プログラミング

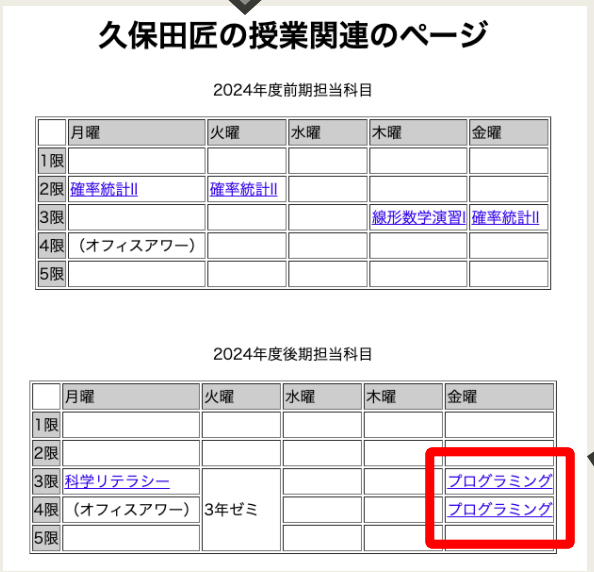
第12回
ユーザー定義関数

久保田 匠

[準備]授業資料にアクセス

いつもの作業

- 久保田の授業ホームページに資料がアップロードされている。
- まずは「愛教大 数学」と検索してみよう。



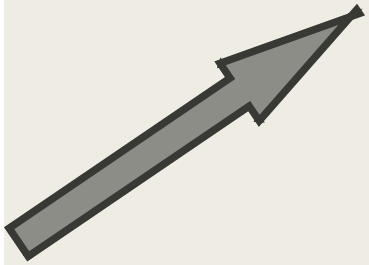
久保田匠の授業関連のページ

2024年度前期担当科目

	月曜	火曜	水曜	木曜	金曜
1限					
2限	確率統計II	確率統計II			
3限				線形数学演習I	確率統計II
4限	（オフィスアワー）				
5限					

2024年度後期担当科目

	月曜	火曜	水曜	木曜	金曜
1限					
2限					
3限	科学リテラシー	3年ゼミ			プログラミング
4限	（オフィスアワー）				プログラミング
5限					



プログラミング

	内容	資料	コード
第1回	いろいろなプログラミング言語 VSCode のインストール	●	Prog_01-1
第2回	Webページを構築する(HTML)	●	Prog_02-1
第3回	Webページの見栄えを整える(CSS)	●	Prog_03-1 Prog_03-2
第4回	JavaScriptに触れてみよう	●	Prog_04-1
第5回	変数と演算	●, ★	（なし）
第6回	条件文	●, ★	（なし）
第7回	[オンデマンド] 繰り返し(0)	●	（なし）
第8回	繰り返し(1)	●, ★	Prog_08-1
第9回	繰り返し(2)	●, ★	（なし）
第10回	オブジェクト	●, ★	（なし）
第11回	配列	●, ★	Prog_11-1
第12回	ユーザー定義関数	●, ★	Prog_12-1
第13回	イベントハンドラ	●, ★	
第14回	数式の表示(TeXについて)	●, ★	
第15回	学習アプリを開発してみよう	●	

[準備]コードの新規作成①

いつもの作業

- 授業用ホームページからサンプルコードをコピーしよう。

プログラミング

	内容	資料	コード
第1回	いろいろなプログラミング言語 VSCode のインストール	●	Prog_01-1
第2回	Webページを構築する(HTML)	●	Prog_02-1
第3回	Webページの見栄えを整える(CSS)	●	Prog_03-1 Prog_03-2
第4回	JavaScriptに触れてみよう	●	Prog_04-1
第5回	変数と演算	●, ★	（なし）
第6回	条件文	●, ★	（なし）
第7回	[オンデマンド] 繰り返し(0)	●	（なし）
第8回	繰り返し(1)	●, ★	Prog_08-1
第9回	繰り返し(2)	●, ★	（なし）
第10回	オブジェクト	●, ★	（なし）
第11回	配列	●, ★	Prog_11-1
第12回	ユーザー定義関数	●, ★	Prog_12-1
第13回	イベントハンドラ	●, ★	
第14回	数式の表示(TeXについて)	●, ★	
第15回	学習アプリを開発してみよう	●	

Prog_12-1

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="UTF-8">
    <title>Prog_12-1</title>
    <!-- 関数を定義するときは head部 を使います -->
</head>

<body>
    <!--
    演習のときに使う配列（適宜コピペして使ってください）
    let scores = [65, 81, 73, 52, 84];
    -->
</body>

</html>
```

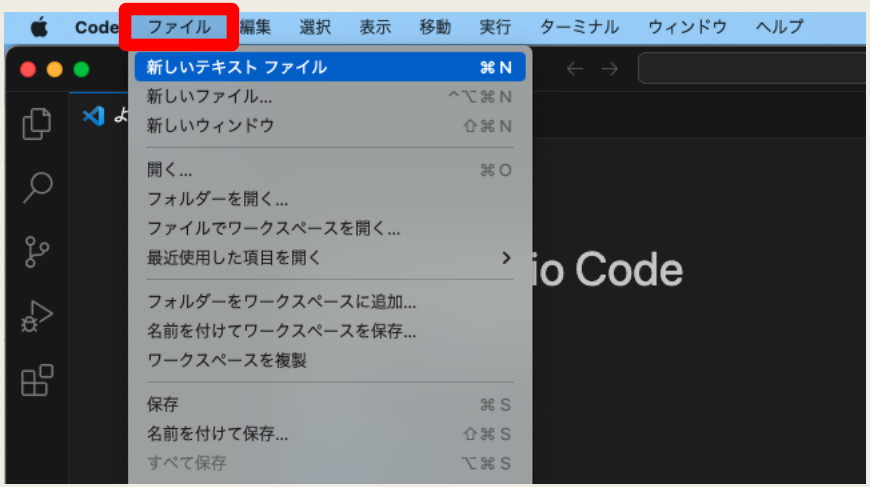
コピー

今日は「Prog_12-1」を選択してください。

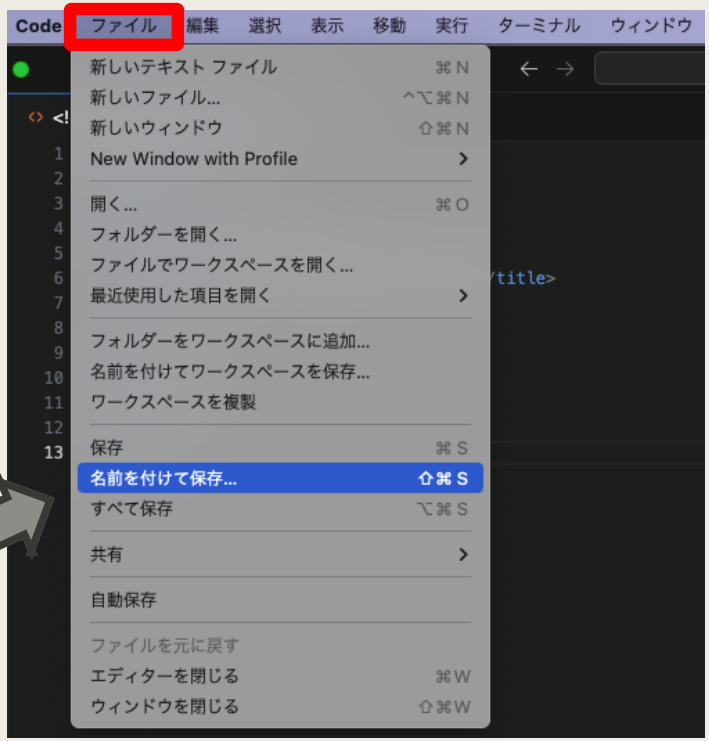
[準備]コードの新規作成②

いつもの作業

- VSCode を起動し「ファイル」から「新しいテキストファイル」を選択。
- そのあと、さきほどコピーした文書をペースト（Ctrl + V）して「名前をつけて保存」。



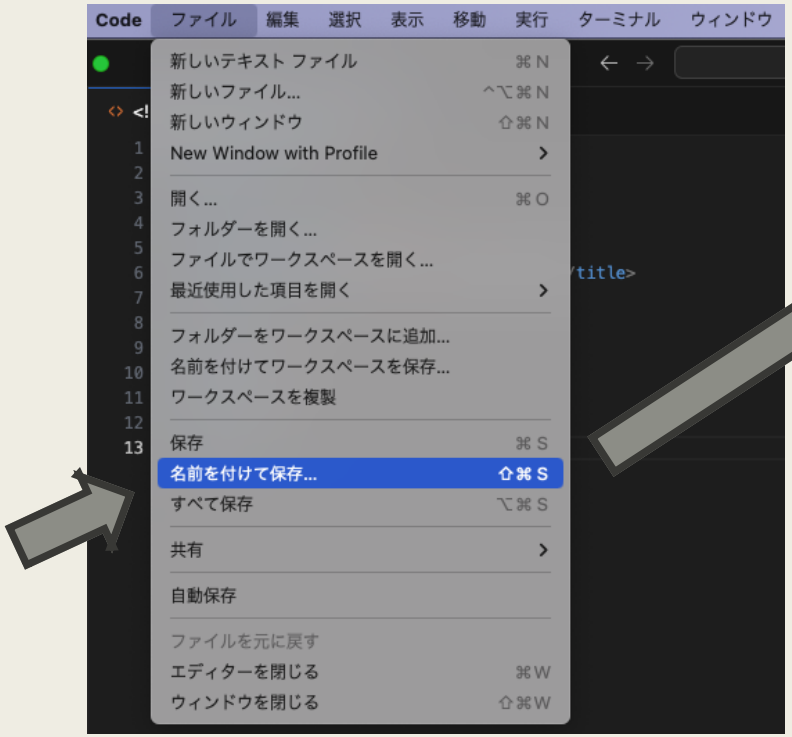
Ctrl + V

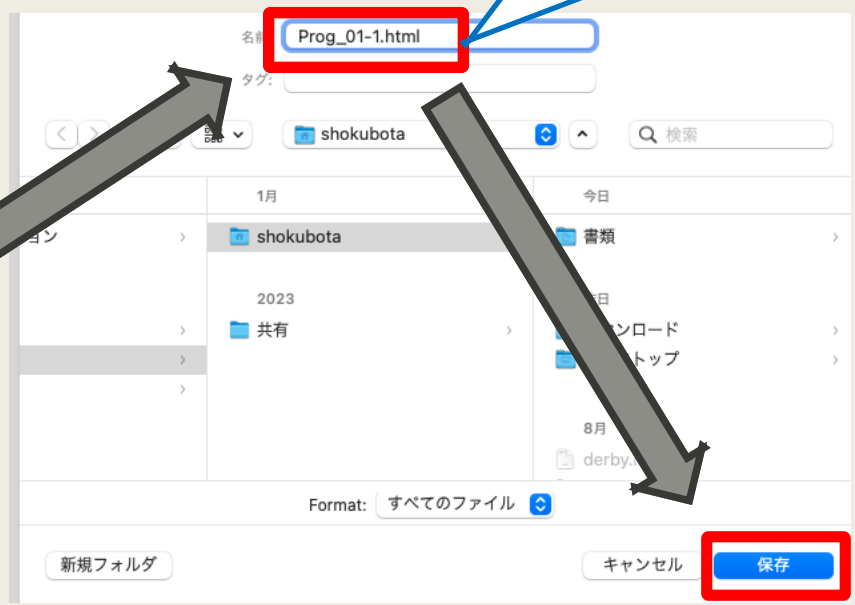


[準備]コードの新規作成②

いつもの作業

- VSCode を起動し「ファイル」から「新しいテキストファイル」を選択。
- そのあと、さきほどコピーした文書をペースト（Ctrl + V）して「名前をつけて保存」。

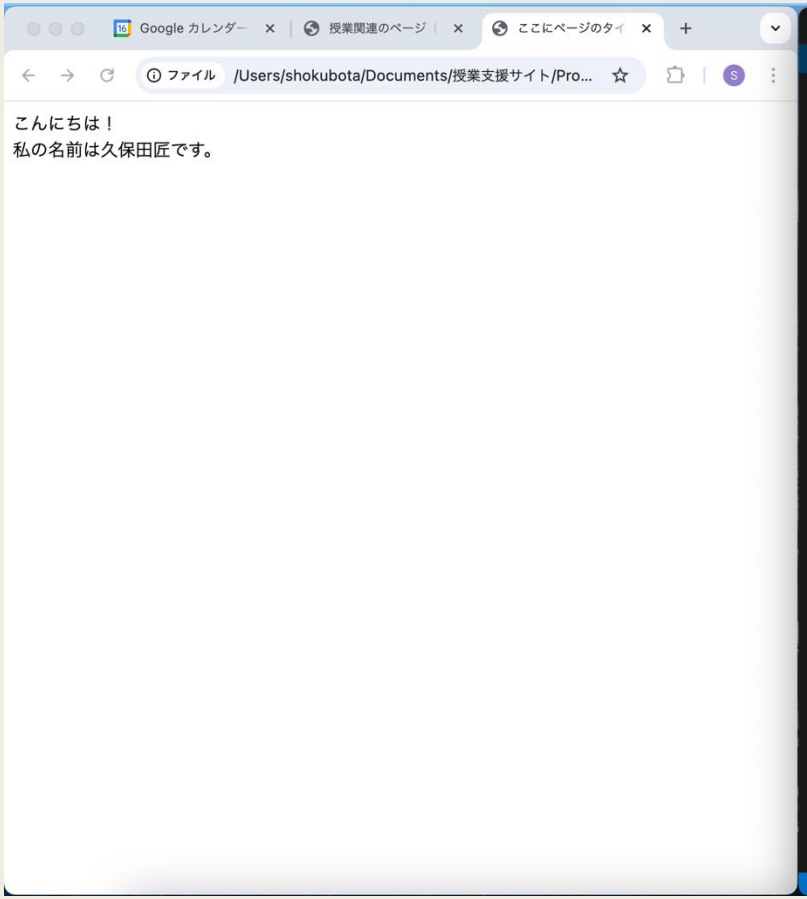


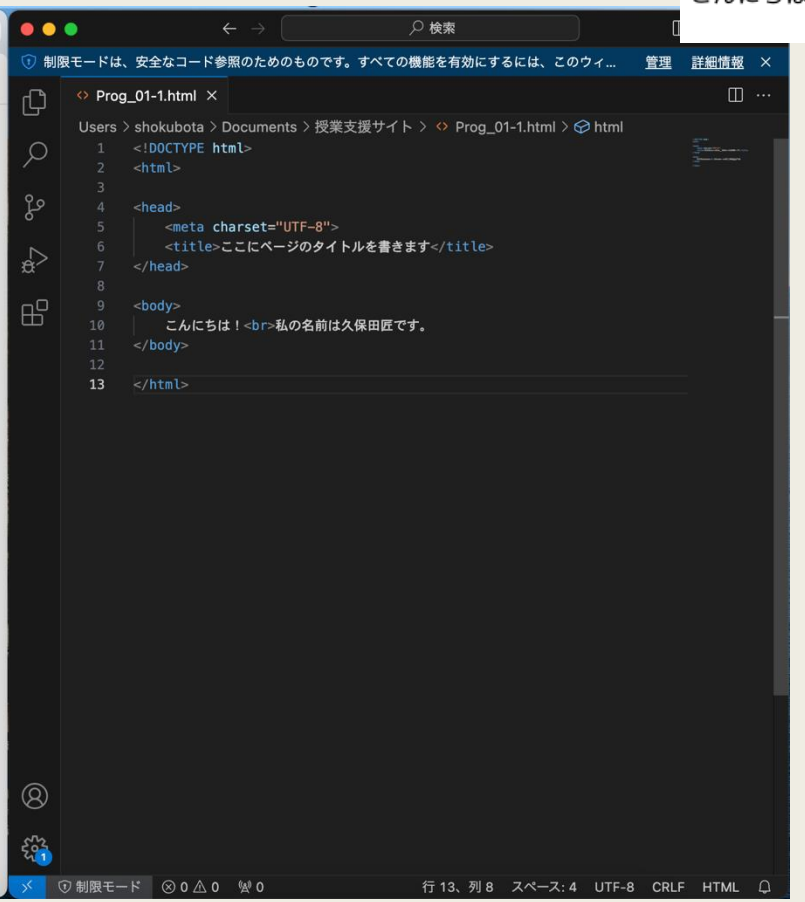


[準備]作業環境を整える

いつもの作業

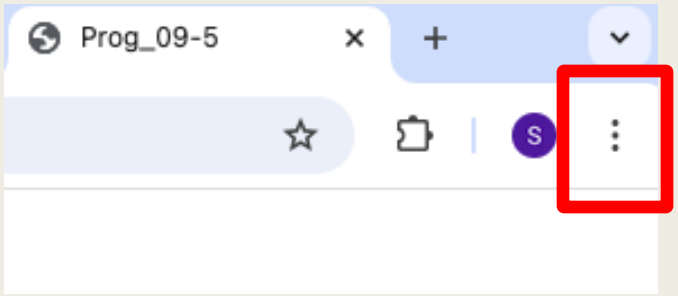
- 保存したhtmlファイルをダブルリックして開いておく。
- PCの画面をふたつに分け、片方はブラウザ、もう片方は VSCode を開いておくと便利。

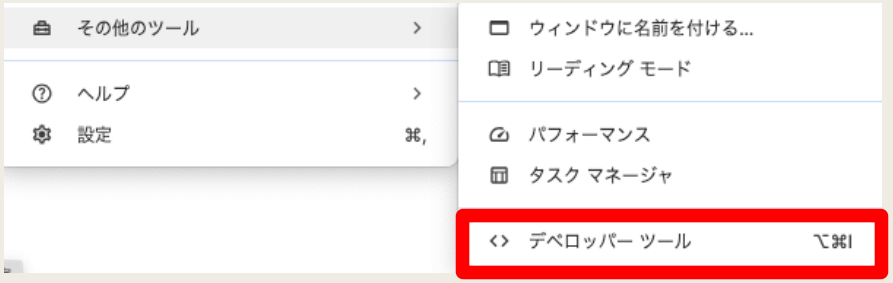


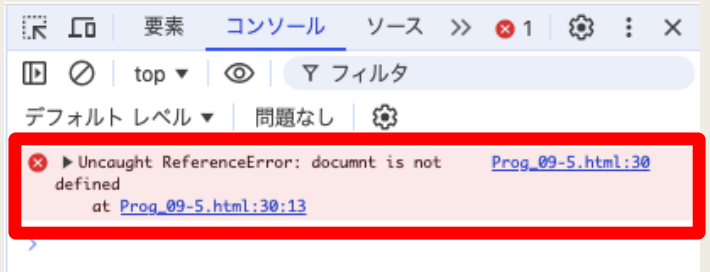


[再掲]デベロッパーツール

- 画面に何も表示されないときや、途中までしか表示されないときはプログラムに間違いがある可能性が高い。
- そのときは「デベロッパーツール」を開き、何行目でエラーが発生しているかを見てみよう。







ショートカットキーもある
Windows → Ctrl + Shift + i
Mac → Option + Command + i

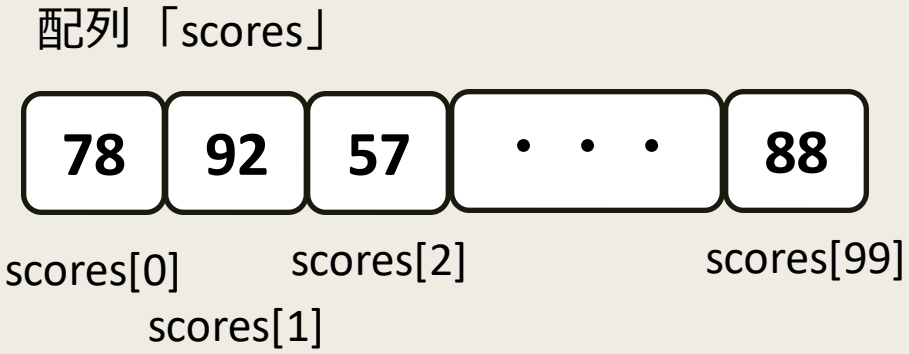
30行目でエラーが発生。
documnt が未定義と言われている
（スペルミスが発生していた）

[復習]配列

- 配列とは、添字（そえじ）と呼ばれる番号を使ってひとつの変数名で複数のデータをまとめて管理できるようにしたもの。
- 例えば、100名の学生の点数を管理したいとき、通常の変数を使用すると個別に100個の変数を宣言する必要がある。
- これに対して、配列を使えば scores のようなひとつの変数名だけで100個のデータをまとめて管理できる。

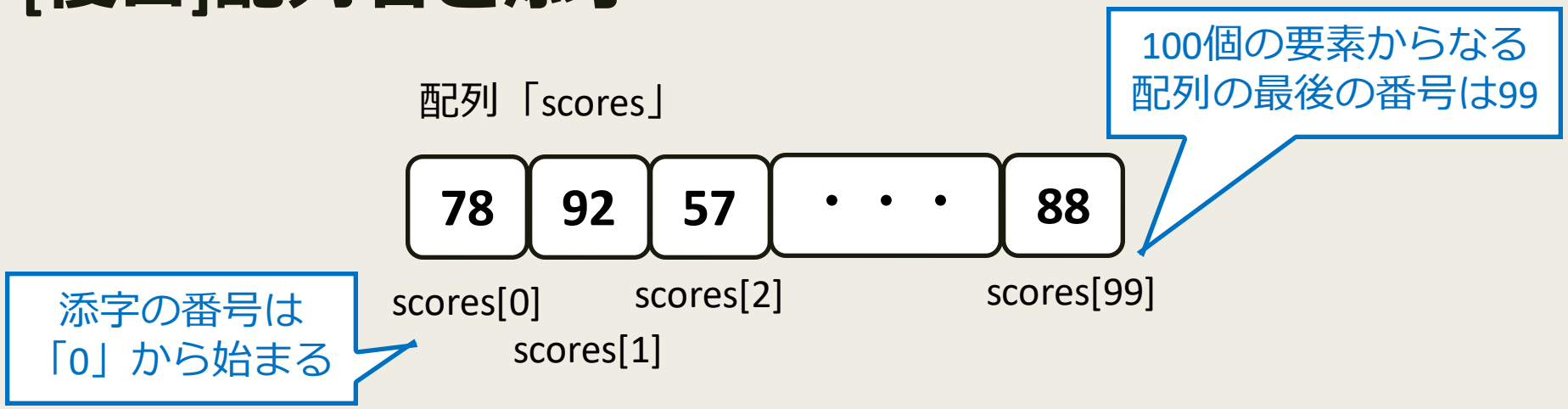
```
let score1 = 78;
let score2 = 92;
let score3 = 57;
  ⋮
let score100 = 88;
```

これまでの方法では
100個の変数を
宣言する必要がある



配列はひとつ宣言すれば
あとはまとめて管理できる

[復習]配列名と添字



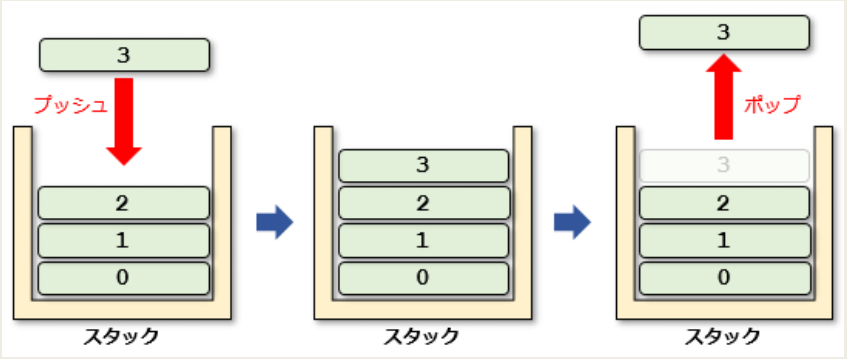
- 配列に格納された個々の値を 要素 という。
- それぞれの要素には次の形式でアクセスする。

  配列名[添字]　（上の例では score[0] で 78 を取得）

```
let array = [65, 81, 73, 52, 84];
// 配列とfor文はよく用いられる
for(let i=0; i<=array.length - 1; i++){
  // 配列の i 番目の要素に対する処理
}
```

[復習] Array オブジェクト

- JavaScript で配列を作った場合、それは必ず Array オブジェクトとなる。
  - Array とは配列のこと。
- よって、Array オブジェクトのプロパティやメソッドが利用できる。
- 配列名.length は Array オブジェクトのプロパティのひとつ。
- push メソッドは配列の最後に新しい要素を追加する。
- pop メソッドは配列の最後の要素を削除し、その削除した要素を返す。

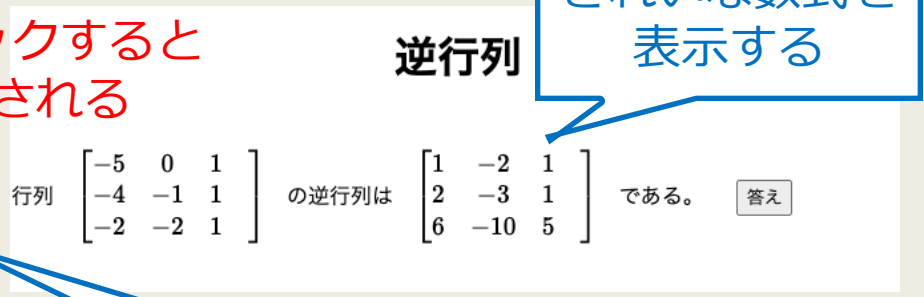



```
let array = [0,1,2];
array.push(3);
console.log(array); // [0,1,2,3]
let x = array.pop();
console.log(array); // [0,1,2]
console.log(x); // 3
```

https://atmarkit.itmedia.co.jp/ait/articles/1908/06/news015.html

[再掲]第10回以降に学ぶこと

ボタンをクリックすると
答えが表示される



[第14回]
きれいな数式を
表示する

[第10回]
乱数を発生させる

[だいたい済]
生成した問題に
対して答えを計算
(透明色で表示)

[第13回]
ボタンを押したときに
特定の処理を行う

[第12回]
処理のかたまりを
定義する

- 第11回では配列を扱う。
- 配列はひとつの変数名で複数のデータをまとめて管理できるようにしたもの。
- 例えば、上の例で配列を使わずにプログラムすると、問題の行列と答えの行列の各成分で合計18個の変数を用意しなければならない。

「処理」の粒度

- 「車に乗って外出する」という行動について考えてみよう。
- ひとつひとつの行動はより細分化することもできる。
- しかし、細分化された行動リスト（右）を最初に見せられると何をしたいのかパッと見では分かりづらい。

◎家から出る
・玄関の扉の鍵を開ける
・玄関の扉を開ける
・外に出る
・玄関の扉を閉める
・玄関の扉の鍵を閉める

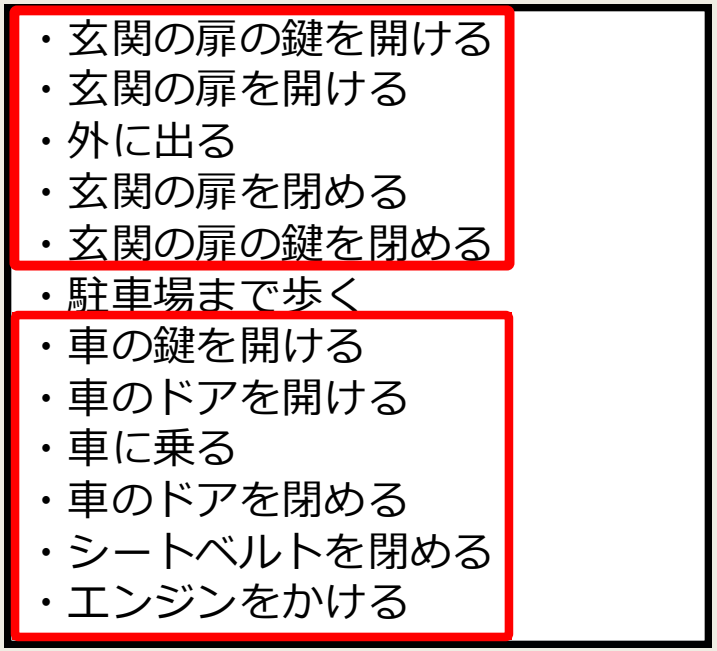
・駐車場まで歩く

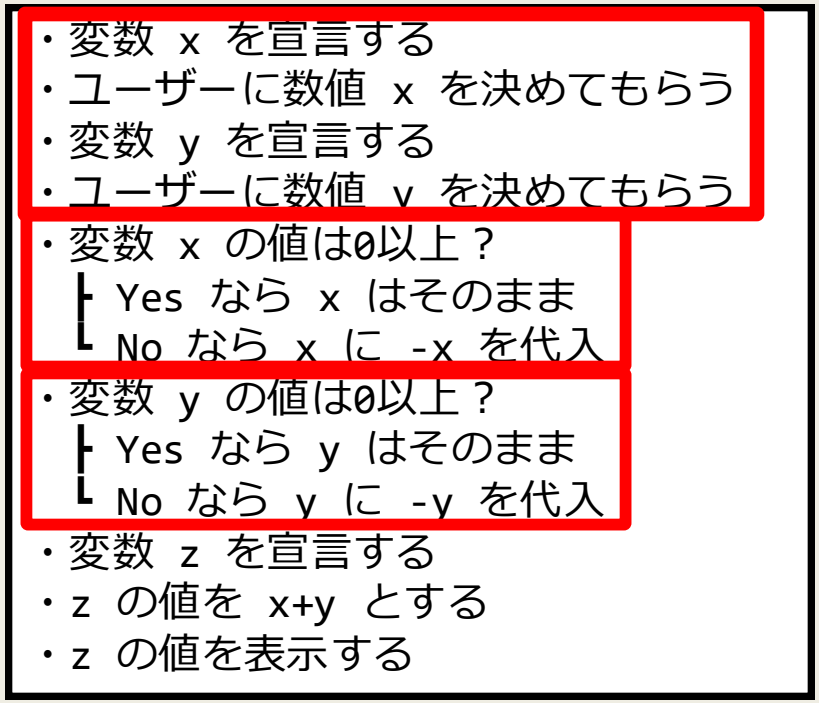
◎車に乗る
・車の鍵を開ける
・車のドアを開ける
・車に乗る
・車のドアを閉める
・シートベルトを閉める
・エンジンをかける

・玄関の扉の鍵を開ける
・玄関の扉を開ける
・外に出る
・玄関の扉を閉める
・玄関の扉の鍵を閉める
・駐車場まで歩く
・車の鍵を開ける
・車のドアを開ける
・車に乗る
・車のドアを閉める
・シートベルトを閉める
・エンジンをかける

プログラムは細分化された行動リスト

- コンピュータプログラムのひとつひとつの命令は、いわば「細分化された行動リスト」のようなもので、コードが長くなると読み手は何をやっているか把握しづらい。
- しかし多くのプログラムは、いくつかの意味のある「処理のかたまり」に分割できることが多い。





|x| + |y| の値を出力

プログラムの抽象化と構造化

- 意図が分かりやすい疑似コードはどちらだろうか？

```
・変数 x を宣言する
・ユーザーに数値 x を決めてもらう
・変数 y を宣言する
・ユーザーに数値 y を決めてもらう
・変数 x の値は0以上？
  ├ Yes なら x はそのまま
  └ No なら x に -x を代入
・変数 y の値は0以上？
  ├ Yes なら y はそのまま
  └ No なら y に -y を代入
・変数 z を宣言する
・z の値を x+y とする
・z の値を表示する
```

```
◎ユーザーに x，y の値を決めてもらう
◎ |x| を求める
◎ |y| を求める
◎ |x|+|y| を表示する
```

「数値 x の絶対値を求める」
という処理をひとかたまりにしておくと
他のプログラムを書くときも
その処理を使う（再利用）ことができる

- 右のように、おおまかな処理をあえて一度記述することでプログラムの構造が瞬時に把握でき、その結果としてプログラムの可読性（見やすさ）が向上する。
- さらに、処理の一部をかたまりにしておくことでプログラムの再利用性も高まる。

関数

必要なときに呼び出す

- プログラミングでは「処理のかたまり」を 関数 という。
  - 数学の関数（や写像）とはまた意味が違うので注意。
- プログラミングにおける関数は 標準関数 と ユーザー定義関数 に分けられる。
- 標準関数とは、JavaScript で用意されている関数のこと。
  - これまでに習ったオブジェクトのメソッドは標準関数の一種。
  - array.push(x), math.random() など。
  - 引数をとるものもあれば取らないものもある。
- ユーザー定義関数とは、プログラムの書き手がオリジナルで作った関数（処理のかたまり）のこと。

関数を定義して使ってみよう

- ユーザー定義関数は、通常 html ファイル内の head部 で定義する。
- 関数を定義するときは右上の形式に沿って定義する。
- 右は与えられた数を2乗する関数 square とその利用例である。入力してみよう。
- この例は関数を定義する方法と使い方を例示するためのものである。実践的には、この程度の処理を関数として定義する必要は全くない。

```
function 関数名(引数1, 引数2, ...){

    // 実行したい処理を記述

    return 戻り値;
}
```

script タグで囲むのを忘れないこと

```
1  <!DOCTYPE html>
2  <html>
3
4  <head>
5      <meta charset="UTF-8">
6      <title>Prog 11-4</title>
7      <script>
8          function square(x){
9              return x**2;
10         }
11     </script>
12 </head>
13
14 <body>
15     <p>
16         <script>
17             let a = 12;
18             document.write(a+"の2乗は"+ square(a) +"です。");
19         </script>
20     </p>
21 </body>
22
23 </html>
```

12の2乗は144です。

素数の列挙

- もう少し実践的な例に取り組んでみよう。
- 素数とは「2以上の自然数で、1と自分自身でしか割り切れない数」をいう。
- 100以下の素数を列挙するプログラムを作成しよう。
- 着手する前にプログラムの構想を考える。

```
・素数を格納する空配列 primes を用意する
・for(let i=2; i<=100; i++){
    ・iが素数なら配列 primes に i をpush
}
```

- 与えられた数が素数かどうかを判定する関数を用意しておくとよさそう。

※ユーザー定義関数は、コードを整理して再利用しやすくするための仕組み。プログラムを動かすために必ず使わなければならないものではない。

素数を判定する関数

・素数を格納する空配列 primes を用意する
・for(let i=2; i<=100; i++){
　　・iが素数なら配列 primes に i をpush
}

- 素数を判定する関数を用意する。
- 右は与えられた整数が素数かどうかを判定する関数 isPrime である。意味を理解しながら入力してみよう。
  - 関数 isPrime(n) は n が素数のとき true を返し、n が素数でないとき false を返す。

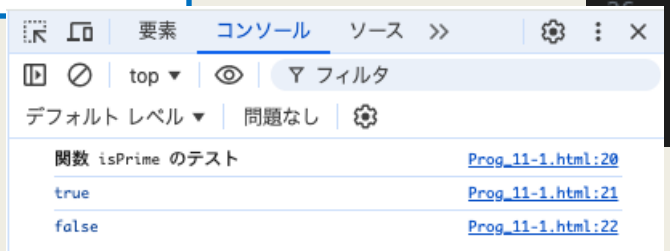
デベロッパーツールを開く
ショートカットキー↓
Windows → Ctrl + Shift + i
Mac → Option + Command + i

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="UTF-8">
    <title>Prog_11-1</title>
    <script>
        function isPrime(n){
            if(n == 2){
                return true;
            } else if(n > 2){
                for(let i=2; i<n; i++){
                    if(n%i == 0){
                        return false;
                    }
                }
                return true;
            }
        }
        console.log("関数 isPrime のテスト");
        console.log(isPrime(7));
        console.log(isPrime(12));
    </script>
</head>

<body>
</body>

</html>
```



[演習]素数の列挙

・素数を格納する空配列 primes を用意する
・for(let i=2; i<=100; i++){
　・iが素数なら配列 primes に i をpush
}

- 先のプログラムの body部 を書き換えて、100以下の素数を列挙するプログラムを作成しよう。

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="UTF-8">
    <title>Prog_11-2</title>
    <script>
        function isPrime(n){
            if(n == 2){
                return true;
            } else if(n > 2){
                for(let i=2; i<n; i++){
                    if(n%i == 0){
                        return false;
                    }
                }
                return true;
            }
        }
        console.log("関数 isPrime のテスト");
        console.log(isPrime(7));
        console.log(isPrime(12));
    </script>
</head>
```

↑ここはさっきと同じ

```
<body>
    <p>
        <script>
            考えてみよう
        </script>
    </p>
</body>

</html>
```

素数を判定する関数 isPrime は
実際にはもっと効率化できる。
余裕のある人は考えてみよう。

100以下の素数は 2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97 です。

変数のスコープ

- 個々の変数には、その変数が利用できる範囲が決まっている。
- これを変数の スコープ という。
  - 関数内で宣言した変数のスコープはその関数の内部のみ。
  - for文やif文で宣言した変数のスコープはそのブロック内（{ }内）のみ。
  - このような、特定のスコープ内でのみ有効な変数を ローカル変数 という。
- これに対して、関数の外部で宣言した変数はプログラム全体で利用できる。このような変数を グローバル変数 という。

```
function myFunc(){
  let x = 5;
  document.write(x); // 5と表示
}

document.write(x); // エラー
```

```
for(let i=0; i<5; i++){
  document.write(i); // 01234と表示
}

document.write(i); // エラー
```

なぜスコープという概念があるか？

- すべての変数はグローバル変数である方が便利であるように思うかもしれない。
- なぜスコープという概念があるのだろうか？
- スコープには予期せぬエラーやバグを予防する効果がある。
- もしスコープという概念がないと、プログラム中に意図せず同じ名前の変数を使ってしまい、前の変数が上書きされる可能性が出る。
  - 例えば、変数 i を何かの個数を数える意味で使っていたとしよう（そもそも意味のある数値の変数名に i とつけるべきではないが）。
  - このとき、for文を書くときに変数 i を使ってしまうと、個数を数えていた変数 i が上書きされる。
- このようなことが起こらないようにスコープという概念がある。

[演習] 分散を求める

- 配列の要素の分散を計算したい。
- 変量 x のデータの分散を s^2 とすると s^2 は次式で計算できる。

$$s^2 = \overline{x^2} - (\overline{x})^2$$

- 分散を計算するために「配列の要素の平均を求める関数」「配列のすべての要素を2乗する関数」を作り、与えられた配列の要素の分散を計算するプログラムを作ろう。

[演習] 分散を求める

$$s^2 = \overline{x^2} - (\overline{x})^2$$

- 分散を計算するために「配列の要素の平均を求める関数」「配列のすべての要素を2乗する関数」を作り、与えられた配列の要素の分散を計算するプログラムを作ろう。

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="UTF-8">
    <title>Prog_11-3</title>
    <script>
        /*
        配列の平均を返す関数 calculateMean
        入力：配列
        出力：数値
        */
        function calculateMean(array){

            考えてみよう

        }

        /*
        配列のすべての要素を2乗する関数 squareElements
        入力：配列
        出力：配列
        */
        function squareElements(array){

            考えてみよう

        }
    </script>
</head>
```

```
<body>
    <p>
        <script>
            let scores = [65, 81, 73, 52, 84];

            考えてみよう

        </script>
    </p>
</body>

</html>
```

分散は 134 です。

※コメント部（緑色の字）は書き写さなくてよい。

[演習]教科書を熟読しよう

- 今日の内容は教科書の p126～p139 がベースになっている
- 残った時間で自分でも該当箇所を熟読してみよう。
- 授業で解説していないコードは自分でも入力してみてどのような出力結果になるか確かめてみよう。
- ユーザー定義関数は、コードを整理して再利用しやすくするための便利な仕組みであるが、プログラムを動かすために必ず使わなければならないものではない。
- まずはコードが汚くても、動くプログラムを書くことを目指そう。
- その後、関数を使って整理することで、コードがさらに分かりやすくなるということを実感できるはずである。