第1章 順次実行と変数

順次実行

- プログラムに書かれた順に、処理を実行する
  - コンピュータが、プログラムを上から下へ読み、書いてある順番通りに実行する
  - それぞれの処理を1回だけ実行する

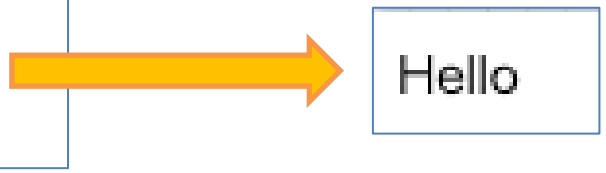
```
print("こんにちは")
print("プログラミング")
```

- ※これだけでは本格的なプログラムにならないが、だいじな基本
  - 他に、「選択（分岐）」「繰り返し（反復）」がある

変数

- 値を代入できる
  - 繰り返し代入できる。最後に代入した値だけ保持する

```
a = "Hello"
print(a)
```



- 代入しておいた値を、後から参照できる
  - aに入れておいた“Hello”を
  - print(a)で使っている

変数

- 最後に代入した値だけ保持する

```
a = "Hello"
a = "こんにちは"
print(a)
```

- 変数aに、2回代入している

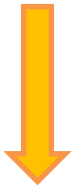
- 実行結果はどうなるだろうか？

  「Hello」と表示するだろうか？

  それとも、「こんにちは」と表示するだろうか

数値と計算

- プログラムの中で、数値を扱う
  - 計算することもできる



```
b = 10
print(b)

c = b + 10
print(c)

print("b =" ,b,"c =",c)
```

- たくさん処理が並んでいる
- 順次実行している

b + 10 の計算結果をcに代入している

- 変数b, cをもう一度参照している
- 文字を結合している

ユーザーからの入力を受け付ける

- ユーザー（プログラムを動かしている人）の操作、入力を受け付ける
  - 入力の結果を使って、プログラムの動作を変える

```
a = input("名前は?")
print("名前:",a)

b = input("年齢は?")
b = int(b)

print("年齢:",b)
```

- input( )で入力を受け付ける
- 入力された値は、変数aに代入される

- input( )で受け付けた値が、変数bに代入される

- 変数bの中の文字列の値を数値に変換する。その値を再び変数bに代入する

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）

値の型を変換する関数

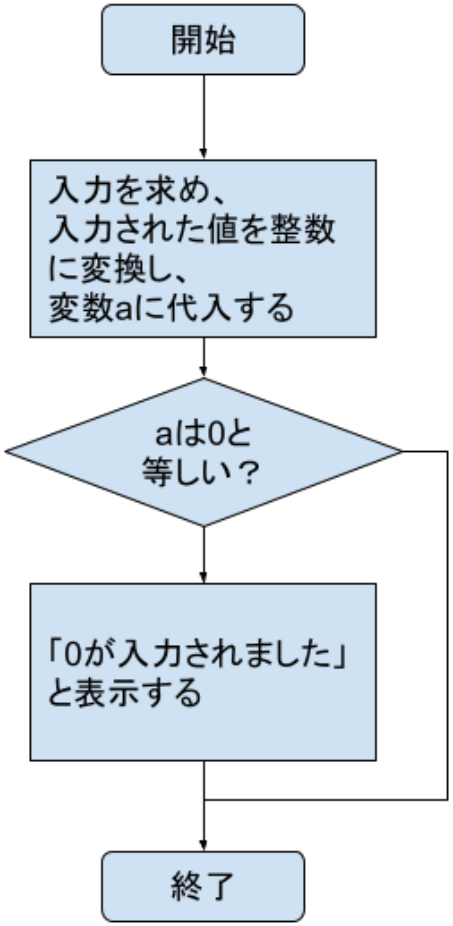
関数名	動作
int( )	int("16")のように、整数として扱える文字列が渡されたとき、数値型の値に変換する。 a = int("16") とすると、 変数aには16という数値の型の値が（"16"という文字列の値ではなく）代入される。 整数として扱えない文字列が渡された場合、プログラムを実行した時にエラーになる。
float( )	float("174.5")のように、小数点を含む数値として扱える文字列が渡されたとき、数（浮動小数点数）に変換する。 浮動小数点数として扱えない文字列が渡された場合、プログラムを実行した時にエラーになる。

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、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。

第2章 条件による選択（分岐）

選択（分岐）

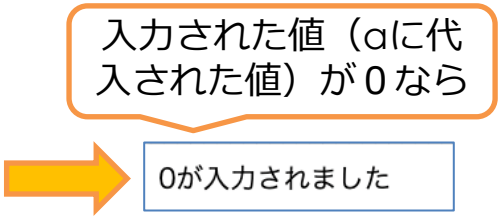


- 条件の評価の結果に応じて、処理が選択される
- プログラムのどの場所が実行されるか、コンピュータが判定して、分岐する
- 3つの制御構造の組み合わせで、複雑なプログラムを作ることができる
  - 順次実行　（※第1章）
  - 選択（分岐）
  - 繰り返し　（※第4章）

if（もしも）

- 値を評価して、結果が真（true）か偽（false）かにより、動作を変える
  - ifキーワードの後ろに、条件を表す式を書く

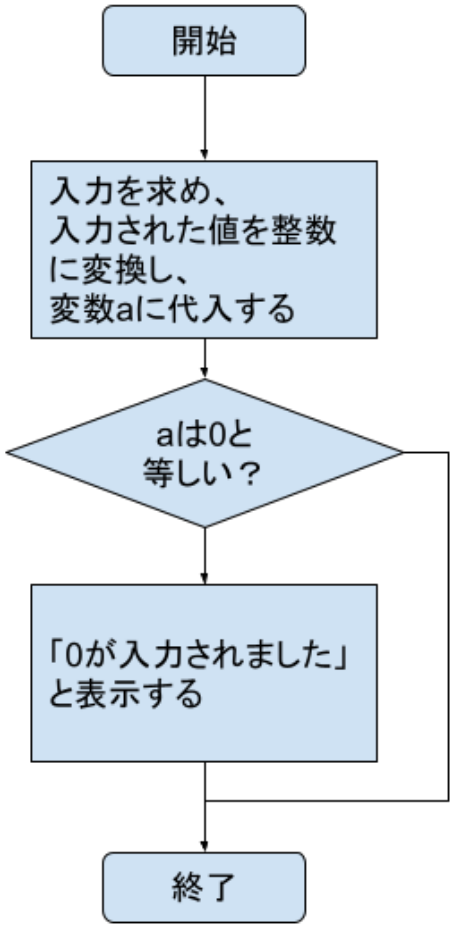
```
a = int( input("0か1を入力してください"))
if a == 0:
    print("0が入力されました")
```



  - 条件を表す式では、比較演算子を使う
    - == は比較演算子の一つ。左辺と右辺が等しいか調べる
  - ifの次の行はインデント（字下げ）して書く

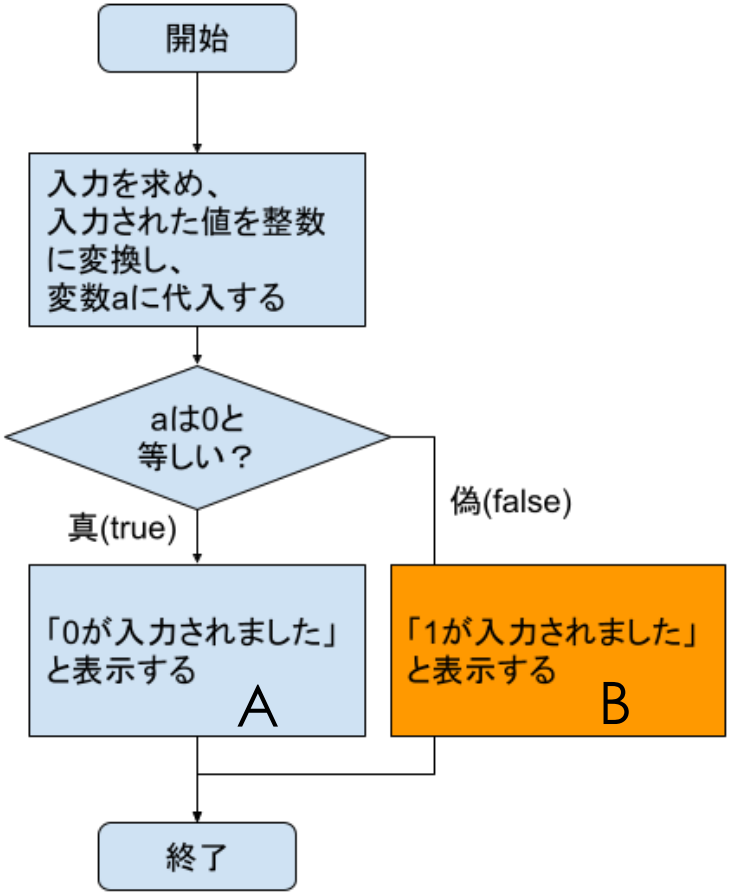
- if文の書き方

```
if 条件式:
    処理
```



- ifキーワード
  - 全て小文字
- 条件式 ：
  - 条件式とは「値を比較して、等しいか等しくないか調べる」などの式
  - 式の後ろに“:”を付ける
- 処理
  - 条件式が「真」（正しい）と判定されたときに実行する処理を書く
  - インデント（字下げ）する

if（もしも）else（でなければ）

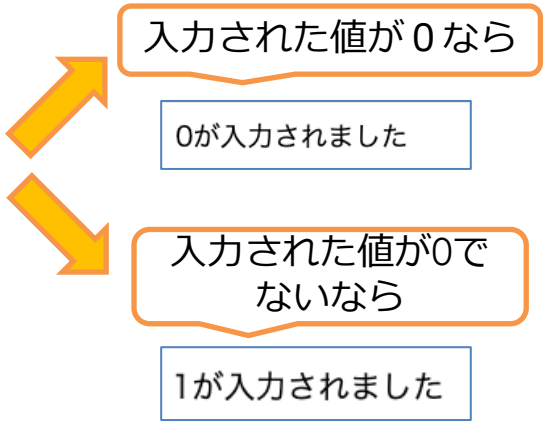


- **条件式を判定**
  - 真ならAの処理を実行
- **条件判定が偽なら**
  - Bの処理を実行

if（もしも）else（でなければ）

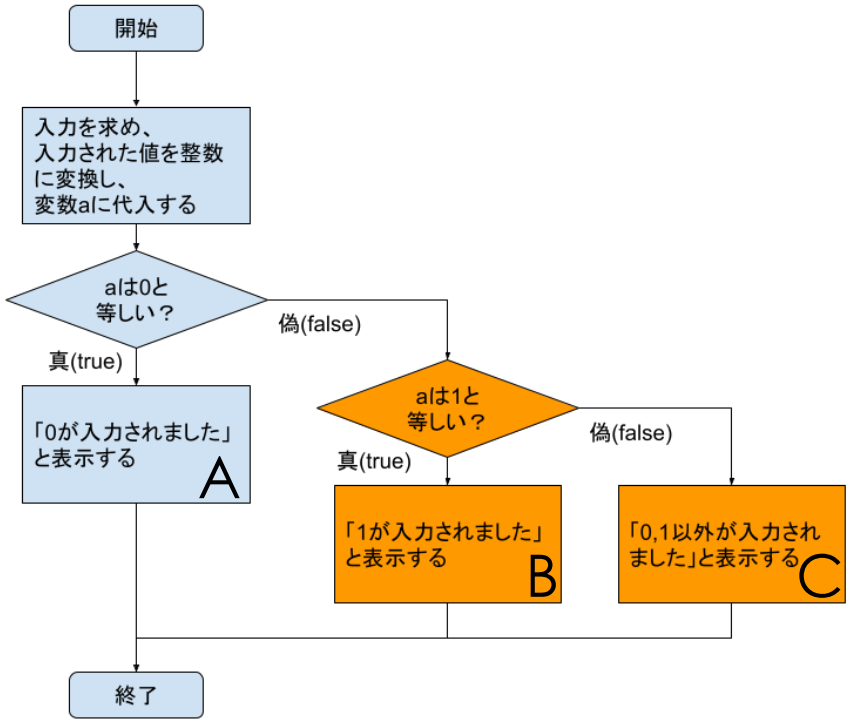
- ifの条件式が真でないときには、else以降の処理を実行する
  - elseキーワードの後ろに処理のブロックを書く

```
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")
```



  - elseの後ろには条件式は書かず、コロン（:）を書き、処理のブロックをインデント（字下げ）して書く

if（もしもAなら）…elif(Bなら)…
else（どれでもなければ）



- 一つめの条件式を判定
  - 真ならAの処理を実行
- 一つめの条件判定の結果が偽なら、二つめの条件式を判定
  - 真ならBの処理を実行
- 二つの条件判定がどちらも偽なら
  - Cの処理を実行

■ ifの条件式を判定して、偽（正しくない）なら、elifの後の2つ目の条件式を判定する

```
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")
```

乱数

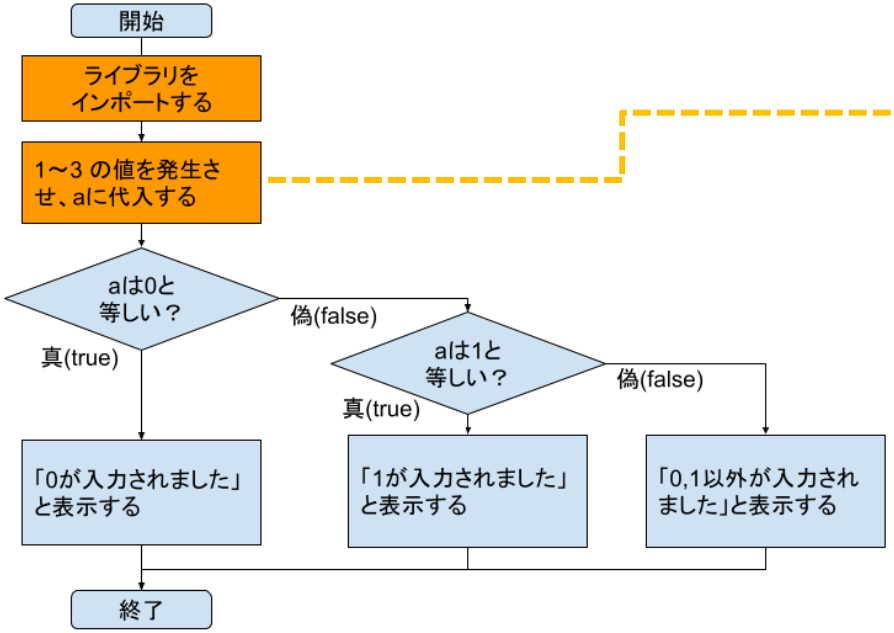
- **数字をランダムに並べたもの**
  - 規則性がなく、一つ一つの数字が現れる確率が等しい
  - コンピュータでは、擬似的な乱数を生成できる
- **Pythonでは**
  - importを使って、乱数を扱うためのライブラリrandomを読み込む
  - random.randint()関数で、指定した範囲から、ランダムに整数を得る

```
import random
a = random.randint(0,2)
```

0,1,2の値が変数aに代入される
どの値が代入されるかは、実行時に決まる

- **ライブラリとは**
  - プログラムから利用できる、機能の集まり
  - Pythonなど現代のプログラム言語には多数のライブラリがある

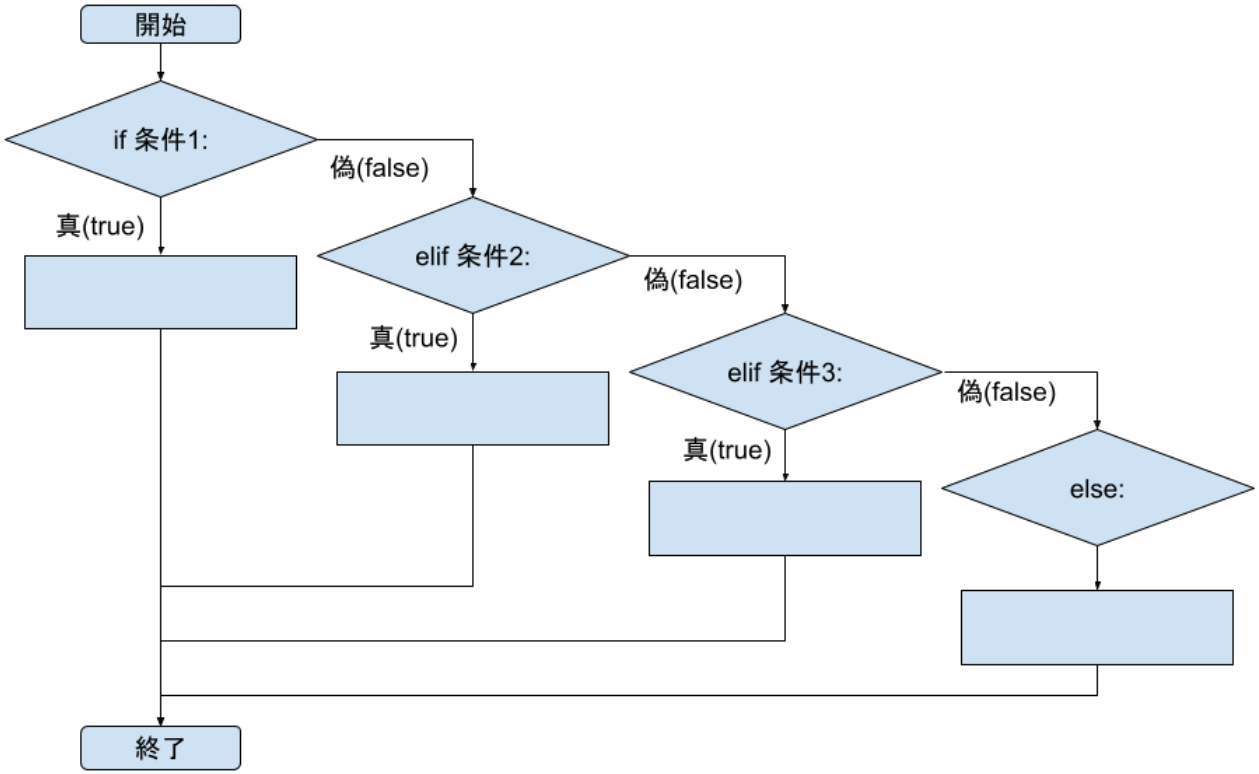
乱数



- Pythonプログラム

```
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")
```

if-elif-else



- ifの後にelifで条件を追加することができる
- elifを複数並べることができる
- elseを最後に付けることで、すべての条件が偽であるときの処理を作ることができる

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。

第3章 配列（リスト）

一つのリストに7つの値を入れる

- 一つのリストに7つの値を入れ、番号を指定して使う

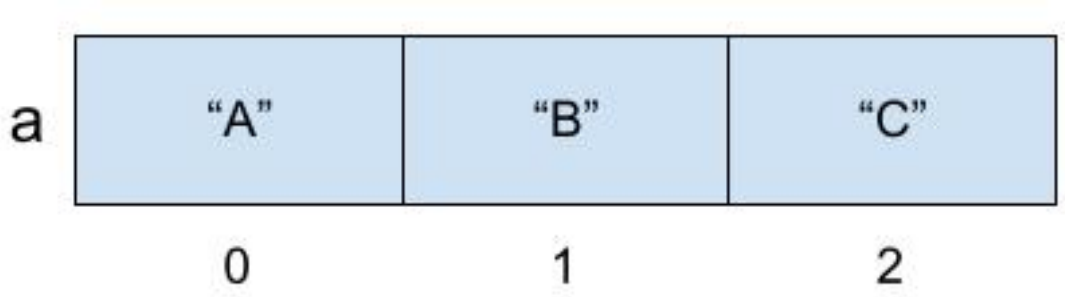

```
youbiList = ["月","火","水","木","金","土","日"]

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

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

指定した日付（例：2021年11月１日）の曜日を表す数字を取り出す

youbiListから数字で番号を指定して曜日の漢字を取り出す

リスト

- 一つの変数に複数の値を入れる



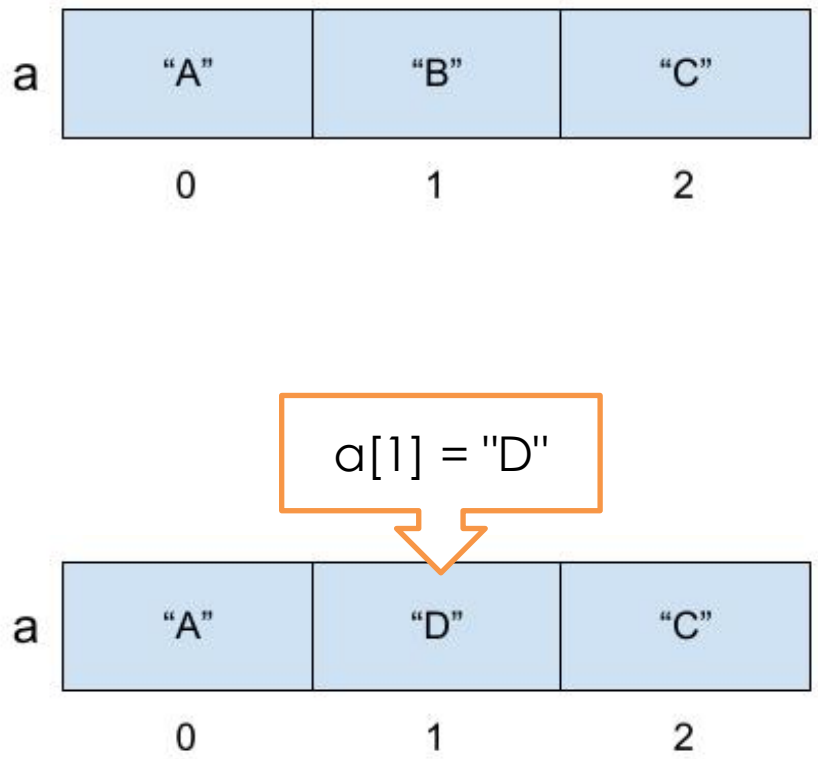
変数名[添字]
で要素を操作する

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

print(b)
```

B

リストの操作: 要素を置き換える

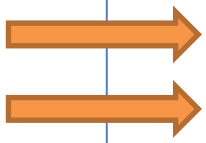
```
a = ["A","B","C"]
a[1]="D"
print(a[1])
```



補足: リストの全てを表示

- print(リスト)
- リスト全件を表示する
    - リストの全ての要素が［ ］で囲まれて表示される

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



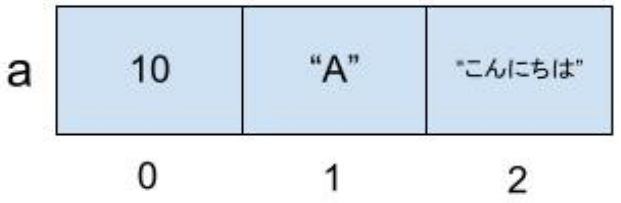
```
D
['A', 'D', 'C']
```

リストの操作:
空のリストを作り、要素を追加する

```
a = []

a.append(10)

a.append("A")

a.append("こんにちは")

print( a[2] )
```

① 空の配列を作り、変数aに代入する

② a.append(要素)で追加する



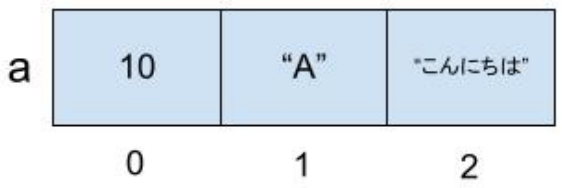
③ aの2番目を表示する

リストの操作:要素を挿入する

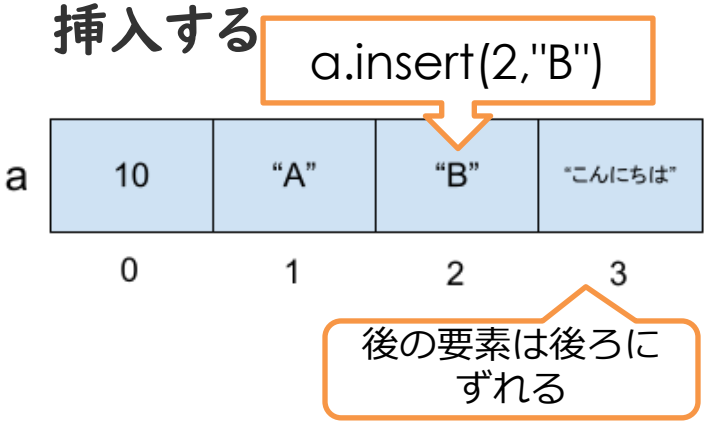
```
a = []

a.append(10)

a.append("A")

a.append("こんにちは")

a.insert(2,"B")

print( a[2] )
```

① 空の配列を作り、変数aに代入する

② a.append(要素)で追加する



③ a.insert(位置,要素)で挿入する



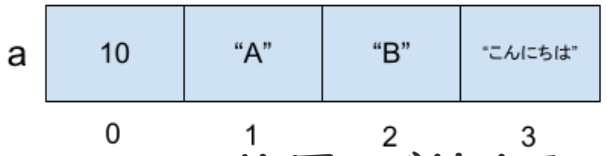
リストの操作:要素を削除する

```
a = []

a.append(10)

a.append("A")

a.append("こんにちは")

a.insert(2,"B")

a.pop(0)

print( a[2] )
```

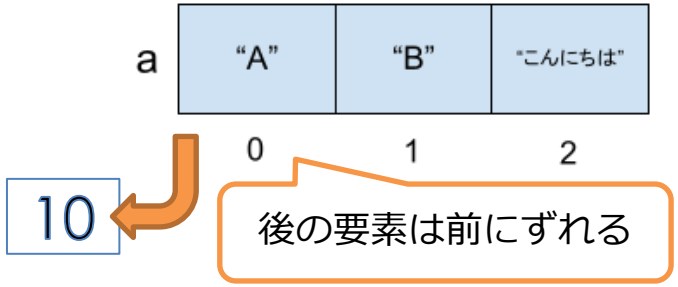
① 空の配列を作り、変数aに代入する

② a.append(要素)で追加する

③ a.insert(位置,要素)で挿入する



④ a.pop(位置)で値を取り出し、削除する



配列を利用・操作するための機能(1)

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.append(10)　# 配列の中身は[1,2,10]になる

配列を利用・操作するための機能(2)

機能名	機能
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。

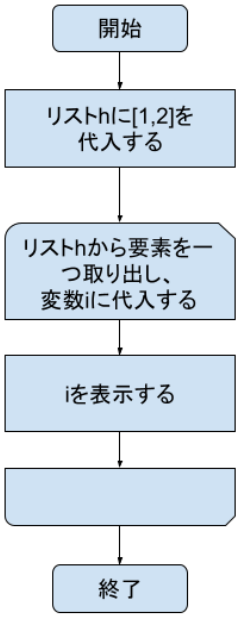
第4章　繰り返し（反復）

for文による繰り返し（反復）

プログラム

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

フローチャート

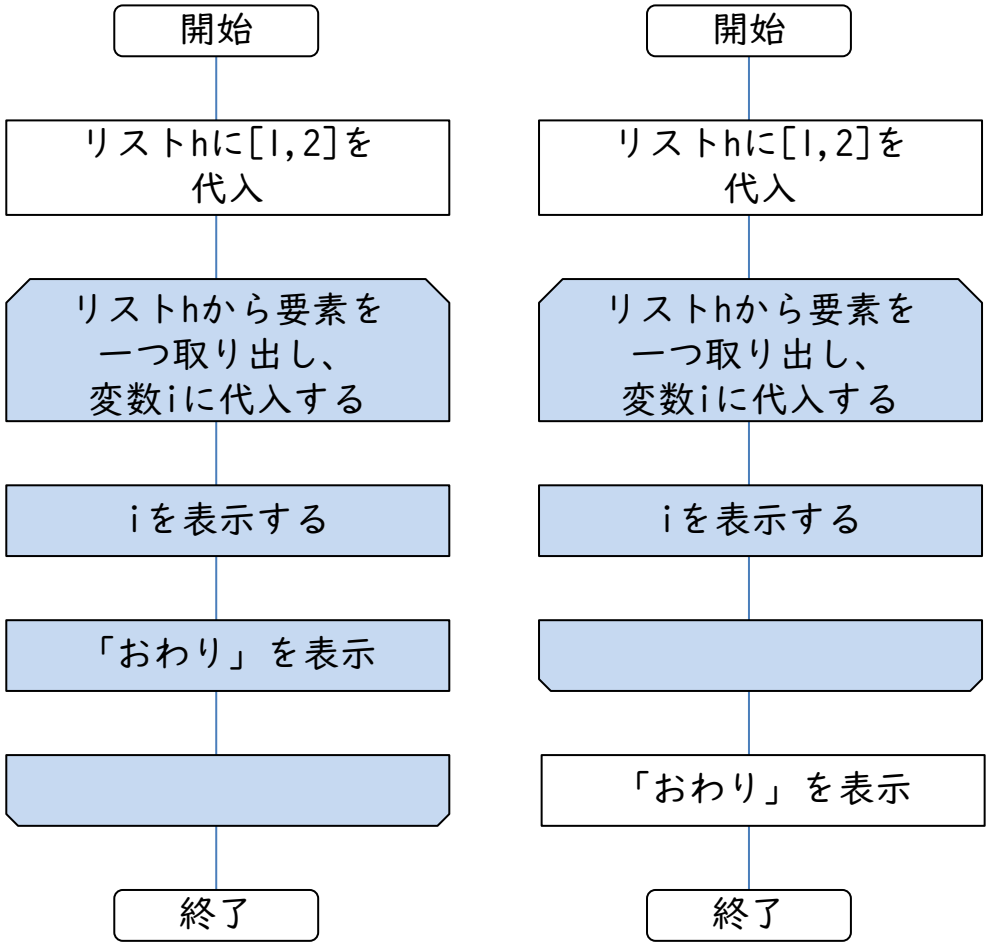


① リストhの先頭から一つ値を取り出して（値は1）、変数iに代入する
② print(i)を実行する。1と表示される
③ リストhから二つ目の値を取り出して（値は2）、変数iに代入する
④ print(i)を実行する。2と表示される

言い換えると

① リストhの先頭から一つ値を取り出して、変数iに代入する
② iを表示する
③ ①と②を、リストhの最後の要素まで繰り返す

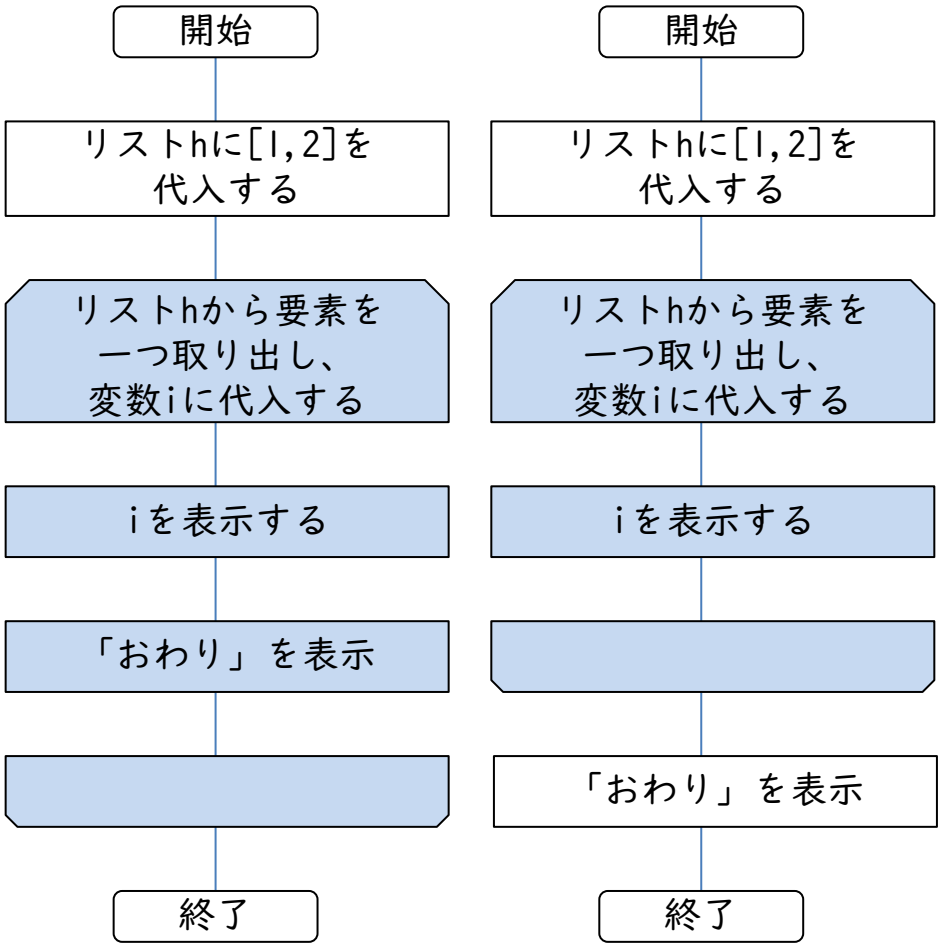
ブロック（フローチャート）



- 左のフローチャート
- 次の2つの処理を繰り返す
  - 変数iを表示する
  - “おわり”を表示する

- 右のフローチャート
- 次の処理を繰り返す
  - 変数iを表示する
- 繰り返しの後に、"終わり"を表示する

ブロック（プログラム）



- **左のフローチャートのプログラム**

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

字下げ
（インデント）

- **右のフローチャートのプログラム**

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

関数range()

- 「範囲」を作る

```
h = [1,2]

のかわりに

h = range(1,3)
```

- この変数hをfor … inに渡す

```
h = range(1,3)
for i in h:
    print(i)
    print("おわり")
```

「1から100まで」のような、大きな範囲を作るときに便利

while文による繰り返し（反復）

- **while文の使用方法**

```
i = 1
while i<3:
        print(i)
        i = i + 1
```

① **変数iに1を代入する**
② **while文で以下を繰り返す。**
- **変数iの値が3より小さいか確認する**
  - ３より小さければ
    - print(i)を実行する
    - 変数iに１を足し、その値を変数iに代入する
  - ３以上なら終了

- **※for文の場合**

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

for文による繰り返し（反復）

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。

範囲（range）（1）

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]

範囲（range）(2)

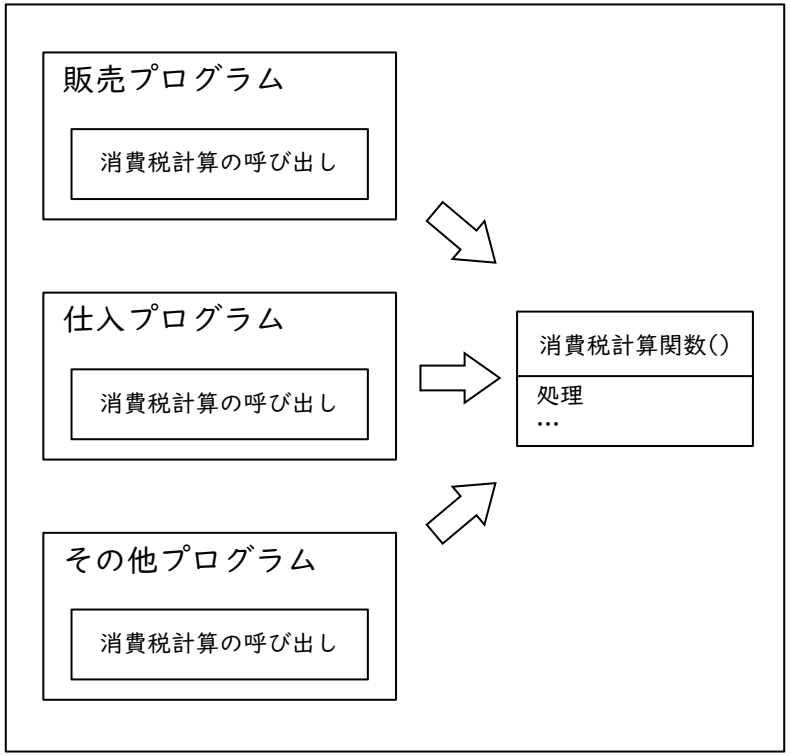
構文	説明
range(開始,終了,ステップ)	三つの整数を渡す。開始の値から、「『終了』より小さい値」の整数の配列を返す。 ステップに指定した数ずつ増える。 例: range(1, 5, 2) -> [1, 3] 「開始」の値を「終了」の値より小さくし、ステップの値をマイナスにすると、徐々に減る範囲が作れる。 例: range(10, 1, -1) -> [10, 9, 8, 7, 6, 5, 4, 3, 2]

while文による繰り返し（反復）

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。

第5章 関数の定義と利用

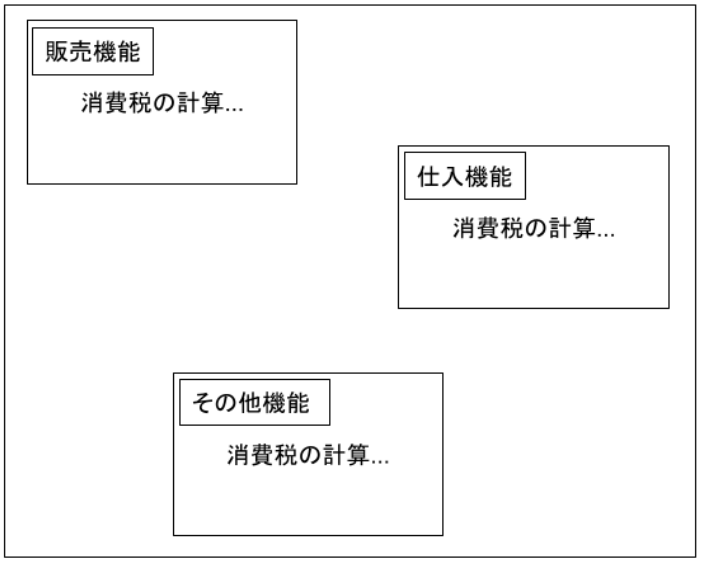
関数の呼び出し



- プログラムの中で、
- たびたび実行させたい機能を
- 関数にまとめておく

- 定義した関数を、
- プログラムの他の場所から
- 呼び出す

関数を使わないと…？



- それぞれの機能ごとに、
- 同じ消費税の計算を書かなければいけない

→ 関数を書くのは一回だけ。
関数を呼び出して使う

- 変更が発生したら（※税率が変わるなど）、
- 全ての消費税計算の処理を書き換えなければいけない

→ 関数を定義して使えば、
書き換えは一つだけ

組み込み関数

- プログラム言語が標準で備えている関数
- 入力（ input() ）、出力（ print() ）
- 乱数（ random.randint() ）
- 範囲を作る（ range() ）
- 日付・時刻を調べる

- ※from … import …が必要なものもある

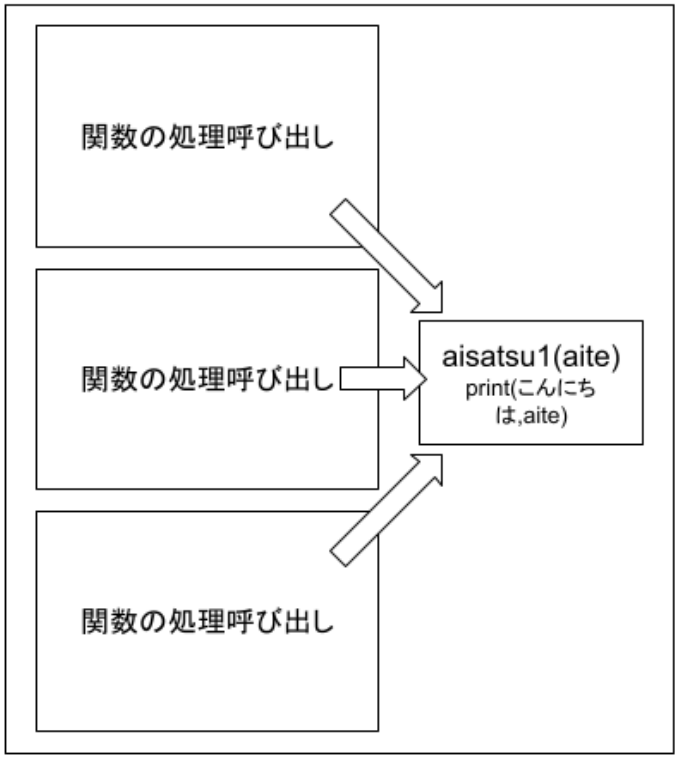
関数の定義

- **関数定義のキーワード:**
  - def
  - 関数名
- **引数**
  - （ ）で囲む
  - 複数の引数を定義するときは”, ”で区切る
- **戻り値**
  - return （値）

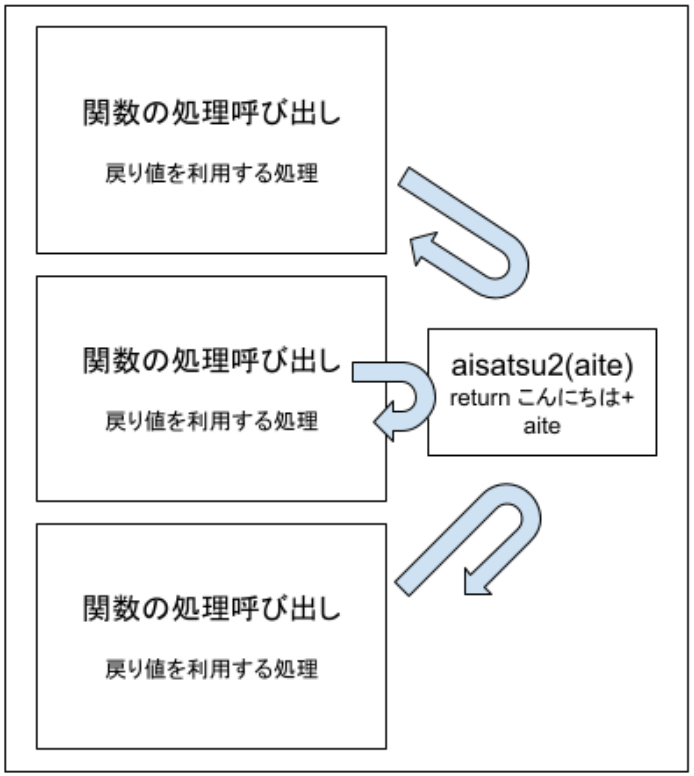
```
def 関数名(引数1, 引数2, ...):
    処理1
    処理2
    return 戻り値
```

関数の戻り値の利用

- 戻り値なし



- 戻り値あり



組み込み関数の例(１) 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。
date.weekday()	使用時の宣言: from datetime import date youbi = date(2021,11,1).weekday() 指定した日付の曜日を数で返す。月曜日が0、火曜日が1、...土曜日が6である。
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。

組み込み関数の例(2)

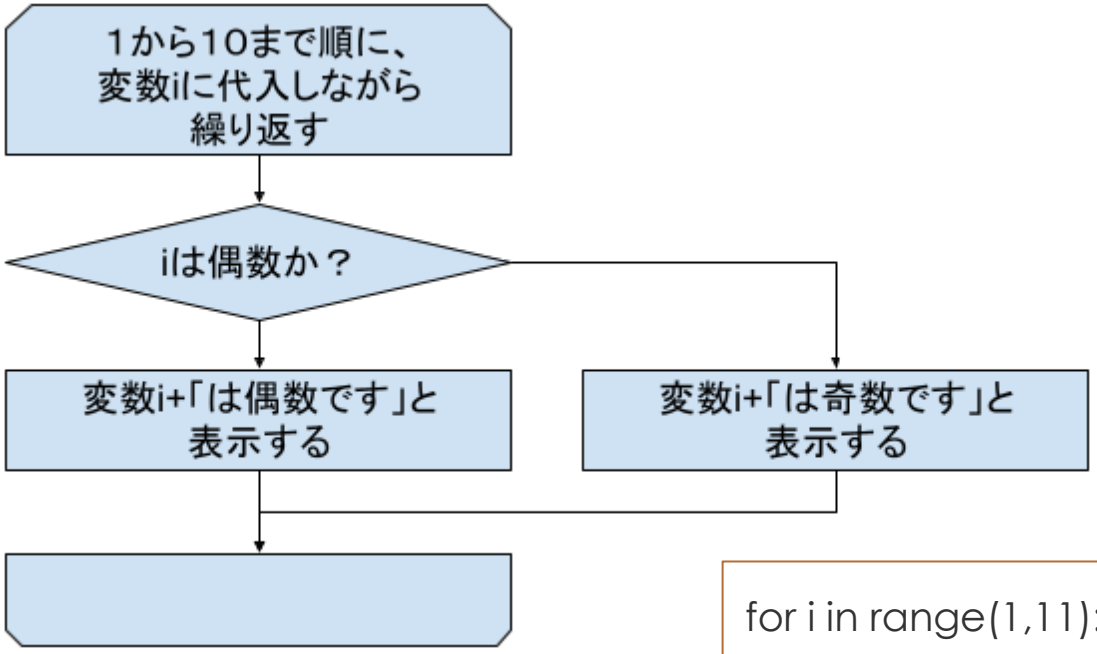
組み込み関数	説明
random.random()	使用時の宣言: import random 0から1の範囲で、ランダムな浮動小数点数を返す。 引数は取らない。
len(リスト)	リストの要素数を調べることができる。 a = [1,2,3] print(len(a)) -> 3
int( )	引数に渡した文字列から、整数のデータを作る。
float( )	引数に渡した文字列から、浮動小数点数のデータを作る。
math.sqrt(値)	引数の値の平方根を返す。square root

関数の定義

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。

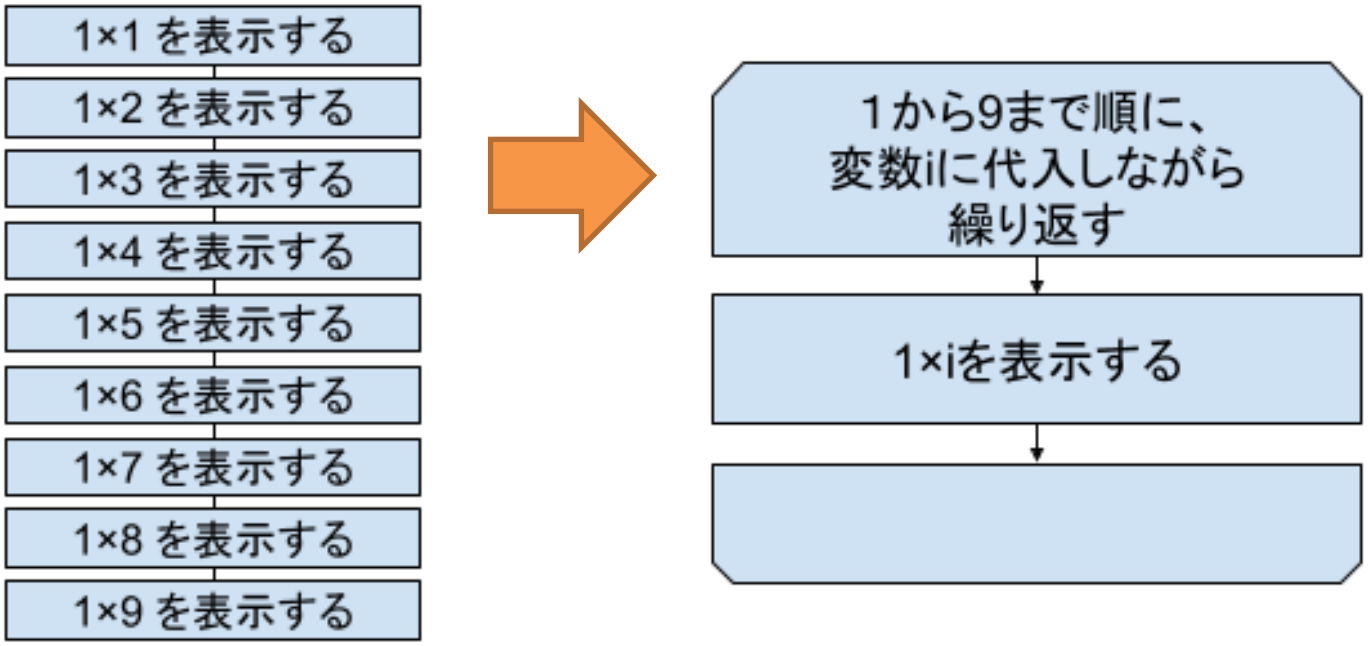
第6章 繰り返し（反復）と選択（分岐）の組み合わせ

繰り返しと選択の組み合わせ

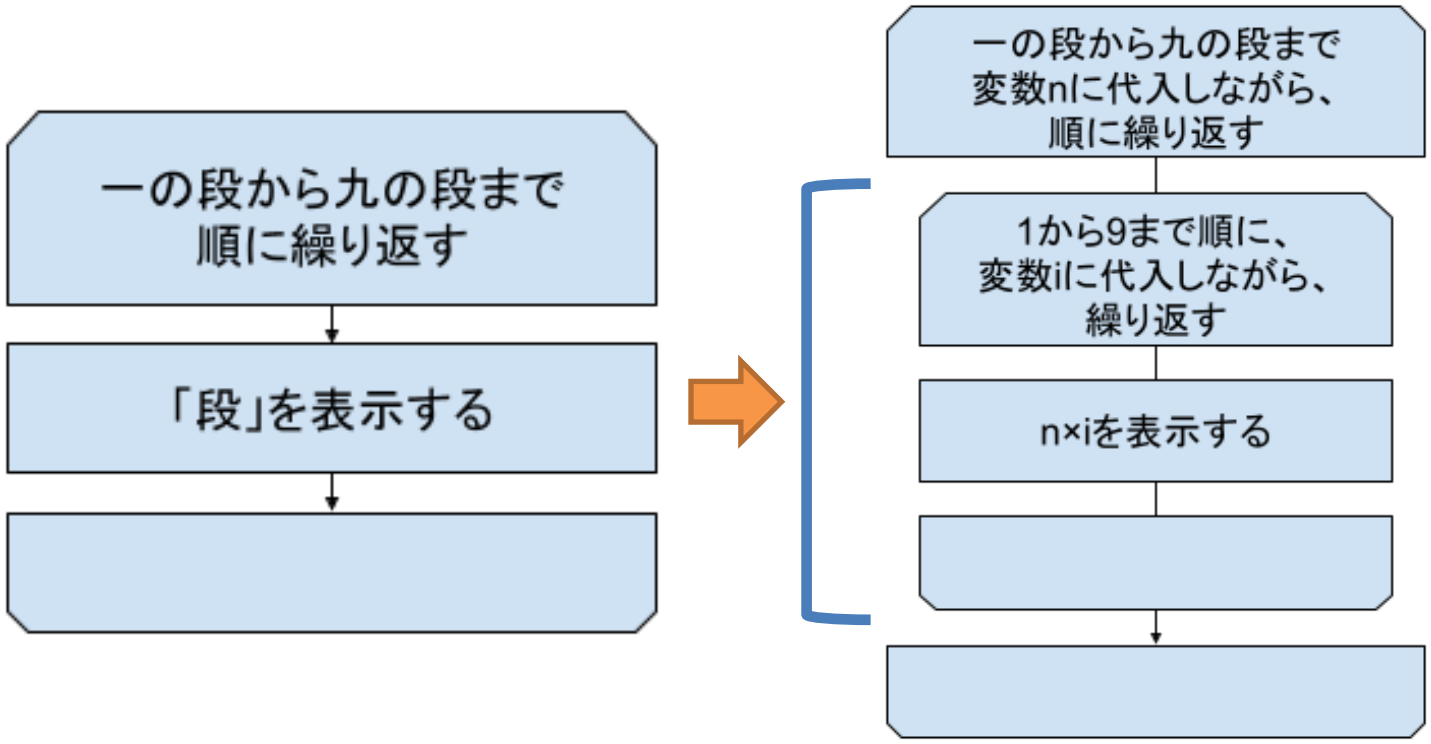



```
for i in range(1,11):
        if i % 2 == 0:
            print(i,"は偶数です")
        else:
            print(i,"は奇数です")
```

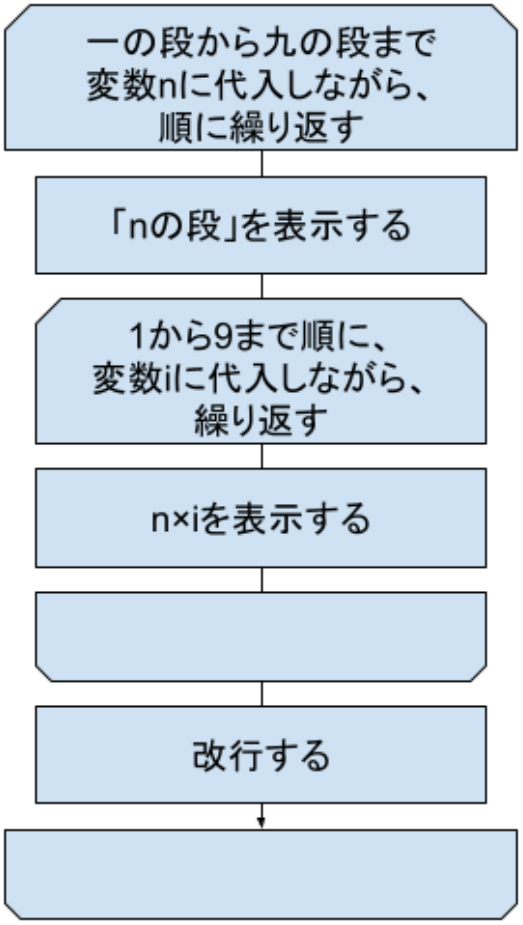
九九を作る：「一の段」の繰り返し



「段」を繰り返す⇨九九にする



二重の繰り返し



- **外側の繰り返し**
  - 「nの段」を表示
  - 内側の繰り返し
  - 改行
- **内側の繰り返し**
  - ×1から×9までの掛け算を表示

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

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。