目次

序章　Monaca Education の利用方法 ・・・・・・・・・・・ 2
第１章　順次実行と変数・・・・・・・・・・・・・・・・・・・ 10
第２章　条件分岐による選択（分岐）・・・・・・・・・・・・ 20
第３章　配列・・・・・・・・・・・・・・・・・・・・・・・・ 34
第４章　繰り返し（反復）・・・・・・・・・・・・・・・・・・ 42
第５章　関数の定義と利用・・・・・・・・・・・・・・・・・・ 52
第６章　繰り返し（反復）と選択（分岐）の組み合わせ・・・ 62
付録　DOM の操作 ・・・・・・・・・・・・・・・・・・・・・ 72

序章　Monaca Education の利用方法

Monaca Education を利用するためにはアカウントが必要です。先生の指示に従ってアカウントを事前に準備して下さい。

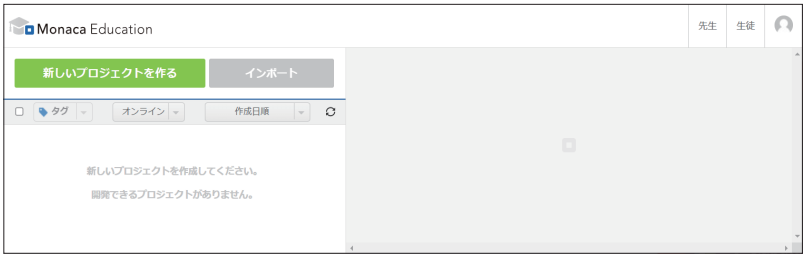
Monaca Education にログインする

Monaca Education の公式サイトにアクセスします。

https://edu.monaca.io

次に右上の「ログイン」をクリックして下さい。ログインフォームが表示されますので、事前に準備したアカウントの ID とパスワードを入力して「ログイン」します。

ログインに成功すると「ダッシュボード」が表示されます。ダッシュボード上では制作中の作品を「プロジェクト」という単位で管理します。



（参考）アカウント連携

Google や Microsoft アカウントを利用している場合、アカウント連携を行うことで次回から Monaca Education の ID やパスワードを入力せずにログインできます。
ダッシュボード右上のアイコンから「アカウント設定」を経由して連携ページに進めます。なお、学校の設定によってはアカウント連携は利用できない場合もあります。先生の指示に従って利用して下さい。

（参考）アカウント作成

自身で Monaca Education のアカウント作成を行うこともできます。アカウントを作成する際には先生の指示に従ってください。また、アカウント作成の手順は Monaca Education の公式サイトにて動画付きで解説を行っております。

https://edu.monaca.io/start

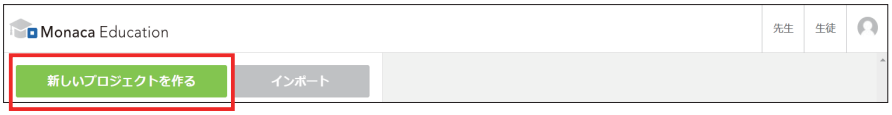
また、自身でアカウント作成を行う場合はライセンスが適用されないため、有料版の機能を使うためには後からライセンスの設定が必要となります。

プロジェクトを作成する

Monaca Education を体験するために、Monaca Education のテンプレートから「ブロック崩し」を選択して動かしたり改造したりしてみましょう。

プロジェクトを作成する

画面左上の『新しいプロジェクトを作る』ボタンをクリックして下さい。



テンプレート一覧が表示されます。今回はブロック崩しを選択します。

プロジェクトにはプロジェクト名や説明を設定できます。今回はテンプレートの元々のプロジェクト名のまま進めることにします。

ブロック崩しプロジェクトを IDE(統合開発環境) で開く

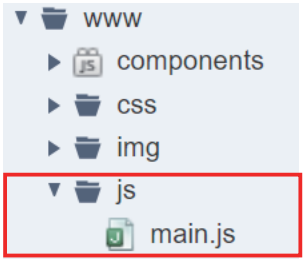
ダッシュボードで作成したプロジェクトを選択し「クラウド IDE で開く」のボタンをクリックして下さい。画面が切り替わり、プログラムを制作するための IDE(統合開発環境) が展開されます。

各部の名称

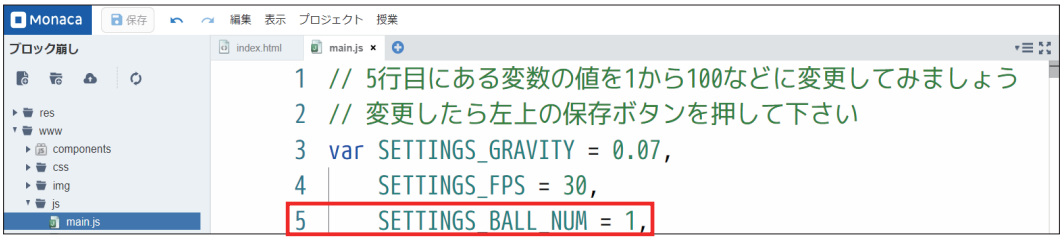
各部の名称は下図の通りです。

プロジェクトパネルから js/main.js を開いて 5 行目を編集する

フォルダのアイコンの左にある三角形の記号をクリックすると中身が一覧で展開されます。また、ファイルをダブルクリックするとエディタで編集できます。今回は js フォルダを展開して main.js ファイルをエディタで編集します。

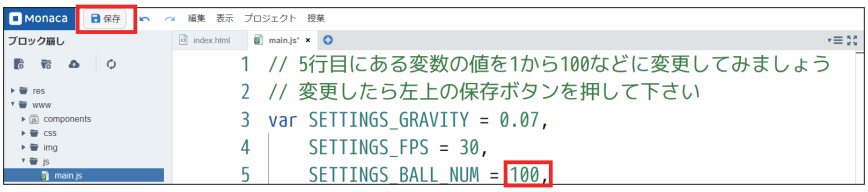


エディタのパネルではプログラムのソースコードを変更できます。今回は main.js の 5 行目を変更して玉の数を 100 個に増やして下さい。

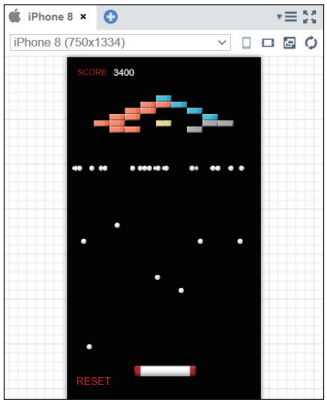


main.js を保存してプレビューパネルに反映させる

変更したら保存も忘れずに行って下さい。IDE メニューの左上に保存ボタンがあり、これをクリックすることで保存できます。またショートカットキーでの保存にも対応しており、Ctrl キーを押しながら s キーを押すことでキーボードから手を放すことなく保存できます。



保存を行うと自動的にプレビューパネルが更新され、玉の数が増えた状態でブロック崩しが動作します。



作品を Web に公開する

作品を Web に公開することもできます。公開中の作品はログインなしで自身のスマートフォンなどから動作を確認できます。また、作品の URL を先生に提出することもできます。

Web 公開

メニューの「授業」から「課題提出（Web 公開）」を選択して下さい。

公開のラジオボタンを「On」にして「適用する」を選択すると作品が公開されます。

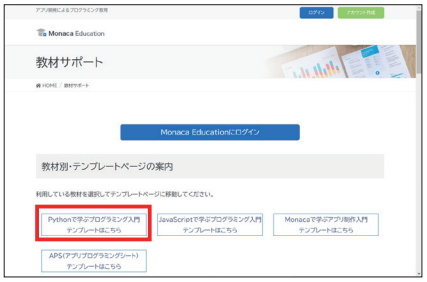
QR コードが表示されたら公開中です。

作品の URL を提出するときには「URL をコピー」を使うと便利です。

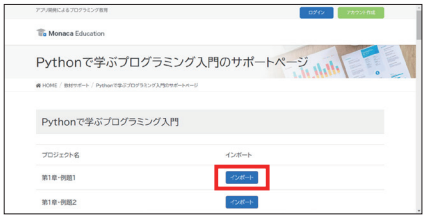
I 章以降の学習に向けて

I 章以降で利用するプロジェクトのテンプレートは、教材サポートページから入手可能です。こちらのテンプレートは「答え合わせ機能」にも対応しており、実習をスムーズに進められます。教材サポートページはダッシュボード右上の「生徒」からアクセスできます。

教材サポートページでは各種印刷教材などのテンプレートを取得できます。以下、「Python で学ぶプログラミング入門」を例に流れを紹介します。



テンプレート一覧から、学習したい内容のテンプレートを「インポート」します。



「インポート」のポップアップが立ち上がり、テンプレートをインポートできます。

JavaScript プロジェクトの編集方法

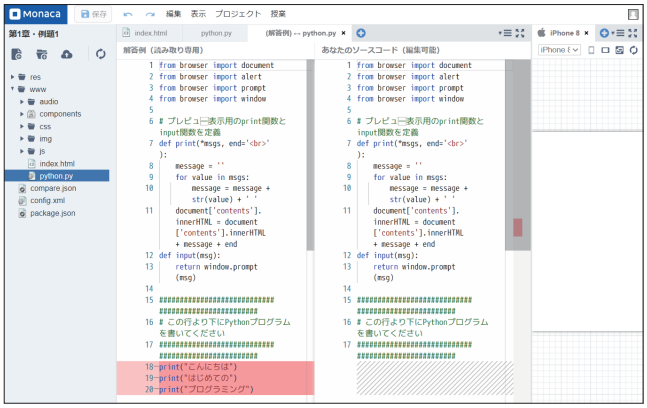
JavaScript プロジェクトのプログラムは index.html ファイルの <script> タグの中に記述して下さい。また、プログレッシブテンプレートを用いて main.js に記述しても構いません。

Python(Brython) プロジェクトの編集方法

Python(Brython) プロジェクトのプログラムは python.py ファイルに記述して下さい。

答え合わせ機能

教材サポートページのプロジェクトは答え合わせ機能に対応しています。例えば「python.py」をエディタで開いた状態でメニュー「授業」の「答え合わせ」を選択すると左に解答例が表示されます。同機能はプロジェクトパネルのファイルを右クリックすることでも呼び出せます。



プロジェクトを開く（セーフモード）

本書の 4 章以降では「繰り返し（反復）」処理を扱うため、無限ループを発生させてクラウド IDE を開くことができなくなる場合があります。その場合は「セーフモード」で開いて問題の箇所を修正して下さい。セーフモードは「クラウド IDE で開く」ボタンの右にある逆三角のボタンをクリックすることで選択できます。

第 1 章　順次実行と変数

プログラムを書くと、コンピュータに計算をさせたり、計算した結果を表示させることができます。コンピュータはプログラムに書いた命令を上から順に実行します。これを**順次実行**といいます。プログラムの中では、計算の結果を一時的に格納しておく**変数**という仕組みを使うことができます。
この章では、短いプログラムを実際に実行させながら、順次実行と変数を学びます。

順次実行

次のプログラムの実行結果を確認しましょう。たった 2 行ですが、立派なプログラムです。

```
document.write(" こんにちは ");
document.write(" プログラミング ");
```

プログラムの中では、英字（アルファベット）、日本語（ひらがな・カタカナ）の文字の他、二重引用符（"）や、丸カッコ（(および)）などの記号が使われています。
「document.write()」は、() の中に指定した文字列（文字の並び）を、画面に表示させます。
プログラム言語 JavaScript は初めからたくさんの命令を備えています。「document.write()」はその中の 1 つです。

```
こんにちはプログラミング
```

このプログラムでは、その document.write() を 2 回使っています。先に書かれている「document.write(" こんにちは ")」が実行された後に、後ろに書かれている「document.write(" プログラミング ")」が実行されています。

このように、プログラムに書かれている順番の通りに実行する構造を**順次構造**と呼びます。

次のプログラムも実行してみましょう。これも順次構造です。

```
alert(" こんにちは ");
alert(" プログラミング ");
```

ただし、メッセージは小さなウィンドウ（画面表示をする部品）に表示されます。最初、ウィンドウに「こんにちは」と表示されます。ウィンドウの中の OK ボタンを押すと、そのウィンドウが閉じ、すぐに次のウィンドウが表示されます。2 個目のウィンドウには「プログラミング」というメッセージが表示されます。

document.write() と、alert() で、メッセージの表示のしかたが違うことを確認してください。

補足 1: JavaScript でプログラムを書くとき、行の終わりには記号「;」（セミコロン）を付けます。
補足 2: document.write() でメッセージを書き出すとき、改行されないのは、HTML を書き出しているからです。HTML では、改行をするためには「
」という HTML タグを付け足す必要があります。

```
document.write(" こんにちは <br>");
document.write(" プログラミング ");
```

例題 1: 順次実行

次のような実行結果が表示されるように、上のプログラムを変更してください。

```
こんにちははじめてのプログラミング
```

改行も入れてみましょう。

```
こんにちは
はじめての
プログラミング
```

補足 1: document.write() や、alert() は、正確には**関数**と呼ばれる部品です。() の中に入れている値を**引数**といいます。関数と引数については、「第 5 章 関数の定義と利用」で詳しく扱います。
補足 2: プログラムの中で英字を使うとき、大文字・小文字は区別されます。たとえば「alert()」と「ALERT()」は別のものとして扱われます。JavaScript には関数 alert はありますが、関数 ALERT は存在しないため、ALERT と書くとプログラムはエラーを発生させます。また、全角文字と半角文字も区別されます。全角の日本語を入力した後に英字を入力するときは、全角・半角に注意してください。
補足 3: プログラムで表示する文字列を扱うときは、二重引用符（""）または一重引用符（''）で囲みます。

変数

次のプログラムを実行してみましょう。

```
let a = "Hello";
alert(a);
```

実行結果は、「a」ではなく、「Hello」になっています。

```
Hello
```

1 行目にある「a」は変数といいます。変数に値を入れて、後で利用することができます。変数に値を入れることを、**代入**するといいます。
最初の行は、JavaScript 言語のキーワード「let」を使い変数 a を宣言し、さらに記号「=」を使って、「Hello」という文字列の値を変数 a に代入しています。

例題 2：変数に値を代入する

前のプログラムを、次のように編集して、保存します。

```
let a = "Hello";
a = " こんにちは ";
alert(a);
```

実行結果はどうなるでしょうか？「Hello」と表示されるでしょうか。「こんにちは」と表示されるでしょうか。

補足 1：ここまでのプログラムでは変数の名前として「a」を使いましたが、変数には好きな名前を付けることができます。プログラムや変数の目的・用途に合わせて名前を付けると分かりやすくなります。たとえば、変数の名前を「a」に代えて「aisatsu」とすることが考えられます。変数名に日本語（ひらがな・カタカナ・漢字）を使うこともできますが、プログラムを編集する環境によって読めなくなる可能性があるため、英数字を使うことを薦めます。

補足 2：JavaScript は記号「=」の前後の空白を無視します。前後の空白は、あってもなくてもプログラムの動作は同じです。

数値と計算

ここまで、文字列ばかり扱ってきましたが、数値を扱うこともできます。
数値を計算することもできます。たとえば、記号「+」を使うと、足し算ができます。

```
let b = 10;
alert(b);
let c = b + 10;
alert(c);
alert("b は " + b + " c は " + c);
```

プログラムが少し長くなりました。実行結果とプログラムを 1 行ずつ見比べて、どんな動作をしたか確認しましょう。

```
10
20
b は 10 c は 20
```

1 行目では、変数 b を宣言して「10」という数値を代入しています。次の 2 行目で関数 alert() を使って、変数 b の値を表示させています。
さらに、3 行目では変数 c を宣言して、値を代入しています。右辺に書かれている、変数 b と「10」の足し算の結果（20）を代入しています。4 行目の「alert(c)」で、変数 c に代入された値を表示しています。
5 行目の「alert()」では、表示させたい値を記号「+」で連結した文字列を表示させています。

補足：記号「+」のように、計算させるための記号を**算術演算子**といいます。JavaScript では文字列に記号「+」を使うと、文字列を結合する（つなぐ）働きをします。

例題 3：計算結果を表示する

34567 + 123456 の結果を表示するようにプログラムを作成してください。その際、値を格納するための 2 つの変数と、足し算の結果を格納する変数の、合計で 3 つの変数を使ってください。

入力を受け付ける

次のプログラムを実行してみましょう。

```
let a = prompt("名前は?");
alert("名前:" + a);
let b = prompt("年齢は?");
b = parseInt(b);
alert("年齢:" + b);
```

すると、次のように、入力を促す表示がされます（※この表示は、使用しているブラウザの種類など、動作させている環境によって異なります）。

何か好きな値（たとえば、「山田」）を入力し、OK ボタンを押します。

続けて、年齢を入力する画面が表示されるので、数字を入力して、OK ボタンを押します。

実行結果は次のようになります（※実際に入力した値が表示されます）。

```
名前： 山田
年齢： 16
```

あらためて、プログラムを確認しましょう。

```
let a = prompt(" 名前は ?");
alert(" 名前 :" + a);
let b = prompt(" 年齢は ?");
b = parseInt(b);
alert(" 年齢 :" + b);
```

prompt() という命令が出てきました。これは、プログラムを動かした人（ユーザー）に、何か値を入力するように求めます。ユーザーが入力した値は、変数に代入して、後の処理で使うことができます。

年齢をたずねている「prompt()」が実行されると、ユーザーに入力を求めます。求めに応じてユーザーが値を入力すると、その値は変数 b に代入されます。

例題 4：ユーザーの入力を受け取る

次のような処理を順次実行するプログラムを作成してください。

1 つめの処理：ユーザーに対して「数字を入力してください」と表示して、入力された値を受け取る。

2 つめの処理：受け取った内容を変数 a に代入する。

3 つめの処理：変数 a を表示する。

補足資料

表示（出力）と入力

関数名	動作
alert()	カッコの中に入れられた値を、ポップアップする小さなウィンドウに表示する。 カッコの中には、値を入れてもよいし（"Hello", 16 など）、変数を入れてもよい（a,b など）。 カッコの中で計算をさせてもよい。計算結果がブラウザのポップアップウィンドウに表示される。
document.write()	カッコの中に入れられた値を、画面に表示する。 カッコの中には、値を入れてもよいし（"Hello"、16 など）、変数を入れてもよい（a,b など）。 カッコの中で計算をさせてもよい。計算結果が表示される。
prompt()	ブラウザのポップアップウィンドウを表示し、ユーザーに値を入力するよう求める。 カッコの中に入れた文字列は、入力を求める画面に表示される（" 名前は？ ", " 年齢は？ " など）

値の型を変換する関数

関数名	動作
parseInt()	parseInt("16") のように、整数として扱える文字列が渡されたとき、数値型の値に変換する。 a = parseInt("16") とすると、変数 a には 16 という数値の型の値が（"16" という文字列の値ではなく）代入される。 整数として扱えない文字列が渡された場合、プログラムを実行した時にエラーになる。
parseFloat()	parseFloat("174.5") のように、小数点を含む数値として扱える文字列が渡されたとき、数（浮動小数点数）に変換する。 浮動小数点数として扱えない文字列が渡された場合、プログラムを実行した時にエラーになる。

算術演算子

演算子	動作
+	数値の足し算を行う。 ※文字列に対して使うと、前後の文字列をつなげることができる。
-	数値の引き算を行う。
*	数値の掛け算を行う。
/	数値の割り算を行う。
%	数値の割り算を行い、余りを得る。剰余計算。 例：10 % 3 -> 1 10 割る 3 は 3 余り 1 。% は余りを返す。

問題集

問 1.　alert() を 3 回使って、「おはよう」「こんにちは」「こんばんは」と表示するプログラムを作ってください。

問 2.　変数 myouji に名字を、変数 namae に名前を代入した上で、名字と名前を 1 回で表示するプログラムを作ってください。

問 3.　prompt() と parseInt()、alert() を使い、ユーザーに数字を入力させた後、入力された値に 10 を足した値を表示するプログラムを作ってください。

問題集

問１.　次のように動作するプログラムを作成してください。
（偶数と奇数を区別する）

① ユーザーに対して「数字を入力してください」と表示して、入力された値を受け取り、
② 受け取った内容を変数 a に代入して、
③ 変数 a が偶数かどうか判別する。
　ア　変数 a が偶数なら、「これは偶数です」と表示する。
　イ　変数 a が奇数なら、「これは奇数です」と表示する。

（ヒント）
・　割り算の余りを求めるには、算術演算子 % を使います。
・　例：７ % ３の計算結果は、１になります。
・　与えられた値について、２で割った余りが０なら、その数は偶数です。

問２.　次のように動作するプログラムを作成してください。
① １から６の整数をランダムで生成し、変数 a に代入する
② 変数 a の値に応じて、「が出ました」という表示をする。ただし、ローマ数字ではなく、漢数字を使う。
　・　１なら、「一が出ました」
　・　２なら、「二が出ました」
　・　３なら、「三が出ました」
　・　４なら、「四が出ました」
　・　５なら、「五が出ました」
　・　６なら、「六が出ました」

（ヒント）
・　Math.floor(Math.random() * 3) という式は、０以上２以下の整数を返す。
・　Math.floor(Math.random() * 3) + 1 とすれば、１以上３以下となる。
・　サイコロには、１から６の目がある。

第２章　条件分岐による選択（分岐）

操作する人の入力や操作に合わせてプログラムの動作を変えられると、便利です。似たようなプログラムをいくつも書く必要がなくなります。
この章では、条件に応じて動作が変わるプログラムを作ります。これを**選択**または**分岐**と呼びます。

条件による選択（分岐）①式が真のとき

まず、次のプログラムを実行してみましょう。

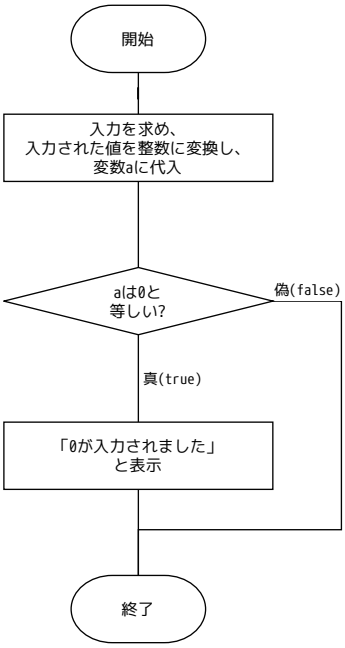
```
let a = parseInt( prompt("0 か 1 を入力してください") );
if (a == 0 ){
    document.write("0 が入力されました");
}
```

最初は、入力欄に、「0」を入力してみます。実行結果はどうなるでしょうか？

```
0 が入力されました
```

プログラムを再度実行します（※ Monaca Education なら、更新ボタンを押すと再度実行できます）。
今度は、「1」を入力してみます。実行結果はどうなるでしょうか？何か文字は表示されたでしょうか？

何も表示されないのが正しい動作です。
上のプログラムのフローチャートを見てみましょう。



1 行目：「prompt()」で入力を受け取り、「parseInt()」で文字列の値を数値に変換しています。

```
let a = parseInt( prompt("0 か 1 を入力してください") );
```

プログラムでは、関数 parseInt() のカッコの内側に、関数 prompt() が書かれています。
関数のカッコの中に関数が書かれているとき、内側の関数が先に実行され、その実行結果を外側の関数に引き渡すという動作をします。

2 行目： if というキーワードで始まっています。続く式「a == 0」の、イコール（=）記号が 2 つ並んでいるのは間違いではありません。これは、== の左右（この例では、変数 a の値と、数値の 0）を比較して、等しいかどうか調べる式です。
変数 a の値と 0 が等しければ（== の左右が等しければ）、条件は真（true）となり、次の 3 行目が実行されます。「0 が入力されました」と表示した後、プログラムは終了します。
変数 a の値と 0 が等しくなければ、条件は偽（false）となり、3 行目は実行されません。何も表示されず、プログラムは終了します。

補足 1: if の後ろにある「== の左右が等しいか調べる」といったプログラムの動作のことを「式を評価する」と言います。
プログラムが実行されたときに、式の評価が行われ、評価結果に応じて実行するプログラムの行が変わる構造を**選択構造**と呼びます。**分岐構造、条件分岐構造**などと呼ぶ場合もあります。

補足 2:「if」のように、プログラムの中で特別な意味のあるキーワードがあります。プログラム言語が予約している語なので**予約語**といいます。予約語は変数の名前に使うことができません。

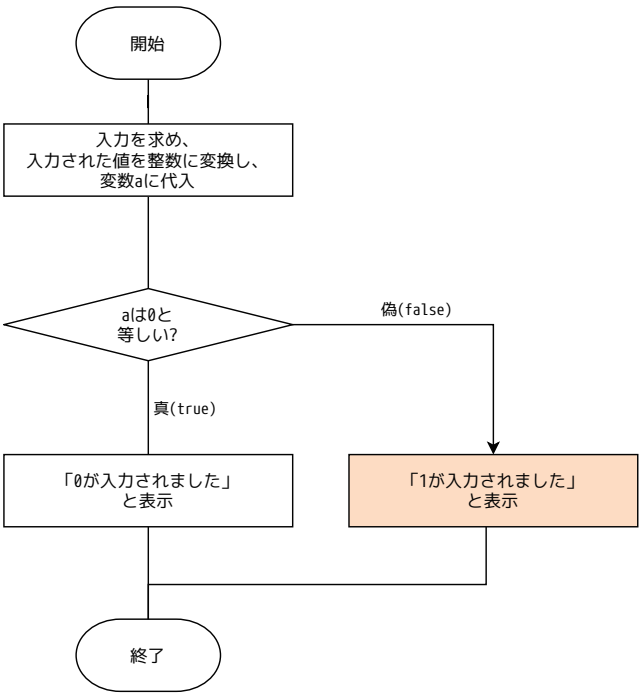
比較演算子とブロック、インデント（字下げ）

```
let a = parseInt( prompt("0 か 1 を入力してください") );
if (a == 0) {
    alert("0 が入力されました");
}
```

if キーワードの後ろの「==」のように、二つの値を比較するための記号のことを**比較演算子**といいます。
比較演算子を使って、変数 a の値と 0 が等しいかどうかを比べる計算をします。計算を小カッコ () で囲んでいることに注意してください。
また、続く処理を { } で囲んでいます。この { } で囲まれた、ひとかたまりのプログラムの部品を**ブロック**と呼びます。
ブロックの中のプログラムは、行の先頭から数文字分、空白を入れてあります。この空白部分をプログラム言語の用語で**インデント（字下げ）**といいます。インデントを入れると、プログラムの見た目が変わり、プログラムが読みやすくなります。

条件による選択（分岐）②式の評価結果が偽のとき

次のフローチャートを見てみましょう。



式「a == 0」の評価が偽（false）だった場合、「1 が入力されました」という表示をするフロー（流

れ）になりました。
JavaScriptによるプログラムは、次の通りになります。elseというキーワードが使われています。

```
let a = parseInt( prompt("0 か 1 を入力してください") );
if (a == 0) {
    alert("0 が入力されました");
} else {
    alert("1 が入力されました");
}
```

プログラムを実行して、「1」を入力してみましょう。実行結果は次の通りです。

```
1 が入力されました
```

もう１度実行して、「0」を入力したときには、「0が入力されました」と表示されることを確認しましょう。
１つのプログラムが、ユーザーの操作・入力に応じて、動作を選択して実行しています。

例題 1：上のプログラムを編集して、次のような動作をするようにしてください。

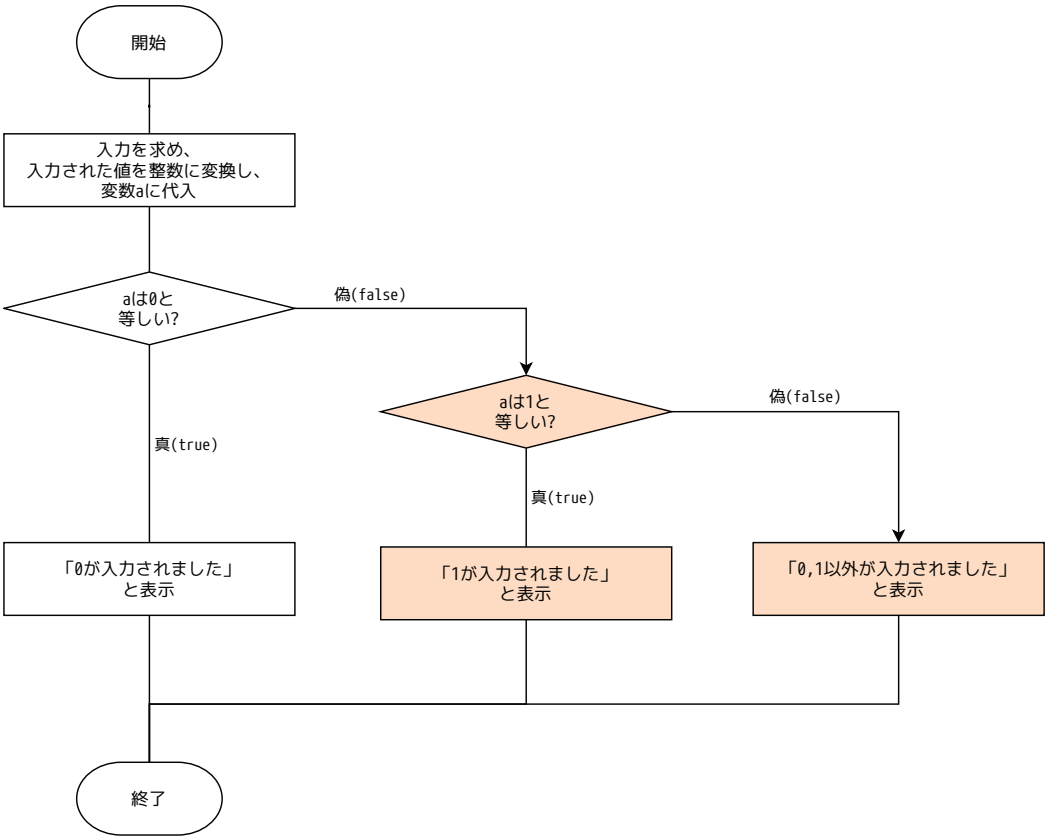
- ユーザーに「0」か「1」を入力するよう促す。
- 「1」が入力されたときに「1が入力されました」と表示する。
- 「1」以外が入力されたときは「1以外が入力されました」と表示する。

条件による選択（分岐）③評価する式を増やす

前のプログラムで、「2」を入力すると、結果はどうなるでしょうか？

```
1 が入力されました
```

この結果は、if 文の書き方が原因です。if キーワードの後ろの式では、「0 と等しいか」を比べ、0 と等しいなら「0 が入力されました」と表示します。0 と等しくないなら（else キーワードの働きにより）「1 が入力されました」と表示するプログラムになっているのです。
本当に「1」が入力されたときに「1 が入力されました」と表示し、0 と 1 のどちらでもない値が入力された場合は「0,1 以外が入力されました」という分岐構造にしてみましょう。
フローチャートは次のようになります。



プログラムは次の通りです。4 行目にある「else if (a == 1)」という記述に注目してください。

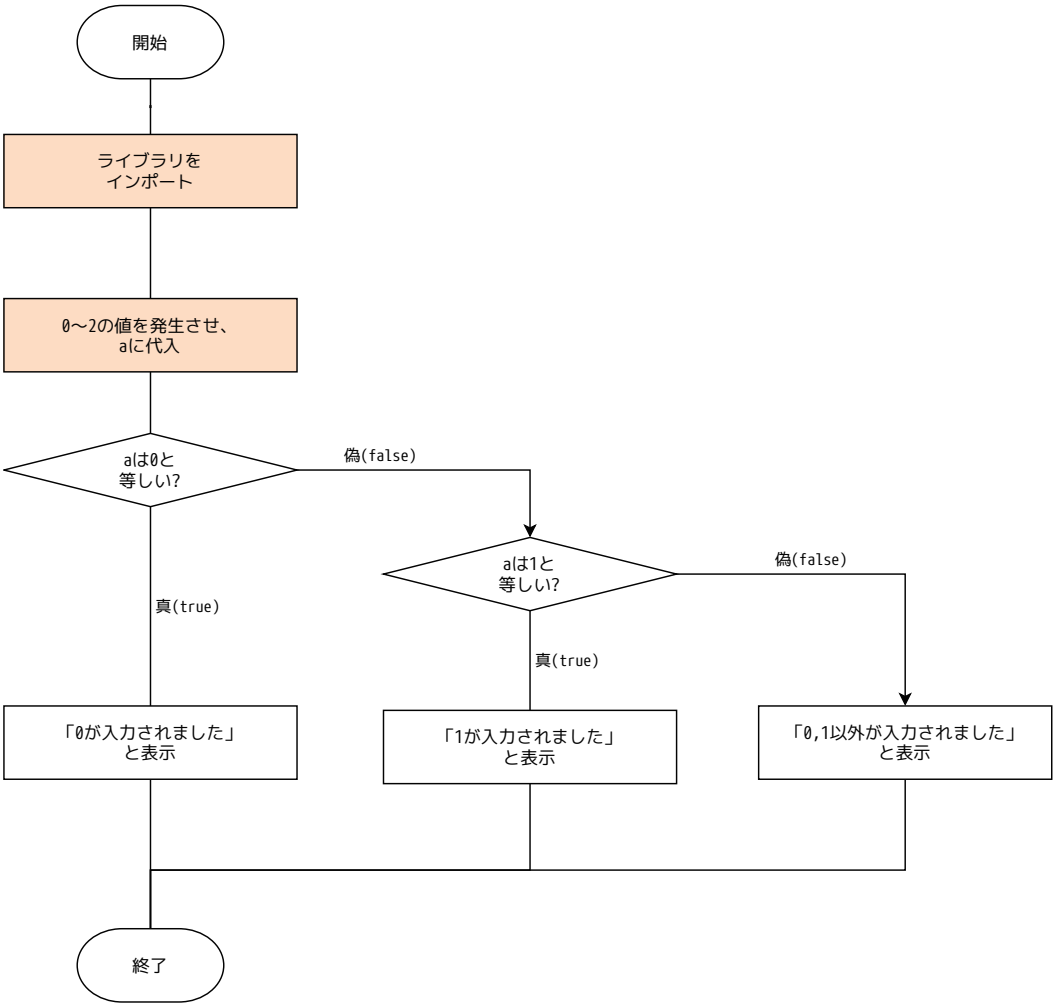
```
let a = parseInt( prompt("0 か 1 を入力してください") );
if (a == 0) {
    alert("0 が入力されました");
} else if (a == 1) {
    alert("1 が入力されました");
} else {
    alert("0,1 以外が入力されました");
}
```

プログラムが実行され、ユーザーが値を入力すると、まず if キーワードの後ろの式が評価されます。
もし「0」が入力されたなら、3 行目が実行され、「0 が入力されました」と表示し、プログラムは終了します。
もし「0」以外が入力されたなら、次の「else if (a == 1)」が実行・評価されます。「1」が入力されていたら、5 行目が実行され、「1 が入力されました」と表示します。さらに、1」以外が入力されたなら、7 行目が実行され、「0,1 以外が入力されました」と表示することになります。
if の式が先で、else if の式が後に評価されることを確認しましょう。

なお、この例では else if を一つだけ使っていますが、if の後ろに else if を必要なだけ付け足すことができます。その場合も、プログラムに書かれている順に if、else if が評価されます。else は、どの if、else if の条件にも当てはまらないときに実行されます。

プログラムにランダムな値を作らせる（乱数）

数字をランダムに並べたもの（規則性がなく、一つ一つの数字が現れる確率が等しいもの）を、乱数といいます。JavaScript を使って、擬似的に乱数を発生させることができます。
フローチャートは以下の通りです。



プログラムは次の通りです。

```
let a = Math.floor( Math.random() * 3 );
if ( a == 0 ){
    alert("0 が入力されました ");
} else if (a == 1){
    alert("1 が入力されました ");
} else {
    alert("0,1 以外が入力されました ");
}
```

1 行目の「=」の右辺にある Math.floor(Math.random() ＊ 3) の内容を順に確認しましょう。Math.floor() は、引数で渡された値の小数点以下の部分を切り捨てます。たとえば、Math.floor(1.1) とすれば、「1」が返されます。Math.floor(2.9) とすると、「2」が返されます。四捨五入ではなく、小数点以下を切り捨てる計算であることに注意してください。
Math.random()（※ Math.floor() の引数のカッコの中にある）は、0 より大きく、1 より小さい、小数点数を返します。上のプログラムでは、その値に 3 を掛けているので、0 より大きく、3 より小さい小数点数が求められます。
0 より大きく 3 より小さい値がランダムに作られ、それの小数点以下を切り捨てるので、0、1、2 のいずれかの値が作られることになります。その値を変数 a に代入します。
以後の処理は、ここまでのプログラムの例と同じ if-else if-else を使った選択構造です。
プログラムを実行すると、実行するごとに 0、1、2 のいずれかの値が変数 a に代入されるので、毎回違う結果が得られます。

例題 2：プログラムでサイコロを作る

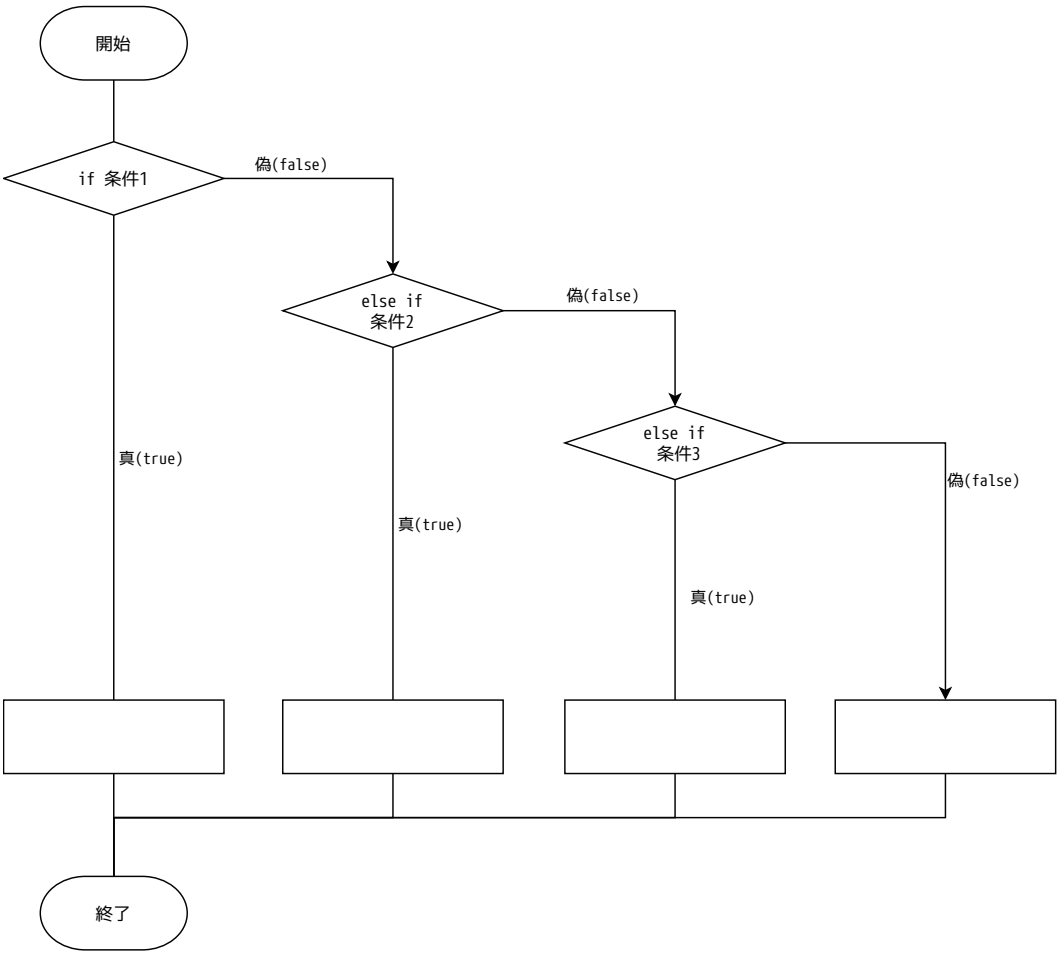
Math.floor() と Math.random() を使って、サイコロのように 1 から 6 の間の数字をランダムに表示するプログラムを作成してください。

（ヒント）Math.random() ＊ 6 とすると、0 より大きく 6 より小さい小数点数が返されます。
0 より大きく 6 より小さい小数点数を Math.floor() に渡すと、0、1、2、3、4、5 のいずれかの値が返されます。

補足資料

if-else if-else

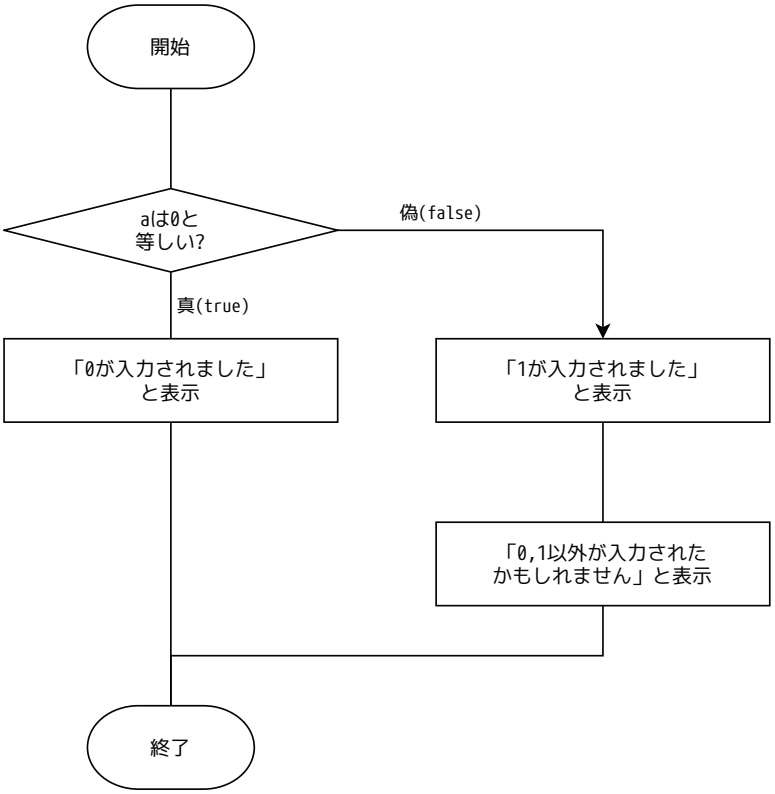
if キーワードだけ用いて、ある条件が真になるときだけ処理を実行させることができます。
else を使うと、if の条件が偽のときに実行する処理を指定することができます。
if の後に else if を使い、別の条件による判定を行い、さらに分岐させることができます。
else if はいくつでも付け足すことができます。



if-else if-else if-else if ... の最後に else を付けて、if,else if のどの条件にも当てはまらないときの処理を指定することができます。

インデント（字下げ）とブロック

次のフローチャートでは、条件分岐の else の中で、複数の処理を実行させています。



プログラムは次のようになります。

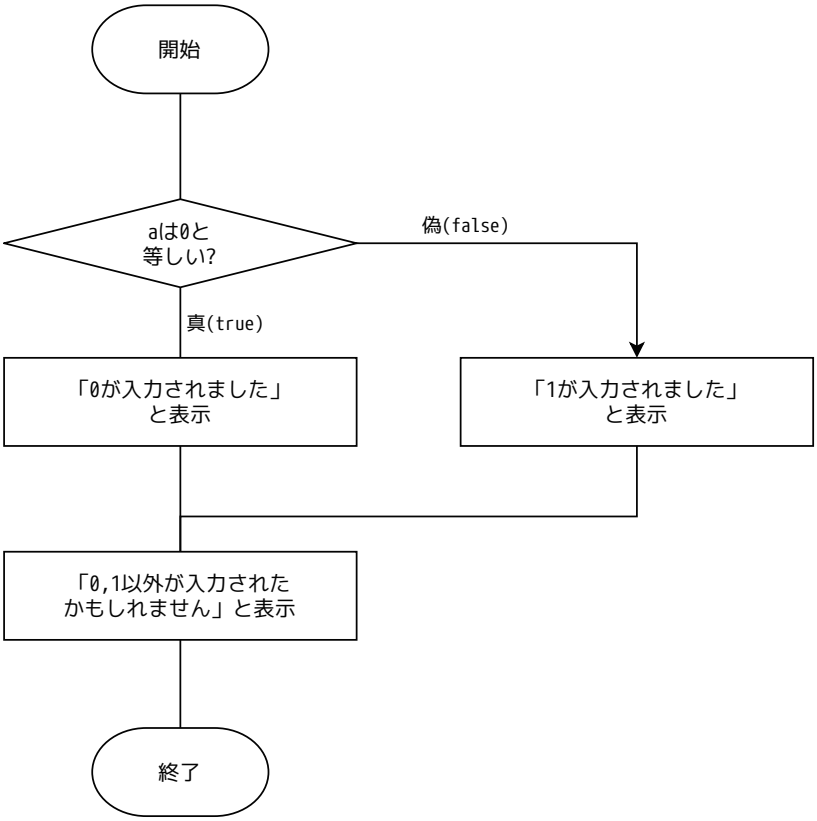
```
if (a == 0) {
    alert("0 が入力されました ");
} else {
    alert("1 が入力されました ");
    alert("0,1 以外が入力されたかもしれません ");
}
```

else の後ろにある二つの alert 文が中カッコ { } で囲まれ、インデント（字下げ）がそろえられています。中カッコ { } で囲まれている文のかたまりを、ブロックと呼びます。if や elseif、else で分岐した後、複数の処理を実行させるためにブロックを作ります。JavaScript では、中カッコ { } で囲んでブロックを作るので、インデントの深さ（空白文字の数）は任意でよいのですが、プログラムを読みやすくするために、空白文字の字数を揃えるようにします。

２つ目のブロックを次のように変更すると、動作が変わります。

```
if (a == 0) {
    alert("0 が入力されました ");
} else {
    alert("1 が入力されました ");
}
alert("0,1 以外が入力されたかもしれません ");
```

フローチャートは次のようになります。



0 が入力された場合も、0 以外の数字が入力された場合も、最後に「0,1 以外が入力されたかもしれません」と表示されます。

比較演算子

演算子	動作
==	左辺と右辺が等しいか調べる。等しければ真（true）、等しくなければ偽（false）を返す。
!=	左辺と右辺が等しくないか調べる。等しくなければ真（true）、等しければ偽（false）を返す。== の反対の結果になる。
>	a > b で、a が b より大きければ真、そうでなければ偽を返す。
>=	a >= b で、a が b 以上なら真、そうでなければ偽を返す。
<	a < b で、a が b より小さければ真、そうでなければ偽を返す。
<=	a <= b で、a が b 以下なら真、そうでなければ偽を返す。

乱数の生成

関数	動作
Math.random()	0 より大きく 1 未満のランダムな小数点数を返す。 整数の乱数が必要な場合には、計算を組み合わせる。 次の例では、0, 1, 2 のいずれかの値がランダムに取得できる。 Math.floor(Math.random() * 3) 1 を足すと、1 から 3 のいずれかの値がランダムに取得できる。 Math.floor(Math.random() * 3) + 1

第３章　配列

これまでに扱ってきた変数は、最後に代入した値を１つだけ持つことができました。
配列という仕組みを使うと、１つの配列変数に、複数の値を格納することができます。配列は、たくさんのデータを扱い、繰り返し同じ処理をさせるプログラムを作成するときに便利です。繰り返しの構造は「第 4 章 繰り返し」で扱います。先に配列を見ておくことにします。

一つの変数に複数の値を入れる

次のプログラムを実行してみましょう。

```
let youbiList = ["日", "月", "火", "水", "木", "金", "土"];
let date = new Date();
let youbi = date.getDay();
alert(youbi);
alert( youbiList[youbi] );
```

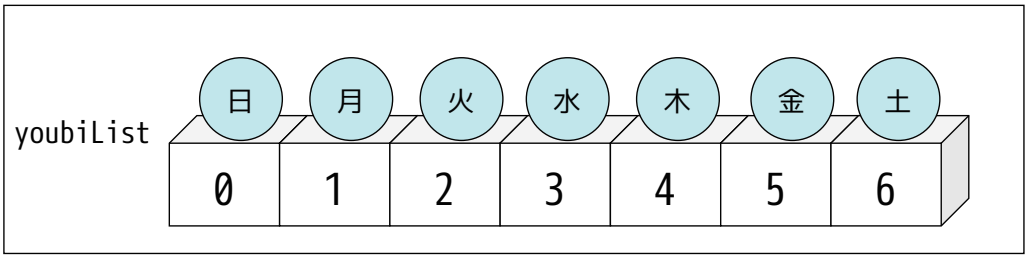
プログラムが実行された日の曜日を表す数字と、その数字が表す曜日が表示されます。

2 行目の new Date() は、プログラムが実行された年月日・時分秒を取得しています。
変数 date には日付・時刻の情報が格納されており、年・月・日、時・分・秒を取り出すことができます。
関数 getDay() は、変数 date の中から、曜日を表す数値を取り出します。

```
let date = new Date();
let youbi = date.getDay();
```

曜日を表す数値とは、日曜日を「0」、月曜日を「1」、…、金曜日を「5」、土曜日を「6」とする数値です。
プログラムの中では、文字の値よりも数値のほうが都合がよいことが多いため、関数 getDay() はこのように作られています。

次の図は、プログラムの１行目で用意している配列の変数 youbiList のイメージです。



配列の変数には複数の値を代入できます。代入している複数の値の一つ一つを**要素**と呼びます。0、１、2…の数字は**添え字**（そえじ、**インデックス**）といい、配列の変数 youbiList の要素を指し示すために使います。添え字は整数で、「0」から始まり、配列の中にある要素の数から１引いた数まで（※配列 youbiList には７個の要素がありますから、７－１で「6」まで）の値です。

例題１：　配列の変数 a を４つの要素「A、B、C、D」で作り、「a[3]」の要素を表示してみましょう。

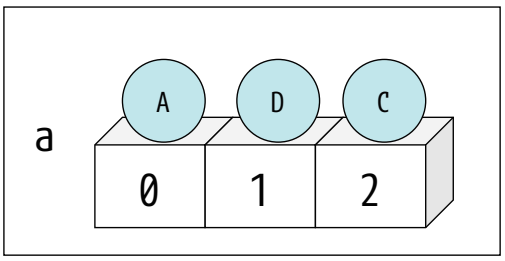
例題２：　関数 alert() のかっこの中に配列の変数を入れると、配列の全ての要素を表示します。プログラムを書いて確かめてみましょう。

（ヒント）　alert(a)

配列の要素を置き換える

「a[1] = "D"」のように、「配列名［添え字］= 新しい値」という書き方をすることによって、指定した要素に新しい値を代入することもできます。

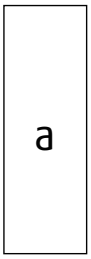
```
let a = ["A", "B", "C"];
a[1] = "D";
alert( a[1] );
```



例題３：　配列「a = ["A", "B", "C"]」について、先頭の要素を「a」に置き換え、配列 a 全体を表示するプログラムを作ってください。

空の配列を作り、要素を追加する

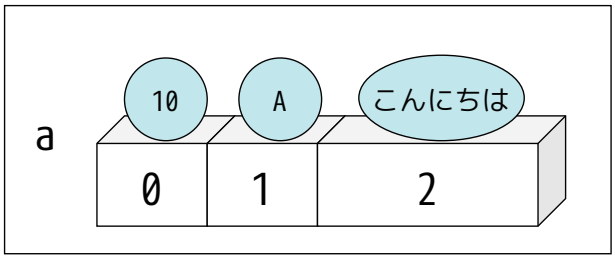
次のプログラムの１行目を見てみましょう。「a = []」という記述があります。これは空の配列を作る処理です。



最初は要素は１つもありません。

```
let a = [];
a.push(10);
a.push("A");
a.push(" こんにちは ");
alert( a[2] );
```

2 行目から、「a.push()」という記述が続きます。これは配列 a の後ろに追加しています。
2 行目から 4 行目までを実行すると、関数 push() を 3 回行ったので、次のような状態になります。



要素を追加した後の利用方法は、最初のプログラムと同じです。「配列名［添え字］」という書き方で値を利用することができます。

例題 4:　配列 aisatsuList を空で作成した後、「おはよう」、「こんにちは」、「こんばんは」と要素を追加し、最後に配列 aisatsuList 全体を表示するプログラムを作ってください。

配列の好きな場所に要素を追加する

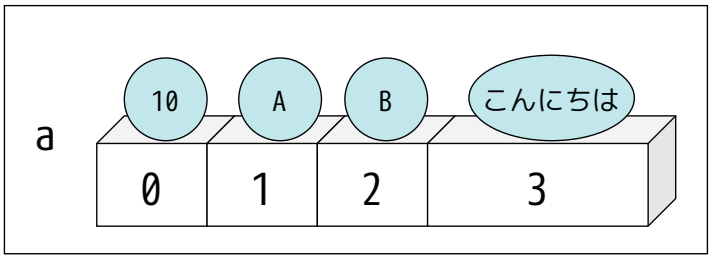
配列の好きな場所に要素を追加するには、関数 splice() を使います。

```
a.splice(2, 0, "B");
```

2 番目の要素の前に「B」を挿入するという意味になります。
1 つ目の引数の「2」は、配列 a の要素を指す添え字です。2 つ目の引数は配列の要素を削除するときに使います。今回は削除を行わないように「0」を指定しています。3 つ目の引数の「B」は追加する要素の値です。

```
let a = [];
a.push(10);
a.push("A");
a.push(" こんにちは ");
a.splice(2, 0, "B");
alert( a[2] );
```

添え字は 0 から始まることを思い出しましょう。



配列の要素を削除する

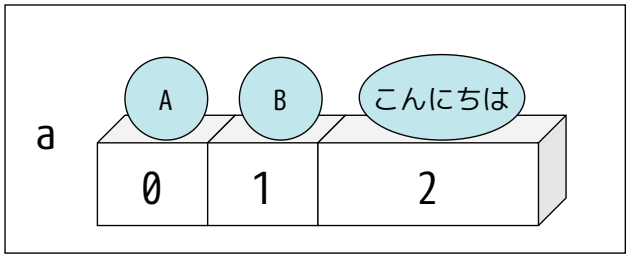
配列の要素を削除する時にも、関数 splice() を使います。

```
a.splice(0, 1)
```

１つ目の引数の「0」は、配列 a の要素を指す添え字です。１つ目の引数の「1」は、削除する要素数を指定します。添え字で指定した要素から指定した個数の要素を削除し、削除した要素よりも後にある要素を前に詰めます。

```
a = [];
a.push(10);
a.push("A");
a.push(" こんにちは ");
a.splice(2, 0, "B");
a.splice(0, 1);
alert( a[2] );
```

5 行目の「a.splice(2, 0, "B")」の後に、6 行目で「a.splice(0, 1)」を実行した結果、配列の内容は次のように変わります。



例題 5:　配列 aisatsuList を空で作成した後、「おはよう」「こんにちは」「こんばんは」と要素を追加します。次に、添え字が 1 の要素を削除します。最後に配列 aisatsuList のすべての要素を表示するプログラムを作ってください。結果はどうなるでしょうか。

補足資料

配列を利用・操作するための機能

機能名	機能
配列名 = []	空の配列を作成し、変数に代入する。 例： a = []
配列名 = [要素 1, 要素 2, …]	要素 1、要素 2、・・・を持つ配列を作成し、変数に代入する。 例： a = [1, 2, 5]
配列名 [添え字]	配列の要素を指定する。配列の要素の値を取得したり、配列の要素に代入できる。添え字は 0 から始まる数値。つまり、0 番目から数える。 例： a = [1, 2, 5] のとき、a[1] -> 2
配列名 .length	配列の要素数を調べる。 例： a = [1, 2, 5] のとき、a.length -> 3
配列名 .push(要素)	配列に、指定している要素を付け足す。要素は配列の最後に追加される。 例： a = [1, 2, 5] のとき、 a.push(10) -> a = [1, 2, 5, 10]
配列名 .pop()	配列の最後の要素を取り出す。その要素は配列から取り除かれる。 例 :a = [1, 2, 5] のとき、 a.pop() -> 5 、配列の中身は [1, 2]

機能名	機能
配列名 .splice(添え字)	添え字で指定した位置よりも後ろの要素全てを削除する。 例：a = [1, 2, 5] のとき、 a.splice(1) -> a = [1]
配列名 .splice(添え字 , 要素数)	添え字で指定した位置よりも後ろの要素について、2 つ目の引数で指定した要素を削除する。 例：a = [1, 2, 5] のとき、 a.splice(1, 1) -> a = [1, 5]
配列名 .splice(添え字 , 0, 要素)	添え字で指定した位置に、3 つ目の引数で指定した要素を挿入する。 例：a = [1, 2, 5] のとき、 a.splice(2, 0, 4) -> a = [1, 2, 4, 5]

問題集

問 1.　次のプログラムを実行したとき、どのように表示されるでしょうか？実際にプログラムを書き、動作を確認してください。

```
let a = ["A", "B"];
document.write("a[1]:" + a[1]);
document.write("<br>");

a.push("C");
document.write("a[1]:" + a[1] + "<br>");
document.write("a[2]:" + a[2] + "<br>");
document.write("<br>");

a.splice(2, 0, "b");
document.write("a[1]:" + a[1] + "<br>");
document.write("a[2]:" + a[2] + "<br>");
document.write("<br>");

b = a.pop( );
document.write("a[1]:" + a[1] + "<br>");
document.write("a[2]:" + a[2] + "<br>");
document.write ("b:" + b);
```

問 2.　次のように動作するプログラムを作成してください。

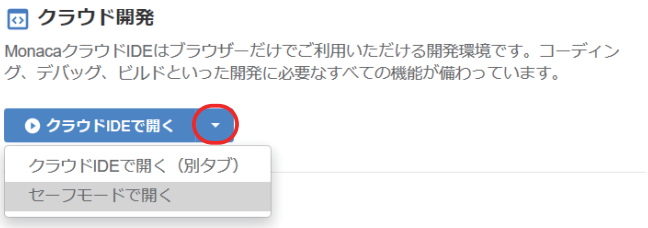
① 配列の変数 b を作る。
② b に要素として 1,2,3,4,5 を追加する（5 個の要素を含む配列になる）。
③ 配列 b の要素を取り出して、順に 1,2,3,4,5 と表示する。
④ 配列 b を空の配列にする。

第４章　繰り返し（反復）

ここまでのプログラムでは、上から順に（順次構造）、または条件に合わせて選択（分岐構造）して、処理を進めました。
この章では、同じ処理を決まった回数だけ繰り返し実行させたり、条件を満たすまで繰り返し実行させる構造を学びます。これを**繰り返し構造**または**反復構造**といいます。

前の「第 3 章 配列」で配列を扱いました。配列に格納されている複数の要素のそれぞれに対して、同じ処理を実行させたい場合、この繰り返し構造が役に立ちます。

なお、無限ループを発生させてクラウド IDE を開くことができなくなる場合があります。その場合は「セーフモード」で開いて問題の箇所を修正して下さい。セーフモードは「クラウド IDE で開く」ボタンの右にある逆三角のボタンを押下することで選択できます。



決まった回数だけ繰り返し実行する

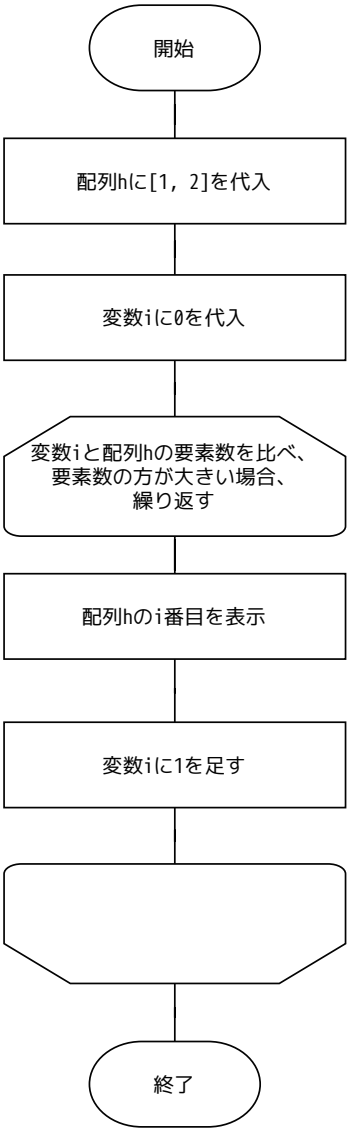
次のプログラムを実行してみましょう。なお、3 行目の先頭では半角スペースを 4 個入れて、字下げ（インデント）をしています。

```
let h = [1, 2];
for (let i = 0; i < h.length; i++){
    document.write(h[i] + "<br>");
}
```

結果は次のようになります。

```
1
2
```

フローチャートは次の通りです。



プログラムの１行目では、配列ｈに２つの要素を代入しています。結果、配列ｈには「1」と「2」という２つの要素が含まれています。

２行目は for 文といいます。実行時には、次のように振る舞います。

① 最初に１回だけ、変数ｉに０を代入する。
② 変数ｉの値と、配列ｈの要素数を比べ、変数ｉの方が小さければ、繰り返し処理を実行する。変数ｉの値が配列ｈの要素数以上ならば、繰り返し処理を終了する。
③ （繰り返し処理）配列ｈのｉ番目の要素を表示する。
④ （繰り返し処理）変数ｉの値に１を足す。
⑤ ②に戻り、繰り返しを続けるか繰り返しを終了するか判定する。

補足：for 文の中にある式「i++」は、「i = i + 1」という式を短く書く方法です。変数 i の値に 1 を足し、その結果を変数 i に代入します。つまり、変数 i の値を 1 増やします。同様に「i--」と書くと、変数 i の値を 1 減らす計算が出来ます。

次のプログラムでは、配列 h の i 番目の要素を表示した後、「おわり」と表示しています。

```
let h = [1, 2];
for (let i=0; i < h.length; i++){
    document.write(h[i] + "<br>");
    document.write(" おわり <br>");
}
```

「変数 i の値を表示した後、『おわり』と表示する」という２つの処理を、繰り返しています。

```
1
おわり
2
おわり
```

例題 1：　上のプログラムを次のように修正すると、実行結果はどうなるでしょうか。

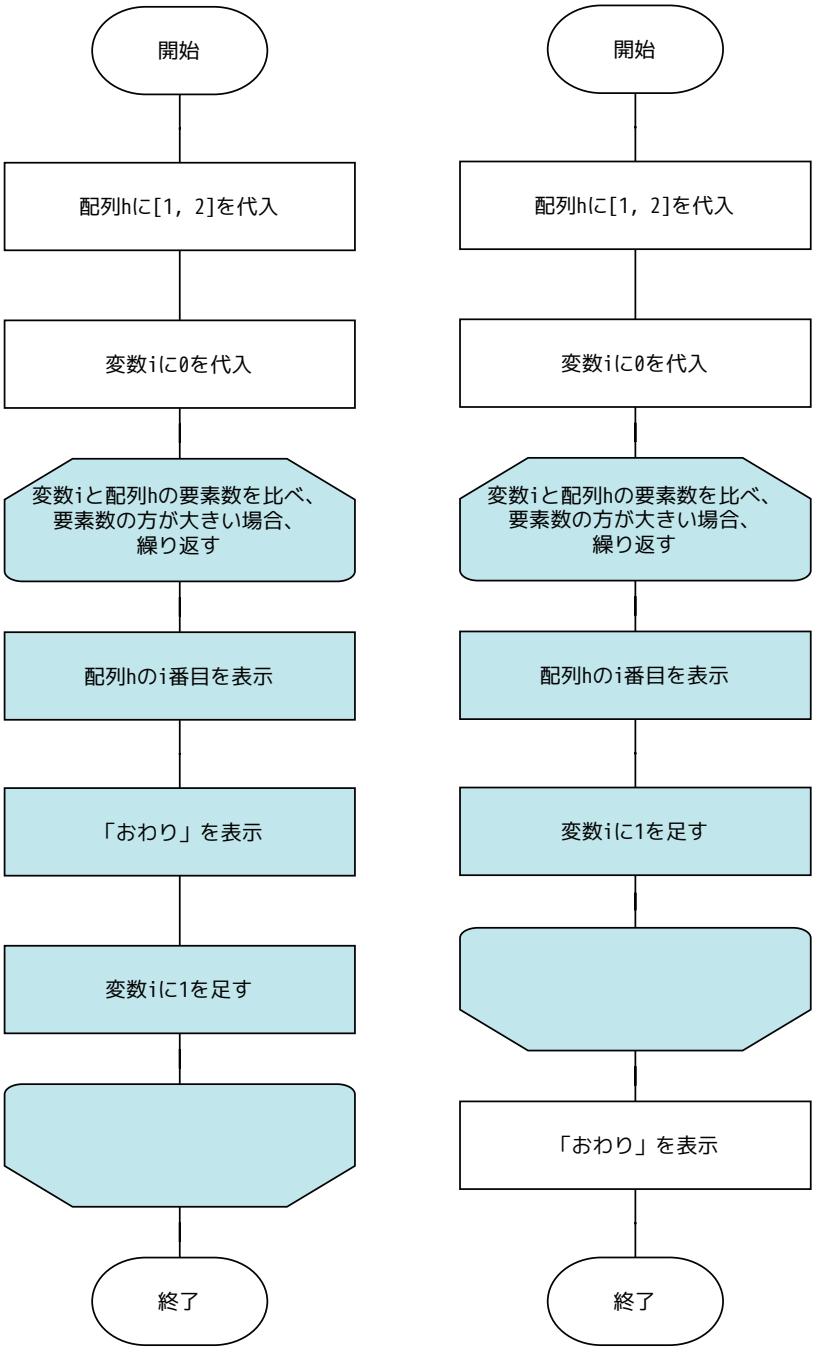
（修正前）

```
let h = [1, 2];
for (let i=0; i<h.length; i++){
    document.write(h[i] + "<br>");
    document.write(" おわり <br>");
}
```

（修正後）4 行目の（" おわり "）の document.write( ) を、for 文のブロックの後ろに出しています。

```
let h = [1,2];
for (let i=0; i<h.length; i++){
    document.write(h[i] + "<br>");
}
document.write(" おわり <br>");
```

実行結果をみて、何が起きたか説明してみましょう。



修正前のフローチャート（左）と、修正後のフローチャート（右）とを見比べてみましょう。
何を繰り返して実行し、何を繰り返さないで実行するでしょうか。

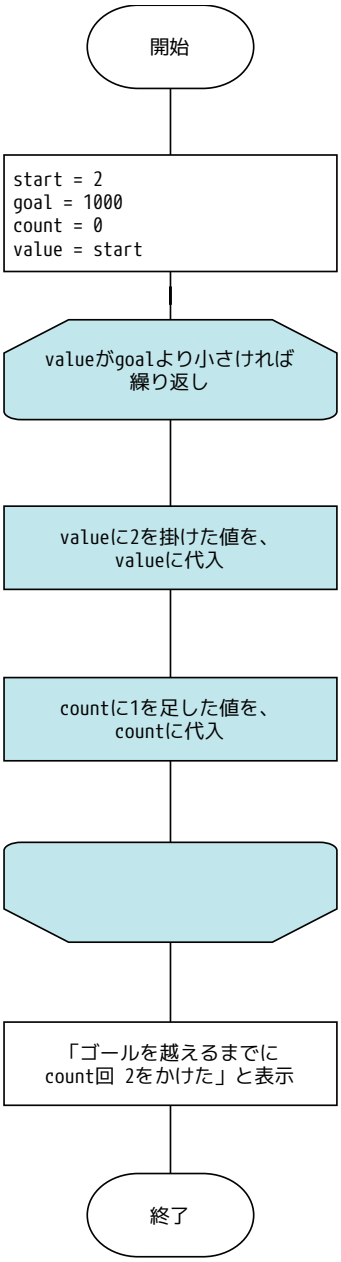
例題 2:　for 文を使って「1」から「4」まで表示するプログラムを作ってください。

例題 3:　for 文を使って、「4」から「1」まで 1 ずつ減らしながら表示するにはどうしたらよいでしょうか？

条件を満たすまで繰り返し実行する

あらかじめ、何回繰り返すか想定できる場合、for 文を使うと簡単です。
一方、プログラムを開始した後でないと、何回繰り返すか分からない場合、次に紹介する while 文を使うほうが、書きやすくなります。
たとえば、「ユーザーからの入力を繰り返し受け付ける。ただし、ユーザーが 0 を入力したら、終了する」というプログラムの場合、ユーザーが値を入力するまで、何回繰り返すか決まりません。この場合には、while 文を使います。

次のプログラムを見てみましょう。「2」の値をスタートにして、何回 2 を掛け算したらゴールの「1000」を超えるか調べるプログラムです。言い換えると「2 を何乗すると、1000 を超えるか？」という問題なので、数学的に解くこともできますが、プログラムで解いてみましょう。
次のような解き方があります。まず、フローチャートを示します。


開始
start = 2
goal = 1000
count = 0
value = start
valueがgoalより小さければ
繰り返し
valueに2を掛けた値を、
valueに代入
countに1を足した値を、
countに代入
「ゴールを越えるまでに
count回 2をかけた」と表示
終了

続いて、プログラムです。

```
let start = 2;
let goal = 1000;
let count = 0;
let value = start;
while ( value < goal ) {
    value = value * 2;
    count = count + 1;
}
document.write(" ゴールを超えるまでに " + count + " 回 2 をかけた ");
```

1 行目： 変数 start に 2 を代入しています。この変数は最初の値です。

2 行目： 変数 goal に 1000 を代入しています。この変数は目標の値です。

3 行目： 変数 count をゼロにしています。この変数は、掛け算をした回数を数えるために使います。

4 行目： 変数 value は、掛け算の答えです。1 回掛け算をするたびに、この value が 2 倍になっていきます。最初は変数 start の値を代入しています。ですから最初の value は「2」です。

5 行目： while 文です。while の後ろに、かっこで囲んだ式を書きます。式「value < goal」は、変数 value と、変数 goal を比べています。変数 goal のほうが大きければ真（true）、value のほうが大きければ偽（false）になります。while 文は、式が真（true）である間、続く処理を繰り返します。

6 行目： 式の右辺に注目しましょう。変数 value に「2」を掛けています。その値を、左辺の変数 value に代入しています。変数 value は、この行が実行される前に比べて 2 倍になります。

7 行目： 式の右辺に注目しましょう。変数 count に「1」を足しています。その値を、左辺の変数 count に代入しています。変数 count は、この行が実行される前に比べて 1 大きい値になります。

ここで、5 行目の while 文に戻ります。再び変数 value と変数 goal の値を比べ、変数 goal の値の方が大きければ 6 行目、7 行目の処理を繰り返します。変数 value の値が大きければ while の繰り返し処理の先に進みます。

9 行目： ここまでに数え上げた変数 count の値を表示して、プログラムを終えます。

while の式

キーワード while の後ろにさまざまな式を書くことで、さまざまな繰り返しの制御を行うことができます。
式を書くときには「第 2 章 条件による選択（分岐）」の章で学んだ比較演算子を使います。

（例：「a」と「b」の値が等しいときに繰り返す）

```
while (a == b)
```

（例：「a」と「b」の値が等しくないときに繰り返す）

```
while (a != b)
```

例題 4: while 文と Math.random() と Math.floor() を使って、「1」から「10」までの整数を生成し、その値を、その回数だけ表示するプログラムを作ってください。

ヒント：

```
let a = Math.floor( Math.random() * 10 ) + 1;
```

実行結果の例 (1):「Math.floor(Math.random( ) * 10) + 1;」で「5」が出た場合、5 回繰り返し表示する

```
a: 5
a: 5
a: 5
a: 5
a: 5
```

例題 5: for 文と Math.random() と Math.floor() を使って、「1」から「10」までの整数を生成し、その値を、その回数だけ表示するプログラムを作ってください。

補足資料

for 文

構文	説明

for(初期化式 ; 条件式 ; 更新式){
　　処理 1
　　処理 2
　　…
}
処理 A
	初期化式を実行する（初回のみ）。
	条件式が真なら繰り返す。条件式が偽なら繰り返さない。
	処理１、処理 2 を実行。
	条件式が偽になり、繰り返しを終えた後、処理 A を実行する。

while 文

構文	動作

while(条件式) {
　　処理 1
　　処理 2
　　…
}
処理 A
	「式」を実行･評価して､真(true)なら処理 1 に進む。処理 2､…と進めて､{} の中に書かれている一連の処理を実行する。
	ブロックの処理が終わったら、ふたたび while に戻り、条件式を実行・評価する。真なら処理１に進み、偽（false）なら処理をせずに、次に進む。
	左の例であれば、処理 A に進む。
	なお、最初に「式」を実行・評価したとき、偽であったら、ブロックは１回も実行されずに、処理 A に進む。

問題集

問 1.　for 文を使って、11 から 20 までの整数を表示するプログラムを作成してください。

問 2.　for 文を使って、0 から 100 までの整数のうち、3 の倍数だけを表示するプログラムを作成してください。

（ヒント）if 文を組み合わせることで実現できます。

問 3.　while 文を使って、次のように動作するプログラムを作成してください。完成したプログラムは、どんな計算をしているといえるでしょうか？

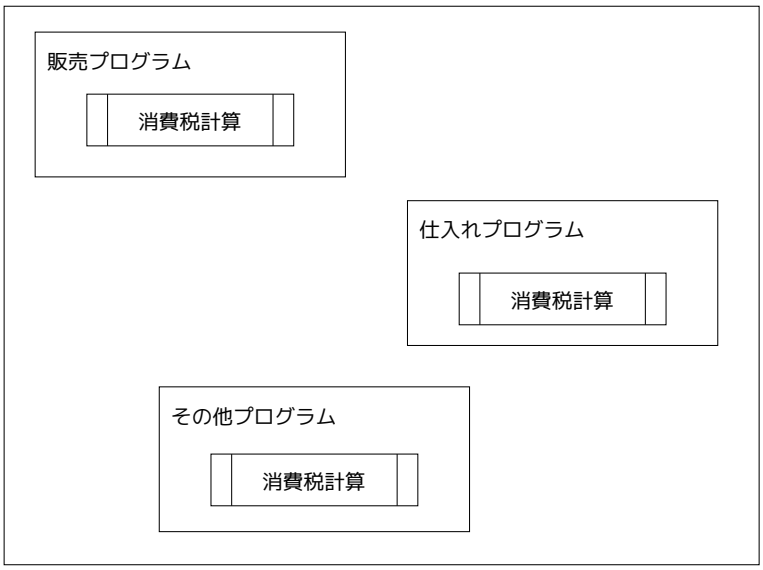
① 1 から 6 の整数をランダムで生成し、変数 a に代入する。
② 変数 b に 0 を代入する。
③ while 文を書き、a がゼロより大きい間、次の処理を繰り返すように、式を書く。
   1. a の値を b の現在の値に足し、結果を b に代入する（ヒント :b に b + a の結果を代入する）。
   2. a の値から 1 引き、結果を a に代入する（ヒント :a の値を 1 減らす）。
④ while 文が終了したら、最後に b の値を表示する。

第５章　関数の定義と利用

ここまで、alert() や document.write() などの関数を利用してきました。これらの関数は、JavaScript 言語があらかじめ用意している関数です。
必要な関数を自分で作って、それを利用することもできます。

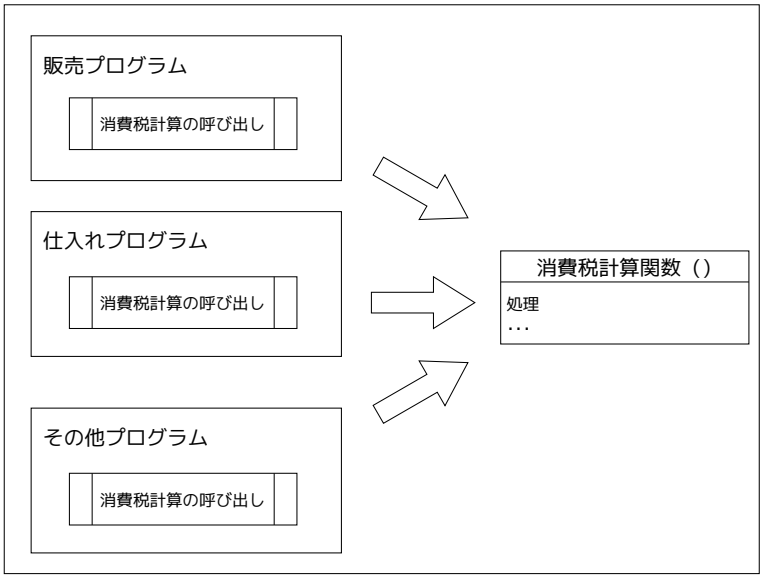
関数とは

プログラムの中で、同じ処理や計算を、複数の個所で行いたいことがあります。たとえば、商品の販売や仕入を管理するプログラムであれば、「金額に対応する消費税を計算する」ことは、プログラムの中の複数の場所で必要になるはずです。



何度も使いたい計算･処理を、必要になるたびにプログラミングするのは面倒です。また、計算･処理の内容を変えるときは、プログラムに書いた個数だけ修正しなければなりません。

プログラム言語には、**関数**（function）という仕組みがあります。
関数を作ることを、関数を**定義**するといいます。
また、あるプログラムから、別の場所にあるプログラムや関数を指定して、自分のプログラムの一部のように実行させることを「呼び出す」と言います。必要なときに作成済みの**関数を呼び出す**ことで、関数の中に書いた一連の処理を、何度も実行させることができます。



たとえば、消費税を計算する関数を作っておけば、必要なときに呼び出すだけになります。
修正が必要になったときも、消費税の計算をする関数だけを作り替えれば済みます。

組み込み関数

ここまでのプログラムでも、関数を利用していました。JavaScript をはじめ、多くのプログラミング言語には言語自体が用意している関数が組み込まれています。あらかじめ用意されている関数であれば、プログラムの中で関数を定義しないでも使うことができます。

- prompt()
- alert()
- document.write()
- Math.random()

これらの関数を、**組み込み関数**と呼びます。ユーザーからの入力を受け付けたり（関数 prompt()）、値を出力したり（関数 alert()，関数 document.write()）、乱数を生成したり（関数 Math.random()）することができます。
組み込み関数は、多くのプログラムで必要になる機能を、あらかじめ用意してくれています。

例題 1：「なにか言葉を入力してください：」と表示して入力を促した後、ユーザーが入力した値をオウム返しにそのまま表示するプログラムを作ってください。

（ヒント）prompt(" 表示するメッセージ ")，alert(変数)

日付・時刻

組み込み関数には「どんな用途のプログラムでも必要になる機能」が用意されています。たとえば、日付や時刻の計算をする組み込み関数があります。次のプログラムを見てみましょう。

```
let hiduke_jikoku = new Date();
alert(hiduke_jikoku);
```

1 行目の「new Date()」で、プログラム実行時の日付・時刻を取得しています。

次のプログラムは、このプログラムを実行したその日・その時の、日付・時刻を取得します。

```
let hiduke_jikoku = new Date(); // プログラム実行時の日付・時刻を取得
ji = hiduke_jikoku.getHours(); // 時を取得
alert(ji);
hun = hiduke_jikoku.getMinutes(); // 分を取得
alert(hun);
byou = hiduke_jikoku.getSeconds(); // 秒を取得
alert(byou);
```

1 行目の「new Date()」で、現在（※プログラムを実行した瞬間）の日付・時刻を、プログラムを実行しているコンピュータから取得しています。

例題 2: プログラムを実行したその日の、年・月・日を表示するプログラムを作成してください。

（ヒント）

```
let hiduke_jikoku = new Date();
```

年：hiduke_jikoku.getFullYear(),
月：hiduke_jikoku.getMonth(),
日：hiduke_jikoku.getDate()
※ getMonth() で取り出される値は、「0」から「11」です。
※ 日付・時刻は、プログラムを実行しているコンピュータから取得しています。そのため、コンピュータの日付・時刻の設定が誤っている場合は、JavaScript プログラムも誤った値を取得することになります。

関数の定義と呼び出し

関数を定義するには、function キーワードを使います。function を使って関数を作る文法は次の通りです。簡単な関数を作り、呼び出しながら、学んでいきましょう。

```
function 関数名 ( 引数 1, 引数 2, ...){
    処理 ;
    return 戻り値 ;
}
```

まず、次のプログラムを見てみましょう。

```
function aisatsu(){
    alert(" こんにちは ");
}

aisatsu();
```

1 行目： aisatsu() という名前の関数を定義しています。
2 行目： {} の中に書かれていることに注意してください。この alert(" こんにちは ") は、関数 aisatsu() の中に含まれています。
5 行目： 関数 aisatsu() を呼び出しています。

例題 3: 上のプログラムで定義した関数 aisatsu() の呼び出し（※最後の行）を 2 回にした場合と、0 回にした場合に、実行結果がどうなるか確認してみましょう。

関数の引数と戻り値

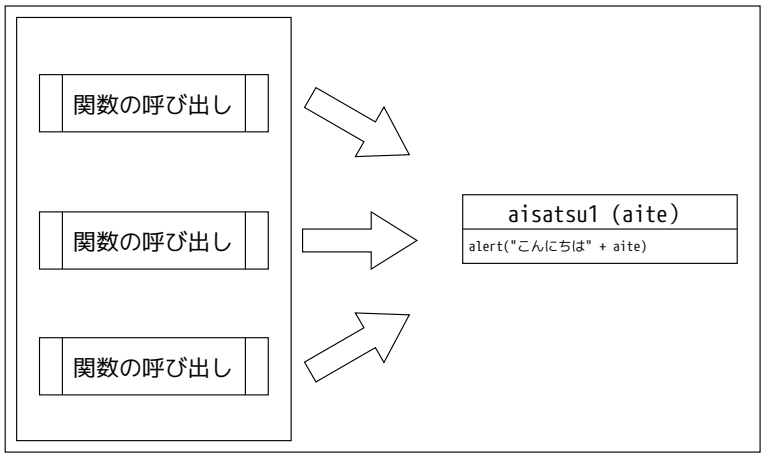
関数に値を渡し、関数内の処理で使うことができます。引数（ひきすう）といいます。

```
function aisatsu1(aite){
    alert(" こんにちは " + aite);
}

aisatsu1("JavaScript");
```

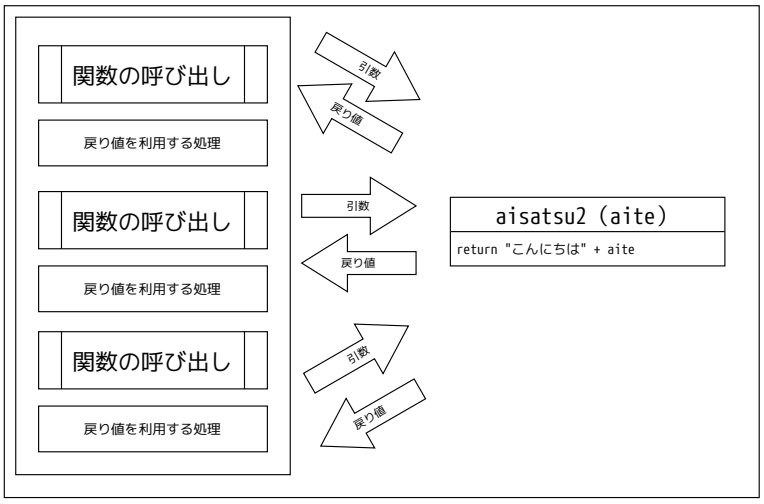
関数から、呼び出し元に値を返すこともできます。これには return キーワードを使います。
この返す値を戻り値（もどりち）といいます。

```
function aisatsu2(aite){
    return " こんにちは " + aite;
}

let kotoba = aisatsu2("JavaScript");
alert( kotoba );
```

上の 2 つのプログラムは、実行結果は同じです。しかし、関数の使い方は異なります。
上のプログラムの関数（aisatsu1）は引数で渡された値を使って画面に表示をしています。
下のプログラムの関数（aisatsu2）は、引数で渡された値と、関数の中に持つ値（”こんにちは ”）をつなげて、返しています。
関数 aisatsu1 は、表示するためにしか使えない関数です。やりたい処理が一つだけなら、とても簡潔です。



一方、関数 aisatsu2 は、返された文字列をどう使うか、呼び出し側（値を返してもらう側）で決めることができます。



どちらが良いということではなく、必要に応じて使い分けることが大切です。

例題 4: 金額を引数で渡すと消費税額を返す関数と、その関数を呼び出すプログラムを作ってください。

補足資料

組み込み関数 日付

組み込み関数	説明
new Date()	let today = new Date(); プログラムを実行した時点の日付・時刻を取得する。

today.getFullYear()	Date オブジェクトの持つ日付・時刻の「年」を返す。
	let today = new Date();
	let fullYear = today.getFullYear();

today.getMonth()	Date オブジェクトの持つ日付・時刻の「月」(0-11) を返す。
	let today = new Date();
	let month = today.getMonth();

today.getDate()	Date オブジェクトの持つ日付・時刻の「日」(1-31) を返す。
	let today = new Date();
	let date = today.getDate();

today.getDay()	Date オブジェクトの持つ日付・時刻の「曜日」を返す。
	(0 が日曜日、1 が月曜日、…6 が土曜日)
	let today = new Date();
	let day = today.getDay();

組み込み関数 時刻

組み込み関数	説明

today.getHours()	Date オブジェクトの持つ日付・時刻の「時」(0-23) を返す。
	let today = new Date();
	let fullYear = today.getHours();

today.getMinutes()	Date オブジェクトの持つ日付・時刻の「分」を返す。
	let today = new Date();
	let month = today.getMinutes();

today.getSeconds()	Date オブジェクトの持つ日付・時刻の「秒」を返す。
	let today = new Date();
	let date = today.getSeconds();

組み込み関数の例

構文	説明
`Math.floor()`	引数の値の小数点以下を切り捨てる。
`Math.random()`	`0` から `1` の範囲で、ランダムな浮動小数点数を返す。 引数は取らない。
`Math.sqrt( 値 )`	引数の値の平方根を返す。
`parseInt()`	引数に渡した文字列から、整数のデータを作る。
`parseFloat()`	引数に渡した文字列から、浮動小数点数のデータを作る。

問題集

問 1.　組み込み関数を使い、次のようなプログラムを作成してください。

①　あいさつの言葉を三つ含むリストを作る。

（ヒント）aisatsu = [" おはよう ", " こんにちは ", " こんばんは "]

②　Math.random() を使い、ランダムに 0 から 2 の値を生成し、変数 rand に格納する。

③　リスト aisatsu の中から、変数 rand の値を使って要素を取り出し、画面に表示する。

問 2.　次の機能を持つ関数を定義して、それを利用するプログラムを作成してください。

・　引数で、金額と税率を渡す。

・　戻り値として税額を返す。

（税額計算処理）

関数名：zeigaku_keisan()

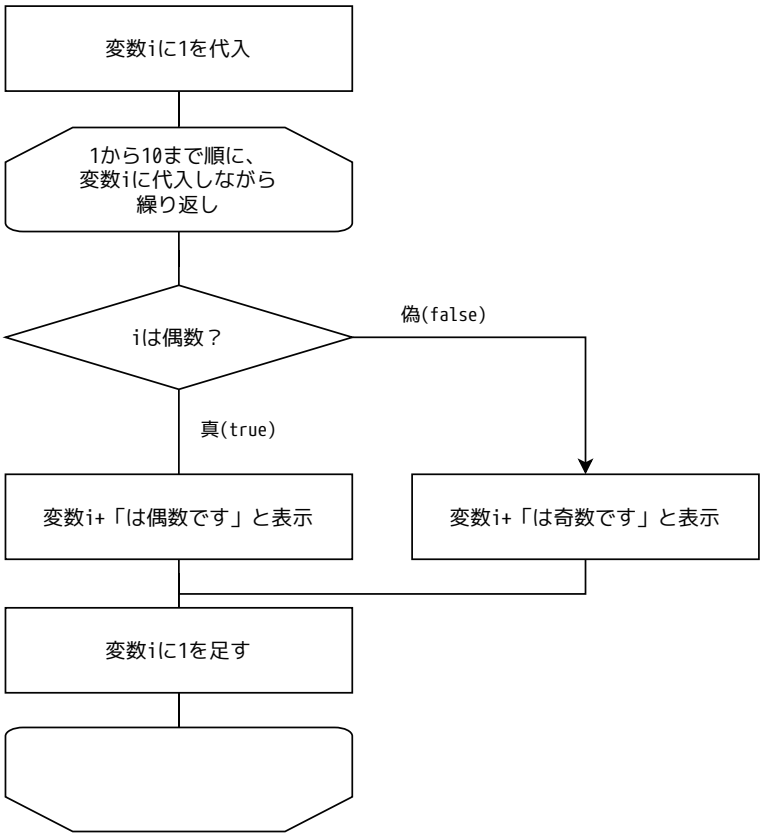
引数：kingaku, zeiritsu

第６章　繰り返し（反復）と選択（分岐）の組み合わせ

これまでに学んだ繰り返し（反復）の制御と、選択（分岐）の制御を組み合わせると、複雑なプログラムを作ることができます。プログラムで何か問題を解決したい場合、制御の組み合わせが必要になります。

繰り返し（反復）と選択（分岐）の組み合わせ

次のように振る舞うプログラムを考えてみましょう。
「1 から 10 までの数字を表示する。ただし、奇数の場合は『1 は奇数です』、偶数の場合は『2 は偶数です』と表示する」
まず、フローチャートから考えます。



繰り返し（反復）構造の中に、選択（分岐）構造があります。
JavaScript プログラムでは次のように書きます。ブロックに注目してください。

```
for (let i = 1; i <= 10; i++){
    if (i % 2 == 0){
        document.write(i + "は偶数です<br>");
    } else {
        document.write(i + "は奇数です<br>");
    }
}
```

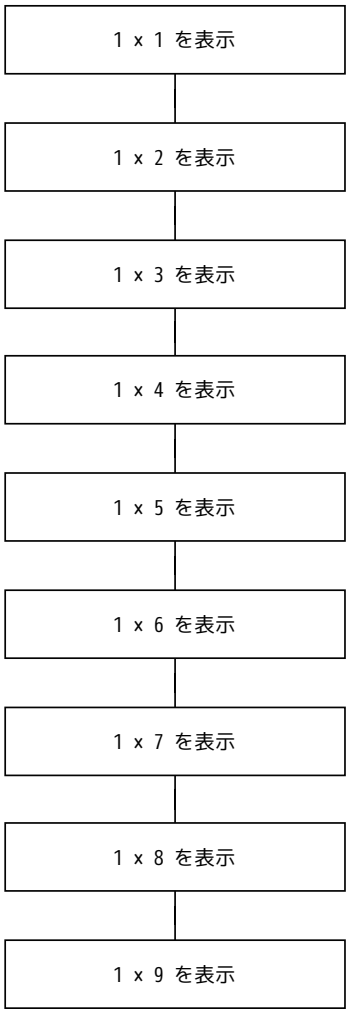
1行目では、変数iに1を代入します（※これは最初の1回だけ実行されます）。変数iの値が10以下の場合繰り返します。一連の繰り返し処理の後、変数iの値に1を足します。
2行目以降、繰り返し実行する処理は、中カッコ{ }の中に書かれています。そこにif文を書くことで、「繰り返しの中で、条件に合わせて処理を分岐する」というプログラムになります。
if文の式では、変数iの値を「2」で割り、余りを求める計算をし、余りが「0」かどうか比較します。余りが0なら偶数なので、関数document.write( )で「偶数です」と表示させます。
else以下では、「変数iの値を2で割った余りが0でないとき」に、「奇数です」と表示させています。
次のような実行結果になります。

```
1は奇数です
2は偶数です
3は奇数です
4は偶数です
5は奇数です
6は偶数です
7は奇数です
8は偶数です
9は奇数です
10は偶数です
```

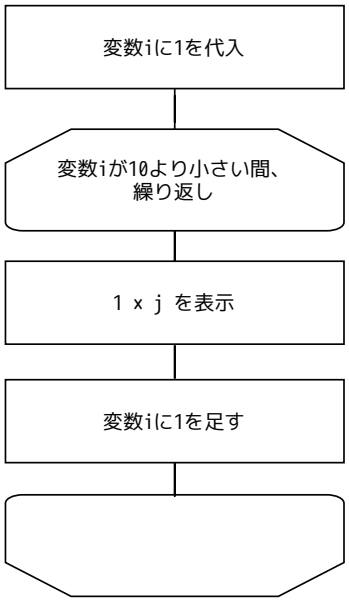
例題：1　繰り返しの構造と、選択の構造を使って、1から30までの間の、3の倍数だけ表示するプログラムを作成してください。

「繰り返し」を繰り返す

掛け算の「九九」をプログラムで表現するにはどうしたらよいでしょうか？
問題をいくつかの部品に分けて考えてみましょう。
まず、一の段から考えます。



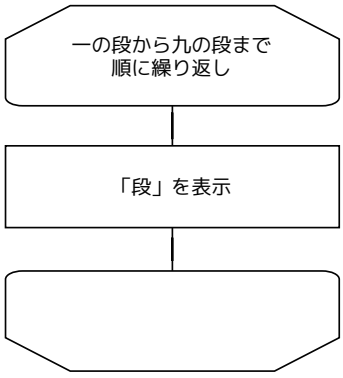
このままでも一の段を計算していますが、「1 に、1 から 9 の値を（繰り返して）掛けている」という構造を見つけられれば、次のようにフローチャートを作ることができます。



プログラムで表現してみましょう。

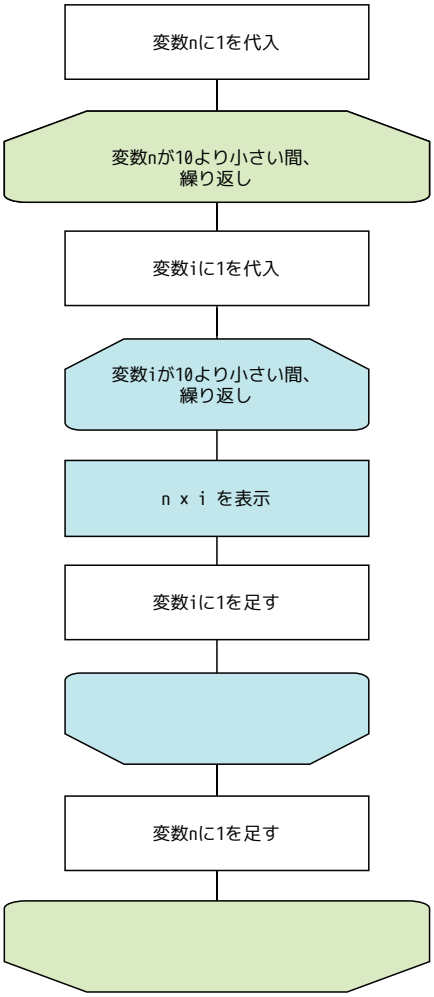
```
for (let i = 1; i < 10; i++){
    document.write(1 * i + " ");
}
```

ところで、九九は一の段の次に、二の段、三の段と続き、九の段まで繰り返します。



「『段』を表示する」の中身は、先に見た「一の段」の内容とほぼ同じことを思い出しましょう。違うのは、変数 i に掛ける値が変わっていく（一、二、三、…、九）ところです。

フローチャートは次のようになります。

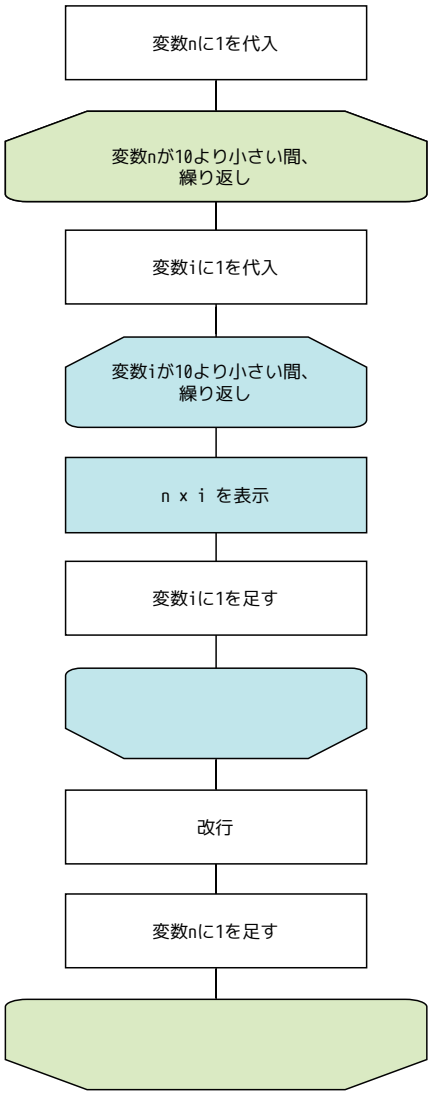


プログラムにすると次のようになります。
結果を見やすくするために、「の段」という表示をする処理を付け足しています。また、１つの段を表示した後に改行を表示して、次の段の表示が始まったことが分かるようにしています。

```
let kotae = 0;
for (let n = 1; n < 10; n++){
    document.write( n + " の段 :");
    for (let i = 1; i < 10; i++){
        kotae = n * i;
        document.write(kotae + " ");
    }
    document.write("<br>");
}
```

最後の８行目にある、document.write("<br>") は、単に改行だけするために書かれています。

追加した処理をフローチャートに書き込むと、次のようになります。
外側の繰り返し（段を繰り返す）の中で「の段」を表示し、内側の繰り返し（特定の段の、×１から×９までの計算結果を表示する）を実行します。内側の繰り返しが終わったら、改行を表示して、外側の繰り返しの次（の段）に進む、という制御になっています。



繰り返しの中に選択を組み込むことができるのと同じように、繰り返しの中に繰り返しを入れることができます。この構造は繰り返し（ループ）が二重になっているので、**二重ループ**と呼ぶことがあります。

例題 2:　配列にあいさつの言葉を 3 つ（「おはよう」「こんにちは」「こんばんは」）入れ、その配列の内容を先頭から順に表示するのを、5 回繰り返すプログラムを作ってください。

繰り返しを中断する

問題によっては、繰り返しを途中で中断したい場合があります。
次のプログラムは、九九のプログラム（二重ループ）の中に、選択の制御を組み込んで、掛け算の結果が 25 を超えたら繰り返しを止めるようにしています。

```
let kotae = 0;
for (let n = 1; n < 10; n++){
    document.write(n + " の段 :");
    for (let i = 1; i < 10; i++){
        kotae = n * i;
        document.write(kotae + " ");
        if (kotae > 25){
            break;
        }
    }
    document.write("<br>");
}
```

1 行目で、掛け算の答えを記録するための変数 kotae を用意しています。
2 行目から、for キーワードを二重に使い、九九の計算を行っています。計算の答えを変数 kotae に代入し、表示します。次に if キーワードを使い、kotae の値と 25 を比べ、25 より大きければ break という処理をしています。繰り返しの中で break キーワードを使うと、繰り返しを中断することができます。

実行結果は次の通りです。

```
1 の段 :1 2 3 4 5 6 7 8 9
2 の段 :2 4 6 8 10 12 14 16 18
3 の段 :3 6 9 12 15 18 21 24 27
4 の段 :4 8 12 16 20 24 28
5 の段 :5 10 15 20 25 30
6 の段 :6 12 18 24 30
7 の段 :7 14 21 28
8 の段 :8 16 24 32
9 の段 :9 18 27
```

それぞれの段は、計算結果が 25 を超えると終了し、次の段に進みます。

例題：3　if キーワードのブロックの書き方を変えると、プログラムの動作が変わります。ブロック内の処理の一部（「document.write(kotae + " ")」）を内側の繰り返しの外に出して、実行結果を確認し、どんな動作をしたか説明してください。

```
let kotae = 0;
for (let n = 1; n < 10; n++){
    document.write(n + " の段 :");
    for (let i = 1; i < 10; i++){
        kotae = n * i;
        if (kotae > 25){
            break;
        }
    }
    document.write(kotae + " ");
    document.write("<br>");
}
```

アルゴリズムを考えて、問題を解くプログラムを作る

これまでに学んだプログラムの制御（順次、選択、繰り返し）や、データの構造（変数、配列）を組み合わせると、複雑な問題を解くプログラムを作ることができます。

与えられた問題を解くために、単純な計算・操作を組み合わせて、有限の（= 無限でない）手続きにまとめたものをアルゴリズムといいます。

逆に言えば、プログラミング言語のさまざまな機能を学ぶのは、自分が考えたアルゴリズムを、実際に動作するプログラムとして実現するためです。

補足資料

繰り返しと反復の中断・継続

キーワード	説明
break	for、while の繰り返し処理を中断する。 キーワード for、while を使って、二重以上の繰り返しをしている場合、break が書かれているレベルの繰り返しを中断する。 外側にさらに繰り返しがある場合は、そのレベルの繰り返しは継続する。
continue	for、while の繰り返し処理の中に continue キーワードがあると、繰り返し処理の中のそれ以降の処理を飛ばす。次の繰り返しの回に進む。 例：次のプログラムは、1、2 を表示した後、3 を飛ばし、4、5 と表示する。 `for (let i = 1; i < 6; i++){` `  if (i == 3){` `  continue;` `  } else {` `  document.write(i + " ");` `  }` `}`

問題集

問 1.　繰り返しと選択の構造を組み合わせて、次のようなプログラムを作成してください。

① あいさつの言葉を三つ含む配列を作る。
（ヒント）let aisatsu = [" おはよう "," こんにちは "," こんばんは "];

② 配列 aisatsu の中から、偶数番目の要素を取り出し、画面に表示する。

問 2.　繰り返しの構造を二重にして、九九を後ろから表示するプログラムを作成してください。

```
（表示順序）
九の段：81 72 63 ... 18 9
...
一の段：9 8 7 6 ... 2 1
```

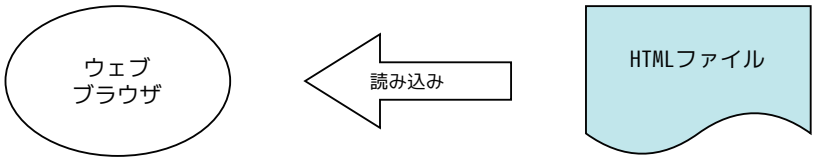
問 3.　九九のプログラムの中の、引数で指定された段の計算を表示する関数を作り、その関数を（引数で渡す値を変えながら）９回呼び出すことで九九の表示をするプログラムを作成してください。

付録： DOM の操作

JavaScript を使って、HTML を書き換え、ウェブブラウザが表示する内容を変更することができます。

ウェブブラウザの動作 HTML と DOM

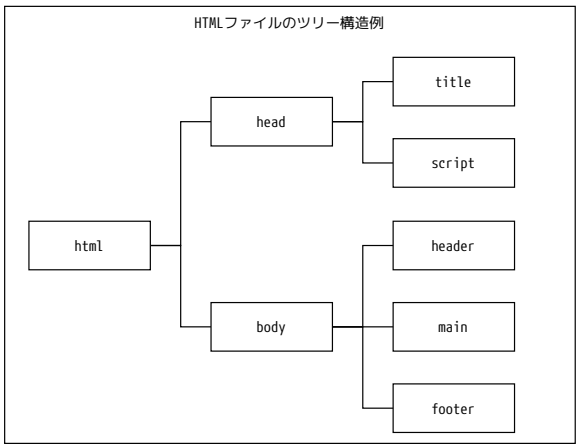
ウェブブラウザは、HTML ファイルの文書の構造を読み取って、内容を画面に表示します。



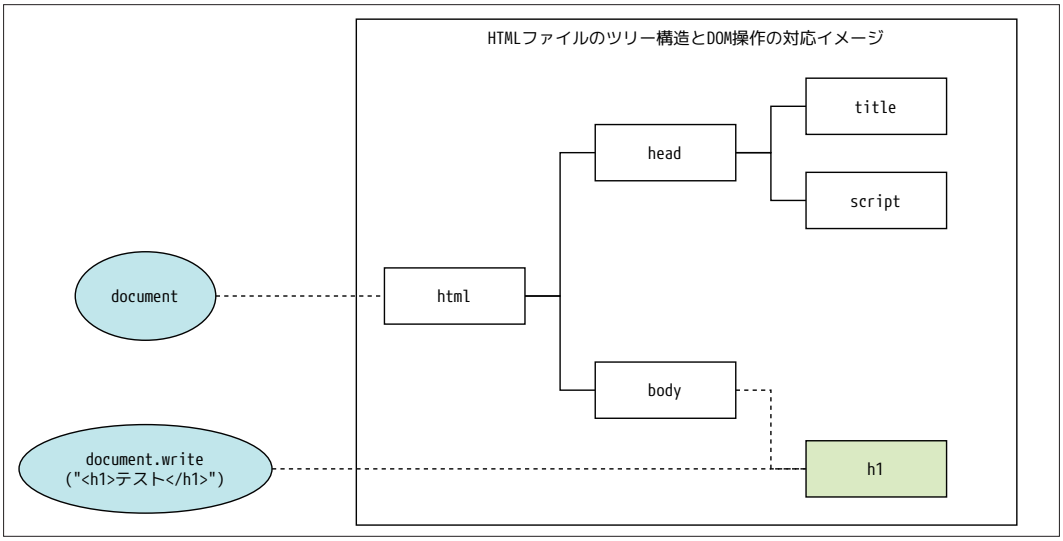
HTML はツリー構造（木構造）のデータです。

HTMLファイルの例

```
<!Doctype html>
<html>
    <head>
        <title>...</title>
        <script>...</script>
    </head>

    <body>
        <header>....</header>
        <main>
            ....
        </main>
        <footer></footer>
    </body>
</html>
```



JavaScript を使って、HTML ファイルの文書構造を操作（追加、変更、削除）し、ウェブブラウザで表示している内容を変更することができます。
JavaScript のようなプログラム言語から、文書を操作するための手段が、DOM（Document Object Model：ドム）です。



document.write() は、引数として渡した文字列を、ウェブブラウザの画面に表示する関数として紹介しました。
「document」は、文書のツリー構造の根（root：ルート）を指しています。
document.write() は、指定した文字列を、文書の構造に付け足す操作だといえます。

```
document.write("<h1> テスト </h1>");
```

ユーザーからの入力を受け付ける HTML タグ

HTML を使って、ユーザーからの入力を受け取ることができます。

```
<input id="nyuuryoku" type="text">
```

タグ <input> は、ユーザーからの入力を受け付けるためのタグです。属性 type を使い、どの種類の入力欄を表示するか決めます。上の例は、タイプ "text" を指定しているので、文字列を入力する欄が表示されます。

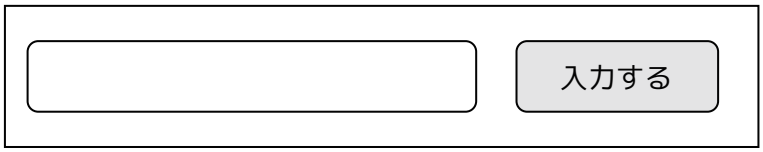
ユーザーが「ボタンを押す」のも、入力の一種です。HTML では、タグ <button> を使って実現できます。

```
<button onclick="syori()"> 入力する </button>
```

上の例では、「入力する」というボタンを表示します。
HTML ファイルの <body> タグの中に、次のように <input> と <button> を書いてみます。

```
<body>
    <input id="nyuuryoku" type="text"></input>
    <button onclick="syori()"> 入力する </button>
</body>
```

ウェブブラウザに、次のように表示されます。



入力や操作をプログラムで受け取る

HTML のタグを書くだけでは、画面に入力欄やボタンを表示することしか出来ません。「ユーザーがボタンを押したら反応して、処理を行う」「ユーザーの入力をプログラムで扱う」ためには、JavaScript のプログラムが必要です。HTML を通じてユーザーの入力・操作を受け付け、その入力内容を JavaScript で扱ってみましょう。

まず、HTML のタグ <body> の中に、次のようにタグを書きます。

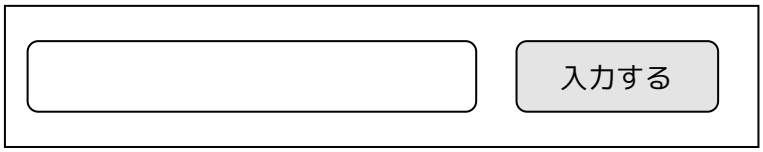
```
<body>
    <input id="nyuuryoku" type="text"></input>
    <button onclick="syori()"> 入力する </button>
    <div id="syutsuryoku"></div>
</body>
```

タグ <input> の中の id="nyuuryoku" に注目してください。HTML 文書の中のタグ <input> の部分に、HTML 文書の他の要素と区別するための ID を付けています。これで、この <input> タグ要素は、プログラムの中から "nyuuryoku"（※入力）という ID で見つけることができるようになりました。
また、タグ <button> の中に onclick="syori()" とあることに注目してください。これで「ボタンがクリックされたときに（on click）、関数 syori() を呼び出す」と設定したことになります。
最後のタグ は、HTML 文書の中に段落を作るタグです。ただし、開始タグ（ ）と終了タグ( )の間に何も書いていないので、最初は何も表示しません。こちらにも id 属性があり、"syutsuryoku"（※出力）という ID が割り当てられています。

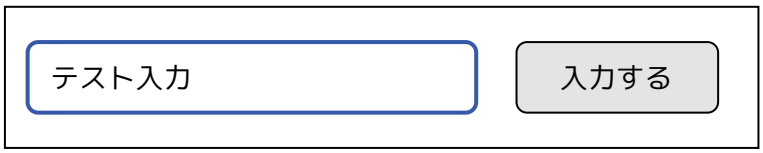
HTML のタグ <script> の中に、次の JavaScript プログラムを用意します。これは関数です（※関数については、第 5 章 関数の定義と利用で説明しています）。

```
function syori(){
    let nyuuryoku = document.getElementById("nyuuryoku").value;
    let syutsuryoku = document.getElementById("syutsuryoku");
    syutsuryoku.innerHTML = nyuuryoku;
}
```

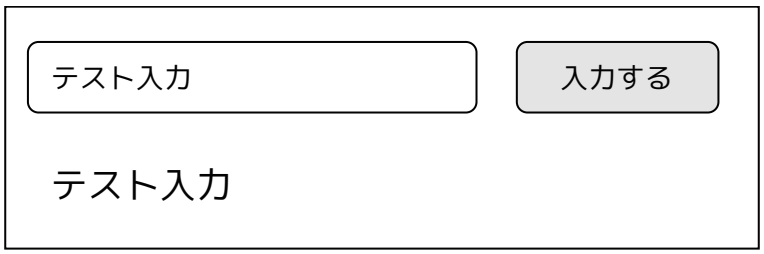
まず、動作を確認してみましょう。



表示されたら、入力欄に何か文字列を入力します。下の例では、「テスト入力」と入力しました。



続けて、ボタン「入力する」を押してみます。



入力欄の下に「テスト入力」と表示されました。ユーザーが入力したのと同じ値が、そのまま表示されるプログラムでした。

入力を扱うプログラムの説明

ユーザーがボタン「入力する」を押すと、関数 syori() が呼び出されます。

```
function syori(){
    let nyuuryoku = document.getElementById("nyuuryoku").value;
    let syutsuryoku = document.getElementById("syutsuryoku");
    syutsuryoku.innerHTML = nyuuryoku;
}
```

関数の中身の最初の document.getElementById("nyuuryoku") という行により、document(HTML の文書）の中の ID が "nyuuryoku" のもの（つまり、HTML の中でタグ <input> を使って宣言した要素）を取得できます。
要素 "nyuuryoku" を見つける処理に続けて、ユーザーが入力した値を取り出すための .value という指定が続いています。左辺の変数 nyuuryoku に、ユーザーが入力した文字列の値が代入されます（※上の実行例でいえば、「テスト入力」です）。
次の行の「let syutsuryoku = document.getElementById("syutsuryoku");」でも、document.getElementById() が使われています。今度は、"syutsuryoku" という ID の要素を指定しています。これはタグ のことです。
.innerHTML = nyuuryoku という代入により、 の内側に値が代入されます。
上の実行例でいえば、

```
<div> テスト入力 </div>
```

というように HTML を編集したことになります。

以上で、document.getElementById(ID) で HTML 文書の要素を見つけて、その要素の値を取得する方法と、取得した要素に値を設定する方法を見ました。

JavaScript のプログラムから、DOM という方法を使って、HTML 文書を書き換える操作の基本を学んだことになります。
DOM には他にも多くの機能があります。
DOM の豊富な機能を利用することで、HTML、CSS、JavaScript を組み合わせた便利で見栄えの良いアプリケーションを作ることができます。

本書の教材サポートページは以下の URL からアクセスできます。

https://edu.monaca.io/template/

JavaScript で学ぶプログラミング入門

2022 年 4 月 1 日 発行
2023 年 4 月 1 日 初版第 3 刷発行

著者　アシアル株式会社（アシアル情報教育研究所）
協力　株式会社 IMAKE
発行　アシアル株式会社
〒113-0034
東京都文京区湯島２丁目３１－１４
ファーストジェネシスビル
https://edu.monaca.io/