※本書は執筆途中の試作品です。書籍の内容は変更される可能性があります。

表紙の裏（印刷用語では表 2 と呼ばれる部分）

目次

序章 Monaca Education をはじめよう

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もあります。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/

※　ビジネス用の Monaca に迷い込まないよう、注意して下さい。

次に右上の「アカウント作成」をクリックして下さい。

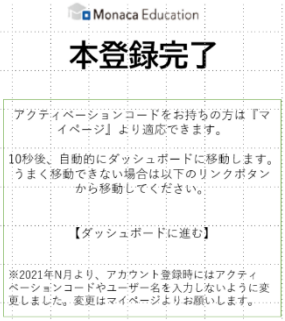
下図のようなアカウント作成フォームが表示されますので、先生の指示に従ってアカウントを作成します。特に、Google アカウントや Microsoft アカウントと連携してア

カウントを作る場合、メールアドレスの入力は行いませんので注意して下さい。

仮登録から本登録に進む(メール登録の場合のみ)

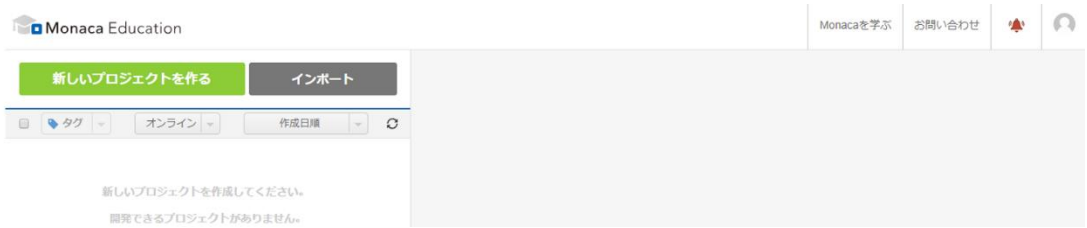
メールアドレスで登録した場合、アドレス確認のために仮登録扱いとなります。メールボックスに仮登録完了メールが届いたら案内に従って本登録を進めて下さい。
なお、Google アカウントや Microsoft アカウントで登録する際には仮登録のステップ無く、本登録が完了します。

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。



注：2022 年度版は本登録時の作業がなくなり、確認メールのクリックだけになる予定です。

登録を完了させるとダッシュボードが表示されます。



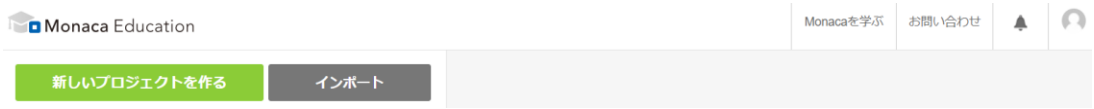
ダッシュボード上では制作中の作品を『プロジェクト』という単位で管理します。以上で Monaca Education のアカウント登録は完了です。

プロジェクトを動かそう

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

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。



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。

注：2022 年度版はテンプレートの入れ替えを予定。ただしブロック崩しは残る。

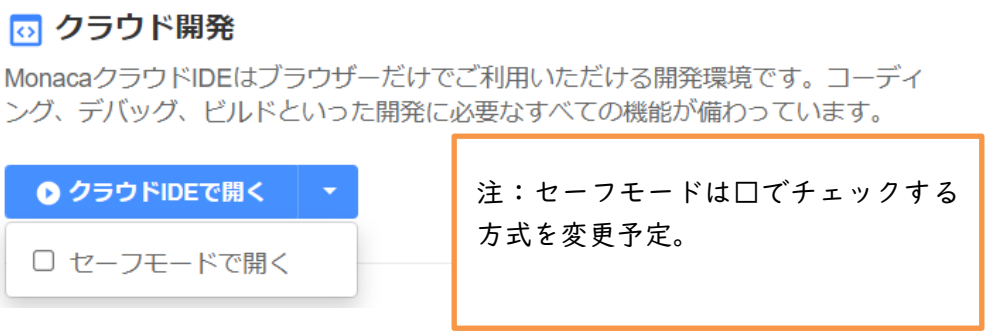
プロジェクトにはプロジェクト名や説明を設定できます。今回はテンプレートのまま進めることにします。

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。



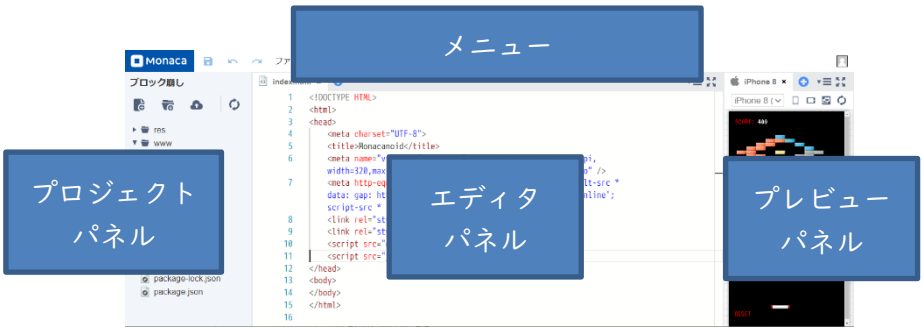
注：セーフモードは□でチェックする方式を変更予定。

Monaca クラウド IDE でプロジェクトを改造しよう

IDE にはアプリ制作に必要となる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。

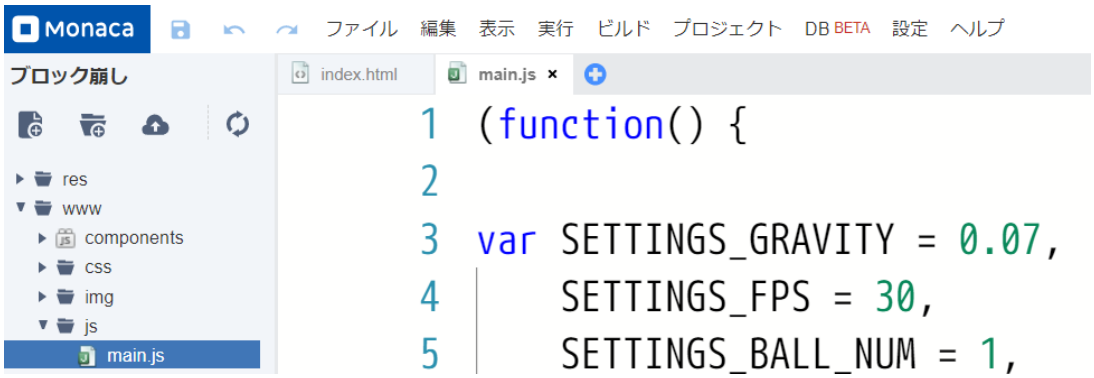


プロジェクトパネルから js/main.js を開く

フォルダのアイコンの左にある▶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。

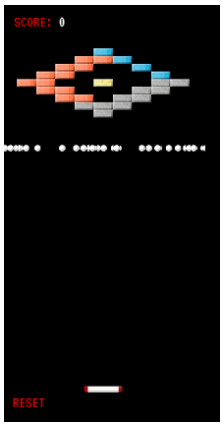


【変更後】

```
    SETTINGS_BALL_NUM = 100,
```

変更したら保存も忘れずに行って下さい。IDE メニューの左上に保存を示す『フロッピー』のアイコンがあり、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ます(Ctrl + s)。

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。

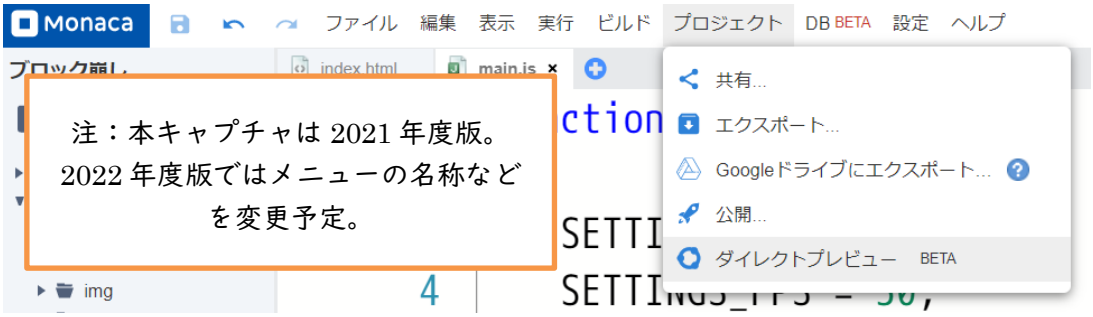


作品を Web で公開しよう

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（旧：ダイレクトプレビュー機能）

メニューの『プロジェクト』から Web 公開を選択して下さい。



注：本キャプチャは 2021 年度版。
2022 年度版ではメニューの名称などを変更予定。

選択すると即座に作品が公開されます。

注：2022 年度版では即座に公開する予定。
加えて、期限を設定できるようにする予定。

QR コードも表示されるため、スマートフォンの利用が許可されている場合、こちらをスマートフォンで読み取ることで、自身の端末でプログラムを動かせます。

予備ページ

注：☆何らかの記事を追加予定。
第一候補は、新規プロジェクト作成
第二候補は、画像編集機能。

第 1 章 変数と順次実行

注：Python テンプレートの入手動線は現在検討中。現時点では、あんこエデュケーションのものをご利用ください。

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")
```

プログラムの中にある”print”は、プログラム言語 Python が備える命令の一つで、()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。

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。

```
こんにちは
プログラミング
```

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

例題 1: 順次実行

上のプログラムを変更して、次のような実行結果が表示されるようにしてください。

```
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。関数については、後のページで詳しく扱います。
補足 2: プログラムの中で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
```

プログラムの最初の行にある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。

数値と計算

ここまで、文字列の値ばかり扱ってきました。プログラムの中で、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。

```
b = 10
print(b)
c = b + 10
print(c)
print("b =",b,"c =",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 = 10c = 20
```

最初に、変数 b に 10 という数値を代入しています。次の行で print()を使って、変数 b の値を表示させています。
さらに変数 c に値を代入しています。変数 b と 10 の足し算の結果（20）を代入しています。print(c)で、変数 c に代入された値を表示しています。
最後の行の print()では、表示させたい値をカンマ（ , ）で区切って、一度に複数の値を表示させています。

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。

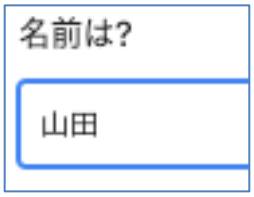
入力を受け付ける

次のプログラムを実行してみましょう。

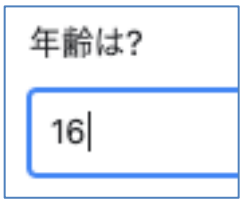
```
a = input("名前は?")
print("名前:",a)
b = input("年齢は?")
b = int(b)
print("年齢:",b)
```

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）。

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、（※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。



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 input()は、変数 b に受け取った値を代入しています。
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 int()は、文字列を数値に変換しています（上の例では、"16”を 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出来ないので、数値に変換するために int()を使います。

例題 4:ユーザーの入力を受け取る

次のようなプログラムを作成してください。

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、け取った内容を変数 a に代入して、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( )	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）。 複数の値をカンマで区切って、カッコの中に書くこともできる。その場合、書いた順に表示される。
input( )	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）

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、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。

値の型を変換する関数

関数名	動作
print( )	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)。 複数の値をカンマで区切って、カッコの中に書くこともできる。その場合、書いた順に表示される。
input( )	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)

問題集

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。
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。
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 条件による選択（分岐）

条件による選択（分岐）

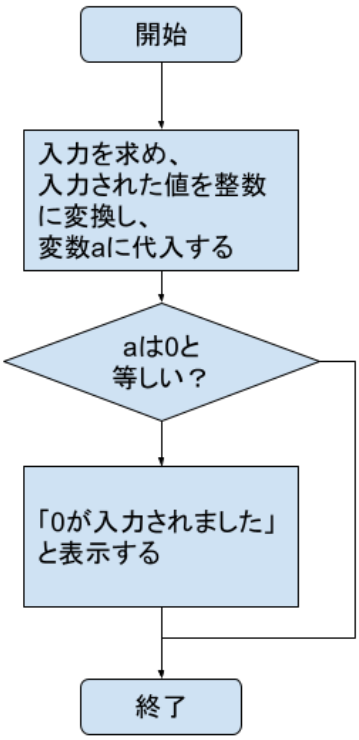
動かすたびに、いつも同じように動くプログラムが必要なのと同じくらい、操作する人の入力や操作に合わせて動作を変えられると、便利です。この章では、条件に応じて動作を変えるプログラムを作ります。
まず、次のプログラムを実行してみましょう。

```
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？

更新ボタンを押して、プログラムを再度実行します。
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。



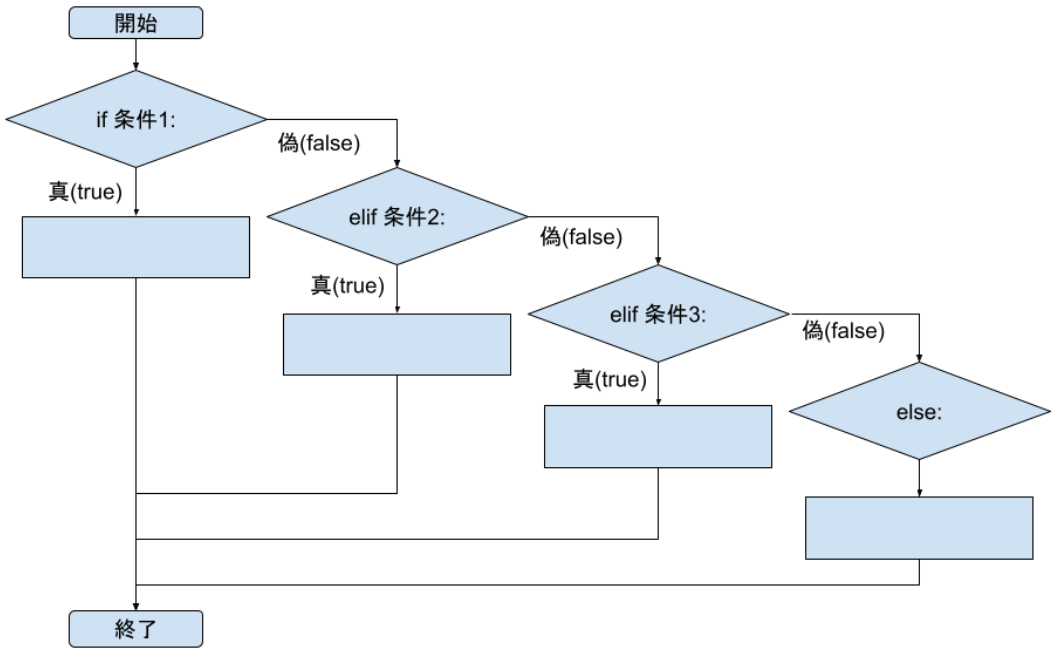
１行目: input()で入力を受け取り、int()で文字列の値を数値に変換しています。

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（=）記号が二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、a と 0）を比較して、等しいかどうか調べる式です。
等しければ、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、偽（false）となり、何も表示されず、プログラムは終了します。

補足資料

if-elif-else

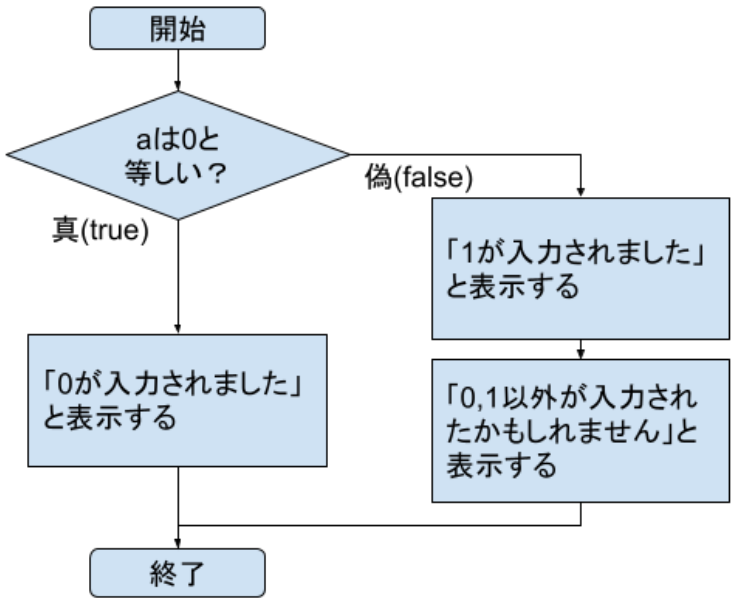
if キーワードだけ用いて、ある条件が真になるときだけ処理を実行させることができます。else を使うと、if の条件が偽のときに実行する処理を指定することができます。
if の後に elif を使い、別の条件による判定を行い、さらに分岐させることができます。
elif はいくつでも付け足すことができます。



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。

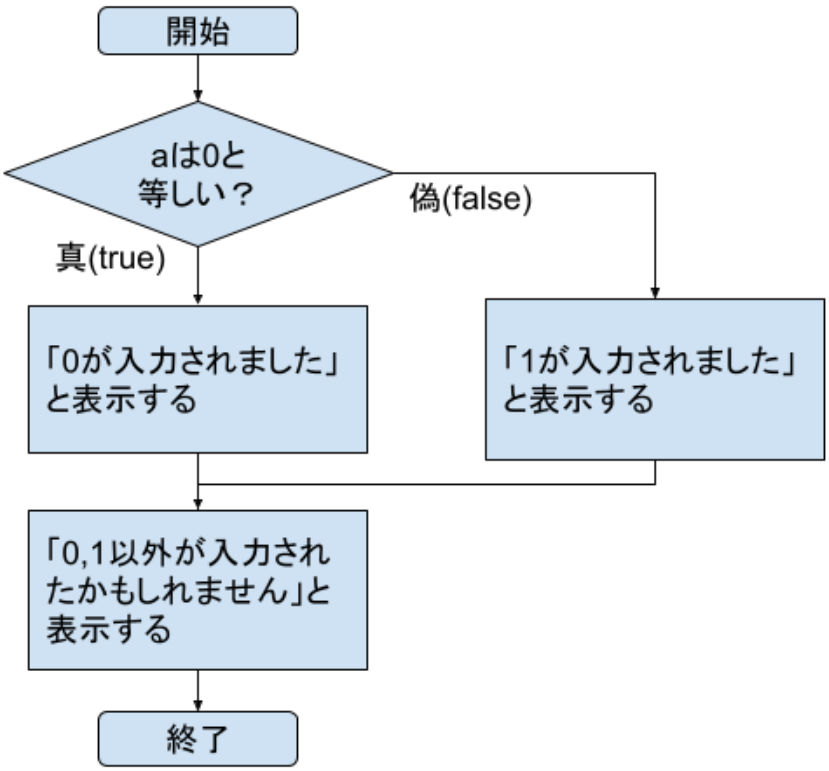
```
if a == 0:
    print("0 が入力されました")
else:
    print("1 が入力されました")
    print("0,1 以外が入力されたかもしれません")
```

else:の後ろにある二つの 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。

二つ目の 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
==	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
random.randint(a, b)	a 以上 b 以下のランダムな整数を返す。 たとえば、random.randint(1,6) なら、1,2,3,4,5,6 のいずれかがランダムに返される。
random.random()	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

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。

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. 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（※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)。
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」

第３章 リスト

一つの変数に複数の値を入れる

これまでに扱ってきた変数は、最後に代入した値を一つだけ持つことができました。
リストを使うと、一つの配列変数に、複数の値を格納することができます。
次のプログラムを実行してみましょう。

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

"B"と表示されます。

この図は、リストの変数 a のイメージです。

０、１、２は添え字（そえじ、インデックス）といい、リストの変数 a 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数から１引いた数まで（※例では 3 個の要素がありますから、3-1 で２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: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"と、「配列名[添え字]=新しい値」という書き方で、指定した要素に新しい値を代入を行うこともできます。

```
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

次のプログラムの最初の行を見てみましょう。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。

最初は要素は一つもありません。

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

次の行から、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。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。配列名[添え字」で値を利用することができます。

例題 4:リスト g 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、最後にリスト g 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/リストの要素を削除する

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、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には、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] )
```

a.insert(2,"B")の後に、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 g 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、添え字が 1 の要素を削除して、最後にリスト g 全体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。

第 3 章 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] ＃上の例なら、２が得られる（０番目の次、1 番目）
配列名.length	配列の要素数を調べる。 例: a.length ＃上の例なら、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.push(10) ＃配列の中身は[1,2,10]になる
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[1,2,10]のとき、 a.insert(0, 100)　＃とすると、[100, 1, 2, 10]となる。
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.pop() ＃５が返され、配列の中身は[1,2]になる。
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.remove(1) ＃配列の中身は[2,10]になる。

第 3 章 問題集

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 )
```

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）

処理を繰り返す

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処理を選択（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。**繰り返し構造**または**反復構造**といいます。

決まった回数だけ繰り返し実行する

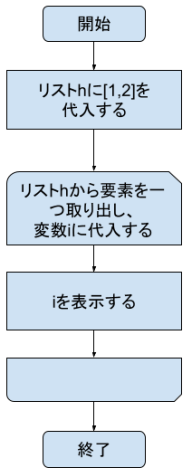
次のプログラムを実行してみましょう。

```
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。



プログラムの最初の行はリストです。リスト h には 1 と２の二つの要素が含まれています。

次の行は 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 の先頭から一つ値を取り出して（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 から二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)を表示する』という処理を繰り返しています。

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』と表示する」という二つの処理を、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）

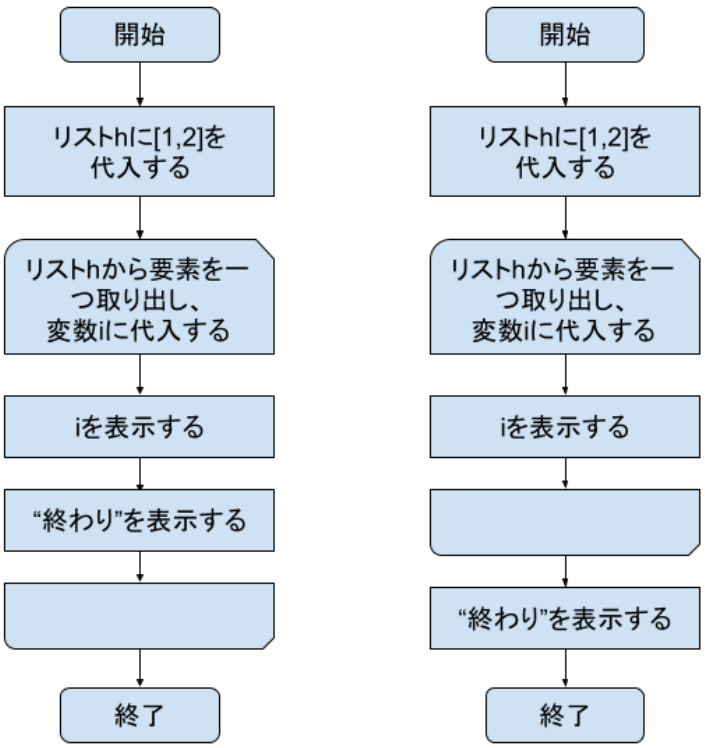
```
h = [1,2]
for i in h:
    print(i)
    print("おわり")
```

（修正後）最後の("おわり")の print 文の字下げ（インデント）を無くしています。

```
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]と書くことも出来ますが、range()関数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
```

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。

```
98
99
100
```

range( )の括弧の中に、二つの値をカンマ記号（ , ）で書くと、一つ目の値（開始）から、二つ目の値（終了）の一つ前の値までの「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、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
    ...
```

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: range( )のいろいろな使い方

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」を実現できます。

例題 4: for 文と 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: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つの値を括弧の中に書いています。

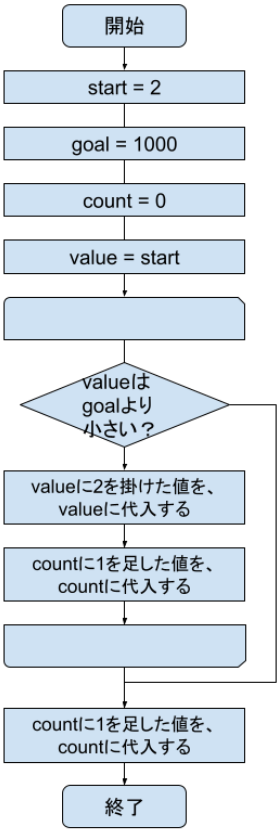
例題 6:range( )の括弧の中を次のようにすると何が起きるか、確認してみましょう。

```
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

あらかじめ繰り返しの回数が見通せる場合、for 文を使うと簡単です。
ところで、「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、次に紹介する while 文を使う方が、書きやすくなります。
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。



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 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 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 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 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 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。value < goal は、変数 value と、 変数 goal を比べています。goal の方が大きければ真（true）、value の方が大きければ偽（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 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 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の while 文に戻ります。再び value と goal の値を比べ、goal の値の方が大きければ⑥、⑦の処理を繰り返します。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 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 の後ろに様々な式を書くことで、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「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 7: while 文と 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 )
```

例題 8: for 文と 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。

第 4 章 補足資料

for 文

構文	説明
for 変数 in リスト： 　　処理 1 　　処理 2 　　… 処理 A	「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。 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。 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。 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。

range( )

構文	説明
range(終了)	一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。 例: range(5) -> [0, 1, 2, 3, 4]
range(開始, 終了)	二つの整数を渡す。開始の値から、「『終了』より小さい値」の整数の範囲を返す。 例: range(1, 5) -> [1, 2, 3, 4]
range(開始,終了,ステップ)	三つの整数を渡す。開始の値から、「『終了』よ

	り小さい値」の整数の配列を返す。 ステップに指定した数ずつ増える。 例: range(1, 5, 2) -> [1, 3] 「開始」の値を「終了」の値より小さくし、ステップの値をマイナスにすると、徐々に減る範囲が作れる。 例: range(10, 1, -1) -> [10, 9, 8, 7, 6, 5, 4, 3, 2]

while 文

構文	説明
while 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、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。 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 I 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。

第４章 問題集

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。

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。
（ヒント）range( )をうまく使うと if 文無しでも実現できます。

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。
(ア) 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)。
(イ) 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。

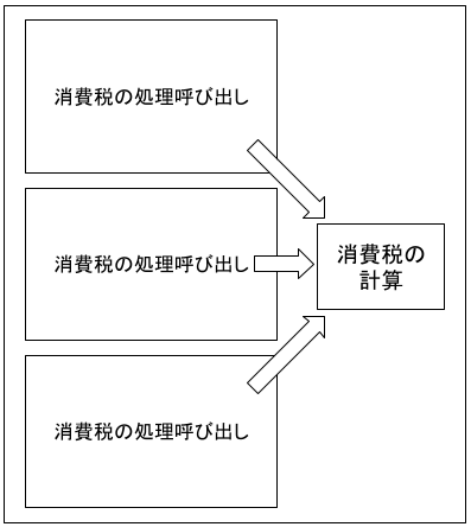
第 5 章 関数の定義と利用

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なら「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・処理の内容を変えなければならなくなったときに、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。関数の中に一連の処理を書きます（これを関数を**定義**するといいます）。プログラムの中で、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、関数の 1 か所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。

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

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（input()）、値を出力したり（print()）、乱数を生成したり（random.randint()）、範囲を作ったり（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()
print( hiduke )                    # 今日の日付を取得する
```

最初の行の from datetime import date は、組み込み関数を使うための準備です。
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 )
```

最初の行の 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: hiduke_jikoku.year, 月: hiduke_jikoku.month, 日: 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 aisatsu()という名前の関数を定義しています。
② 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 関数 aisatsu()を呼び出しています。

例題 3 上のプログラムの関数 aisatsu()の呼び出し（※最後の行）を①2 回にした場合、②0 回にした場合どうなるか確認してみましょう。

関数の引数と戻り値

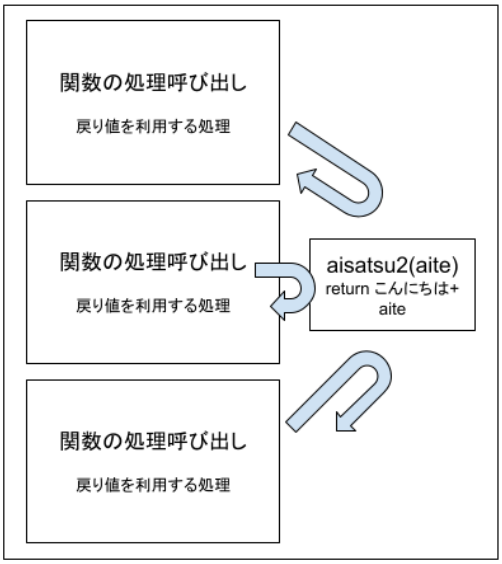
関数に値を渡し、関数内の処理で使うことができます。引数（ひきすう）といいます。

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


aisatsu("Python")
```

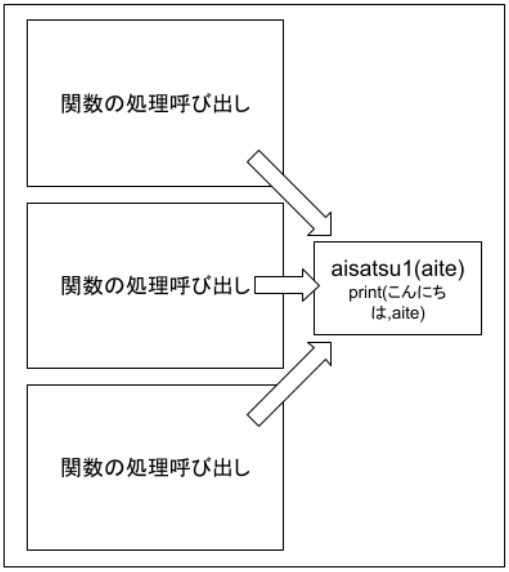
関数から、呼び出し元に値を返すこともできます。return キーワードを使います。この返す値を戻り値（もどりち）といいます。

```
def aisatsu2(aite):
    return "こんにちは" + aite


kotoba = aisatsu("Python")
print( kotoba )
```

上の二つのプログラムは、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。

第 5 章 補足資料

日付・時刻

組み込み関数	説明
date.today()	使用時の宣言: from datetime import date hiduke = date.today() プログラムを実行したその日の日付を取得する。
datetime.now()	使用時の宣言: from datetime import datetime hiduke_jikoku = datetime.now() 現在の日付・時刻を取得する。 年（hiduke_jikoku.year)、月（month)、日（day） 時刻（hour)、分（minute)、秒（second）
time.time()	使用時の宣言: import time 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。

さまざまな組み込み関数

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

関数定義の構文

構文	**説明**
def 関数名(引数 1, 引数 2, ...): 処理 1 処理 2 return 戻り値	def キーワードに続いて、関数名を書く。 関数名の後ろに( )を書き、引数を書く。複数の引数を定義する場合、","で区切って並べる。 関数内の処理は、インデントを揃える。 return 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。

第 5 章 問題集

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 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。

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。

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

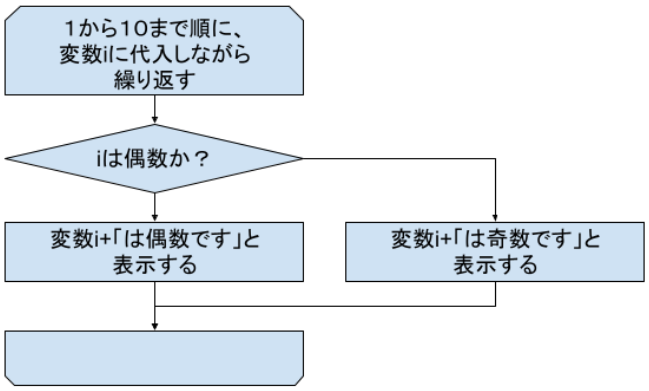
（税額計算処理）
関数名: zeigaku_keisan( )
引数: kingaku, zeiritsu

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

「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」
このように振る舞うプログラムを考えてみましょう。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")
```

最初の行では、range()を使い、１から 10 までの範囲を作っています。一つずつ値を取

り出し、変数 i に代入しています。

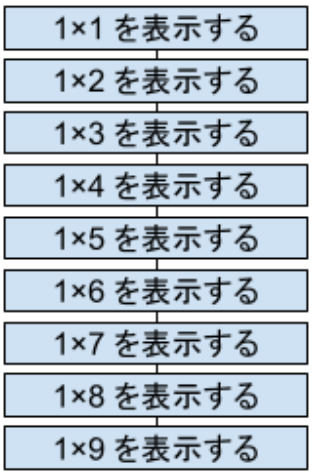
```
1 は奇数です
2 は偶数です
3 は奇数です
4 は偶数です
5 は奇数です
6 は偶数です
7 は奇数です
8 は偶数です
9 は奇数です
10 は偶数です
```

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かどうか比較します。余りが０なら偶数なので、print()で「偶数です」と表示させます。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。
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。

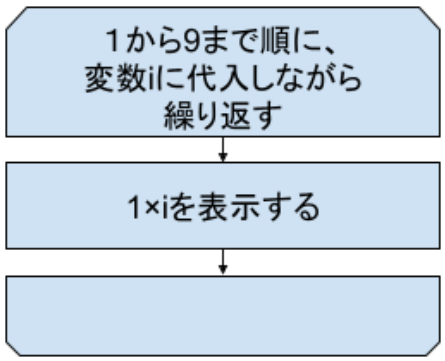
例題 1 繰り返しの構造と、選択の構造を使って、１から３０までの間の、３の倍数だけ表示するプログラムを作成してください。

「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。

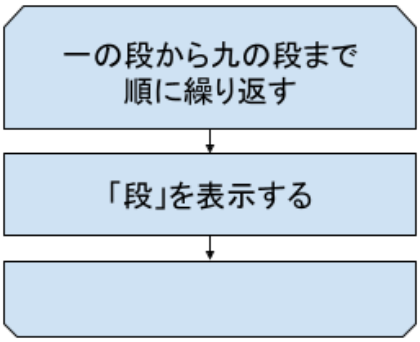


プログラムで表現してみましょう。

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

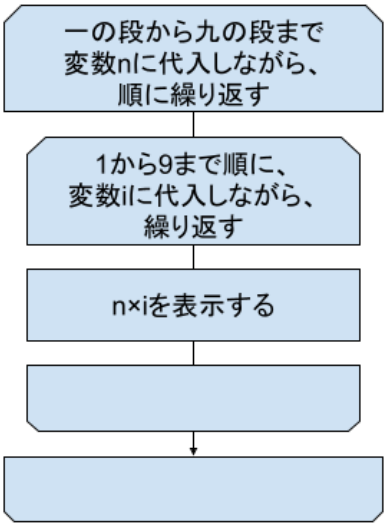
※ここで 関数 print()の引数に、"end=' '"という値が渡されています。普通の print( )では値を表示した後に改行しますが、かわりに空白をひとつ分、表示するようにしています。

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。

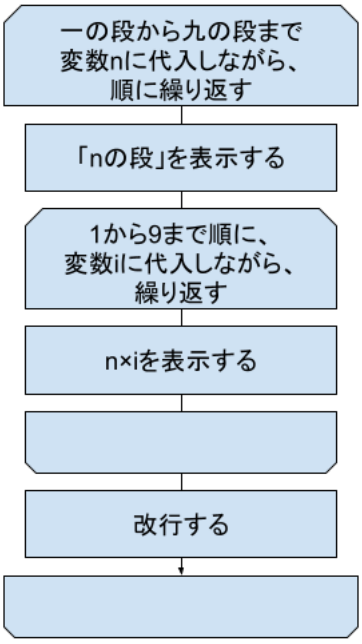


プログラムにすると次のようになります。
結果を見やすくするために、"の段"という表示をする処理を付け足しています。また、ひとつの段を表示した後に改行を表示して、次の段の表示が始まったことが分かるようにしています。

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

最後の行にある、引数を取らない 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、×1 から×9 までを表示する）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のを、3 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（二重になった繰り返し。"ニ重ループ"）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()
```

最初の行で、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。
次の行から、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。
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

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。
ある問題を解くためのアルゴリズムは一つとは限りません。複数のアルゴリズムが考えられ、アルゴリズムごとに、計算する速度が速い/遅い、使用するコンピュータの資源（変数やリストの数）が多い/少ないなどの特徴が現れます。

良いアルゴリズムを実現するためには、問題をよく理解した上で、問題を分割して考えるなどの工夫をし、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）を組み合わせて考える必要があります。

第 6 章 補足資料

関数 print()のオプション

オプション	説明
print(値, end='最後に表示する値')	「値」の後に、 end=で指定した値を末尾に付け足して表示する。
print()	引数無しで print()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
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 i in range(1,6):
	　if i == 3:
	　　continue
	　else:
	　　print(i)

第 6 章 問題集

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。

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

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。

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。

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。

奥付

☆最終ページは奥付が入ります
☆64 ページ目を持って最終ページとする予定です