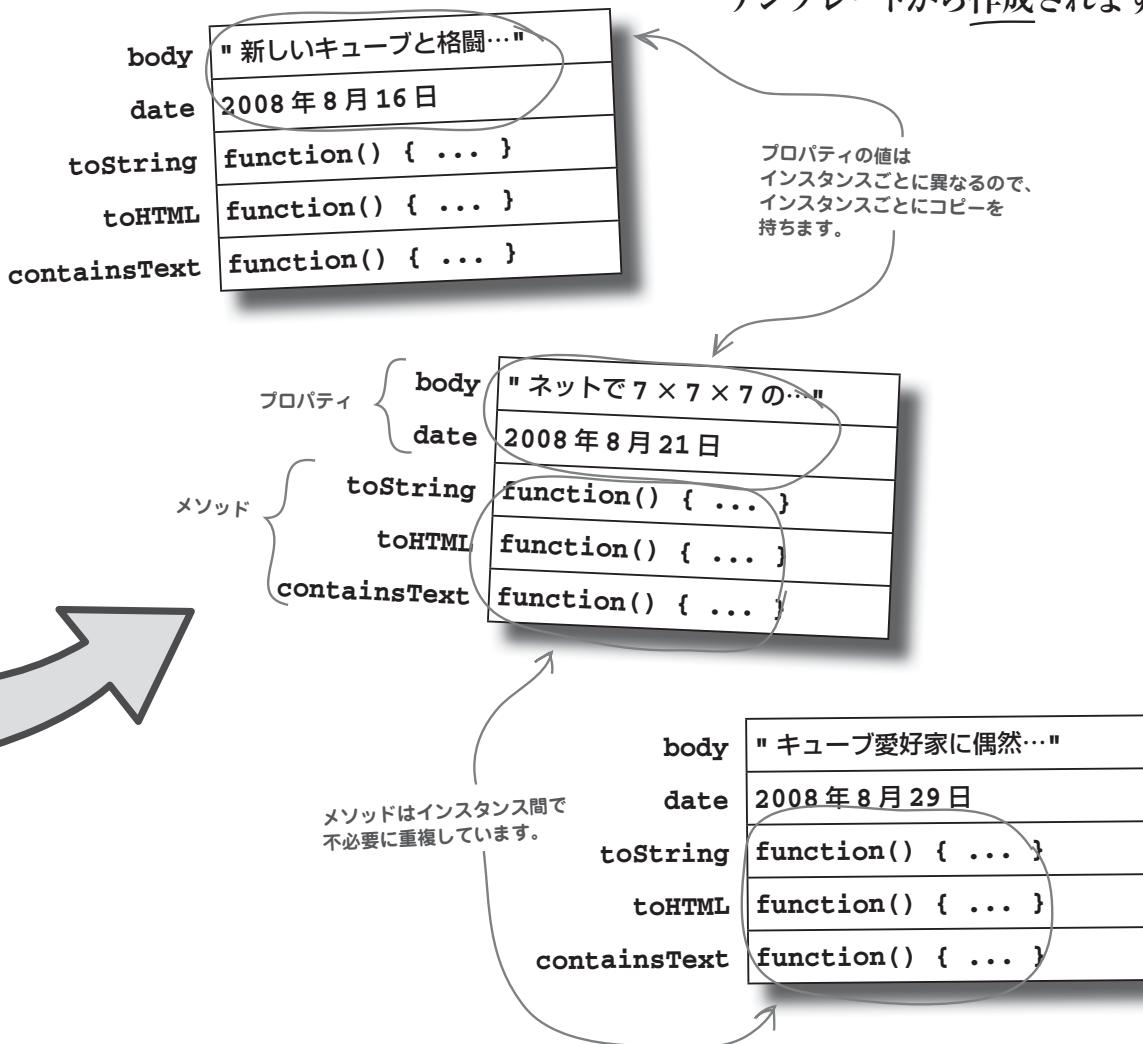


インスタンスはクラスから作成される

クラスはオブジェクトのプロパティとメソッドを記述します。インスタンスはプロパティに実際のデータを代入し、動作できるものになります。各インスタンスごとにプロパティのコピーがあるので、それぞれのインスタンスを区別することができます。

オブジェクトクラスは
テンプレートです。

オブジェクトのインスタンスは
テンプレートから作成されます。



thisを使ってインスタンスのプロパティにアクセスする

ここまで扱ってきたプロパティはインスタンスプロパティです。つまりインスタンスが持っています。ここで重要なのはインスタンスごとにコピーを持っていることです。インスタンスプロパティはコンストラクタの中でthisを使って設定されるので簡単に識別できます。

```
function Blog(body, date) {  
    this.body = body;  
    this.date = date;  
    ...  
}
```

キーワードthisを
使って参照されているので
インスタンスプロパティです。

インスタンスマетодはインスタンスまたはクラスが持つので、ちょっと巧妙になります。これまで作ってきたのはインスタンスマethodです。thisを使って設定したのでインスタンスがmethodを持っています。このためmethodのコードがインスタンスごとに重複するわけです。

```
function Blog(body, date) {  
    ...  
    this.toString = function() {  
        ...  
    }  
    this.toHTML = function() {  
        ...  
    }  
    this.containsText = function() {  
        ...  
    }  
}
```

Blogインスタンスごとに
これらのmethodは
コピーされます。

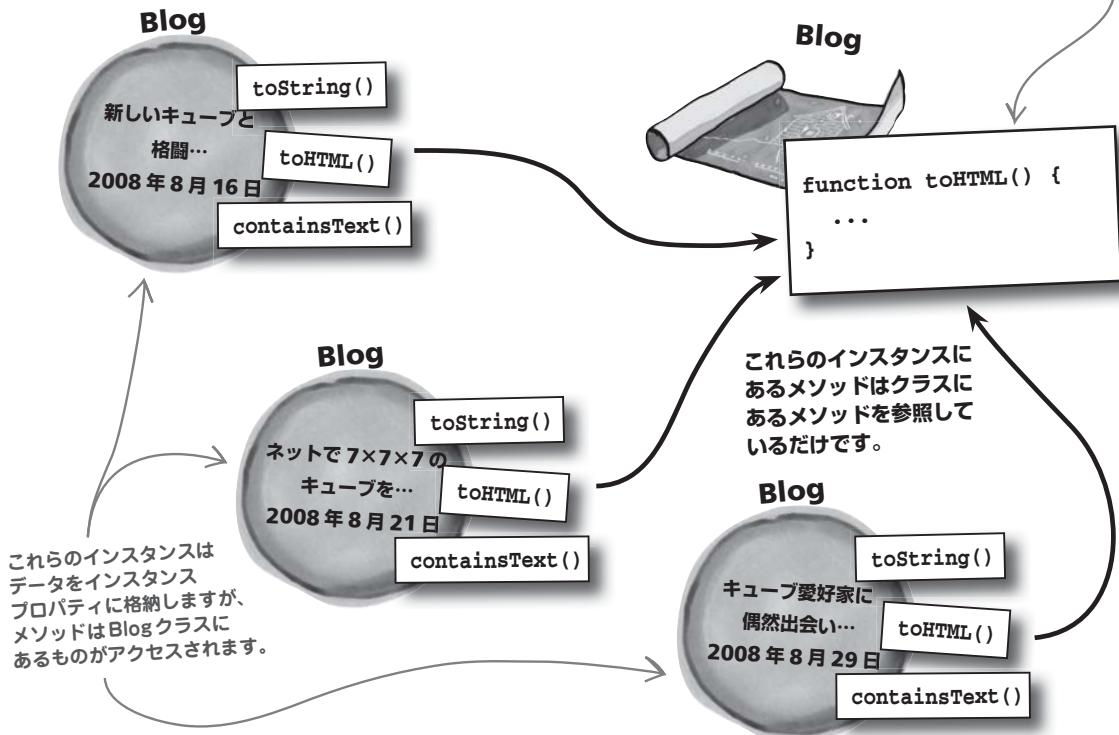
コンストラクタの中でキーワード
thisが使われているので
インスタンスマethodです。

カスタムオブジェクトでは、新しいインスタンスごとにmethodのコードを無駄に重複してしまうのを絶対に避けられないわけではありません。methodのコードのコピーをひとつ作り、それをインスタンスが共有するようにmethodを作ることも可能なのです。

クラスが持つメソッドは、ひとつのメソッドを共有して実行できます

インスタンスメソッドにはもうひとつ種類があります。クラス自身が持つインスタンスメソッドがあり、すべてのインスタンスでひとつのコピーを共有します。クラス所有のインスタンスメソッドは、インスタンスごとにメソッドのコピーを格納するよりはるかに効率的です。

このメソッドはBlogクラスが所有するので、インスタンスはコピーを持つ必要がありません。



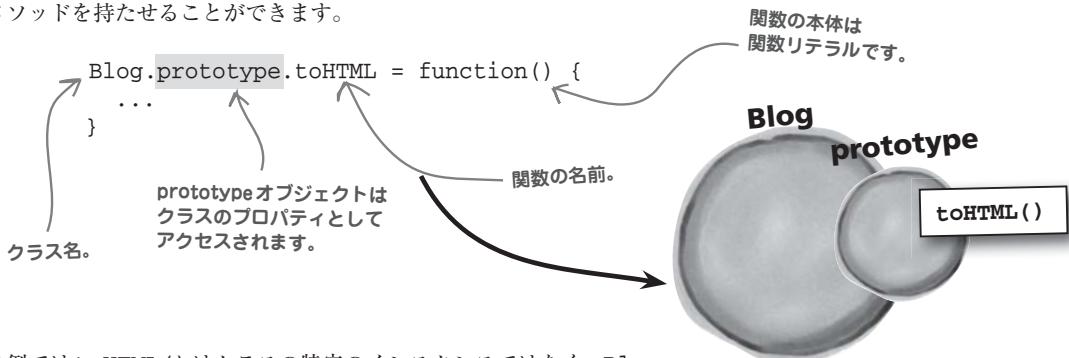
クラスがメソッドを持つ場合、クラスのすべてのインスタンスがこのメソッドにアクセスできるので、インスタンスごとにコピーを持つ必要がありません。アプリケーションがたくさんのオブジェクトインスタンスを作成する場合、メソッドが何度もコピーされ、メモリを浪費することを考えると、このアプローチの方が効率が優れています。YouCubeの場合、ブログエントリが作成される度に3つのメソッド (`toString()`、`toHTML()`、`containsText()`) が不必要に重複しています。

メソッドを個々のインスタンスごとに持たせるのではなく、メソッドの持ち主をクラスに変えるための仕掛けが必要です。

クラスにメソッドを格納すると、ひとつのコピーをすべてのインスタンスが共有できます。

prototypeを使ってクラスレベルで動かす

JavaScriptのクラスは、prototypeと呼ばれる隠れたオブジェクトのおかげで、実現することができます。prototypeはすべてのオブジェクトにプロパティとして存在しています。prototypeオブジェクトを使うと、インスタンスの中ではなく、クラスレベルで所有されるプロパティとメソッドを設定することができます。メソッドの場合、以下のようにprototypeオブジェクトを使って、クラスにメソッドを持たせることができます。



この例では`toHTML()`はクラスの特定のインスタンスではなく、`Blog`クラスそのものに追加されます。`Blog`オブジェクトのインスタンスが何回作成されても`toHTML()`のコピーはひとつだけです。

`toHTML()`は`Blog`クラスの一部になったので、このメソッドが呼び出されると、クラスに置かれたコードが実行されます。しかし、このメソッドはインスタンスオブジェクトから呼び出され、インスタンスプロパティにアクセスできるので、やはりインスタンスマソッドです。



```
var blogEntry1 = new Blog("特に何も起こらなかった。", ...);
blogEntry1.toHTML();
```

`toHTML()`を呼び出すとクラスに
あるコードが実行されます。

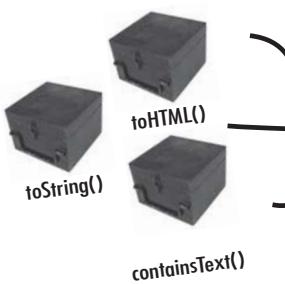
`Blog`オブジェクトのインスタンスをもうひとつ作成して`toHTML()`を呼び出すると、クラスにある同じコードが実行されます。クラスにメソッドを格納すれば、どのインスタンスからでも実行できるわけです。

```
var blogEntry2 = new Blog("まだグズグズしている。", ...);
blogEntry2.toHTML();
```

他のインスタンスもクラスに
ある同じメソッドを共有します。

クラスとprototypeとYouCube

ルビーはクラスとプロパティに関する話をきいて、すこし圧倒されています。でも彼女は頭がいいので、prototypeオブジェクトの力を借りればBlogメソッドによるメモリの無駄が省けてYouCubeにプラスになると思っています。



自分で考えてみよう

```
function Blog(body, date) {
    // プロパティを代入します
    this.body = body;
    this.date = date;
}

// ブログエントリの文字列表現を返します
Blog.prototype.toString = function() {
    return "[" + (this.date.getMonth() + 1) + "/" + this.date.getDate() + "/" +
        this.date.getFullYear() + "] " + this.body;
};

// ブログエントリの HTML 表現を返します
Blog.prototype.toHTML = function(highlight) {
    // グレイの背景色を使ってハイライト表示にします
    var blogHTML = "";
    blogHTML += highlight ? "<p style='background-color:#EEEEEE'>" : "<p>";

    // ブログの HTML コードを生成します
    blogHTML += "<strong>" + (this.date.getMonth() + 1) + "/" + this.date.getDate() + "/" +
        this.date.getFullYear() + "</strong><br />" + this.body + "</p>";
    return blogHTML;
};

// ブログ本体にテキスト文字列が含まれているか調べます
Blog.prototype.containsText = function(text) {
    return (this.body.toLowerCase().indexOf(text.toLowerCase()) != -1);
};
```

自分で考えてみよう の答え

```
function Blog(body, date) {
  // プロパティを代入します
  this.body = body;
  this.date = date;
}
```

Blogコードはprototypeオブジェクトを使ってメソッドを格納して、メソッドがクラスによって所有されるようにしています。各メソッドが何を処理するか説明してください。

```
// ブログエントリの文字列表現を返します
```

```
Blog.prototype.toString = function() {
  return "[" + (this.date.getMonth() + 1) + "/" + this.date.getDate() + "/" +
    this.date.getFullYear() + "] " + this.body;
};
```

```
// ブログエントリのHTML表現を返します
```

```
Blog.prototype.toHTML = function(highlight) {
  // グレイの背景色を使ってハイライト表示にします
  var blogHTML = "";
  blogHTML += highlight ? "<p style='background-color:#EEEEEE'>" : "<p>";
  // ブログのHTMLコードを生成します
  blogHTML += "<strong>" + (this.date.getMonth() + 1) + "/" + this.date.getDate() + "/" +
    this.date.getFullYear() + "</strong><br />" + this.body + "</p>";
  return blogHTML;
};
```

個々のBlogインスタンスにはメソッドを代入しないので、コンストラクタの外で代入します。

```
// ブログ本体にテキスト文字列が含まれているか調べます
```

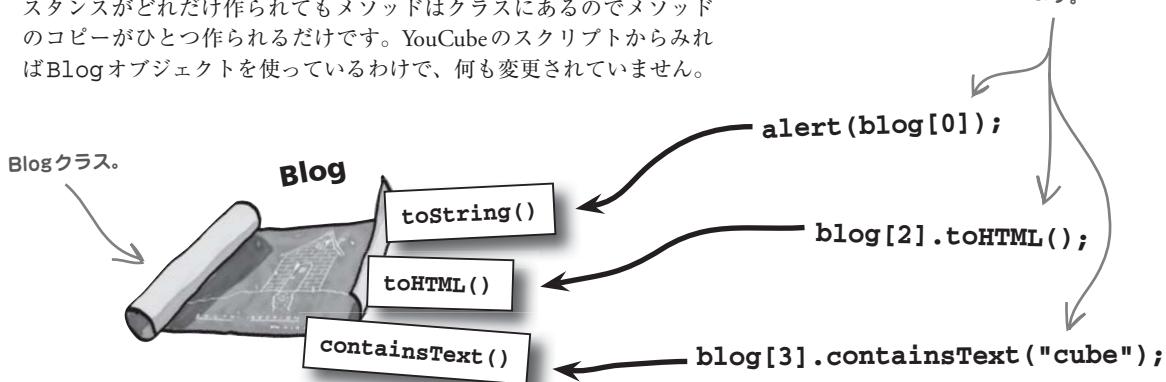
```
Blog.prototype.containsText = function(text) {
  return (this.body.toLowerCase().indexOf(text.toLowerCase()) != -1);
};
```

各メソッドはBlog()コンストラクタの中でキーワードthisを使うのではなく、prototypeオブジェクトに設定れます。

YouCubeの効率が向上しました

YouCubeのブログは、オブジェクトのプロトタイピングのおかげで、クラス所有のメソッドを使え、無駄なコードが省けました。Blogインスタンスがどれだけ作られてもメソッドはクラスにあるのでメソッドのコピーがひとつ作られるだけです。YouCubeのスクリプトからみればBlogオブジェクトを使っているわけで、何も変更されていません。

Blogインスタンスはクラスにあるメソッドを呼び出します。





重要ポイント

- クラスはオブジェクトの記述です。インスタンスはこの記述をもとに作成される実際のオブジェクトです。
- クラスはオブジェクトのプロパティやメソッドを配置します。インスタンスはプロパティに実際のデータを配置し、メソッドがこれを処理できるようにします。
- キーワードthisはインスタンス自体のコードの中からインスタンスにアクセスするのに使います。
- prototypeオブジェクトを使うと、クラスにメソッドを格納できるので、インスタンスが不必要に重複したコードになるのを防ぐことができます。

素朴な疑問に 答えます

Q: クラスとインスタンスの役割の違いがよくわかりません。役割分担はどうなっているのですか？

A: クラスの背景にある発想は、オブジェクトの作成と再利用を簡単にすることです。オブジェクトリテラルとして一回限りのオブジェクトを毎回作ることもできますし、それによって不必要的労力はかかるものの、問題があるわけではありません。でも同じ労力を繰り返すのは無駄ですよね。同じ住宅を建てているのに、建てるたびに毎回設計図を作り直すようなものです。たくさんのインスタンスを作るときは、テンプレートを使えばもっと労力が少なくてすみます。ここでクラスの登場です。ひとつクラスを作り、これを使って必要なだけインスタンスを作成すればいいのです。

Q: 互いに似ているオブジェクトの作成を簡単にするのがクラスというのわかりました。thisとprototypeはクラスとどういう関係なんでしょうか？

A: キーワードthisはインスタンス自身のメソッドの中からそのインスタンスにアクセスするための手段です。主にインスタンスプロパティにアクセスするときに使われます。あるメソッドから名前がxのプロパティにアクセスする場合、this.xと書きます。ただxと書いただけでは、どのインスタンスのプロパティにアクセスしようとしているのかコードにはわからないので、xは変数と見なされてしまいます。コンストラクタが行うア

ロパティの作成と初期化においてthisを使う必要があるのはこのためです。prototypeはまったく違った動きをします。クラスを作成するための仕組みですが、他のプログラミング言語、たとえばC++やJavaと違って、JavaScriptではクラスは具体的にはサポートされていません。そのわり、JavaScriptではprototypeを使ってクラスをシミュレートしています。最終結果は同じのですが、JavaScriptではprototypeオブジェクトの操作によって「クラス」を作る必要があるわけです。このprototypeオブジェクトは、JavaScriptオブジェクトの「隠された」プロパティとして現れます。prototypeオブジェクトにプロパティやメソッドを格納すると、それらをインスタンスの一部としてではなく、クラスの一部として効率よくアクセスすることができます。

Q: コンストラクタはクラスにどう収まるのでしょうか？

A: コンストラクタはJavaScriptクラスを作成する上で非常に重要な部分になります。コンストラクタはオブジェクトのインスタンスを作る責任があるからです。コンストラクタとプロトタイプは、JavaScriptクラスにおける2つの大きな部品です。コンストラクタはインスタンスに関するすべてを設定するのに対し、プロトタイプはクラスレベ

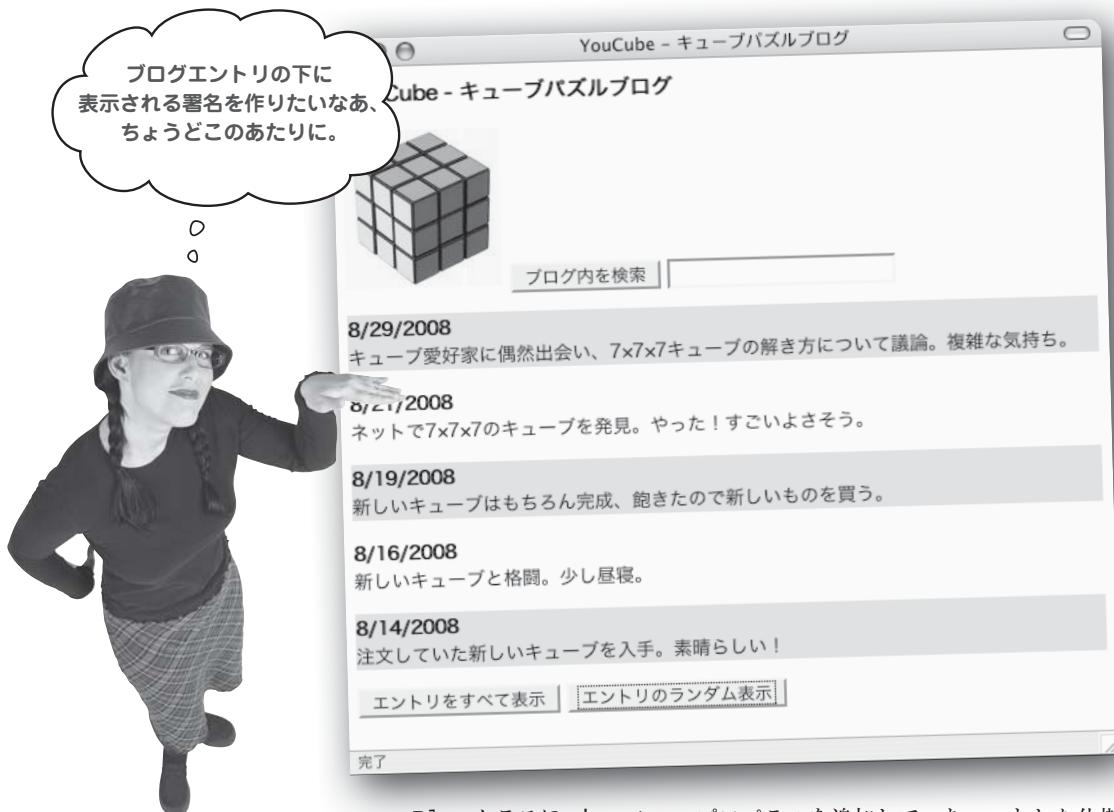
ルでのすべてを処理します。この2つを合わせることで、かなりすごいことが実現できます。というのも、あるメンバーはインスタンスレベルに置き、あるメンバーはクラスレベルに置かなければならない理由があるからです。この点については本章でさらに説明を続けます。

Q: オブジェクトの名前の付け方の慣習に戸惑っています。大文字で始まることもあるあれば、小文字で始まることもあります。なにか気づいていない規則があるんでしょうか？

A: クラス名は大文字で始まり、インスタンス名は小文字で始まるキャメルケースになります。インスタンスは変数にすぎないので、変数の命名規則である小文字で始まるキャメルケースを使います。名前の付け方が一貫していないのは「オブジェクト」という語の使い方がかなり大雑把だったためです。正確に説明すると、Blogのようにクラスは大文字で始まり、blogEntryやblog[0]のようにインスタンスは小文字で始まるキャメルケースを使います。この命名規則は、これまで使ってきた標準でないオブジェクトを読み直すとき役に立ちます。現在の日付/時刻はnowという変数(インスタンス)に格納します。このnowはDateオブジェクト(クラス)から作成されます。

ブログに署名をつける

ルビーはYouCubeにオブジェクト指向プログラミング(OOP)を導入することで効率と編成を改善しているところですが、画面の裏にあるコードを改善するだけでなく、新機能を追加したいと思っています。



Blogクラスにsignatureプロパティを追加して、ちょっとした仕掛けを施します。コンストラクタでプロパティを設定すれば、ブログエントリごとに署名が表示できる、というわけです。



頭の体操

ルビーは署名をインスタンスプロパティとして作成すべきでしょうか？ それがあまりいいやり方でないとしたら、その理由は何だと思いますか？

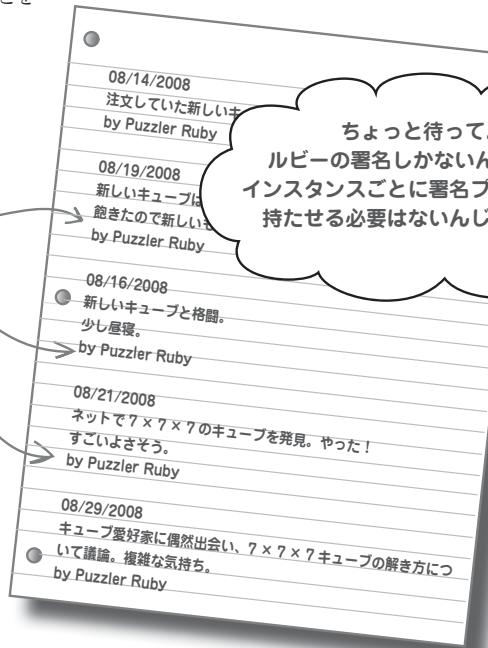
Q: オブジェクト指向という語をみたことがあります。どういう意味なのでしょうか？

A: オブジェクト指向という語はプログラミングのサークルでちょっと乱用されているくらいがあり、人によって意味されることが違っています。一般には、オブジェクト指向プログラミング(OOP)はソフトウェアをオブジェクトから作成することを

素朴な疑問に 答えます

OOPだと考えています。理論的には、眞のオブジェクト指向プログラムであれば、互いにやり取りを交わすオブジェクトの集まりに分解することができます。オブジェクト指向の純粹主義者は、JavaScriptはOOP言語を名乗る資格がないと主張しています。こうした神学論争には関わらないようにしましょう。どちらの主張も正しいので決着がつきません。

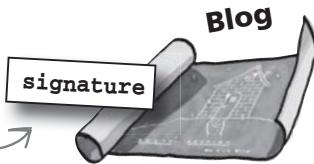
すべてのブログエントリで署名は同じです。



署名はひとつで十分かもしれません。

ブログの署名はすべてのインスタンスで同じなので、インスタンスごとに署名プロパティを持つ必要はありません。ルビーに必要なのはクラスプロパティです。個々のインスタンスにプロパティのコピーを作るのではなく、クラスの中にひとつだけコピーを作るのでです。

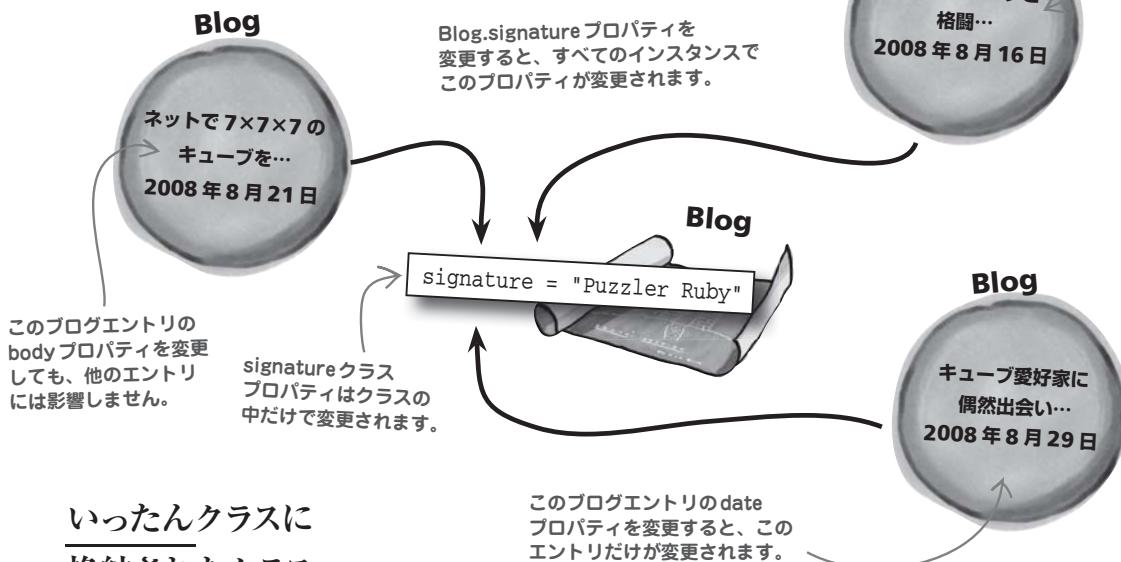
signatureプロパティはブログインスタンスではなくBlogクラスに格納すべきです。



クラスプロパティも共有される

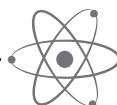
クラスプロパティはクラス所有のインスタンスマетодと非常に似ています。クラスが所有するプロパティの一つのコピーがすべてのインスタンスで利用できるからです。クラスプロパティにはすべてのインスタンスで共有される値がひとつだけあることになります。YouCubeブログには署名がひとつだけなのですから、このsignatureプロパティこそルビーが探していた機能です。

各インスタンスにはそれぞれ
インスタンスプロパティが
格納されます。



いったんクラスに
格納されたクラス
プロパティはすべての
インスタンスから
アクセスできます。

signatureプロパティはBlogクラスに格納されますが、ブログの著者の署名にアクセスしたいすべてのインスタンスから容易にアクセス可能です。

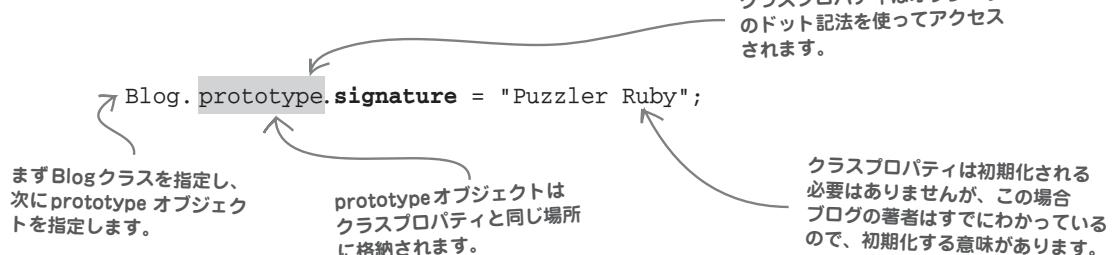


頭の体操

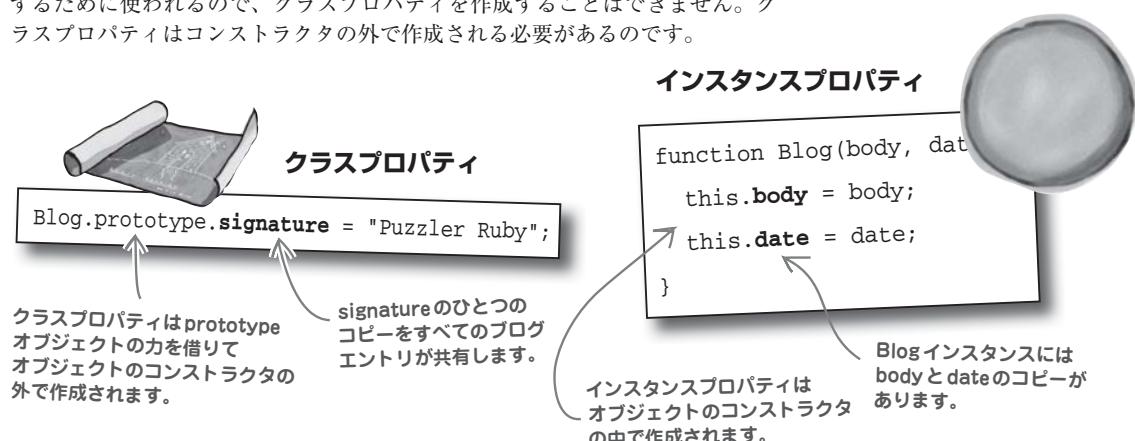
クラスプロパティを作成するにはどうしたらいいと思いますか？

prototypeを使ってクラスプロパティを作成する

クラスプロパティがどこに格納されるのか、またその節約効果について説明してきましたが、クラスプロパティの作り方は驚くほどありきたりです。以下のように、たった1行のコードで済んでしまいます。



このコードで最も興味深いのは、これを見ただけではその価値が十分に理解できない点です。インスタンスプロパティを作成するコードと違って、コンストラクタの中にはコードはありません。コンストラクタはインスタンスを実現するために使われる所以、クラスプロパティを作成することはできません。クラスプロパティはコンストラクタの外で作成される必要があります。



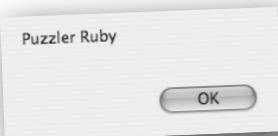
 **自分で考えてみよう**

アラーとボックスにsignatureプロパティを表示するコードを書いてください。
ヒント: Blogメソッドの中にコードがあると想定します。

.....
.....

自分で考えてみよう の答え

アラーとボックスにsignatureプロパティを表示するコードを書いてください。
ヒント：Blogメソッドの中にコードがあると想定します。



`alert(this.signature);`

インスタンスプロパティと同様に
クラスプロパティはキーワード
`this`を使ってアクセスできます。

素朴な疑問に答えます

Q：YouTubeの署名をプロパティに格納する必要があるのはなぜですか？ 各エントリの本文テキストの一部として入力できると思うのですが。

A：本文テキストの一部として各ブログエントリに署名を持たせることは可能ですが、ブログにポストする人が一人しかいない場合、時間と労力の無駄になります。JavaScriptを使って署名を処理するともっとすっきりするのに、ブログエントリごとに署名していると、うんざりしてきます。それにうつかりタイプミスをして「Puzzled Ruby」になってしまふと、困ったことになります。もうひとつの選択肢として、ブログエントリをHTML形式にして、文字列リテラルを使って署名を格納する方法があります。このアプローチは、データの重要な情報である署名がブログの中に埋もれてしまい、探したり維持管理するのが困難になります。署名をクラスプロパティに格納すれば、簡単にアクセスできるので、ブロガーには識別と変更が簡単になります。

Q：署名をインスタンスプロパティにすると、ブログエントリの作成方法が変わりますか？

A：オブジェクトのインスタンスごとにコンストラクタで初期化され

るインスタンスプロパティのセットがあります。署名をインスタンスプロパティにすると、Blog()コンストラクタはインスタンスごとにこのプロパティを設定しなければなりません。コンストラクタは署名の文字列をプロパティに設定できるので、からならずしもコーディングが面倒になるわけではありませんが、画面の裏ではインスタンスごとに署名を何回もコピーすることになり、非常に無駄な処理を行うことになります。それだけでなく、他のインスタンスの署名を変更できてしまいます。

Q：YouTubeを変更して複数のブロガーが使えるようにしたいのですが、インスタンスプロパティの署名を変更することになりますか？

A：はい、複数のブロガー対応にするシナリオは、署名のプロパティは各インスタンスごとに別々の値を保持する必要があるので、いいアプローチです。Blog()コンストラクタに引数を追加して、署名文字列を渡せるようにするアプローチはベストです。この文字列を使って署名インスタンスプロパティを初期化します。つまり、Blogインスタンスの他のプロパティと同じように署名プロパティを扱うということです。

Q：クラスプロパティはグローバル変数の一種のように思えます。これらの違いは何ですか？

A：実際のところクラスプロパティはどこからでもアクセスできるので、かなりグローバル変数と似ています。クラスプロパティが作られる場所もグローバル変数と似ていて、他のスクリプトの外側にあるメインスクリプトのレベルに作られます。クラスプロパティとグローバル変数の違いは、前者がクラスと関連している点です。そのためインスタンスオブジェクトと関連しています。つまりクラスプロパティはインスタンスとしてアクセスする必要があります。

Q：ちょっと待ってください。クラスプロパティはインスタンスからアクセスする必要があるんですか？

A：クラスプロパティはprototypeオブジェクトを使って作成され、このオブジェクトがプロパティをクラスに格納するのですが、インスタンスからアクセスする必要があります。つまりクラスプロパティはインスタンスプロパティと同じように、キーワードthisとオブジェクト(ドット)記法を使ってアクセスされます。両者はプロパティが格納される場所に違いがあります。クラスプロパティはクラスに、インスタンスプロパティはインスタンスに格納されます。

署名が完成しました

`signature`クラスプロパティを作成し、初期化したら準備完了です。ブラウザに表示されるブログエントリは`toHTML()`で各ブログエントリに署名を追加した表示形式になります。

`toHTML()`は署名をプログエントリの一部として表示します。

```
Blog.prototype.toHTML = function(highlight) {
    // グレイの背景色を使ってハイライト表示にします
    var blogHTML = "";
    blogHTML += highlight ? "<p style=background-color:#EEEEEE>" : "<p>";
    blogHTML += "<strong>" + (this.date.getMonth() + 1) + "/" + this.date.getDate() + "/" +
        this.date.getFullYear() + "</strong><br />" + this.body + "<br /><em>" + this.signature +
        "</em></p>";

    return blogHTML;
};
```

ルビーの署名は
プログエントリの
一部として表示
されます。

9/1/2008 意を決して7x7x7キューブを注文。頭の準備体操を開始する。
by Puzzler Ruby

8/29/2008 キューブ愛好家に偶然出会い、7x7x7キューブの解き方について議論。複雑な氣
by Puzzler Ruby

8/21/2008 ネットで7x7x7のキューブを発見。やった！すごいよさそう。
by Puzzler Ruby

8/19/2008 新しいキューブはもちろん完成、飽きたので新しいものを買う。
by Puzzler Ruby

これで
プログエントリを
誰が書いたかはっきり
するわね。

クラスプロパティである
`signature`は通常のイン
スタンスプロパティと
同じように参照されます。



ルビーはJavaScript言語を拡張するOOPの手法を使って、`Blog`クラスに`signature`クラスプロパティを追加しました。YouCubeブログにより親密な感じを与えることができたのが、彼女にとって収穫です。

特別座談会



今夜の対談：

インスタンスプロパティとクラスプロパティがデータの所有権と秘密結社について話します。

インスタンスプロパティ：

私は別世界の人だと聞いていたので、どうしてあなたがこの席にいるのか不思議です。私の仕事は、オブジェクトのインスタンスが他と重複しないようにすること、それぞれのインスタンスのプロパティの値を把握することです。

それは信じがたい話だ。続けてください。

つまり握手の仕方を格納する私のやり方は最善ではないとおっしゃるんですか？

わかりました。ではパスワードはどうですか？ 私が格納してもいいんですか？

それは良かった。では、さっそくですが、秘密結社を結成して、それぞれパスワードを持つことにしましょう。

おっと、こいつはやられたなあ。本気で心配なんですよ。

クラスプロパティ：

あなたの仕事はよく存じています。立派な仕事ですよね。でも、インスタンスが自分のデータを把握するのを面倒に思うときもあるのはご存知でしたか？

そうですねえ、すべてのインスタンスが共有するデータ断片がある状況を考えてください。たとえば、フリーメイソンでは秘密の握手が交わされる。結社のメンバーは全員、この握手の仕方を知っています。結社の女性が彼女だけの秘密の握手の仕方をしてしまうと、すべて台無しになってしまいます。他の女性もまた彼女だけの握手の仕方をするので、握手の仕方があふれすぎて、誰もわからなくなってしまいます。

そう思います。あ、責めているのではなくて、握手の仕方がひとつだけあれば十分ではないかと。メンバー全員が共有している必要がありますが。

各人が自分のパスワードを持つのであれば、問題ありません。秘密のパスワードを格納するのにふさわしいですね。

でもあなたはフリーメイソンの握手の仕方をご存知ないですよね。

コードの重複はダメですよ

ルビーは忙しそうです。YouCubeコードの効率をさらに向上させることにしました。日付を表示するコードが重複しているのを見つけたので、OOPの原理を応用すれば重複をなくせるはずです。

このコードは無駄に重複しているようね。
どうしたらいいかな？

```
<html>
<head>
<title>YouCube - The Blog for Cube Puzzlers</title>
<script type="text/javascript">
// Blogオブジェクトのコンストラクタ
function Blog(body, date) {
    // Assign the properties
    this.body = body || "Nothing going on today.";
    this.date = date || new Date();
}

// ブログエントリの文字列表現を返します
Blog.prototype.toString = function() {
    return "[" + (this.date.getMonth() + 1) + "/" + this.date.getDate() + "/" +
        this.date.getFullYear() + "] " + this.body;
};

// ブログエントリのHTML表現を返します
Blog.prototype.toHTML = function(highlight) {
    // Use a gray background as a highlight, if specified
    var blogHTML = "";
    blogHTML += highlight ? "<p style=background-color:#EEEEEE>" : "<p>";
    blogHTML += "<strong>" + (this.date.getMonth() + 1) + "/" + this.date.getDate() + "/" +
        this.date.getFullYear() + "</strong><br />" + this.body + "<br /><em>" + this.signature +
        "</em></p>";
    return blogHTML;
};

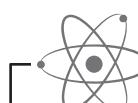
// ブログ本文にテキスト文字列が含まれているか調べます
Blog.prototype.containsText = function(text) {
    return (this.body.indexOf(text) != -1);
};

// ブログの署名を設定します
Blog.prototype.signature = "by Puzzler Ruby";

```



この日付整形の
コードは同じなので
すこし冗長です。

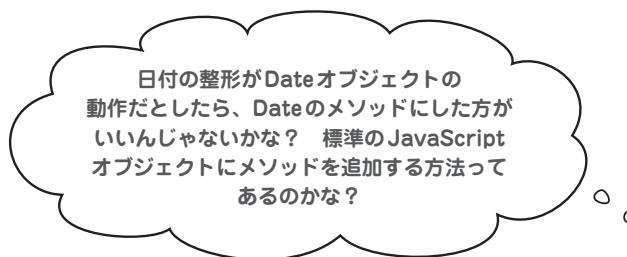


頭の体操

YouCubeで日付整形のコードの重複をなくすにはどうしたらいいですか？

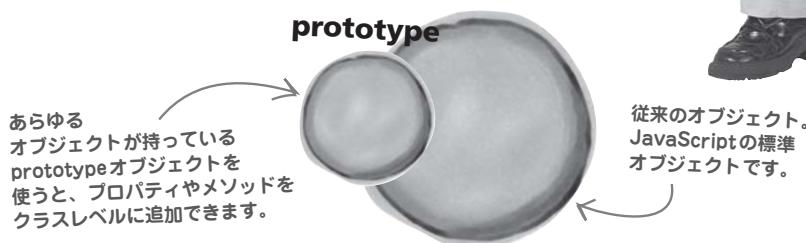
日付を整形するメソッド

ルビーは日付を整形するコードの重複を解決するために、Blogオブジェクトに日付を整形するメソッドを追加することにしました。コードを再利用するには関数かメソッドにコードを移す必要がありますが、Blogオブジェクトはプロパティの表示形式の一部として日付を整形する責任もあるので、ルビーはメソッドを選びました。他の選択はないのでしょうか？



prototypeオブジェクトに戻る

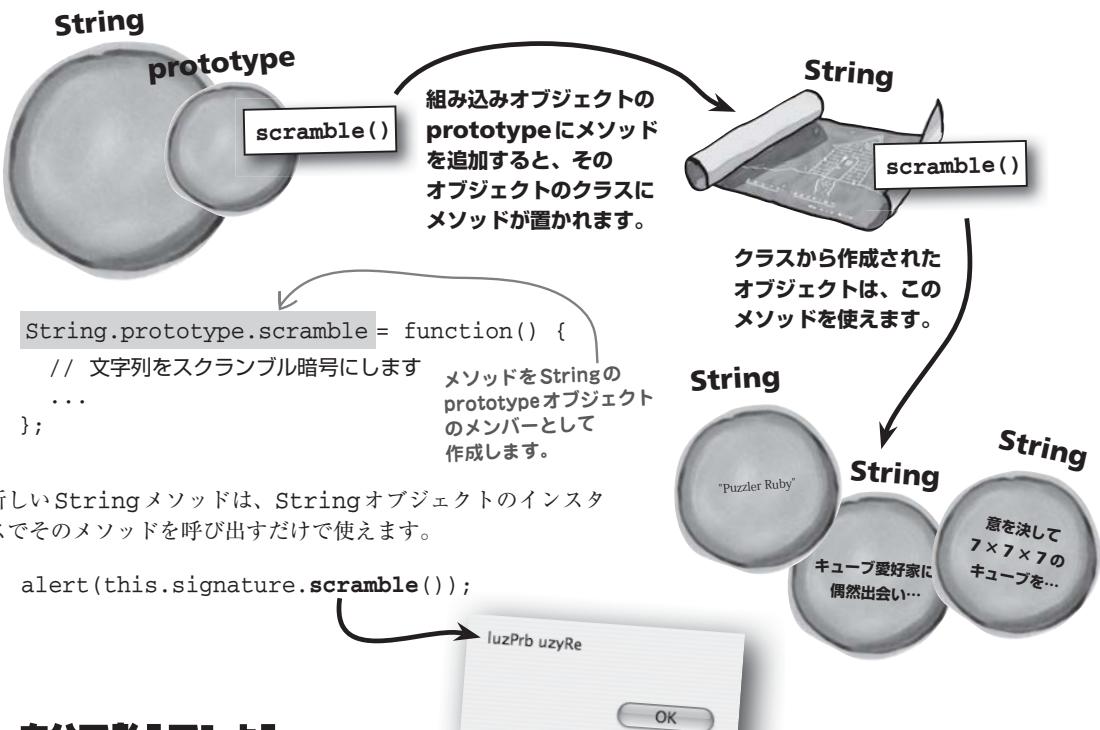
既存のオブジェクトをより強力にすることは可能でしょうか？ 実は標準オブジェクトを変更する方法、JavaScript言語を拡張する究極の選択肢があります。標準オブジェクト、あるいはすべてのJavaScriptオブジェクトを拡張する場合、prototypeオブジェクトが鍵なのです。すでにprototypeオブジェクトを使ってBlogクラスにクラスプロパティとクラス所有のメソッドを追加しました。これと同じ拡張が、組み込みのJavaScriptクラスにも適用できるのです。



標準オブジェクトを拡張する

オブジェクトを拡張する鍵はprototypeオブジェクトにあります。JavaScriptのオブジェクトはすべてprototypeオブジェクトを持っています。prototypeオブジェクトにプロパティやメソッドを追加すると、クラスプロパティとクラス所有のメソッドができるので、オブジェクトを拡張できるのです。組み込みのJavaScriptオブジェクトの場合、prototypeオブジェクトにプロパティやメソッドを追加すると、その組み込みオブジェクトの新しいインスタンスは追加したプロパティやメソッドにアクセスできるのです。

prototypeオブジェクトを使うとJavaScriptの組み込みオブジェクトを拡張できます。



自分で考えてみよう

標準のDateオブジェクトを拡張して`shortFormat()`というメソッドのコードを書いてください。このメソッドはMM/DD/YYYY形式で日付を整形します。

自分で考えてみよう の答え

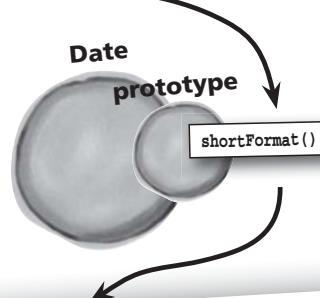
Dateオブ
ジェクトの
prototypeに
メソッドを
追加します。

標準のDateオブジェクトを拡張してshortFormat()というメソッドのコードを書いてください。このメソッドはMM/DD/YYYY形式で日付を整形します。

```
Date.prototype.shortFormat = function() {  
    return (this.getMonth() + 1) + "/" + this.getDate() + "/" + this.getFullYear();  
};
```

カスタム日付オブジェクトで YouCubeを改良する

Dateオブジェクトの機能を拡張すると、YouCubeはより効率的になります。日付の形式を1カ所で変更するだけでブログ全体の日付の表示形式が変わるので、YouCubeの維持管理も容易になります。OOPによるスクリプトの改善は、すぐに外見の変化に現れるものばかりではなく、長期的にみてコードの動作がより良くなることもあります。



9/3/2008
キューブパズルの取り扱いをやめてしまった近所のおもちゃ屋の抗議集会に参加。パズラーたちよ、
頑張れ！
by Puzzler Ruby

9/1/2008
意を決して7x7x7キューブを注文。頭の準備体操を開始する。
by Puzzler Ruby

8/29/2008
キューブ愛好家に偶然出会い、7x7x7キューブの解き方について議論。複雑な気持ち。
by Puzzler Ruby

8/21/2008
ネットで7x7x7のキューブを発見。やった！すごいよさそう。
by Puzzler Ruby

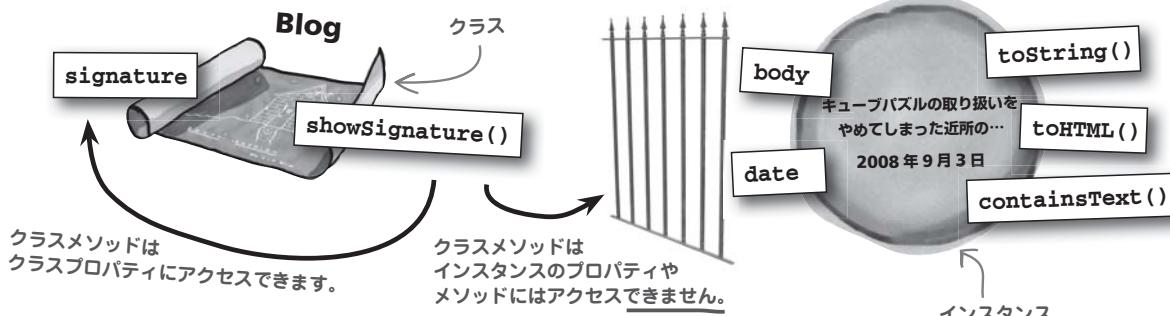
8/19/2008
新しいキューブはもちろん完成、飽きたので新しいものを買う。
by Puzzler Ruby

ブログの日付は
Dateオブジェクト
のカスタムカスタム
メソッドを使って
整形されます。

クラスに独自のメソッドをもたせる

DateオブジェクトのカスタムメソッドshortFormat()は**クラスが所有するインスタンスメソッド**です。クラスが所有しているメソッドですが、インスタンスプロパティにアクセスできます。このためあるインスタンスの中に格納された日付を整形するメソッドが実現可能になります。また**クラスメソッド**も作成することができます。クラスメソッドはクラスが所有するメソッドであり、インスタンスプロパティには**アクセスできません**が、Blogクラスのsignatureプロパティなどクラスプロパティにはアクセスできます。

クラスメソッドはクラスが所有し、クラスプロパティだけにアクセスできます。



クラスメソッドの作成方法は、prototypeオブジェクトは使わず、クラスにメソッドを設定するだけです。クラス名とドット記法を使ってクラスにメソッドを代入します。

```
Blog.showSignature = function() {
  alert("This blog created by " + Blog.prototype.signature + ".");
};
```

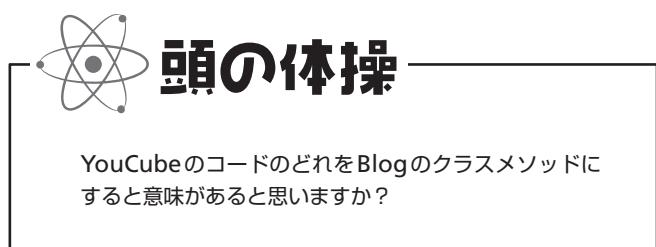
クラスメソッドはインスタンスと何の関係もないで、クラス名だけを参照して呼び出します。これはインスタンスからクラスメソッドを呼び出すことができないという意味ではなく、クラス名を使って呼び出すことができます。

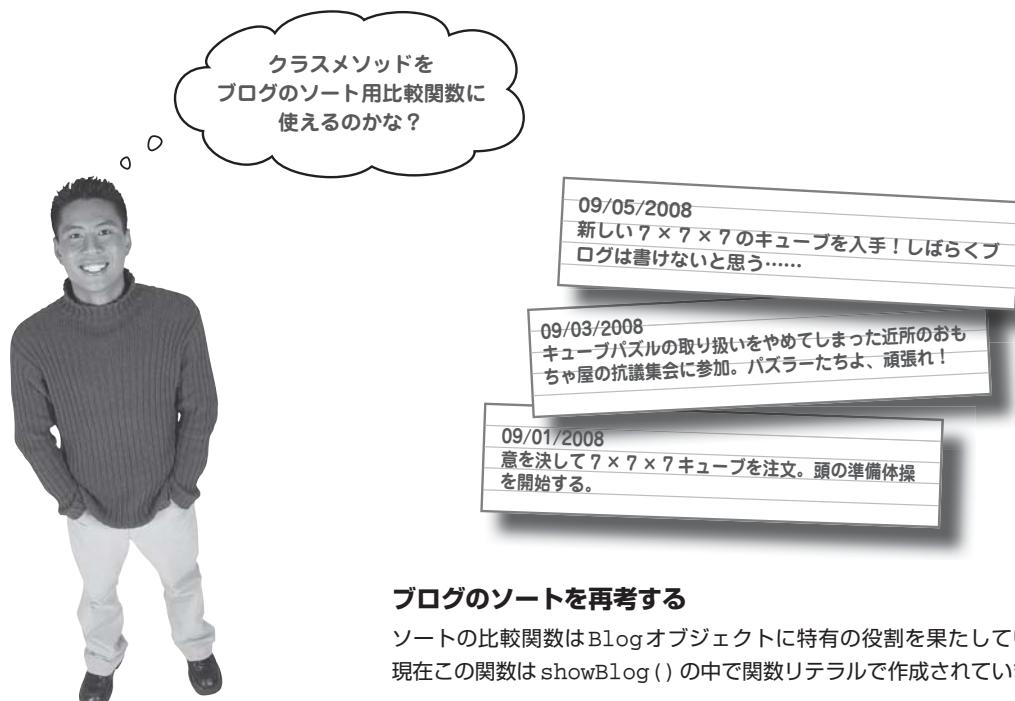
```
Blog.showSignature();
```

クラスメソッドを呼び出すときクラス名が鍵になります。

クラスメソッドからクラスプロパティにアクセスするとき、prototypeプロパティを掘り下げる必要があります。

signatureはクラスプロパティなのでクラスメソッドからアクセスできます。





ブログのソートを再考する

ソートの比較関数はBlogオブジェクトに特有の役割を果たしています。
現在この関数はshowBlog()の中で関数リテラルで作成されています。

showBlog()の中で
ブログがソート
されます。これは
Blogオブジェクトの
一部ではありません。

```
function showBlog(numEntries) {  
    // まず、日付を基にブログを逆順にソートします（最新のものが先頭になるように）  
    blog.sort(function(blog1, blog2) { return blog2.date - blog1.date; });  
    ...  
}
```

ソート用比較コードをクラスメソッドに
移動することができそうです。

OOPの基本概念のひとつは、オブジェクトそのものの中にオブジェクトの機能を
カプセル化することです。オブジェクトが自分で実行するコードをオブジェクトの
外にあるコードから実行すべきではない、という考えです。この場合、ブログエン
トリをソート用に比較する処理は、showBlog()ではなくオブジェクトの中で実
行されます。では、ソート用の比較コードをBlogクラスのクラスメソッドにする
ことはできるでしょうか？この問い合わせに答えるには、そのメソッドがインスタンス
のデータやメソッドにアクセスする必要があるかどうかを調べる必要があります。
クラスメソッドはインスタンスの中にあるものにアクセスできないので、これは重
要な問題です。

ソート用比較関数を調べる

実現可能かどうかを判断するには、関数で何が実行されているのか調べるしかありません。ソート用の比較関数は関数リテラルであり、通常の関数とかなり似た形式になっています。この関数には2つのブログのインスタンスが引数で渡されます。

```
function(blog1, blog2) {
    return blog2.date - blog1.date;
}
```

2つのブログインスタンスが
引数として渡されます。

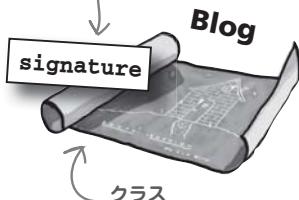
ソート比較では2つの
引数を引き算します。

この関数はブログのインスタンスを直接扱いますが、インスタンスは引数として渡されます。これはインスタンスの中からthisを使ってプロパティやメソッドにアクセスするやり方とは違っています。クラスメソッドでは、このやり方はできません。ソート用の比較関数はインスタンスの中にあるものにアクセスする必要がないので、クラスメソッドに最適です。

実際、ソート用比較関数はクラスプロパティを必要としません。

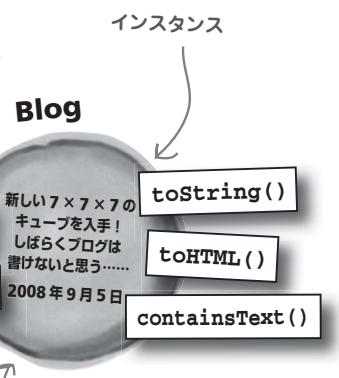
もちろん必要な場合はクラスメソッドはクラスプロパティに
アクセスできるので、それを実現することは可能です。

クラスメソッドはクラスプロパティに
アクセスできます。



ソート用比較関数はインスタンスやクラスデータへアクセスする必要がありません。

```
function(blog1, blog2) {
    return blog2.date - blog1.date;
}
```



ソート用比較関数がインスタンスにアクセスする必要があるときは、これをクラスメソッドにすることはできません。

自分で考えてみよう

YouCubeのブログソート用比較関数をBlogオブジェクトのクラスメソッドblogSorter()として書き直してください。



YouCubeのブログソート用比較関数をBlogオブジェクトのクラスメソッド blogSorter()として書き直してください。

```
Blog.blogSorter = function(blog1, blog2) {  
    return blog2.date - blog1.date;  
};
```

ソート用比較メソッドはBlogオブジェクトの
クラスメソッドblogSorter()になりました。

クラスメソッドを呼び出す

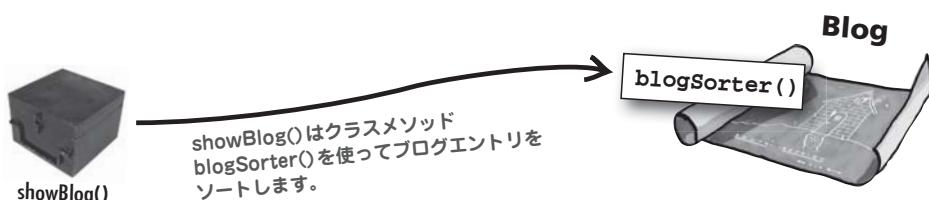
ブログのソート用比較関数をBlogメソッドに移すと、そのメソッドを呼び出すコードの動作が読みやすくなります。

```
function showBlog(numEntries) {  
    // まずブログをソートします  
    blog.sort(Blog.blogSorter);  
    ...  
}
```

ブログのソートの詳細はBlogクラスの
クラスメソッドblogSorter()に
委ねられています。

このコードが美しいのは、些細なことが重要です。showBlog()はブログエントリがソートされる方法をもはや気にする必要はありません。ブログエントリのソート方法の詳細はBlogクラスの中で処理されます。

ソートそのものはBlogクラスの外にあるshowBlog()から開始されます。ソート処理はブログのインスタンスの集まり全体に影響を及ぼすので、これはうまいやり方です。個々のブログエントリがどのようにソートされるか、その詳細はBlogクラスが扱える領域の中になります。優れたOOP設計では、オブジェクトとそのまわりのコードとの間でうまく調和がとれているのです。



blog

一枚の写真は千語に匹敵する

OOPによってYouCubeが改善されて、ルビーはワクワクしているのですが、彼女の熱い思いをユーザと共有できないことも分かっています。OOPの効果はユーザ体験に劇的に影響を与えるものではないからです。そこで彼女は目を引く機能をブログに追加することにしました。



ルビーは個々のブログエントリに日付と本文テキストと一緒にオプション画像が表示できるようにしようと考えました。すべてのブログエントリに画像があるわけではないので、画像をオプションにするのは重要です。そうすれば、すでに書かれた既存のブログエントリの表示が壊れることもありません。


頭の体操

YouCubeのBlogオブジェクトで画像をサポートするにはどうしたらいいでしょう？

YouCubeに画像を組み込む

YouCube ブログに画像を追加するには、すでに動いているオブジェクトに干渉することなく、Blog オブジェクトに画像を組み込む方法を調べる必要があります。この設計を実現するにあたって、2つの重要な課題があります。

① ブログの画像を Blog オブジェクトの中に格納する最善の方法は？

② ブログ画像をオプションとして YouCube に追加する方法は？

ブログ画像を格納する方法に関係なく、YouCube のウェブページに表示される画像は HTML の タグに最終的には帰結します。

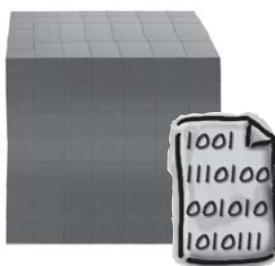
```

```

ブログ画像はファイル名の文字列を使って記述すれば十分です。

このコードから分かることは、ブログに関する限り、画像の表示は文字列で制御されるということです。この文字列は、ウェブサーバに格納されている画像ファイルを参照しますが、Blog オブジェクトから見れば、ただの文字列というわけです。

Blog オブジェクトに
関しては画像は
文字列にすぎません。



7x7x7 のキューブの
画像は cube777.png と
いうファイル名で
格納されます。

つまり画像は、body プロパティと同じように、文字列を格納する
プロパティとして Blog オブジェクトに追加できるということです。

Blog

toString()

toHTML()

containsText()

body と image のプロパティ
はどちらも文字列として
Blog オブジェクトに
格納されます。

body

date

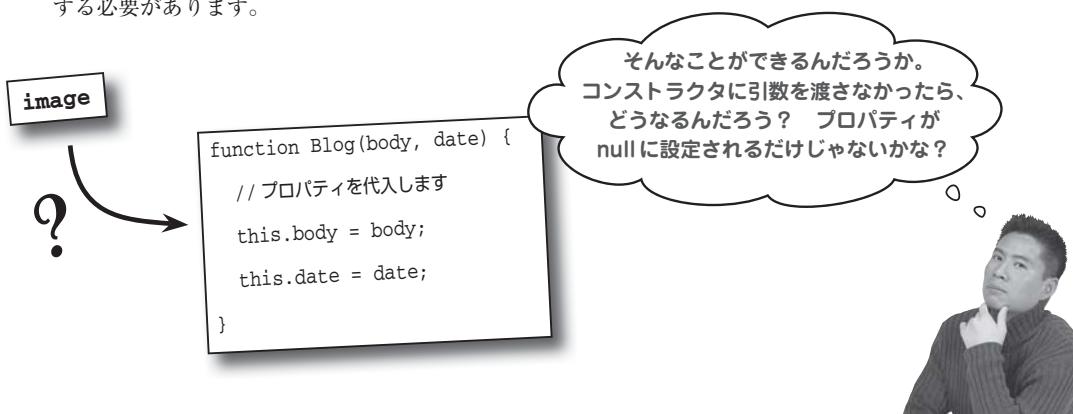
image

やった！ 2 週間も
かかったけれど…
2008 年 9 月 19 日

ブログ画像をオプションにする

2

ブログ画像はBlogオブジェクトにimageという名前の文字列プロパティとして格納されますが、このプロパティを純粋にオプション機能としてブログに追加する方法については、まだ解決できていません。この問題は、オブジェクトを作成し初期化するコンストラクタに最終的には帰結します。このプロパティをオプションとして扱えるように、コンストラクタに特別なコードを追加する必要があります。



見つからなかった引数はnullになります。

関数、メソッド、コンストラクタのいずれも、引数が渡されなかったら、引数を使おうとするコードでは引数の値がnullになります。特にコンストラクタの場合、渡されなかった引数に関するプロパティはnullになるということです。これは必ずしも悪いことではありません。オプションにするコンストラクタの引数を引数リストの最後に配置すれば、他の引数に影響を与えずにすみます。このテクニックは、関数やメソッドにも適用できますが、Blog()コンストラクタの引数imageには特に適しています。



自分で考えてみよう

YouCubeのBlog()コンストラクタを画像プロパティをサポートするように書き換えて、ブログエントリ画像を格納できるようにしてください。

.....

.....

.....

.....

.....

自分で考えてみよう の答え

作成されたimage
プロパティは
引数imageで
初期化されます。

YouCubeのBlog()コンストラクタを画像プロパティをサポートするように書き換えて、ブログエントリ画像を格納できるようにしてください。

```
function Blog(body, date, image) {  
    // プロパティを代入します  
    this.body = body;  
    this.date = date;  
    this.image = image;  
}
```

引数imageを
コンストラクタの
最後に追加します。

素朴な疑問に 答えます

Q: Blog() コンストラクタの引数
リストで引数imageが最後な
のは重要な意味があるのでしょうか？

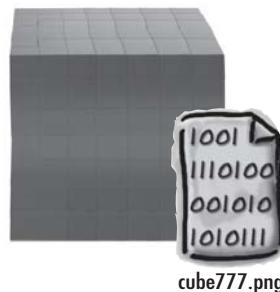
A: はい、画像はブログエントリのオ
プションとみなされるので最後に
あります。ここで重要なのは、関数に引
数を渡す方法、とくにオプション引数と
して渡す方法です。関数に2つの引数
がある場合、両方とも渡す、最初の引数
だけ渡す、どの引数も渡さない、この3
つの選択肢があります。2番目の引数だ
け渡すことはできません。

そのためオプション引数にするときは、
引数リストの最後に置いて省略するこ
とが可能か検討してください。また引数
のリストで重要な引数ほど先に置かれて
いるか確認してください。Blog() の引
数imageはオプションなので、簡単に
省略できるように、引数リストの最後に
なっています。

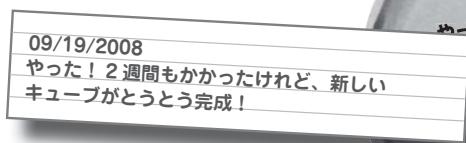
YouCubeに画像を追加する

画像をサポートする新しいBlog() コンストラクタは、それを使う
ブログエントリがなければ、それほど役に立つものではありません。
画像をサポートするブログエントリを作成するには、以下の2つが必
要です。

- 1 ブログ画像ファイルをYou
Cubeウェブページと同じ
ウェブサーバのフォルダに
配置する。



- 2 YouCubeスクリプトコー
ドで新しいブログエントリ
をBlogオブジェクトとして
作成する。



この手順が完了すれば、以下のコードが実行できます。Blog() コンストラクタの最後の引数として画像の文字列を渡せば、新しいブログエントリが作成されます。

```
new Blog("やった！2週間もかかったけれど、新しいキューブがとうとう完成",
```

```
new Date(09/19/2008), "cube777.png")
```

ブログ画像はコンストラクタ
Blog()の最後の引数として
渡されます。

ブログ画像を表示する

画像のあるブログエントリが作成できたので、YouCube の画像機能に必要な作業はあとひとつです。コンストラクタとオプション引数に関するこれまでの説明は、ブログエントリを表示するコードが新しいimage プロパティに影響されないという意味ではありません。

このコードはtoHTML() の中にあります。このコードはブログを HTML コードで整形するので、image プロパティに有効な値が設定されているかどうかを考慮する必要があります。ブログエントリの表示は2通りになります。ひとつは画像がある場合、もうひとつは画像がない場合です。画像があるかどうかによって、ブログの表示方法が決まります。

ブログエントリは
この疑似コードの
論理に従って
表示されます。

If (image が存在する)

画像付きでブログエントリを表示

Else

画像なしでブログエントリを表示



自分で考えてみよう

Blog オブジェクトのtoHTML() メソッドからオプション画像を表示するコード断片が消えてしましました。消えたコード断片を書いてください。
またその実行内容も注記してください。

```
if ( ..... ) {
    blogHTML += "<strong>" + this.date.shortFormat() +
    "</strong><br /><table><tr><td><img src=' " + this.image +
    "' /></td><td style='vertical-align:top'>" + this.body + "</td></tr></table><em>" +
    this.signature + "</em></p>";
}
else {
    blogHTML += "<strong>" + this.date.shortFormat() + "</strong><br />" + this.body +
    "<br /><em>" + this.signature + "</em></p>";
}
```

自分で考えてみよう の答え

```
if (this.image ..... ) {  
    blogHTML += "<strong>" + this.date.shortFormat() +  
    "</strong><br /><table><tr><td><img src=' " + this.image + " ..... </td><td style='vertical-align:top'>" + this.body + "</td></tr></table><em>" +  
    this.signature + "</em></p>;  
}  
else {  
    blogHTML += "<strong>" + this.date.shortFormat() + "</strong><br />" + this.body +  
    "<br /><em>" + this.signature + "</em></p>;  
}
```

結果がtrueでなければ、ブログエントリは
画像なしで表示されます。

テスト条件の結果がtrueなら、
画像プロパティに実際の画像に
設定され、画像が表示されます。

オブジェクトで強化された YouCube

ルビーは大喜びです。彼女のブログはオブジェクトのおかげで飛躍的に成長しました。ユーザも新しい画像機能を気に入ることでしょう。

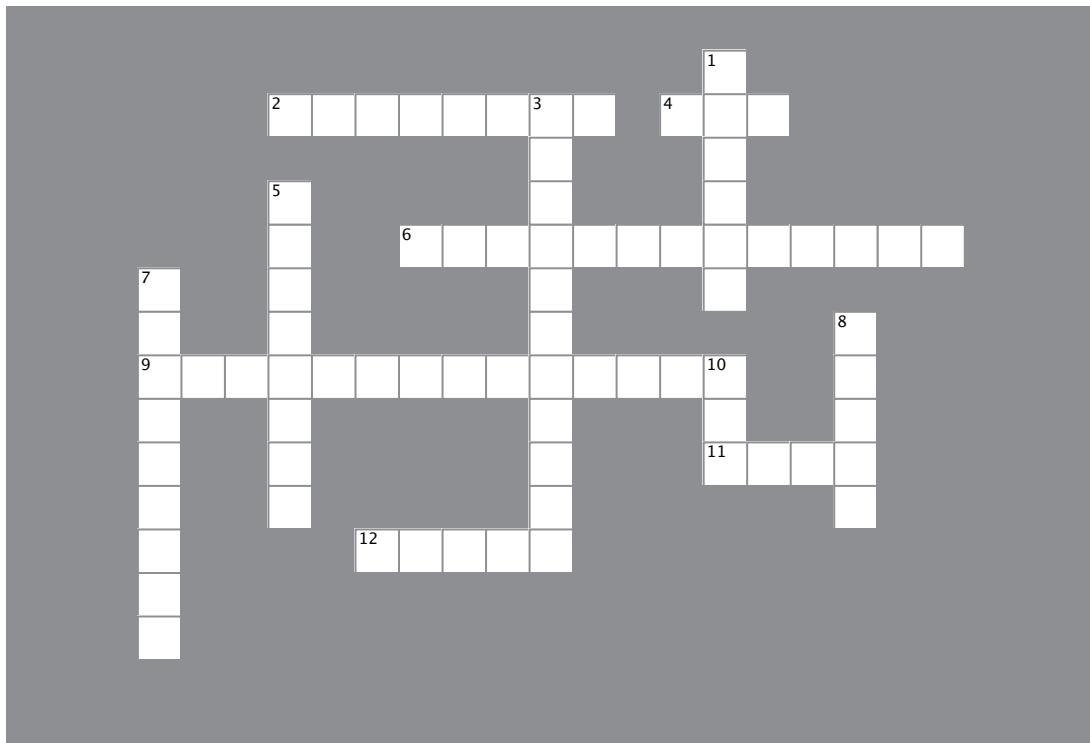
画像のある
ブログエントリ。





JavaScript クロスワード

ではドリルです…箱に語を埋めてください！



ヨコのカギ

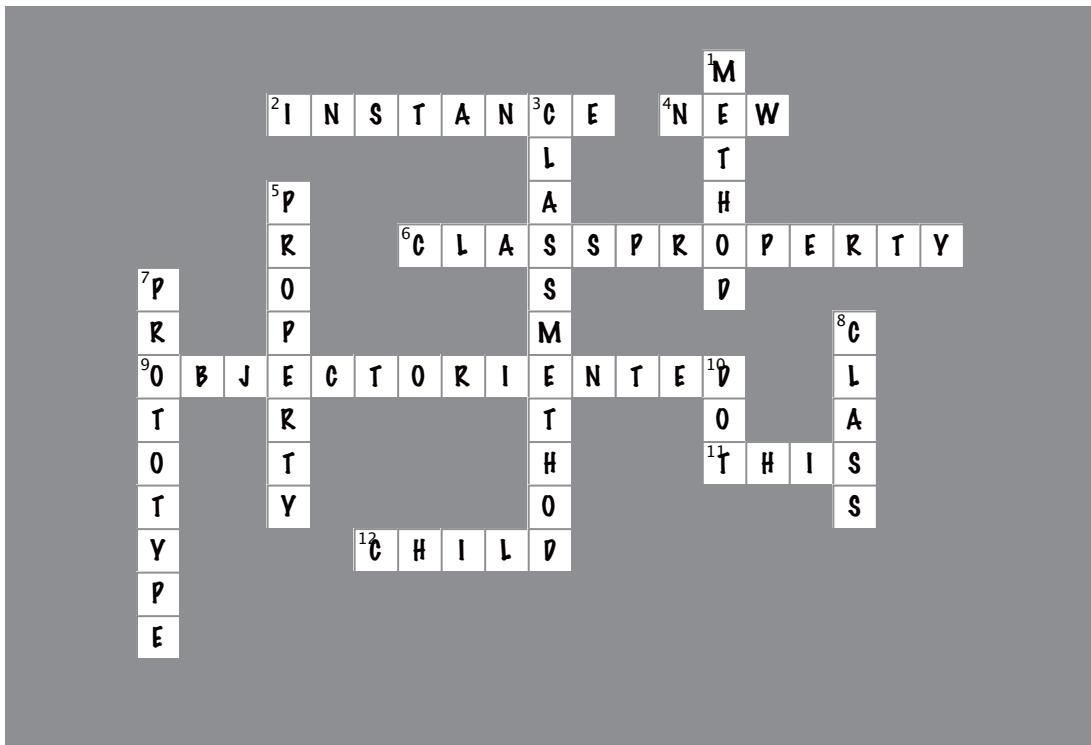
2. 固有のデータを持つ実際のオブジェクト。
 4. オブジェクトインスタンスを作成するのに使う演算子。
 6. YouCubeの署名はこれらのひとつです。
 9. オブジェクトを使うように設計されたソフトウェア。
 11. オブジェクト自体のコードからオブジェクトを参照するのに使うキーワード。
 12. 別のオブジェクトからプロパティやメソッドを継承するオブジェクト。

タテのカギ

1. オブジェクトにおいて関数に相当するもの。
 3. YouCubeのブログのソート用比較コードはこれら
のひとつです。
 5. オブジェクトの中にあるデータ断片は と呼ば
れます。
 7. あらゆるオブジェクトには、これらのオブジェクト
のひとつが隠されています。
 8. オブジェクトのインスタンスを作成するのに使われ
るテンプレート。
 10. オブジェクトのプロパティやメソッドのアクセスに
使われる表記法。



JavaScriptクロスワードの答え

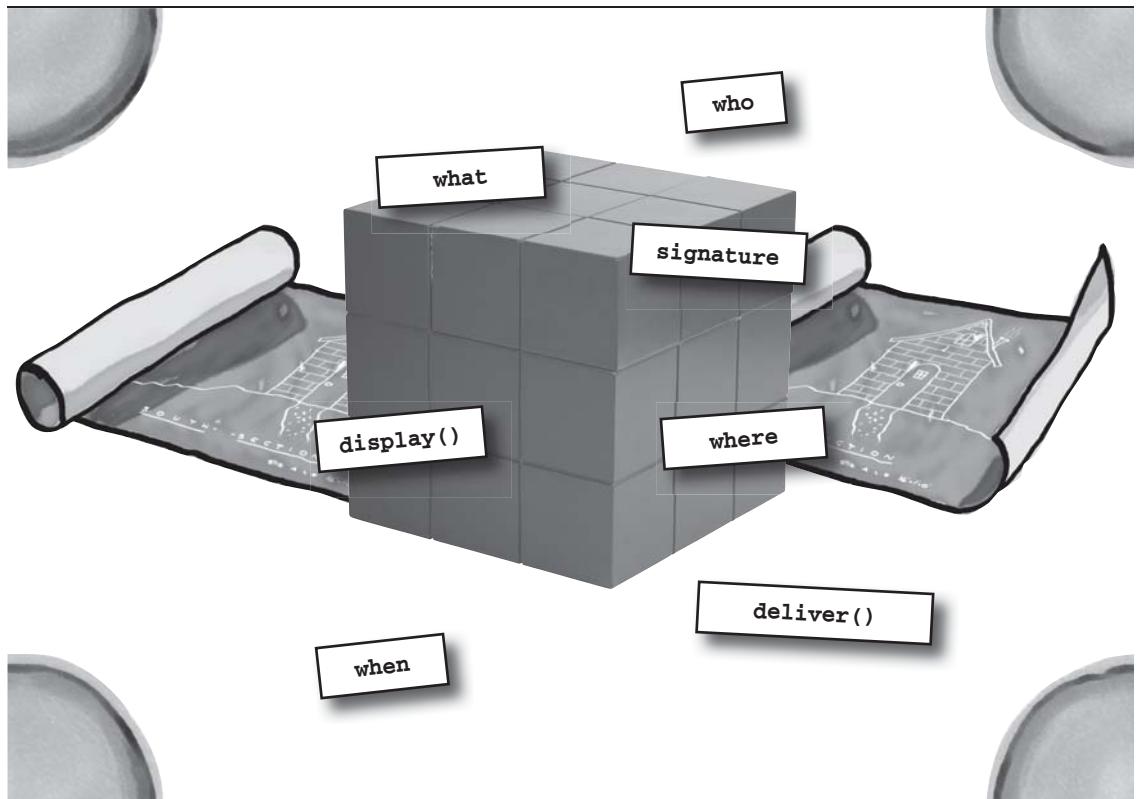
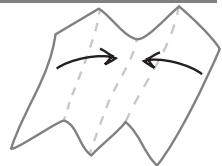


折り畳みページ

脳のところで
折り畳んでから
謎を解決しましょう。

ほとんどのスクリプトにオブジェクトは
何を追加するのでしょうか？

左脳と右脳がご対面！



オブジェクトはスクリプトにたくさんのこと들을追加するので、
ひとつだけ取り上げるのは難しいです。いくつかのオブジェクトは他の
オブジェクトより勝っているだけでなく、もっと強力にすることさえできます。
いずれにせよ、答はもう明らかですね。

11章 バグをなくせ

*良いスクリプトも悪くなる



どんなに慎重に考えられたJavaScriptでもエラーになることがあります。そんなときは、慌てないことです。JavaScriptの優秀な開発者だったら、ぜったいバグのないスクリプトを書ける、なんて嘘です。バグの原因をつきとめて、それをなくすことができるのが優秀な開発者なのです。JavaScriptのバグを最高の腕前で駆除できる人は、目に見えにくく意地の悪いバグができるだけ少なくするために、優れたやり方で開発を進めます。ちょっとした予防策をとることで、長持ちするコードになるのです。とはいえ、バグがみつかったら、それと戦うための武器が必要になります。

現実のデバッグ

ほんとにあったショッキングな話。チョコバーに小さな虫が60匹もうごめいでいたそうです。ちょっとした話だけど怖いですね。でもJavaScriptコードの虫（バグ）を怖がることはあります。JavaScriptコードはチョコレートの製造機械より厳重に管理することができます。それにJavaScriptのバグを取り除く専門のタスクフォースもあるのです。



BSI (Bug Scene Investigators) は、バグ対策調査の専門会社です。BSIにJavaScriptの調査員として入社したオーエンは、ウェブからJavaScriptのバグを取り除く仕事に熱心に取り組んでいます。

オーエンは、BSIの
JavaScript検査官。
チョコレートが
大好きです。

オーエンは気をひきしめて
取り組むべき課題がいくつか
抱えています。JavaScriptの
凄腕探偵になるには、
JavaScriptのデバッグの
黒魔術を習得する必要が
あります。



マニア向け情報

米国の食品医薬品局によると、
チョコバーに「虫のかけら」が混
入していても、最大60までは許容され
るそうです。こうした「現実世界」とは
対照的に、BSIの社員はJavaScriptの
バグを1つも許さないポリシーで取り組
んでいます。

バグだらけのIQ計算機のケース

オーエンの最初の課題はIQ計算スクリプトです。IQの配列からIQの平均を計算するページの一部になっています。平均が近いユーザをグループにまとめる処理も行います。スクリプトには数値の配列が渡され、これをもとに平均を計算し結果を表示します。

「このスクリプトはバグだらけで動かないよ」と言われたオーエンには、いまのところなんの手掛りになる情報はありません。

オーエンのファイルは
<http://www.headfirstlabs.com/books/hfjs/>から
 ダウンロードできます。

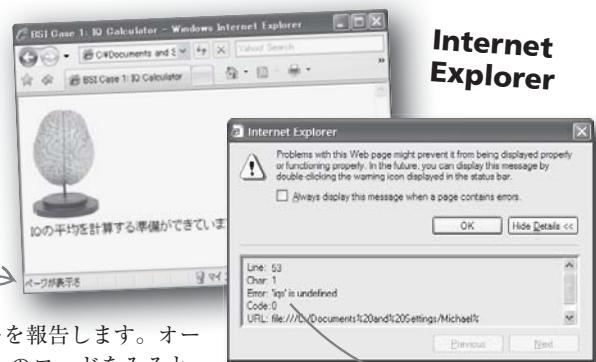


別のブラウザを試してみる

問題のあるスクリプトを別のブラウザで動かしてみれば、問題の手掛りが得られるかもしれません。オーエンはまず Internet Explorer で動かしてみました。

IEの左下にある黄色のマークをダブルクリックするとエラーウィンドウが開きます。

Internet Explorer は最初にページを読み込むときエラーを報告します。オーエンはそれを信用していいか半信半疑です。スクリプトのコードをみると、変数 `iqs` は存在していますが、IE ブラウザはそれがないと言っています。ブラウザがエラーを報告するとき、かならずしも内容が正確ではないことを知っているので、彼は次に Safari ブラウザを試してみることにしました。

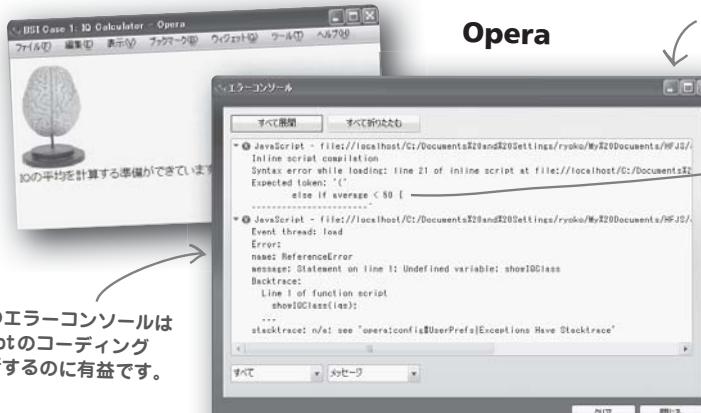


変数 `iqs` はコードで定義されているので IE のエラー表示は意味がありません。



コードを 1 行目から数えていくと、最初問題ないと思えた行にエラーがあることを Safari は指摘しています。

Safari はまったく別の行をエラーで報告しました。オーエンはそれをみても、何が悪いのかすぐにはわかりません。そこでさらに Opera でエラーがどうなるか試してみることにしました。



行番号は違いますが Opera が指摘したコードのエラーは Safari のエラーと同じです。

ブラウザのエラーコンソールは JavaScript のコーディング問題を診断するのに有益です。

なにか変です。Operaが表示する行番号はSafariと違っていますが、あきらかに同じ行を指しています。オーエンにとっては一歩前進ですが、やはりコードのどこが悪いのかわかりません。そこでさらにFirefoxを試してみました。

Firefoxが指摘する問題のあるコードの行番号もさらに違っています。
Firefoxはバグを正確に突きとめてくれるので便利です。

IQの平均を計算する準備ができています。

```

<html>
  <head>
    <title>BSI Case 1: IQ Calculator</title>
    <script type="text/javascript">
      var iqs = [ 113, 97, 86, 75, 92, 105, 146, 77, 64, 114, 165, 96,
        function showIQClass(data) {
          alert("OKボタンをクリックして IQ計算をスタート");
          document.getElementById("output").innerHTML = "You are dealing with <em>" +
        } // IQの平均値を等級に変換します
        var average = 0;
        for (var i = 0; i < data.length; i++) {
          average += data[i];
        average = Math.floor(average / data.length);
      }
      if (average < 20) {
        return "テレビを取り上げたほうかよいかも";
      } else if (average < 50) {
        return "猛烈勉強が必要かも";
      } else if (average < 70) {
        return "勉強が必要かも";
      } else if (average < 81) {
        return "頭の体操が必要かも";
      } else if (average < 91) {
        return "少し鈍いかも";
      } else if (average < 111) {
        return "普通の知性の持ち主";
      } else if (average < 121) {
        return "優れた知性の持ち主";
      } else if (average < 141) {
        return "非常に優れた知性の持ち主";
      } else {
        return "天才";
      }
    </script>
  </head>
  <body onload="showIQClass(iqs);">
    
    <div id="output">IQの平均を計算する準備ができています。</div>
  </body>
</html>
```

エラー: missing (before condition
ソースファイル: file:///Volumes/editors/work/Ryoko/app/bsi/case1_1.html
else if average < 70 {
行: 25

式を評価 行: 1

あ、これが原因かな。

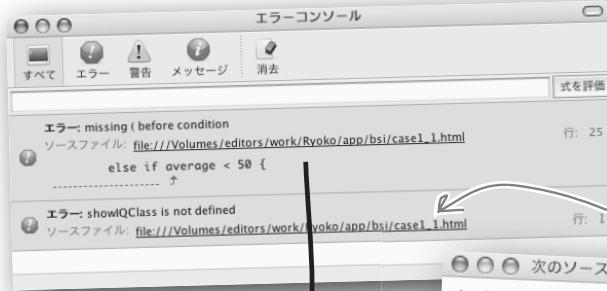
Firefoxはエラーの原因がこの行だと指摘しています。

頭の体操

ウェブブラウザの力を借りてオーエンが見つけたコーディングエラーはどれでしょう？

Firefoxで解決

Firefoxが表示するバグの説明は独特ですが、オーエンはFirefoxを使って原因を掘り下げることにしました。Firefoxのエラーコンソールウィンドウでリンクをクリックしたら、問題のある行の前にある行が表示されました。



リンクをクリックすると
ウェブページのコードが
開きます。疑わしい
コードの前でハイライト
表示されます。

これが問題の原因となった
コードです。

Firefoxはデバッグに最適なブラウザと
認められています、少なくとも現時点では。



ミニア向け情報

Firefoxは組み込みのバグ検出能力
に優れているだけではありません。
Firebugと呼ばれるデバッガプラグイン
があり、これを使うとデバッグ作業全体のレベ
ルが上がります。FirefoxのFirebugデバッガは
<http://www.getfirebug.com/>から自由にダウ
ンロードできます。

しばらく2番目のエラーは無視して、
最初のエラーに取り組みましょう。

```
<html>
<head>
<title>BSI Case 1: IQ Calculator</title>
<script type="text/javascript">
var iqS = [ 113, 97, 86, 75, 92, 105, 146, 77, 64, 114, 165, 96, 97, 88, 108
function showIQClass(data) {
    alert("OKボタンをクリックしてIQ計算をスタート");
    document.getElementById("output").innerHTML = "あなたは <em>" + calcIQClass(data) + "</em>。";
}

function calcIQClass(data) {
    // IQの平均を計算します
    var average = 0;
    for (var i = 0; i < data.length; i++) {
        average += data[i];
    }
    average = Math.floor(average / data.length);

    // IQの平均値を等級に変換します
    if (average < 20) {
        return "テレビを取り上げたほうがよいかも";
    }
    else if (average < 50) {
        return "勉強が必要かも";
    }
    else if (average < 70) {
        return "勉強が必要かも";
    }
}
</script>
```

Firefoxのエラーメッセージを分析したら、Safariが指摘した行番号(25)
は実際には問題ないことがわかりました。Firefoxがハイライト表示した
のは24行目で、25行目のコードは問題ないことを示しています。Firefoxは、
見落としがちな問題であっても、コードの問題を正確に説明します。

else if average < 50 {

↑
if文でテスト条件を
囲む括弧がありません。

素朴な疑問に答えます

Q: ブラウザにエラーコンソールを表示させる方法がわかりません。どうやって開けばいいんですか？

A: 残念なことに、ブラウザによってJavaScriptのエラーコンソールを表示する方法は異なります。MacのSafariでは、開発メニューからエラーコンソールを選びますが、デフォルトでは無効になっています。この設定を変更して開発メニューを有効にするには、以下のコマンドを1行にしてターミナルアプリケーションで実行する必要があります。

```
defaults write com.apple.Safari
IncludeDebugMenu 1
```

エラーコンソールを開いてスクリプトエラーを表示する方法については、それぞれのブラウザのドキュメントで調べてください。Firefoxのエラーコンソールは、ツールメニューからエラーコンソールを選べば開きます。

Q: Firefoxはなぜ他と違うのでしょうか？

A: Firefoxの開発者はエラーを報告する機能を充実させています。スクリプトのエラーを的確に検出し表示できる点が他のブラウザより優れているだけです。他のブラウザがそのうち追いつき、追い越す、という意味ではなく、JavaScriptコードを含むページのデバッグに非常に優れたブラウザだということです。

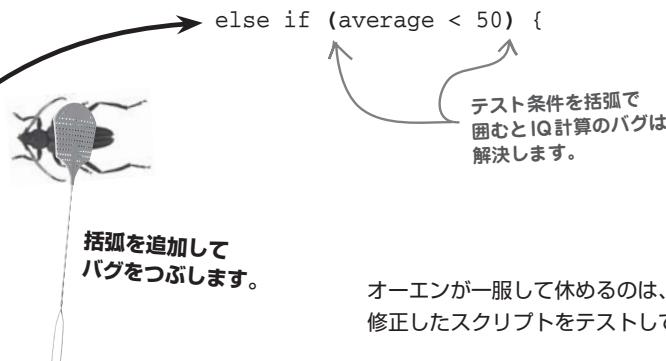
Q: インターネットエクスプローラはどんなエラーを報告しますか？

A: エラーを確実に知る方法が実はありません。インターネットエクスプローラが報告するエラーは、プロパティを読み込まないスクリプトコードに関するものなので、JavaScriptインタープリタが検出したエラーの結果になります。変数iQSが「undefined」と報告されているので、コードの読み込みが正しく行われなかったことがわかります。iQSを作成するコードになっていたとしても。このエラーが発生する原因是、どこか他にエラーがあって、そのエラーがスクリプトの完全な読み込みを妨げていると考えられます。

そのため、スクリプトに他にエラーがないか探し、「undefined」になった他の原因を調べる必要があります。

デバッグを楽にする

IQ計算スクリプトのバグを早々に特定できたので、オーエンはすっかり興奮しています。コードの修正も簡単なので、BSIの仕事を続けていく自信がつきました。



オーエンが一服して休めるのは、まだ先のようです。
修正したスクリプトをテストしてみる必要があります。

このデバッグは
簡単だな。Firefoxの力を
ちょっと借りて片付けよう。
コーヒーとケーキが
待ってるぞ。



報告されるバグがいつもバグの原因とは限りません。

残念ながら、IQ計算機のバグはまだつぶれていません。Firefoxはまったく別の問題を報告してきました。Firefoxの報告を信用して同じやり方を続けたいオーエンですが、今回のバグに関してはすこし疑っています。

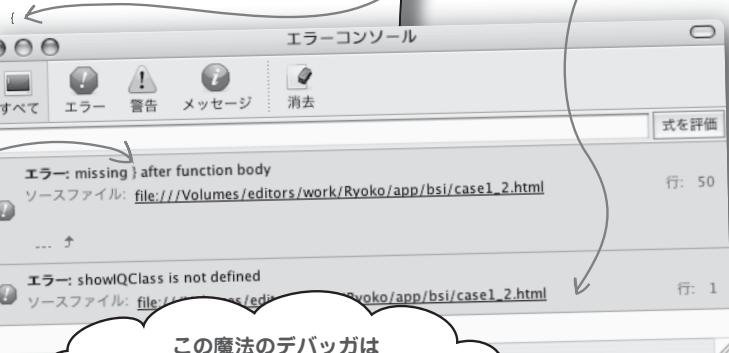
この関数の波括弧は Firefox が指摘した波括弧と対応が取れているので、関数の波括弧に関しては問題ありません。

この波括弧が見つからないと Firefox は指摘しています。

```
<html>
<head>
<title>BSI Case 1: IQ Calculator</title>
<script type="text/javascript">
var iqgs = [ 113, 97, 86, 75, 92, 105, 146, 77, 64, 114, 165, 96, 97, 88, 108 ];
function showIQClass(data) {
    alert("OKボタンをクリックしてIQ計算をスタート");
    document.getElementById("output").innerHTML = "You are dealing with <em>" +
}
function calcIQClass(data) {
    // IQの平均を計算します
    var average = 0;
    for (var i = 0; i < data.length; i++) {
        average += data[i];
    }
    average = Math.floor(average / data.length);
    // IQの平均を等級に変換します
    if (average < 20) {
        return people who score below 20
    } else if (average < 50) {
        return "テレビを取り上げる";
    } else if (average < 70) {
        return "猛烈強が需要かも";
    } else if (average < 81) {
        return "勉強が必要かも";
    } else if (average < 91) {
        return "頭の体操が必要かも";
    } else if (average < 111) {
        return "少し鈍いかも";
    } else if (average < 121) {
        return "普通の知性の持ち主";
    } else if (average < 141) {
        return "非常に優れた知性の持ち主";
    } else {
        return "天才";
    }
}
</script>
</head>
<body onload="showIQClass(iqgs);">

<br />
<div id="output">IQ の平均を計算する準備ができます。</div>
</body>
</html>
```

ここでも引き続きこのバグについて無視することにしましょう。



この魔法のデバッガは
メチャクチャだわ。このコードの関数を
囲む括弧は間違ってないのに。



いつもブラウザを信用できるとは限りません。

関数の括弧には問題がありません。Firefoxは見当違いのバグを報告しているように見えます。とはいっても、波括弧に関して問題が報告されているので、これを手掛りにコードの中に波括弧の抜けがないか詳しく調べてみる価値はあります。

インタープリタになつてみよう

```

<html>
  <head>
    <title>BSI Case 1: IQ Calculator</title>

    <script type="text/javascript">
      var iq = [ 113, 97, 86, 75, 92, 105, 146, 77, 64, 114, 165, 96, 97, 88, 108 ];

      function showIQClass(data) {
        alert("OKボタンをクリックして IQ 計算をスタート");
        document.getElementById("output").innerHTML = "You are dealing with <em>" +
          calcIQClass(data) + "</em>.";
      }

      function calcIQClass(data) {
        // IQ の平均を計算します
        var average = 0;
        for (var i = 0; i < data.length; i++) {
          average += data[i];
        }
        average = Math.floor(average / data.length);

        // IQ の平均値を等級に変換します
        if (average < 20) {
          return "テレビを取り上げたほうがよいかも";
        }
        else if (average < 50) {
          return "猛勉強が必要かも";
        }
        else if (average < 70) {
          return "勉強が必要かも";
        }
        else if (average < 81) {
          return "頭の体操が必要かも";
        }
        else if (average < 91) {
          return "少し鈍いかも";
        }
        else if (average < 111) {
          return "普通の知性の持ち主";
        }
        else if (average < 121) {
          return "優れた知性の持ち主";
        }
        else if (average < 141) {
          return "非常に優れた知性の持ち主";
        }
        else {
          return "天才";
        }
      }
    </script>
  </head>

  <body onload="showIQClass(iq);">
    
    <br />
    <div id="output">IQ の平均を計算する準備ができます。</div>
  </body>
</html>
```

JavaScript インタープリタになったつもりで
コードの波括弧を追って、どこで
間違ってるか見つけましょう。



インタープリタになつてみようの答え

```

<html>
  <head>
    <title>BSI Case 1: IQ Calculator</title>

    <script type="text/javascript">
      var iqs = [ 113, 97, 86, 75, 92, 105, 146, 77, 64, 114, 165, 96, 97, 88, 108 ];

      function showIQClass(data) {
        alert("OKボタンをクリックして IQ 計算をスタート");
        document.getElementById("output").innerHTML = "You are dealing with <em>" +
          calcIQClass(data) + "</em>.";
      }

      function calcIQClass(data) {
        // IQ の平均を計算します
        var average = 0;
        for (var i = 0; i < data.length; i++) {
          average += data[i];
          average = Math.floor(average / data.length);
        }

        // IQ の平均値を等級に変換します
        if (average < 20) {
          return "テレビを取り上げたほうがよいかも";
        }
        else if (average < 50) {
          return "猛勉強が必要かも";
        }
        else if (average < 70) {
          return "勉強が必要かも";
        }
        else if (average < 81) {
          return "頭の体操が必要かも";
        }
        else if (average < 91) {
          return "少し鈍いかも";
        }
        else if (average < 111) {
          return "普通の知性の持ち主";
        }
        else if (average < 121) {
          return "優れた知性の持ち主";
        }
        else if (average < 141) {
          return "非常に優れた知性の持ち主";
        }
        else {
          return "天才";
        }
      }
    </script>
  </head>
  <body onload="showIQClass(iqs);">
    
    <br />
    <div id="output">IQ の平均を計算する準備ができます。</div>
  </body>
</html>

```

ここに波括弧の閉じるがないので、変数 averageへの加算が波括弧で囲まれていません。

JavaScript インタープリタになったつもりでコードの波括弧を追って、どこで間違ってるか見つけましょう。



この波括弧の開くに対応する波括弧の閉じるがありません。



波括弧を追加してバグをつぶす。

この for ループはコードを 1 行しか実行しないので、for ループの後にある波括弧の開くを削除できます。ただし波括弧で囲んだ方がループで実行されるコードが明確になります。

波括弧の対応がとれてないのは JavaScript でよくあるバグです。このバグは細部に気をつけければ避けられます。

変数が未定義になる

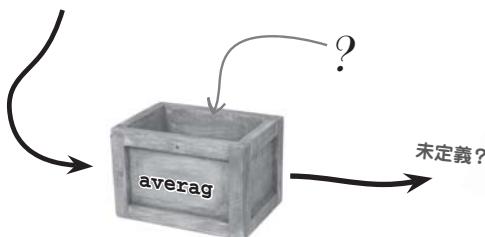
IQ計算機のバグがなくなるないので、オーエンは一服できそうもありません。Firefoxは変数が「not defined」(定義されていない)と報告しています。これはInternet Explorerが報告した偽のエラーに似ています。ただし今回報告された未定義の変数は `iqs` ではなく `averag` です。

2番目のエラーが消えました。
バグを1つぶすと、他の
エラーも解消することが
あります。



```
else if (averag < 81) {
    return "頭の体操が必要かも";
}
```

自信はないけど、
変数未定義になるのは
変数が作成されて
いないからかな。



自分で考えてみよう

オーエンが調べたバグの文脈における「未定義」の意味は何だと思いますか？

.....

.....

.....



オーエンが調べたバグの文脈における「未定義」の意味は何だと思いますか？

「未定義」は、変数が(varを使って)作成されていないか、作成されても値が代入されていないことを示します。どちらにせよ問題は、値のない変数を参照していることがあります。

ちょっとしたミスが原因になるもあります

この場合の "undefined" は、変数が作成されていないのに、その変数を使おうとしたことを示しています。変数が undefined になる原因是、変数名のタイプミスにあります。JavaScript インタープリタはそれを新しい変数とみなしてしまいます。

ちょっとしたタイプミスでもスクリプトに大被害をひきおこすことがあります。

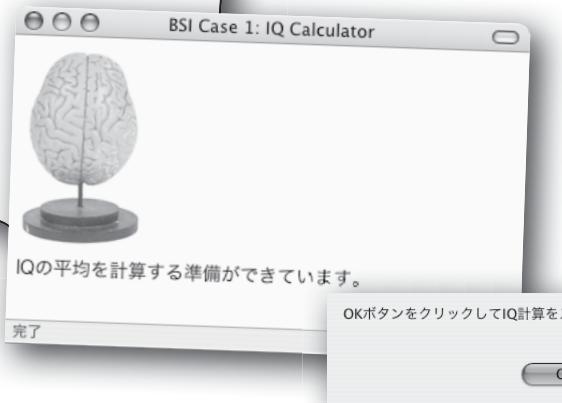
```
function calcIQClass(data) {
    // IQの平均を計算します
    var average = 0;
    for (var i = 0; i < data.length; i++) {
        average += data[i];
    }
    average = Math.floor(average / data.length);
    // IQの平均値を等級に変換します
    if (average < 20) {
        return "テレビを取り上げたほうがよいかも";
    } else if (average < 50) {
        return "猛烈勉強が必要かも";
    } else if (average < 70) {
        return "勉強が必要かも";
    } else if (average < 81) {
        return "頭の体操が必要かも";
    } else if (average < 91) {
        return "少し鋭いかも";
    } else if (average < 111) {
        return "普通の知性の持ち主";
    } else if (average < 121) {
        return "優れた知性の持ち主";
    }
}
```

タイプミスを見つけるのは大変ですが、修正するのは簡単です。

else if (average < 81) {
 return "people who should consider
 brain exercises";
}

タイプミスされた変数を修正すると、未定義の変数の問題は解決します。

averag
!=
average



OKボタンをクリックしてIQ計算をスタート

素朴な疑問に 答えます

Q: "undefined"と"not defined"は違うのでしょうか?

A: いいえ、意味はまったく同じです。ブラウザによってどちらを使うかが違うだけです。

Q: わかりました。では

Q: "undefined"とnullは違うのでしょうか?

A: これは単純ではないですね。極めて技術的レベルでいうと、違いがあります。だけど、心配することはあります。nullと違って "undefined"は変数に代入することができる値ではありません。変数に値がまだ代入されていないとき、自動的に undefinedというデータ型が想定されるのです。一方、オブジェクトの初期化の際に、オブジェクトの変数を nullに設定しておくと、オブジェクトがすでに作成されたかどうかが明確になります。技術的な核心部分の詳細はひとまず置いておいて、"undefined"

数として解釈するだけです。この新しい変数にはまだ値が代入されていないので、if文の中でこれを何かと比較すると、問題になります。どの映画を見るか決めてもらいないときに、映画のレビューを書くようなものです。

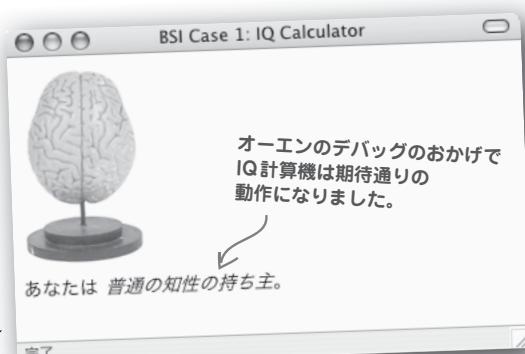
Q: 私をからかってるんですか?

Q: ワープロでいつもタイプミスしていますが、それが原因でエラーになったりしませんよ。

A: はい、深呼吸して落ち着きましょう。習うより慣れろ、といいますよね。スクリプトは人間に読ませるのではなく、機械に読ませるためにあります。機械はスクリプトが一文字でも間違っていると、スクリプトを発狂させてしまいます。JavaScriptコードでは空白や改行などは多少融通がききますが、コードそのものは一字一句正確でなければならないのです。

IQの計算が成功しました

IQ計算スクリプトは、タイプミスによるバグが解決し、いまでは正しく動作しています。配列から平均を計算し、結果をテキストで表示します。オーエンはこの課題を終了し、満足感にひたっています。でも、それも長くは続きません。



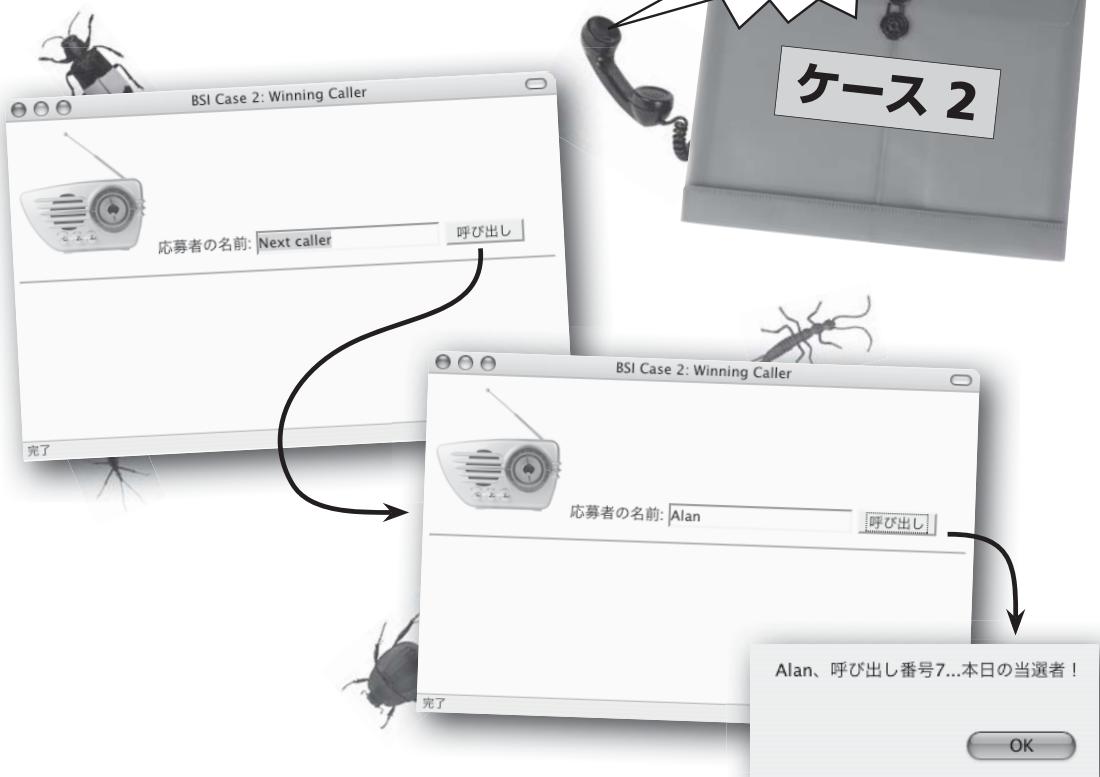


重要ポイント

- ほとんどのブラウザはJavaScriptエラーの情報を表示するエラーコンソールを提供していますが、その内容がいつも完全に正確であると信頼することはできません。
- ときどきブラウザは不完全なエラー情報を表示してしまいますが、通常はトラブルの原因を探す手掛りになります。
- コードのブロックを囲む波括弧はバグの原因になりやすいので、波括弧の開くと閉じるは常に対応させるように注意しましょう。
- ちょっとしたタイプミスで簡単にエラーになりますが見つけるのは簡単ではありません。識別子の名前は常にチェックしましょう。

ラジオ番組の着呼のバグ

オーエンが最初の課題を解決して喜んだのも束の間、次の課題があがってきました。ラジオ局のコンテストへの電話の着呼を処理するためのスクリプトです。呼び出し番号によって当選者が決まります。このスクリプトは電話をかけた人の数を数えて、呼び出し番号がたとえば7だったら、7番目にかけた人を当選者にします。



調査の開始

ラジオ局の着呼ページをブラウザで調査する前に、オーエンはコードを調べて、おおよそどんな動作になっているのか感触をつかもうと思いました。あきらかに間違っているコードが見つかるかもしれませんし、そうでなくとも、すくなくともどんな動作を想定したコードなのかを理解できるからです。

formオブジェクトと一緒に応募者の名前と当選者の番号をcheckWinner()に渡します。

```

<html>
  <head>
    <title>BSI Case 2: Winning Caller</title>
    <script type="text/javascript">
      // 呼び出しの総数
      var callNum = 0;

      function checkWinner(form, caller, winningNum) {
        // 呼び出し番号をインクリメント
        var callNum;
        ++callNum;

        // 当選者をチェック
        if (callNum == winningNum) {
          alert(caller + "、呼び出し番号 " + callNum + "... 本日の当選者！");
          form.submit();
        }
        else {
          // 次の応募者のためにフィールドをリセット
          var callerField = document.getElementById('caller');
          callerField.value = "次の応募者";
          callerField.focus(); ← focus() は入力フォーカスをページの要素に設定します。
          callerField.select();
        }
      }
    </script>
  </head>
  <body>
    <form name="callform" action="radiocall.php" method="POST">
      
      Caller name: <input id="caller" name="caller" type="text" />
      <input type="button" value="呼び出し" onclick="checkwinner(this.form, document.getElementById('caller').value, 7)" />
    </form>
  </body>
</html>

```

呼び出しカウンタを0に初期化します。

呼び出しカウンタをインクリメントします。

当選者がいなかった場合次の応募者のためにフィールドをリセットします。

呼び出し番号が当選番号と等しかったら、ユーザにアラートを表示し、フォームを送信します。

select() はテキスト要素に格納された値を選択します。

CallボタンがクリックされたときcheckWinner()を呼び出します。

当選者を格納するサーバスクリプト。



自分で考えてみよう

オーエンを助けましょう。ラジオ局の着呼スクリプトコードにあるバグを数えて、該当する数を丸で囲みましょう。

None

One

Two

Three

Four

Five



オーエンを助けましょう。ラジオ局の着呼スクリプトコードにあるバグを数えて、該当する数を丸で囲みましょう。

None

One

Two

Three

Four

Five

オーエンが4つのバグを見つけて解決するのを手伝いましょう…

構文が妥当か調べる（バグその1）

どんな動作を想定しているコードなのか、おおよそ把握したら、Firefoxを使ってスクリプトが実行されるとき実際に何が起きているか見てみましょう。これまで見てきたエラーと同じように、Firefoxはすぐに構文エラーを報告します。JavaScript言語の文法に違反しているコードのエラーがあるということです。

エラー報告が有効になっている場合、ブラウザは常に構文エラーを報告します。

エラー: syntax error
ファイル:///Volumes/editors/work/Ryoko/app/bsi/case2_1.html 行: 16
alert(caller + ", caller number " + callNum + "...today's winner!");
^

var callNum;
++callNum;

```
// Check for a winner
if (callNum == winningNum) {
    alert(caller + ", caller number " + callNum + "...today's winner!");
}
else {
    // 次の応募者のためにフィールドをリセット
    var callerField = document.getElementById('caller');
    callerField.value = "Next caller";
    callerField.focus();
    callerField.select();
}
```

</script>
</head>

行 16, 列 16

文字列を連結するコードの行にエラーがあると指しています。

構文エラーがあると、ブラウザがエラーを報告できる設定になつていれば、かならずなんらかのメッセージが表示されます。これはエラーの原因を突き止める際の重要な手掛りになります。

文字列に注意する

Firefoxは文字列を連結している行をピンポイントで指摘しています。この行を手掛りに注意深く分析してみましょう。このコードは、`caller`と`callNum`を文字列リテラルと連結して`alert()`を呼び出しています。

```
if (callNum = winningNum)
    alert(caller + "呼び出し番号" + callNum + "...本日の当選者！");
```

↑
↑
この2つの文字列リテラルは
2つの変数と連結されます。

引用符は常に
対になっているのが
重要なんですね？



JavaScriptのコードでは引用符の対が重要です

引用符は常に対になっている必要があります。そうでないと、JavaScriptは、文字列がどこで終わり、次の文字列がどこから始まるのか、わからなくなってしまいます。ラジオ局の着用コードの場合、文字列連結にある文字列リテラルで閉じる引用符が抜けています。このためJavaScriptは文字列の終わりがわからず、構文エラーになります。

バグを修正するために、文字列の終わりに二重引用符を追加します。

```
if (callNum = winningNum)
```

```
    alert(caller + "呼び出し番号" + callNum + "...本日の当選者！");
```

```
if (callNum = winningNum)
```

```
    alert(caller + "呼び出し番号" + callNum + "...本日の当選者！");
```

文字列の中の
引用符を修正して
このバグをつぶします。



二重引用符と単一引用符を一貫させる

引用符が対になっていないエラーは、文字列の引用符に関するエラーの半分にすぎません。JavaScript と HTML どちらも、二重引用符と単一引用符を使って文字列 (JavaScript) や属性 (HTML) を囲みます。二重引用符と単一引用符を混在させるときは、それらを一貫させることが重要です。

```
<input type="button" value="呼び出し" ←  
    onclick="checkwinner(this.form, document.getElementById('caller').value, 7)" />
```

二重引用符
二重引用符
を使って HTML 属性を
囲みます。

このように HTML 属性に二重引用符を使い、JavaScript 文字列に単一引用符を使うアプローチは完璧に動きます。また二重引用符と単一引用符を入れ替えることも可能です。

```
<input type='button' value='呼び出し'  
    onclick='checkwinner(this.form, document.getElementById("caller").value, 7)' />
```

HTML 属性は単一引用符で
囲まれているので JavaScript
文字列は二重引用符で囲みます。

最近の XHTML ウェブページ
標準では属性を単一引用符で
囲むことはできません。

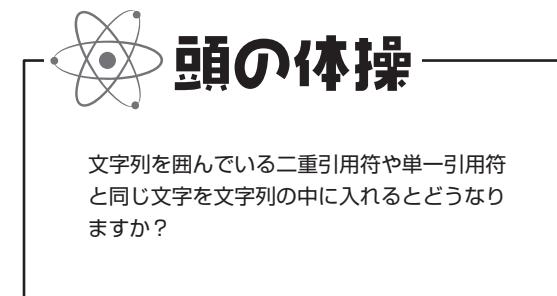
コードの種類に応じた二重引用符と単一引用符の使い分けを一貫させることが重要です。HTML のバージョンである XHTML では属性を二重引用符で囲む必要があるので、属性の中にある JavaScript 文字列を囲むときは單一引用符を使います。

すでに囲まれている文字列の中に二重引用符や単一引用符を指定する必要があるときは、問題が発生します。以下のコードをみてみましょう。

```
alert('It's so exciting!');
```

このコードは
動作しますか？

HTML 属性で
JavaScript の文字列を
使うときは二重引用符
と単一引用符を互いに
入れ替えます。



引用符を引用符として使わないときは エスケープ文字を追加する

文字列の中で二重引用符や单一引用符をそのまま文字として使うと、それが文字列の境界として解釈されてしまうので、バグになってしまいます。JavaScript インタープリタは、单一引用符が文字列の境界なのか、それともアポストロフィなのか、区別がつかないので、アラートコードで構文エラーが表示されたわけです。幸いにして、引用符を文字として宣言する簡単な方法があります。**エスケープ文字**がそれです。文字の前にバックスラッシュ(\)を追加することで、文字通りの文字として扱うことができます。

```
alert('It \'s so exciting!');
```

单一引用符をエスケープしたので、JavaScript はこれが文字列の終わりではなく、文字列の中のアポストロフィ文字として扱ってほしいことがわかります。もちろん、文字列を二重引用符で囲んで、エスケープを使わないですますやり方も可能です。

```
alert("It's so exciting!");
```

これは問題ありません。でも次のコードはどうでしょう。

```
alert("They said, "you've won! "");
```

文字列に二重引用符と单一引用符の両方が含まれる場合、エスケープを使うしかありません。このような場合、单一引用符をエスケープする必要はないのですが、すべてのリテラル文字をエスケープしておいた方が安全です。

```
alert("They said, \"you've won! \"");
```

エスケープする必要はありません。

エスケープする必要はありませんが安全です。



エクササイズ

以下のコード断片にある二重引用符と单一引用符をできるかぎりエスケープ文字を使って修正してください。

```
var message = 'Hey, she's the winner!';
```

```
var response = "She said, "I can't believe I won. ""
```

```
<input type="button" value="Winner" onclick="givePrize("Ruby");" />
```



以下のコード断片にある二重引用符と単一引用符をできるかぎりエスケープ文字を使って修正してください。

エクササイズ

答え

```
var message = 'Hey, she's the winner!';  
var message = 'Hey, she\'s the winner!';  
  
二重引用符で var response = "She said, \"I can't believe I won.\""  
文字列が  
囲まれて var response = "She said, \'I can't believe I won.\'"  
いるので、  
その中にある  
单一引用符を  
エスケープ  
する必要は  
ありません。
```

```
<input type="button" value="Winner" onclick="givePrize("Ruby");" />  
<input type="button" value="Winner" onclick="givePrize('Ruby');"/>
```

HTML 属性の中に JavaScript 文字列があるので
エスケープ文字は動作しません。二重引用符と
単一引用符を併用すると問題が解決します。

undefinedは変数のためだけにあるのではない(バグその2)

バグのひとつはつぶれましたが、オーエンの仕事はまだ終わりません。ラジオ局の着呼スクリプトはエラーが表示されなくなりましたが、名前を入力して「呼び出し」ボタンをクリックすると、別の問題が発生します。どうやら checkWinner() に問題があるようです。

応募者の名前: Haward

呼び出し

呼び出しボタンをクリックすると checkWinner() でエラーが発生します。

完了

何らかの原因で
関数が定義されて
いません。

エラー: checkwinner is not defined
ソースファイル: file:///Volumes/editors/work/Ryoko/app/bsi/case2_2.html

行番号は役に立ちません。
ページの HTML コードの
第 1 行には問題ありません。

ありがちなエラーのチェックリスト

デバッガの経験をつんだオーエンは、JavaScriptのありがちなエラーのチェックリストを新しく作ることにしました。今回のバグもこれらのバグのどれかに該当するかもしれません。

checkWinner()
はコードの
2カ所で記述
されています。

```
function checkWinner(form, caller, winningNum) {
    //呼び出し番号をインクリメント
    var callNum;
    ++callNum;

    //当選者をチェック
    if (callNum == winningNum) {
        alert(caller + "、呼び出し番号" + callNum + "...本日の当選者！");
        form.submit();
    } else {
        //次の応募者のためにフィールドをリセット
        var callerField = document.getElementById('caller');
        callerField.value = "次の応募者";
        callerField.focus();
        callerField.select();
    }
}
</script>
</head>

<body>
<form name="callform" action="radiocall.php" method="POST">
    
    Caller name: <input id="caller" name="caller" type="text" />
    <input type="button" value="呼び出し"
        onclick="checkwinner(this.form, document.getElementById('caller').value, 7)" />
</form>
</body>
</html>
```

- * 括弧が不一致、もしくは抜けがある。
- * 波括弧が不一致、もしくは抜けがある。
- * 識別子のタイプミス。
- * 二重引用符や單一引用符の誤用。

オーエンの
バグチェック
シート。

ふーむ…



自分で考えてみよう



オーエンを助けましょう。ラジオ局の着呼スクリプトに潜伏していると思われる問題の種類をチェックしてください。

- | | |
|--|---|
| <input type="checkbox"/> 波括弧が不一致、もしくは抜けがある | <input type="checkbox"/> 括弧が不一致、もしくは抜けがある |
| <input type="checkbox"/> 識別子のタイプミス | <input type="checkbox"/> 二重引用符や單一引用符の誤用 |
| <input type="checkbox"/> まったく新しい種類の問題 | |



オーエンを助けましょう。ラジオ局の着呼スクリプトに潜伏していると思われる問題の種類をチェックしてください。

波括弧が不一致、もしくは抜けがある

括弧が不一致、もしくは抜けがある

識別子のタイプミス

まったく新しい種類の問題

二重引用符や単一引用符の誤用

checkWinner() の呼び出しが誤って小文字の w になっています。このタイプミスで JavaScript は checkwinner() をまったく別の関数と見なすので、関数が見つかりません。

checkWinner() の呼び出しだけを大文字の W に修正します。

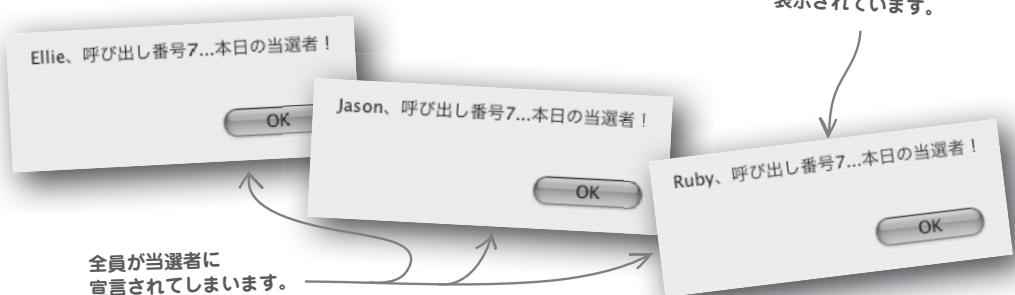
```
<input type="button" value="呼び出し"
      onclick="checkWinner(this.form, document.getElementById('caller').value, 7)" />
```



全員が勝利してしまう（バグその3）

タイプミスによって undefined になるバグはつぶしましたが、ラジオ局の着呼スクリプトはまだエラーが発生しています。ただし、プラウザが問題を報告する状態はなくなりました。今回のエラーは、名前を入力した全員が当選者になってしまっています。スクリプトでは正しく呼び出し番号は宣言されています。このままオーエンがバグを修正できないと、これまでの努力が無駄になってしまいます。

不思議なことに呼び出し番号が当選番号でないものも当選番号で表示されています。



アラートボックスを使ってデバッグ

当選者の番号を検査する箇所で、変数callNumとcheckWinnder()の引数winningNumを比較しています。このコードは問題なさそうですが、変数callNumの値がどうなっているか、もう少し詳しく調べる必要があります。

```
...  
if (callNum = winningNum) {  
...  
}
```

このコードを見る限り、明らかに間違いはなさそうです。

このコードの前後でcallNumの値が変わっていないか調べてみる価値はあります。

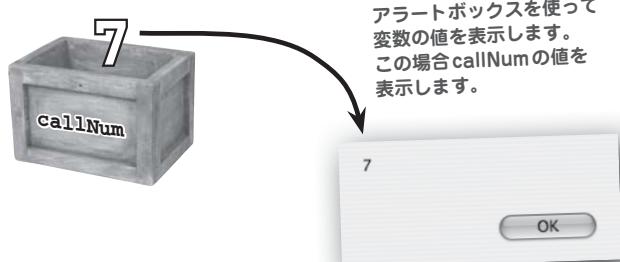


スクリプトの
あちこちにある変数の値を
見る方法ってあるかな?

アラートボックスは
変数の値をすぐに
見るので便利です。

アラートボックスをデバッグのウィンドウとして使う

アラートボックスはユーザーに情報をポップアップで表示するだけではありません。JavaScriptコードの開発中に一時的に変数の値を調べるときにも使えます。それだけでなく、コードのある部分が期待通りに呼ばれているか確認する目的にもアラートは使えます。このバグに関しては、変数callNumの値をアラートを使って調べてみる必要があります。



アラートを使って変数の値をウォッチする

ウォッチとはデバッグの用語で、プログラムの実行中に変数の値を調べることを意味します。アラートは簡単なウォッチの手段を提供します。変数の値が常に表示されつづけるわけではありませんが、それでも非常に便利です。変数の値を調べたい箇所であればJavaScriptのコードのどこでも使えるからです。

```
alert(callNum);
if (callNum = winningNum) {
  alert(caller + "、呼び出し番号" + callNum + "...本日の当選者！");
  form.submit();
}
```

何か不具合があります…callNumには現在の着呼番号が設定されていなければなりません。

NaN

OK

そうか、原因が
わかったぞ。

```
if (callNum = winningNum) {
  alert(callNum);
  alert(caller + "、呼び出し番号" + callNum + "...本日の当選者！");
  form.submit();
}
```

ifのテスト条件の直後で変数
callNumは7で表示されます。

7

OK



頭の体操

アラートを表示する1行コードで調べると不思議なことに
callNumは7に設定されています。バグの原因は何だと
思いますか？ オーエンは何をみつけたのでしょうか？



アラートでデバッグ

今週のインタビュー：アラートもバグにはうるさい

Head First：あなたに関して、さまざまな人物像が語られていると言わざるをえません。なにかとうるさい人ともデバッガの親友とも聞いています。どのアラートが本物なのか教えてください。

アラート：うるさい人というのは偏見ですね。いい奴ですよ。とても単純だし。何か情報をくれれば、それを表示しますし、音だってならします。それだけです。どこにも悪意なんてないですから。

HF：たぶん「ポップアップ」が問題なんだと思います。最近、あちこちにばかげた広告が表示されるので、ポップアップは嫌われているようです。

アラート：ああ、なるほどねえ。そこがうるさがられているわけですか。でもそれは、金槌の使い方を知らない馬鹿な大工がいたとして、金槌を責めてるようなもんですよ。言いたいこと、わかりますよね？

HF：つまり、あなたに関する悪い噂は、実はあなたの使われ方が間違っていた、というわけですね。

アラート：その通り。私は言われた通りのことをするだけです。ばかな広告を何度もポップアップで表示しろ、と言われたら、そうするだけですよ。それが好きでやってるなんて言ってませんし、私には選ぶことなんてできないんですから。そうだ、デバッガの世界への私の貢献について質問されていましたね。

HF：これは失礼しました。はい、JavaScriptプログラマがコードのバグを見つけ出すとき、あなたがたいへん力を貸してくれると聞いています。どのように協力されるのですか？

アラート：簡単なことです。たとえば、変数がおかしくなって、意味のない値が設定されているとしますね。そうなると、プログラマはイララして、コーヒーを飲みまくって、スクリプトのあちこちで変数の値がどう変わっているのか調べる方法がどうしても必要になるんですね。

HF：でも、どうやってスクリプトのあちこちで変数の値を表示することができるのですか？ 難しそうに思えますが。

アラート：いいえ、少しも。スクリプトのあちこちで、何回も私を呼び出せばいいだけですよ。

HF：わかりました。デバッgt;ルツールを助けるとき、何か問題になったことはありませんか？

アラート：そうですね、ループなどで難解もコード断片が繰り返されるとき、デバッgt;情報ポップアップするのは問題だったと思います。

HF：それはどうしてですか？

アラート：わたしはポップアップウィンドウなので、クリックしないと消えないのです。何回もポップアップすると、そのたびにクリックしないといけませんよね。

HF：そういう意味ですか。探しているデータがないときでも、あなたは役に立つとも聞いていますが。

アラート：はい。コードの断片がいつ呼ばれるのか、そもそも呼ばれたのかどうか、はっきりしないことがよくあります。コードの中で私を呼び出せば、実際にコードが呼ばれたことがわかります。コードが呼ばれたときだけアラートを表示することができますからね。

HF：こうしたデバッgt;のシナリオで、あくまでそのときだけ、あなたは表示してくれるわけですね。

HF：まったくその通りです。なんだか定職についていないみたいですが、気にしていません。ちょっとした公益事業みたいに、デバッgt;のとき側にいてお手伝いするだけです。

HF：バグ探しのときのあなたの役割を説明していただいて感謝しています。またお目にかかる日を楽しみにしています。

ありがとう。またね！

OK

論理エラーは合法ですがバグになります

オーエンは論理エラーを追求しました。このエラーはJavaScriptの文法に違反していませんが、やはり間違っているのです。この場合、`==`ではなく`=`が使われているので、`winningNum`の値と比較するのではなく、`callNum`に代入されてしまいます。ちょっとした間違いですが、これが大問題になるのです。

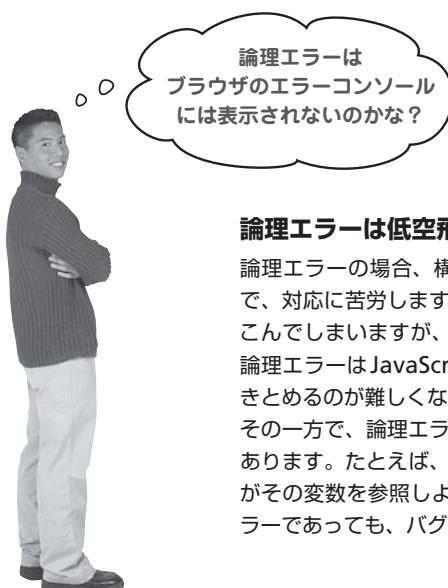
問題なさそうに見えていた
コードに、見つけにくい
些細なバグがあることが
わかりました。

```
...  
if (callNum == winningNum) {  
...  
...  
if (callNum == winningNum) {  
...
```



`=`を`==`に変更
してバグ
その3を粉碎。

このエラーのほんとうの問題は、ブラウザの動作をつまづかせて、構文エラーのようなエラーを表示させるわけではない点です。JavaScript インターブリタは、代入された値で代入文を評価するので、この場合`winningNum`の値で評価されます。この値は0でないので、`if`文のテスト条件は自動的に`true`に変換されます。つまり、このコードは期待される動作とは違った動作にもかかわらず文法的にはまったく合法なのです。



論理エラーは低空飛行のようなものでレーダーにかかりません。

論理エラーの場合、構文エラーのように間違ったコードが表示されるわけではないので、対応に苦労します。ブラウザでスクリプトエラーが表示されると、すこし気分がへこんでしまいますが、バグが検出されているわけなので、悪いことではないわけです。論理エラーはJavaScriptの文法に違反しているわけではないので、それだけ原因をつきとめるのが難しくなります。

その一方で、論理エラーはスクリプトの実行中にスクリプトエラーを発生させることができます。たとえば、論理エラーが原因で変数が初期化されなかった場合、スクリプトがその変数を参照しようとすると、`undefined`エラーになります。そのため、論理工
エラーであっても、バグ探しの苦しみが少ないときもあります。



重要ポイント

- 構文エラーはJavaScript言語の文法に違反したコードによって発生します。JavaScriptインターペリタによって捕捉されます。
- 文字列は注意深く二重引用符または単一引用符で囲む必要

があります。

- HTMLのイベントハンドラ属性の中にJavaScriptコードを含める場合、二重引用符と単一引用符を混在させます（ただし対になっていること）。

- アラートボックスはスクリプトの中で変数を監視するための単純ですが便利な手段です。
- テスト条件で実際には`==`のつもりが誤って`=`になるのはよくあるエラーです。

素朴な疑問に 答えます

Q: エスケープ文字は二重引用符と単一引用符をエスケープするために使うのですか？

A: JavaScriptではこの他にもエスケープ文字の用途があります。たとえば、\tを使って文字列にタブを挿入することができます。同様に、\nは改行を表します。文字通りのバックスラッシュは\\と書く必要があります。エスケープ文字が効果的に使われる場所のひとつが、アラートボックスに表示されるテキストです。アラートのテキストのインデントや改行を\tや\nを使って制御することができます。

Q: HTML属性でエスケープ文字を使うとき、どんな制約がありますか？

A: HTML属性はJavaScriptの文法の適用範囲外です。すくなくとも属性値にエスケープ文字を使うことはできません。JavaScript文字列の中で文字をエスケープして、それを囲んでHTML属性にすることはできますが、属性を囲む文字と同じ文字はエスケープできません。

まだ混乱しているようでしたら、こんな風に考えてください。HTMLは二重引用符または単一引用符で囲まれた中に現れる値を単純に属性とみな

します。どちらの引用符を使うにせよ、開始と終了の引用符は同じでなければなりません。HTMLでは属性の値がJavaScriptのエスケープ文字として処理されることはありません。エスケープ文字をHTML属性の中で使うには、属性を囲むのに使った文字



と衝突しないようにする必要があります。イベントハンドラ属性の場合、属性値がJavaScriptコードとして解釈されるからです。

Q: デバッグ作業を細かく制御できるJavaScriptのデバッガはありますか？

A: はい、いくつかあります。それらを一通り調べてみて、どれかひとつを試してみるのも悪くありません。この章で説明しているデバッグ手法と良いコーディングを組み合わせれ

ば、エラーのないスクリプトを作るのに役立つでしょう。

Q: JavaScriptコードで未定義の変数や関数を参照しようと、実際にはどうなるのでしょうか？

A: 未定義の変数とは、まだ作成されていない変数か、あるいは作成されているけれども値が設定されていない変数です。どちらの場合も変数に格納されている値は不明(未定義)です。その値を読もうとしたり、何か処理をしようとしても、まったく意味がないので、JavaScriptはエラーにしてしまいます。

関数呼び出しのときJavaScriptがその名前の関数を見つけられなかったとき、同じ状況になります。関数が未定義なので、この呼び出しは意味がなく、何も呼ばれません。実行するコードがないので、JavaScriptはエラーとみなします。

Q: if文のテスト条件の前で変数callNumがNaNになるのは何が原因でしょう？

A: わかりません。とはいえ、スクリプトコードで何か間違いがあることを示しているので、他にバグがないか調べるのは重要です。

特別座談会



今夜の対談：
構文エラーと論理エラーが下手なスクリプトへの
愛情について語ります

構文エラー：

あんた、正体不明の影の黒幕って噂だけど。オレも似たようなもんさ。下手くそなスクリプトをからかうのが楽しいんだよな。

そいつは違うね。オレは表に現れてるスクリプトの不具合が好きなんだ。そこは誰にも負けないぜ。あんたはあんたで、オレの力を借りるまでもなく、ブラウザに悲鳴をあげさせる自信があるみたいだけど。

あんたのひねくれた物の見方、買ってるぜ。でも、あんたがスクリプトの実行を妨げないのは問題だな。そこはオレと違うところだ。オレならすぐ実行を止めさせる。

残念ながらオレたちにはそんなに深刻な被害を与えるだけの力はないぜ。たしかに、ページをめちゃくちゃにして動かなくするのは楽しいけど、そうなって何もアクセスできなくなるのは困りもんだ。重要なデータがつまたハードディスクを扱えたら、どんなに楽しいかな。

論理エラー：

もちろんさ。表面的には問題ないように見えて、見えないところに変な問題を抱えているスクリプトほど面白いものはないね。

それで楽しいかい？ オレみたいに闇討ちする方が効果的だぜ。やつらをなだめて、ひとまず落ち着かせる。そして、あちこちにある些細な問題をゆっくりと明らかにさせるんだ。あんたがいくらいい仕事をしても、どうせやつらはブラウザがちゃんと動いているか疑うにきまってる。

いいところを突いたね。あんたみたいにスクリプトを止めて問題を明らかにしなかったのは恥じるよ。ブラウザ全体を煙に巻いて壊した方がよかったかもなあ。

そりやねえよ、オレたちがつけいる隙なんて。

構文エラー：

JavaScript インタープリタはオレたちを閉め出してるからなあ。

いいや、どんなトリックなんだい？

どうやって身を隠すんだ？

オレにも似たようなことがあるぜ。プログラマは JavaScript の文の最後にセミコロンを付け忘れることがある。セミコロンがなくても、インターフリタはひとつの文が複数行にまたがっているとみなし、次の行に続けて処理するからな。自信過剰なプログラマは、コードを「最適化」して、複数の文を 1 行にするんだな。そのときオレの出番というわけさ。ちっとも成長しやがらねえ。オレはいつも高笑いだぜ。

インターフリタが気づかないってところがいいよな。オレだったら、すぐさま駆けつけて、エラーを表示して驚かしてやるところだ。そうだ、オレたちはお互い手を組んだ方がよくないか。そうすりゃ、もっとやつらにダメージを与えられるぜ。

あんたの力になるぜ。

論理エラー：

まあ、楽しみはたくさんあるさ。`=` と `==` のささいなトリックについて話したっけ？

2 つの値を比較するために `==` とタイプしたつもりが、間違えて `=` とタイプしてしまうと、これは代入になるよな。プログラマが問題を見つけられずに何時間も困っているのを見るのは愉快なもんだぜ。コードそのものは技術的に間違っていながら、JavaScript インタープリタには問題がわからないわけさ。

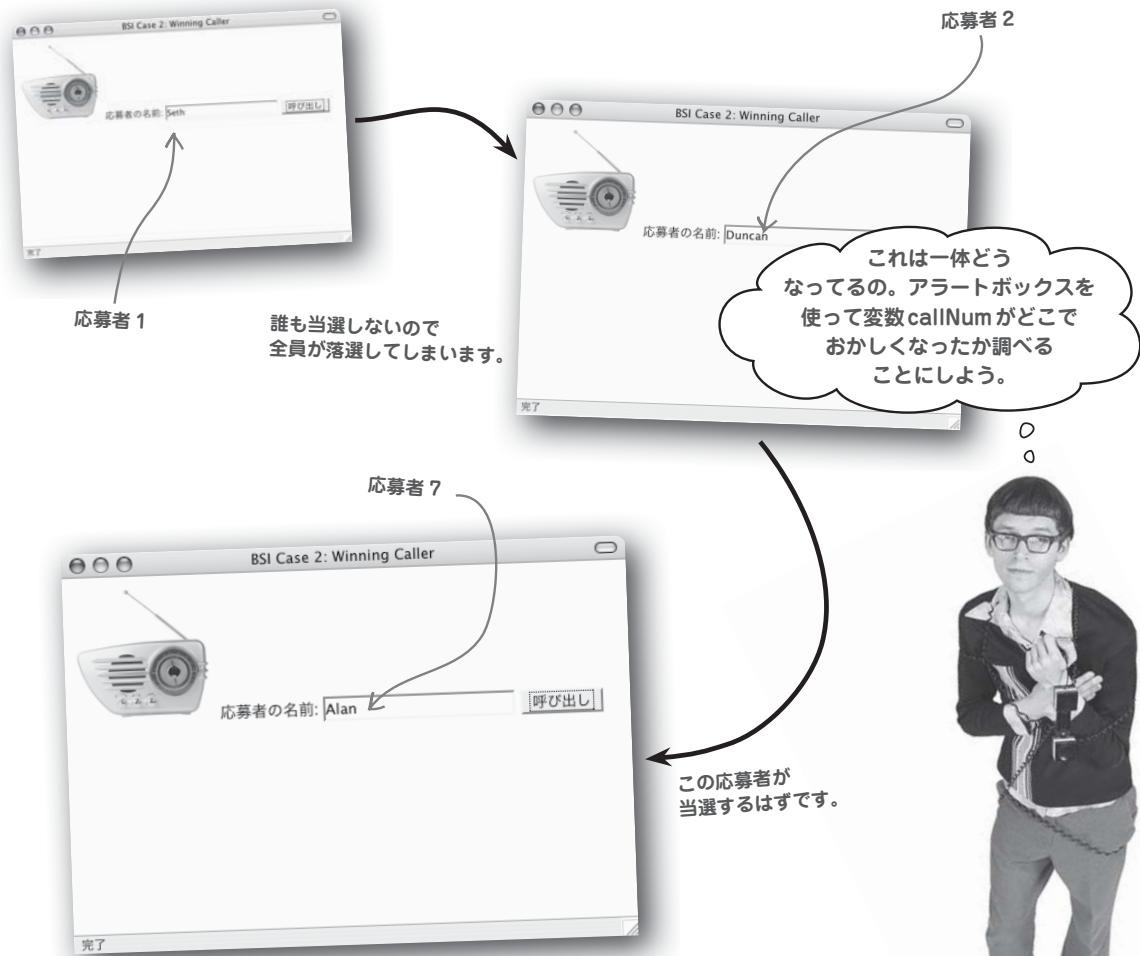
いや、ぜんぜん隠れたりしてないな。大手を振って歩いてるぜ。法に触れない範囲でトラブルを起こすわけだ。

その話を聞いて、ひとつ思い出した。プログラマが関数を書いた後で関数の引数を変更するのを、オレは楽しみにしてるんだ。新しい引数にあわせて関数の呼び出しを変更しないといけないので、変更漏れがあるんだな。一見うまくいっているように見えるし、インターフリタも問題に気がつかないけど、引数が違っているので予期せぬ結果になるわけだ。

そりゃいいや。さっそく始めるか。

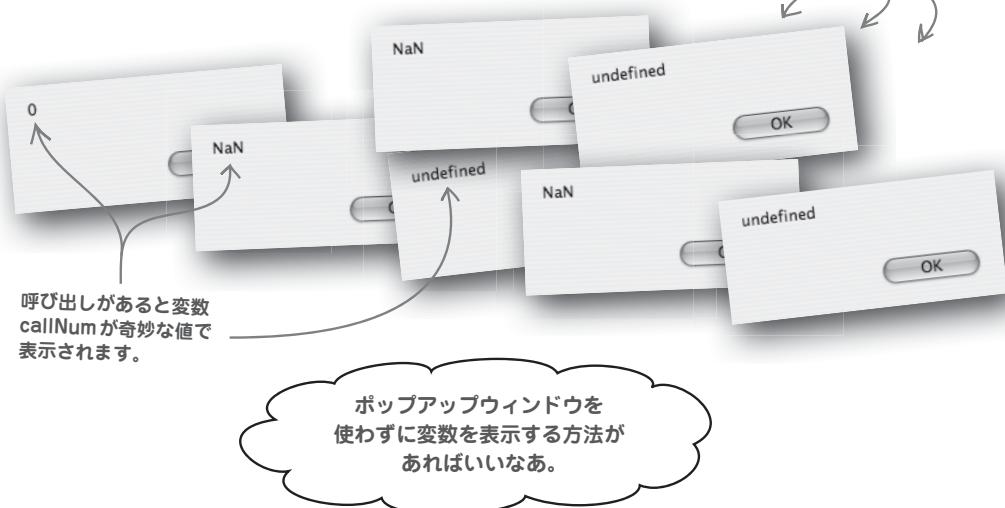
今度は誰も当選者になれない！（バグその4）

今回のデバッグは今までのように簡単には解決できないことにオーエンは気づいてきました。if文の論理エラーを修正したら、今度は誰も当選者になれなくなってしまったのです。全員が当選者になってしまふ状態から、誰も当選者になれない状態に変わってしまいました。オーエンが早く解決できないと、このバグは自尊心の強い人を怒らせてしまうでしょう。



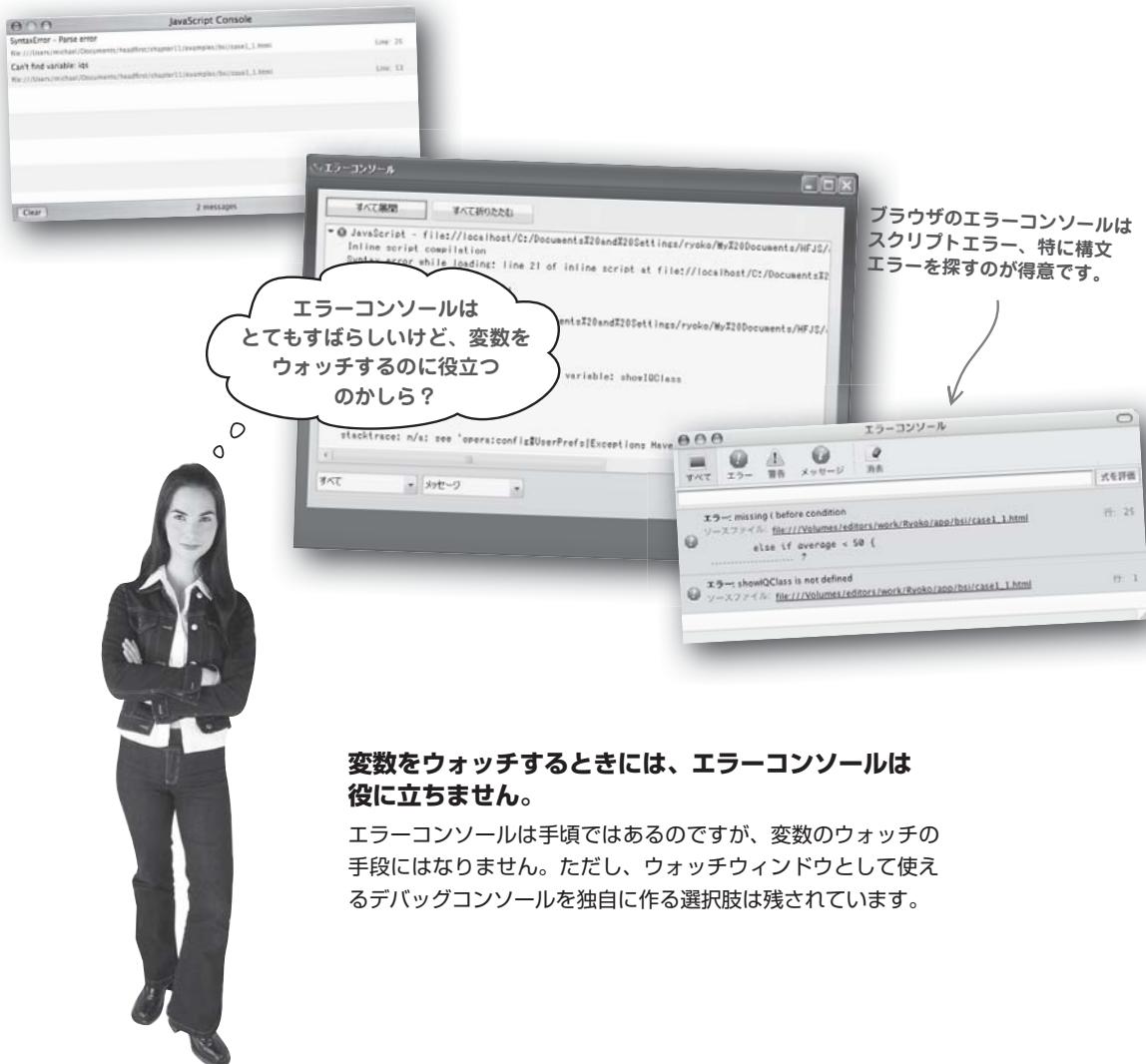
アラートの連発に打ちのめされる

オーエンはアラートを使って変数callNumをウォッチして、何が起きているのか調べようとした。しかし、7番目まで行く途中でうんざりするほど何度もアラートボックスが表示されます。コードの別の場所でもいくつかアラートを使ってみましたが、どれも役に立たないものばかりで、どこから手をつけていいか分からぬくらい打ちのめされています。



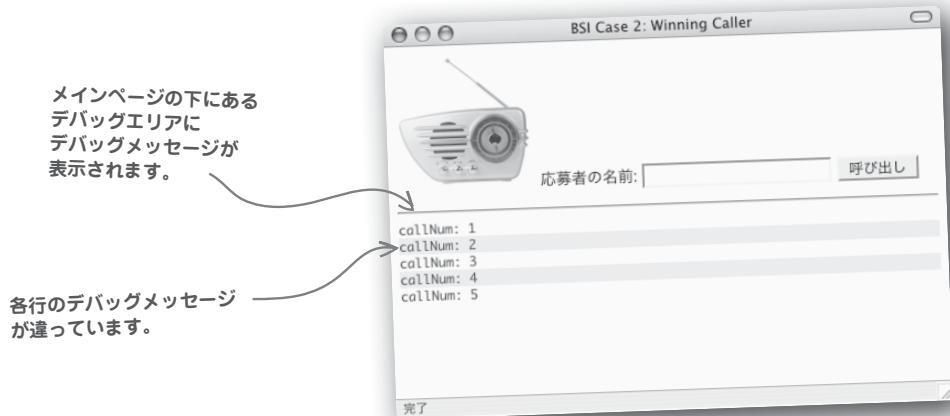
ブラウザのデバッグコンソールが役に立ちます

ほとんどのブラウザにデバッグコンソール、あるいはエラーコンソールがあり、スクリプトで発生したエラーを表示します。エラーコンソールはスクリプトに何か問題が発生したとき、その原因を調べるのに役立ちます。多くの場合、発生した問題を正確に診断するのに役立ちます。とりわけFirefoxのエラーコンソールは優秀です。



デバッグのためにカスタムコンソールを作る

カスタムデバッグコンソールを作るなんて言うと、思わず萎縮してしまいそうですが、必要なときにテキストを表示するだけですから難しくありません。アラートボックスではなくページに直接デバッグ情報を表示する必要があります。ユーザにOKをクリックさせる必要がなければ、別のポップアップウィンドウを使うこともできますが、ページに直接デバッグメッセージを表示する方が効率的です。

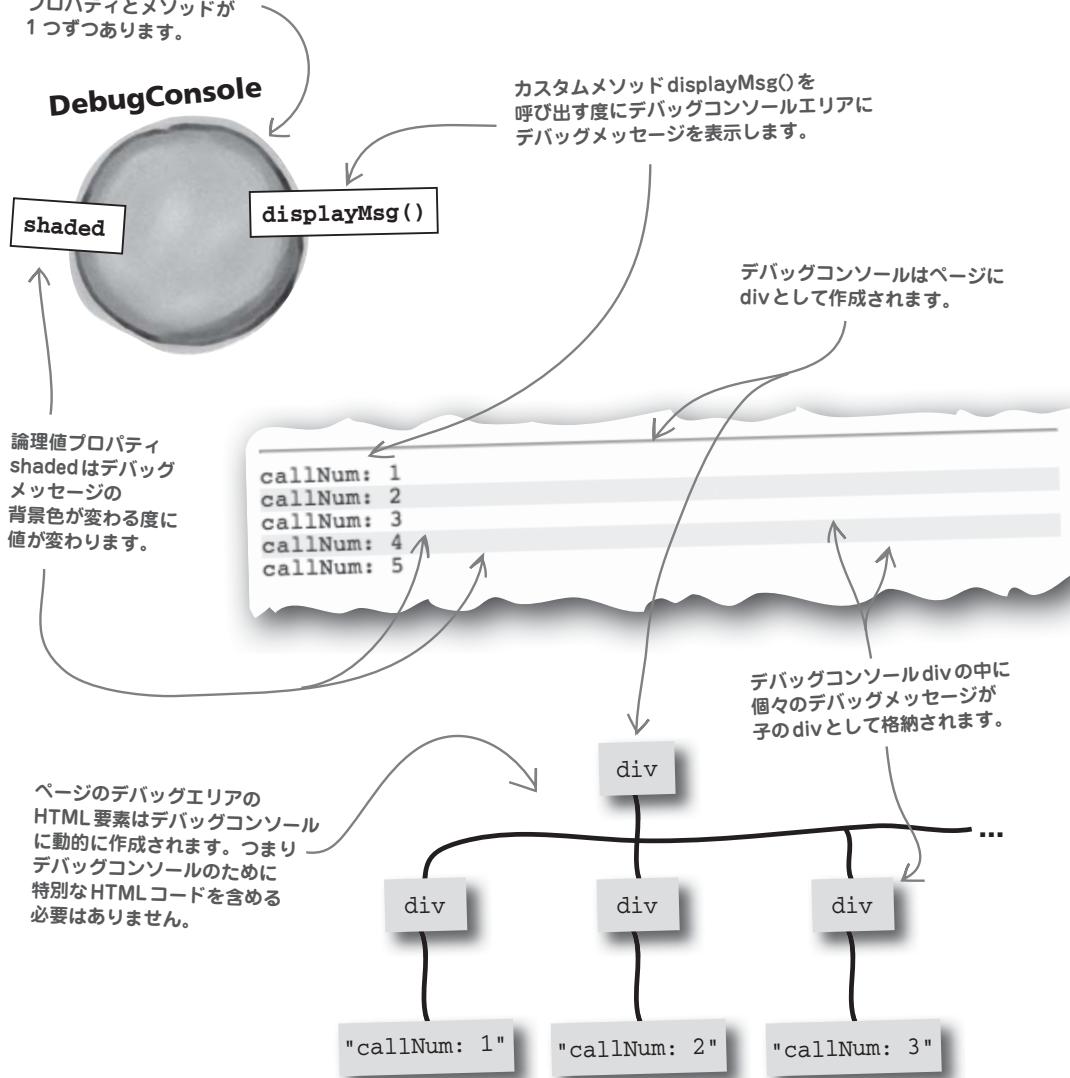


自分で考えてみよう

ページのある領域にデバッグメッセージのリストが動的に表示されるように設計されたJavaScriptのデバッグコンソールを想像してみてください。この設計に必要な構成要素（デバッグコンソールのためのJavaScriptオブジェクトを含む）を列挙し、これらを連携させる方法を考えてください。

自分で考えてみよう の答え

デバッグコンソールは DebugConsole オブジェクトとして設計されています。プロパティとメソッドが 1つずつあります。





JavaScriptマグネット

デバッグコンソールのためのコードの一部がなくなりました。空欄をマグネットで埋めて DebugConsoleオブジェクトを完成させてください。

```
function DebugConsole() {
  // デバッグコンソールエリアを作成

  var consoleElem = document. .... ( ..... );
  consoleElem.id = "debug";
  consoleElem.style.fontFamily = "monospace";
  consoleElem.style.color = "#333333";

  document.body. .... (consoleElem);

  consoleElem. .... (document. .... (hr));

  // 別の背景色プロパティを作成

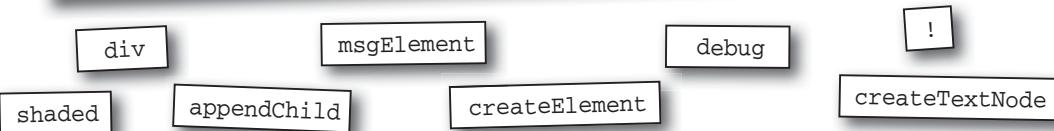
  this. .... = false;

}

DebugConsole.prototype.displayMsg = function(msg) {
  // メッセージを作成
  var msgElement = document.createElement("div");
  msgElement.appendChild(document. .... (msg));
  msgElement.style.backgroundColor = this.shaded ? "#EEEEEE" : "#FFFFFF";
  var consoleElem = document.getElementById( .... );
  consoleElem.appendChild( .... );

  // 別の背景色プロパティに置き換えます

  this.shaded = ..... this.shaded;
}
```





JavaScript マグネットの答え

デバッグコンソールのためのコードの一部がなくなりました。空欄をマグネットで埋めて DebugConsole オブジェクトを完成させてください。

```

function DebugConsole() {
    // デバッグコンソールエリアを作成
    var consoleElem = document. createElement( "div" );
    consoleElem.id = "debug";
    consoleElem.style.fontFamily = "monospace";
    consoleElem.style.color = "#333333";
    document.body.appendChild( consoleElem );
    consoleElem.appendChild( document.createElement( "hr" ) );
    // 別の背景色プロパティを作成
    this.shaded = false;      false で開始されるので
                                背景色は白で初期化されます。
}
DebugConsole.prototype.displayMsg = function(msg) {
    // メッセージを作成
    var msgElement = document.createElement("div");
    msgElement.appendChild( document.createTextNode( msg ) );
    msgElement.style.backgroundColor = this.shaded ? "#EEEEEE" : "#FFFFFF";
    var consoleElem = document.getElementById( "debug" );
    consoleElem.appendChild( msgElement );
    // 別の背景色プロパティに置き換えます
    this.shaded = !this.shaded;
}

```

デバッグコンソールdivはドキュメント本文に追加されます。つまりページの終わりに追加されます。

デバッグコンソールの最初の子ノードをhrにして、残りのページとコンソールメッセージの間に罫線を引きます

デバッグコンソールに子のdivとしてメッセージを追加します。

メッセージごとに背景色を入れ替えて読みやすくなります。

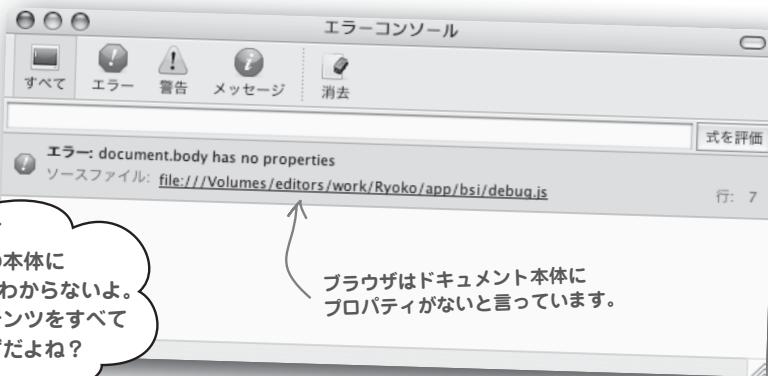
デバッガをデバッグする

オーエンは待ちきれずに新しいデバッグコンソールを追加して、ラジオ局の着呼スクリプトのどこに問題があるのか探しはじめました。ページに debug.js を取り込んで、ページのヘッダ部で DebugConsole オブジェクトを作成します。

```
<script type="text/javascript">
    // デバッグコンソールのグローバル変数
    var console = new DebugConsole();
    ...

```

このコードは DebugConsole オブジェクトをグローバル変数として ページのヘッダ部で作成します。



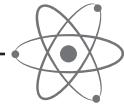
スクリプトに
まったく新しい
バグがある
ようです。

エラーになった行のコードは、文書の本文に子ノード (div) を追加しているだけで、これは問題にならないでしょう。

```
document.body.appendChild(console);
```

新しい DebugConsole オブジェクトを扱う処理ですが、何か他に見落としがあります。





頭の体操

このコードのどこに問題があってドキュメントの本体が空白になってしまうのでしょうか？

ページを待っている

デバッグコンソールの問題は、ページが読み込まれる方法とスクリプトコードがそのページの本文にアクセスするタイミングが関係しています。

ページのボディ部の前にヘッダ部が読み込まれるので、この時点ではボディ部のコンテンツはありません。

ページの実際のHTML要素はヘッダ部の後にあるボディ部が読み込まれるまで読み込まれません。

あ、なるほど。
ページのヘッダ部で実行されるスクリプトコードは、
ページのHTML要素にアクセスできないのか。



```
<html>
<head>
  <title>BSI Case 2: Winning Caller</title>
  <script type="text/javascript" src="debug.js"></script>
  <script type="text/javascript">
    // デバッグコンソールエリアを作成
    var console = new DebugConsole();
    // 呼び出しの総数
    var callNum = 0;

    function checkWinner(form, caller, winningNum) {
      // 呼び出し番号をインクリメント
      var callNum;
      ++callNum;

      // 当選者をチェック
      if (callNum == winningNum) {
        alert(caller + "、呼び出し番号 " + callNum + "... 本日の当選者！");
      } else {
        // 次の応募者のためにフィールドをリセット
        var callerField = document.getElementById(caller);
        callerField.value = "次の応募者";
        callerField.focus();
        callerField.select();
      }
    }
  </script>
</head>
<body>
  <form name="callform" action="radiocall.php" method="POST">
    
    Caller name: <input id="caller" name="caller" type="text" />
    <input type="button" value="呼び出し" onclick="checkWinner(this.form, document.getElementById('caller').value, 7)" />
  </form>
</body>
</html>
```

ページのヘッダ部で実行されるJavaScriptコードがウェブページのコンテンツにアクセスできていないのです。

ページのヘッダ部はページの本文より先に読み込まれるので、ページのヘッダ部で直接実行されるスクリプトコードは、ページの本文にあるHTML要素にアクセスしないように注意する必要があります。これは奇妙な制約に思われるかもしれませんのが、すべてのコードがページのヘッダ部で実行されるわけではないことを考えれば、この制約の意味がわかるでしょう。

でもヘッダ部にある
関数はどうなるの?
間違うことはないの?

ページのヘッダ部にあるコードがすべてページのヘッダ部で 実行されるわけではありません。

ページのヘッダ部に書かれた関数のコードは、ページのヘッダ部で実行されるコードと同じではありません。関数のコードは、その関数が呼ばれるまで実行されません。しかし関数の外にあるコードは、ヘッダ部が読み込まれたらすぐに実行されます。このコードが問題の原因になるのです。

DebugConsoleオブジェクトの場合、ページのヘッダ部で直接作成することはできません。なぜなら、このコンストラクタはページの本文にあるコンテンツに非常に依存しているからです。



自分で考えてみよう



ページの要素に安全にアクセスできるかどうかを確認するために
DebugConsoleオブジェクトを作成するとしたら、いつ、どこに
作成したらいいか書いてください。

.....
.....
.....
.....
.....



ページの要素に安全にアクセスできるかどうかを確認するために DebugConsoleオブジェクトを作成するとなったら、いつ、どこに作成したらいいか書いてください。

ブラウザはページの読み込みが完了したことを onload イベントを発生させて知らせます。onload イベントに応答して DebugConsole オブジェクトを作成するべきです。ただし、console はグローバル変数になるように、ページのヘッダ部で宣言します。onload イベントが発生してからオブジェクトを作成するコンストラクタを呼び出します。



DebugConsole
オブジェクトの
作成場所を移すと、
デバッグコンソールの
バグがなくなります。

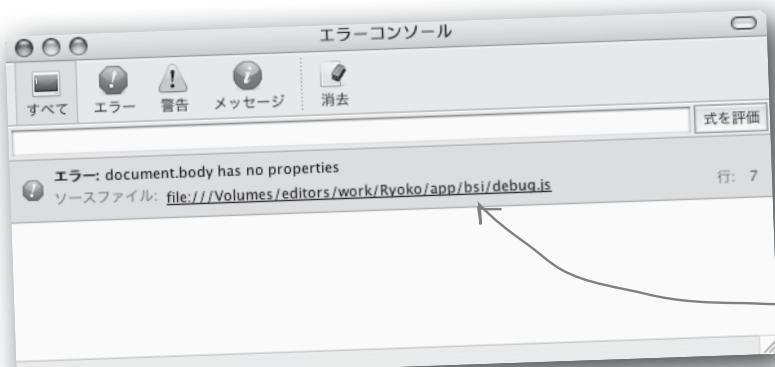
```
<body onload="console = new DebugConsole();">
```

DebugConsole オブジェクトは
onload イベントに応答して
作成されます。

実行時のしつこいエラー

文書の本文の読み込みが終わっていない問題は、**実行時エラー**の例になります。スクリプトが実際に実行されている間、ある条件のときに表示されるエラーです。実行時エラーは、たとえばユーザ入力がある特定の型になったときや、ループでの反復がある回数になったときなど、ある非常に特殊な条件のときだけに発生するエラーです。実行時エラーは、予測するのが難しいので、あらゆるエラーの中でも最も厄介なエラーです。実行時エラーは、それを再現するだけでも困難な場合が少なくありません。

実行時エラーは
スクリプトが実行中で
ある特定の条件の
ときだけ発生します。



デバッグコンソールの
バグは、データが
読み込まれる前にデータに
アクセスしようとして
発生する実行時エラーなので、
スクリプトが実行されないと
わかりません。

JavaScriptの3大バグ

実行時エラーとこれまでみてきた構文エラーと論理エラー、これら3つのエラーはJavaScriptの3大バグになります。これらのエラーは、どんなスクリプトでも発生する可能性があり、それらが同時に発生することもあります。これらのエラーを見つけ、それを取り除くには、それぞれのエラーの違いを理解することが重要です。



実行時エラー

実行時の条件で発生するエラーです。たとえば、ユーザがフォームにあるデータを入力したとき、まだ作成も初期化もされていないオブジェクトを処理しようとしたり、アクセスしようとしたとき、このエラーになります。



論理エラー

論理エラーはロジックの誤りが原因で発生します。ある処理を意図したコードが間違って意図と反する処理を実行しまうことが原因です。論理エラーのあるコードは、書かれた通りに処理を実行するので、プログラマが処理の内容を誤解したことが原因になります。



エクササイズ

if文のテスト条件で括弧が抜けている。

.....

カウンタ変数を0で初期化するのを忘れた。

.....

配列の最後の要素を越えてループしてしまう。

.....

関数の最後を波括弧で閉じるのを忘れた。

.....

構文エラー

JavaScript言語の文法に違反したのが原因で起きるエラーです。JavaScriptインタープリタでの実行に適さないコードがあることを意味します。

```
<html>
<head>
<title>BSI Case 2: Winning Caller</title>
<script type="text/javascript" src="debug.js"></script>
<script type="text/javascript">
// デバッグソースエリアを作成
var console = new DebugConsole();

//呼び出しの総数
var callNum = 0;

function checkWinner(form, caller, winningNum) {
    //呼び出し番号をインクリメント
    var callNum;
    ++callNum;

    console.displayMsg("callNum: " + callNum);

    //当選者をチェック
    if (callNum == winningNum) {
        alert(caller + "、呼び出し番号 " + callNum + "...本日の当選者！");
        form.submit();
    } else {

```

以下のエラーの記述に対応するエラーのタイプを書いてください。



エクササイズ

以下のエラーの記述に対応するエラーのタイプを書いてください。

- if文のテスト条件で括弧が抜けている。
- カウンタ変数を0で初期化するのを忘れた。
- 配列の最後の要素を越えてループしてしまう。
- 関数の最後を波括弧で閉じるのを忘れた。

構文

論理

実行時

構文

callNumを表示したら
「数字でない」になるって、
奇妙だなあ。

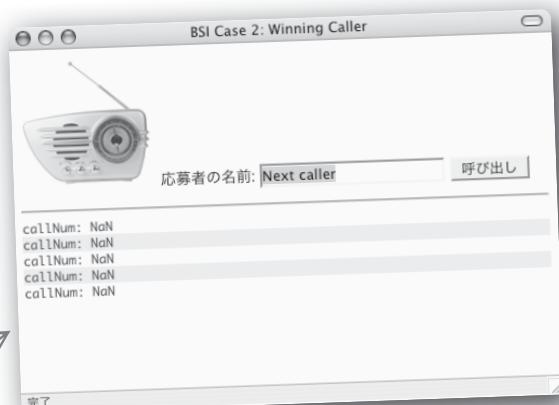
これ、数値じゃない

変数callNumの値を調べることができます。その結果、オーエンが無視していた問題がぶり返していることがわかりました。変数callNumはNaNで表示されているので、これは数値ではないことを意味します。なぜ数値でないのでしょうか。

console.displayMsg("callNum: " + callNum);

1行コードで変数
callNumのウォッチを
設定します。

とりあえず
デバッグコンソールは
動作しています。



ウォッチだけでは十分でないときもある

変数をウォッチすると、それが答ではなく謎になることもあります。なぜcallNumは数値でないのでしょうか？ どうして値が加算されないのでしょう？ このデバッグコンソールは、すでにわかっていることを表示するだけです。何が原因でそうなったのかを調べるには、どうしたらいいでしょうか？

いまどうなってる？

callNumの値が
変わるものまでコードを
取り除いてみようかな。



バグを探し出すときは、スクリプトからコードを取り除いてみるやり方もあります。

JavaScriptのデバッグでは、減らすことで得することもあります。この場合、コードを取り除いてみて、どんな変化が起きるか見てみるのはいい考えです。ただしコードを削除してしまうのは、いただけません。デバッグが終わったときにもとに戻すことがほとんどだからです。そこで、コードを削除するのではなく、一時的に無効にする方法が必要になります。

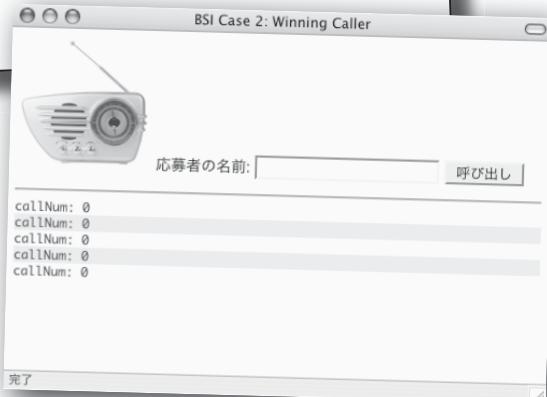
コメントを使って一時的にコードを無効にする

デバッグの間コードを無効にするには、実行可能なコードをコメントの中に隠すのが、もっとも簡単なやり方になります。コードそのものを削除することなく、特定のコードを選択してスクリプトの実行から外すことができます。バグを特定する必要があるときだけ、コードの行やある塊をコメントにするわけです。

```
function checkWinner(form, caller, winningNum) {  
    console.displayMsg("callNum: " + callNum);  
  
    /*  
     * 呼び出し番号をインクリメント  
     */  
    var callNum;  
    ++callNum;  
  
    // 当選者をチェック  
    if (callNum == winningNum) {  
        alert(caller + "、呼び出し番号 " + callNum + "... 本日の当選者！");  
        form.submit();  
    }  
    else {  
        // 次の応募者のためにフィールドをリセット  
        var callerField = document.getElementById('caller');  
        callerField.value = "次の応募者";  
        callerField.focus();  
        callerField.select();  
    }  
}
```

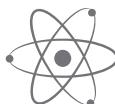
複数行コメントで関数の中にある
デバッグメッセージを表示する
以外のコードを無効にします。

callNumは0なので
無効にしたコードに中に
問題があるはずです。



コードを一時的に
無効にするには
コメントが便利です。

callNumの表示がやっと
0になったよ。無効にしたコードの
どこかで「数字でない」になってるんだな。



頭の体操

callNumをインクリメントしているコードの行だけもとに戻したらどうなると思いますか？

問題が多少は解決されました

1行ずつコメントにすれば、無効にするコードを選択しやすくなります。変数callNumを加算するコードをもとに戻すと、変数callNumは本来の動作に戻ります。このことから、無効にされている残りのコードで問題が発生していることがわかります。

```

function checkWinner(form, caller, winningNum) {
    console.displayMsg("callNum: " + callNum);

    //呼び出し番号をインクリメント
    // var callNum;
    ++callNum; ← インクリメントしているコードの行のコメントを外してcallNumの動作を調べます。

    //当選者をチェック
    // if (callNum == winningNum) {
    //     alert(caller + "、呼び出し番号 " + callNum + "... 本日の当選者！");
    // }
}

// フォームをリセット
document.getElementById('caller');
= "次の当選者";
;
();
();

callNum: 1
callNum: 2
callNum: 3
callNum: 4
callNum: 5 ← callNumは呼び出しごとに期待通りにインクリメントされています。

```

完了



自分で考えてみよう

デバッグコンソールでのcallNumのバグの原因とその修正方法を書いてください。

.....
.....
.....
.....
.....
.....

自分で考えてみよう の答え

デバッグコンソールでのcallNumのバグの原因とその修正方法を書いてください。



ローカル変数を作成していたコードの行を削除したらバグその4が消えました。

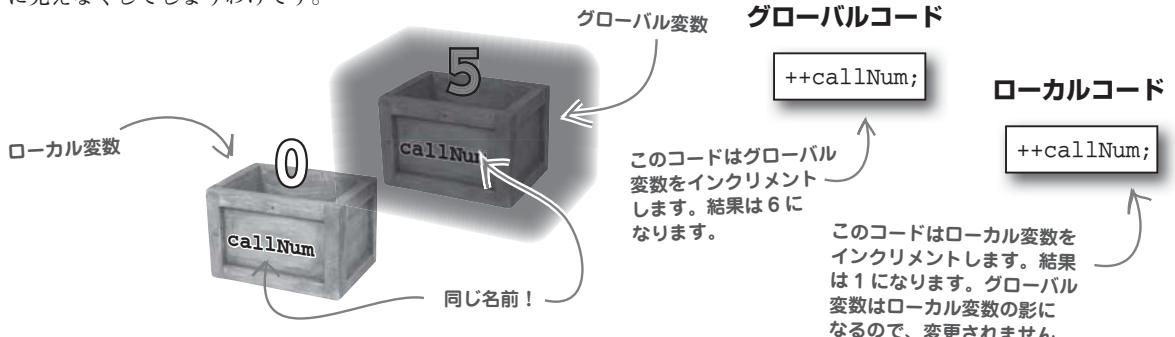
```
//呼び出し番号をインクリメント
var callNum;
++callNum;
```

ローカル変数callNumを誤って作成している行を削除すれば、当初の期待通りグローバル変数callNumがインクリメントされます。

影の変数は危険

ラジオ局の着呼スクリプトでのcallNumのバグは、影の変数の事例にあたります。これはある変数が同じ名前の別の変数によって誤って隠れてしまう現象です。この問題は、グローバル変数と同じ名前のローカル変数が作成されたときに発生します。JavaScriptがローカル変数を作成すると、コードのローカルブロックの中で同じ名前のグローバル変数より優先されます。ローカル変数に対して実行された処理は、同じ名前のグローバル変数には影響しません。ローカル変数はグローバル変数を影で覆い、スクリプトから一時的に見えなくしてしまうわけです。

影の変数が発生するのはローカル変数とグローバル変数が同じ名前で作成されたときです。これは問題です。



素朴な疑問に答えます

Q: バグを見つけるためにコードをコメントにすると
き、どれだけコードを無効にすればいいか判断する
方法はありますか？

A: 個人的の意見ですが、JavaScriptのデバッグの経験を
つんでいけば、わかるようになるでしょう。問題のある箇所のまわりをすべてコメントにしないまでも、無効にした
コードが多すぎて失敗するのも、経験のひとつです。ほんとうに厄介な問題が生じたときは、ページのスクリプトコードを
すべて無効にするのをためらうことはありません。ページに
外部コードを取り込むタグを一時的に消去するのを忘れない
でください。スクリプトコードの特定の部分にバグがあること
がわかっているときは、別のアプローチがとれます。バグ
がなくなるまで、一度に1行だけ無効にするのです。一度に
まとめて無効にして、バグが現れるまでゆっくりと1行ずつ

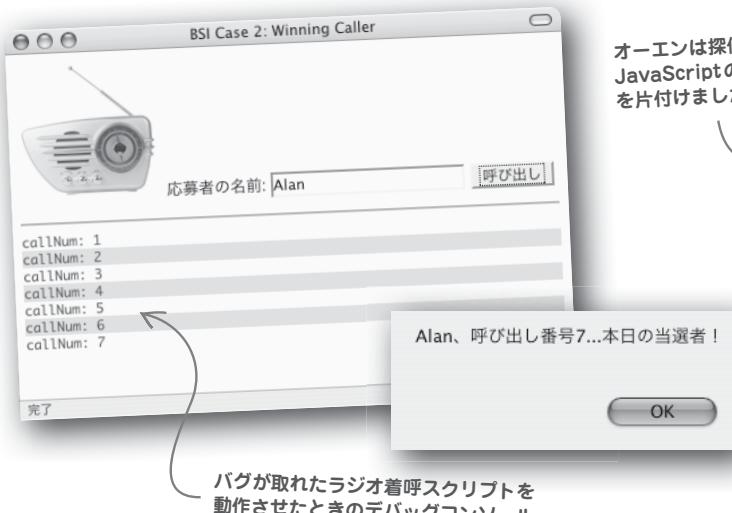
有効にしていくのではなく、バグが消えるまでゆっくりと1行
ずつ無効にしていくわけです。前者のアプローチは、どこに
バグがあるか、その手掛りがないときに適しています。後者のアプローチは、ある程度バグの場所が特定できているとき
に適しています。

Q: 影の変数を作成しようかと思っているのですが、
大丈夫でしょうか？

A: その質問は、自分の足を折ってもいいですか、とき
いているようなもんですよ。答はノーです。故意に
自分を傷つけて苦しむのはよくありません。それに、完璧に
動作させる目的でコードのデバッグを行うときには苦勞がつきものなので、故意に危険な行為を選択すべきではありません。影の変数はJavaScriptのコードに見つけにくい混乱をもたらすだけなので、どんな状況であれ避けるべきです。

一件落着！

新しいデバッグ手法を使って辛抱強い調査を続けた結果、
オーエンはこの件を解決しました。BSIのJavaScript探偵に
なる日も遠くありません。



オーエンは探偵のように
JavaScriptのデバッグ
を片付けました。



● オーエンのバグ退治チェックリスト

括弧が対になっているか確認する。

 コードのブロックを囲む波括弧が対になっているか確認する。コードをインデントしておくと波括弧の対応が調べやすくなるので便利。

識別子のタイプミスがないか徹底的に調べる。変数名と関数名は名前が一貫していないと大きな問題になる。

二重引用符と単一引用符の使い方が一貫しているか調べる。HTML属性の中でこの2つを必要に応じて注意深く混在させること。

 文字列の中で特別な意味をもつ文字にはエスケープ文字を使う。たとえば二重引用符 (`") や単一引用符 (`') など。

絶対に == のつもりで = を使わないこと。JavaScriptではこれをエラーと見なさないけれどもコードは意図と違った動作になってしまふ。

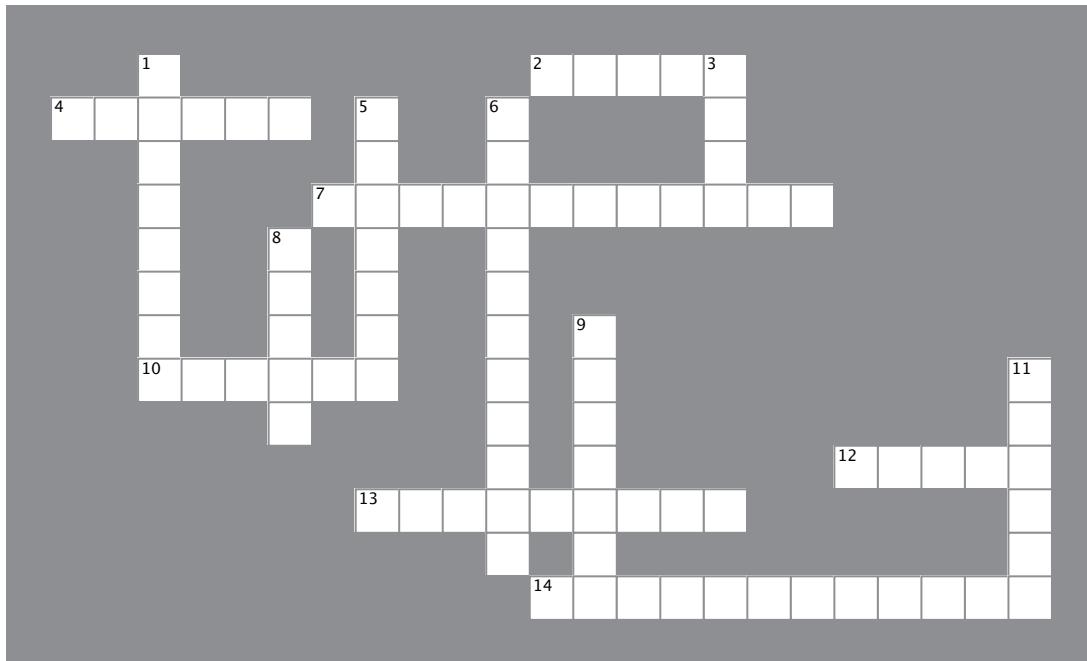
 オブジェクトにアクセスする前にオブジェクトを作成しておく。とくにウェブページの要素の場合、 onload イベントが発生しないと作成されないので、注意すること。

ローカル変数名とグローバル変数名を同じにしないこと。同じになるとローカル変数がグローバル変数を隠してしまい、予期できない動作になてしまう。



JavaScriptクロスワード

新たに見つけたバグの箇所を蟻の巣にする前に、このパズルに挑戦してみましょう。



ヨコのカギ

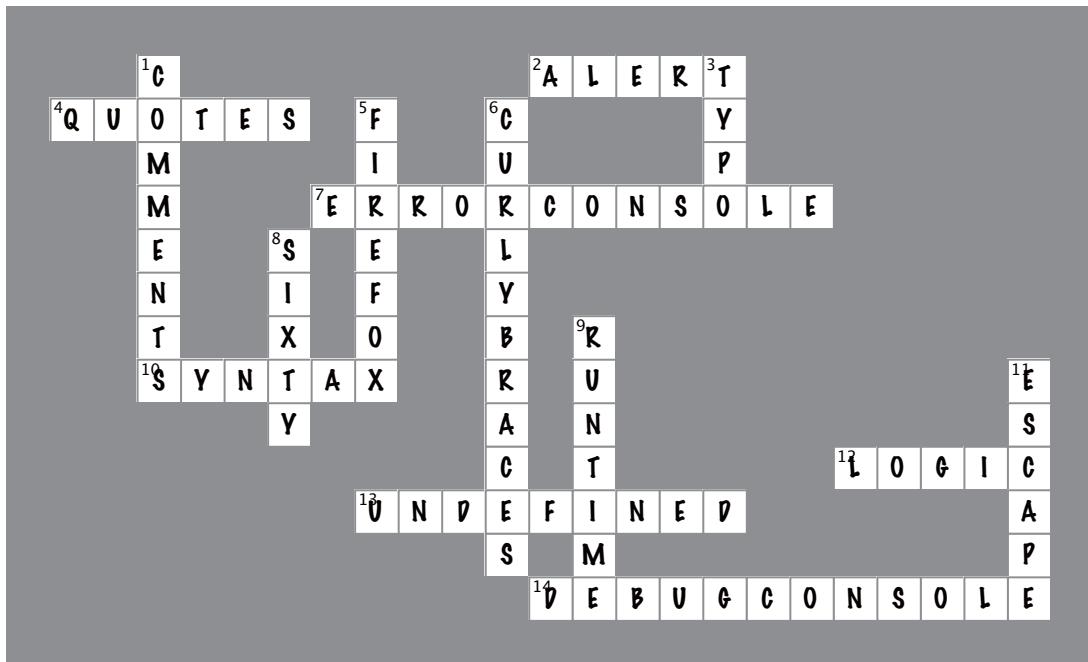
2. 変数の値をすぐに調べたいときこれを使います。
4. HTML属性にJavaScriptの文字列を入れると、單一引用符とこれを混在させて使います。
7. エラー表示に使う特別なウインドウブラウザ。
10. JavaScript言語の文法に違反したエラー。
12. JavaScriptの文法に完全に従っているけれども間違った結果になるエラー。
13. 値が代入されていない変数は.....です。
14. オーエンがバグ退治のために作成したカスタムオブジェクト。

タテのカギ

1. 一時的にコードを無効にするときにこれを使います。
3. 変数名にこれがあると問題になります。
5. JavaScriptのデバッグに最適なウェブブラウザ。
6. コードのブロックにこれがないと問題になります。
8. チョコレートバーで許容される虫の数。
9. スクリプトの実行中にだけ現れるエラー。
11. 文字列の中に特別な文字を入れるときこれを使います。



JavaScript クロスワードの答え

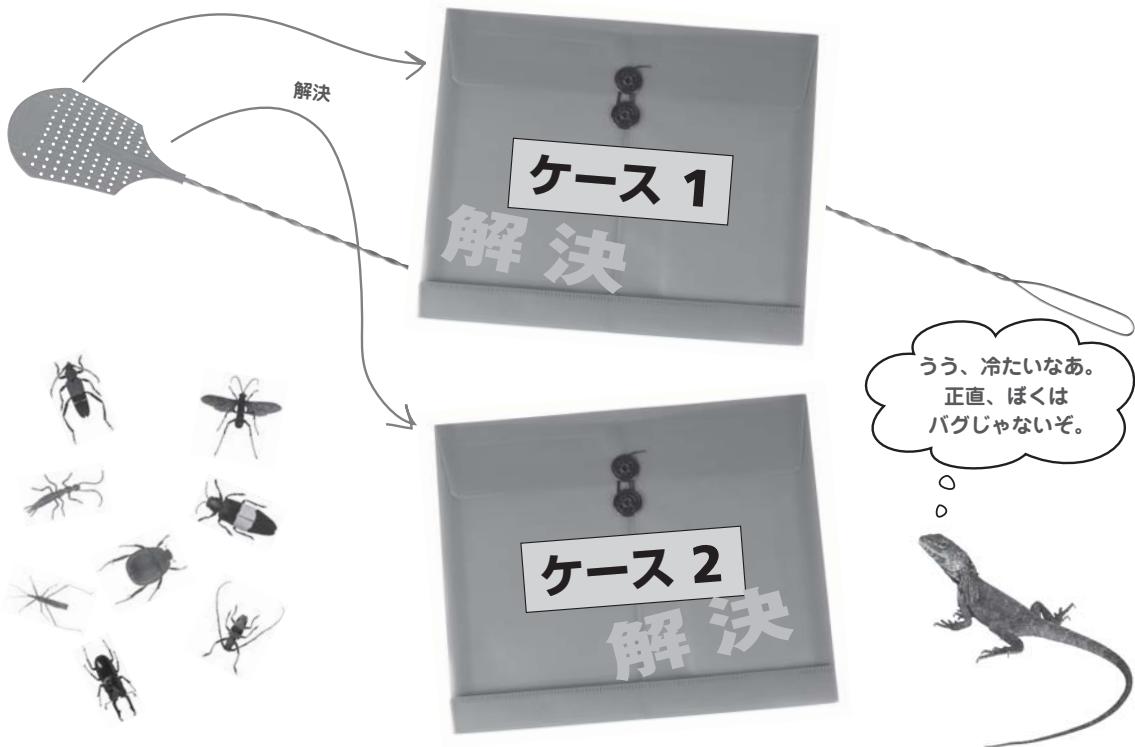


折り畳みページ

脳のところで
折り畳んでから
謎を解決しましょう。

JavaScriptのバグとどう
付き合えばいいんでしょう?

左脳と右脳がご対面!



他人にチェックしてもらうのは間違ったやり方です。
ますますバグに悩まされることになるかもしれません。
しつこいバグはあなたのコードを
衰弱させるので問題です。

12章 ダイナミックなデータ

感度良好な ウェブアプリケーション

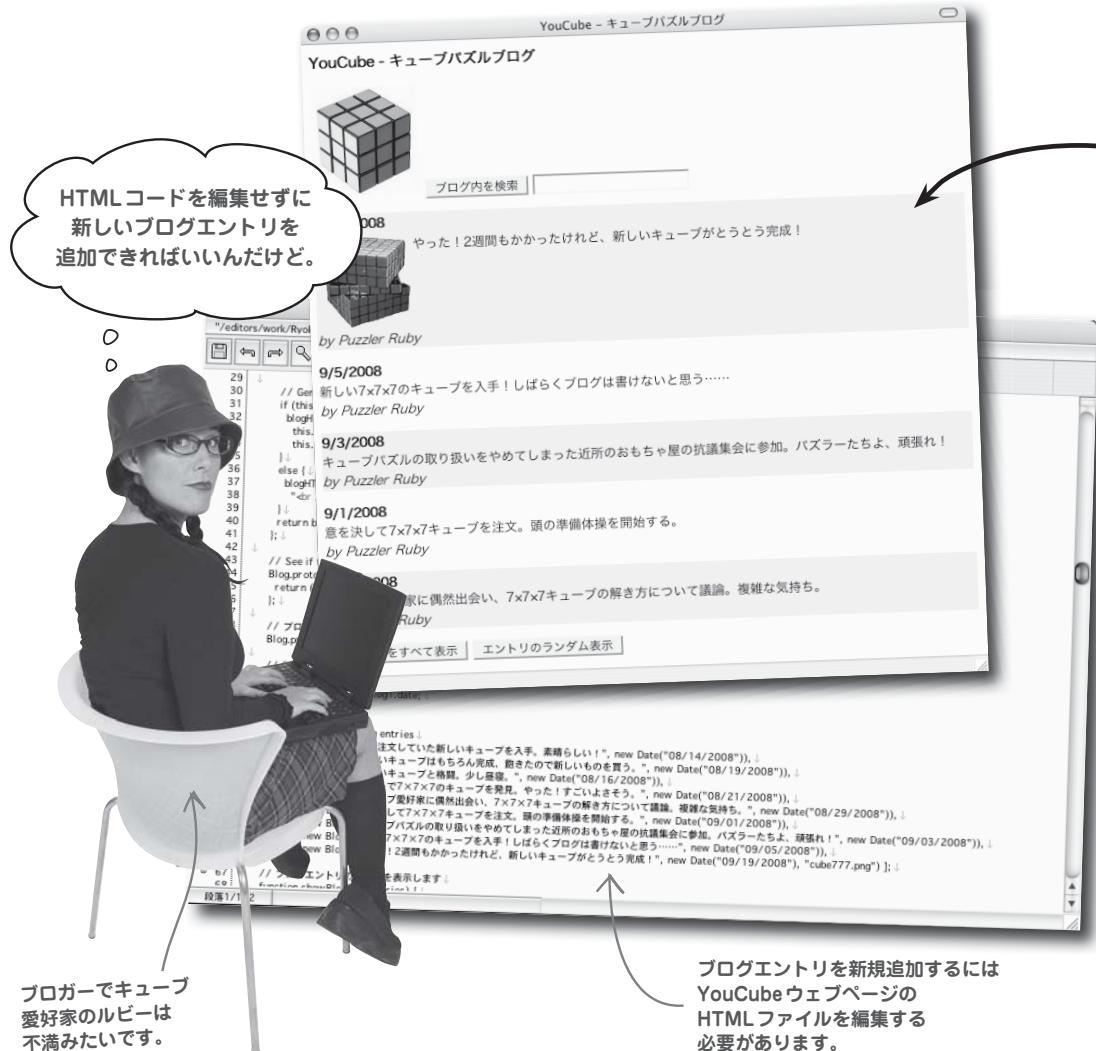


現代のウェブではユーザのあらゆる思いつきに反応できるページが求められています。それは多くのウェブユーザと開発者の夢と言えるでしょう。JavaScriptはこの夢を実現する重要な役割を担っています。Ajaxと呼ばれるプログラミング手法は、ウェブページの「感触」を劇的に変える機能を実現します。Ajaxを使うと、ウェブページの更新やブラウザの裏技を使わずに、リアルタイムにユーザの操作に応答しながら、データの読み込みと保存が動的にすばやく実行できる、本格的なアプリケーションになります。

動的なデータを実現したい

キューブパズルの熱烈な愛好家でブロガーのルビーを憶えてますか？ルビーはJavaScript対応のYouCubeブログを気に入っていますが、新しいエントリを追加するたびにページ全体のHTMLファイルを編集する必要があるので、イライラしています。ブログページを記述するHTMLコードからブログエントリをなんとか分離できれば、ブログのコンテンツそのものに集中できるのですが。

YouCube ウェブページの
ブログエントリを追加するとき
ルビーは HTML ファイルを
編集する必要があります。



データ駆動の YouCube

ルビーはあることに気づきました。彼女のブログでブログコンテンツをウェブページの構造から分離するには、ブラウザがページを処理するときにページに動的にデータが流し込まれる、そうした動的データが必要です。動的データで作られるウェブページはデータ駆動のページとして知られています。ページにはデータを流し込む構造が定義されているだけだからです。言い換えると、データはページのコンテンツを担当するわけです。

データ駆動の YouCube のファイルは
<http://www.headfirstlabs.com/books/hfjs/>
 から入手できます。

ブログデータは、ウェブページに
 触らずに編集できるように、
 別ファイルに格納されます。

ブログデータ

```

<blog>
  <title>YouCube - The blog for Cube Puzzlers</title>
  <author>Puzzler Ruby</author>
  <entries>
    <entry id="1">
      <date>2010/01/20</date>
      <content>I'm back! I'm back! I'm back! It's a real pearl.</body>
    </entry>
    <entry id="2">
      <date>2010/01/21</date>
      <content>I've solved the new cube but of course, now I'm bored and shopping for a new one.</body>
    </entry>
    <entry id="3">
      <date>2010/01/22</date>
      <content>Decided to get a headache tailing over the new cube. Gotta nap.</body>
    </entry>
    <entry id="4">
      <date>2010/01/23</date>
      <content>I've solved a 7x7 cube for online. Yikes! That one could be a banger.</body>
    </entry>
    <entry id="5">
      <date>2010/01/24</date>
      <content>I've solved a 7x7 cube again falling cubers to discuss the prospect of a 7x7x7 cube. Mixed feelings.</body>
    </entry>
    <entry id="6">
      <date>2010/01/25</date>
      <content>I've solved a 7x7x7 cube and ordered the scary 7x7x7 cube. Starting a mental exercise regimen to prepare.</body>
    </entry>
    <entry id="7">
      <date>2010/01/26</date>
      <content>I've solved a 7x7x7 cube and ordered a really outside of a local toy store that stopped carrying cube puzzles. Power to the puzzlers!</body>
    </entry>
    <entry id="8">
      <date>2010/01/27</date>
      <content>I've solved a 7x7x7 cube. Could be my last blind fold for a while...</body>
    </entry>
    <entry id="9">
      <date>2010/01/28</date>
      <content>I've solved a 7x7x7 cube again! I've solved a 7x7x7 cube but the new cube is finally activated!</body>
    </entry>
    <entry id="10">
      <date>2010/01/29</date>
      <content>I've solved a 7x7x7 cube again last night & huge cube was chasing me, and it kept yelling my name tauntingly... That's it!</body>
    </entry>
  </entries>
</blog>
```

ウェブページ

```

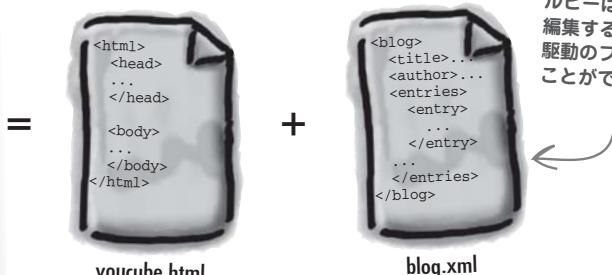
<html>
  <head>
    <title>YouCube - The blog for Cube Puzzlers</title>
    <script type="text/javascript" src="style.js"></script>
    <script type="text/javascript" src="youcube.js"></script>
    <script type="text/javascript" src="youcube.xml" as="xml"></script>
  </head>
  <body>
    <h1>YouCube</h1>
    <p>This is the blog for Cube Puzzlers</p>
    <ul>
      <li>Recent Posts</li>
      <li>Search</li>
      <li>Archives</li>
      <li>Categories</li>
      <li>RSS</li>
    </ul>
    <div id="post">
      <h2>Recent Posts</h2>
      <ul>
        <li>2010/01/27 I've solved a 7x7x7 cube. Could be my last blind fold for a while...</li>
        <li>2010/01/26 I've solved a 7x7x7 cube and ordered a really outside of a local toy store that stopped carrying cube puzzles. Power to the puzzlers!</li>
        <li>2010/01/25 I've solved a 7x7x7 cube and ordered the scary 7x7x7 cube. Starting a mental exercise regimen to prepare.</li>
        <li>2010/01/24 I've solved a 7x7 cube for online. Yikes! That one could be a banger.</li>
        <li>2010/01/23 I've solved a 7x7 cube again falling cubers to discuss the prospect of a 7x7x7 cube. Mixed feelings.</li>
        <li>2010/01/22 I've solved the new cube but of course, now I'm bored and shopping for a new one.</li>
        <li>2010/01/21 I'm back! I'm back! I'm back! It's a real pearl.</li>
        <li>2010/01/20 I'm back! I'm back! I'm back!</li>
      </ul>
    </div>
    <div id="search">
      <input type="text" value="Search the blog" disabled="disabled" onfocus="searchLog()"/>
    </div>
    <div id="archive">
      <input type="button" value="Archives" onclick="archiveLog()"/>
    </div>
    <div id="category">
      <input type="button" value="Categories" onclick="categoryLog()"/>
    </div>
    <div id="rss">
      <input type="button" value="RSS" onclick="rssLog()"/>
    </div>
  </body>
</html>
```

ウェブページはウェブページの構造に対応するHTMLコードとブログデータをページに動的に組み込むJavaScriptコードで構成されます。

プロジェクトリスナー
別ファイルから
ブログページに
取り込まれます。

JavaScriptはブログデータを
処理してHTMLウェブページに
取り込む責任があります。

JavaScriptを使うと、生のブログデータを動的にHTMLコードに流し込むことで、元のページとまったく同じ YouCube のページを出力できます。このデータ駆動ページはそれぞれ別々の部品で組み立てられます。その部品とは、構造だけのページとブログデータです。ブログデータを別ファイルに分離すれば、ウェブページのHTML、CSS、JavaScript コードから切り離されたブログコンテンツを自由に操作することができます



ルビーはこのファイルを
編集するだけでデータ
駆動のブログを更新する
ことができます。



動的データってとても
複雑そうね。厄介な JavaScript
コードをたくさん書くことに
なるのかしら？

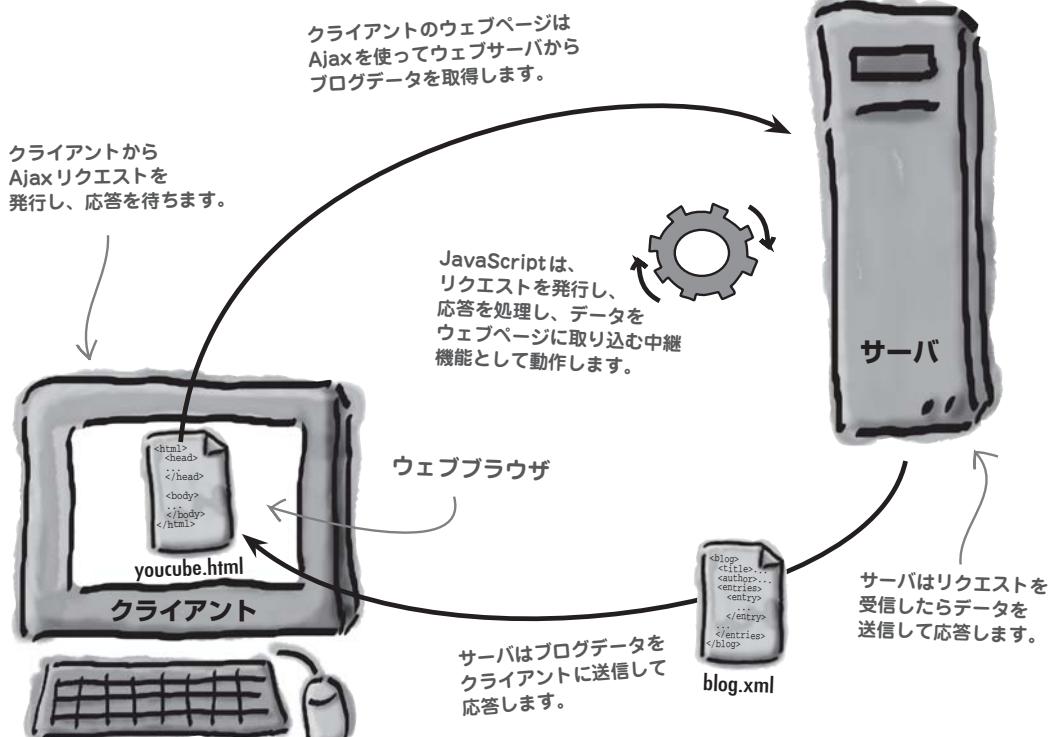
動的データを実現するには、フロント部のコーディングに
労力を要しますが、それだけの見返りがバックエンド部に
もたらされます。

動的データで駆動されるページにはフロント部の計画と実装に労力を
要しますが、長い目でみれば、ページの更新がすばやく簡単にできるようになるメリットがあります。Ajaxと呼ばれる巧みなプログラミング手法のおかげで、JavaScriptには動的データをサポートする機能が組み込まれています。

Ajaxとは要するに通信のこと

Ajaxは、クライアントであるウェブブラウザとウェブサーバの間で小さな「通信」を実行することで、動的データを実現します。スクリプトはサーバに対して、たとえばブログエントリの集まりのようないくつかのデータを要求し、サーバはこれに応答してデータを配信します。スクリプトはブログデータを受け取り、これをページに動的に組み込みます。

Ajaxを使うとウェブサーバから動的にデータを受信する
ウェブページを作ることができます。



頭の体操

ブログデータの文脈では「XML」は何を意味するでしょう？
どうしたら動的データの役に立つと思いますか？

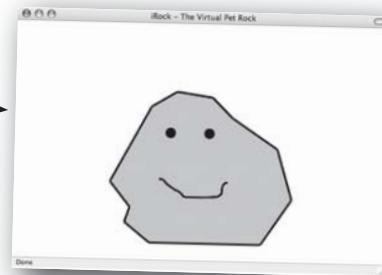
HTMLの枠を越えたのが XML

HTMLの「ML」はマークアップ言語の略です。HTMLはハイパーテキスト（「HT」）を作成するためのタグと属性のことです。HTMLはウェブページのハイパーテキストの作成に使われますが、XMLというマークアップ言語はあらゆるものを作成するために使われる。「X」とはつまり、何でもという意味です。あらゆる種類のデータをタグと属性で格納すれば、いいことあるんじゃないか、という発想です。マークアップ言語の範囲を広げずに、どうしてデータの問題を解決できるのでしょうか？

XMLはあらゆる
種類のデータ形式
に対応できるマー
クアップ言語です。

```
<html>
<head>
<title>iRock - The Blog for Cube Purists</title>
<script type="text/javascript" src="iRock.js"></script>
<script type="text/javascript" src="iRockData.js"></script>
<script type="text/javascript">
    var RockData = function(data) {
        this.data = data;
        this.image = "img/rock.png";
        this.getHTML = function() {
            return ('<div>' + this.data + '</div>');
        }
    };
    var rock = new RockData("I'm a real pearl!");
    window.onload = function() {
        document.getElementById("content").innerHTML = rock.getHTML();
    }
    </script>
</head>
<body>
<h1>iRock</h1>
<div id="content"></div>
</body>
</html>
```

ウェブページ



```
<html>
<head>
<title>iRock - The Blog for Cube Purists</title>
<script type="text/javascript" src="iRock.js"></script>
<script type="text/javascript" src="iRockData.js"></script>
<script type="text/javascript">
    var RockData = function(data) {
        this.data = data;
        this.image = "img/rock.png";
        this.getHTML = function() {
            return ('<div>' + this.data + '</div>');
        }
    };
    var rock = new RockData("I'm a real pearl!");
    window.onload = function() {
        document.getElementById("content").innerHTML = rock.getHTML();
    }
    </script>
</head>
<body>
<h1>iRock</h1>
<div id="content"></div>
</body>
</html>
```

買い物のトランザクション

Items in your cart	Quantity	Your Price
Head First HTML, CSS & XHTML, 1st Ed.	1	\$39.99
Head First SQL, 1st Ed.	1	\$44.99
<small>Have a Discount Code? Enter it here: _____</small>		
<small>NOTE: Discounted prices will be shown on the order summary page.</small>		
<small>Tax and Shipping will be applied at checkout.</small>		
<small>Shipping Details: _____</small>		
Subtotal		
\$84.98		

曲のリスト

Fish In The Jailhouse	4:22	Tom Waits
Bottom Of The World	5:43	Tom Waits
Lucinda	4:53	Tom Waits
Ain't Goin' Down To The Well	2:28	Tom Waits
Lord I've Been Changed	2:28	Tom Waits
Puttin' On The Dog	3:39	Tom Waits
Road To Peace	7:17	Tom Waits

ブログのエントリ

9/24/2008 巨大的キューブが私の名前を呼びながら追いかけてくる夢を見る。 by Puzzler Rudy
9/19/2008 やった！2週間もかかったけれど、新しいキューブがとうとう完成！ by Puzzler Rudy
9/5/2008 ついに3x3x7のキューブ入手！しばらくブログは書けないと思う…… by Puzzler Rudy
9/3/2008 キューブパズルの取り扱いをやめてしまった近所のおもちゃ屋の抗議集会に参加。パズラーちよ、強張れ！ by Puzzler Rudy

XMLが強力な理由はその柔軟性にあります。HTMLはタグと属性が固定されているのに対して、XMLはタグと属性が定義されているのではなく、タグと属性の作成と利用のルールが決められているのです。特定のデータを表現するタグと属性を決めるのは、XMLの個々のアプリケーションの仕事なのです。

XMLを使うとあなたのデータをあなたのやり方で扱えます

XMLの素晴らしいところは、ちょっとしたタグと属性の鍊金術を使えば、いかなる用途のマークアップ言語でも料理できるカスタムタグを誰でも作れるところにあります。さまざまな問題を解決するXML言語がすでにたくさん作られているので、あなたのニーズにあったものがあれば、それを使うのも悪くありません。でもまったく独自にマークアップ言語を作る誘惑に抗うのは難しいものです。

HTMLコードと同様、このXMLコードは要素の階層で構成されます。

```
<movie>
  <title>ローリング・キッズ </title>
  <releaseDate>01/13/1989</releaseDate>
  <director>グレーム・クリフォード</director>
  <summary>スケートボーダーの少年が、兄の死を調べる。</summary>
</movie>
```

映画の詳細情報はそれぞれ一意のタグの中に格納されます。

映画の詳細は<movie>タグの中に含まれています。

このようなXMLマークアップ言語をみたことはないでしょうが、これらのタグはデータを解読することを可能にします。重要なのは、これらのタグは格納されているデータに特有なものであることです。<director>というタグは映画のディレクターが格納されているとき意味があります。



エクササイズ

以下のタグと説明を対応付け、さらにタグの前にHTMLとXMLのどちらのタグか書いてください。

.....	<itunes:author>	ウェブページの中の太文字のテキスト。
.....		オンラインニュースのタイトル。
.....	<title>	ウェブページの入力コントロール。
.....		電話をかけた人のために音声に変換されるテキスト。
.....	<input>	iTunesポッドキャストの制作者。
.....	<prompt>	ウェブページのインラインコンテンツ。



以下のタグと説明を対応付け、さらにタグの前にHTMLとXMLのどちらのタグか書いてください。

エクササイズ

答え

XML	<itunes:author>	ウェブページの中の太文字のテキスト。
HTML		オンラインニュースのタイトル。
XML	<title>	ウェブページの入力コントロール。
HTML		電話をかけた人のために音声に変換されるテキスト。
HTML	<input>	iTunesポッドキャストの制作者。
XML	<prompt>	ウェブページのインラインコンテンツ。

XMLはただのテキストにすぎない

HTMLと同様、XMLデータは单なるテキストです。通常のプレインテキストに格納されています。ただし、HTMLファイルの拡張子が.htmlや.htmであるのに対して、XMLファイルの拡張子は.xmlになります。

XMLデータは
拡張子が.xmlの
ファイルに
格納されます。



XML + HTML = XHTML

XMLとHTMLは拡張子が違いますが、重要な接点があります。XHTMLと呼ばれるHTMLのバージョンは、XMLの厳格なルールに従っています。たとえば、XHTMLウェブページの開始タグには、かならず終了タグがなければなりません。HTMLは高速だけどゆるい構文になっているので、たとえば<p>と</p>のようにタグを対応させなくても、見逃してくれます。XHTMLは大目に見てくれないので、こうしたタグは必ず対応している必要があります。

XHTMLはXMLの
厳格な文法を
適用したHTMLの
バージョンです。

HTML

これはHTMLのテキストパラグラフです。<p>

<p>タグはHTMLコードではよく
単独で使われます。パラグラフの
先頭または末尾を示します。

XHTML

<p>これはXHTMLのテキストパラグラフです。</p>

タグを含むコンテンツは
XHTMLの中ではタグが
対になっている必要があります。

HTMLとXHTMLでもうひとつ重要な違いは、
のような空タグの扱いです。
終了タグがないことを示すために、空白とスラッシュを追加する必要があります。

HTML

これは文です。

改行タグはスラッシュなしの
HTMLでよく書かれます。

XHTML

これは文です。

XHTMLでは空白と
スラッシュが必要です。

HTMLとXHTMLの違いでさらに重要なのは、XHTMLではすべての属性の値を引用符で囲む必要がある点です。

HTML

Go home

属性の値は二重引用符では
ないのでXHTMLのルールに
違反しています。

XHTML

Go home

XHTML属性の値は二重引用符で
囲む必要があります。

ルビーが必要としているブログデータのXMLによるモデル化をXHTMLが直接解決するわけではありませんが、XMLの最も重要な構文ルールのいくつかを説明するのに適しています。このルールは、ルビーのブログデータ言語を含む、あらゆるXMLベースの言語に適用されます。

特別座談会



今夜の対談：

HTMLとXMLがウェブデータについて勝負します

HTML:

まったく君には混乱させられるよ。私はウェブの背骨な
わけだけど、君のおかげでいまでは多くの人が混乱して
いるよ。

XML:

それは私のせいじゃない。君の視野が狭いだけだよ。
ウェブページのことだけしか頭にないんだろう。私は心が
広いから、あらゆる種類のデータを表現できるんだ。

でも、私がいないと困るだろう。ブラウザはHTMLコー
ドしか表示できないから、君の扱いがわからないんだ。

私は謎の男なんだな。顔のない男というか透明人間とい
うか。だから自分の姿を表すときは、君を必要とするわ
けだ。

そんなことがありえるのかい？ 姿のないデータを誰が
気にする？

ちょっとそれは言いすぎじゃないかい？ 銀行取引、世
論調査、天気予報など、見えないデータを扱う世界もあ
るんだぜ。

こうしたデータがウェブで見られるのは私のおかげだよ。

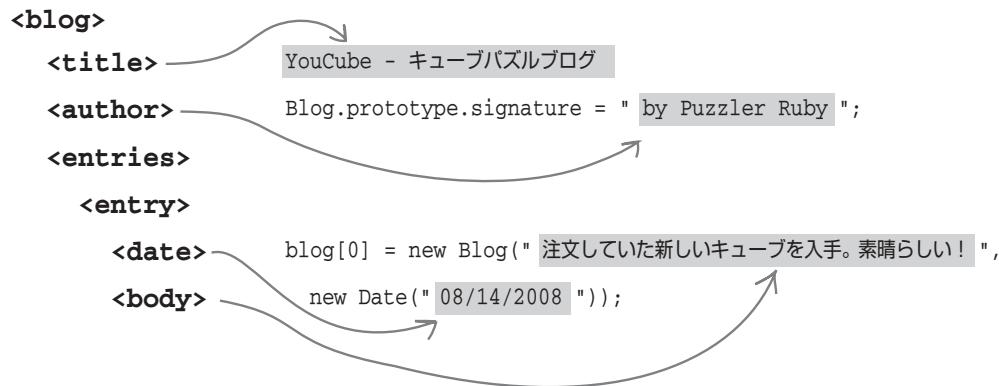
そうなんだけど、ウェブブラウザに表示される前、それ
らのデータがどう格納されているかわかるかい？ パラ
グラフやテーブルとしてではなく、私を使って格納され
るんだぜ。それは私が構造と文脈を提供できるからさ。
つまり私はデータを処理しやすくしているわけさ。

それを聞いて安心したよ。

まさにそうさ。私はデータがどう見えるのかは一切考
えない。かわりに、データの意味に集中しているんだ。
ウェブブラウザが使われるかぎり、私がデータを表示す
るときには、君の力が必要というわけさ。

XMLをYouCube ブログデータに適用する

XHTMLはウェブページの構造と信頼性を向上させるXMLのアプリケーションです。しかし、YouCubeブログに関しては、ブログデータに特有なモデルをXML言語でカスタマイズする必要があります。そのためには、ブログで要求されるさまざまなデータを見積もり、XMLタグ階層の文脈に適合させる方法を考える必要があります。



自分で考えてみよう

ブログの格納に使うXML言語を考えて、それを使ってブログエントリのコードを書いてください。タイトル、日付、著者、エントリそのものなどを項目として考慮してください。

<blog>

自分で考えてみよう の答え

ブログの格納に使うXML言語を考えて、それを使ってブログエントリのコードを書いてください。タイトル、日付、著者、エントリそのものなどを項目として考慮してください。

ブログ全体が
<blog>タグに
含まれて
います。

ブログエントリ
の集まりは
<entries>の
中に格納
されます。

```
<blog>
  <title>YouCube - キューブパズルブログ</title>
  <author>Puzzler Ruby</author>
  <entries>
    <entry>
      <date>11/14/2007</date>
      <body>注文していた新しいキューブを入手。素晴らしい！</body>
    </entry>
  </entries>
</blog>
```

<title> タグはブログの
タイトルを格納します。

ブログの著者が誰なのか
格納するタグです。

ブログエントリは
<entry>タグで
表現されます。

ブログエントリの日付と本文には
それぞれタグがあります。

素朴な疑問に答えます

⑨：ブログデータを通常のテキストで格納しないのはなぜですか？

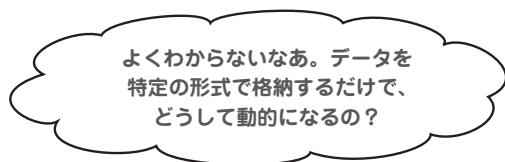
A: そういうこともできますが、データを調べて、日付と本文をもつ個々のブログエントリに分割するスクリプトコードを作るのは苦労します。XMLはわかりやすい構造をデータに付加するので、データの個々のフィールドを容易に区別できるようになります。ブログの場合、ブログのタイトルや書き手はもちろん、日付や本文でブログエントリを区別できます。

（）：<entries>タグはXMLプログデータにほんとに必要ですか？

A: 厳密に言えばなくてもいいのですが、あった方がデータの形式がより構造化され、理解しやすくなります。たとえば、ブログデータに<entries>タグがなかったとすると、<title>と<author>はひとつだとして、<entry>タグが複数あってもいいのか、わからなくなります。<entries>タグがあると、ブログエントリを複数集めたことがわかるので、データがより構造化され、データの使い方がより明確になります。

⑨ : XMLとAjaxの関係はどうなっていますか?

Ajax : AjaxはAsynchronous JavaScript And XMLの略です。ある時期までXMLはAjaxと直接関係していましたが、それも今はや過去のことです。XMLはAjaxの方程式の一部だったのですが、Ajaxの概念は拡張され、XML以外のデータにも対応しています。とはいえ、いまでもAjaxアプリケーションの大部分でXMLが基礎になっています。XMLはデータのモデリングのための優れた機構を提供してくれるからです。この章の後半で説明しますが、AjaxとXMLの関係は、JavaScriptがAjaxをサポートすることで成立しています。JavaScriptは、Ajaxのリクエストと応答を実行するにあたって、データ形式をXMLだけに固定しているわけではありませんが、些細なデータを扱うとき以外はXMLを扱う方が処理がより簡単になります。Ajax純粋主義者は、XMLとAjaxが互いに関係していないと主張するかもしれませんのが、実際には互いに手をとりあっています。むかしのXMLの意味が嫌われてるとしても、いまでも十分通用します。非同期(asynchronous)の部分については、後ほど説明します。プログの格納に使うXML言語を考えて、それを使ってプログエントリのコードを書いてください。タイトル、日付、著者、エントリそのものなどを項目として考慮してください。



。。



XMLだけでは動的になりませんが、AjaxとDOMを組み合わせれば、歯車が動き出します。

XMLはAjaxで最も頻繁に使われるデータ形式なので、YouCubeのデータ駆動バージョンでサーバとクライアントの間でやり取りされるブログデータを表現するのに論理的に適しています。XMLは高度に構造化されているので、データのやり取りに最適です。

XMLはHTML(XHTML)と似たところがあるので、DOMを使えばXMLデータをノードの木としてたどることができます。つまり、XMLノードの木をたどり、注意深く目的のデータを特定し、それをウェブページに動的に組み込む処理がJavaScriptで書けるということです。こうした機能を実現できるので、XMLは動的なデータ駆動ページを構築するためのデータの優れたソリューションなのです。

YouCubeにAjaxを導入する

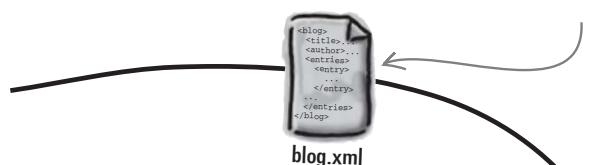
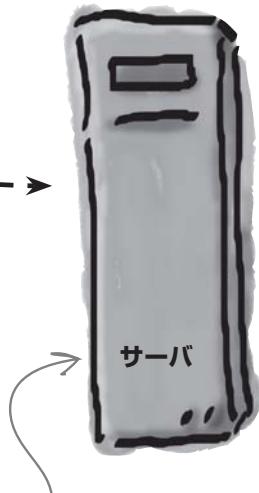
XMLのブログ文書ができたので、ルビーはAjaxの力を借りて YouCubeのページに動的に読み込ませようとしています。



3

サーバはブログファイルのデータをひとつにまとめてブラウザに応答します。

Ajax レスポンスとして XML ブログファイルのコンテンツが返されます。



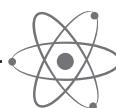
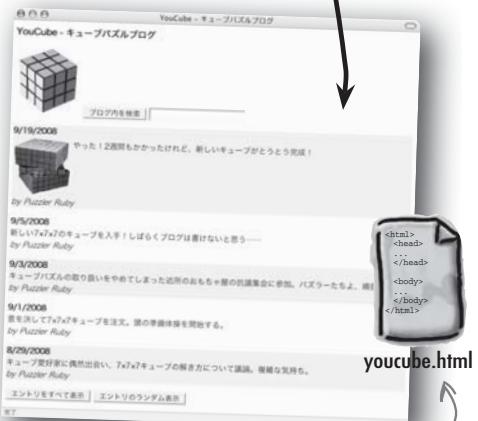
レスポンス

4

ブラウザは XML データをほどいてウェブページに取り込みます。

サーバ側スクリプト (JavaScript ではありません) は Ajax リクエストを処理してレスポンスデータを準備する必要があります。

XML データがウェブページの HTML コードに取り込まれるとブラウザに表示されます。



頭の体操

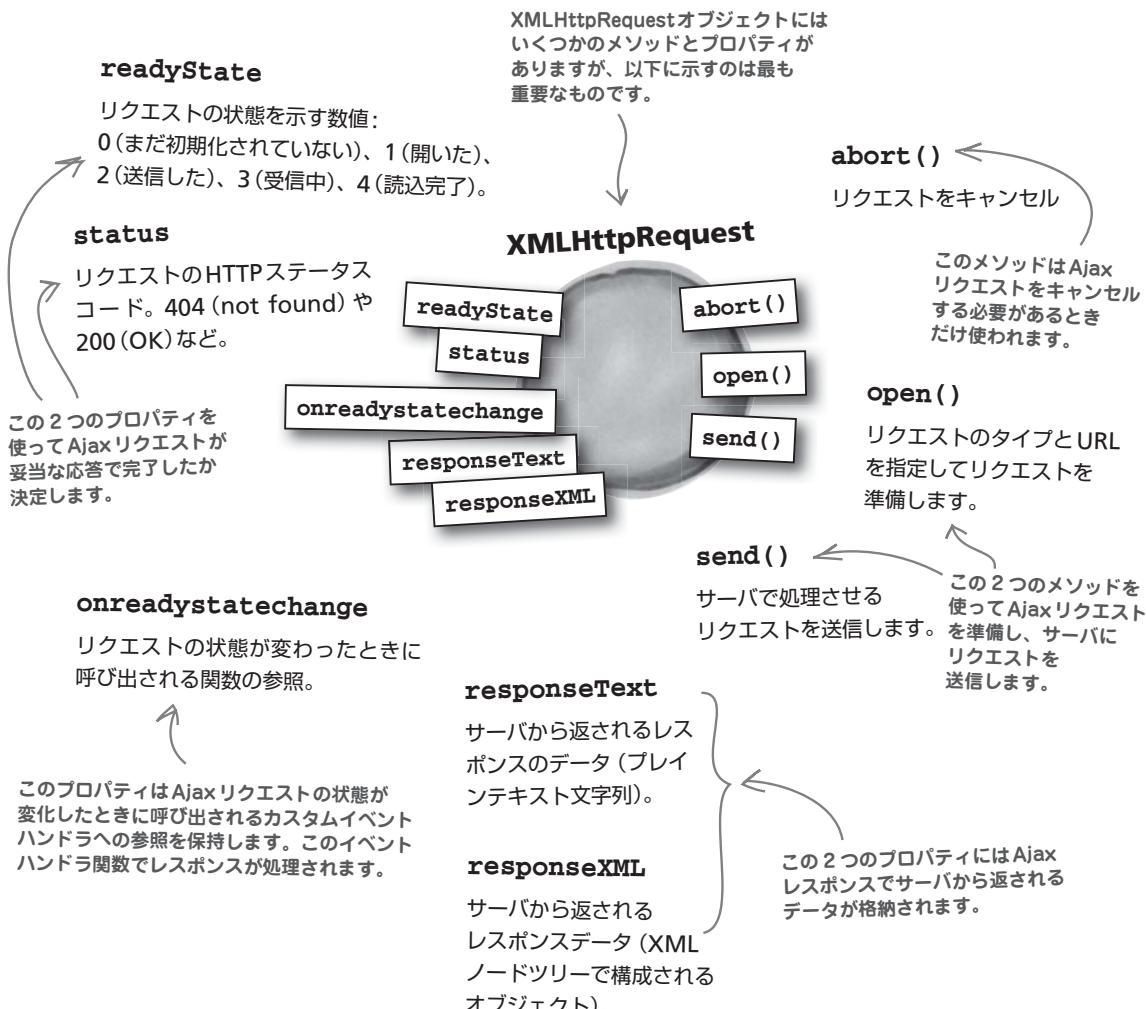
Ajax のリクエストとレスポンスを扱う JavaScript コードはどんな内容だと思いますか？

JavaScript コードは Ajax リクエストを作成し、レスポンスの処理をウェブページの中で実行します。



JavaScript と Ajax の架け橋： XMLHttpRequest

JavaScript には XMLHttpRequest と呼ばれる組み込みオブジェクトがあり、これを使って Ajax のリクエストを開始し、レスポンスを処理することができます。このオブジェクトはかなり複雑であり、Ajax をサポートするためのさまざまなメソッドとプロパティが含まれています。



XMLHttpRequestはとても複雑

XMLHttpRequestは信じられないくらい強力で驚くほど柔軟です。それだけに複雑でもあり、基本的なAjaxリクエストさえ、JavaScriptのコードはかなりの量になります。これはブラウザの一貫性のなさも原因のひとつなのですが、オブジェクトの動作を調整するためのさまざまなオプションがあるため、いくつかのデータを動的に迅速に移動したいだけであっても、混乱してしまうでしょう。

以下のコードは、さまざまなブラウザで動くように XMLHttpRequestオブジェクトを作成しているだけですが、かなりの量になります。

```
var request = null;
if (window.XMLHttpRequest) {
    try {
        request = new XMLHttpRequest();
    } catch(e) {
        request = null;
    }
    // ActiveX (IE) バージョンを試してみます
} else if (window.ActiveXObject) {
    try {
        request = new ActiveXObject("Msxml2.XMLHTTP");
        // IEの古いバージョンに対応するActiveXオブジェクトを試してみます
    } catch(e) {
        try {
            request = new ActiveXObject("Microsoft.XMLHTTP");
        } catch(e) {
            request = null;
        }
    }
}
// このtry-catch文はJavaScriptの高度なエラー処理機構です。
// これを使うと実行時エラーをスクリプトで優雅に対処できます。
```

XMLHttpRequestオブジェクトを作成したら、リクエストハンドラ関数を設定して、リクエストを開きます。

```
request.onreadystatechange = handler;
request.open(type, url, true); // 非同期にします
// リクエストをopenで開くと送信できる状態になります。
// リクエストの種類(GETかPOST)も決めます。
```

リクエストを開くときには、リクエストのタイプ ("GET" か "POST")、サーバのURL、リクエストが非同期かどうかの論理値を指定する必要があります。非同期リクエストは、スクリプトを中断させることなく、バックグラウンドで処理を実行します。ほとんどのAjaxリクエストは非同期です。

XMLHttpRequest
オブジェクトは
強力ですが使うのに
骨が折れます。

このコードでは XMLHttpRequest
オブジェクトを作成するためにいく
つかのアプローチを試しています。
ブラウザ(IE)による違いを吸収する
ためです。



マニア向け情報

XMLHttpRequest
オブジェクトを作成する
ときの問題として、ブラウザ
がこのオブジェクトの実装
を提供しなければならない
ことがあります。幸い、メ
ソッドとプロパティはどの
ブラウザでも一貫しています。
ブラウザによる違いが
あるのはオブジェクトの作
成方法です。

GETとPOSTについて

Ajax リクエストのタイプは非常に重要です。何をサーバに送信するかに影響するだけでなく、リクエストの意図にも関わります。リクエストのタイプ（あるいはリクエストのメソッドとも言われます）のひとつである GET は、主にサーバからデータを引き出すのに使われます。GET はサーバになんら影響を与えません。もうひとつの POST は、主にサーバにデータを送信するときに使われ、送信されたデータにサーバが応答するとき、サーバの状態がなにかしら変わります。

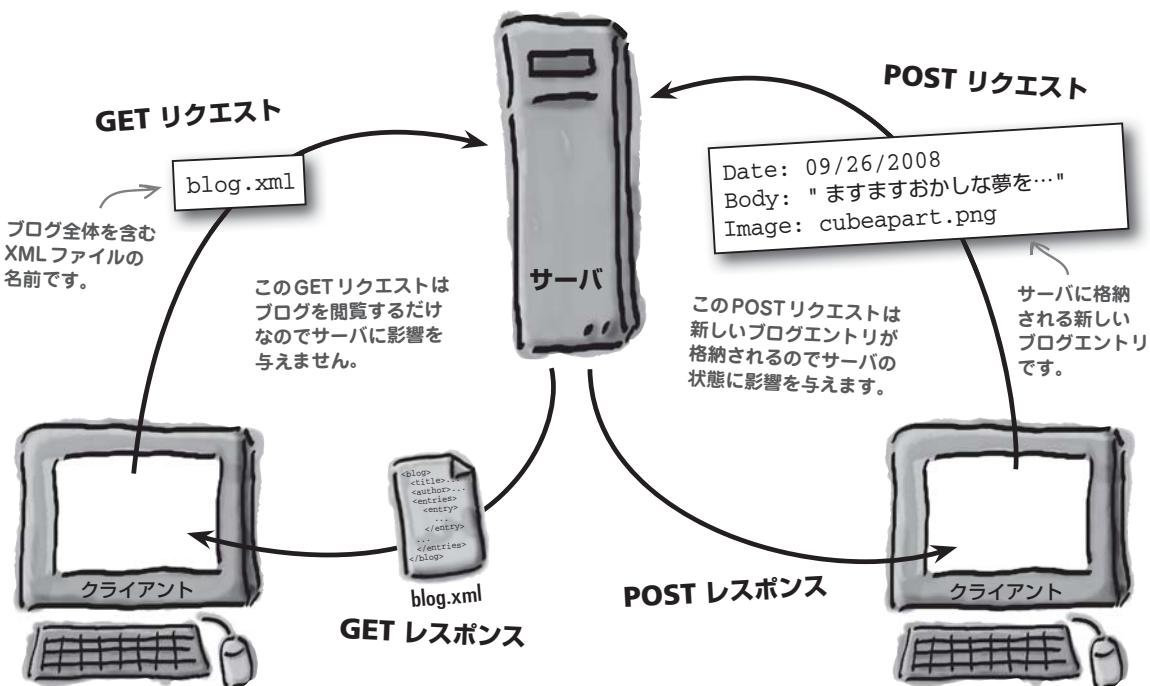
Ajax で使われるリクエストには GET と POST の 2 つのタイプがあります。どちらも HTML フォームを送信するときに使うリクエストと同じです。

GET

サーバの状態を変更することなく、データを引き出すときに使います。必要ならば、サーバに送信する小さなデータを URL に指定できます。GET は、サーバにある XML ファイルからブログデータを引き出すのに最適です。

POST

データを送信することで、サーバの状態がなにかしら変わる、たとえば、データベースにデータを保存する、といった場合に使います。もちろんレコードでデータが返されます。POST はウェブフォームを使ってブログに新しいエントリを動的に追加する、といった作業に最適です。



GETかPOSTか？ XMLHttpRequestのリクエスト

リクエストのタイプを決めて、リクエストを開く際にそれを指定したら、いよいよサーバにリクエストを送信します。リクエストを送信するコードは、リクエストがGETかPOSTかによって変わってきます。

GETリクエストとURLを指定してリクエストを開きます。

```
request.open("GET", "blog.xml", true); // 非同期にします
```

データなしでリクエストを送信します。このためsend()の引数はnullになります。

XML ブログデータをリクエストします。サーバにあるblog.xmlファイルをGETリクエストで取得します。

GET リクエスト

blog.xml



POST リクエスト

Date: 09/26/2008
Body: "ますますおかしな夢を…"
Image: cubeapart.png

新しいブログエントリをPOSTリクエストでサーバに送信します。

POSTリクエストとサーバのURL(この場合サーバスクリプト)を指定してリクエストを開きます。

```
request.open("POST", "addblogentry.php", true); // 非同期にします  
  
request.setRequestHeader("Content-Type", "application/x-www-form-urlencoded; charset=UTF-8");  
  
request.send("09/26/2008& ますますおかしな夢を…&cubeapart.png");
```

send()の引数にデータを渡してリクエストをデータ付きで送信します。



GETやPOSTにまつわるあれこれで悩む必要はありません。

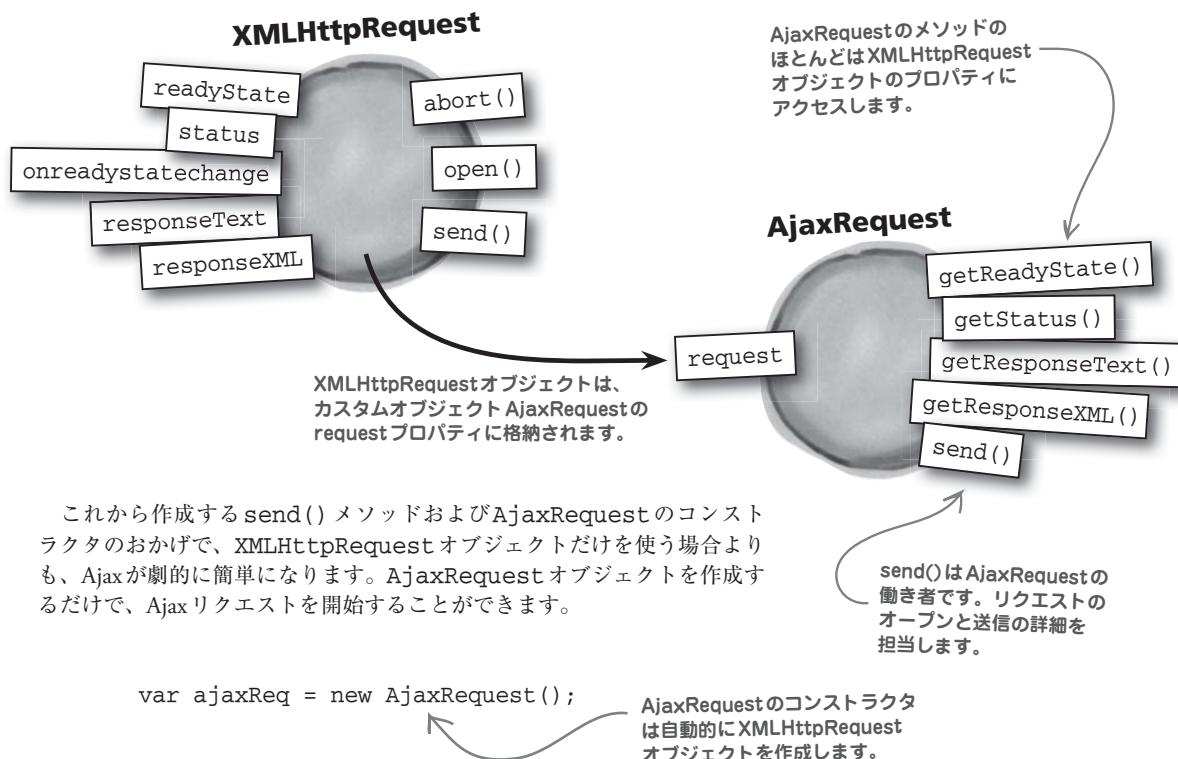
HTMLからGETやPOSTを使った経験がなくても心配しないで。YouTubeにおける役割がはっきりしてくれれば、だんだん意味もわかるでしょう。

XMLHttpRequestを楽に使えるようにする

XMLHttpRequestオブジェクトは信じられないくらい強力ですが、すでにご承知のとおり、その学習曲線はかなり急なカーブを描きます。それだけでなく、これを使うあらゆるAjaxアプリケーションにある量の決まりきったコードを書く必要があります。このため、は XMLHttpRequestオブジェクトを簡単に使えるようにサードパーティのライブラリが多数作られています。それらの多くはJavaScriptの機能を拡張しているので、さらに学習の必要があります。

このような理由から、YouCubeの戦略を支援するために、 XMLHttpRequestを補助する最小限のカスタムオブジェクトを作成します。 XMLHttpRequestオブジェクトに苦戦したり、サードパーティのライブラリを勉強するかわりに、Ajaxを使ってできることに集中することができます。このカスタムオブジェクト AjaxRequestには、 XMLHttpRequestオブジェクトをもっと使いやすくするための最小限の工夫が施されています。

カスタムオブジェクト
AjaxRequestを使うと
Ajaxリクエストが
簡単に作れます。



これから作成する send() メソッドおよび AjaxRequest のコンストラクタのおかげで、 XMLHttpRequest オブジェクトだけを使う場合よりも、 Ajax が劇的に簡単になります。 AjaxRequest オブジェクトを作成するだけで、 Ajax リクエストを開始することができます。



JavaScript マグネット

カスタムオブジェクトAjaxRequestは標準オブジェクトXMLHttpRequestをラップしたもので、Ajaxリクエストの送信とそのレスポンスを処理するための簡単なインターフェースを提供します。AjaxRequestオブジェクトのsend()メソッドから鍵となるコード断片が消えてしまいました。マグネットを使ってこのメソッドを完成させてください。

```
AjaxRequest.prototype.send = function(type, url, handler, postDataType, postData) {
    if (this.request != null) {
        // 以前のリクエストをクリアします。
        this.request.abort();
    }

    // ダミーのパラメータを付加してブラウザのキャッシュを回避します。
    url += "?dummy=" + new Date().getTime();

    try {
        this.request.onreadystatechange = .....;
        this.request.open(....., ..... , true); // 非同期にします。

        if (type.toLowerCase() == "get") {
            // GETリクエストをデータなしで送信します。
            this.request.send(.....);
        } else {
            // POSTリクエストを送信します。最後の配列はデータです。
            this.request.setRequestHeader("Content-Type", .....);
            this.request.send(.....);
        }
    } catch(e) {
        alert("サーバとの通信でAjax エラー。\\n" + " 詳細 : " + e);
    }
}
```

handler

url

null

type

postDataType

postData



JavaScript マグネットの答え

カスタムオブジェクトAjaxRequestは標準オブジェクトXMLHttpRequestをラップしたもので、Ajaxリクエストの送信とそのレスポンスを処理するための簡単なインターフェースを提供します。AjaxRequestオブジェクトのsend()メソッドから鍵となるコード断片が消えてしまいました。マグネットを使ってこのメソッドを完成させてください。

send()はAjaxリクエストを送信します。
リクエストの詳細を引数に指定します。

```
AjaxRequest.prototype.send = function(type, url, handler, postDataType, postData) {
    if (this.request != null) {
        // 以前のリクエストをクリアします。
        this.request.abort();
    }
    // ダミーのパラメータを付加してブラウザのキャッシュを回避します。
    url += "?dummy=" + new Date().getTime();
    try {
        this.request.onreadystatechange = handler;
        this.request.open(type, url, true); // 非同期にします。
        if (type.toLowerCase() == "get") {
            // GETリクエストをデータなしで送信します。
            this.request.send(null);
        } else {
            // POSTリクエストを送信します。最後の配列はデータです。
            this.request.setRequestHeader("Content-Type", postDataType);
            this.request.send(postData);
        }
    } catch(e) {
        alert("サーバとの通信でAjaxエラー。" + " 詳細 : " + e);
    }
}
```

リクエストが
POSTリクエ
ストのとき
だけデータは
サーバに送信
されます。

カスタムハンドラ関数は
リクエストに対するサーバ
のレスポンスを処理する
ために呼び出されます。

send()の引数typeでリクエストが
GETとPOSTのどちらなのか決めます。

このコードはAjaxRequestのコンストラクタ
と他のメソッドとともにJavaScriptの
外部ファイルajax.jsに格納されます。



ajax.js

Ajaxリクエストの意味を理解する

カスタムオブジェクト AjaxRequest はコンストラクタといくつかのメソッドで構成されます。メソッドのひとつ send() は特に便利です。サーバに対してAjaxリクエストを準備し発行するときに使われます。send() を使って発行されるすべてのAjaxリクエストは、GETまたはPOSTのどちらかのリクエストになります。これはHTMLフォームで送信されるリクエストと同じです。両者の違いは、Ajaxリクエストではページの再読み込みが要らない点です。

`send(type, url, handler, postDataType, postData)`

type

リクエストのタイプ (GETかPOST)。

url

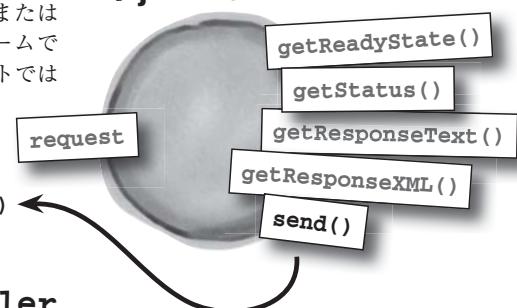
サーバのURL (YouTubeの場合 blog.xml)。

必要なら URL にデータを含めます。

postDataType

送信されるデータのタイプ (POSTの場合だけです。GETでは必要ありません)。

AjaxRequest



handler

レスポンスの処理に使われるコールバック関数。

postData

送信されるデータ (POSTの場合だけです。GETでは必要ありません)。POSTデータを送信する形式はいくつかあります。



リクエストの
処理のこと
で
パニックにな
らないで。

すべてのAjaxリクエストで、これらと同じ情報断片が必要です。ただしGETリクエストの場合、最後の2つの引数は省略できます。send() の最初の3つの引数が最も重要で、ほとんどの簡単なAjaxリクエストではこの3つで十分です。例題では send() の呼び出しでは最初の3つの引数を使って、サーバにある movies.xml というファイルから XML データを (GETで) リクエストしています。

リクエストの
タイプ

リクエストされたデータ
ファイルのURL。

ajaxReq.send("GET", "movies.xml", handleRequest);

このコードはすでに
AjaxRequestオブジェクト
が作成され、変数 ajaxReq
に格納されていると
想定しています。

リクエストに対する
レスポンスを処理するため
呼び出されるカスタム関数。

AjaxリクエストがJavaScriptコードでどのように扱われているか、これから詳しく説明していきます。いまのところ、カスタムリクエストハンドラ関数をリクエストに対して設定する必要があること、この関数はリクエストが完了したときに呼び出されること、このふたつを理解してください。

このリクエストはあなたのです

AjaxRequestオブジェクトのsend()が呼ばれると、Ajaxリクエストがサーバに送信され、サーバがリクエストの処理を行っている間でも、ウェブページではそのまま処理を続けられます。これはAjaxの非同期処理が輝くところです。リクエストが同期で処理されると、ページ操作は中断されてしまい、サーバがレスポンスを返すまで何もできなくなります。リクエストが非同期で処理されるので、ページ操作は中断されず、ユーザ体験は損なわれません。



リクエストが処理される間でもページ操作は中断されないので、ユーザは待つ必要がありません。ユーザがどんな操作を続けるかはページに依存します。YouCubeの場合、ブログの表示が成功するかどうかは、サーバからのAjaxレスポンスでブログデータが取得できるかどうかによります。この場合、ユーザ体験はAjaxレスポンス次第です。



非同期の
Ajaxリクエストは
リクエストがサーバで
処理される間でも
ページ操作を中断
することなく実現
できます。

重要ポイント

- XMLHttpRequestオブジェクトはAjaxリクエストを実行するための標準オブジェクトですが、これを使うのは骨が折れます。
- カスタムオブジェクトAjaxRequestは XMLHttpRequestを直接扱わなくてもAjaxを使うことができる便利な方法を提供します。
- AjaxリクエストではGETとPOSTのどちらかのタイプを使います。どちらを使うかは、サーバに送信するデータやサーバでのデータの作用によって決まります。
- AjaxRequestオブジェクトのsend()はAjaxリクエストを開いて発行する鍵になります。

素朴な疑問に 答えます

Q: Ajaxリクエストを実行するとき AjaxRequestオブジェクトは必要ですか？

A: いいえ。XMLHttpRequestオブジェクトを使って、直接Ajaxリクエストを発行し、その応答を処理することができます。しかし、AjaxRequestオブジェクトを使えば、もっと簡単になる場合があります。AjaxRequestオブジェクトは、あとで驚くようなことはしません。ただAjaxを使う作業を簡単にするために便利なオブジェクトなのです。Ajaxリクエストの組み立てに関わる「忙しい仕事」を引き受けってくれます。

Q: Ajaxのリクエスト/レスポンスは、HTTPのリクエスト/レスポンスと何か違いはありますか？

A: ウェブブラウザはHTTPのリクエスト/レスポンスを使ってウェブサーバからHTMLウェブページを取得します。Ajaxのリクエスト/レスポンスはHTTPのリクエスト/レスポンスと非常に似ていますが、重要な違いがあります。Ajaxはいつでも実行する

ことができ、また扱うデータもHTMLである必要はありません。実際、Ajaxの大きな利点のひとつは、いろいろな種類のデータをリクエストに使える点です。

Ajaxがいろいろな種類のデータを扱える点は重要なですが、どんなデータにもそれに適した大きさがあります。Ajaxは一度にページや文書データの全部を処理するように限定されではありません。実際、Ajaxは小さなデータ断片の処理に適合しています。そのため、小さなデータをリクエストし、そのデータをページに組み込むことで、ページ自体を動的に変更します。これらの処理では、ページの再読み込みを必要としません。

Q: Ajaxを使えば、ウェブページを動的に組み立てることが可能なんですか？

A: はい、Ajaxの要点はそれです。でも、ページを断片から組み

立てるだけではありません。組み立てるタイミングにも関わっています。Ajaxのリクエストと応答は、ページの使い勝手を妨げることなく、リアルタイムに発生します。言い換えると、ページのある小さな部分を更新する必要があるとき、ページ全体が再読み込みされるまでユーザは待たなくてすみます。ページの更新される部分の読み込みは、バックグラウンドで実行されるので、ページの残りの部分でのやり取りは継続できます。

Q: GETとPOSTの処理の違いは何でしょうか？

A: GETとPOSTの違いは、サーバで処理されるAjaxリクエストの方式の違いです。ただし、どちらもデータを動的にリクエストできる点では違いはありません。AjaxはGETとPOSTのどちらでも使えます。GETとPOSTは、データベースへの格納などのデータの状態を変更する処理がサーバで行われるかどうかで使い分けます。データベースへの格納がある場合はPOSTを、そうでなければGETを使います。

役割は何？

以下のAjax関連の用語をその内容説明と対応づけてください。

XMLHttpRequest

データを受信しますがサーバ上で何も変ません。

GET

Ajaxリクエストをサーバに送信します。その結果がレスポンスになります。

send()

データを受信します。サーバ上で何か変化があります。

AjaxRequest

Ajaxを実現する標準JavaScriptオブジェクト。

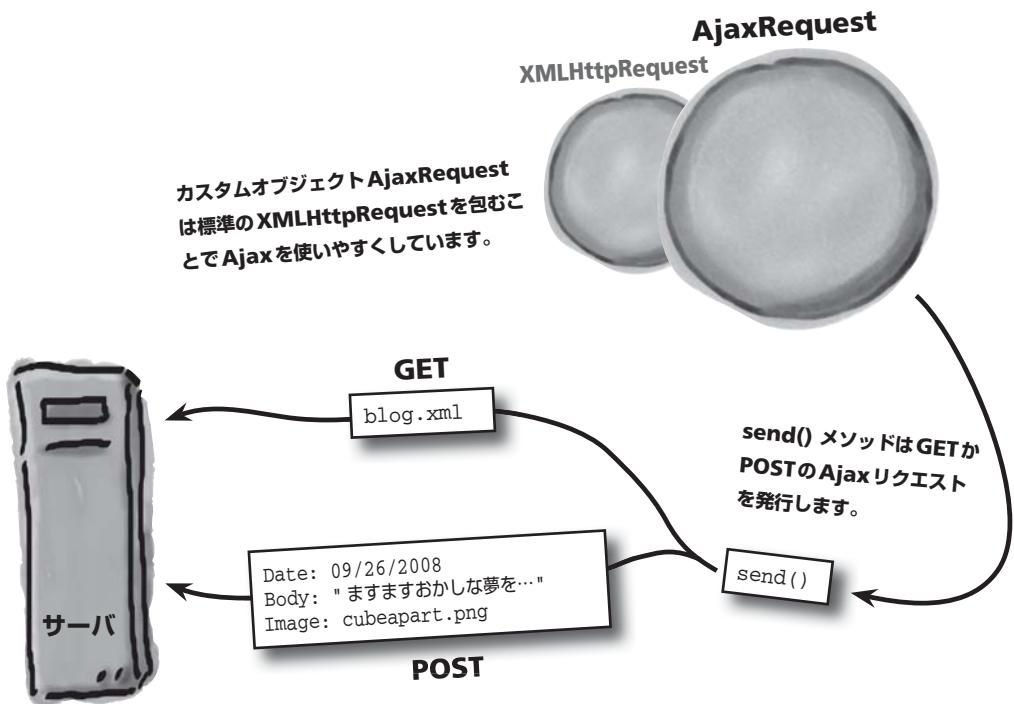
POST

Ajaxのリクエストとレスポンスを簡単にするために使われるカスタムオブジェクト。

役割は何？の答え

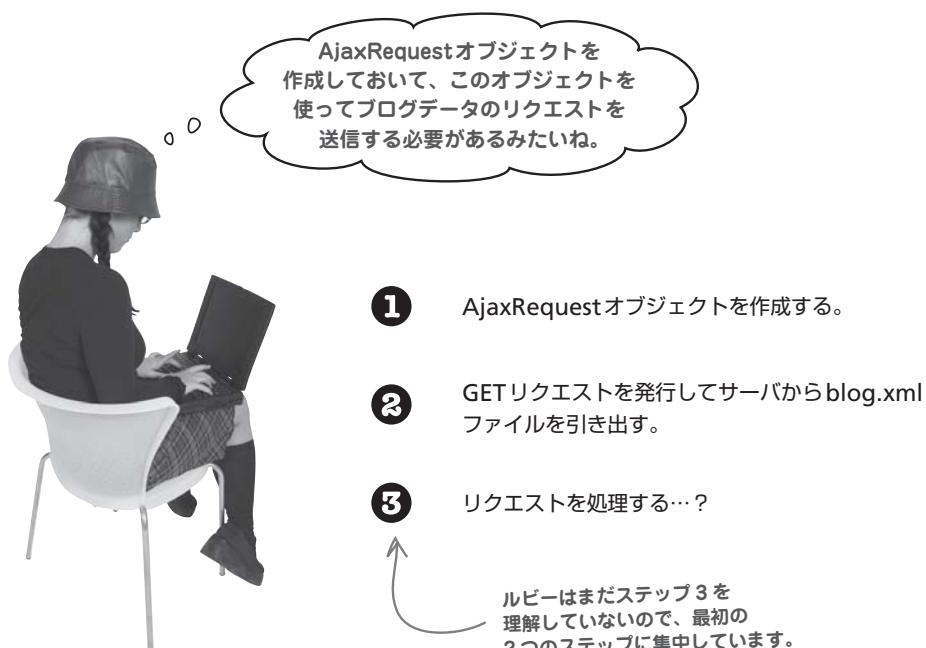
以下のAjax関連の用語をその内容説明と対応づけてください。

- | | |
|-----------------------|--|
| XMLHttpRequest | データを受信しますがサーバ上で何も変ません。 |
| GET | Ajaxリクエストをサーバに送信します。その結果がレスポンスになります。 |
| send() | データを受信します。サーバ上で何か変化があります。 |
| AjaxRequest | Ajaxを実現する標準JavaScriptオブジェクト。 |
| POST | Ajaxのリクエストとレスポンスを簡単にするために使われるカスタムオブジェクト。 |



リクエストオブジェクトを使って ページをインタラクティブにする

Ajaxがどのように使われているか、あるいはどんなデータにアクセスしようとしているのか、そうしたことに関係なく、Ajaxのデータ通信はリクエストから始まります。YouCubeをデータ駆動アプリケーションにするためのルビーの最初の仕事は、ブログデータを含むXMLファイルに対するAjaxリクエストを発行することです。



自分で考えてみよう

AjaxRequestオブジェクトを作成するコードを書いてください。これを使ってXMLブログデータのリクエストを送信してください。

```
.....  
.....
```

自分で考えてみよう の答え

AjaxRequestオブジェクトを作成するコードを書いてください。これを使ってXML ブログデータのリクエストを送信してください。

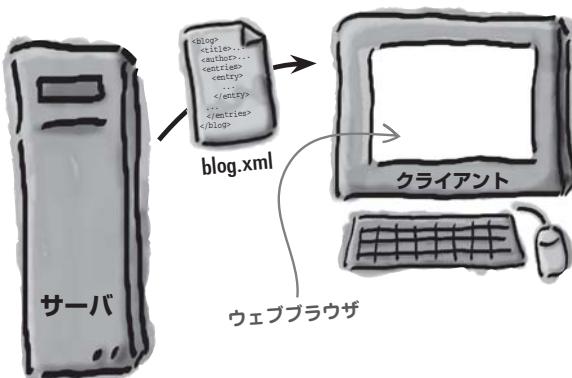
- 1 var ajaxReq = new AjaxRequest();

2 ajaxReq.send("GET", "blog.xml", handleRequest);

レponsを
カスム関数
handleRequest()
で処理します。

仕事が終わったら呼び出して

といったんAjaxリクエストが送信されると、ブラウザの役割が変わります。サーバからのレスポンスを待たないので、Ajaxリクエストは通常は**非同期**で実行されるので、ブラウザが画面の裏でレスポンスを待つ間でも、ユーザはページの操作を続けられます。Ajaxリクエストがサーバで処理されている間でも、ページは中断されません。サーバでリクエストの処理が完了すると、そのレスポンスはJavaScriptコードで処理されます。このときコールバック関数である**リクエストandler**が使われます。



- ① AjaxRequestオブジェクトを作成する。
 - ② GETリクエストを発行してサーバから blog.xml ファイルを引き出す。
 - ③ リクエストを処理する。

コールバック関数
handleRequest() は
カスタム関数として
スクリプトで提供する
必要があります。

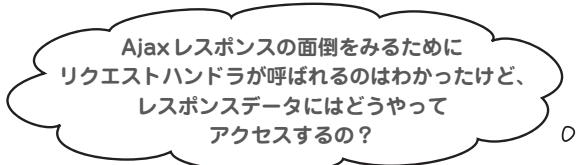
サーバからブラウザにレスポンスが送信されます。カスタムコールバック関数を使ってリクエストを処理します。

```
function handleButtons() {
    if ($('a').getElementsByAttribute('rel') == 4 && $('a').getElementsByAttribute('rel')[0].value != null) {
        var element = $('a').getElementsByAttribute('rel')[0].getElementsByTagName('img')[0];
        // Set the blog-wide signature
        blog.prototype.signature = ' ' + getSelf().getElementsByAttribute('author')[0];
        // Create the string for the entry object
        var entryString = 'entry' + getSelf().getElementsByAttribute('entry')[0];
        for (var i = 0; i < entries.length; i++) {
            if (entries[i].getElementsByAttribute('id')[0].value == element.value) {
                // Create the blog entry as the blog object
                var entryObject = new Object();
                getSelf().entries[i].getElementsByAttribute('img')[0].value;
                entryObject['date'] = getSelf().entries[i].getElementsByAttribute('date')[0];
                entryObject['text'] = getSelf().entries[i].getElementsByAttribute('text')[0];
                entryObject['url'] = getSelf().entries[i].getElementsByAttribute('url')[0];
                entryObject['view'] = getSelf().entries[i].getElementsByAttribute('view')[0];
                // Enable the blog buttons
                document.getElementsByAttribute('rel')[0].value;
                document.getElementsByAttribute('rel')[1].value;
                document.getElementsByAttribute('rel')[2].value;
                document.getElementsByAttribute('rel')[3].value;
                // Show the blog
                showBlog();
            }
        }
    }
}
```

```
handleRequest();
```

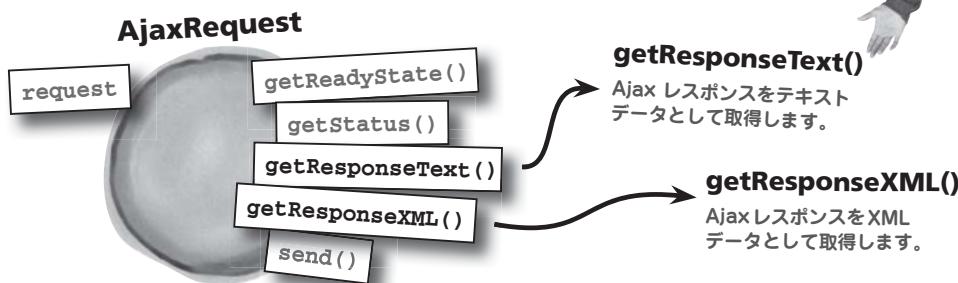
レスポンスを継ぎ目なく処理する

Ajaxリクエストが完了すると、リクエストハンドラであるコールバック関数 `handleRequest()` が呼ばれます。この関数は、リクエストが成功して完了したことを探らせるだけでなく、サーバから返されるレスポンスデータをもとにアクションを行います。



**AjaxRequestオブジェクトのメソッドは Ajax レスポンス
データにアクセスします。**

Ajax レスポンスで返されたデータにリクエストハンドラ関数からアクセスするとき、`AjaxRequest` の 2 つのメソッド `get.responseText()` と `get.responseXML()` が使えます。



レスポンスのデータにアクセスするときは、これらのメソッドのどれかを使います。どのメソッドを使うかは、データの形式によって決まります。レスポンスデータが XML の場合、`get.responseXML()` を使います。`get.responseText()` を使うと、意味のないデータが返ってきてしまいます。逆も真なりで、構造化された XML コードではなく、生のテキストの場合には `get.responseText()` を使います。

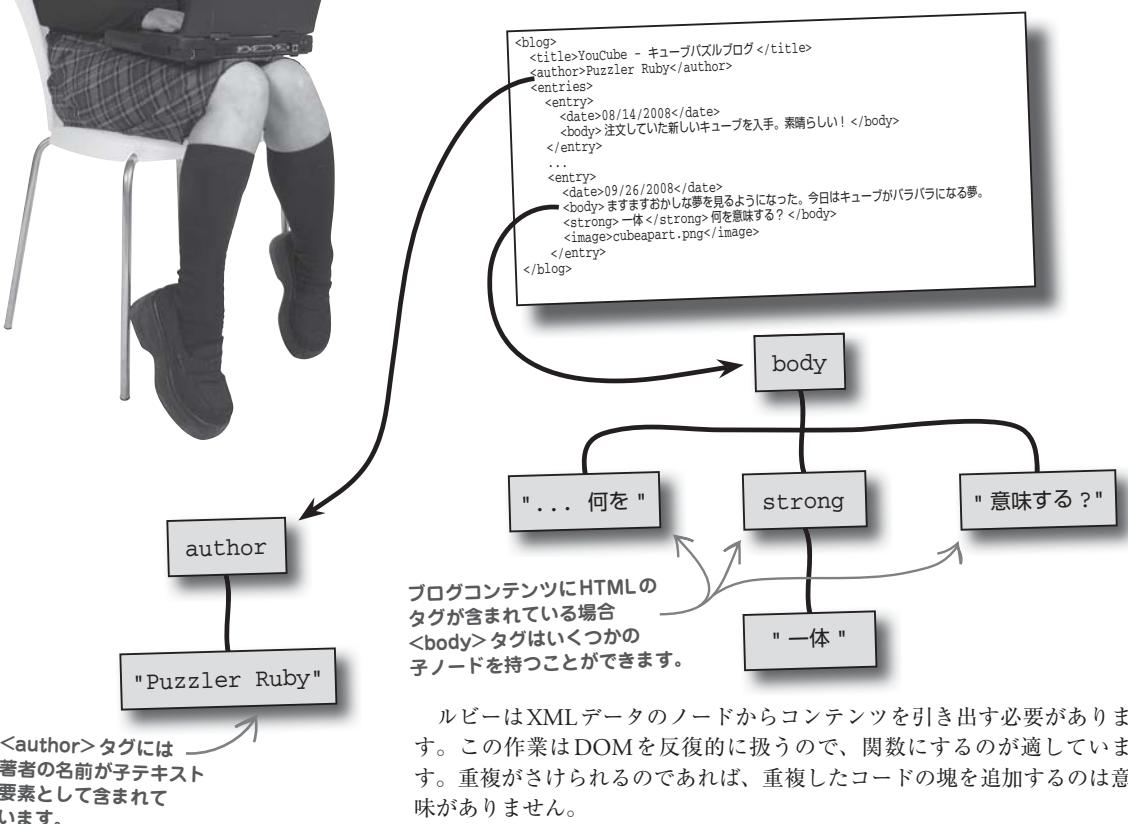
頭の体操

XMLコードはかなりHTMLコードと構造が似ているとしたら、リクエストハンドラでXMLログデータにどのようにアクセスしますか？



DOMからデータを引き出す

ルビーのパズルの腕前はかなりのものです。XMLレスポンスデータの処理でDOMを使うアイデアを閃いたのですから。DOMはHTMLデータをノードの木として扱います。しかしDOMはHTMLだけに限定されるわけではありません。XMLをノードの木として扱うこともできるのです。ルビーはYouCubeのXMLログデータをノードとしてとらえ直そうとしています。



ブログコンテンツにHTMLのタグが含まれている場合<body>タグはいくつかの子ノードを持つことができます。

ルビーはXMLデータのノードからコンテンツを引き出す必要があります。この作業はDOMを反復的に扱うので、関数にするのが適しています。重複がさけられるのであれば、重複したコードの塊を追加するのは意味がありません。



getText()の詳細

カスタム関数 `getText()` は DOM ツリーの要素（ノード）をたどってコンテンツに展開する作業を処理します。

```
function getText(elem) {
    var text = "";
    if (elem) {
        if (elem.childNodes) {
            for (var i = 0; i < elem.childNodes.length; i++) {
                var child = elem.childNodes[i];
                if (child.nodeValue)
                    text += child.nodeValue;
                else {
                    if (child.childNodes)
                        if (child.childNodes[0].nodeValue)
                            text += child.childNodes[0].nodeValue;
                }
            }
        }
    }
    return text;
}
```

この引数にはコンテンツを展開するべき要素が含まれています。

要素の子ノードをすべてループでたどります。

子ノードのコンテンツを変数 `text` に追加します。

子ノードの中に子ノードがある場合、最初の子ノードからテキストコンテンツを取得し、次に進みます。

変数 `text` を返します。要素の子ノードのコンテンツがすべて含まれています。



自分で考えてみよう

XML レスポンスデータはすでに `xmlData` という名前の変数に格納されているとします。YouCube ブログの署名を XML の `<author>` タグに格納された名前で設定するコードを書いてください。

自分で考えてみよう の答え

XML レスポンスデータはすでに `xmlData` という名前の変数に格納されているとします。YouCube ブログの署名を XML の `<author>` タグに格納された名前で設定するコードを書いてください。

```
Blog.prototype.signature = "by" + getText(xmlData.getElementsByTagName("author")[0]);
```

`signature` はクラスプロパティなので、`Blog` の `prototype` オブジェクトに設定する必要があります。

カスタムヘルパー関数 `getText()` を使って `<author>` タグからテキストコンテンツを抽出します。

XML データの `<author>` タグはひとつだけなので、最初の要素だけ取得します。



Ajax レスポンスを処理する

今週のインタビュー：Ajax リクエストハンドラ `handleRequest()`

Head First：Ajax リクエストに反応するのが得意だと聞きました。どんな関わり方なのでしょうか？

handleRequest()：Ajax のリクエストが降りてくると、応答を処理するのですが、その応答にはサーバから送信されたデータが含まれていることがあります。私の仕事は、まずそのリクエストがサーバで正しく処理されたか確認し、問題がなければ、データを受け取り、必要ならばウェブページに反映させます。

Head First：あなたが実際に呼び出されるのは、リクエストが完了したときですか？

handleRequest()：うーん、リクエストの処理を通じて何回も呼び出されるのですが、私がリクエストの終わりに行うことだけしか皆さん関心がないようです。

Head First：そうですか。ではリクエストが終わりであるのはどうやってわかるのですか？

handleRequest()：AjaxRequest オブジェクトには、リクエストの状態をチェックするためのメソッドがあるので、これを呼び出して、問題なくリクエストが完了したかどうかを確認します。

Head First：リクエストが完了したとき、実行する処理をどうやって知るのですか？

handleRequest()：それは私の仕事ではありません。私はカスタム関数なので、アプリケーションごと

に異なるのです。

Head First：それはどうしてですか？

handleRequest()：だってアプリケーションごとに応答データの使い方は異なりますよね。なので、私もアプリケーションごとに異なるのです。

Head First：ちょっと待ってください、つまり誰かがアプリケーションごとに毎回あなたを書く必要があるということですか？

handleRequest()：そうなんです。買い物かごのアプリケーションで Ajax の応答を処理する方法は、ブログのそれとは違っていますよね。サーバがリクエストの処理を完了したとき、Ajax によって確実に私が呼ばれますか、その後の処理は完全にアプリケーション次第なのです。

Head First：ということは、Ajax 対応のウェブページを作成する作業には、リクエストハンドラの作成も含まれるわけですね。

handleRequest()：その通りです。Ajax アプリケーションが行っている実際の処理のほとんどがそれなんですね。

Head First：いろいろ教えていただいて、たいへん感謝しています。

handleRequest()：応答するのが私の生き甲斐ですから。



JavaScriptの注釈者になつてみよう

JavaScriptの注釈者になってhandleRequest()で
何が処理されているか説明する注釈を加えてください。

7は魔法の数字です。リクエストを成功させる
7つのキーポイントがあります。

```

function handleRequest() {
    if (ajaxReq.getReadyState() == 4 && ajaxReq.getStatus() == 200) {
        // XML レスポンスデータを格納します
        var xmlData = ajaxReq.getResponseXML().getElementsByTagName("blog")[0];

        // ブログの署名を設定します
        Blog.prototype.signature = "by " + getText(xmlData.getElementsByTagName("author")[0]);

        // Blog エントリオブジェクトの配列を作成します
        var entries = xmlData.getElementsByTagName("entry");
        for (var i = 0; i < entries.length; i++) {
            // ブログエントリを作成します
            blog.push(new Blog(getText(entries[i].getElementsByTagName("body")[0]),
                new Date(getText(entries[i].getElementsByTagName("date")[0])),
                getText(entries[i].getElementsByTagName("image")[0])));
        }

        // ブログを表示します
        showBlog(5);
    }
}

```



JavaScriptの注釈者になつてみようの答え

JavaScriptの注釈者になってhandleRequest()で
何が処理されているか説明する注釈を加えてください。

7は魔法の数字です。リクエストを成功させる

7つのキーポイントがあります。

ブログの署名を
<author>タグの
コンテンツに設定します。

Ajaxリクエストの状態を
チェックして無事完了したか
確認します。

XMLデータには<blog>
タグがひとつしかないので
getElementsByTagName()
が返す配列の最初の要素を
取得します。

```
function handleRequest() {
    if (ajaxReq.getReadyState() == 4 && ajaxReq.getStatus() == 200) {
        // XML レスポンスデータを格納します
        var xmlData = ajaxReq.getResponseXML().getElementsByTagName("blog")[0];
    }
    // ブログの署名を設定します
    Blog.prototype.signature = "by " + getText(xmlData.getElementsByTagName("author")[0]);
}
```

```
// Blog エントリオブジェクトの配列を作成します
var entries = xmlData.getElementsByTagName("entry");
for (var i = 0; i < entries.length; i++) {
    // ブログエントリを作成します
    blog.push(new Blog(getText(entries[i].getElementsByTagName("body"))[0]),
              new Date(getText(entries[i].getElementsByTagName("date"))[0]),
              getText(entries[i].getElementsByTagName("image"))));
}
```

entry要素をすべて取得します。これには
ブログエントリが保持されています。
各エントリに直接アクセスできるときは
entry要素を取得する必要はありません。

```
// ブログを表示します
showBlog(5);
}
```

エントリに対応する
ブログオブジェクトを新規作成し、
Arrayオブジェクトのpush()を
使って配列の末尾に追加します。

showBlog()を呼び出して
最新の5つのブログエントリを
ページに表示します。

① AjaxRequestオブジェクトを作成する。

② GETリクエストを発行してサーバから
blog.xmlファイルを引き出す。

③ リクエストを処理する。 完了！

YouCubeはそのデータで稼働する

ルビーは YouCube を Ajax で作り替えたことで、時間が節約できて大喜びです。でも、ブログデータの読み込み中のページの使い勝手が気になっています。

YouCube の最新バージョンのファイルは <http://www.headfirstlabs.com/books/hfjs/> からダウンロードできます。

データをXMLコードに
切り離したおかげで、ブログはほんとに
よく動いているけど、ブログが読み込みで
忙しいことをユーザに知らせる方法ってないかしら?
ページが仕事していることをユーザに知らせる
手段があればいいんだけど。



このブログはXML
データで駆動して
います…

…ただしデータの読み込み中
空白のページが表示されるので、
混乱するユーザもいます。

自分で考えてみよう

`loadBlog()`のコードの一部が消えてしまいました。この関数はブログデータの読み込み中に `wait.gif` という画像を表示します。

ヒント：「blog」というIDでメインのブログのdivにアクセスします。

```
function loadBlog() {
```

```
ajaxReq.send("GET", "blog.xml", handleRequest);
```



自分で考えてみよう の答え

```
function loadBlog() {  
    document.getElementById("blog").innerHTML = "<img src='wait.gif' alt='Loading...'/>";  
    ajaxReq.send("GET", "blog.xml", handleRequest);  
}
```

アニメ対応画像のwait.gifを
ブログエントリ内で使って
ブログデータの読み込み中で
あることをユーザーに知らせます。

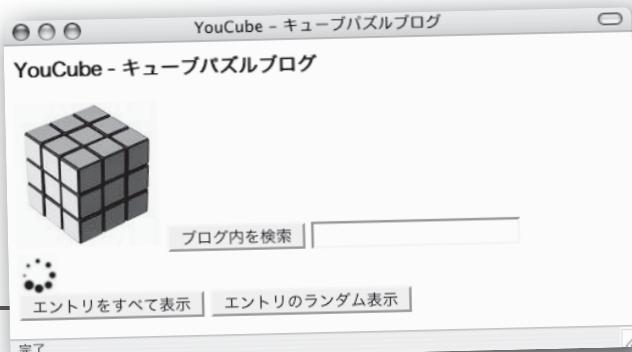


loadBlog()のコードの一部が消えてしまいました。この関数はブログデータの読み込み中にwait.gifという画像を表示します。

ヒント：「blog」というIDでメインのブログのdivにアクセスします。

画像をいくつかの属性をもたせて
追加するので、この場合innerHTMLは
DOMよりも使いやすいです。

ブログデータの読み込み中、
ブログコンテンツを“wait”
画像で完全に置き換えます。



Q: YouCubeの最後のプロ
グエントリにはHTMLの
タグが含まれています。
XMLコードでは実現可能ですか？

A: XMLコードを使うといろいろな種類のデータを表
現することができます。この場合、ブログエントリの
本体がウェブページに投入されることがわかれれば、HTML
タグを挿入してページの外見を変えることは技術的に可能で
す。つまり特定のブログエントリの本文コンテンツにHTML
タグを含めることができます。このタグはXML
コードの特殊なノードとして渡されます。しかし、XMLデータ
をページに挿入するときページのHTMLコードにおける
HTMLノードとして再構築する必要があるので、かなり手
の込んだ操作になります。YouCubeのコードでは、その方
向で対応するのではなく、単にHTMLタグからテキストコン
テンツを引き出すだけです。YouCubeの将来のバージョンで
対応できるように、ブログのコンテンツをHTMLタグを追加

素朴な疑問に 答えます

することはできますが、タグは無視
されるので本来の機能は働きません
が、テキストはそのままです。

Q: Ajaxのレスポンスの状態はどのように動作する
のでしょうか？

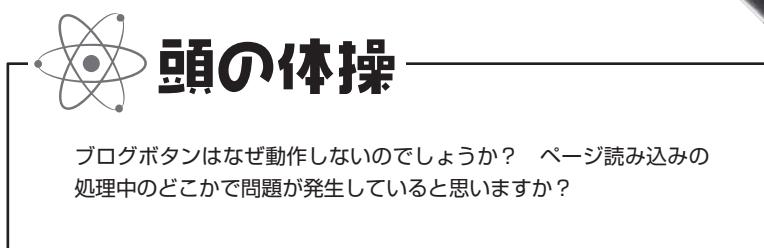
A: XMLHttpRequestオブジェクトには最終的に2
つのプロパティがあり、これでリクエストの状態を
把握しています。一方は（クライアント側の状態である）未
初期化(0)や読み込み完了(4)を示し、もう一方は（サーバ
側の状態である）404(見つからない)や200(OK)を示しま
す。これらのプロパティを詳細に追うこともできますが、そ
うする必要はありません。必要なのは、Ajaxリクエストが成
功して完了したとき、（クライアント側の状態は）4(読み込み
完了)とサーバ側の状態は200(OK)になります。このため
handleRequest()では、これらの状態が条件を満たし
ているときだけアクションを開始します。

ボタンの機能不全

YouCubeはユーザからは見えない画面の裏でAjaxに刷新されましたが、そろそろユーザインターフェースに光をあてる段階になったようです。具体的にはページのボタンが思うように動いていないようなのです。



壊れたボタン = 不満なユーザ



ルビーは落ちていますが、問題を解決する必要があります。

ボタンに必要なデータ

YouCubeボタンの問題は、ログデータが使用できるときしか使えないことです。ログデータは外部のXMLファイルから読み込まれるので、通常はほんの一瞬ですが、ページにデータがない状態になります。この間はボタンがあっても機能ないので、ユーザを混乱させる結果になりました。

ログデータが
利用可能になるまで
ボタンを無効にすることが
できますか？



ボタンを無効にすることは最善の対策です。

ログデータの読み込み中、ボタンを無効にしてしまうのは、簡単ですがエレガントな解決方法です。ログデータを読み込むAjaxリクエストはページが最初に読み込まれたときに発行されるので、ボタンを最初は無効にしておいて、Ajaxリクエストが完了したらhandleRequest()の中でボタンを有効にすることができます。ボタンを実際に無効にするには、<input>タグのdisabled属性を使う必要があります。このタグで値が"disabled"に設定されるとボタンが無効になります。逆に、ボタン要素を有効にするときは、JavaScriptでこれをfalseに設定する必要があります。

```
<input type="button" value=" ブログ内を検索 "
```

ブログ内を検索

disabled="disabled" />

ブログ内を検索

```
buttonElem.disabled = false;
```



JavaScript マグネット

マグネットを使って YouCube のページのコードを完成させてください。ブログデータの読み込みが完了するまでブログボタンを無効にします。いくつかのマグネットは何回でも使えます。

```

<html>
  <head>
    <title>YouCube - キューブパズルブログ </title>
    <script type="text/javascript" src="ajax.js"> </script>
    <script type="text/javascript" src="date.js"> </script>
    <script type="text/javascript">
      ...
      function handleRequest() {
        if (ajaxReq.getReadyState() == 4 && ajaxReq.getStatus() == 200) {
          ...
          // ブログのボタンを有効にします
          document.getElementById( ..... ). .... = ....;
          document.getElementById( ..... ). .... = ....;
          document.getElementById( ..... ). .... = ....;

          ...
        }
      ...
    </script>
  </head>

  <body onload="loadBlog();">
    <h3>YouCube - キューブパズルブログ </h3>
    
    <input type="button" id="search" value=" ブログ内を検索 "
      ..... = ..... onclick="searchBlog();"/>
    <input type="text" id="searchtext" name="searchtext" value="" />
    <div id="blog"></div>
    <input type="button" id="showall" value=" エントリをすべて表示 "
      ..... = ..... onclick="showBlog();"/>
    <input type="button" id="viewrandom" value=" エントリのランダム表示 "
      ..... = ..... onclick="randomBlog();"/>
  </body>
</html>

```

true

"search"

"viewrandom"

disabled

false

"showall"

"disabled"



JavaScript マグネットの答え

マグネットを使って YouCube のページのコードを完成させてください。ブログデータの読み込みが完了するまでブログボタンを無効にします。いくつかのマグネットは何回でも使えます。

```

<html>
  <head>
    <title>YouCube - キューブパズルブログ </title>
    <script type="text/javascript" src="ajax.js"> </script>
    <script type="text/javascript" src="date.js"> </script>

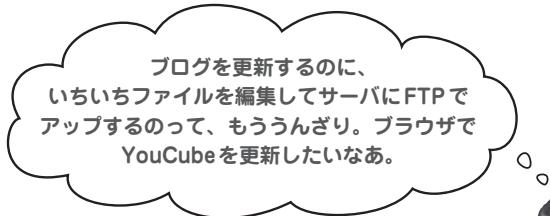
    <script type="text/javascript">
      ...
      function handleRequest() {
        if (ajaxReq.readyState() == 4 && ajaxReq.getStatus() == 200) {
          ...
          // ブログのボタンを有効にします
          document.getElementById("search").disabled = false;
          document.getElementById("showall").disabled = false;
          document.getElementById("viewrandom").disabled = false;
          ...
        }
        ...
      }
    </script>
  </head>

  <body onload="loadBlog()">
    <h3>YouCube - キューブパズルブログ </h3>
    
    <input type="button" id="search" value=" ブログ内を検索 "
          disabled = "disabled" onclick="searchBlog();"/>
    <input type="text" id="searchtext" name="searchtext" value="" />
    <div id="blog"></div>
    <input type="button" id="showall" value=" エントリをすべて表示 "
          disabled = "disabled" onclick="showBlog();"/>
    <input type="button" id="viewrandom" value=" エントリのランダム表示 "
          disabled = "disabled" onclick="randomBlog();"/>
  </body>
</html>

```

ウェブでブログを追加して時間を節約する

YouCubeは動的データで駆動していますが、ルビーはまだ完全に満足しているわけではありません。YouCubeの動的データは、ルビーがウェブベースのインターフェースを使ってブログエントリを追加できるようになって、はじめてメリットになるのです。ブログのエントリを追加するとき、XMLファイルを編集するのではなく、ウェブページで新しいエントリを入力するだけで、サーバに保存できるようにしたいと思っています。



コードを編集+ファイルをアップロード=退屈!

ルビーが描いているのは、フォームに記入するだけで、新しいブログエントリが投稿されるウェブページです。クリックするだけで、ブログの更新が完了します。必要なのはブラウザだけです。テキストエディタもFTPクライアントも要りません。パズルへの情熱があればいいのです。

YouCube - キューブパズルブログに追加

Date: 10/04/2008

Body: 私は本当に月末のこのパズルパーティを楽しみにしていました。

Image (optional):

新しいエントリを追加

完了

フィールドに記入してボタンをクリックするだけでブログエントリを新規追加できます。

ブログページの新規追加のためのフォームは3つの入力フィールドで構成されます。





頭の体操

ウェブページのインターフェースにXMLのブログエントリを追加するにはAjaxをどう使えばいいでしょう？

ブログデータを書く

Ajaxのやり方でブログの追加を考えると、AjaxのPOSTリクエストを使ってサーバに新しいブログエントリのデータを送信できるように思えます。サーバではデータを blog.xml ファイルに書き出して新しいブログエントリにします。Ajaxのレスポンスは何も返ってこないので、この場合何もする必要がありません。



新しいブログエントリをサーバにある
blog.xml ファイルに書き出すにはどうしたら
いいの？ JavaScript ではファイルの書き出しが
できないと思うけど。JavaScriptはクライアント
技術なのよね？

JavaScriptではサーバーにファイルを書き出すことはできません。

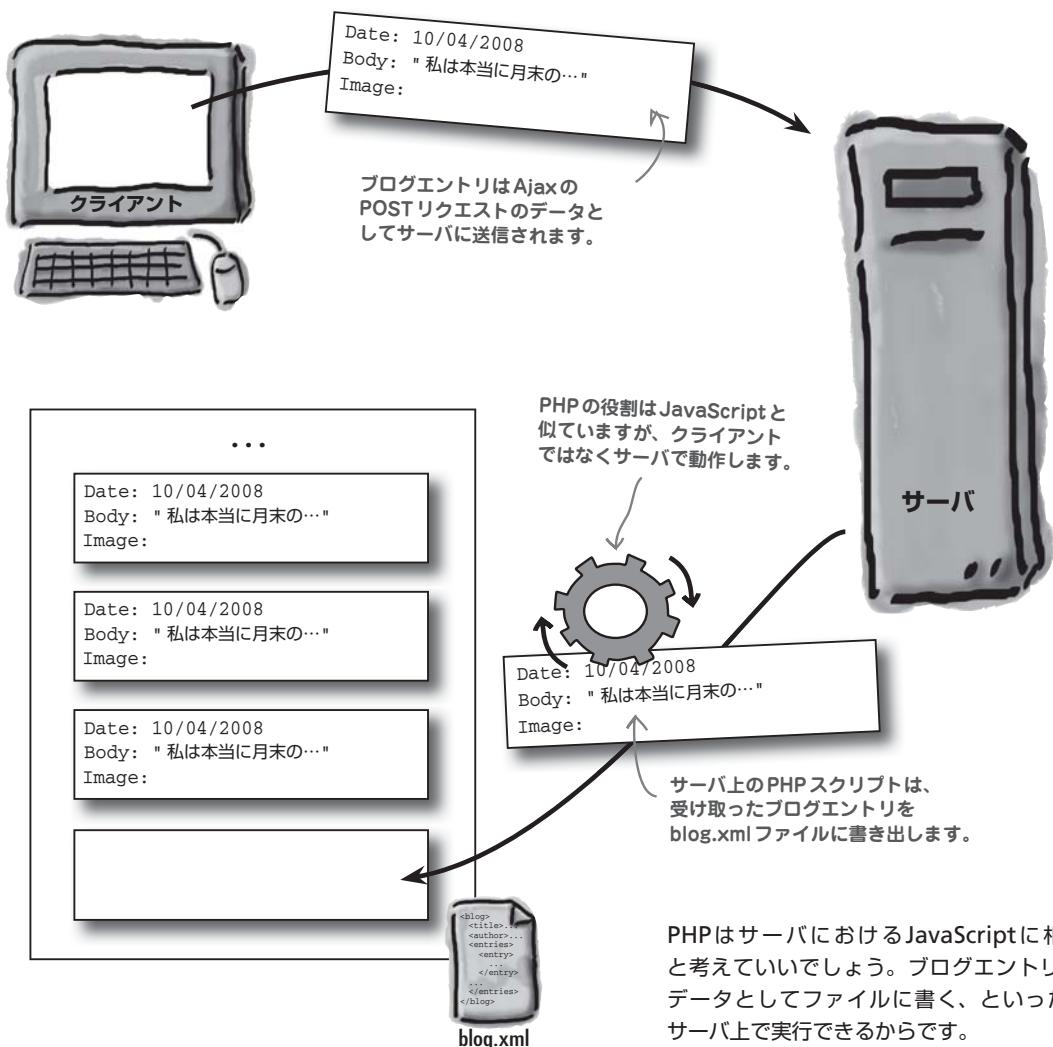
JavaScriptを使ってサーバにある blog.xml ファイルに書き出すことはできません。サーバ上では JavaScript コードを実行することすらできません。JavaScript はウェブブラウザでしか実行できないように設計されたクライアント技術なのです。サーバにファイルを書き出す必要があるので JavaScript は使えません。この問題はけして特殊ではなく、JavaScript とサーバサイドの技術が組み合わせて使われる理由はここにあります。

JavaScript に似ているけれどもサーバで実行できる技術が必要です。そうした技術はいくつかありますが、XML データを驚くほどうまく扱えてそれほど複雑でない技術がひとつあります。

今回はPHPの助けを借ります

PHPと呼ばれるスクリプト言語は、ログデータをサーバのXMLファイルに書くのに必要な機能を提供します。実際の処理は、XMLファイルを読み、新しいログエントリを既存のエントリに追加し、ログエントリをすべて元のファイルに書き戻します。新しいログエントリのデータは、Ajaxリクエストとしてクライアントブラウザから受け取ります。

PHPはサーバでのタスクを実行できるスクリプト技術です。





後は焼くだけ PHP

YouCubeのサーバサイドでPHPスクリプトを使うと、blog.xmlに格納されたXML ブログデータに新しいブログエントリを追加することができます。

XML データを変数 \$rawBlog に読み込みます。

ブログファイルが存在するか確認します。

ブログデータを XML データ構造に変換します。これは JavaScript で処理する DOMツリーに似ています。

```
<?php
$filename = "blog.xml";

if (file_exists($filename)) {
    // XML ファイルからブログエントリを読み込みます
    $rawBlog = file_get_contents($filename);
}
else {
    // 空の XML 文書を作成します
    $rawBlog = "<?xml version=\"1.0\" encoding=\"utf-8\" ?>";
    $rawBlog .= "<blog><title>YouCube - The Blog for Cube Puzzlers</title>";
    $rawBlog .= "<author>Puzzler Ruby</author><entries></entries></blog>";
}

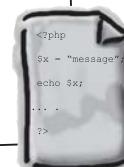
$xml = new SimpleXmlElement($rawBlog);
// 新しいブログエントリを子ノードとして追加します
$entry = $xml->entries->addChild("entry");
$entry->addChild("date", $_REQUEST["date"]);
$entry->addChild("body", stripslashes($_REQUEST["body"]));
if ($_REQUEST["image"] != "") {
    $entry->addChild("image", $_REQUEST["image"]);
}

// ブログ全体をファイルに書き出します
$file = fopen($filename, 'w');
fwrite($file, $xml->asXML());
fclose($file);
```

もしブログファイルが存在しなければ、空の XML ブログ文書を作成します。

ブログエントリを XML データ構造の中に子ノードとして新規追加します。

ブログファイルを新しいブログデータで上書きします。



addblogentry.php

このPHPスクリプトは addblogentry.php というファイルに格納します。

Q: サーバのファイルは PHP を使って書く必要がありますか?

A: その必要はありません。サーバのスクリプトを書くための技術はいろいろあります。Perl (CGI) や Java サーブレットでも、PHPと同じ処理を実現できます。これらの技術のどちらかに通じていれば、ぜひそれを使って Ajax アプリケーションのサーバ側コンポーネントを作成してください。

Q: サーバのプログラムを使わずに Ajax を使うことはできますか?

A: 例外はありますが、ほとんどの場合サーバのプログラムは多かれ少なかれサーバのスクリプトを

素朴な疑問に 答えます

必要です。最も簡単な Ajax リクエストの場合、サーバはクライアントからのデータを受け取り、そのデータを使ってデータベースから何かを探したりファイルやデータベースに書き込むといった処理を行います。YouCube のブログページはサーバのスクリプトを必要としない簡単な Ajax リクエストの例になります。Ajax アプリケーションの大部分は多かれ少なかれサーバのスクリプトを

必要とします。問題は blog.xml のようにファイル全体を送信するだけなのか、それともサーバでデータに対して何らかの処理をするのか、ということです。多くの Ajax アプリケーションで必要とされるサーバのスクリプトはかなり簡単なので、サーバのスクリプト技術に習熟していないても作成することができるでしょう。

PHPを動かすのに必要なこと

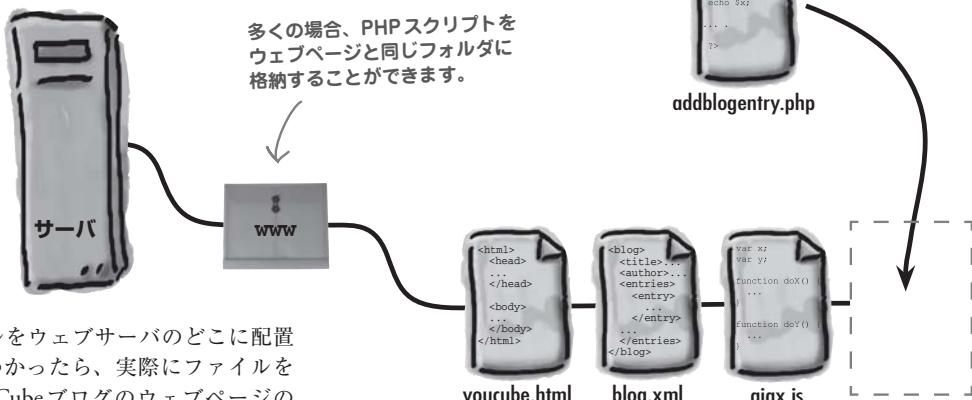
JavaScriptはブラウザでサポートされているわけですが、PHPはサーバでいつもサポートされているとは限りません。ウェブサーバにPHPファイルを配置する前に、システム管理社やウェブホスティングサービスにPHPがサポートされているか尋ねた方がいいでしょう。サポートされていなかつたら、自分でPHPを導入するか、別のウェブサーバを探すことになるかもしれません。YouCubeのPHPスクリプトは、サーバでPHPがサポートされていないと、まったく動きません。

PHPスクリプトを実行するにはウェブサーバでいくつか調整が必要になるかもしれません。



PHPがサポートされていなければ、自分でインストールするかサーバ管理者にインストールしてもらいます。Ajaxを実現するためには必須なのです。

ウェブサーバでPHPがサポートされているかどうかが第一のハードルです。第二のハードルは、PHPファイルをサーバのどこに配置すればいいか調べることです。多くの場合、HTMLのウェブページやJavaScriptファイルがあるのと同じフォルダにPHPファイルを置くことができます。しかし、PHPのインストールのされ方によっては、PHPスクリプトを特別なフォルダに配置する必要があります。これに關しても、システム管理者に尋ねてください。



PHPスクリプトにデータを送り込む

PHPがサーバで動かせて、PHPスクリプトファイルも配置できたら、サーバ上でXMLファイルにデータを書き出すためにPHPスクリプトで何をする必要があるのか、詳しく調べることにしましょう。サーバで実行する必要のある作業がわかれれば、Ajaxリクエストを設計するのに役立ちます。

PHPスクリプトが想定している新しいブログエントリのデータは、少なくとも2つの情報断片、最大で3つの情報で構成されます。

データはAjaxリクエストを介してPHPスクリプトに渡されます。

Date

ブログエントリの日付。

Body

ブログエントリの本文テキスト。

Image

ブログエントリのオプション
画像。

Date: 10/04/2008
Body: "私は本当に月末の…"
Image:



この情報はひとつにまとめて、Ajaxリクエストとしてサーバに送信される必要があります。
サーバでは、これが処理されて、blog.xmlファイルに保存されます。

この時点で新しいブログエントリはblog.xmlに追加されているので、
ブログページが読み込まれたり更新されたとき、YouCubeブログに自動的に表示されます。



```
<entry>
<date>10/04/2008</date>
<body>私は本当に月末の…</body>
<image></image>
</entry>
```



PHPスクリプトはブログエントリをXMLに変換し、blog.xmlファイルに保存します。

ブログウェブページの設計を考えると、まず新しいブログエントリを入力するためのユーザインターフェースを提示し、次に情報を集めてAjaxリクエストでサーバに送り込みます。後は新しいブログエントリが問題なく保存されたか確認すれば十分です。リクエストに対するレスポンスでこれ以外に何か必要な処理はありません。



自分で考えてみよう

YouCubeのブログエントリをウェブページに追加するための設計の概要を書いてください。Ajaxのリクエストとレスポンスがデータフローにどう位置づけられるか、正確に示してください。

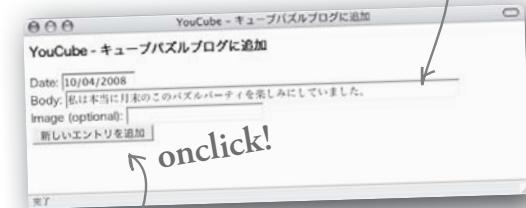
自分で考えてみよう の答え

YouCubeのブログエントリをウェブページに追加するための設計の概要を書いてください。Ajaxのリクエストとレスポンスがデータフローにどう位置づけられるか、正確に示してください。

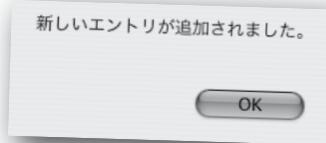
このAjaxリクエストはPOSTリクエストです。
以下のデータ断片で構成されます：

ブログ追加のウェブページには
ブログエントリの入力に使う
フォームフィールドがあります。

- * ブログ日付
- * ブログ本文
- * ブログ画像（オプション）



Add ボタンをクリックするとAjaxリクエストが送信されます。



クライアントは
ブログエントリが
追加されたことを
ユーザーに
通知します。

クライアントではサーバ
からのデータを何も必要と
していないので、このAjax
レスポンスはデータを
返しません。

ブログデータはPOST
リクエストのデータとして
サーバに送信されます。

Date: 10/04/2008
Body: "私は本当に月末の..."
Image:



サーバは新しいブログエントリを
XMLデータとして blog.xml
ファイルに書き出します。



ブログデータをサーバにポストしてみましょう

AjaxのPOSTリクエストは、サーバにデータを送信する必要があるので、GETリクエストよりすこし面倒です。POSTリクエストは、サーバに送るデータフィールドを**URLエンコーディング**の手法でまとめることもできますし、別の方法でデータをまとめることもできます。ブラウザはサーバに送るデータのフィールドをウェブページのURLで渡しますが、これと同じ手法です。データの各断片はアンパサンド (&) を使って区切りられます。

```
Date: 10/04/2008
Body: "私は本当に月末の…"
Image:
```

```
" date=10/04/2008 & body= 私は本当に月末の… &image ="
```

データの各断片は名前と
値の組で構成されます。

データの各断片は&で
区切られます。

このデータ形式では、データの各断片は名前と値を等号 (=) でつなぎます。名前 / 値の組はそれぞれアンパサンド (&) で区切れます。この形式はURLエンコードと呼ばれ、AjaxのPOSTリクエストで設定される独自のデータ型をもちます。

これはURLエンコードデータの
公式のデータタイプです。POST
リクエストの一部として指定
する必要があります。

```
"application/x-www-form-urlencoded; charset=UTF-8"
```

ブログエントリーデータをURLエンコード形式にして、POSTリクエストのデータ型にしたら、リクエストコードをまとめて、サーバにデータを送信することができます。サーバではblog.xmlファイルに保存されます。



エクササイズ

以下のデータ断片をPOSTリクエストで使えるようにURLエンコードの形式に
まとめてください。

```
releaseDate: 01/13/1989
```

```
title: Gleaming the Cube
```

```
director: Graeme Clifford
```



エクササイズ

答え

以下のデータ断片をPOSTリクエストで使えるようにURLエンコードの形式にまとめてください。

releaseDate: 01/13/1989

title: Gleaming the Cube

director: Graeme Clifford

“title=Gleaming the Cube&releaseDate=01/13/1989&director=Graeme Clifford”

Q: YouCubeのブログに追加したスクリプトがAjaxリクエストでサーバからのデータを必要としない場合、どうしてリクエストの処理に関して気にする必要があるのでしょうか？

A: リクエストが完了したことを知ることがやはり重要だからです。サーバがリクエストに応答してデータを返す必要がない場合でも、リクエストが正常に完了したか、いつ完了したかを知る必要があります。これによって、新しいブログエントリが追加されたことを知らせるアラートをいつ表示すればいいか、そのタイミングをスクリプトは知ることができます。

Q: GETリクエストはブログに追加したスクリプトで使うこともできますか？

A: はい、技術的には可能です。GETリクエストを使ってサーバにデータを送信することもできますが、その場合データをURLに直接指定する必要があります。これ自体は問題ではないのですが、サーバの状態が変わったときにGETを使うと問題になります。この場合、新しいブログエントリをblog.xmlに書き込むことで、サーバの状態が変わります。そのため、POSTリクエストが正しいアプローチであるのは、単にサーバへの通信の意図がはっきりしているからです。

素朴な疑問に 答えます

Q: サーバがAjaxリクエストを処理し、ブログエントリを保存するには時間がかかります。リクエストが完了する前に「追加」ボタンが何度もクリックされた場合、問題がありますか？

A: はい、問題があります。追加ボタンがクリックされるたびに、処理中のAjaxリクエストはキャンセルされ、新しいリクエストが発行されます。二度クリックできるようにするのもいいですが、リクエストが処理中の場合、単純にボタンをクリックできなくすれば、ユーザインターフェースはもつとすっきりするでしょう。なので、新しくブログエントリに追加するコードは、Ajaxリクエストの実行中は追加ボタンを無効にし、リクエストが完了したらボタンを再び有効にするといいでしよう。このようにJavaScriptアプリケーションのユーザインターフェースを少し調整するだけで、格段に直感的で使いやすくなり、ユーザを満足させることになります。

Q: URL文字列に組み込まれるブログデータに空白がある場合、どうなりますか？ URLに空白が含

まれていると、問題が起きるようになります。

A: この場合、空白があっても問題になりません。Ajaxでは送信するデータを自動的に問題が起きない形式に変換するからです。

Q: ブログでは画像はオプション扱いですが、新しいブログエントリを追加するとき、サーバに常に画像を渡す必要がありますか？

A: いいえ、渡す必要はありません。URL文字列において等号(=)の後に値がなければ、空のデータ断片が送信されるので、何も問題ありません。

“date=...&body=...&image=”

この例では、画像データフィールドは、実際にはデータが含まれていませんが、サーバに送信されます。サーバでPHPは、画像フィールドが空であることがわかるので、新しいブログエントリには画像は含まれません。





自分で考えてみよう

YouCube ブログにエントリを追加するスクリプトにある `addBlogEntry()` と `handleRequest()` の消えたコードを書いて関数を完成させてください。

```
function addBlogEntry() {  
    // 追加ボタンを無効にし、状態をビジーに設定します  
  
    .....  
  
    // 新しいブログエントリデータを Ajax リクエストとして送信します  
  
    ajaxReq.send("POST", "addblogentry.php", handleRequest,  
        "application/x-www-form-urlencoded; charset=UTF-8",  
  
        .....  
  
        );  
}  
  
function handleRequest() {  
    if (ajaxReq.readyState == 4 && ajaxReq.status == 200) {  
        // 追加ボタンを有効にして状態をクリアします  
  
        .....  
  
        // ブログエントリが追加されたことを通知します  
  
        alert("新しいエントリが追加されました。");  
    }  
}
```



自分で考えてみよう の答え

YouCube ブログにエントリを追加するスクリプトにある addBlogEntry() と handleRequest() の消えたコードを書いて関数を完成させてください。

ブログエントリのサーバへの
保存が完了するまで、
追加ボタンを無効にします。

```
function addBlogEntry() {
    // 追加ボタンを無効にし、状態をビジーに設定します
```

```
→ document.getElementById("add").disabled = true;
```

```
document.getElementById("status").innerHTML = "追加中…";
```

ページの状態エリアには
"busy" というメッセージが
表示されるので、ユーザは
処理中だと分かります。

これは POST リクエスト // 新しいブログエントリデータを Ajax リクエストとして送信します

リクエスト ajaxReq.send("POST", "addblogentry.php", handleRequest,

です。 "application/x-www-form-urlencoded; charset=UTF-8",

サーバで PHP
スクリプトを使って
ブログエントリを
処理し、サーバの
ブログファイルに
保存します。

```
"date=" + document.getElementById("date").value +
```

```
"&body=" + document.getElementById("body").value +
```

```
"&image=" + document.getElementById("image").value
```

```
)
```

日付、本文、画像のフォームフィールドから
リクエストの POST データを組み立てます。

```
function handleRequest() {
```

```
if (ajaxReq.readyState == 4 && ajaxReq.status == 200) {
```

// Add ボタンを有効にして状態をクリアします

```
document.getElementById("add").disabled = false;
```

ブログの保存
リクエストが
無事完了しました。

```
document.getElementById("status").innerHTML = "";
```

// ブログエントリが追加されたことを通知します

```
alert("新しいエントリが追加されました。");
```

```
}
```

```
}
```

ブログエントリの
保存が完了したので、
追加ボタンを有効にして
状態エリアをクリア
します。

ブログの操作が簡単になった

ブログ操作が一変したのでルビーは驚きを隠せません。ブログを更新するために、もうファイルを開いてコードを編集してファイルをサーバにアップロードする作業がなくなりました。彼女のブログがまさにデータ駆動になっただけでなく、新しいブログデータを書く意欲もわいてきました。

ブログエントリの追加が成功したことが通知されます。

新しいブログエントリが追加中であることをユーザに知らせます。

ルビーは新しいブログエントリをブログ追加のページで入力します。

新しいブログエントリは YouCube ブログに表示されます。

新しいエントリが追加されました。

OK

動的な
ブログデータって
すごいわよ！

YouCube をもっと使いやすくする

細かいところに神経が行き届かないとキューブパズルの黒帯三段にはなれません。ルビーがブログページを完璧にしたいと思うのは、それほど驚くべきことでもありません。Ajax アプリケーションでは使いやすさに関連する細部の作り込みが重要なことをルビーは学びました。そこで彼女は新しいページを現代のウェブページと同じくらいに使いやすくしたいと思っています。

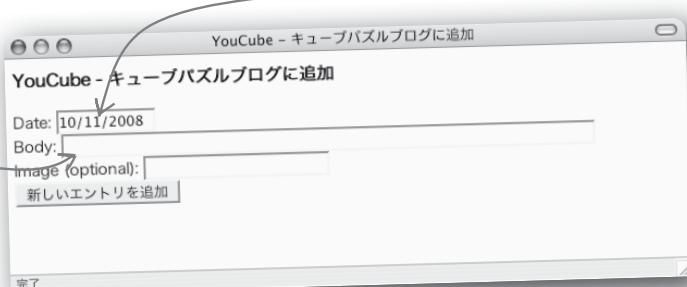
ブログの入力をもっと
効率的にして、もっと簡単に、
なるだけ頻繁に投稿できる
ようにしたいなあ。



YouCube ブログデータの入力効率を 最大にする

今までかなり多くのブログエントリを作っていましたが、ルビーはブログの日付が自動的に今日の日付で入力されたらキー入力の手間を減らせるに気づきました。それに、ブログエントリの日付が今日の日付になれば、フォームの本文フィールドに集中でき、ブログの入力をすぐに終わらせることができます。いまのブログにこうした変更を加えることは、そんなに重要なことではありませんが、ページの「感触」は劇的に変わります。これこそAjaxらしいところです。ブログに常に新着情報があるように、定期的に投稿する上でも役に立ちます。

今日の日付をフィールドに
自動入力します。



YouCube - キューブパズルブログに追加

Date: 10/11/2008

Body:

Image (optional):

新しいエントリを追加

完了

ここに入力フォーカスを設定すれば、
ルビーはすぐにブログテキストの
入力を始められます。

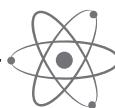
自動入力フィールドにする

YouCube ブログの日付は MM/DD/YYYY の形式でした。そのため、自動入力のフォームフィールドに設定する現在の日付も同じ形式にする必要があります。現在の日付の形式を MM/DD/YYYY にするコードが必要です。

Date メソッドはすでに作ったけれど、
YouCube ブログのメインページに格納されて
いるよね。これをブログ追加のページで
共有する方法はないかな？

**コードの重複をなくすには、共通するコードを
共有するのが一番です。**

YouCube のあちこちに重複したコードがあると、維持管理するのに大変ですから、それを避けるためにも、2 つのページで日付形式のコードを共有した方がいいでしょう。コードを一ヵ所に格納して、そのコードを必要とするページで共有できるようにするのです。



頭の体操

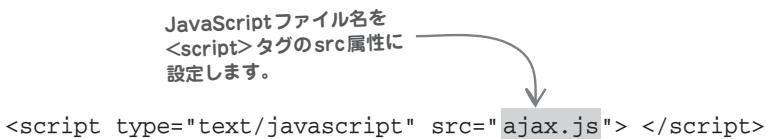
shortFormat() のコードを YouCube のページで共有するにはどうしたらいいでしょう？



タスクが重複していたら、関数にしてみては？

複数のウェブページでJavaScriptのコードを共有するには、そのコードを別ファイルに分離して、それぞれのページがそのファイルを取り込むようにします。このやり方はすでに経験済みです。AjaxRequestオブジェクトをajax.jsに格納して、以下のコードでこれを取り込んでいます。

JavaScriptファイル名を
<script>タグのsrc属性に
設定します。

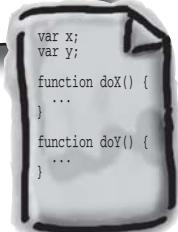


```
<script type="text/javascript" src="ajax.js"> </script>
```

おなじみの<script>
タグは外部ファイルに
格納されたJavaScript
コードを取り込むときに
使います。

`shortFormat()`はDateオブジェクトのメソッドです。`date.js`というファイルにこれを格納し、YouCubeウェブページにこれを取り込めば、目標達成です。

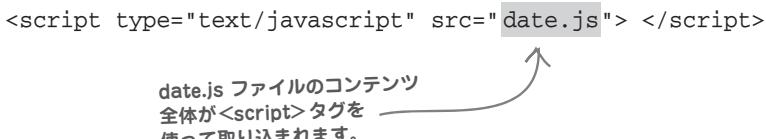
```
Date.prototype.shortFormat = function() {
    return (this.getMonth() + 1) + "/" + this.getDate() + "/" + this.getFullYear();
}
```



Ajaxコードのために<script>タグを使ったのと同じように、`date.js`に格納したコードをYouCubeのページに取り込むことができます。

<script type="text/javascript" src="date.js"> </script>

date.js ファイルのコンテンツ
全体が<script>タグを
使って取り込まれます。



JavaScriptコードを
外部ファイルに格納した
ので、2カ所で共有されて
います。

再利用可能なJavaScriptコードを別ファイルに分離して、複数のページで共有できるようにするのは、悪くない発想です。



自分で考えてみよう

`initForm()`という関数のコードを書いてみましょう。この関数はYouCubeのスクリプトの `onload`イベントハンドラーで呼び出されます。この関数は日付フィールドを現在の日付で初期化し、本文フィールドに入力フォーカスをあてる必要があります。

.....
.....
.....

自分で考えてみよう の答え

日付フィールドは今日の
日付に設定されます。

initForm() という関数のコードを書いてみましょう。この関数は YouCube のスクリプトの onload イベントハンドラで呼び出されます。この関数は日付
フィールドを現在の日付で初期化し、本文フィールドに入力フォーカスをあて
る必要があります。

```
function initForm() {  
    document.getElementById("date").value = (new Date()).shortFormat();  
    document.getElementById("body").focus();  
}
```

本文フィールドに
入力フォーカスを
設定します。

ブログの生産性が急上昇

ルビーは YouCube ブログに超満足しています。Ajax のおかげで、データ駆動
のユーザフレンドリなブログになったからです。細部への惜しみない改善方法
は、彼女がパズルのマスターだからこそマスターできたのでしょう。



The screenshot shows a web browser window for "YouCube - キューブパズルブログに追加". The form fields are filled with "Date: 10/11/2008", "Body: ふう、ようやくブログスクリプトが完成！", and "Image (optional):". A button labeled "新しいエントリを追加" is visible. Below the form, the blog interface displays a post titled "YouCube - キューブパズル" with the date "完了" (Completed). The post content reads: "10/11/2008 ふう、ようやくブログスクリプトが完了！ by Puzzler Ruby". Another post below it says: "10/4/2008 私は本当に月末のこのパズルパーティを楽しみにしていました。 by Puzzler Ruby". The bottom post is dated "9/26/2008" and includes a small image of a Rubik's cube.

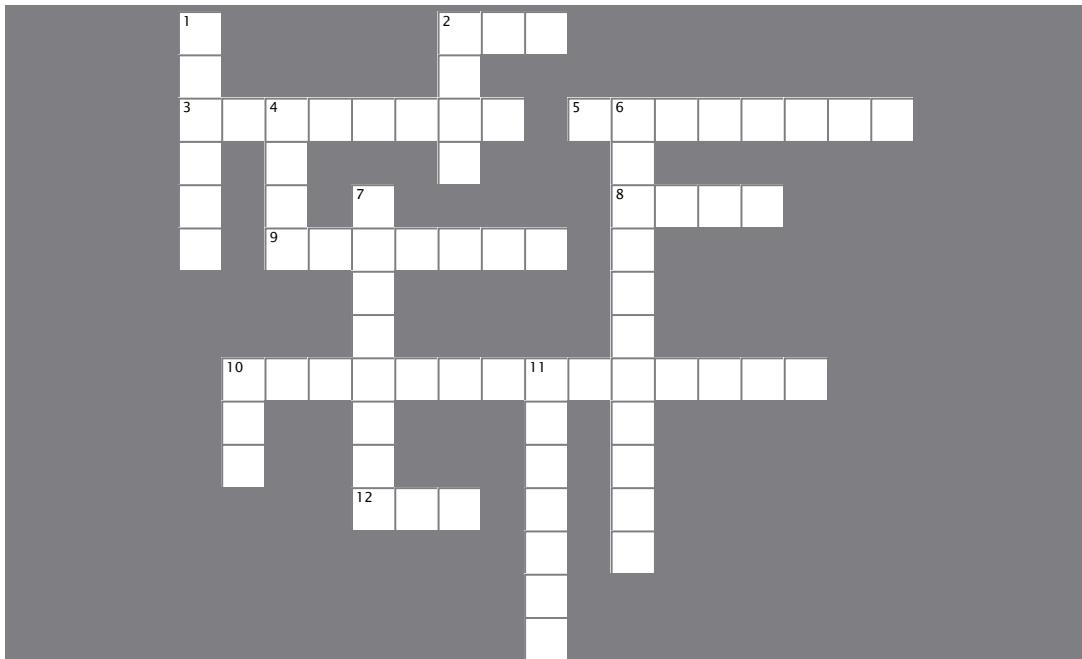
ページが最初に
読み込まれたとき、
入力フォーカスが本文に
設定されます。

ページが開かれたとき、
日付は自動的に今日の
日付で設定されます。



JavaScript クロスワード

動きがあるっていいですよね。動きのあるデータを扱ってみませんか？
クロスワードのデータって動きがありますよ。



ヨコのカギ

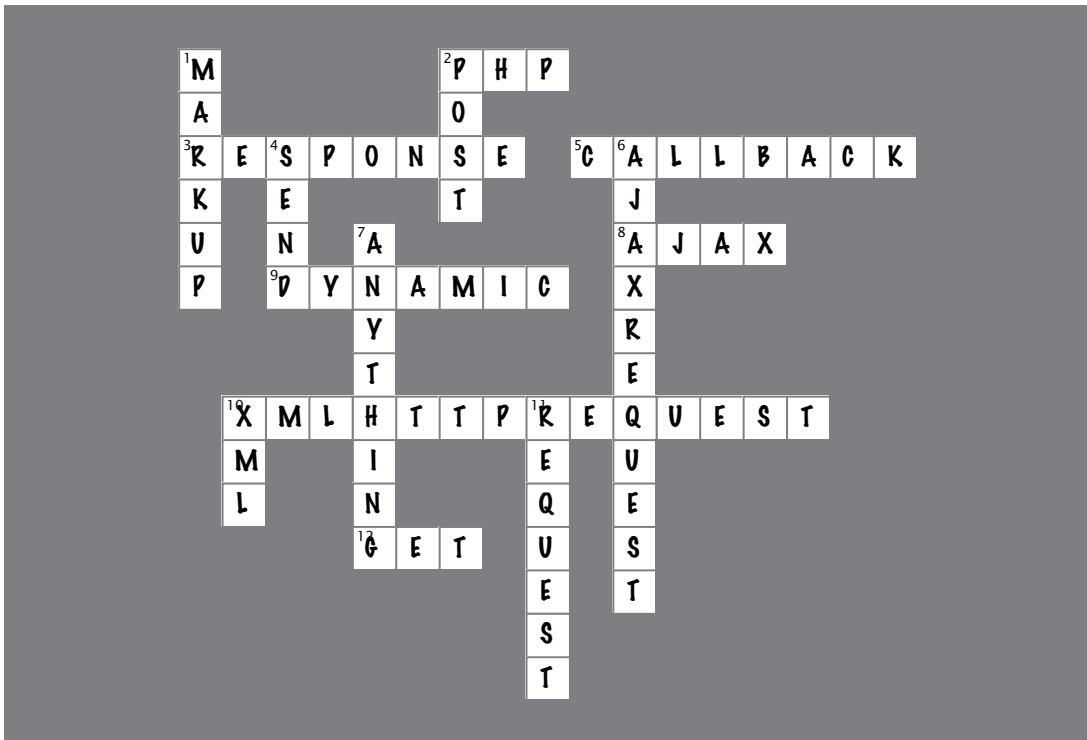
2. AjaxアプリケーションでJavaScriptを補完するサーバスクリプトの技術。
 3. Ajaxリクエストに対するサーバの応答。
 4. Ajaxリクエストが官僚したときに呼び出される関数の種類。
 5. この技術を使うとウェブページの応答性がよくなります。
 6. ウェブページの情報をより充実させるデータの種類。
 7. Ajaxの機能をサポートするために使われるJavaScriptの標準オブジェクト。
 8. このリクエストタイプはサーバからデータを要求するだけです。

タテのカギ

1. HTMLのMLは 言語の略です。
 2. このリクエストタイプはサーバでの状態が変わります。
 4. リクエストの発行に使うAjaxRequestオブジェクトのメソッドです。
 6. Ajaxの使い方を簡単にするために作られたカスタムオブジェクト。
 7. XMLのXは何の略。
 10. これが<blog>、<author>、<entry>を実現します。
 11. Ajaxアプリケーションがサーバにデータを依頼すること。



JavaScript クロスワードの答え



折り畳みページ

脳のところで
折り畳んでから
謎を解決しましょう。

ルビーはAjaxから
何を受け取ったのでしょうか？

左脳と右脳がご対面！



ルビーはAjaxから数えきれないくらいたくさんのことを受け取っています。そのひとつが動的データです。これによってキューブのブログはとても簡単になりました。ルビーは簡単にキューブパズルの話ができるようになりました。

これからどこに行きましょう？

『Head First JavaScript』はこれで終わりですが、JavaScriptを使ってインタラクティブなユーザ体験を創造する旅はこの先も続きます。次はどこに行ったらいいでしょう？ ウィルドなワールドワイドウェブのアプリケーションを開発する次のステップで、あなたが関心をもつと思われる参考情報を紹介しておきます。

Head First JavaScript フォーラムにアクセス

式でイライラしたり、演算子で困っていませんか？ あるいはJavaScriptで作った最新スクリプトをHeadFirstコミュニティで共有することに関心ありませんか？ Head First Labs (<http://www.headfirstlabs.com>) にあるHead First JavaScript フォーラムにしばらく立ち寄って、議論に参加してみませんか？

他の本も読んでみよう



基本は習得できたので、より高度なJavaScriptの詳細を深く掘り下げましょう。

『JavaScript 第5版』
『JavaScript クイックリファレンス 第5版』
『JavaScript & DHTML クックブック 第2版』



他のサイトで学んでみよう

Quirksmode

<http://www.quirksmode.org>

残念ながら、ブラウザによってJavaScriptの内容が異なることがあります。ブラウザによるJavaScriptの違いをまとめてあるサイトがQuirksmodeです。



Mozilla JavaScript リファランス
<http://developer.mozilla.org/en/docs/JavaScript>

しばらくするとJavaScript組み込みオブジェクトについて調べる必要に迫られるでしょう。MozillaのオンラインリファランスではJavaScriptが隅々まで解説されています。



Prototype JavaScript フレームワーク

<http://www.prototypejs.org>

サードパーティの再利用可能なライブラリコードを使ってJavaScriptをまったく新しいレベルにしたいと思いませんか？ Prototypeは最も優れたフレーバーのひとつです。



索引

記号、数字

- ! (否定演算子) 164, 165, 218
- != (不等演算子) 163
- && (AND演算子) 217
- || (OR演算子) 218
- (デクリメント演算子) 202
- ++ (インクリメント演算子) 202
- == (等しい演算子) 163
 - =との比較 164
 - =と == の違い 510, 532
- < (より小さい演算子) 164
- <= (以下演算子) 164
- > (より大きい演算子) 164
- >= (以上演算子) 164
- 1回限りのタイマー 93
- 2次元 Mandango の詳細 236-237
- 2次元配列 231-234
 - 2次元 Mandango 詳細 236-237
 - データアクセス 233

A

- Ajax 541
 - Ajax レスポンスの状態 572
 - Ajax レスポンスの処理 568
 - GETリクエスト 554-555
 - handleRequest() 565
 - PHP 579-584
 - スクリプトの実行 581-582
 - POSTリクエスト 554-555, 585
 - URLエンコーディング 585
 - XML 549
 - XML と Ajax の関係 548
 - YouCube プロジェクト 550-551

B

- Bannerocity プロジェクト 290-292
- 3桁の年を除外 330-331
- placeOrder() 関数 313-314
- 空でないかの検証 300-303
- データ検証 302
 - 電話番号 333
 - 日付 315-316
 - メールアドレス 334-337
 - 郵便番号 310
- データの長さの検証 305-306
- Blog オブジェクト 404, 406, 408, 412-413
 - containsText() メソッド 441-442, 450
 - signature プロパティ 460-465
 - this キーワード 451

| | |
|------------------|---------------|
| toHTML() メソッド | 450, 456, 479 |
| toString() メソッド | 450 |
| アクション | 441 |
| オブジェクトの拡張 | 467-470 |
| 画像の追加 | 476-479 |
| 更新 | 443-444 |
| ソート | 472-473 |
| 追加メソッド | 439-441 |
| メソッドのオーバーロード | 451 |
| blogSorter() クラス | 473-474 |
| break 文 | 178, 179, 212 |

C

| | |
|---------------------------|---------------|
| callNum 変数 | |
| NaN で表示 | 526-527 |
| ウォッチ | 515 |
| CamelCase | 52, 62 |
| changeScene() 関数 | 157, 162, 367 |
| checkWinner() 関数 | 504-505 |
| className プロパティ | 379 |
| clearInterval() 関数 | 97 |
| compare() 関数 | 421 |
| const キーワード | 46 |
| continue 文 | 212 |
| cookieEnabled プロパティ | 126 |
| createElement() メソッド | 383, 385 |
| createTextNode() メソッド | 361, 371 |
| CSS | 6-7 |
| DOM と CSS でスタイル変更 | 372-373 |
| HTML の分離 | 270-271 |
| インターラクティブ | 8 |
| CSS スタイルクラス | 375 |
| JavaScript クラスとの比較 | 372, 374 |
| width プロパティと height プロパティ | 105 |
| 個々のスタイル変更 | 379 |

D

| | |
|--------------------|------------------|
| Date オブジェクト | 97, 410-411, 414 |
| getDate() メソッド | 417 |
| getFullYear() メソッド | 417 |
| getMonth() メソッド | 417 |

| | |
|----------------------------|----------|
| shortFormat() メソッド | 592 |
| toString() メソッド | 416 |
| オブジェクトのテキストへの変換 | 415-417 |
| ソート用の比較 | 423 |
| データの断片にアクセス | 417 |
| <div> タグ | 346, 362 |
| div 要素と span 要素の比較 | 371 |
| document.body.clientHeight | 100 |
| document.body.clientWidth | 100 |
| DOM | 352-392 |
| appendChild() メソッド | 361 |
| createElement() メソッド | 383 |
| createTextNode() メソッド | 361 |
| DOM ツリー | 357 |
| DOM と CSS でスタイル変更 | 372-373 |
| DOM ノードの分類 | 354 |
| DOM の積み木 | 362 |
| element 属性 | 354 |
| ELEMENT ノード | 354, 362 |
| removeChild() メソッド | 361 |
| TEXT ノード | 354, 362 |
| XML | 549 |
| XML データのノードからコンテンツを引き出す | 566 |
| ウェブ標準 | 365 |
| 個々のスタイル変更 | 379 |
| 子ノード | 353, 363 |
| トップノード | 354 |
| ノード | 353 |
| style プロパティ | 377 |
| visibility スタイルプロパティ | 377 |
| 型 | 354 |
| プロパティ | 357 |
| ノードテキストの置き換え | 368-369 |
| ノードの階層木 | 353 |
| ノードのテキスト変更 | 360-361 |
| プロパティ | 359 |
| DOM の積み木 | 362 |

E

| | |
|-------------|----------|
| ELEMENT ノード | 354, 362 |
|-------------|----------|

F

- findSeat() 関数 206, 211, 258
 Firefox のデバッグ 489-491
 floor() メソッド 436, 437
 for ループ 192-193, 226-228
 Mandango 197
 実行しない 227
 無限ループ 201

G

- getElementById() メソッド 71, 76, 101, 102, 347,
 352, 358, 375
 getElementsByTagName() メソッド 358
 getSeatStatus() 関数 267-268
 getTemp() 関数 262
 getText() 関数の詳細 567
 GET リクエスト 554-555, 561
 greetUser() 関数 121

H

- heat() 関数 251, 263
 HTML 6-7
 size 属性 309
 XHTML 545
 XML との比較 542
 インターラクティブ 8
 コールバック 関数 277
 分離 270-271
 要素にアクセス 347-348, 522
 HTTP リクエスト / レスポンス 561

I

- ID (要素の) 71
 id 属性 (<div> タグの) 346
 if 文 138, 150
 入れ子 155, 158
 セミコロン 143
 if/else 文 150
 2 つのどちらかを選ぶ 141

- false 条件 143
 入れ子 175
 書式 143
 制限 147
 複数 154
 複数の else 143
 複数の決定 142
 タグ 204
 initSeats() 関数 205, 248
 innerHTML プロパティ 348, 350, 352, 364, 408
 コンテンツの設定 349
 <input> タグの disabled 属性 574
 Internet Explorer のデバッグ 488, 491
 IQ 計算スクリプト 487-497
 タイプミス 496
 波括弧の対応が取れていない 493-494
 未定義変数エラー 495-496
 iRock プロジェクト
 iRock 画像を動的に変更 108-109
 iRock の画像の大きさの計算での 100 の意味 105
 画像の大きさ変更 101-103
 クッキー 120-126
 書き出し 122-123
 メッセージ 129
 ユーザに挨拶 120-121
 利用できない 126
 表情を増やす 90-91
 ブラウザウィンドウの大きさ変更 106-107

J

- JavaScript
 HTML の分離 270-271
 インターフリタ → インターフリタを参照
 クラスと CSS スタイルクラスとの比較 372
 サーバにファイルを書き出す 578
 JavaScript コード
 開始 13
 外部ファイルからの取り込み 592-593
 外部ファイルに格納したときのデメリット 125
 行末に 1 行コメント 173
 コメント中 173
 サーバ 89

| | |
|--------------------|---------|
| 識別子 | 50 |
| 実行 | 88 |
| 重複をなくす | 254-256 |
| 定数 → 定数を参照 | |
| 波括弧の対応が取れていない | 493-494 |
| ブラウザ | 13 |
| 変数 → 変数を参照 | |
| 未定義の変数や関数を参照しようとする | 511 |

L

| | |
|--------------|---------|
| length プロパティ | 306-307 |
|--------------|---------|

M

| | |
|------------------------|-----------------------|
| Mandango プロジェクト | 194-220 |
| 1 座席を探す | 209 |
| 2 次元 Mandango 詳細 | 236-237 |
| 2 次元配列 | 231-234 |
| データアクセス | 233 |
| 3 席を探す良い方法 | 217 |
| break 文 | 212 |
| continue 文 | 212 |
| findSeat() 関数 | 206, 211 |
| findSeats() 関数 | 258 |
| getSeatStatus() 関数 | 267-268 |
| initSeats() 関数 | 205, 248 |
| JavaScript から HTML を操作 | 203-206 |
| onload イベントハンドラ | 283 |
| selSeat 変数 | 206, 211 |
| setSeat() 関数 | 257-258 |
| showSeatStatus() 関数 | 268-269, 272-273, 279 |
| コード詳細 | 215 |
| 座席の画像 | 204 |
| 座席の初期化 | 205 |
| 配列を座席の画像に対応付ける | 204 |
| 無限ループ | 210 |
| Math オブジェクト | 434, 437 |
| Math オブジェクトの真実 | 435 |

N

| | |
|--------------------|--------|
| NaN (Not a Number) | 62, 69 |
|--------------------|--------|

| | |
|------------------|----------|
| navigator オブジェクト | 126 |
| new 演算子 | 405, 413 |
| NOT 演算子 | 219 |
| null | 165 |
| 見つからなかった引数 | 477 |
| 未定義との比較 | 497 |

O

| | |
|-----------------------|----------|
| onblur イベント | 295-297 |
| validateNonEmpty() 関数 | 301 |
| 間違ったフォームデータ | 298-299 |
| onchange イベント | 295 |
| 間違ったフォームデータ | 298-299 |
| onfocus イベント | 295 |
| onload イベントハンドラ | 280, 283 |
| onmouseout イベント | 375 |
| onmouseover イベント | 375 |
| onresize イベント | 107-109 |
| Opera のデバッグ | 488 |

P

| | |
|------------------|--------------|
| parseDonuts() 関数 | 79 |
| parseFloat() 関数 | 65 |
| parseInt() 関数 | 65, 77 |
| 小数 | 76 |
| PHP | 579-584 |
| スクリプトの実行 | 581-582 |
| placeOrder() 関数 | 313-314 |
| 詳細 | 314 |
| POST リクエスト | 554-555, 561 |

R

| | |
|----------------------|--------------|
| random() メソッド | 436 |
| reload() メソッド | 97 |
| replaceNodeText() 関数 | 369 |
| request メソッド | 554 |
| Return の真実 | 264 |
| return 文 | 262-264, 269 |
| round() メソッド | 437 |

S

| | |
|--------------------------|-----------------------|
| Safariのデバッグ..... | 488 |
| <script> タグ..... | 13, 119, 592 |
| setInterval() 関数..... | 95, 97, 104 |
| setSeat() 関数..... | 257-258 |
| setTimeout() 関数..... | 94, 104 |
| showSeatStatus() 関数..... | 268-269, 272-273, 279 |
| sort() メソッド..... | 420-421 |
| タグ..... | 302, 362 |
| span要素とdiv要素の比較..... | 371 |
| String オブジェクト..... | 413 |
| charAt() メソッド..... | 426 |
| indexOf() メソッドとの比較..... | 432 |
| indexOf() メソッド..... | 426-427 |
| toLowerCase() メソッド..... | 426 |
| toUpperCase() メソッド..... | 426 |
| 検索..... | 427 |
| 長さ..... | 426 |
| style オブジェクト..... | 102-104 |
| switch/case文..... | 143, 177-180 |
| break文..... | 178, 179 |
| default枝..... | 178 |
| Switchの真実..... | 180 |
| 括弧..... | 178 |
| Switchの真実..... | 180 |

T

| | |
|----------------------|------------------------------|
| TEXT ノード..... | 354, 362 |
| this キーワード..... | 300, 309, 375, 403, 405, 459 |
| Blog オブジェクト..... | 451 |
| インスタンス..... | 454 |
| toString() メソッド..... | 416, 423 |
| touchRock() 関数..... | 122-123 |

U

| | |
|-------------------|-----|
| undefined | |
| definedとの違い | 497 |
| nullとの違い | 497 |
| URLエンコーディング | 585 |
| 空白..... | 586 |

V

| | |
|-----------------------------|--------------|
| validateDate() 関数 | 329 |
| validateLength() 関数 | 306 |
| validateNonEmpty() 関数 | 300-303, 329 |
| validateRegEx() 関数 | 326-327 |
| var キーワード | 44 |

W

| | |
|-----------------|---------|
| while ループ | 222-229 |
| 実行しない..... | 227 |

X

| | |
|------------------------------|--------------|
| XHTML..... | 545-546 |
| XML | |
| Ajax..... | 549 |
| DOM..... | 549 |
| HTMLとの比較..... | 542 |
| HTMLの タグ | 572 |
| XHTML..... | 545 |
| XMLデータのノードからコンテンツを引き出す | 566 |
| XML と Ajax の関係 | 548 |
| カスタムタグ..... | 543 |
| ログデータ | 547 |
| .xml ファイル拡張子..... | 544 |
| XMLHttpRequest オブジェクト | 552-556, 560 |
| abort() | 552 |
| GETかPOSTか | 555 |
| onreadystatechange | 552 |
| open() | 552 |
| readyState | 552 |
| responseText | 552 |
| responseXML | 552 |
| send() | 552 |
| 状態 | 552 |
| 複雑 | 553 |
| 楽に使えるようにする | 556 |

Y

| | |
|------------------|-----|
| YouCube 詳細 | 407 |
|------------------|-----|

| | |
|------------------------------------|---------|
| YouCube プロジェクト | 399-448 |
| Ajax | 541-571 |
| Ajax の導入 | 550-551 |
| Ajax リクエスト | 563-564 |
| Blog オブジェクト → Blog オブジェクトを参照 | |
| Blog クラスに signature プロパティを追加 | 460-465 |
| Date オブジェクト | 410-411 |
| shortFormat() メソッド | 592 |
| データの断片にアクセス | 417 |
| handleRequest() 関数 | 569-570 |
| HTML の タグ | 572 |
| innerHTML | 408 |
| JavaScript コードのインポート | 592-593 |
| PHP | 579-584 |
| スクリプトの実行 | 581-582 |
| URL エンコーディング | 585 |
| 空白 | 586 |
| XML データのノードからコンテンツを引き出す | 566 |
| 新しいブログエントリの追加 | 585 |
| 画像 | 586 |
| エントリをすべて表示ボタン | 408 |
| オブジェクト | |
| オブジェクトの中にある | 413 |
| 拡張 | 467-470 |
| カスタム | 400-401 |
| コンストラクタ | 402-403 |
| テキストへ変換 | 415-417 |
| 画像の追加 | 476-479 |
| 関数をメソッドに変える | 441-442 |
| クラスが持つメソッド | 455-458 |
| クラスメソッドの呼び出し | 474 |
| 検索機能 | 424-432 |
| 自動入力 | 590-591 |
| ソート | 472-473 |
| 追加ボタン | 586 |
| 動的なデータ | 538-539 |
| 日付の整形 | 593 |
| ブラウザからのブログ更新 | 577-584 |
| ブログエントリの順番 | 408-410 |
| ブログ追加スクリプト | 586 |
| ブログデータ | 547-548 |
| XML | 547 |
| ソート | 418-423 |

| | |
|--------------|---------|
| ボタン | 573-576 |
| 無効にする | 574-576 |
| ランダム表示 | 433-438 |

あ行

| | |
|----------------------------|-----------------|
| アクション | 6-7 |
| アポストロフィー | |
| エスケープ文字 | 511 |
| デバッグ | 502, 532 |
| アラートでデバッグ | 509 |
| アラートボックス | 297, 314 |
| 空の | 162 |
| 解決方法 | 163 |
| 検証 | 309 |
| デバッグ | 507-509 |
| 問題 | 515 |
| 問題 | 345 |
| 一意の識別子 | 62 |
| 一時的にコードを無効にする | 527-529, 531 |
| イベント | |
| onresize | 107-109 |
| 関数 | 272 |
| 関数リテラルを通じて呼び出す | 279-280 |
| → コールバック関数も参照 | |
| イベントハンドラー | 278 |
| 入れ子になった if 文 | 155, 158 |
| 入れ子になった if/else 文 | 175 |
| 入れ子のループ | 227 |
| インスタンス | |
| this キーワード | 454 |
| クラスとの比較 | 452-453, 459 |
| クラスプロパティとインスタンスプロパティ | |
| との比較 | 464, 466 |
| プロパティとメソッド | 454-455, 463 |
| インターバルタイマー | 93, 95, 97 |
| 停止 | 97 |
| インターフェリタ | 86, 89, 493-494 |
| インタラクティブ | |
| HTML と CSS | 8 |
| JavaScript を使わない | 13 |
| インデックス配列 | 199, 201, 202 |
| ウイルス | 124 |

| | |
|----------------------------------|------------------------|
| ウェブクライアント → クライアントを参照 | |
| ウェブスクリプティング..... | 244 |
| ウェブ標準..... | 365 |
| ウェブページ | |
| インタラクティブ性..... | 86-87 |
| 再読み込み..... | 111 |
| データフォームをつかむ方法..... | 71 |
| 要素とその値の違い..... | 76 |
| 永続性..... | 112-115 |
| クライアントとサーバの比較..... | 116 |
| エスケープ文字..... | 503, 511, 532 |
| 制約..... | 511 |
| エラー | |
| 構文 → 構文エラーを参照 | |
| 実行時..... | 524-525 |
| → デバッグも参照 | |
| 論理 → 論理エラーを参照 | |
| エラーコンソール..... | 491 |
| 変数ウォッチ..... | 516 |
| 大文字..... | 52, 62, 69 |
| オブジェクト..... | 459 |
| キーワード..... | 46 |
| オブジェクト..... | 395 |
| Blog オブジェクト | 404, 406, 408, 412-413 |
| Date オブジェクト → Date オブジェクトを参照 | |
| Math オブジェクト..... | 434 |
| new 演算子..... | 405, 413 |
| prototype..... | 456-457 |
| String オブジェクト → String オブジェクトを参照 | |
| toString() メソッド..... | 416 |
| 大文字..... | 459 |
| オブジェクトの中にある..... | 413 |
| 拡張..... | 467-470 |
| カスタム..... | 400-401, 450-484 |
| 検索可能..... | 426-427 |
| コンストラクタ..... | 402-403, 405 |
| データ型..... | 398 |
| テキストへの変換..... | 415-417 |
| ドット演算子..... | 396-397 |
| 配列としての..... | 420 |
| プロパティとメソッド..... | 396, 398 |
| 文字列としての..... | 432 |
| オブジェクト指向..... | 461 |

| | |
|------------------------|-----------------------------|
| オブジェクト指向スクリプト..... | 399 |
| オブジェクト指向設計 (OOP) | 460, 461, 465, 467, 470-475 |
| オブジェクトリテラル..... | 413 |

か行

| | |
|------------------------|-------------|
| 階層木..... | 353 |
| 外部からのスクリプトの取り込み..... | 119 |
| 格納 | |
| 定数 → 定数を参照 | |
| 変数 → 変数を参照 | |
| 影の変数..... | 530-532 |
| カスタムオブジェクト..... | 450-484 |
| 画像の大きさ変更..... | 99-109 |
| 動的..... | 108-109 |
| 括弧 | |
| switch/case 文..... | 178 |
| 対応がとれていない..... | 493-494 |
| デバッグ..... | 532 |
| 論理演算子..... | 219 |
| カプセル化..... | 472-473 |
| 空でないかの検証..... | 300-303 |
| 空のアラートボックス..... | 162 |
| 解決方法..... | 163 |
| 関数 | |
| changeScene()..... | 162 |
| clearInterval() | 97 |
| parseFloat() | 65 |
| parseInt() | 65 |
| 小数..... | 76 |
| placeOrder() | 313-314 |
| setInterval() | 95, 97, 104 |
| setTimeout() | 94, 104 |
| validateRegEx() | 326-327 |
| いつ関数にするのか..... | 249 |
| イベント..... | 272 |
| カスタム | |
| changeScene() | 157, 367 |
| checkWinner() | 504-505 |
| createTextNode() | 371 |
| findSeat() | 206, 211 |
| findSeats() | 258 |

| | | | |
|----------------------|-----------------------|-----------------------|--------------|
| getSeatStatus() | 267-268 | touchRock() 関数 | 122-123 |
| getTemp() | 262 | サーバにデータを格納 | 125 |
| greetUser() | 121 | 永続的 | 125 |
| heat() | 251, 263 | 格納場所 | 125 |
| initSeats() | 205 | 名前 | 125 |
| parseDonuts() | 79 | ブラウザ間での共有 | 125 |
| replaceNodeText() | 369 | ブラウザセキュリティ | 124 |
| setSeat() | 257-258 | ブラウザの検知 | 129 |
| showSeatStatus() | 268-269, 272-273, 279 | メッセージ | 129 |
| validateDate() | 329 | ユーザーに挨拶 | 120-121 |
| validateLength() | 306 | クライアント | 86, 89 |
| validateNonEmpty() | 300-303, 329 | ウェブクライアント、ブラウザ、クライアント | |
| コードの重複をなくす | 254-256 | ウインドウ、ブラウザウインドウの違い | 101 |
| コールバック → コールバック関数を参照 | | 永続性 | 116 |
| 参照 | 273-275, 278, 282 | サーバの相互作用 | 86 |
| 仕組み | 247 | 制御 | 89 |
| 情報を渡す | 252 | ブラウザの真実 | 98 |
| ノードテキストの置き換え | 368-369 | クライアントウインドウ | 99-101 |
| ノーマル関数とコールバック関数との違い | 276 | iRockの大きさ変更 | 102-103 |
| 場所 | 260 | 何パーセント占めるか | 103 |
| 引数 | 252 | 幅と高さ | 104 |
| 上限 | 260 | クラス | |
| フィードバック | 261-263 | インスタンスとの違い | 452-453, 459 |
| 複文との比較 | 149 | クラスプロパティとインスタンスプロパティ | |
| 変数との比較 | 274 | との比較 | 464, 466 |
| 見つからなかった引数 | 477 | クラスプロパティとグローバル変数との比較 | 464 |
| 未定義の変数や関数を参照しようとする | 511 | プロパティ | 462-463 |
| 命名規則 | 249 | メソッド格納 | 455 |
| メソッドに変える | 441-442 | クラスが所有するインスタンスマソッド | 471 |
| 目的 | 249, 260 | クラスメソッド | 471 |
| 問題解決のための | 246 | 呼び出し | 474 |
| 呼び出し | 273 | 繰り返し → ループを参照 | |
| 関数参照 | 273-275, 278, 282 | グローバルスコープ | 171 |
| 関数リテラル | 279, 281, 282 | グローバル変数 | 169-173 |
| onload イベントハンドラ | 283 | クラスプロパティとの比較 | 464 |
| イベント呼び出し | 279-280 | スクリプトレベル | 173 |
| キーワード | 44 | 結合 (文字列) | 64, 69, 76 |
| 大文字 | 46 | 決定 | 146 |
| 擬似コード | 158 | 段階的 | 154 |
| 機能の分離 | 270-271 | 複雑 | 218 |
| キャメルケース | 52, 62 | 決定ツリー | 152-153 |
| 空文字列 | 72 | 擬似コード | 158 |
| タッキー | 88, 112-115 | パス | 161 |

| | |
|-----------------|-------------------|
| 検索できるオブジェクト | 424-432 |
| 検証 | 294-296, 302 |
| 構造 | 6-7 |
| 構文 | 51 |
| 構文エラー | 500, 525 |
| コード詳細 | |
| getText() | 567 |
| Mandango プロジェクト | 215 |
| placeOrder() | 313-314 |
| 棒人形の冒險 | 351 |
| コードの重複をなくす | 254-256 |
| コードの分離 | 270-271, 274, 283 |
| コールバック関数 | 277, 281, 282 |
| onload イベントハンドラ | 283 |
| ノーマル関数との違い | 276 |
| 子ノード | 353, 363 |
| コメント | 166, 167 |
| 一時的にコードを無効にする | 527-529, 531 |
| コード内 | 173 |
| セミコロン | 173 |
| 小文字で始まるキャメルケース | 52, 62, 249 |
| コンストラクタ | 402-403, 405, 459 |
| コンテキスト | 169 |
| コンテンツから機能を分離する | 270-271 |

さ行

| | |
|----------------------|---------|
| サーバ | |
| JavaScript コード | 89 |
| 永続性 | 116 |
| データの格納とクッキー | 125 |
| 再読み込み | 111 |
| 再利用可能な作業 | 254-256 |
| 作業、再利用可能 | 254-256 |
| 識別子 | 50 |
| 一意の | 62 |
| 大文字 | 52 |
| 実行時エラー | 524-525 |
| 重要ポイント | |
| 2 次元配列 | 235 |
| AjaxRequest オブジェクト | 560 |
| break 文 | 224 |
| createElement() メソッド | 385 |

| | |
|-----------------------|------------|
| CSS スタイルクラス | 379 |
| Date オブジェクト | 414 |
| DOM | 364 |
| if 文 | 150 |
| innerHTML プロパティ | 364 |
| NaN (Not a Number) | 69 |
| prototype オブジェクト | 459 |
| return 文 | 269 |
| this キーワード | 459 |
| XMLHttpRequest オブジェクト | 560 |
| 引用符 | 511 |
| 大文字 | 69 |
| オブジェクト | 399 |
| 関数参照 | 282 |
| 関数リテラル | 282 |
| クライアントウィンドウ | 104 |
| クラス | 459 |
| グローバル変数とローカル変数の違い | 173 |
| 構文エラー | 511 |
| コールバック関数 | 282 |
| コメント | 173 |
| 初期化 | 69 |
| 正規表現 | 330 |
| ソート | 432 |
| タイマー | 104 |
| データ型 | 49 |
| デバッグ | 498 |
| ノードの className プロパティ | 379 |
| 配列 | 202 |
| フォームフィールド | 307 |
| 複文 | 150 |
| ループ | 202, 224 |
| 初期化 | 69 |
| 定数 | 60-61 |
| 変数 | 45 |
| ループ | 191 |
| 数値 | |
| 加算 | 64, 69, 76 |
| テキストを数値に変換 | 65 |
| 文字列を数値に加算しようとする | 76 |
| スクリプティング | 244 |
| スクリプトの外部からの取り込み | 119 |
| スクリプトレベル | 173 |

| | |
|--------------------------|--------------|
| スコープ..... | 169, 170 |
| グローバル..... | 171 |
| スタイル..... | 6-7 |
| スタイルクラス..... | 302 |
| 正規表現..... | 318-337 |
| \$ | 320 |
| () | 322 |
| * | 322 |
| .(ピリオド) | 320 |
| ? | 322 |
| ^ | 320 |
| {n} | 322 |
| {最小,最大}..... | 329 |
| + | 322 |
| \d | 320 |
| \s | 320 |
| \w | 320 |
| 空データ..... | 329 |
| スラッシュ (/)..... | 320 |
| データ検証..... | 326-327 |
| 電話番号の検証..... | 333 |
| 特殊文字のエスケープ..... | 336 |
| メールアドレスの検証..... | 334-337 |
| メタ文字..... | 320-321, 324 |
| 文字クラス..... | 336 |
| 郵便番号..... | 319 |
| 量化子..... | 322-323 |
| セキュリティ..... | 13 |
| セミコロン | |
| if文..... | 143 |
| コメント..... | 173 |
| ソート..... | 418-423 |
| compare() 関数..... | 421 |
| 関数リテラル..... | 422 |
| 属性 (フォーム)..... | 293 |
| 素朴な疑問に答えます | |
| ! (否定演算子) | 165 |
| 2回以上出現する部分文字列の検索 | 432 |
| Ajax | 580 |
| リクエスト / レスポンス | 561 |
| AjaxRequest オブジェクト | 561 |
| Ajax レスポンスの状態..... | 572 |
| Array.sort() メソッド | 423 |

| | |
|--|-------------------|
| Blog() コンストラクタの引数リスト | 478 |
| Blog オブジェクト | 441 |
| break 文 | 213 |
| callNum 変数 | 511 |
| charAt() メソッドと indexOf() メソッドの比較 | 432 |
| clearInterval() 関数 | 97 |
| createTextNode() 関数 | 371 |
| CSS スタイルクラス | 375 |
| JavaScript クラスとの比較 | 374 |
| CSS スtyleプロパティ | 105 |
| Date オブジェクト | 414 |
| ソート用の比較 | 423 |
| div 要素と span 要素の違い | 371 |
| floor() メソッドと round() メソッドの比較 | 437 |
| form オブジェクト | 294 |
| getElementById() メソッド | 76, 352, 358, 375 |
| getElementsByTagName() メソッド | 358 |
| GET リクエストと POST リクエスト | 561 |
| HTML | |
| size 属性 | 309 |
| タグ | 572 |
| if/else 文 | |
| false 条件 | 143 |
| 複数の else | 143 |
| if 文中のセミコロン | 143 |
| iRock の大きさ変更 | 101 |
| iRock の画像の大きさの計算での 100 の意味 | 105 |
| JavaScript コード | |
| 開始 | 13 |
| 行末に 1 行コメント | 173 |
| サーバ | 89 |
| ブラウザ | 13 |
| 外部ファイルに格納したときのデメリット | 125 |
| Math オブジェクト | 437 |
| NaN (Not a Number) | 62 |
| NOT 演算子 | 219 |
| null | 165 |
| onload イベント | 105 |
| onload イベントハンドラ | 283 |
| onmouseout イベント | 375 |
| onmouseover イベント | 375 |
| parseInt() 関数 | 76 |
| PHP | 580 |

| | |
|---|---------------|
| placeOrder() 関数 | 314 |
| prototype オブジェクト | 459 |
| <script> タグ | 13 |
| setInterval() 関数 | 97 |
| switch/case 文 | 179 |
| this キーワード | 309, 405, 459 |
| searchBlog() 関数で toLowerCase() を呼び出す | 432 |
| toString() メソッド | 423 |
| undefined と not defined の違い | 497 |
| undefined と null の違い | 497 |
| URL エンコーディング | 586 |
| while ループ | 227 |
| XML と Ajax の関係 | 548 |
| YouCube ブログ追加スクリプト | 586 |
| YouCube プロジェクト | |
| Blog オブジェクト | 408 |
| innerHTML | 408 |
| 新しいブログエントリの追加 | 586 |
| エントリをすべて表示ボタン | 408 |
| 追加ボタン | 586 |
| プロパティに署名を格納 | 464 |
| アラートボックス | 314 |
| 検証 | 309 |
| 一意の識別子 | 62 |
| 入れ子になった if 文 | 158 |
| 入れ子のループ | 227 |
| インターバルタイマー | 97 |
| 停止 | 97 |
| インターフリタ | 89 |
| インターラクティブ | 13 |
| ウェブクライアント、ブラウザ、クライアントウインドウ、ブラウザウインドウの違い | 101 |
| 永続的クッキー | 125 |
| エスケープ文字 | 511 |
| 制限 | 511 |
| オブジェクト | |
| new 演算子 | 405 |
| コンストラクタ | 405 |
| データ型 | 398 |
| ドット演算子 | 398 |
| プロパティとメソッド | 398 |
| オブジェクト指向 | 461 |
| オブジェクトとしての文字列 | 432 |
| 影の変数 | 531 |
| 加算と連結の違い | 76 |
| 関数 | |
| いつ関数にするのか | 249 |
| 参照 | 274 |
| 場所 | 260 |
| 複文との比較 | 149 |
| 変数との比較 | 274 |
| 命名規則 | 249 |
| 目的 | 249, 260 |
| 関数リテラル | 281 |
| 擬似コード | 158 |
| キャメルケースと小文字で始まるキャメルケース | 62 |
| クッキー | |
| 格納場所 | 125 |
| 名前 | 125 |
| ブラウザ間での共有 | 125 |
| クライアント | |
| 制御 | 89 |
| クライアントウインドウ | 101 |
| クラスとインスタンスの比較 | 459 |
| クラスプロパティ | |
| インスタンスプロパティとの比較 | 464 |
| グローバル変数との比較 | 464 |
| グローバル変数 | |
| スクリプトレベル | 173 |
| ローカル変数との違い | 171 |
| コード中のコメント | 173 |
| コードの分離 | 274 |
| コードを無効にする | 531 |
| コールバック関数 | 281 |
| onload イベントハンドラー | 283 |
| 子ノード | 363 |
| コメント中のセミコロン | 173 |
| コンストラクタ | 459 |
| サーバとデータどちらにデータを格納するか | 125 |
| 指定時間を経過する前にブラウザを閉じる | 97 |
| スクリプトコードをメソッドにする | 441 |
| 正規表現 | 324 |
| 空データ | 329 |
| メタ文字 | 324 |
| セキュリティ | 13 |
| タイプミス | 497 |

た行

| | |
|--------------------------|---------------|
| タイマー..... | 97 |
| 通常の演算子と論理演算子の違い..... | 219 |
| 定数をいつ使うのか..... | 49 |
| データ型..... | 49 |
| デバッガ..... | 511 |
| デバッグ | |
| Firefox..... | 491 |
| IE | 491 |
| 配列..... | 234 |
| 2次元 | 234 |
| インデクス..... | 201 |
| データの格納..... | 201 |
| データの追加..... | 234 |
| 引数 | |
| 数の上限..... | 260 |
| 変更..... | 260 |
| フォーム..... | 13 |
| onblur イベント..... | 296 |
| イベント発生..... | 296 |
| フィールド..... | 294 |
| フォームフィールドの size 属性..... | 309 |
| 複文..... | 149 |
| for ループ中 | 201 |
| ブラウザ..... | 89 |
| ブラウザのエラーコンソール..... | 491 |
| ログデータ..... | 548 |
| ヘルプテキスト要素..... | 309 |
| ヘルプメッセージのクリア..... | 309 |
| 変数 | |
| いつ使うのか..... | 49 |
| 初期化..... | 45 |
| 複文中..... | 173 |
| 明示的な指定..... | 49 |
| 棒人形の冒險と変数..... | 149 |
| 未定義の変数や関数を参照しようとする..... | 511 |
| 無限ループ..... | 201 |
| 文字列を数値に加算しようとする..... | 76 |
| 要素とその値の違い..... | 76 |
| ループ | |
| 実行しない..... | 227 |
| 条件テストがfalse..... | 201 |
| ループカウンタ..... | 213 |
| 論理演算子..... | 219 |
| タイプミス..... | 496, 497, 532 |
| タイマー..... | 88 |
| 1回限り | 93 |
| インターバル..... | 93, 95, 97 |
| 経過時間..... | 94-97 |
| ミリ秒..... | 95 |
| 指定時間を経過する前にブラウザを閉じる..... | 97 |
| 設定..... | 94 |
| 理解..... | 92 |
| 段階的決定..... | 154 |
| ダンカンドーナツプロジェクト..... | 55-84 |
| parseDonuts() 関数..... | 79 |
| ウェブページからデータをつかむ..... | 71 |
| 計算の問題..... | 58-59 |
| 注文の処理..... | 56-57 |
| 直感的なユーザ入力..... | 77 |
| 文字列の連結と数値の加算..... | 64 |
| ユーザ入力の検索..... | 78 |
| 単項演算子..... | 219 |
| 定数..... | 40, 46-47 |
| いつ使うのか..... | 49 |
| 初期化しない..... | 60-61 |
| 変数との比較..... | 43 |
| 明示的な指定..... | 49 |
| データ | |
| XMLのカスタムタグ..... | 543 |
| 空でないかの検証..... | 300-301 |
| 検証..... | 294-296, 302 |
| アラートボックス..... | 297 |
| → 正規表現も参照 | |
| 電話番号..... | 333 |
| 長さ..... | 305-306 |
| 日付..... | 315-316, 329 |
| メールアドレス..... | 334-337 |
| 郵便番号..... | 310-312 |
| 整合性の保証..... | 75 |
| 動的 → 動的なデータを参照 | |
| フォームデータにアクセス..... | 293 |
| フォームに入力..... | 295 |
| 間違ったフォームデータ..... | 298-299 |
| データ可視性..... | 169 |

| | |
|---------------------------|---------------|
| オブジェクト..... | 398 |
| 明示的な指定..... | 49 |
| テキストデータ型..... | 35 |
| テキストを数値に変換..... | 65, 69 |
| デバッグ..... | 511 |
| デバッグ..... | 486-536 |
| == の違い | 510, 532 |
| Firefox..... | 489-491 |
| Internet Explorer..... | 488, 491 |
| Opera..... | 488 |
| Safari..... | 488 |
| アポストロフィー..... | 502, 532 |
| アラートボックス..... | 507-509 |
| 問題..... | 515 |
| 一時的にコードを無効にする..... | 527-529 |
| 引用符..... | 501-502, 532 |
| ウォッчи..... | 508, 515 |
| エスケープ文字..... | 503, 532 |
| 制限..... | 511 |
| 影の変数..... | 530-532 |
| カスタムデバッグ..... | 517 |
| コンソール..... | 517-520 |
| デバッグ..... | 521-523 |
| 括弧..... | 532 |
| 対応が取れていない..... | 493-494 |
| 構文エラー → 構文エラーを参照 | |
| 実行時エラー..... | 524-525 |
| タイプミス..... | 496, 497, 532 |
| チェックリスト..... | 505, 532 |
| ブラウザ..... | 492 |
| エラーコンソール..... | 491 |
| ページ読み込みの問題..... | 522-523 |
| 下手なスクリプト..... | 512-513 |
| 変数ウォッчи → 変数ウォッчиを参照 | |
| 未定義の変数や関数を参照しようとする..... | 511 |
| 未定義変数エラー..... | 495 |
| 論理エラー → 論理エラーを参照 | |
| 電話番号の検証..... | 333 |
| 動的なデータ..... | 538-598 |
| Ajax → Ajax を参照 | |
| HTTPリクエスト / レスポンス..... | 561 |
| PHP..... | 579-584 |
| URLエンコーディング | 585-586 |
| XHTML..... | 545 |
| 動的なテキスト決定..... | 370 |
| 動的に画像の大きさを変更..... | 108-109 |
| 特別座談会 | |
| forループとwhileループ | 226-227 |
| XHTML..... | 546 |
| 永続的データ格納..... | 114 |
| クラスプロパティとインスタンスプロパティ | |
| との比較..... | 466 |
| ノーマル関数とコールバック関数との違い..... | 276 |
| 下手なスクリプト..... | 512-513 |
| 変数と定数の違い..... | 43 |
| 間違ったフォームデータ..... | 298-299 |
| ローカル変数とグローバル変数の比較..... | 172 |
| 匿名関数..... | 279 |
| ドット演算子..... | 396-397 |
| <h2>な行</h2> | |
| ノード..... | 353 |
| styleプロパティ | 377 |
| visibilityスタイルプロパティ | 377 |
| 型..... | 354, 362 |
| ノードテキストの置き換え..... | 368-369 |
| ノードのテキスト変更..... | 360-361 |
| 複数の子ノード..... | 360 |
| プロパティ | |
| childNodes | 357 |
| firstChild | 357 |
| lastChild | 357 |
| nodeType | 357 |
| nodeValue | 357 |
| <h2>は行</h2> | |
| 配列..... | 198 |
| 2次元 | 231-234 |
| 2次元 Mandango 詳細 | 236-237 |
| データアクセス | 233 |
| インデックス | 199, 201, 202 |
| オブジェクトとしての | 420 |
| 検索できるオブジェクト | 424-432 |
| 座席の画像に対応付ける | 204 |

| | |
|---|---------------|
| ソート | 418-423 |
| compare() 関数 | 421 |
| 関数リテラル | 422 |
| データの格納 | 201 |
| 配列の真実 | 200 |
| フォームフィールド | 293 |
| 配列の真実 | 200 |
| バグ → デバッグを参照 | |
| パターンの達人 | 325 |
| 比較演算子 | 164 |
| 引数 | 252 |
| 上限 | 260 |
| 変更 | 260 |
| 日付、検証 | 315-316, 329 |
| 非同期のリクエスト | 560 |
| 表データ | 233 |
| フィードバック | 261 |
| フォーム | 13, 292 |
| HTML の size 属性 | 309 |
| onblur イベント | 295-297 |
| onchange イベント | 295 |
| onfocus イベント | 295 |
| アラートボックス → アラートボックスを参照 | |
| イベント発生 | 296 |
| 空でないかの検証 | 300-301 |
| 属性 | 293 |
| データ検証 | 294-296, 302 |
| アラートボックス | 297 |
| → 正規表現も参照 | |
| 日付 | 315-316 |
| 郵便番号 | 310-312 |
| データ入力 | 295 |
| データの長さの検証 | 305-306 |
| 配列 | 293 |
| フィールド | 294 |
| form オブジェクトにアクセス | 294 |
| サイズ | 309 |
| フォームデータにアクセス | 293 |
| ヘルプテキスト要素 | 309 |
| ヘルプメッセージのクリア | 309 |
| 間違ったフォームデータ | 298-299 |
| 複文 | 148, 149, 150 |
| for ループ内 | 201 |
| 変数内 | 173 |
| 部分文字列 | 426 |
| ブラウザ | 89 |
| const キーワード | 46 |
| iRock の大きさ変更 | 102-103 |
| JavaScript コード | 13 |
| 実行 | 88 |
| ウェブクライアント、ブラウザ、クライアントウイ
ンドウ、ブラウザウインドウの違い | 101 |
| エラーコンソール | 491 |
| 画面の大きさの不満 | 99-101 |
| クッキー | 88 |
| セキュリティ | 124 |
| クライアントとサーバの応答 | 86 |
| 計測 | 88 |
| 構文エラー | 500 |
| タイマーの有効期限がくる前にブラウザを閉じる | 97 |
| デバッグメッセージ | 492 |
| ブラウザウインドウの大きさ変更 | 106-107 |
| ページ読み込みの問題 | 522-523 |
| 変数を壊す | 111 |
| 履歴 | 88 |
| ブラウザの真実 | 98 |
| プレゼンテーション | 270-271 |
| ページのヘッダ部のコード | 522-523 |
| ページ読み込みの問題 | 522-523 |
| 下手なスクリプト | 512-513 |
| ヘルプテキスト要素 | 309 |
| ヘルプメッセージのクリア | 309 |
| 変数 | 40 |
| NaN (Not a Number) | 62 |
| 新しい変数の作成 | 44 |
| いつ使うのか | 49 |
| 影 | 530-532 |
| 関数との比較 | 274 |
| グローバル | 169-173 |
| 壊す | 111 |
| 初期化 | 45 |
| スコープ → スコープを参照 | |
| 定数との比較 | 43 |
| データ型 | 45 |
| 場所 | 168 |
| 複文中 | 173 |

| | |
|-------------------------------------|-----------------------|
| 棒人形の冒険..... | 170 |
| 未定義の変数や関数を参照しようとする..... | 511 |
| 未定義変数エラー..... | 495-496 |
| 明示的に指定..... | 49 |
| ローカル..... | 169-173 |
| ローカル変数とグローバル変数が同じ名前.... | 530-531 |
| 変数ウォッチ..... | 508, 515 |
| エラーコンソール..... | 516 |
| 十分でない場合..... | 526-527 |
| 編成されたオブジェクト..... | 434 |
| 棒人形の冒険プロジェクト..... | 144-171 |
| changeScene() 関数 | 157, 162, 367 |
| createTextNode() 関数 | 371 |
| CSS スタイルクラスと JavaScript クラスの比較..... | 374 |
| curScene 変数..... | 146 |
| div 要素と span 要素の比較..... | 371 |
| DOM → DOM を参照..... | |
| HTML コードの組み立て | 384 |
| id 属性 (<div> タグ)..... | 346 |
| if 文..... | 155 |
| if/else 文..... | 154 |
| innerHTML プロパティ | 348-350 |
| インターラクティブなオプション..... | 371 |
| 空のアラートボックス..... | 162 |
| 解決方法..... | 163 |
| 空の選択肢..... | 376 |
| 決定ツリー..... | 152-153, 382-383, 387 |
| 擬似コード..... | 158 |
| パス..... | 161 |
| 決定変数..... | 146 |
| コード詳細..... | 351 |
| 個々のスタイル変更..... | 379 |
| コメント..... | 166, 167 |
| ストーリー展開..... | 380-381 |
| 段階的決定..... | 154 |
| 動的なテキスト決定..... | 370 |
| ノードテキストの置き換え..... | 368-369 |
| はじめり..... | 151 |
| 複文..... | 148 |
| 変数..... | 149 |
| チェック | 170 |
| ユーザの決定..... | 146 |
| ボタンを無効にする..... | 574-576 |

ま行

| | |
|-------------------------|-----------------------|
| マークアップ言語..... | 542 |
| 未定義変数エラー..... | 495-496 |
| ミリ秒..... | 95, 104 |
| 無限ループ..... | 201, 223 |
| Mandango プロジェクト | 210 |
| メールアドレスの検証..... | 334 |
| メソッド..... | |
| getElementById() | 71, 76, 101, 102 |
| reload() | 97 |
| 関数をメソッドに変える..... | 441-442 |
| クラスへの格納..... | 455 |
| スクリプトコードをメソッドにする..... | 441 |
| メタ文字..... | 320-321, 324 |
| 文字クラス..... | 336 |
| 文字列..... | |
| 2 回以上出現する部分文字列の検索 | 432 |
| length プロパティ | 306-307 |
| 引用符のデバッグ | 501-502 |
| エスケープ文字 | 503 |
| オブジェクトとしての | 432 |
| 空 | 72 |
| 数値に加算しようとする | 76 |
| 文字列中の | 426 |
| 連結 | 64, 69, 76 |
| 問題の分割..... | 244-245 |
| prototype オブジェクト | 456-457, 459, 468-469 |

や行

| | |
|--------------|---------|
| ユーザ入力..... | 4-5 |
| 検索 | 78 |
| 信用 | 83 |
| 直感的 | 77 |
| ユーザの決定..... | 146 |
| 決定ツリー | 152-153 |
| 擬似コード | 158 |
| パス | 161 |
| 段階的決定 | 154 |
| 郵便番号の検証..... | 310-311 |
| 正規表現 | 319 |

| | |
|--------------------------------|---------------|
| 要素 | |
| ID | 71 |
| その値との違い | 76 |
| 要注意！ | |
| = と == の比較 | 164 |
| break 文 | 179 |
| const キーワード | 46 |
| CSS スタイルクラスと JavaScript クラスの比較 | 372 |
| Date メソッド | 417 |
| id 属性 | 346 |
| while ループ | 223 |
| 画像の大きさ変更 | 109 |
| クッキー | 124 |
| 正規表現の特殊文字 | 336 |
| タイマーの経過時間 | 95 |
| 日付の検証 | 329 |
| 文字列の連結 | 64 |
| 郵便番号 | 312 |
| 乱数 | 434-437 |
| 量化子 | 322-323 |
| ループ | 189-242 |
| break 文 | 212, 213, 224 |
| continue 文 | 212 |
| for → for ループを参照 | |
| while → while ループを参照 | |
| アクション | 191 |
| 入れ子のループ | 227 |
| 更新 | 191 |
| 条件テストが false | 201 |
| 初期化 | 191 |
| テスト条件 | 191 |
| 無限ループ | 201, 223 |
| ループカウンタ | 213 |
| ローカル変数 | 169-173 |
| 論理エラー | 510-511, 525 |
| 下手なスクリプト | 512-513 |
| 論理演算子 | |
| 括弧 | 219 |
| 通常の演算子と論理演算子の違い | 219 |

ら行

| | |
|------------------|--------------|
| ラジオ番組の着呼スクリプト | 498-499 |
| callNum 変数 | 507-508, 511 |
| checkWinner() 関数 | 504-505 |
| アラートボックス | 507-508 |
| 引用符のデバッグ | 501-502 |

| | |
|-----|-----|
| ワーム | 124 |
|-----|-----|

わ行

●訳者紹介

豊福 剛 (とよふく つよし)

Webアプリの企画、開発、運営を手がけるIT仕事人。主な訳書『Java クックブック』
『Linux サーバセキュリティ』(オライリー・ジャパン)。

2004年以来のMacOSXユーザで、Eclipse、TextMate、MySQL、Ruby on Rails、
Plagger、Symfony、Seasar2などなどを使っている。

Head First JavaScript—頭とからだで覚えるJavaScriptの基本

2008年8月8日 初版第1刷発行

2009年6月26日 初版第2刷発行

著 者 Michael Morrison (マイケル・モリソン)

訳 者 豊福 剛 (とよふく つよし)

発 行 人 ティム・オライリー

制 作 合同会社ビーンズ・ネットワークス

印 刷 株式会社ルナテック

製 本 株式会社越後堂製本

発 行 所 株式会社オライリー・ジャパン

〒160-0002 東京都新宿区坂町26番地27 インテリジェントプラザビル1F

Tel (03)3356-5227

Fax (03)3356-5263

電子メール japan@oreilly.co.jp

発 売 元 株式会社オーム社

〒101-8460 東京都千代田区神田錦町3-1

Tel (03)3233-0641 (代表)

Fax (03)3233-3440

Printed in Japan (ISBN978-4-87311-373-9)

乱丁本、落丁本はお取り替え致します。

本書は著作権上の保護を受けています。本書の一部あるいは全部について、株式会社オライリー・ジャパンから文書による許諾を得ずに、いかなる方法においても無断で複写、複製することは禁じられています。