YouTube版
JavaScriptで学ぶ
プログラミング入門　丸１日コース

マーリンアームズ株式会社
武舎広幸

演習問題の解答例や参考資料

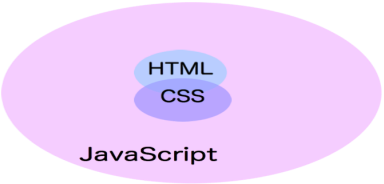
演習問題の解答例

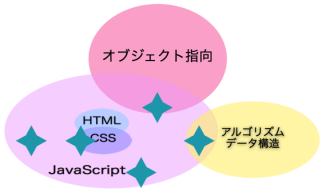
`https://www.musha.com/sc/jsia`

・演習問題の解答例
・資料へのリンク

本日の目標

✦JavaScript（プログラム）でどんなことができるのか
✦どんな風にやればよいのか
✦全体像をつかむ





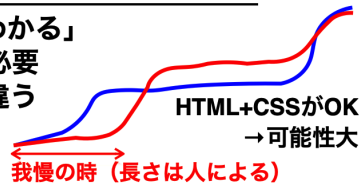
肝となるところに
狙いを定めて

プログラミング言語と[　]- 共通点

- 意味や意図を相手に伝えるための道具
  - [　] — 人間との対話
  - JavaScript — コンピュータへの依頼
- 文法を覚える必要があるが覚えただけでは使えない —— 練習が必要
  単語数 [　] > JavaScript
  【何年か必要】【最短で数ヵ月】
- 対象の領域に関する知識も必要
  - 位置ゲー、将棋、翻訳ソフト...

プログラミング言語と英語 - 共通点

- ある日突然「わかる」
  - 浸る時間が必要
  - 人によって違う



- 理系、文系はあまり関係がない
  ただし、題材・教材に偏りがある
  - プログラミング — 理系の話題
  - 英語 — 文系の話題

プログラミング言語と英語

✦英語ができたほうがよい　プログラムは形式的な英語

- 形式的な英語でコンピュータに命令
- 「コード」を「読む」ことができれば意味がわかる
- 命令（コマンド、動詞）や操作対象（名詞）は（基本）すべて英語
- エラーメッセージなども英語（これは普通の英語）
- 英語の綴りを間違うと動かない
- 入力が速いほうが能率が上がる
  https://www.e-typing.ne.jp/english/
- ドキュメント（資料）も英語が多い
- ただし、そんなに難しい英語は使われない

役に立つ英語を覚えるチャンス

本日の単語帳（1）≒JSの単語帳

✦単語全体の「感じ」を覚える

- alert [ələ́rt] or [ələ́ːt] - 注意（せよ）
- prompt - 促進する、（入力を）促す
- function - 機能、関数
- for - 〜に対して、〜の間、〜のために
- loop - 繰り返し、ループ
- variable（var）- 変化する、変数
  vary + able - 変わる+〜ができる
- let - 〜にする、~させる
  let x = 3; ←xの値を3にする

本日の単語帳（2）

- if - もし...ならば、...のとき、...の場合
- else - そうでなければ、ほかの（時）
- true - 真、正しい
- false - 偽、正しくない、間違い
- warning [wɔ́ːrniŋ] - 警告（エラーの可能性大）
- error - 間違い、エラー
- math - 数学（←mathematics）
- random - ランダムな、無作為の
- floor - 床、切り下げる
- operator - 演算子、オペレータ
- operand - 何かをされる人（物）、被演算子
- division - 部分、〜部、分割 ← タグ
- span - 及ぶ、期間、間の距離 ←タグ

本日の単語帳（3）Part II

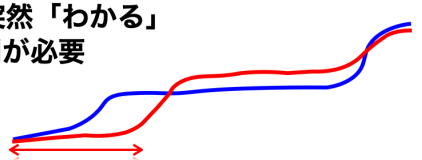
- object - もの、物体、オブジェクト
- property - 性質、属性、プロパティ
- method - 方法、やりかた、手段、メソッド
- document -文書、ドキュメント、書類
- element - 要素、構成要素、成分
- ID ←identification - 識別、身分証明（書）
- inner - 内側の、内部の
- height [hait]（ハイト）- 高さ
- width [widθ]（ウィドゥス） - 幅
- margin - 余白、余裕、へり、利幅
- integer - 整数 (0, 1, 2, ..., -1, -2,...)
- location - 場所、ロケーション、位置
- strict - 厳格な、厳密な

今日の講座

- 「文法の概要」と「練習の仕方の例」
- 「感じ」をつかむ。まず「読み方」を覚える
- 細かい点は資料を見ながら「コード」を書いていくうちに自然に覚える
  今日の実習 ＋（自習 or「実習編」）
- 意識的に覚える必要は（あまり）ない
- 重要な概念は繰り返し登場するので、そのたびに復習すればよい
- 実習をすれば徐々に「納得」できる

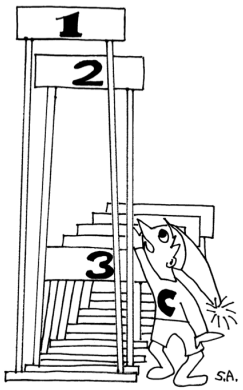
楽しめれば上達は早い

- 「もの」を作る喜び。バーチャルな「もの」
  cf. DIY、裁縫　　小説、作曲
- 何時間でも付き合ってもらえる ——
  英会話のように「他人」がいなくても大丈夫
- ある日突然「わかる」
  浸る時間が必要

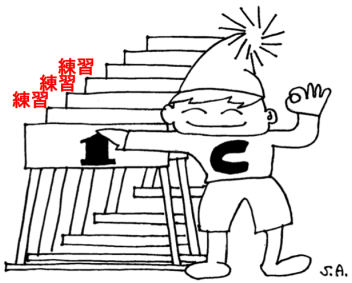


我慢の時（人によって長さは違う）

現在



終了時



今日の講座

- JSを知っている方
  - 新機能（ES2015）に注目
  - 概念の整理
- 他の言語を知っている方
  - HTMLを使って比較的簡単に面白いことができる（とくに後半の話）
  - 画像を動かしたり変化させたり

念のため

- JavaScript（JS）とJavaは別の言語
- HTMLやCSSはプログラミング言語ではない

今日の時間割

基本概念1　関数　分岐　ループ

基本概念2　自作関数　タイマー　配列

オブジェクト指向
イベント
まとめ

「登場人物」

- 関数（function）
- 分岐（if）
- 変数（let）
- 演算子（+, -, <, <=, ...）
- 繰り返し（for、setinterval）
- 配列
- オブジェクト指向（プロパティ, メソッド）
- イベント

JavaScriptで学ぶ プログラミング入門　丸１日コース PART I

- 基本概念
  - 関数
  - 分岐（選択肢）
  - 変数
  - 繰り返し
  - 演算子
  - 配列

初めてのJavaScriptプログラム

まずとても単純なプログラムを見てみましょう

初めてのJSプログラム

課題　ダイアログボックスの表示

ダイアログボックスを使って
「こんにちは」と表示せよ。

- JavaScriptを使わなければ（HTMLでは）ダイアログボックスの表示はできない
- 今日の主題はブラウザのJavaScript
- HTMLファイルに書いてブラウザで実行するのが簡単
- Node.jsなどを使うとブラウザ以外でも使える
- 基本は共通

JavaScriptの「コード」

```
<!DOCTYPE html>
<html>
<head>                                          <head>部に書いてもOK
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>hello, world</title>
</head>
<body>
  <script type="text/javascript">
    alert("こんにちは");
  </script>
  <p>
    本文
</p>
</body>
</html>
```

JavaScriptのプログラム

関数（メソッド）

```
<script type="text/javascript">
alert("こんにちは");          行末の「;」は省略可
</script>                     PHP, Java等では必須
```

関数名（引数 or パラメータ）

半角の括弧

プログラムは形式的な英語

「こんにちは」とalert（警告せよ）
→ダイアログボックスに「こんにちは」表示せよ
ダイアログボックスに書くのは「お約束」

関数

関数（メソッド）はプログラムの重要な構成要素

Excelの関数と類似

1. 関数（function）

1.何らかの機能（function）を提供

```
alert("こんにちは");
document.write("こんにちは");
```

プログラムは形式的な英語

「こんにちは」とdocument領域にwrite（書け）
HTMLコードとして書く（お約束）

2.引数を変えると動作が変化

```
alert("こんにちは")
alert("こんばんは")
document.write("こんにちは<br>");
document.write("こんばんは<br>");
```

3.プログラムは上から下に順番に実行される（原則）

プログラムは上から下に順番に実行される（原則）

alertやdocument.writeの引数

- 引数の文字列は"..." でも '...' でも`...`でも

```
document.write("こんにちは<br>");
document.write('おはようございます<br>');
document.write(`こんにちは<br>`);
```

自分でどれかに決める（一貫性）

- `...`（バッククオート）の利用

画像タグなど文字列中に引用符があるときに便利
（Shift+@。英語配列では左上にある）

- document.write（など）で任意のHTMLを出力できる

```
document.write(`<img src="picture000.jpg">`);
document.write(`<img src="picture001.jpg">`);
```

Math.random— ランダムな値（0以上1未満）

Math.random()　0以上1未満の値を返す

```
document.write(Math.random());
```

① ②

プログラムは形式的な英語

① Math（数学）というまとまり（オブジェクト）のうちrandom（乱数を発生する）という機能を呼び出して値をもらえ

②その値をdocument領域にwrite（書け）

0以上1未満の小数がdocument領域に表示される

```
alert(Math.random());
```

0以上1未満の小数がダイアログボックスに表示される

関数　まとめ

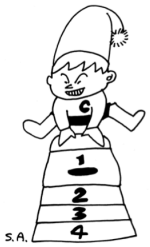
✦関数（組み込みの関数）

- システム（JS処理系）が色々な機能を提供
- 組み合わせてプログラムを作っていく
- alert —— ダイアログボックスにメッセージ
- document.write —— HTMLのコードを出力
- Math.random
- ..... などなど、たくさんある

分岐（選択肢）

条件によって処理を変える

if文



2. 分岐（選択肢）

課題　おみくじ

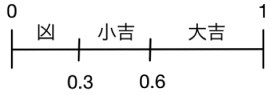
今日の運勢（大吉、小吉、凶のいずれか）を表示せよ

✦手順

- Math.random() —— 0以上1未満の小数をランダムに返す

返ってきた値（戻り値）によって表示を変える

・0.3未満なら「凶」
・0.3未満でなくて、0.6未満なら「小吉」
・それ以外なら（0.6以上なら）「大吉」



if文を使う

if文

プログラムは形式的な英語

```
let x = Math.random();   xをMath.random()と同じにせよ
if (x < 0.3) {           x < 0.3なら（if）
  x = "凶";              xを"凶"と同じにせよ
}
else if (x < 0.6) {      そうでなくて（else） x < 0.6なら（if）
  x = "小吉";                                つまり0.3≦x<0.6なら
}
else {                   そうでなければ（else） つまり0.6≦xなら
  x = "大吉";
}
document.write(x);       xの値をdocument領域にwrite（書け）
```

変数の宣言と代入　let x = ...

1.「変数」の「宣言」 —— 情報の記憶場所の確保

```
let x = Math.random();
```

x	

var も使えるが…
変数は最初に一度だけ宣言する
JSの変数に「型」の指定は不要

2.「代入」 —— 変数に値を入れて記憶

```
let x = Math.random();
```

x	0.314

xをMath.random()が返した値と同じにする
→xという領域に値を記憶する

「x = Math.random()」の「=」は数学の「=」とは意味が違う
むしろ let x ← 0.314;
「状態」ではなく「アクション」！

if文

プログラムは形式的な英語

```
let x = Math.random();   xをMath.random()と同じにせよ
if (x < 0.3) {           x < 0.3なら（if）
  x = "凶";              xを"凶"と同じにせよ
}
else if (x < 0.6) {      そうでなくて（else） x < 0.6なら（if）
  x = "小吉";                                つまり0.3≦x<0.6なら
}
else {                   そうでなければ（else） つまり0.6≦xなら
  x = "大吉";
}
document.write(x);       xの値をdocument領域にwrite（書け）
```

変数の値の変化

```
let x = Math.random();
if (x < 0.3) {
  x = "凶";
}
else if (x < 0.6) {
  x = "小吉";
}
else {
  x = "大吉";
}
document.write(x);
```

x	0.314

xの値を「参照」

x	"小吉"

xに「代入」。同じxを使うときletは書かない

x	"小吉"

再度宣言してしまうと...

```
let x = Math.random();
if (x < 0.3) {
  let x = "凶";
}
else if (x < 0.6) {
  let x = "小吉";
}
else {
  let x = "大吉";
}
document.write(x);
```

x	0.314

x	"小吉"

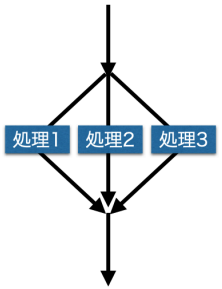
使われずに終わってしまう

x	0.314

x	0.314

if文

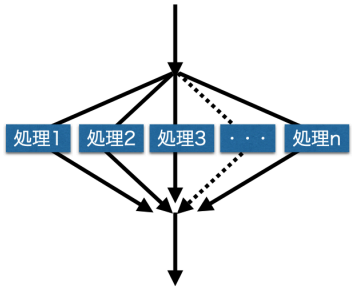
```
let x = Math.random();
if (x < 0.3) {
  x = "凶";
}
else if (x < 0.6) {
  x = "小吉";
}
else {
  x = "大吉";
}
document.write(x);
```



分岐のパターン

✦if文の一般形

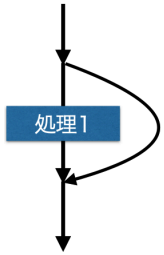
```
if (条件1) {
  <処理1>
}
else if (条件2){ //省略可
  <処理２>
}
else if (条件3){ //省略可
  <処理3>
...
}
else { //省略可
  <処理n>
}
```



分岐のパターン

✦if文 —— やるかやらないか

```
if (条件1) {
  <処理1>
}
```

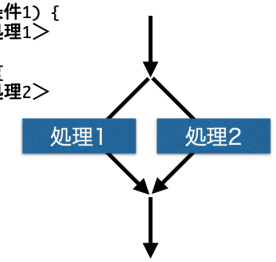



```
if (条件1) {
  <処理1>
}
else if (条件2){ //省略可
  <処理２>
}
else if (条件3){ //省略可
  <処理3>
...
}
else { //省略可
  <処理n>
}
```

分岐のパターン

✦if ... else ... —— どちらかの処理をする

```
if (条件1) {
  <処理1>
}
else {
  <処理2>
}
```

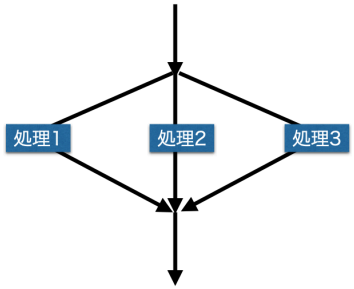



```
if (条件1) {
  <処理1>
}
else if (条件2){ //省略可
  <処理２>
}
else if (条件3){ //省略可
  <処理3>
...
}
else { //省略可
  <処理n>
}
```

分岐のパターン

✦三択

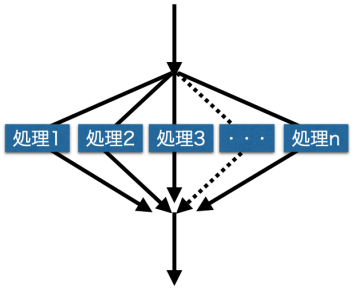
```
if (条件1) {
  <処理1>
}
else if (条件2){
  <処理２>
}
else {
  <処理3>
}
```



分岐のパターン

✦if文の一般形

```
if (条件1) {
  <処理1>
}
else if (条件2){ //省略可
  <処理２>
}
else if (条件3){ //省略可
  <処理3>
...
}
else { //省略可
  <処理n>
}
```



分岐（選択肢）　まとめ

✦if文

```
if (条件1) {
  <処理1>
}
else if (条件2){
  <処理２>
}
else if (条件3){
  <処理3>
...
}
else { //省略可
  <処理n>
}
```

```
if (条件1) {
  <処理1>
}

if (条件1) {
  <処理1>
}
else {
  <処理2>
}
```

- （switch文）

プログラムは上から下に順番に実行される（原則）

分岐

TOPIC

コメント

TOPIC　コメント

✦開発者（他人+将来の自分）用のメモ。処理に影響はない

```
let x = Math.random();// 0≦x<1
```
← コメント

✦コメントの別の形式（複数の行を使ったコメント）

```
/*                    CSSのコメントと同じ形式
1. Math.random()で 0以上1未満の小数を得る
2. 0.3未満なら「凶」
3. 0.3以上0.6未満なら「小吉」
4. その以外なら「大吉」
*/
```

TOPIC

文と演算子

TOPIC　文と演算子

```
alert("こんにちは");
let x = Math.random();
let カウンタ = 10;
document.write(x);
```

ひとつひとつを **文** という

JavaScriptでは行末の「;」は省略できる。PHPなどでは必須

なぜ「;」（セミコロン）か？

```
alert("こんにちは");
```

✦区切り文字（punctuation）の強さ

, < : < ; < .!?

プログラムは形式的な英語

✦CSS

width; 200px: height; 180px: ✕

width: 200px; height: 180px;

プログラムは形式的な英語

TOPIC　文と演算子

```
alert("こんにちは");
let x = Math.random();
let カウンタ = 10;
document.write(x);
```

ひとつひとつを **文** という

JavaScriptでは行末の「;」は省略できる。PHPでは必須

if文

```
if (x < 0.3) {
  x = "凶";  //←代入文
}
else if (x < 0.6) {
  x = "小吉";
}
else {
  x = "大吉";
}
```

私は太郎さんが沖縄に行ったことを知っている。

論理演算子（ロジカルオペレータ）

✦if文などの条件で使われる

```
if (x < 0.3) {     // ←演算子（オペレータ）
  x = "凶";        // 論理演算子
}
```

```
x < y   xがyより小さければ真(true)→条件成立
x > y
x <= y  xがyと同じか小さければ真（以下）
x >= y
x === y  xとyが等しければ真（x==y）
x !== y  xがyと等しくなければ真（x!=y）
```

○ <= "less than or equal to"
>= "greater than or equal to"
✕ =<, =>

プログラムは形式的な英語

ループ（繰り返し）

プログラムでは何度でも（1万回でも1億回でも）同様の操作を繰り返し行うことができる

for文

3. ループ（繰り返し）

課題　画像の連続表示

順番に名前が付けられた8枚の画像を連続して表示せよ

✦HTMLだけでもできる

```
<img src="pictures/picture000.jpg">
<img src="pictures/picture001.jpg">
<img src="pictures/picture002.jpg">
<img src="pictures/picture003.jpg">
<img src="pictures/picture004.jpg">
<img src="pictures/picture005.jpg">
<img src="pictures/picture006.jpg">
<img src="pictures/picture007.jpg">
```

✦JavaScriptを使えば楽ができる

```
for (let i=0; i<8; i++) {
  document.write(`<img src="pictures/picture00${i}.jpg">`);
}
```

forループ（for文）

✦同じようなことを何度でも繰り返せる

```
for (let i=0; i<8; i++) {←for three years（3年間）のfor
  document.write(`<img src="pictures/picture00${i}.jpg">`);
}
```

(let i=0; i<8; i++) の間（for）
iを0からはじめて;
iが8より小さい間;
iを1ずつ増やして
「<img ...>」をdocument領域にwrite（書け）

プログラムは形式的な英語

for -- 対象を示す。中核の意味は「〜に対して」＋「〜のために」的なニュアンス
to -- 似ているけど、「〜のために」はない

i
0
1
2
3
4
5
6
7

テンプレートリテラル（可変部分付き文字列）

✦変数（や計算式）を含む文字列

```
for (let i=0; i<8; i++) {
  document.write(`<img src="pictures/picture00${i}.jpg">`);
}
```

絶対覚える

「`」はバッククオート
文字列中に変数を使う時は必ず `...`（バッククオート）

「<img src="....${i}.jpg">」の文字列をdocument領域にwrite。
ただし`...`の中の${...}の部分は計算（評価）してから

```
i=0 → <img src="pictures/picture000.jpg">
i++ → <img src="pictures/picture001.jpg">
i++ → <img src="pictures/picture002.jpg">
i++ → <img src="pictures/picture003.jpg">
...
i++ → <img src="pictures/picture007.jpg">
```

テンプレートリテラル `...`

`...`（バッククオート）を使わずに
"..."や'...'（普通の引用符）を使うと${i}をそのまま出力

```
for (let i=0; i<8; i++) {
  document.write('<img src="pictures/picture00${i}.jpg">');
}
```

```
i=0 → <img src="pictures/picture00${i}.jpg">
i++ → <img src="pictures/picture00${i}.jpg">
i++ → <img src="pictures/picture00${i}.jpg">
i++ → <img src="pictures/picture00${i}.jpg">
...
i++ → <img src="pictures/picture00${i}.jpg">
```

pictures/picture00${i}.jpgというファイルはない
→ 画像は表示されない

forループ

✦同じようなことを何度でも繰り返せる

```
for (let i=0; i<8; i++) {
  document.write(`<img src="pictures/picture00${i}.jpg">`);
}
```

✦まったく同じことの繰り返しにももちろん使える

```
for (let i=0; i<10; i++) {
  document.write(`蓼鶏湯が食べたい。<br>`);
} // 蓼鶏湯（サムゲタン）は韓国料理
```

コメント

```
蓼鶏湯が食べたい。
蓼鶏湯が食べたい。
蓼鶏湯が食べたい。
蓼鶏湯が食べたい。
...
```

forループ ― スタイルの一斉変更

✦同じようなことを繰り返すのが普通

```
for (let i=1; i<=10; i++) {
  document.write(`${i}. 蓼鶏湯が食べたい。<br>`);
}
```

1. 蓼鶏湯が食べたい。
2. 蓼鶏湯が食べたい。
...

1箇所を変えるだけで、すべてが変わる
何箇所もコピペする必要がない

✦1箇所変えるだけですべての繰り返しが変わる

```
for (let i=0; i<8; i++) {
  document.write(
    `<img src="xxx00${i}.jpg" style="width: 25%">`);
}
```

```
<img src="pictures/picture000.jpg" style="width: 25%">
<img src="pictures/picture001.jpg" style="width: 25%">
...
```

ループ まとめ

✦人手でもできるが「楽ができる」典型例
✦同じ（ような）ことを何度もやる

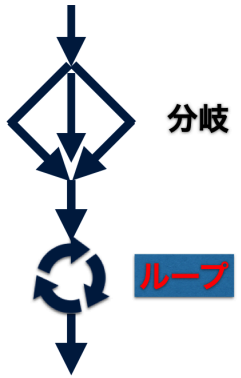
```
for (let i=0; i<8; i++) {
  document.write(`<img src="pictures/picture00${i}.jpg">`);
}
```

```
    ①      ②    ③
for (初期設定; 条件; 再設定) {
  ＜本体：繰り返す作業＞ ④
}
⑤
```

i
0
1
2
3
4
5
6
7

1回目　①→②→④/⑤
2回目以降　③→②→④（②が真（true）のあいだ）
条件不成立　③→②→⑤

✦（forの別の構文、while、 forEach）



TOPIC

算術演算子

Computer（計算をするもの）
なので計算は得意

TOPIC　算術演算子（加減乗除）

```
x * 10;   掛け算は「x」ではなくて「*」
x + 1;    引き算は「-」 割り算は「/」
```

算術演算子

✦代入演算子

```
x = x * 10;
```

右辺の式x*10の値を
左辺の変数xに代入

x ← x * 10;

代入演算子

全体は代入文

演算子（オペレータ）

✦おもしろい（便利な）演算子 +=, -=, *=, /=

x += 5;　⇔　x = x+5;　どちらも現在の値に5を足す

```
let x = 1;
x += 5;
alert(x);
```

x
1
6
6

```
x -= 4;   ⇔  x = x-4;   let x = 8; x -= 4; alert(x);
x *= 8;   ⇔  x = x*8;   let x = 3; x *= 8; alert(x);
x /= 2;   ⇔  x = x/2;   let x = 6; x /= 2; alert(x);
```

演算子（オペレータ）

✦もう1種類 ++, --　1を足す、1を引くは非常によく使う

x++ ⇔ x += 1 ⇔ x = x+1
x-- ⇔ x -= 1 ⇔ x = x-1

✦次の結果は？

```
let x = 4;
x += 5;
x -= 2;
x++;
document.write(x);
```

x
4
9
7
8

```
let x = 4;
let y = 10;
x = y + 5;
y -= x;
y--;
document.write(y);
```

x	y
4	--
4	10
15	10
15	-5
15	-6

TOPIC

文字列の連結と テンプレートリテラル

TOPIC 文字列連結演算子

✦"…" あるいは '…' あるいは `…`で「文字列」を表す
✦演算子+は文字列の連結にも
✦次の結果は？

```
let x = "初めての";
let y = "JavaScript";
let z = "HTML+CSS";
document.write(x+y);
document.write(x+z);
document.write(" 『"+x+y+"』 ");
document.write(` 『${x+y}』 `);
document.write(` 『${x}${y}』 `);
```

x	"初めての"
y	"JavaScript"
z	"HTML+CSS"

初めてのJavaScript
初めてのHTML+CSS
『初めてのJavaScript』
『初めてのJavaScript』
『初めてのJavaScript』

+= で自分の後ろに追加

```
let html="";
for (let i=0; i<10; i++) {
  html +=`<img src="pictures/picture00${i}.jpg">`;
}
document.write(html)
```


i	html
	""
0	<img src="pictures/picture000.jpg">
1	<...000.jpg"><...001.jpg">
2	<...000.jpg"><...001.jpg"><...002.jpg">
3	<...000.jpg"><...001.jpg"><...002.jpg"><...003.jpg">
	...
9	<...000"><...001><...002><...003>...<...008><...009.jpg">

テンプレートリテラル（可変部分付き文字列）

```
document.write(` 『${x+y}』 `);
```

✦文字列の中に${...}で囲んで変数や計算式を書ける
✦ES2015（ES6）から使えるようになった
✦「+」で繋がなくても済むようになった

```
let 画像タグ =
  `<img src="picture00${i}.jpg" style="width: 4cm;">`
```

✦以前の書き方 「+」を使って連結

```
var 画像タグ =
  '<img src="picture00' + i + '.jpg"'
          + ' style="width: 4cm;">';
```

「"」と「'」の組み合わせ → すごく大変だった

今日の時間割

基本概念1　関数　分岐　ループ

基本概念2　自作関数　タイマー　配列

オブジェクト指向
イベント

まとめ

関数（組み込みの関数）

- システム（JS処理系）が色々な機能を提供
- 組み合わせてプログラムを作っていく
- alert —— ダイアログボックスにメッセージ
- document.write —— HTMLのコードを出力
- Math.random
- ..... などなど、たくさんある

分岐（選択肢）　まとめ

✦if文

```
if (条件1) {
  <処理1>
}
else if (条件2){
  <処理２>
}
else if (条件3){
  <処理3>
...
}
else { //省略可
  <処理n>
}
```

```
if (条件1) {
  <処理1>
}

if (条件1) {
  <処理1>
}
else {
  <処理2>
}
```

- （switch文）

ループ

✦人手でもできるが「楽ができる」典型例
✦同じ（ような）ことを何度もやる

```
for (let i=0; i<8; i++) {
  document.write(`<img src="pictures/picture00${i}.jpg">`);
}
```

```
        ①     ②    ③
  for (初期設定; 条件; 再設定) {
    <本体：繰り返す作業> ④
  }
  ⑤
```

1回目　①→②→④/⑤
2回目以降　③→②→④（②が真（true）のあいだ）
条件不成立　③→②→⑤

✦（forの別の構文、while、 forEach）

i
0
1
2
3
4
5
6
7

「登場人物」

- 関数（組み込み）+関数（自作）
- 分岐（if）
- 変数（let）
- 演算子（+, -, <, <=, ...）
- 繰り返し（for）
- 配列
- オブジェクト指向（プロパティ, メソッド）
- イベント

今日の時間割

基本概念1　関数　分岐　ループ

基本概念2　自作関数　タイマー　配列

オブジェクト指向
イベント

まとめ

自作の関数

関数は自分でも作れる

```
function xxx(p1, p2) {
  ...
  return y;
}
```

4. 自作の関数

- 一連の処理をまとめる

```
let x = Math.random();
if (x < 0.3) {
  x = "凶";
}
else if (x < 0.6) {
  x = "小吉";
}
else { // それ以外なら
  x = "大吉";
}
x = `今日の運勢は${x}です`;
document.write(x);
```

- 関数は呼び出されて初めて実行される。呼び出し時は引数がなくても()付き
- 定義だけでは実行されない

```
let x = おみくじを引く();
document.write(`今日の運勢は${x}です`);

function おみくじを引く() {
  // 関数名には日本語も可
  let y = Math.random();
  if (y < 0.3) {
    y = "凶";
  }
  else if (y < 0.6) {
    y = "小吉";
  }
  else { // それ以外なら
    y = "大吉";
  }
  return y; // 戻り値
}
```

何回も呼び出す

```
document.write(`西野さんの運勢：${おみくじを引く()}<br>`);
document.write(`平原さんの運勢：${おみくじを引く()}<br>`);
document.write(`家入さんの運勢：${おみくじを引く()}<br>`);

function おみくじを引く() {
  let x = Math.random();
  if (x < 0.3) {
    x = "凶";
  }
  else if (x < 0.6) {
    x = "小吉";
  }
  else {
    x = "大吉";
  }
  return x; // return は1箇所にまとめておいたほうがデバッグが楽
}

      document.write(`今日の運勢：${おみくじを引く()}<br>`);
      document.write(`あしたの運勢：${おみくじを引く()}<br>`);
      document.write(`あさっての運勢：${おみくじを引く()}<br>`);
```

`...` の中で計算もできる

関数を呼び出すこともできる

自作の関数 —— 引数

- 引数を指定

```
運勢を表示("西野", "今日");
運勢を表示("平原", "明日");
運勢を表示("家入", "明後日");

function 運勢を表示(人名, ひにち) {
  let x = Math.random();
  if (x < 0.3) {
    x = "凶";
  }
  else if (x < 0.6) {
    x = "小吉";
  }
  else {
    x = "大吉";
  }
  x = `<span style="color: red;">${x}</span>`;
  document.write(`${人名}氏の${ひにち}の運勢は${x}である。<br>`);
}
```

実引数　呼び出す時。実際の値を指定

関数は値（戻り値）を返さなくてもよい

仮引数　呼ばれたときに値が入る

西野氏の今日の運勢は凶である。
平原氏の明日の運勢は大吉である。
家入氏の明後日の運勢は小吉である。

自作の関数 —— 引数

- 別解

```
運勢を表示("西野", "今日");
運勢を表示("平原", "明日");
運勢を表示("家入", "明後日");

function 運勢を表示(人名, ひにち) {
  let x = おみくじを引く(); // さらに関数を呼ぶ
  x = `<span style="color: red;">${x}</span>`;
  document.write(`${人名}氏の${ひにち}の運勢は${x}である。<br>`);
}

function おみくじを引く() {
  let x = Math.random();
  if (x < 0.3) {
    x = "凶";
  }
  else if (x < 0.6) {
    ...
    ...
}
```

実引数　呼び出す時。実際の値を指定

関数は値（戻り値）を返さなくてもよい

西野氏の今日の運勢は凶である。
平原氏の明日の運勢は大吉である。
家入氏の明後日の運勢は小吉である。

関数の中で別の関数を呼ぶこともできる

数学の関数との比較

何らかの計算

入力 → 数学の関数 → 値（戻り値）を返す

$f(x) = x^2+1$ → $f(3) = 3^2+1 = 10$

入力（引数） →	JSの関数1 →	同じ入力には同じ戻り値（数学的な関数）
入力（引数） →	JSの関数2 →	同じ値を返すとは限らない
入力（引数） →	JSの関数3 →	一連の処理をし戻り値なし（手続的な関数）
入力（引数） →	JSの関数4 →	一連の処理をし戻り値あり

「手続き」をもつ（古い）言語もある

手続き
procedure（Pascal）
subroutine（Fortran）

関数
function
戻り値があるものだけ

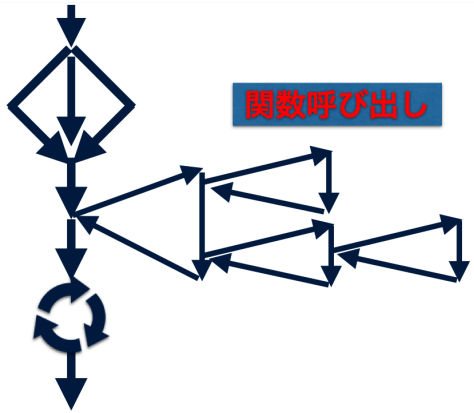
入力（引数） →	JSの関数1 →	同じ入力には同じ戻り値 戻り値あり
入力（引数） →	JSの関数2 →	同じ値を返すとは限らない
入力（引数） →	JSの関数3 →	一連の処理をし戻り値なし
入力（引数） →	JSの関数4 →	一連の処理をし戻り値あり

JavaScriptではみんな関数

関数
function
戻り値の有無に
かかわらず

自作の関数 —— 長所

- 記述の繰り返しが避けられる
  - 同じ（ような）処理が何度も現れる
    - → 関数にまとめるとわかりやすくなる
  - 変更が簡単になる（1箇所直すだけで済む）
- 流れがわかりやすくなる
  - 詳細は関数を参照！
- 処理の「部品化」「概念化」「抽象化」
  - —— 「タイマー」や「イベント」などの処理に利用
- 関数はプログラムを読みやすくする重要な道具



関数 —— 抽象化の道具

- トップレベルは関数の呼び出しの連続（長いプログラム）

```
//全体の流れを関数呼び出しで記述する
処理1を実行する関数(引数...);
処理2を実行する関数(引数...);
if (●● < ××) {
  処理3_1を実行する関数(引数...);
}
else {
  処理3_2を実行する関数(引数...);
}
処理4を実行する関数(引数...);

function 処理1を実行する関数(引数...) {
  ... // 処理1_1、処理1_2、処理1_3、...をさらに関数で記述
}
function 処理2を実行する関数(引数...) {
  ... // 処理2_1、処理2_2、処理2_3、...をさらに関数で記述
}
...
...
```

自作の関数　まとめ

- 関数
  - 組み込みの関数 — document.write(), alert(), ...
  - 自作の関数

```
function doSomething(引数1, 引数2, ...) {
  let x = 引数1 ...引数2
  let y = …
  ...
  return y;
}
```

何らかの処理
入力 → 関数 → 値を返す
引数で指定（ない場合もあり）
（ことが多い）

TOPIC

字下げ（インデント）と可読性（よみやすさ）

TOPIC 字下げ（インデント）と可読性

```
function おみくじを引く() {
    let x = Math.random();
    if (x < 0.3) {
        x = "凶";
    }
    else if (x < 0.6) {
        x = "小吉";
    }
    else {
        x = "大吉";
    }
    return x;
}
```

字下げ（indent）によって範囲を明確に。同じレベルの処理をわかりやすくする。

色々な流儀
・2文字分　←おすすめ
・4文字分
・タブ1個分

TOPIC 字下げ（インデント）と可読性

```
function おみくじを引く() {
    let x = Math.random();
    if (x < 0.3) {
        x = "凶";
    } else if (x < 0.6) {
        x = "小吉";
    } else {
        x = "大吉";
    }
    return x;
}
```

色々な流儀
・{ else if　1行に

TOPIC 字下げ（インデント）と可読性

```
function おみくじを引く() {
    let x = Math.random();
    if (x < 0.3) {
x = "凶";
 }
       else if (x < 0.6) {
     x = "小吉";
    }
    else {
        x = "大吉";
        }
    return x;
}
```

こんなふうに書いてもエラーにはならない。ただし読みにくい。

タイマー

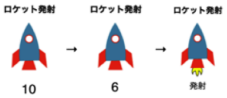
一定時間ごとの繰り返し

setInterval
clearInterval

5. タイマー（一定時間ごとの繰り返し）

課題　ロケット打ち上げ

ロケットの画像を表示して、その下にカウントダウンする数字を表示して打ち上げの様子を真似よ



サブ課題　カウントダウン

10から0までカウントダウンせよ

10 9 8 7 6 5 4 3 2 1 0 発射！

サブ課題　カウントダウン

```
let カウンタ = 10;                          第1引数 関数   第2引数 時間間隔
let タイマーID = setInterval(カウントダウン, 1000);
                                   1秒（1/1000*1000）ごとに繰り返す
function カウントダウン() {
  if (カウンタ >= 0) {
    document.write(カウンタ + " ");  // 連結
    カウンタ--; // 自分を1減らす（算術演算子）
  }
  else {
    document.write("発射！");
    clearInterval(タイマーID); // タイマー停止
  }
}
```

setInterval()が返した値（タイマーID）を覚えておくと、これを使って終了できる

サブ課題　カウントダウン

```
let カウンタ = 10;
let タイマーID =
   setInterval(カウントダウン,1000);

function カウントダウン() {
  if (カウンタ >= 0) {
    document.write(カウンタ + " ");
    カウンタ--; // 自分を1減らす
  }
  else {
    document.write("発射！");
    clearInterval(タイマーID);
  }    // タイマー停止
}
```

表示	カウンタ
	10
10_	9
10_9_	8
10_9_8_	7
10_9_8_7_	6
10_9_8_7_...	...
10_9_...2_	1
10_9_...2_1_	0
10_9_...2_1_0_	-1
10_9_...1_0_発射！	

サブ課題　カウントダウン

```
let カウンタ = 10;                          第1引数 関数   第2引数 時間間隔
let タイマーID = setInterval(カウントダウン, 1000);
                                   1秒（1/1000*1000）ごとに繰り返す
function カウントダウン() {
  if (カウンタ >= 0) {
    document.write(カウンタ + " ");  // 連結
    カウンタ--; // 自分を1減らす（算術演算子）
  }
  else {
    document.write("発射！");
    clearInterval(タイマーID); // タイマー停止
  }
}
```

- setInterval()が返した値（タイマーID）を覚えておくと、これを使って終了できる
- 複数のタイマーを使うときもあるので止めるものを指定

無名（匿名）関数

```
let カウンタ = 10;          1回しか使わないのなら名前は不要
let タイマーID = setInterval(   「無名（匿名）関数」を引数に渡す
  function() {                                          第1引数
    if (カウンタ >= 0) {
      document.write(カウンタ + " ");  // 連結
      カウンタ--; // 自分を1減らす（算術演算子）
    }
    else {
      document.write("発射！");
      clearInterval(タイマーID); // タイマー停止
    }
  }
, 1000);   第2引数
```

```
let タイマーID = setInterval(カウントダウン, 1000);
function カウントダウン() { ... }
```

タイマー　まとめ

```
let カウンタ = 10;          タイマー呼び出し
let タイマーID = setInterval(カウントダウン, 1000);
                                   関数を引数に渡す
```

関数の定義

```
function カウントダウン() {
  if (カウンタ >= 0) {
    document.write(カウンタ + " ");  文字列の連結
    カウンタ--;  算術演算子（自分から1を引く）
  }
  else {
    document.write("発射！");
    clearInterval(タイマーID);  タイマー終了
  }
}
```

```
let タイマーID = setInterval(カウントダウン(), 1000);
```

関数（の定義）を渡すときには()はつけない

配列

表計算ソフトのセルのように
まとめて記憶する

x = [a, b, c, ...]

「登場人物」

- ✦関数（組み込み）
- ✦分岐（if）
- ✦変数（let）
- ✦繰り返し（for）
- ✦演算子（+, -, <, <=, ...）
- ✦自作の関数
- ✦配列

Part I

- ✦オブジェクト指向（プロパティ, メソッド）
- ✦イベント

6. 配列

課題　漢数字でカウントダウン

カウントダウンを漢数字で行え

５４３２１０発射！ 発射！

フジテレビONE「鎧美女」より

TBS「Count Down TV」より

6. 配列

課題　漢数字でカウントダウン

カウントダウンを漢数字で行え

```
let 漢数字 = ["零", "壱", "弐", "参", "肆", "伍"];
let カウンタ = 5;
let タイマーID = setInterval(カウントダウン, 1000);


function カウントダウン() {
  if (カウンタ >= 0) {
    document.write(漢数字[カウンタ]+ " ");
    カウンタ--;
  }
  else {
    document.write("発射！");
    clearInterval(タイマーID);
  }
}
```

伍 肆 参 弐 壱

配列　まとめ

✦配列 —— 添字（index）が整数

```
let 漢数字 = ["零", "壱", "弐", "参", "肆", "伍"];
```

表記	値
漢数字[0]	"零"
漢数字[1]	"壱"
漢数字[2]	"弐"
漢数字[3]	"参"
漢数字[4]	"肆"
漢数字[5]	"伍"

```
// 漢数字.length で大きさ
// この配列の場合 → 6
```

表記	値
会員名簿[0]	"東野ナカ"
会員名簿[1]	"平野絢香"
会員名簿[2]	"家出オレ"
会員名簿[3]	"武田信玄"
会員名簿[4]	"月野元"
会員名簿[5]	"真田幸村"
会員名簿[6]	"松ひく子"

```
let 会員名簿 = ["東野ナカ", "平野絢香", ..., "松ひく子"];
// 会員名簿.length → 7
```

「登場人物」

- ✦関数（組み込み＋自作）
- ✦分岐（if）
- ✦変数（let）
- ✦演算子（+, -, <, <=, ...）
- ✦繰り返し（for、setinterval）
- ✦配列
- ✦オブジェクト指向（プロパティ, メソッド）
- ✦イベント

今日の時間割

基本概念1　関数　分岐　ループ

基本概念2　自作関数　タイマー　配列

オブジェクト指向
イベント

まとめ

JavaScriptで学ぶ プログラミング入門　丸１日コース PART II

- ✦オブジェクト指向とJavaScript
- ✦ブラウザオブジェクトの利用
- ✦イベント

今日の時間割

基本概念1　関数　分岐　ループ

基本概念2　自作関数　タイマー　配列

オブジェクト指向
イベント

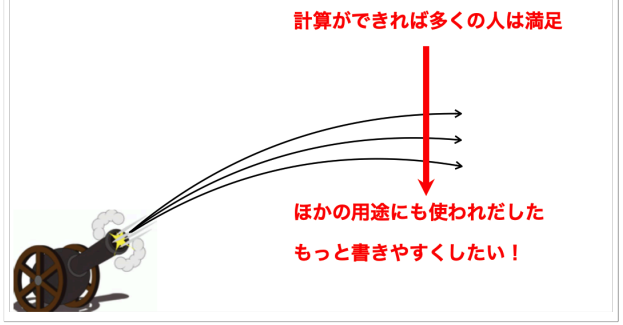
まとめ

オブジェクト指向と JavaScript

「オブジェクト指向」とはどのような考え方なのでしょうか
JavaScriptとどのような関係があるのでしょうか

8. オブジェクト指向とJS

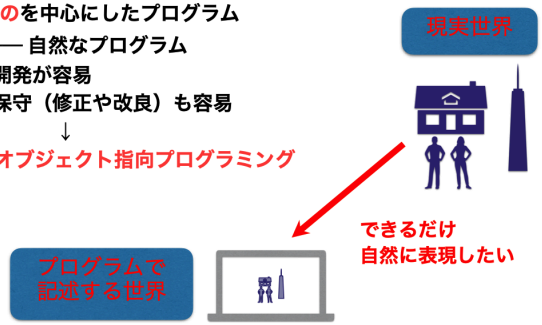
✦ 最初は数値計算が主



オブジェクト指向の考え方

✦世の中はもの（object）でできている
✦ものを中心にしたプログラム
—— 自然なプログラム
- 開発が容易
- 保守（修正や改良）も容易

↓
オブジェクト指向プログラミング



オブジェクト指向の考え方

✦世の中はもの（object）でできている
✦ものを中心にしたプログラム
—— 自然なプログラム
- 開発が容易
- 保守（修正や改良）も容易

↓
オブジェクト指向プログラミング
- 1960年代から研究（Simula, Smalltalk）
- 1990年頃から一般に広まる
- 最近の言語 —— ほとんどオブジェクト指向に対応
- JavaScriptもオブジェクト指向の機能をもつ

オブジェクト指向の考え方

✦プログラムで扱うもの（オブジェクト）を２つの側面から捉える
1. どんな性質をもつか
2. どんな動作をするか

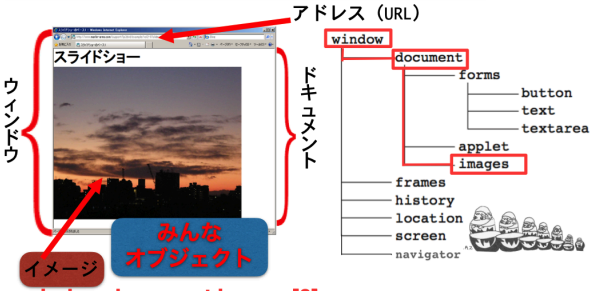
✦たとえばテレビを扱うなら

• 性質	変数で表現→プロパティ
・何インチか	tv1.サイズ
・録画機能はあるか	tv1.録画できる
・HDMIポートはいくつ	tv1.HDMIポート数
• 動作	関数で表現→メソッド
・チャンネルを変える	tv1.チャネルを選ぶ(7)
・ボリュームを変える	tv1.音量をセットする(22)
・入力先を換える	tv1.入力ポートを選ぶ(HDMI1)

JSのオブジェクトの定義

```
let tv1 = { サイズ: 32,  //プロパティの集まり
            色: "ブラック",
            音量: 24,
            音量をセット: function (n) {
              this.音量 = n;
            } // 関数もプロパティの一種
          }
...
if (tv1.音量 > 最大音量) {
  tv1.音量をセット(10);
  ...
}
```

ブラウザのオブジェクト



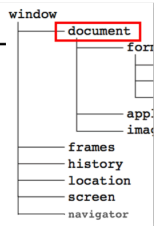
window.document.images[0]

ブラウザの各項目に対応するオブジェクトが用意されている
windowオブジェクトをトップとする階層構造（入れ子）

Windowオブジェクト

• どんな性質を持つか	変数で表現→プロパティ
• 高さ（height）	window.innerHeight
• 幅（width）	window.innerWidth
• 中身は？	window.document
• どんな動作をするか	関数で表現→メソッド
• ダイアログボックス表示	window.alert()
• 入力を受け取る	window.prompt(x, y)
• タイマー	window.setInterval(関数, 時間)

Documentオブジェクト



✦ メソッド
- document.write()　HTMLコードを書き出す
  window.document.write() と書いてもOK
- document.getElementById("abc")
  abcのIDをもつ部分のオブジェクトを得る

```
let x = document.getElementById("abc");
x.innerHTML —— HTMLコード（●●●）がわかる
```

```
<div id="abc"> <!-- HTML -->
  ●●●
</div>
```

x.innerHTML = "▲▲▲"; // どうなるでしょう？

// document.getElementById("abc").innerHTML = ... と同じ

```
<div style="width: 320px; ...">
  ロケット発射<br>
  <img src="pictures/rocket1.png" style="...">
  <div id="counterArea"></div>
</div>

<script type="text/javascript">
let カウンタ = 10;
let タイマーID = setInterval(カウントダウン, 1000);

function カウントダウン() {
  if (カウンタ>=0) {
    document.getElementById("counterArea").innerHTML = カウンタ;
    カウンタ--; // 次に備える
  }
  else {
    clearInterval(タイマーID); // タイマー停止
    document.getElementById("counterArea").innerHTML = "発射！";
    document.images[0].src = "pictures/rocket2.png";
  }
}
</script>
```

ロケット発射

document.writeの代わりに使える

● 次のような構文を使えば、動的に追加できる

```
let html="";
for (let i=0; i<10; i++) {
  html +=`<img src="pictures/picture00${i}.jpg">`;
}
document.getElementById("xxx").innerHTML = html;
```

●document.writeは非効率！ → 表示が遅くなってしまう

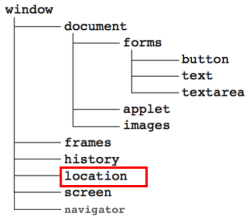
実践では（ほとんど）使われない
練習には使ってもOK
初心者にわかりやすい！ ←この講座で使っている理由

●JSの主用途はイベント処理などの「動的な」処理。
「静的な」コードはサーバ側（node.js、PHP、...）で出せる。

Locationオブジェクト

プロパティ

location.href
  URL（代入も可能→任意のURLを強制的に表示）
location.hostname
  URLのホスト名部分
location.protocol
  URLのプロトコル部分
location.host
  URLのホスト名とポート
...



オブジェクト指向とJavaScript　まとめ

- オブジェクト指向プログラミング
  - わかりやすいプログラムを書くために生まれた
  - オブジェクト = 関連する変数+関数（プロパティ+メソッド）の集まり
    JSの場合は公式にはプロパティがメソッドを含むが、関数については「メソッド」と呼ぶことも多い
- JSとオブジェクト
  - ブラウザオブジェクト —— windowをトップとする階層構造で、いろいろな機能を提供

オブジェクト指向とJavaScript　まとめ

- ブラウザの内容はすべてオブジェクトで表現される
  - ブラウザ内はプログラマーにとっての「キャンバス」
  - アイディア次第で何を書く（描く）こともできる
  - 動かすこともできる
  - ブラウザオブジェクトのプロパティ（変数）やメソッド（関数）を利用

ブラウザオブジェクトの利用

ブラウザオブジェクトでどんなことができるか例を見てみましょう

9. ブラウザオブジェクトの利用

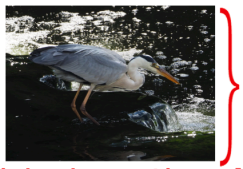
課題　スライドショー

画像を順に表示するスライドショーを作れ

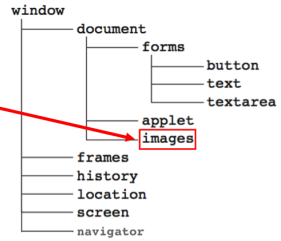
- 手順
  - 画像ファイルを用意
    picturesフォルダの下に
    picture000.jpg, picture001.jpg, picture002.jpg,
    picture003.jpg, ...
  - 順に画像を読み込んで表示
    Documentオブジェクトを使って画像を更新
  - ベースとなるHTMLコード

```
<body>
  <h1>スライドショー</h1>
  <img src="pictutres/picture000.jpg"
       style="width: 640px;">
</body>
```

Images の利用



window.document.images[0]
1番目の画像なので [0]
2番目の画像 なら[1]



document.images[0].src　→ pictures/picture000.jpg
ソース（ファイル名）がわかる
document.images[0].src = "pictures/picture001.jpg"
代入もできる　→　画像が変わる

タイマーで定期的に画像を更新

Windowのメソッド

```
let 画像番号 = 1;
let タイマーID =
        setInterval(画像を表示, 1000*2);

function 画像を表示() {
  let ファイル名 = `pictures/picture00${画像番号}.jpg`;
  document.images[0].src = ファイル名;
  画像番号++;
  if (5 <= 画像番号) {
    clearInterval(タイマーID);
  }
  //  return;
}
```

引数に関数を渡す
関数　時間
2000でもOK。
２秒ごとに
画像を表示()
を繰り返す

この条件でタイマーを終了

setInterval()が返した値を覚えておくと、これを使って終了できる

ブラウザオブジェクトの利用　まとめ

- JavaScriptとオブジェクト
  - ブラウザに表示される内容はオブジェクト
    たとえば1番目の画像は window.document.images[0]
- タイマー（復習）
  タイマーID = setInterval(画像を表示, 1000*2);
  clearInterval(タイマーID);

イベント

「イベント」に反応する
処理の方法をみましょう

今日の時間割

基本概念1　関数　分岐　ループ
基本概念2　自作関数　タイマー　配列

オブジェクト指向
イベント

まとめ

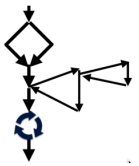
10. イベント

課題　Photo Gallery

マウスを写真の上に持っていくとその写真が拡大表示されるPhoto Galleryを作れ

- 「イベント（event）」を処理する
  - マウスが画像の上に移動した、離れた
  - マウスをクリックした
  - return（enter）キーを押した
  - ファイルを読み込んだ

プログラムの実行順序



- 上から下に順番に実行される（大原則）
- 関数 —— 別の部分が実行されてまた戻る
- 分岐 —— 一部だけが実行される
- ループ —— 繰り返される
- イベント —— 何かが起こると突然行われる
  「罠」を仕掛けておく ——イベントハンドラー
  - 読み込み時に実行できるものは実行し、イベント処理は罠を仕掛ける
  - 罠が仕掛けられたイベントが起こるのを待つ
  - 該当するイベントが起こる → イベント処理

イベント

- 何かが起こると突然行われる
  →罠を仕掛ける

```
obj.addEventListener("mouseover", function() {
   // マウスが上に来た時の処理
});
obj.addEventListener("mouseout", function() {
   // マウスが外に出た時の処理
});
obj.addEventListener("click", function() {
   // クリックした時の処理
});
...
```

Photo Gallery

```
<img src="picture000.jpg" style="width: 120px;">
<div id="pictureFrame">
  <img src="picture000.jpg"
                  style="width: 480px;">
</div>
```

```
document.images[0].addEventListener("mouseover", function() {
  document.getElementById("pictureFrame").innerHTML =
    `<img src="pictures/picture000.jpg" style=...`;
});                  // イベントハンドラ（罠）を定義しておく

document.images[0].addEventListener("mouseout", function() {
  document.getElementById("pictureFrame").innerHTML ="";
});
```

Photo Gallery

```
<img src="picture000.jpg" style="width: 120px;">
<div id="pictureFrame">


</div>

document.images[0].addEventListener("mouseover", function() {
  document.getElementById("pictureFrame").innerHTML =
    `<img src="pictures/picture000.jpg" style=...`;
});                  // イベントハンドラ（罠）を定義しておく

document.images[0].addEventListener("mouseout", function() {
  document.getElementById("pictureFrame").innerHTML ="";
});
```

イベント　まとめ

- 「イベント」の処理の例

  罠を仕掛けておく —— イベントハンドラー

```
obj.addEventListener("mouseover", 【関数定義】);
obj.addEventListener("mouseout", 【関数定義】);
```

- カーソルが上に移動した、離れた mouseover, mouseout
- クリックした click
- 送信ボタン（returnキー）を押した submit
- ファイルを読み込んだ load
- キー入力した keydown, keypress, keyup
- ...

今日のまとめ　PART I

プログラミングに共通の概念

```
let html="";  変数 代入
ループ for (let i=0; i<10; i++) {  論理演算子 算術演算子
  html +=`<img src="pictures/picture00${i}.jpg">`;
}                                  (テンプレートリテラル)
document.getElementById("xxx").innerHTML = html;

let 漢数字 = ["零", "壱", "弐", "参"];  配列
//  データをまとめて記憶←表計算のセル
let カウンタ = 10;   (タイマー呼び出し)
let タイマーID = setInterval(カウントダウン, 1000);
                                   関数引数
function カウントダウン() {
  自作の関数 分岐 if (カウンタ >= 0) {
    document.write(漢数字[カウンタ]+ " ");
    カウンタ--;  算術演算子   文字列連結演算子
  }
  else {
    document.write("発射！");
    clearInterval(タイマーID);  (タイマー終了)
  }
}
```

今日のまとめ　PART II

- オブジェクト指向プログラミング
  - わかりやすいプログラムを書くために生まれた
  - JavaScriptでブラウザ内の表示を自由に変えられる
- ブラウザオブジェクトの利用
  - たくさんのメソッドとプロパティが用意されている
- イベント
  - イベント処理の方法 —— イベントハンドラ
  - スタイル（CSS）との連携

プログラミング言語と英語

- 意味や意図を相手に伝えるための道具
  - 英語 — 人間との対話
    - いろいろなことを柔軟に行う
    - 指示は適当で大丈夫 →自然言語
  - JavaScript — コンピュータへの依頼
    限られたことを凄まじい速度で行う
    - 数の計算 computer
    - 文字列（記号）の処理
    - 指示は厳密に記述 →プログラミング言語

プログラミングに共通の概念

✦フロー制御 —— 実行の順番の制御 →「アルゴリズム」

- 書かれている順に実行するのが大原則
- 関数 - function ← 何かの処理をする箱
  組み込みの関数 alert(), document.write(), ...
  自作の関数 function xxx(引数) { ... }
- 分岐（選択肢）
  if (条件) {...} else if (条件2) {...} ... else { ... }
- 繰り返し ←強力

プログラミングに共通の概念

✦フロー制御 —— 実行の順番の制御 →「アルゴリズム」

- 書かれている順に実行するのが大原則
- 関数 - function ← 何かの処理をする箱
  組み込みの関数 alert(), document.write(), ...
  自作の関数 function xxx(引数) { ... }
- 分岐（選択肢）
  if (条件) {...} else if (条件2) {...} ... else { ... }
- 繰り返しとタイマー - for, setInterval()
- イベント - 罠を仕掛けておく。条件が整うとFire!!

プログラミングに共通の概念

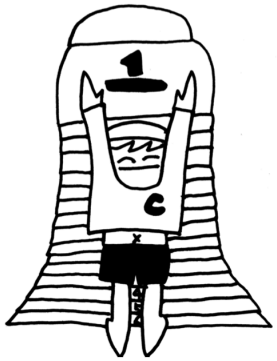
✦データ構造 —— データの記憶手法

- 変数 let —— 記憶場所。varは旧式なので...
- 配列 - let x = [a, b, c,...] ←まとめて記憶
- オブジェクト ←「もの」を表現
  xx.yy で オブジェクトxxのyyプロパティを表す
  JSでは「名前:値」が集まったもの
  「値」としては他のオブジェクトや関数も含まれる

✦演算子（オペレータ）

- +, -, *, /, %（余り）
- x<y, x>y, x<=y, x>=y, x===y, x!==y, !x

ひとまず
お疲れ様でした



実習

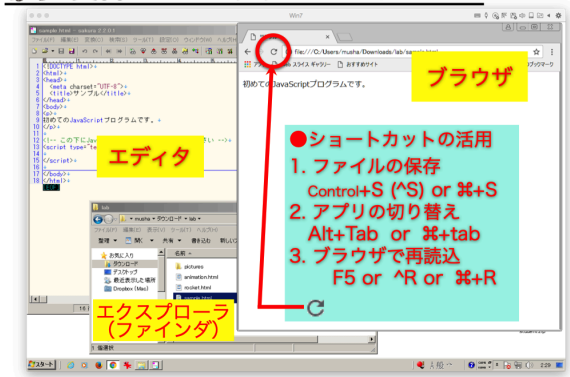
実習編のさわり

- 「練習問題」と「プレゼンのスライド」参照
- 登場した概念の復習（説明あり）
  説明が「難しい！」と思った人も大丈夫！
  手を動かしてみてください！

ダウンロードはこちら→

www.musha.com/sc/jsut

実習 —— おすすめのレイアウト



今日の講座

✦「文法の概要」と「練習の仕方の例」
✦「感じ」をつかむ。まず「読み方」を覚える
✦細かい点は資料を見ながら「コード」を書いていくうちに自然に覚える
今日の実習 +（自習 or「実習編」）
✦意識的に覚える必要は（あまり）ない
✦重要な概念は繰り返し登場するので、そのたびに復習すればよい
✦実習をすれば徐々に「納得」できる

7. デバッグ

- 一発で動くことは滅多にない
- プログラムを修正しては、再実行（ブラウザに読み込んで確認）を繰り返す。忍耐が必要

- デバッグ（debug） - プログラムがうまく動かないときに、動くようにする作業
  - ➡デバッグの由来　　de（取る）bug（虫）
    - https://gigazine.net/news/20120910-first-computer-bug/
    - https://ja.wikipedia.org/wiki/継電器

- 慣れないと自分で見つけるのは難しい

「"use strict";」の指定

- 冒頭に指定しておくと、未定義の変数がエラーになる

```
"use strict";
let x = 3;
let y = x*12;
X = y+10;     // ←エラー → スペルミスがわかる
alert(x);
```

Uncaught ReferenceError: X is not defined

「コンソール」にエラーが表示される

- 指定しておかないと、変数定義してなくても使えてしまう

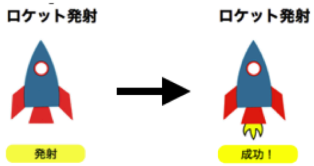
```
let x = 3;
let y = x*12;
X = y+10;  // ←エラーにならない。-> おかしな動作
alert(x);  // 3が表示される
```

デバッグ

- コンソール – 「デバッグ」の道具
  - 右クリック → [検証] → [Console] で表示
    （ショートカット：Ctrl+Shift+J or ⌘+option+J）
- エラーがある場合「コンソール」に表示される
  - メッセージを読んで理解を試みましょう
    （そのうち意味がわかってくる）
  - 行番号→どこにエラーがあるかが（だいたい）わかる

4 イベントの処理

4.1 ロケットの画像の下に「発射」ボタンを置き、ボタンを押したらロケットの画像を変えて、発射したように 見せよ



練習問題4 イベントの処理

rocket.html

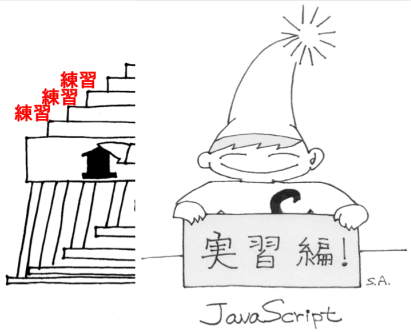
```
<style>
   #pict { width: 160px; }
   #button {
        font-size: 18pt; background-color: yellow; width: 180px;
        text-align: center; border-radius: 15px;
        cursor: pointer; /* カーソルを手の形に変える */
        }
</style>
</head>
<body>
<h1>ロケット発射</h1>
  <img id="pict" src="pictures/rocket1.png">
  <div id="button">発射</div>
<script type="text/javascript">
"use strict";
document.getElementById("button").addEventListner("click",
  function() {
    document.getElementById("●●").src =
                        "pictures/rocket2.png"; // 画像を変える
    document.getElementById("●●").innerHTML = "成功！";
  });
</script>
```

rocket.html

```
<style>
   #pict { width: 160px; }
   #button {
        font-size: 18pt; background-color: yellow; width: 180px;
        text-align: center; border-radius: 15px;
        cursor: pointer; /* カーソルを手の形に変える */
        }
</style>
</head>
<body>
<h1>ロケット発射</h1>
  <img id="pict" src="pictures/rocket1.png">
  <div id="button">発射</div>
<script type="text/javascript">
"use strict";
document.getElementById("button").addEventListner("click",
  function() {
    document.getElementById("pict").src =
                        "pictures/rocket2.png"; // 画像を変える
    document.getElementById("●●").innerHTML = "成功！";
  });
</script>
```

rocket.html

```
<style>
   #pict { width: 160px; }
   #button {
        font-size: 18pt; background-color: yellow; width: 180px;
        text-align: center; border-radius: 15px;
        cursor: pointer; /* カーソルを手の形に変える */
        }
</style>
</head>
<body>
<h1>ロケット発射</h1>
  <img id="pict" src="pictures/rocket1.png">
  <div id="button">発射</div>
<script type="text/javascript">
"use strict";
document.getElementById("button").addEventListner("click",
  function() {
    document.getElementById("pict").src =
                        "pictures/rocket2.png"; // 画像を変える
    document.getElementById("button").innerHTML = "成功！";
  });
</script>
```



実習編

- 今日の復習（+α）をしながら、色々な問題を解く
  - α ≒ 定数　フォームの処理　ライブラリ
    デバッガ
- 今日、曖昧模糊でも、実習編で徐々に解決！
- 手を動かして問題を解いていると、徐々にわかる
- 最初はなかなかエラーを自分では解決できないので、教えてもらえる環境がおすすめ
- 複数回受講可能

演習問題の解答例や参考資料

演習問題の解答例

https://www.musha.com/sc/jsia

- 演習問題の解答例
- 資料へのリンク

お疲れ様でした