

JavaScriptプログラミング 明快入門

JavaScript Programming: A Crystal Clear Introduction

サポートページ

www.musha.com/1

マーリンアームズ株式会社

武舎広幸

1



ライブトレーニングのサポートページ

www.musha.com/1

- 例題データのダウンロード
- 例題の実行
- その他の資料

www.musha.com/9

• スライド (1ページ9枚)

www.musha.com/2

• 例題ページ

2



自己紹介

武舎 広幸

www.marlin-arms.co.jp

- ◆ 翻訳ソフトの開発 (機械翻訳)
- ◆ 書籍の翻訳 (人間翻訳)



3



自己紹介

武舎 広幸

www.marlin-arms.co.jp

- ◆ 翻訳ソフトの開発 (機械翻訳)
- ◆ 書籍の翻訳 (人間翻訳)
- ◆ プログラミング講座

4



プログラミング講座

- ◆ 『初めてのJavaScript』



- ◆ DVD講座



5



プログラミング講座

- ◆ 対面講座 (2014年～)
 - ゼロから手作り
 - 2014年～月2回 (入門編+実習編)
 - のべ2000人以上受講



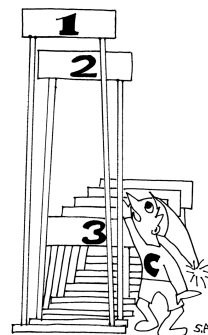
6



なぜ初心者向け講座？

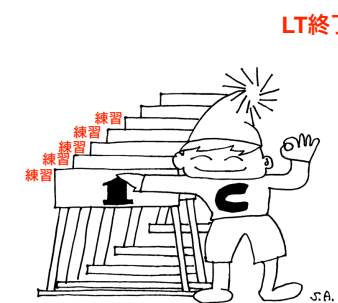
- ◆ そもそも プログラミングは楽しい！ (自身の体験)
- ◆ 「もの」を作る喜び。バーチャルな「もの」
cf. DIY、裁縫 小説、作曲
- ◆ 理系の人が書いた理系の人向けの書籍が多い
→ 「普通の人」には難しい？
- ◆ 理系の人でなくてもプログラミングが好きになる (得意になる) 人は少なくない
- ◆ そういう人がスムーズにこの世界に入れるような
- ◆ いろいろな人がプログラミングの楽しみに触れられる

7



現在

8



LT終了時

9

なぜJavaScript

- ◆開発環境の準備がいらない
- ◆HTMLを使って比較的簡単に面白いことができる
 - ・画像を動かしたり変化させたり
 - ・動画も簡単に扱える
- ◆ウェブサイトを作りたければ（ほぼ）必須
- ◆エディタの操作やHTMLなども含めて、数時間で説明するのはちょっと無理？
→ ウェブページを作れる人

10

10

プログラミング講座

- ◆対面講座（2014年～）
 - ・ゼロから手作り
 - ・2014年～月2回（入門編+実習編）
 - ・のべ2000人以上受講
- ◆オンライン+動画+書籍（2020年～）
 - ・このライブトレーニングのベース



11

11

参考書

- ◆『実践JavaScript!』
 - ・今日+来週の内容をより詳しく解説
 - ・練習問題（100問以上）

www.musha.com/scjs



12

12



13

13

プログラミングはAIがやってくれる？

- ◆プログラミングを学んでも無駄だ！
- ◆AIが全部やってくれるからプログラミングを学んでも役に立たない！

NOT TRUE
少なくとも当面は
(あと数十年は?)

14

14

プログラミングはAIがやってくれる？

- ◆AIの解答が「正解」とは限らない
- ◆人間がしないような間違いをする
→ 細かなチェックが必要
 - ・AIは（本当の）「意味」を理解できない
 - ・我々の意図を（完全には）理解できない
- ◆AIを最大限に活用するにはプログラミングに関する（深い？）理解が必須
- ◆細かいことまで覚える必要はない
 - ・AIが助けてくれる

15

15

プログラミングはAIがやってくれる？

- ◆AIを活用してプログラム（システム）を作るためにはプログラミングを学ばなければならない
- ◆AIが助けてはくれるが、手を動かして、この世界に浸る時間を作らないとダメ！？
- ◆ある日突然「わかる」



16

16

プログラミングはAIがやってくれる？

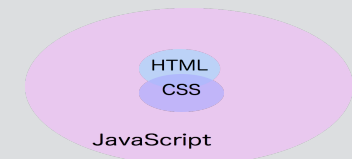
- ◆そもそもプログラミングは楽しい！
- ◆AIには思いつかないようなものを作ろう！

17

17

本日（と来週）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



18

18

19



今日の時間割 Timetable

Sec. 1 プログラミングってどんな感じのもの？

Sec. 2 プログラミングに登場する基本概念 (1)

- 2-1 関数 function
- 2-2 分岐 (変数、代入を含む) branching
- 2-3 繰り返し loop
- 2-4 演算子 operator

<休憩 break>

Sec. 3 実習 (簡単な問題) exercise

Sec. 4 プログラミングに登場する基本概念 (2)

- 4-1 自作の関数 (ユーザー定義関数) user-defined function
- 4-2 配列 array

まとめ+Q&A

20

20



第5回 来週の時間割 Next Week

JavaScript でHTML を操作する DOM を使った動的なWeb ページの作成

- Sec. 1 JavaScript の構文の概要（1 週間前の復習 + a）
- Sec. 2 オブジェクト指向はなぜ誕生したか
- Sec. 3 オブジェクト指向とJavaScript
＜休憩 10 分＞
- Sec. 4 タイマーとイベント処理
 - ・4-1 タイマー
 - ・4-2 イベント処理
- Sec. 5 実習（簡単な問題を一緒に解く）

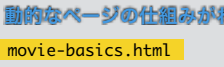
まとめ+Q&A

21

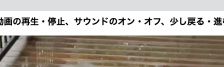


最終目標

動画の再生・停止、サウンドのオン・オフ、少し戻る・進む



動画の再生・停止、サウンドのオン・オフ、少し戻る・進む



ムービーギャラリー



22

The slide is titled "あととは" (And then) in red. It features a cartoon character wearing a pointed hat and shorts, climbing a tall stack of books. Each book has the text "楽しく練習" (Practice happily) written on it. A black arrow points upwards from the bottom of the stack. To the left of the stack are two book covers: one for "Java Script!" and another for "JavaScriptの教科書". Below the books, there is a small diagram showing a sequence of steps or a flowchart.

23

 ライブトレーニングのサポートページ

www.musha.com/1

- 例題データのダウンロード
- 例題の実行
- その他の資料

www.musha.com/9 www.musha.com/2

- スライド (1ページ9枚)
- 例題ページ

24

Section 1

プログラミングって
どんな感じのもの？

A small cartoon character with a lightbulb on its head, wearing a white shirt with a black '1' on it, and a small 'GA' logo at the bottom right.

25

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...

26

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

HTML+CSSがOK
→可能性大

我慢の時（長さは人による）

27



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）
- 英語の綴りを間違えると動かない
- 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

28

28



本日+来週の単語帳 (1) JSの単語帳

- ◆単語全体の「感じ」を覚える
- alert [ə'lɜ:t] or [ə'lɔ:t] - 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

29

29



本日+来週の単語帳 (2) JSの単語帳

- 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 [wɔːrniŋ] - 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 ←<div>タグ
- span - 及び、期間、間の距離 ←タグ

30

30



本日+来週の単語帳 (3)

- 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 [haɪt] (ハイト) - 高さ
- width [wɪð] (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

31

31



このライブトレーニング

- ◆「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
- ◆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
- ◆わからなくてもAIに聞ける（人間のほうが効果的だが、AIのほうが気楽！）

32

32



このライブトレーニング

- ◆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（とくに来週の話）
 - 画像を動かしたり変化させたり、動画を操作したり
- ◆JSを知っている方
 - 概念の整理
 - 他の人に教えるときの参考
- 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

33

33

Section 2

プログラミングに登場する 基本概念 (1)

- 2-1 関数
- 2-2 分岐（変数、代入）
- 2-3 ループ
- 2-4 演算子



S.A

34

2-1 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

35



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
- ◆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

36

36

JavaScriptの「コード」

hello.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>hello, world</title>
</head>
<body>
  <script>
    alert("こんにちは");
  </script>
</body>
</html>
```

本文

JavaScriptのプログラム

- <head>部にも書いてOK
- 別ファイルにも書ける
→当面は同じファイルに

37

ダイアログボックスの表示

```
<script>
alert("こんにちは");
</script>
```

行末の「;」は省略可
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」

38

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
");
document.write("こんばんは
");
parameter.html
3. プログラムは上から下に順番に実行される(原則)

39

プログラムは上から下に順番に実行される(原則)

40

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
");
document.write('おはようございます
');
document.write(`こんにちは<br>`);
自分でどこかに決める(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(``);
document.write(``);
parameter.html

41

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());
function1.html

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());
function2.html

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

42

2-1 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 — 0以上1未満の小数を返す
 - などなど、たくさんある

43

2-2 分岐

条件によって処理を変える

if文
変数
代入

44

2-2 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

45

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)

```

46

46

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」！

```

47

47

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)

```

48

48

変数の値の変化

```

let x = Math.random(); x 0.314
if (x < 0.3) { xの値を「参照」
  x = "凶";
}
else if (x < 0.6) { x "小吉"
  x = "小吉"; ← xに「代入」。同じxを使う
                ときletは書かない
}
else {
  x = "大吉";
}
document.write(x); x "小吉"

```

49

49

letで再度宣言してしまうと...

```

let x = Math.random(); x 0.314
if (x < 0.3) {
  let x = "凶";
}
else if (x < 0.6) { x "小吉"
  let x = "小吉";
}
else {
  let x = "大吉";
}
document.write(x); x 0.314

```

スコープ
内側の領域用の
xができる

使われずに
終わってしまう

50

50

変数の値の変化

```

let x = Math.random(); x 0.314
if (x < 0.3) { xの値を「参照」
  x = "凶";
}
else if (x < 0.6) { x "小吉"
  x = "小吉"; ← xに「代入」。同じxを使う
                ときletは書かない
}
else {
  x = "大吉";
}
document.write(x); x "小吉"

```

51

51

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);

```

処理1 処理2 処理3

52

52

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>
}

```

処理1 処理2 処理3 ... 処理n

53

53

分岐のパターン

◆if文 —— やるかやらないか

```

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>
}

```

処理1

54

54

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>
}

```

55

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>
}

```

56

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>
}

```

57

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

```

- (switch文)

58

プログラムは上から下に順番に実行される (原則)

59

分岐

60

TOPIC

コメント

61

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)

```

/*
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. その他なら「大吉」
*/

```

62

TOPIC

文と演算子

63

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、Java等では必須

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。

64

なぜ「;」(セミicolon) 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

65

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がyより大きければ真 (以下)
x <= y xがyと同じか小さければ真 (以下)
x >= y xがyと同じか大きければ真 (以下)
x === y xとyが等しければ真 (x==y)
x !== y xとyが等しくなければ真 (x!=y)

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

プログラムは形式的な英語

66

小休止

Stretch Break

67

2-3 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

68

2-3 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

```










```

◆JavaScript (プログラム) を使えば楽ができる

```

for (let i=0; i<8; i++) {
  document.write(`);
}

```

69

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(`);
}

```

(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
「」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	1
2	2
3	3
4	4
5	5
6	6
7	7

70

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(`);
}

```

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)

「」の文字列をdocument領域にwrite。
ただし...の中の\${i}の部分は計算 (評価) してから

```

i=0 → 
i++ → 
i++ → 
i++ → 
...
i++ → 

```

71

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(`);
}

```

i=0 → // 000.jpg ではない
i++ →
i++ →
i++ →
...
i++ →

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

72

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('');
}
```

◆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('  夢鶏湯が食べたい。<br>');
} // 夢鶏湯（サムゲタン）は韓国料理
```

loop2.html

夢鶏湯が食べたい。
夢鶏湯が食べたい。
夢鶏湯が食べたい。
夢鶏湯が食べたい。
...

コメント

73

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}. 夢鶏湯が食べたい。<br>');
}
```

1. 夢鶏湯が食べたい。
2. 夢鶏湯が食べたい。
...

1箇所を変えるだけで、すべてが変わる
何箇所もコピーする必要がない

◆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(
    );
}
```

loop3.html

...

74

2-3 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('');
}
```

① 初期設定; ② 条件; ③ 再設定; ④ 本体: 繰り返す作業

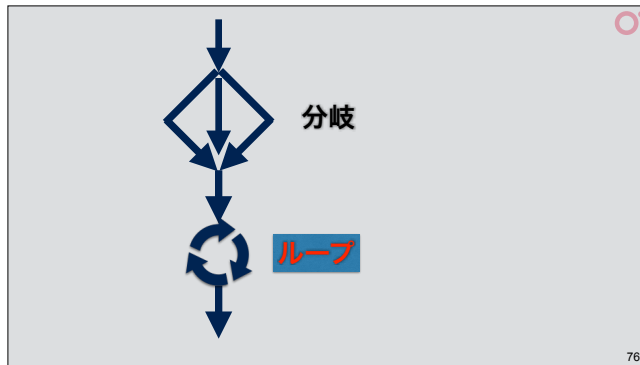
⑤

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

75



76

TOPIC

変数を使ってわかりやすく

77

どちらがわかりやすい?

loop3.html

```
for (let i=0; i<8; i++) {
  document.write('');
}
```

結果は同じ

```


...
```

let style = `style="width: 25%;";` // 「style」という変数

```
for (let i=0; i<8; i++) {
  let filename = `pictures/picture00${i}.jpg`;
  let imgTag = ``;
  document.write(imgTag);
}
```

78

TOPIC 変数を使ってわかりやすく

```
let style = `style="width: 25%;";` // 「style」という変数
for (let i=0; i<8; i++) {
  let filename = `pictures/picture00${i}.jpg`;
  let imgTag = ``;
  document.write(imgTag);
}
```

途中経過を一旦変数に記憶

style	style="width: 25%;"
filename	pictures/picture000.jpg
imgTag	

脳への負担が減る→間違いが減る
まあ、これくらいは覚えられる?

```


...

```

79

日本語の変数も使える わかりやすい?

```
let スタイル = `style="width: 25%;";` // 「スタイル」という変数
for (let i=0; i<8; i++) {
  let ファイル名 = `pictures/picture00${i}.jpg`;
  let 画像タグ = ``;
  document.write(画像タグ);
}
```

// 変数や関数の名前には日本語可

スタイル	style="width: 25%;"
ファイル名	pictures/picture000.jpg
画像タグ	

```


...

```

80

TOPIC

算術演算子

Computer (計算をするもの)
なので計算は得意

81

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
 全体は代入文

$x \leftarrow x * 10$;

82

82

演算子 (オペレータ)

◆おもしろい (便利な) 演算子 $+=$, $-=$, $*=$, $/=$

$x += 5$; $\Leftrightarrow x = x + 5$; どちらも現在の値に5を足す

x
1
6
6

$let\ x = 1$;
 $x += 5$;
 $alert(x)$;

◆もう1種類 $++$, $--$ 1を足す、1を引くは非常によく使う

$x++ \Leftrightarrow x += 1 \Leftrightarrow x = x + 1$
 $x-- \Leftrightarrow x -= 1 \Leftrightarrow x = x - 1$

83

83

TOPIC

文字列の連結とテンプレートリテラル

84

84

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	y	z
"初めての"	"JavaScript"	"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』

85

85

$+=$ で自分の後ろに追加

Loop4.html

```
let html = "";
for (let i=0; i<10; i++) {
  html += ``;
}
document.write(html)
```

i	html
0	
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">

86

86

テンプレートリテラル (可変部分付き文字列)

◆文字列`...`の中に $\${...}$ で囲んで変数や計算式を書ける

```
imgTag = ``;
document.write(`"${x+y}"`);
```

◆テンプレートリテラルの導入は比較的最近
以前は「 $+$ 」を使って連結

```
imgTag = ``;
```

「 $`$ 」と「 $`$ 」の組み合わせ $\rightarrow$ すごく大変だった

87

87

Section 3

実習



88

88

演算子 (オペレータ)

◆おもしろい (便利な) 演算子 $+=$, $-=$, $*=$, $/=$

$x += 5$; $\Leftrightarrow x = x + 5$; どちらも現在の値に5を足す

x
1
6
6

$let\ x = 1$;
 $x += 5$;
 $alert(x)$;

◆練習問題

$x -= 4$; $\Leftrightarrow x = x - 4$; $let\ x = 8$; $x -= 4$; $alert(x)$;
 $x *= 8$; $\Leftrightarrow x = x * 8$; $let\ y = 3$; $y *= 8$; $alert(y)$;
 $x /= 2$; $\Leftrightarrow x = x / 2$; $let\ z = 6$; $z /= 2$; $alert(z)$;

89

89

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

$x++ \Leftrightarrow x += 1 \Leftrightarrow x = x + 1$
 $x-- \Leftrightarrow x -= 1 \Leftrightarrow x = x - 1$

◆次の結果は?

x	y
4	--
4	10

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

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

90

90

課題 今日のラッキーナンバー その1

今日のラッキーナンバー (1~10) を表示せよ

```
let x = Math.random(); // 0 ≤ x < 1 (小数)
x *= 10 // x = x*10 (自分を10倍) // 0 ≤ x < 10 (小数)
x = Math.floor(x); // 切り下げ // 0 ≤ x ≤ 9 (整数)
x++; // 自分に1を足す // 1 ≤ x ≤ 10 (整数)
document.write(`今日のあなたのラッキーナンバーは${x}です`);
// 1行で書くとき
// document.write(`ラッキーナンバーは${Math.floor(Math.random()*10) + 1}です`);
// lucky-number-10-1.html.html
// lucky-number-10-2.html.html
```

◆別解

```
let x = Math.random(); // 0 ≤ x < 1 (小数)
x *= 10 // 0 ≤ x < 10 (小数)
x++; // 1 ≤ x < 11 (小数)
x = Math.floor(x); // 1 ≤ x ≤ 10 (整数)
document.write(`今日のあなたのラッキーナンバーは${x}です`);
// lucky-number-10-3.html.html
// document.write(`ラッキーナンバーは${Math.floor(Math.random()*10) + 1}です`);
```

91

今日の時間割 Timetable

Sec. 1 プログラミングってどんな感じのもの？

Sec. 2 プログラミングに登場する基本概念 (1)

- 2-1 関数 function
- 2-2 分岐 (変数、代入を含む) branching
- 2-3 繰り返し loop
- 2-4 演算子 operator

<休憩 break>

Sec. 3 実習 (簡単な問題) exercise

Sec. 4 プログラミングに登場する基本概念 (2)

- 4-1 自作の関数 (ユーザー定義関数) user-defined function
- 4-2 配列 array

まとめ+Q&A

92

Section 4

プログラミングに登場する 基本概念 (2)

4-1 自作の関数

4-2 配列



93

4-1 自作の関数 (ユーザー定義関数)

関数は自分でも作れる

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

94

4-1. 自作の関数

user-function1.html

◆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);
```

- 関数は定義だけでは実行されない
- 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

```
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
```

95

何回も呼び出す

user-function2.html

```
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箇所にとめておいたほうがデバッグが楽

  ...`の中で計算もできる
  関数を呼び出すこともできる

  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>`);
```

96

自作の関数 — 引数 user-function3.html

◆引数を指定

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

```
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>`);
}
```

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。

97

自作の関数 — 引数 user-function3.html

◆引数を指定

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

```
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>`);
}
```

仮引数 呼ばれたときに値が入る

98

自作の関数 — 引数 user-function4.html

◆別解

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

```
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) {
    ...
  }
}
```

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

99

97

98

99

数学の関数 vs. JSの関数

何らかの計算

入力 → 数学の関数 → 値 (戻り値) を返す

$f(x) = x^2 + 1 \rightarrow f(3) = 3^2 + 1 = 10$

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

100

以前よく使われた言語の場合

手続き (戻り値なし)
procedure (Pascal)
subroutine (Fortran)

関数 (戻り値あり)
function

入力 (引数)	JSの関数1	同じ入力には同じ戻り値
入力 (引数)	JSの関数2	戻り値あり
入力 (引数)	JSの関数3	同じ値を返すとは限らない
入力 (引数)	JSの関数4	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

101

C言語以降のほとんど (JS含む) の関数

関数
function

戻り値の有無に関係なし

入力 (引数)	JSの関数1	同じ入力には同じ戻り値
入力 (引数)	JSの関数2	戻り値あり
入力 (引数)	JSの関数3	同じ値を返すとは限らない
入力 (引数)	JSの関数4	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

102

自作の (ユーザー定義) 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

103

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
}
...
```

104

自作の関数 まとめ

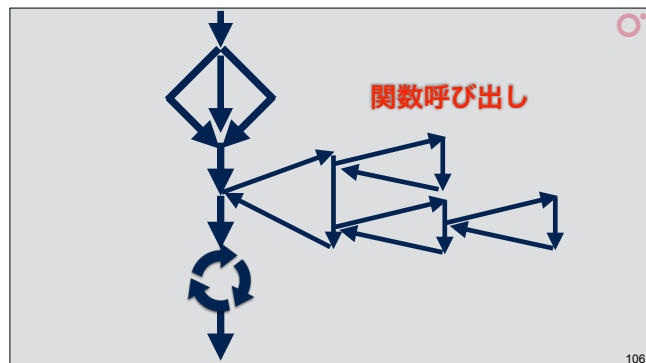
- ◆ 関数
 - ・ 組み込みの関数 — document.write(), alert(), ...
 - ・ 自作の関数 (ユーザー定義関数)

```
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 (ことが多い)
(引数で指定 (ない場合もあり))

105



106

課題 今日のラッキーナンバー その2

今日のラッキーナンバー (1~100) を表示せよ

```
let x = Math.random(); // 0 ≤ x < 1 (小数)
x *= 100; // x = x * 100 (自分を100倍) // 0 ≤ x < 100 (小数)
x = Math.floor(x); // 切り下げ。 // 0 ≤ x ≤ 99 (整数)
x++; // 自分に1を足す // 1 ≤ x ≤ 100 (整数)
document.write(`今日のあなたのラッキーナンバーは${x}です`);
```

- ◆ 別解

```
document.write(`今日のあなたのラッキーナンバーは${getRandomInt(100)+1}です`);

function getRandomInt(n) { // 0以上n未満の整数をランダムに得る
  return Math.floor(Math.random()*n);
}
```

lucky-number100-1.html

107

4-2 配列

表計算ソフトのセルのように
まとめて記憶する

$x = [a, b, c, \dots]$

108

4-2 配列

課題 ラッキーカラー

今日の「ラッキーカラー」を表示せよ（4色から選ぶ）

```

let colors = ["黒", "青", "赤", "白"];
let i = getRandomInt(4); // 4色ある
document.write(`今日のラッキーカラーは「${colors[i]}」です`);

function getRandomInt(n) { // 0以上n未満の整数をランダムに得る (0<n)
  return Math.floor(Math.random()*n);
}

```

array1.html

まとめて記憶する

表記	値
colors[0]	"黒"
colors[1]	"青"
colors[2]	"赤"
colors[3]	"白"

109

配列

課題 ラッキーカラー（色を増やす）

今日の「ラッキーカラー」を表示せよ（8色から選ぶ）

```

let colors = ["黒", "青", "赤", "白", "緑", "オレンジ", "灰色", "紫"];
let i = getRandomInt(8);
document.write(`今日のラッキーカラーは「${colors[i]}」です`);

function getRandomInt(n) { // 0以上n未満の整数をランダムに得る (0<n)
  return Math.floor(Math.random()*n);
}

```

array2.html

表記	値	表記	値
colors[0]	"黒"	colors[4]	"緑"
colors[1]	"青"	colors[5]	"オレンジ"
colors[2]	"赤"	colors[6]	"灰色"
colors[3]	"白"	colors[7]	"紫"

110

配列のサイズ

課題 ラッキーカラー（改良版）

今日の「ラッキーカラー」を表示せよ（8色から選ぶ）

```

let colors = ["黒", "青", "赤", "白", "緑", "オレンジ", "灰色", "紫"];
let i = getRandomInt(colors.length-1); // xxx.lengthで配列のサイズ
document.write(`今日のラッキーカラーは「${colors[i]}」です`);

function getRandomInt(n) { // 0以上n未満の整数をランダムに得る (0<n)
  return Math.floor(Math.random()*n);
}

```

array3.html

表記	値	表記	値
colors[0]	"黒"	colors[4]	"緑"
colors[1]	"青"	colors[5]	"オレンジ"
colors[2]	"赤"	colors[6]	"灰色"
colors[3]	"白"	colors[7]	"紫"

111

変更強くする

```

let colors = ["黒", "青", "赤", "白", "緑", "オレンジ", "灰色", "紫"];
let i = getRandomInt(7); // 色数が変わるたびに変更
document.write(`今日のラッキーカラーは「${colors[i]}」です`);

function getRandomInt(n) { // 0以上n未満の整数をランダムに得る (0<n)
  return Math.floor(Math.random()*n);
}

```

112

112

変更強くする

```

let colors = ["黒", "青", "赤", "白", "緑", "オレンジ", "灰色", "紫"];
let i = getRandomInt(colors.length-1); // 何色でも変更の必要なし
document.write(`今日のラッキーカラーは「${colors[i]}」です`);

function getRandomInt(n) { // 0以上n未満の整数をランダムに得る (0<n)
  return Math.floor(Math.random()*n);
}

```

113

- 他の部分への依存を少なくする
- 変えなくても済む（1箇所だけ変えるだけで済む）ように
- 「疎結合」にする

113

配列 まとめ

◆配列 —— 添字 (index) が整数

```

let colors = ["黒", "青", "赤", "白"];

```

114

表記	値
colors[0]	"黒"
colors[1]	"青"
colors[2]	"赤"
colors[3]	"白"

```

// colors.length で大きさ（長さ）
// この配列の場合 → 4

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

```

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"

114

TOPIC

字下げ（インデント）と可読性

115

115

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;
}

```

116

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

116

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;
}

```

117

色々な流儀

- ・{ else if 1行に

117



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;
}
```

こんなふうにも書いてもエラーにはならない。ただし読みにくい。

118

118

まとめ

Q&A



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

120

120



今日のまとめ

```
let html=""; 変数 代入
for (let i=0; i<10; i++) { 論理演算子 算術演算子
  html += ;
} 文字列連結演算子 (テンプレートリテラル)

let colors = ["黒", "青", "赤", "白"];
// データをまとめて記憶 ← 表計算のセル 配列

function おみくじを引く() {
  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
}
```

自作の関数 分岐

プログラミングに共通の概念

121

121



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
 - 組み込み関数 alert(), document.write(), ...
 - ユーザー定義 (自作) 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
 - イベント処理 タイマー



122

122



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
 - 配列 - let x = [a, b, c, ...] ← まとめて記憶
 - オブジェクト ← 「もの」を表現
- ◆演算子 (オペレータ)
 - +, -, *, /, % (余り)
 - x < y, x > y, x <= y, x >= y, x === y, x !== y, !x, &&, ||

123

123



今日の時間割 Timetable

- Sec. 1 プログラミングってどんな感じのもの?
- Sec. 2 プログラミングに登場する基本概念 (1)
 - 2-1 関数 **function**
 - 2-2 分岐 (変数、代入を含む) **branching**
 - 2-3 繰り返し **loop**
 - 2-4 演算子 **operator**
- <休憩 break>
- Sec. 3 実習 (簡単な問題) **exercise**
- Sec. 4 プログラミングに登場する基本概念 (2)
 - 4-1 自作の関数 (ユーザー定義関数) **user-defined function**
 - 4-2 配列 **array**
- まとめ+Q&A

124

124



来週の時割

JavaScript でHTML を操作する DOM を使った動的なWeb ページの作成

- Sec. 1 JavaScript の構文の概要 (1 週間前の復習 + a)
- Sec. 2 オブジェクト指向はなぜ誕生したか
- Sec. 3 オブジェクト指向とJavaScript
- <休憩>
- Sec. 4 タイマーとイベント処理
 - 4-1 **タイマー**
 - 4-2 **イベント処理**
- Sec. 5 実習 (簡単な問題を一緒に解く)
- まとめ+Q&A

125

125



最終目標

動的なページの仕組み (コード) が理解できる

movie-basics.html



movie-gallery.html



126

126