

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）	18
第3章	リスト	32
第4章	繰り返し（反復）	40
第5章	関数の定義と利用	52
第6章	順次実行と変数	62

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。アカウントは誰でも無料で作成できます。また、学校が用意したアカウントで利用する場合があります。先生の指示に従ってアカウントを準備して下さい。

Monaca Education アカウントの作成

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「アカウント作成」をクリックして下さい。アカウント作成フォームが表示されますので、先生の指示に従ってアカウントを作成します。特に、Google アカウントや Microsoft アカウントと連携してアカウントを作る場合は注意して下さい。連携の場合、メールアドレスの入力は不要です。

The image shows the 'アカウント作成' (Account Creation) form on the Monaca Education website. The form has a title 'アカウント作成' at the top. Below the title, there are two input fields: 'メールアドレス' (Email Address) and 'パスワード' (Password). Both fields are marked as required with a red asterisk and the text '【必須】'. The password field has a hint: '半角英字と数字を組み合わせた7文字以上' (7 or more characters combining lowercase letters and numbers). Below the input fields, there is a link to the '利用規約' (Terms of Service). At the bottom of the form, there are three buttons: a green button labeled 'アカウント新規作成' (Create New Account), a blue button labeled 'Googleアカウントで作成' (Create with Google Account), and a black button labeled 'Microsoftアカウントで作成' (Create with Microsoft Account). Below these buttons, there is a small text link: '既にアカウントをお持ちですか?' (Do you already have an account?).

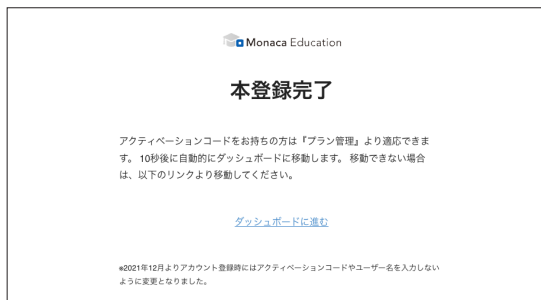
仮登録から本登録に進む（メール登録の場合のみ）

メールアドレスで登録した場合、アドレス確認のために仮登録扱いとなります。メールボックスに仮登録完了メールが届いたら案内に従って本登録を進めて下さい。

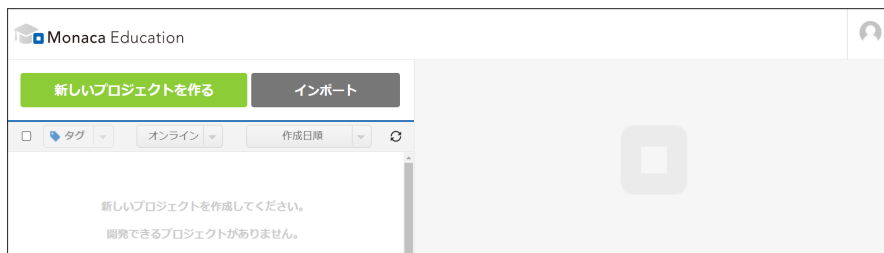
なお、Google アカウントや Microsoft アカウントで登録する際には仮登録のステップは無く、本登録が完了します。



メールに記載された URL にアクセスすることで登録が完了します。



登録を完了させるとダッシュボードが表示されます。

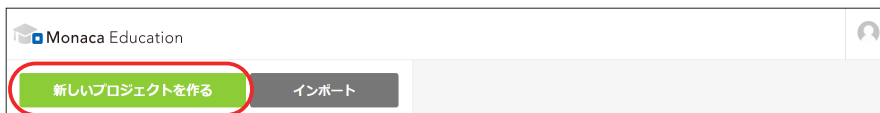


ダッシュボード上では制作中の作品を『プロジェクト』という単位で管理します。以上で Monaca Education のアカウント作成は完了です。

プロジェクトを動かす

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。なお、プロジェクトの作成方法は複数存在します。1 章以降では、印刷教材用に別途用意されたテンプレート集からプロジェクトを作成します。

プロジェクトを作成する



今回はテンプレートの中から『ブロック崩し』を選択して作成します。



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。



なお、『クラウド IDE で開く』のボタンにはオプションがあります。重要なオプションとして、作成したプログラムが無限ループなどで開けなくなった場合に利用できる『セーフモード』があります。もし必要になったときは、使ってみて下さい。オプションは逆三角形のボタンを押下することで表示できます。



Monaca クラウド IDE でプロジェクトを改造する

IDE にはアプリ制作に必要な様々な機能が用意されています。例えばプログラムのソースコードを編集したり画像などの素材を管理したりする仕組みが利用できます。今回はプログラムの一部を改造してブロック崩しの玉の数を増やします。

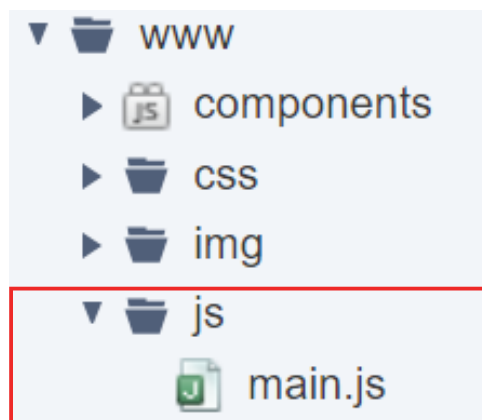
各部の名称

各部の名称は下図の通りです。



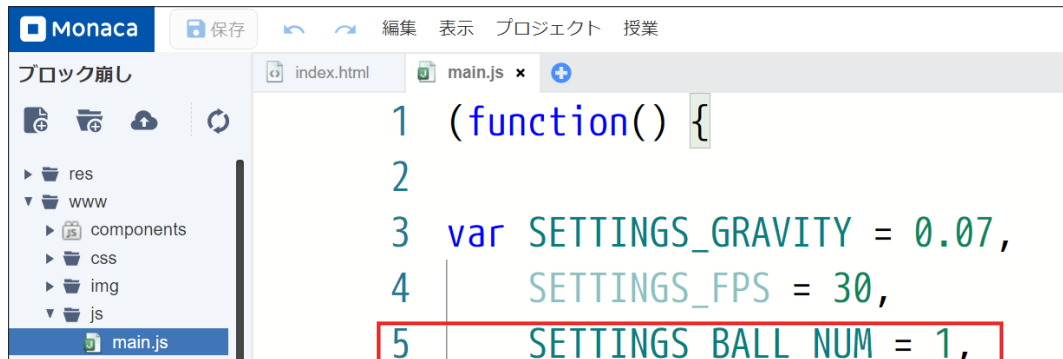
プロジェクトパネルから js/main.js を開く

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。



エディタで main.js を編集して保存する

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 個に増やしてみたいと思います。



【変更後】

```
SETTINGS_BALL_NUM = 100,
```

変更したら保存も忘れずに行ってください。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。

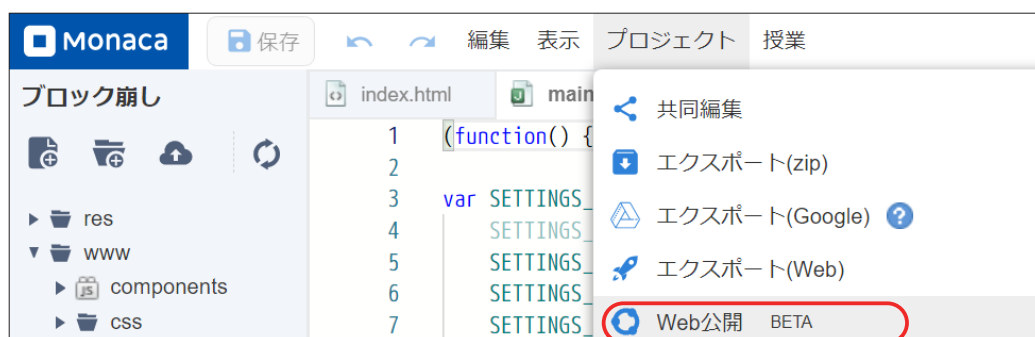


作品を Web に公開する

Monaca Education には作品を Web に公開する仕組みがあります。公開された作品は自身のスマートフォンで動作を確認できます。また、作品の URL を先生に提出するといった利用方法も考えられます。

Web 公開

メニューの『プロジェクト』から Web 公開を選択して下さい。



選択すると公開の「On / Off」が選択できます。公開を On にして右下のボタンを押下することで作品を公開できます。公開中の作品はログインしていなくても公開用の URL にアクセスすることで閲覧できます。また、QR コードも表示されるため、スマートフォンやタブレットから読み取って動かすこともできます。

(本機能の紹介は BETA 版の仕様に基づいています、2022 年度以降、一部の機能が変更される可能性があります。)



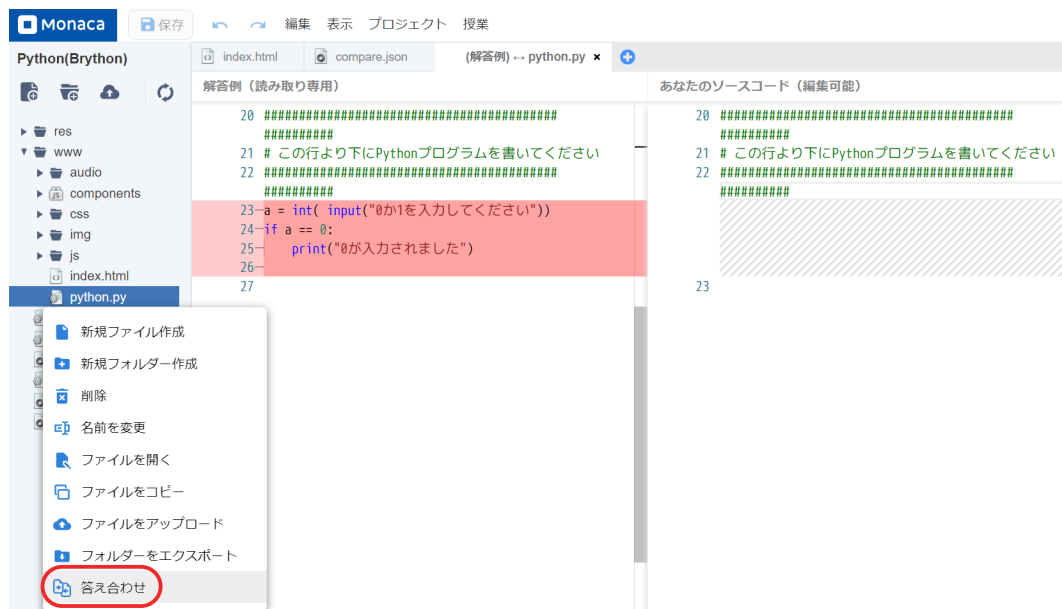
教材サポートページを活用する

教材サポートページにアクセスすることで、印刷教材に従って学習する上で役に立つコンテンツを入手できます。例として、各章で学ぶ実習のひな形となるサンプルプロジェクトが掲載されています。

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

答え合わせ機能

模範解答を確認しながら実習を行える機能です。あらかじめ模範解答が設定されたサンプルプロジェクトでプロジェクトパネルのファイルのメニューを開き、「答え合わせ」を選択することで表示できます。



詳しい利用方法は、教材サポートページを参照してください。

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。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。

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。

```
print(" こんにちは ")  
print(" 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。

「print」は、直後にある（ ）の中に指定した文字の並び（文字列）を、画面に表示させます。プログラム言語 Python は初めからたくさんの命令を備えています。「print」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 print() を2回使っています。先に書かれている「print(" こんにちは ")」が実行された後に、後ろに書かれている「print(" 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。

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
```

補足1: print() 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「print()」と「PRINT()」は別のものとして扱われます。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。

```
a = "Hello"
print(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、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。

```
a = "Hello"
a = " こんにちは "
print(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:Pythonは記号「=」の前後の空白を無視します。前後の空白が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。

```
b = 10
print(b)
c = b + 10
print(c)
print("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` に「10」という数値を代入しています。次の 2 行目で関数 `print()` 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` に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「`print(c)`」で、変数 `c` に代入された値を表示しています。

5 行目の「`print()`」では、表示させたい値をカンマ（ , ）で区切って、1 度に複数の値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。

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。

```
a = input(" 名前は?")  
print(" 名前 :", a)  
b = input(" 年齢は?")  
b = int(b)  
print(" 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）。

console.monaca.education の内容

名前は?

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。

console.monaca.education の内容

名前は?

続けて、年齢を入力する画面が表示されるので、（※半角文字入力に変更してから）数字を入力して、OK ボタンを押します。

console.monaca.education の内容

年齢は?

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。

```
a = input(" 名前は?")
print(" 名前 :", a)
b = input(" 年齢は?")
b = int(b)
print(" 年齢 :", b)
```

`input()` 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る関数です。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「`input()`」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。

さらに、新しく出てきた `int()` という命令に `b` が渡されています。この関数 `int()` は、`()` 内の文字列を数値に変換しています（上の例では、「(文字列である) 16」を「(数値である) 16」に変えています）。

プログラムは、ユーザーが入力した値を、いったん文字列として扱います。文字列では足し算や掛け算のような計算はできないので、数値に変換するために関数 `int()` を使います。

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
print()	<p>カッコの中に入れられた値を、画面に表示する。</p> <p>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)。</p> <p>複数の値をカンマで区切って、カッコの中を書くこともできる。その場合、書いた順に表示される。</p>
input()	<p>ユーザーに値を入力するよう求める。</p> <p>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)。</p>

値の型を変換する関数

関数名	動作
int()	<p>int("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。</p> <p>a = int("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。</p> <p>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。</p>
float()	<p>float("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、数値型（浮動小数点数）に変換する。</p> <p>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。</p>

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。 ※文字列と数値の掛け算を行うと、文字列を繰り返すことができる。
/	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. `print()` 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 `nae` 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. `input()` と `int()`、`print()` 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。

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。

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。

```
a = int( input("0 か 1 を入力してください "))
if a == 0:
    print("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？

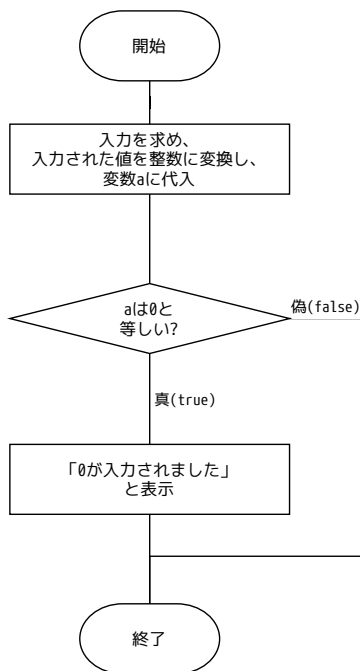
```
0 が入力されました
```

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。

先ほど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：「input()」で入力を受け取り、「int()」で文字列の値を数値に変換しています。

```
a = int( input("0 か 1 を入力してください"))
```

プログラムでは、関数 int() のカッコの内側に、関数 input() 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ばれる場合があります。

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。

比較演算子とインデント（字下げ）

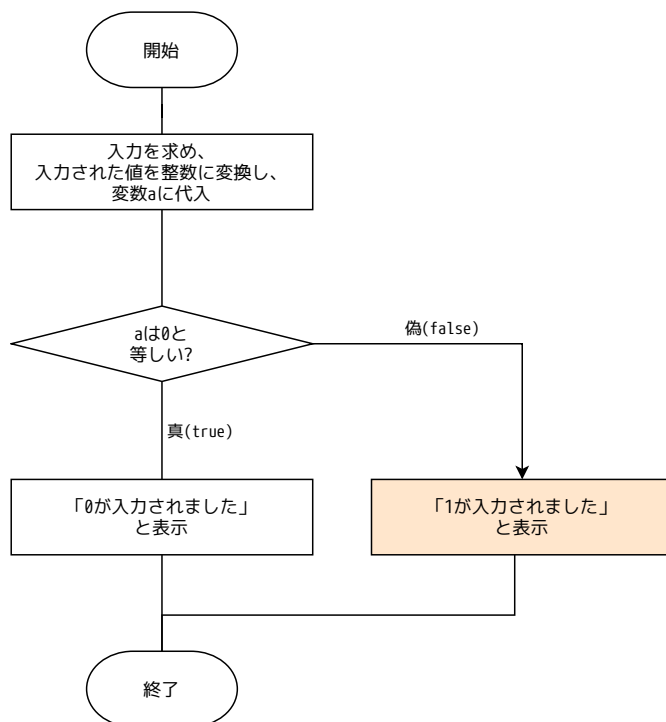
```
a = int( input("0 か 1 を入力してください"))
if a == 0:
    print("0 が入力されました ")
```

2 行目にある if キーワードの後ろの「==」のように、2 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。計算の後に、記号「:」（コロン）が付けてあることに注意してください。記号コロンの後、改行しています。続く 3 行目行の print() 文の前に空白があり、if 文よりも後ろから行が始まっています。これ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。Python ではインデント（字下げ）が同じだけ取られている行が並んでいるとき、ひとかたまりのプログラム（**ブロック**）として扱います。

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。

Python 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。

```
a = int( input("0 か 1 を入力してください"))
if a == 0:
    print("0 が入力されました ")
else:
    print("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

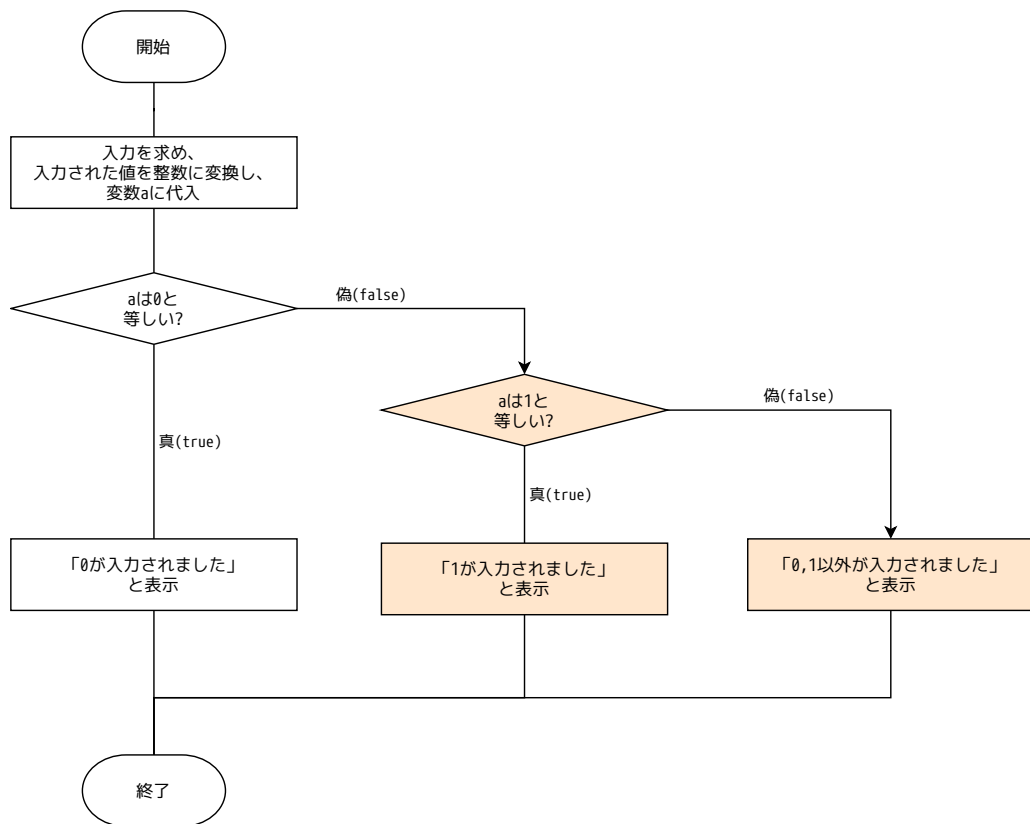
前のプログラムで、「2」を入力すると、結果はどうなるでしょうか？

1が入力されました

この結果は、if文の構造が原因です。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行目の「`elif a == 1:`」という記述に注目してください。

```
a = int( input("0 か 1 を入力してください"))
if a == 0:
    print("0 が入力されました ")
elif a == 1:
    print("1 が入力されました ")
else:
    print("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。

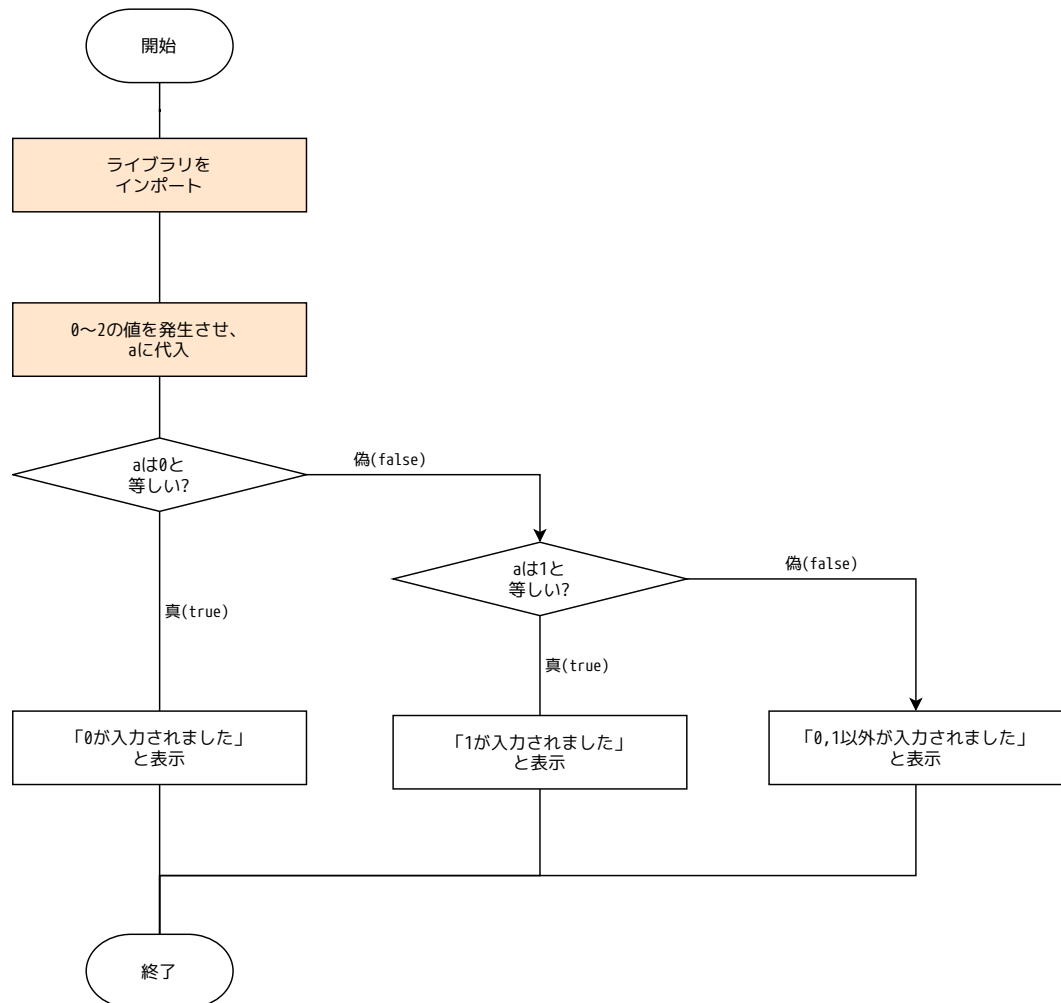
もし「0」以外が入力されたなら、4行目の「`elif a == 1:`」が実行・評価されます。「1」が入力されていたら、5行目が実行され、「1が入力されました」と表示します。さらに、「1」以外が入力されたなら、7行目が実行され、「0,1 以外が入力されました」と表示することになります。`if` の式が先で、`elif` の式が後に評価されることを確認しましょう。

なお、この例では `elif` キーワードを1つだけ使っていますが、`if` キーワードの後ろに `elif` 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`, `elif` のキーワードが評価されます。`else` キーワードは、どの `if`, `elif` 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、1つ1つの数字が現れる確率が等しいもの）を、**乱数**といいます。Python を使って、擬似的に乱数を発生させることができます。

フローチャートは次の通りです。



プログラムは次の通りです。

```
import random
a = random.randint(0, 2)
if a == 0:
    print("0が入力されました ")
elif a == 1:
    print("1が入力されました ")
else:
    print("0,1以外が入力されました ")
```

プログラムの先頭にある1行目の「import」（インポート）は、Pythonのキーワードで、続いて書かれている「random」（ランダム）という機能をプログラムに読み込んでいます。

2行目の「=」の右辺にある「random.randint(0, 2)」は、「0」から「2」の間の整数、つまり0、1、2のいずれかの値をランダムに作り出します。作り出された値は変数aに代入されます。

以後の処理は、ここまでのプログラムの例と同じif-elif-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

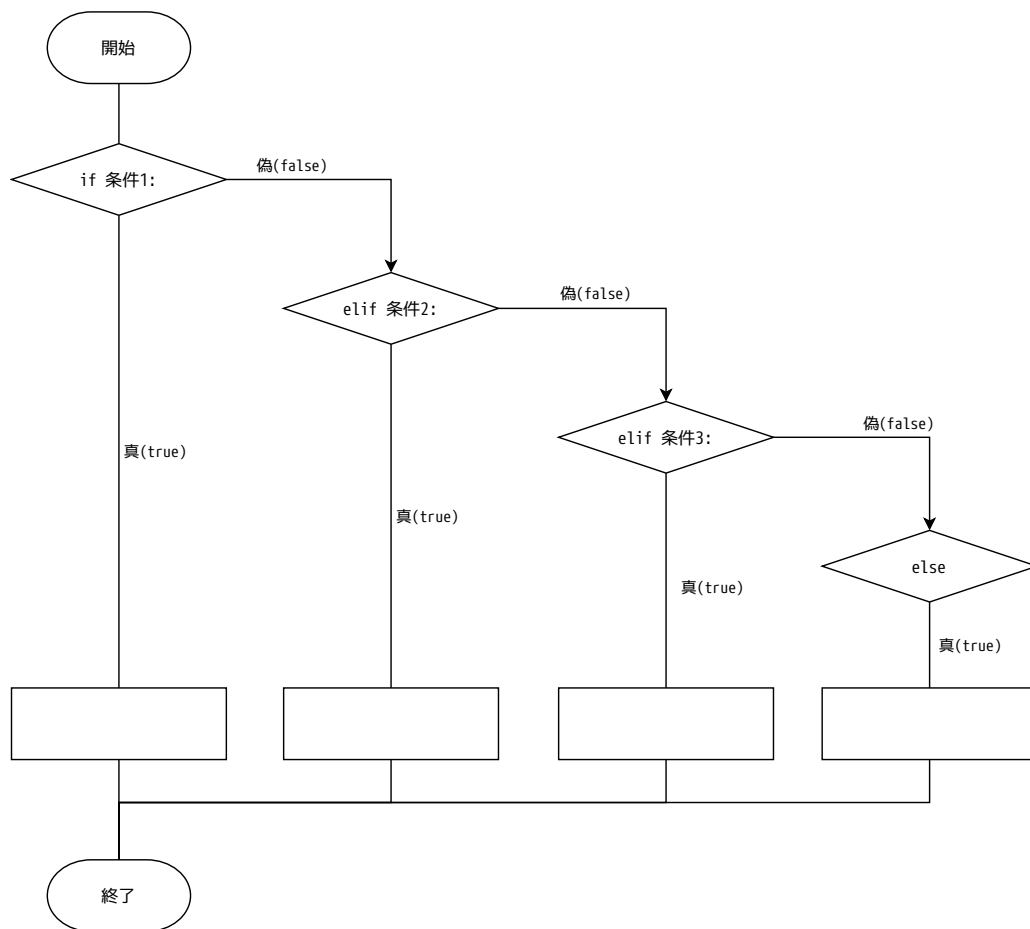
関数randomをインポートし、関数random.randint()を使って、サイコロのように1から6の間の数字をランダムに表示するプログラムを作成してください。

（ヒント）「random.randint(a, b)」と記述すると、a以上b以下の整数が返される。

補足資料

if-elif-else

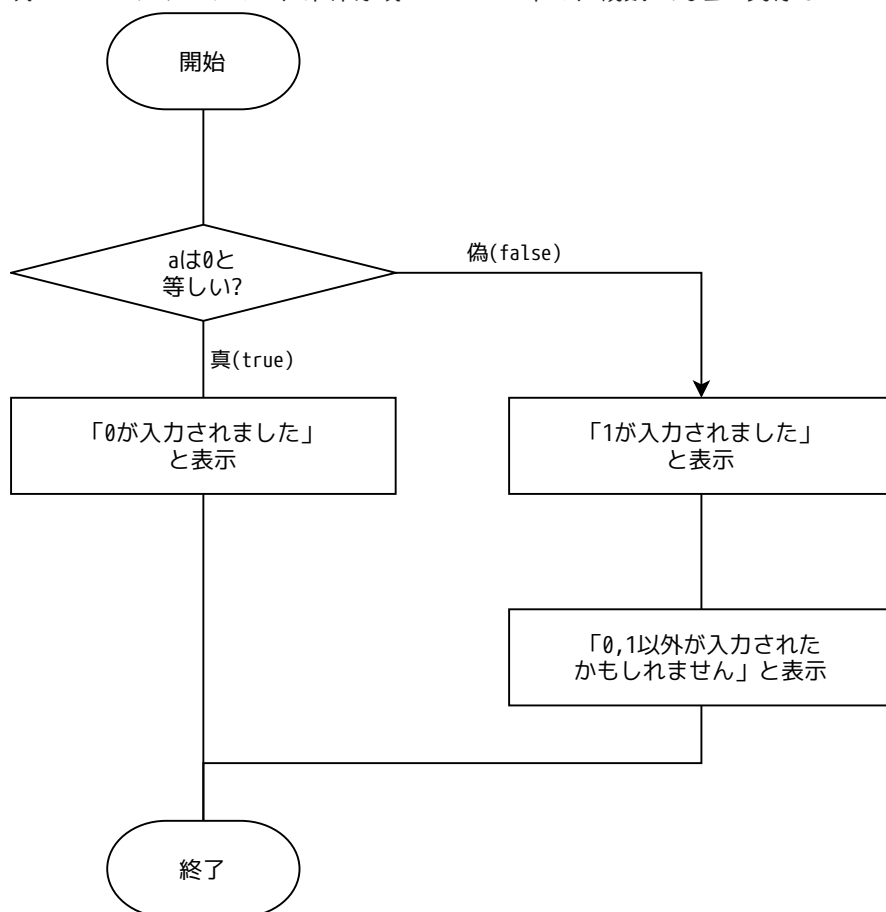
if キーワードだけ用いて、ある条件が真になるときだけ処理を実行させることができます。else を使うと、if の条件が偽のときに実行する処理を指定することができます。if の後に elif を使い、別の条件による判定を行い、さらに分岐させることができます。elif はいくつでも付け足すことができます。



if-elif-elif-elif ... の最後に else を付けて、if, elif 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。

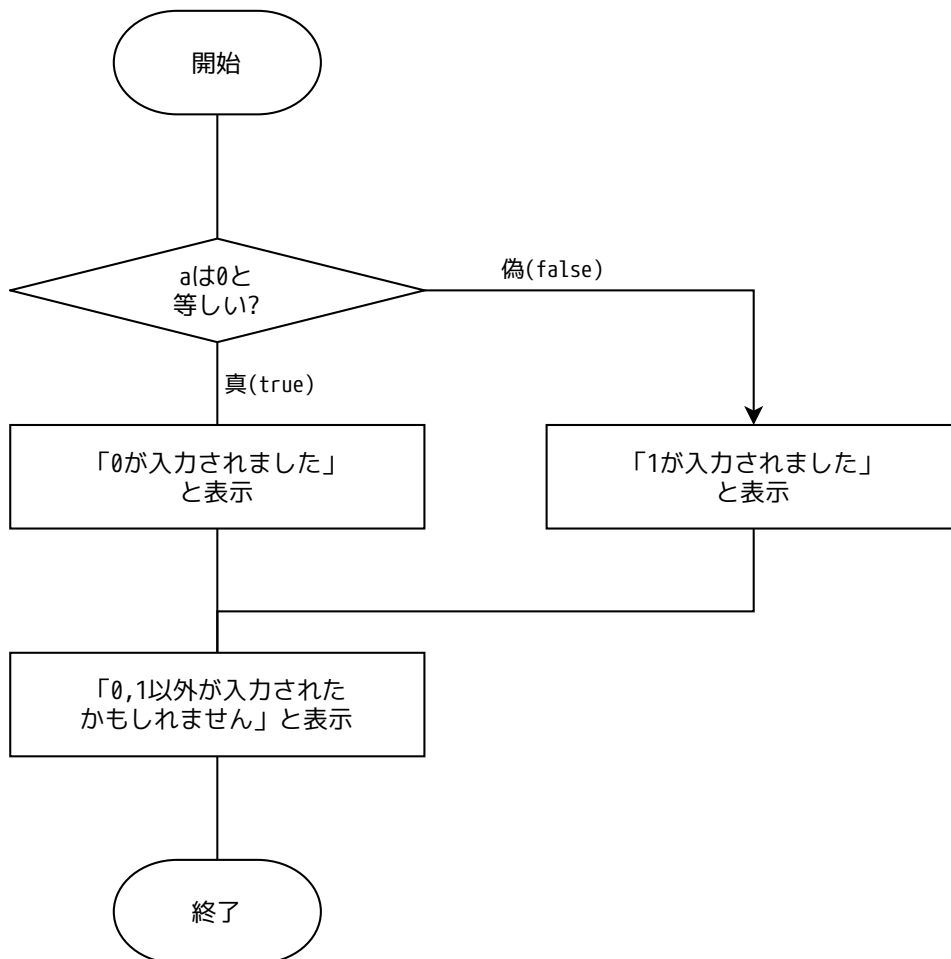
```
if a == 0:
    print("0が入力されました")
else:
    print("1が入力されました")
    print("0,1以外が入力されたかもしれません")
```

`else:` の後ろにある2つの `print` 文の、インデント（字下げ）がそろえられています。インデントがそろっている文のかたまりを、ブロックと呼びます。if や `elif`、`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。Python では、インデントによってブロックを作るので、インデントの深さ（空白文字の数）はとても重要です。

2 つ目の print 文のインデントを取り除くと、動作が変わります。

```
if a == 0:
    print("0 が入力されました ")
else:
    print("1 が入力されました ")
print("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
<code>==</code>	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。
<code>!=</code>	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。 <code>==</code> の反対の結果になる。
<code>></code>	<code>a > b</code> で、 <code>a</code> が <code>b</code> より大きければ真、そうでなければ偽を返す。
<code>>=</code>	<code>a >= b</code> で、 <code>a</code> が <code>b</code> 以上なら真、そうでなければ偽を返す。
<code><</code>	<code>a < b</code> で、 <code>a</code> が <code>b</code> より小さければ真、そうでなければ偽を返す。
<code><=</code>	<code>a <= b</code> で、 <code>a</code> が <code>b</code> 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
<code>random.randint</code>	<code>a</code> 以上 <code>b</code> 以下のランダムな整数を返す。 たとえば、 <code>random.randint(1, 6)</code> なら 1,2,3,4,5,6 のいずれかがランダムに返される。
<code>random.random()</code>	0 以上 1 以下の範囲のランダムな浮動小数点数が返される。

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 `random` をインポートする。
- ② 1 から 6 の整数をランダムで生成し、変数 a に代入する。
- ③ 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）

- ・ `random.randint(1, 3)` は、1 以上 3 以下の整数を返す。
- ・ サイコロは、1 から 6 の目がある。

第3章 リスト

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。

リストという仕組みを使うと、1つのリスト変数に、複数の値を格納することができます。リストは、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。先にリストを見ておくことにします。

1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。

```
youbiList = ["月", "火", "水", "木", "金", "土", "日"]

from datetime import date
youbi = date(2021, 11, 1).weekday()

print(youbi)
print( youbiList[youbi] )
```

「0」「月」と表示されます。

プログラムの3行目では、Pythonの中で日付を扱うための準備をしています。

4行目では、日付をあらわす年、月、日の3つの値を渡した上で、weekday()という関数を呼び出しています。

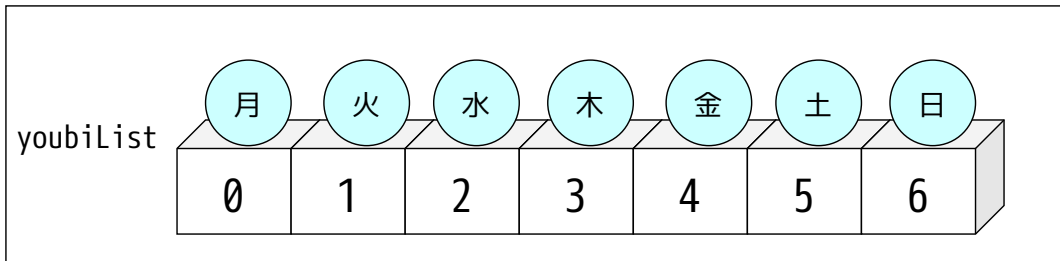
```
from datetime import date
youbi = date(2021, 11, 1).weekday()
```

この関数 weekday() は、指定された日付の曜日を返します。ただし、「月」、「火」、のような曜日を表す文字ではなく、「0」から「6」の値を返します。

「0」が月曜日を、「1」が火曜日を…、「6」が日曜日を意味します。

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 weekday() 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リストの変数 `youbiList` のイメージです。



リスト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、リストの変数 `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、リストの中にある要素の数から1引いた数まで（※リスト `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: リストの変数 `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。

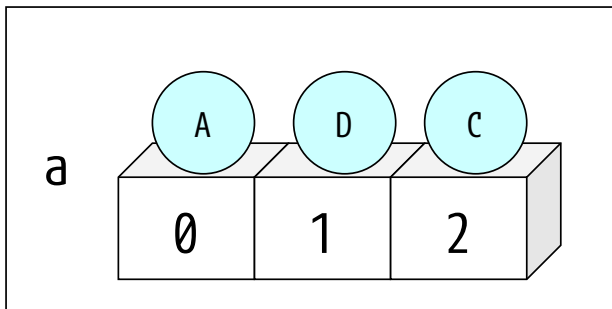
例題2: 関数 `print()` のかっこの中にリストの変数を入れると、リストの全ての要素を表示します。
プログラムを書いて確かめてみましょう。

(ヒント) `print(a)`

リストの要素を置き換える

「a[1] = "D"」のように、「リスト名[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。

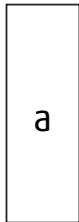
```
a = ["A", "B", "C"]  
a[1] = "D"  
print( a[1] )
```



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

空のリストを作り、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。これは空のリストを作る処理です。

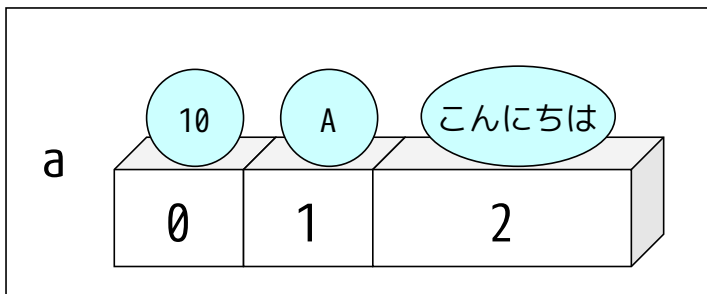


最初は要素は1つ也没有せん。

```
a = []  
a.append(10)  
a.append("A")  
a.append(" こんにちは ")  
print( a[2] )
```

次の2行目から、「a.append()」という記述が続きます。これはリストaの後ろに追加（append）しています。2行目から4行目までを実行すると、関数append()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。「リスト名[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: リストaisatsuListを空で作成した後、「おはよう」、「こんにちは」「こんばんは」と3つの要素を追加し、最後にリストaisatsuList全体を表示するプログラムを作ってください。

リストの好きな場所に要素を追加する

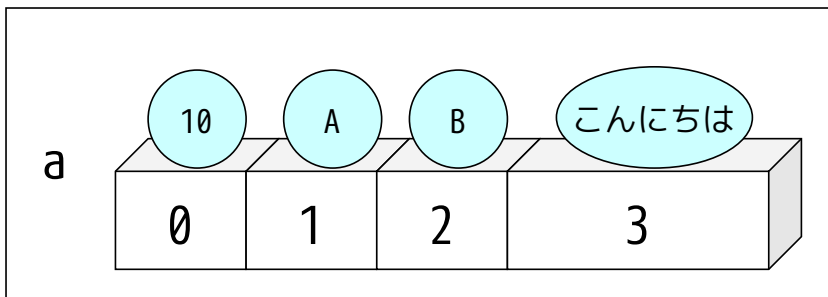
リストの好きな場所に要素を追加することができます。

```
a.insert(2, "B")
```

この「2」は、添え字ですので、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。

```
a = []  
a.append(10)  
a.append("A")  
a.append(" こんにちは ")  
a.insert(2, "B")  
print( a[2] )
```

添え字は0から始まることを思い出しましょう。



リストの要素を削除する

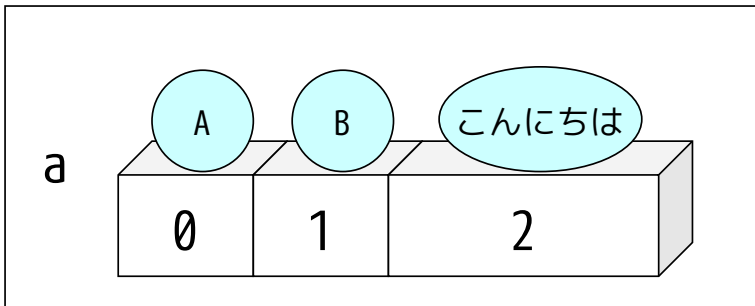
リストの要素を削除するには、関数 `pop()` を使います。

```
a.pop(0)
```

「0」は添え字です。添え字で指定した要素を削除し、削除した要素よりも後にある要素を前に詰めます。

```
a = []  
a.append(10)  
a.append("A")  
a.append(" こんにちは ")  
a.insert(2, "B")  
a.pop(0)  
print( a[2] )
```

5行目の「`a.insert(2, "B")`」の後に、6行目で「`a.pop(0)`」を実行した結果、リストの内容は次のように変わります。



例題 5: リスト `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。最後にリスト `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。 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 .append(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.append(10) -> a = [1, 2, 5, 10]
配列名 .insert(添え字 , 要素)	添え字で指定した番号の直前に、要素を追加する。添え字が 0 の場合、先頭 (0 番目の前) に追加する。 例：a が [1, 2, 5] のとき、 a.insert(0, 100) -> [100, 1, 2, 5]
配列名 .pop(添え字)	添え字で指定した配列の要素を取り出す。添え字を指定しなかった場合、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]
配列名 .remove(要素)	配列の中から指定した要素を取り除く。 例：a = [1, 2, 5] のとき、a.remove(1) -> [2, 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。

```
a = ["A", "B"]
print("a[1]:", a[1])
print()

a.append("C")
print("a[1]:", a[1])
print("a[2]:", a[2])
print()

a.insert(2, "b")
print("a[1]:", a[1])
print("a[2]:", a[2])
print()

b = a.pop(2)
print("a[1]:", a[1])
print("a[2]:", a[2])
print("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 リストの変数 `b` を作る。
- ② `b` に要素として 1, 2, 3, 4, 5 を追加する（5 個の要素を含むリストになる）。
- ③ リスト `b` の要素を取り出して、順に 1, 2, 3, 4, 5 と表示する。
- ④ リスト `b` を空のリストにする。

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」でリストを扱いました。リストに格納されている複数の要素それぞれに対して、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。

決まった回数だけ繰り返し実行する

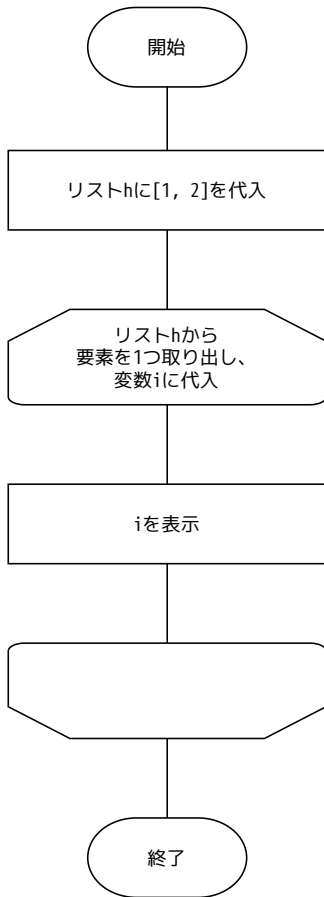
次のプログラムを実行してみましょう。なお、3行目の先頭では半角スペースを4個入れて、字下げ（インデント）をしています。

```
h = [1, 2]
for i in h:
    print(i)
```

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、リスト h に2つの要素を代入しています。結果、リスト 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 リスト h の先頭から1つ目の値を取り出して（値は「1」）、変数 i に代入する。
- ② 「print(i)」を実行する。「1」と表示される。
- ③ リスト h から2つ目の値を取り出して（値は「2」）、変数 i に代入する。
- ④ 「print(i)」を実行する。「2」と表示される。

これは、リストの要素の数だけ「変数 i に要素を代入し、print(i) を実行して、変数 i を表示する」という処理を繰り返しています。

第4章 繰り返し（反復）

なお、次のように、字下げ（インデント）を揃えることで、複数の処理を繰り返すプログラムを作ることができます。

```
h = [1, 2]
for i in h:
    print(i)
    print(" おわり ")
```

「変数 *i* の値を表示した後、『おわり』という文字列を表示する」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、実行結果はどのようなでしょうか。

（修正前）

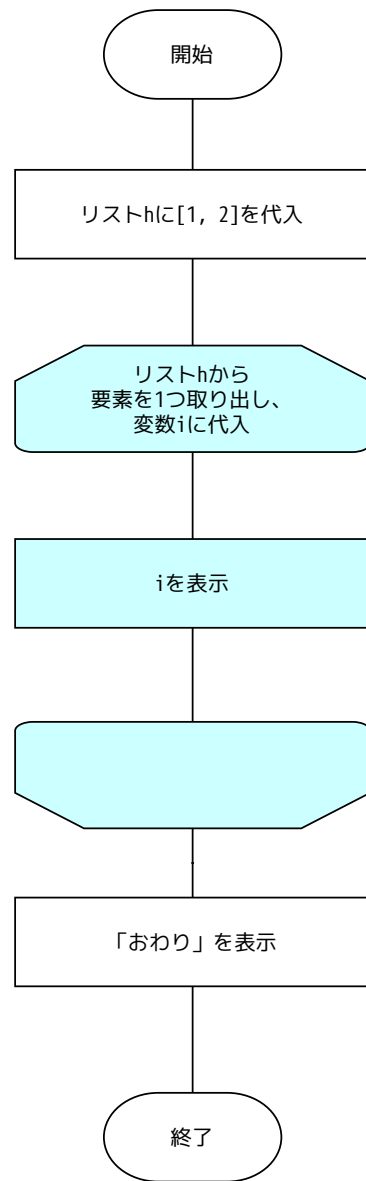
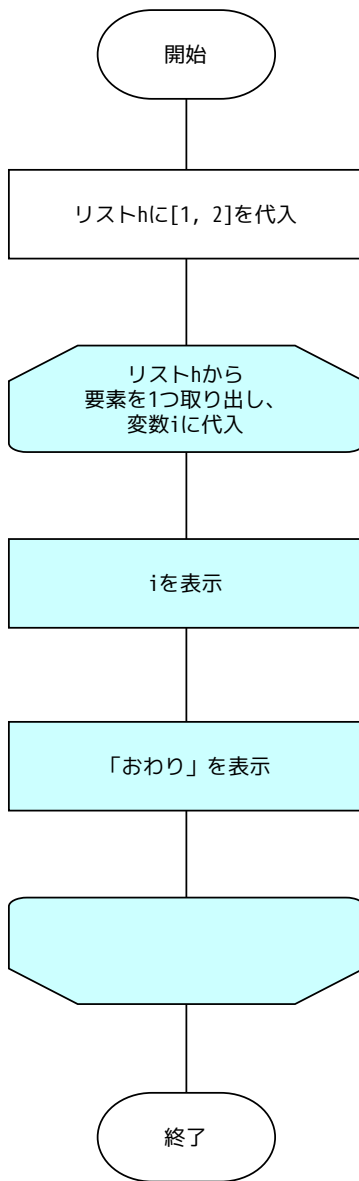
```
h = [1, 2]
for i in h:
    print(i)
    print(" おわり ")
```

（修正後）4行目の「（“おわり”）」の `print` 文の字下げ（インデント）を削除しています。

```
h = [1,2]
for i in h:
    print(i)
print(" おわり ")
```

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。



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？

もっとたくさんの回数、繰り返し実行する

「1」から「100」まで表示するプログラムを考えてみましょう。

リストを使って「`h = [1, 2, 3, 4, ..., 99, 100]`」と書くこともできますが、関数 `range()` を使うともっと短く実現できます。次のプログラムを見てみましょう。

```
h = range(1, 101)
for i in h:
    print(i)
```

実行すると、「1」から表示をはじめ、

```
1
2
3
4
5
```

「100」まで表示します。プログラムは「`range(1, 100)`」ではなく「`range(1, 101)`」であることに注意しましょう。

```
97
98
99
100
```

関数 `range()` のカッコの中に、2つの値をカンマ記号（,）で書くと、1つ目の値（開始）から、2つ目の値（終了）の1つ前の値までの「範囲」を表すデータが作られます。

```
range( 開始 , 終了 )
```

厳密には `range` 型（レンジ、範囲型）のデータと呼びますが、ここでは `for` 文の中でリストと同じように使えるものだと考えてください。

for 変数 in range()

ここまでのプログラムでは、関数 range() の結果を変数 h に受け取り、その変数 h を for 文の in の後ろに書いていました。この変数 h の利用をやめることができます。

```
for i in range(1, 101):  
    繰り返し実行する処理  
    ...
```

関数 range() は最初に 1 回だけ実行され、以後は「1」から「100」までの値を順に変数 i に渡します。

補足：関数 range() のいろいろな使い方

関数 range() にはいろいろな使い方があります。

開始、終了の値を自由に指定できるほか、3 つ目の値を指定することで「2 ずつ増やす」や「10 ずつ減らす」などの処理を実現できます。

例題 4: for 文と関数 range() を組み合わせて、「15」から「35」まで表示するプログラムを作ってみましょう。

例題 5: 関数 range() の括弧の中を次のようにすると何が起きるか、確認してみましょう。

```
range(1, 50, 2)
```

ヒント：カンマを 2 つ使い、3 つの値をカッコの中に書いています。

例題 6: 関数 range() の括弧の中を次のようにすると何が起きるか、確認してみましょう。

```
range(100, 1, -10)
```

（ヒント）開始の値が終了の値より大きいことと、3 つ目の値がマイナスの値であることに注意してください。

条件を満たすまで繰り返し実行する

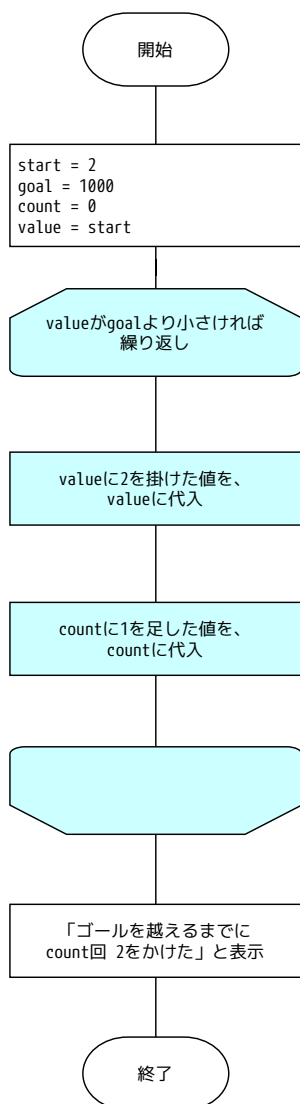
あらかじめ、何回繰り返すか想定できる場合、for 文を使うと簡単です。

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」の値からスタートして、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。



続いて、プログラムです。

```
start = 2
goal = 1000
count = 0
value = start
while value < goal:
    value = value * 2
    count = count + 1
print(" 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 をゼロに代入しています。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、2 倍になっていきます。最初の変数 start の値を代入しています。ですから変数 value の最初の値は「2」です。
- 5 行目 : while 文です。while の後ろに、式を書き、記号コロン「:」で式の終わりを示しています。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)、変数 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 の値のほうが大きければ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。
- 8 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）」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 7: `while` 文と関数 `random()` 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）

```
import random  
a = random.randint(1, 10)
```

実行結果の例 (1): 「`randint(1, 10)`」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 8: `for` 文と関数 `random()` 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
<pre>for 変数 in リスト : 処理 1 処理 2 ... 処理 A</pre>	<p>「リスト」または「範囲」から値を一つずつ取り出し、「変数」に代入する。</p> <p>処理 1、処理 2、…と、字下げ（インデント）が揃っている一連の処理（ブロック）を実行する。</p> <p>字下げが揃っている一連の処理が終わったら、リストの次の要素を取り出し、変数に代入して、また処理 1、処理 2、…を実行する。</p> <p>リストの要素すべてについて処理が終わったら、for のブロックの次の処理に進む。左の例なら、処理 A に進む。</p>

range()

構文	動作
range(終了)	<p>1 つの整数の値を渡すと、0 から、「『終了』より小さい値」の整数の範囲を返す。</p> <p>例：range(5) -> [0, 1, 2, 3, 4]</p>
range(開始 , 終了)	<p>2 つの整数を渡す。開始の値から、「『終了』より小さい値」の整数の範囲を返す。</p> <p>例：range(1, 5) -> [1, 2, 3, 4]</p>
range(開始 , 終了 , ステップ)	<p>3 つの整数を渡す。開始の値から、「『終了』より小さい値」の整数の配列を返す。</p> <p>ステップに指定した数ずつ増える。</p>

while 文

構文	説明
while 式 : 処理 1 処理 2 ... 処理 A	<p>「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、字下げ (インデント) が揃っている一連の処理 (ブロック) を実行する。</p> <p>ブロックの処理が終わったら、ふたたび while 式に戻り、「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。</p> <p>左の例であれば、処理 A に進む。</p> <p>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。</p>

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。
- ・ range() をうまく使うと if 文無しでも実現できます。

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 random をインポートする。
- ② 1 から 6 の整数をランダムで生成し、変数 a に代入する。
- ③ 変数 b に 0 を代入する。
- ④ 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)。
- ⑤ while 文が終了したら、最後に b の値を表示する。

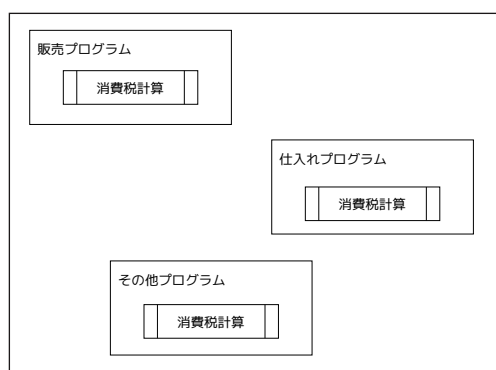
第5章 関数の定義と利用

ここまで、`input()` や `print()` などの関数を利用してきました。`input()` や `print()` は、Python 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。

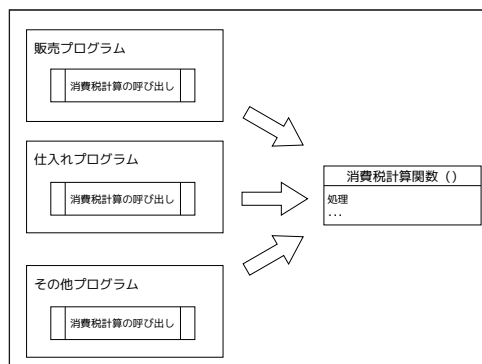


何度も使いたい計算・処理を、必要になるたびにプログラミングするのは面倒です。また、計算・処理の内容を変えるときは、プログラムに書いた個数だけ修正しなければなりません。

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。Python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くても使うことができます。

- `input()`
- `print()`
- `random.randint()`
- `range()`

これらの関数を、**組み込み関数**と呼びます。関数を使うと、ユーザーからの入力を受け付けたり（関数 `input()`）、値を出力したり（関数 `print()`）、乱数を生成したり（関数 `random.randint()`）、範囲を作ったり（関数 `range()`）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）

`input("表示するメッセージ")`, `print(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。

```
from datetime import date
hiduke = date.today() # 今日の日付を取得する
print(hiduke)
```

1行目の「from datetime import date」は、組み込み関数を使うための準備です。

2行目の「date.today()」で今日の日付を取得しています。

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。

```
from datetime import datetime
hiduke_jikoku = datetime.now() # 現在の日付・時刻を取得する
print(hiduke_jikoku)
ji = hiduke_jikoku.hour # 現在の日付・時刻から、時を取得する
print(ji)
hun = hiduke_jikoku.minute # 現在の日付・時刻から、分を取得する
print(hun)
byou = hiduke_jikoku.second # 現在の日付・時刻から、秒を取得する
print(byou)
```

1行目の「from datetime import datetime」は、組み込み関数を使うための準備です。

2行目の「datetime.now()」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）

```
from datetime import datetime
hiduke_jikoku = datetime.now()
```

年：変数 hiduke_jikoku.year, 月：変数 hiduke_jikoku.month, 日：変数 hiduke_jikoku.day

※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、Python 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、def キーワードを使います。def 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。

```
def 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。

```
def aisatsu():  
    print(" 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 :字下げ（インデント）していることに注意してください。この「print(" こんにちは ")」は、関数 aisatsu() の中に含まれています。

3 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。これを引数（ひきすう）といいます。

```
def aisatsu1(aite):  
    print(" こんにちは ", aite)  
aisatsu1("Python")
```

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。

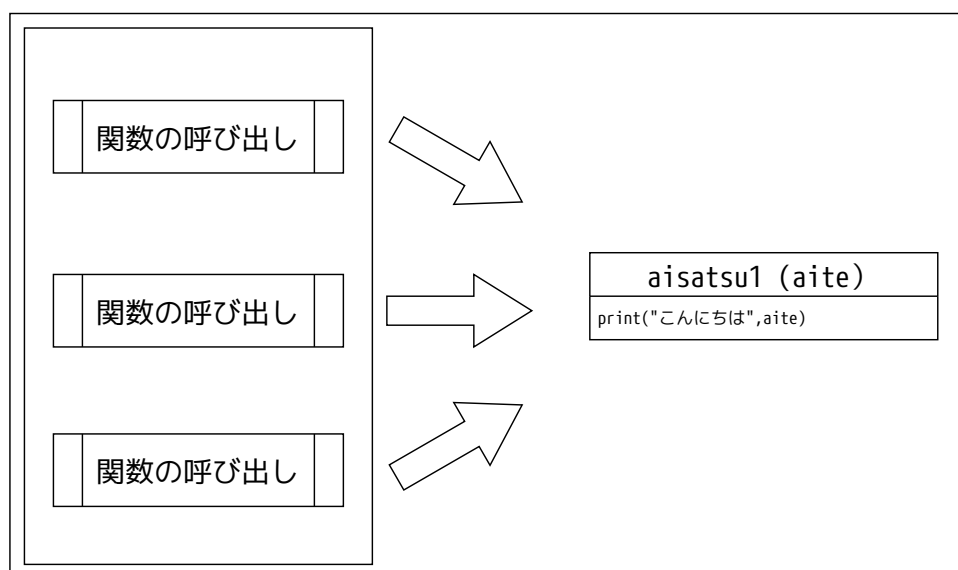
```
def aisatsu2(aite):  
    return " こんにちは " + aite  
kotoba = aisatsu2("Python")  
print( kotoba )
```

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。

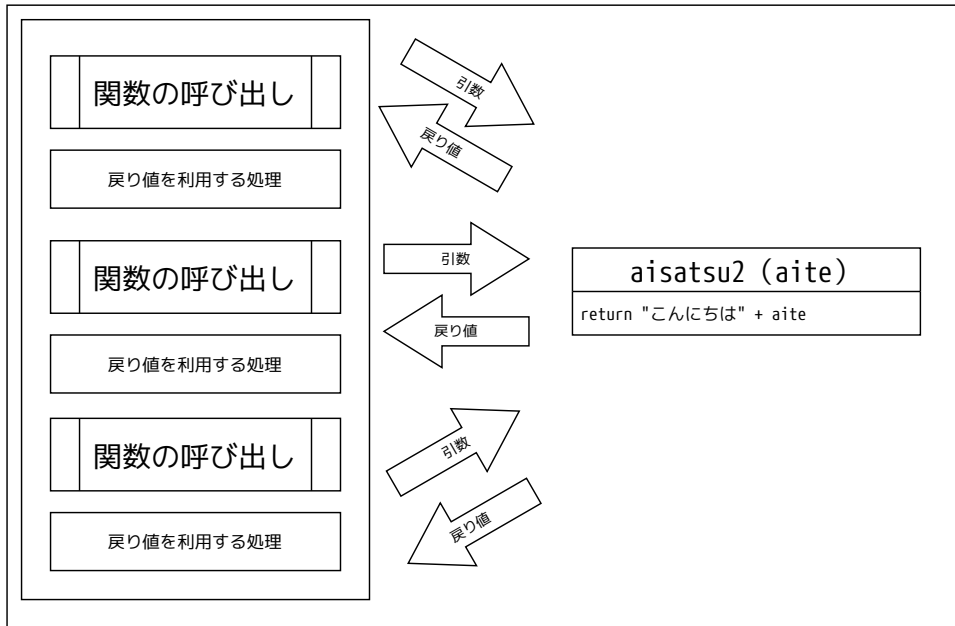
上のプログラムの関数（`aisatsu1`）は、引数で渡された値を使って画面に表示しています。

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1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

日付・時刻

組み込み関数	説明
<code>date.today()</code>	<p>使用時の宣言： <code>from datetime import date</code> <code>hiduke = date.today()</code></p> <p>プログラムを実行したその日の日付を取得する。</p>
<code>date.weekday()</code>	<p>使用時の宣言： <code>from datetime import date</code> <code>youbi = date(2021, 11, 1).weekday()</code></p> <p>指定した日付の曜日を数で返す。</p> <p>月曜日が 0、火曜日が 1、... 土曜日が 6 である。</p>
<code>datetime.now()</code>	<p>使用時の宣言： <code>from datetime import datetime</code> <code>hiduke_jikoku = datetime.now()</code></p> <p>現在の日付・時刻を取得する。</p> <p>年 (<code>hiduke_jikoku.year</code>)、月 (<code>month</code>)、日 (<code>day</code>) 時刻 (<code>hour</code>)、分 (<code>minute</code>)、秒 (<code>second</code>)</p>
<code>time.time()</code>	<p>使用時の宣言： <code>import time</code></p> <p>POSIX タイム (1970 年 1 月 1 日午前 0 時 0 分 0 秒からの経過秒数) を取得する。</p>

さまざまな組み込み関数

組み込み関数	説明
<code>random.random()</code>	使用時の宣言 : <code>import random</code> 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。
<code>len(リスト)</code>	リストの要素数を調べることができる。 <code>a = [1, 2, 3]</code> <code>print(len(a)) -> 3</code>
<code>int()</code>	引数に渡した文字列から、整数のデータを作る。
<code>float()</code>	引数に渡した文字列から、浮動小数点数のデータを作る。
<code>math.sqrt(値)</code>	引数の値の平方根を返す。

関数定義の構文

構文	説明
<pre>def 関数名 (引数 1, 引数 2, ...): 処理 1 処理 2 return 戻り値</pre>	<p><code>def</code> キーワードに続いて、関数名を書く。</p> <p>関数名の後ろに <code>()</code> を書き、引数を書く。複数の引数を定義する場合、<code>,</code> <code>"</code> で区切って並べる。</p> <p>関数内の処理は、インデントを揃える。</p> <p><code>return</code> キーワードを使って、呼び出し元に返す値や変数を指定する。</p>

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。
- ② `random.randint()` 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。

(ヒント)

```
aisatsu = ["おはよう", "こんにちは", "こんばんは"]
```

問 2. 組み込み関数 `len()` を使い、次のようなプログラムを作成してください。なお、`len()` は、引数としてリストを渡すと、その要素数を返します。

- ① あいさつの言葉を 5 つ以上含むリストを作る。5 つ以上ならいくつ入れてもよい。
- ② リストの要素数を表示する。

問 3. 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

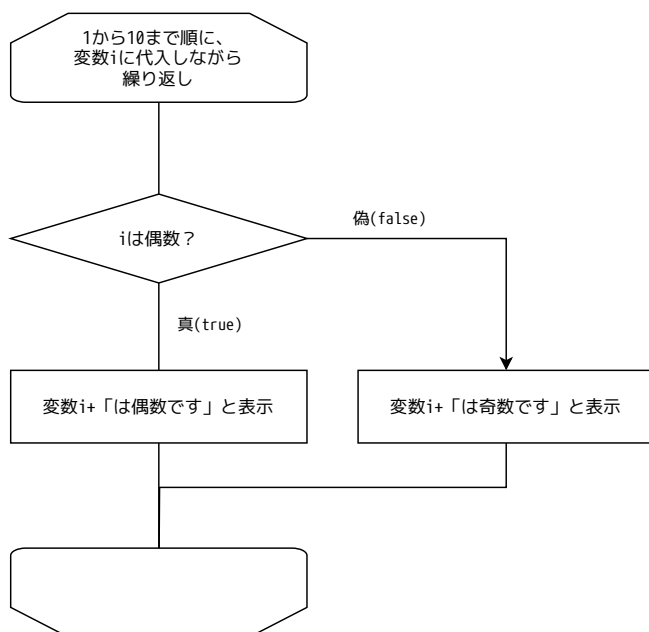
これまでに学んだ繰り返し（反復）の制御と、選択（分岐）の制御を組み合わせると、複雑なプログラムを作ることができます。プログラムで何か問題を解決したい場合、制御の組み合わせが必要になります。

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。

Python プログラムでは次のように書きます。インデント（字下げ）に注目してください。

```
for i in range(1, 11):
    if i % 2 == 0:
        print(i, "は偶数です ")
    else:
        print(i, "は奇数です ")
```

1 行目では、関数 `range()` を使い、1 から 10 までの範囲を作っています。1 つずつ値を取り出し、変数 `i` に代入しています。

2 行目は字下げされています。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」かどうかを比較します。余りが 0 ならば偶数なので、関数 `print()` で「偶数です」と表示させます。関数 `print()` の行は、2 回目の字下げがされていることに注目しましょう。

4 行目の「`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。

実行結果は次のようになります。

```
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
```

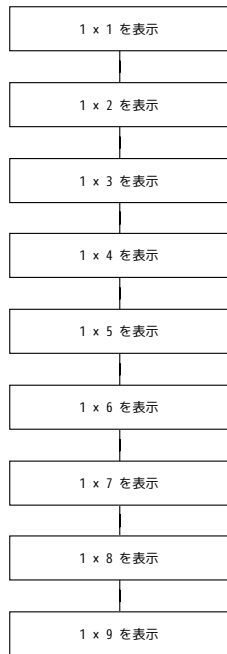
例題 1: 繰り返しの構造と、選択の構造を使って、1 から 30 までの間の、3 の倍数だけ表示するプログラムを作成してください。

「繰り返し」を繰り返す

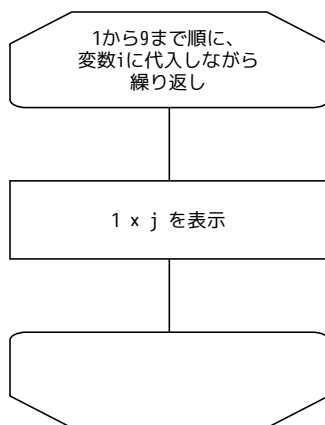
掛け算の「九九」をプログラムで表現するにはどうしたらよいでしょうか？

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。

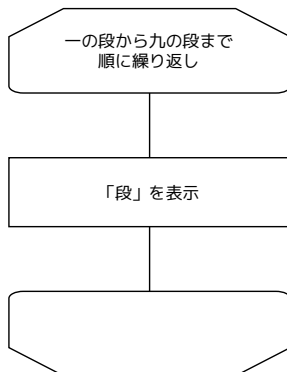


プログラムで表現してみましょう。

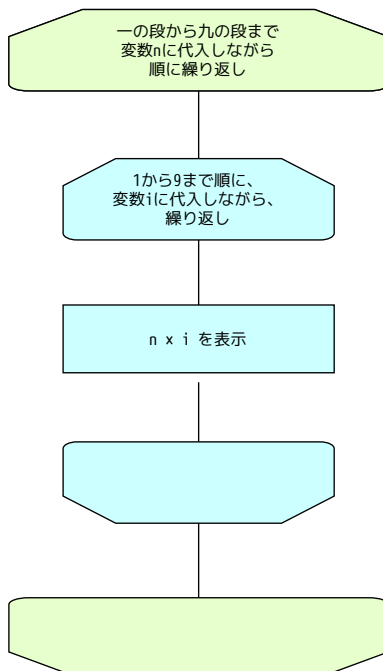
```
for i in range(1, 10):
    print(1 * i, end=' ')
```

※ここで 関数 `print()` の引数に、「`end=' '`」という値が渡されています。普通の関数 `print()` では値を表示した後に改行しますが、かわりに空白を1つ分、表示するようにしています。

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。

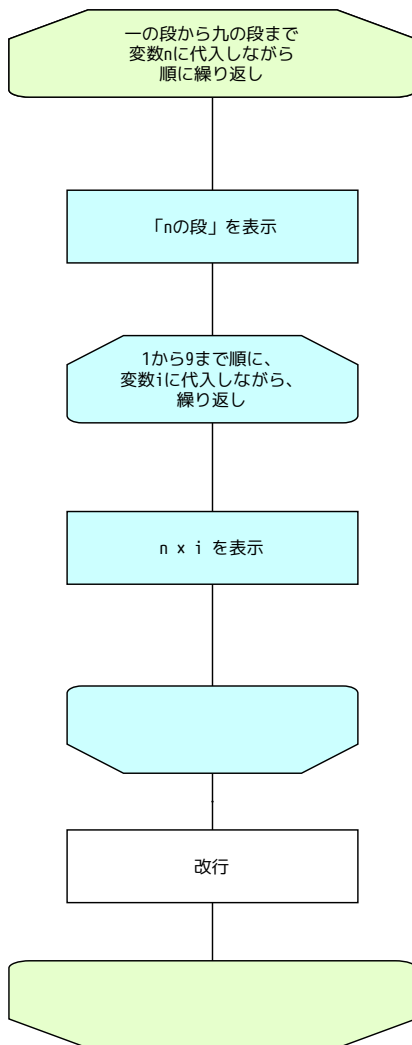
結果を見やすくするために、「の段」という表示をする処理を付け足しています。また、1つの段を表示した後に改行を表示して、次の段の表示が始まったことが分かるようにしています。

```
for n in range(1, 10):
    print(n, " の段 ")
    for i in range(1,10):
        print(n * i, end=' ')
    print()
```

最後の5行目にある、引数を取らない関数 `print()` は、単に改行だけするために書かれています。字下げに注目してください。

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、 $\times 1$ から $\times 9$ 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: リスト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、そのリストの内容を先頭から順に表示する処理を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。

```
kotae = 0
for n in range(1, 10):
    print(n, " の段 ")
    for i in range(1, 10):
        kotae = n * i
        print(kotae, end=' ')
    if kotae > 25:
        break
    print()
```

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、変数 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
```

「27」を出力し、25を超えたところで終了します。

例題 3: if キーワードの字下げ（インデント）のレベルを変えると、プログラムの動作が変わります。「print(kotae)」と同じレベルに変えて、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。

```
kotae = 0
for n in range(1, 10):
    print(n, "の段")
    for i in range(1, 10):
        kotae = n * i
        print(kotae, end=' ')
        if kotae > 25:
            break
    print()
```

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、リスト）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、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

関数 print() のオプション

オプション	説明
<code>print(値 , end=' 最後に表示する値 ')</code>	「値」の後に、 <code>end=</code> で指定した値を末尾に付け足して表示する。
<code>print()</code>	引数無しで <code>print()</code> を実行する。 改行だけ出力される。

繰り返しと反復の中断・継続

キーワード	説明
<code>break</code>	<p><code>for</code>、<code>while</code> の繰り返し処理を中断する。</p> <p>キーワード <code>for</code>、<code>while</code> を使って、二重以上の繰り返しをしている場合、<code>break</code> 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。</p>
<code>continue</code>	<p><code>for</code>、<code>while</code> の繰り返し処理の中に <code>continue</code> 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。</p> <p>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。</p> <pre>for i in range(1, 6): if i == 3: continue else: print(i)</pre>

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リストを作る。
- ② リスト `aisatsu` の中から、偶数番目の要素を取り出し、画面に表示する。

(ヒント)

```
aisatsu = ["おはよう", "こんにちは", "こんばんは"]
```

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
```

(ヒント)

`range()` の3つ目の引数に `-1` を指定すると、1ずつ減る範囲を作れます。

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。

本書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/>

Python で学ぶプログラミング入門

2022 年 4 月 1 日 初版第 1 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3-1-4

ファーストジェネシスビル

<https://edu.monaca.io/>

(C)ASIAL CORPORATION 2022 Printed in Japan

本書のコピー、スキャン、デジタル化等の無断複製は著作権法上での例外を除き禁じられています。本書を代行業者等の第三者に依頼してスキャンやデジタル化することは、たとえ個人や家庭内の利用でも一切認められておりません。