Python 学習教材

筑波大学 システム情報系 三谷純
最終更新日 2023/4/19

(C) 2023 Jun Mitani　　図表出典：三谷純著『Python ゼロからはじめるプログラミング』（翔泳社 刊）

本資料の位置づけ

本資料は

『Python ゼロからはじめるプログラミング』

を専門学校・大学・企業などで教科書として採用された教員・指導員を対象に、教科書の内容を解説するための副教材として作られています。

上記に該当する場合は、自由にご使用ください。授業の進め方などに応じて、改変していただいて結構です。※ このページを削除して構いません

ただし、民間企業が商用、ビジネス目的で利用する際には別途許諾が必要ですので、著者までご連絡ください。

出版社　：翔泳社
発売日　：2021/5/24
ISBN　：9784798169460

はじめに

プログラミングを学ぶ意義（1/2）

- ソフトウェアを使う立場から、作る立場へ
  - パソコン、スマートフォンなどで動作するアプリ
  - TV、エアコン、洗濯機などの電子機器の制御
  - 電子決済、電子申請、各種のシステム

- 個人での簡単な開発
  - 電卓・Excelの一歩先
  - データ処理・データ解析
  - 個人用途のアプリ開発

- スキルアップ
  - 情報処理関係の資格取得
  - プログラミングコンテスト

プログラミングを学ぶ意義（2/2）

- 実際に開発する立場になる予定が無くても
  - アルゴリズム的思考（ものごとを処理する手順に関する合理的な考え方）が身につく
  - ソフトウェアの開発の様子、動作原理がわかることによる広い視野の獲得
  - 専門用語の理解、ITエンジニアとのコミュニケーション
  - 将来にプログラミングを独習したくなったときに役立つ（異なるプログラミング言語でも考え方の基本は同じ）

この講義で学ぶこと

- プログラミング言語に拠らない、プログラミング全般の基礎知識
- プログラミングに関する基本的な用語と考え方
- Pythonというプログラミング言語活用の基礎

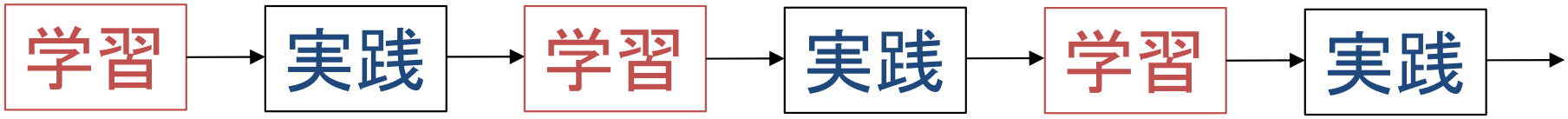
よりよく学ぶために

- 実際に手を動かしてプログラムコードを入力する、一部を変更して実験する

いろいろ試してみよう！

- Webで公開されているプログラムコードを動かしてみる、一部を変更して実験する

- プログラミングコンテストに参加してみる
  （モチベーションアップ、実力向上、他者のコードからの学び）



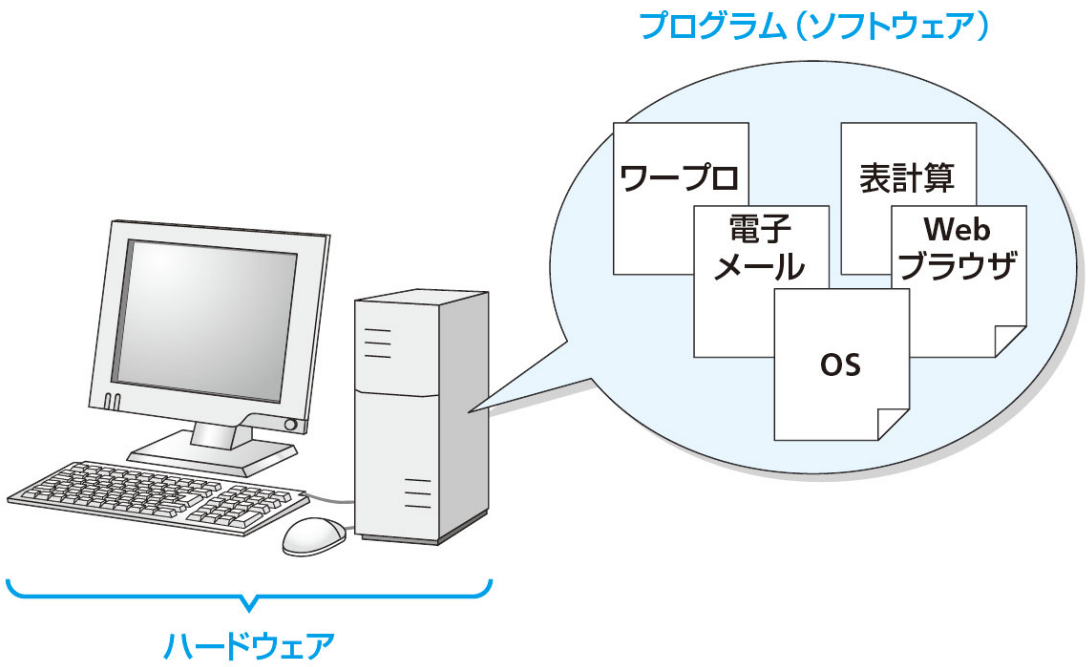
全体の流れ

1. Pythonに触れる
2. Pythonの基本
3. 条件分岐と繰り返し
4. 組み込み型とオブジェクト
5. ユーザー定義関数
6. クラスの基本
7. 発展と応用

第1章 Pythonに触れる

プログラムとは

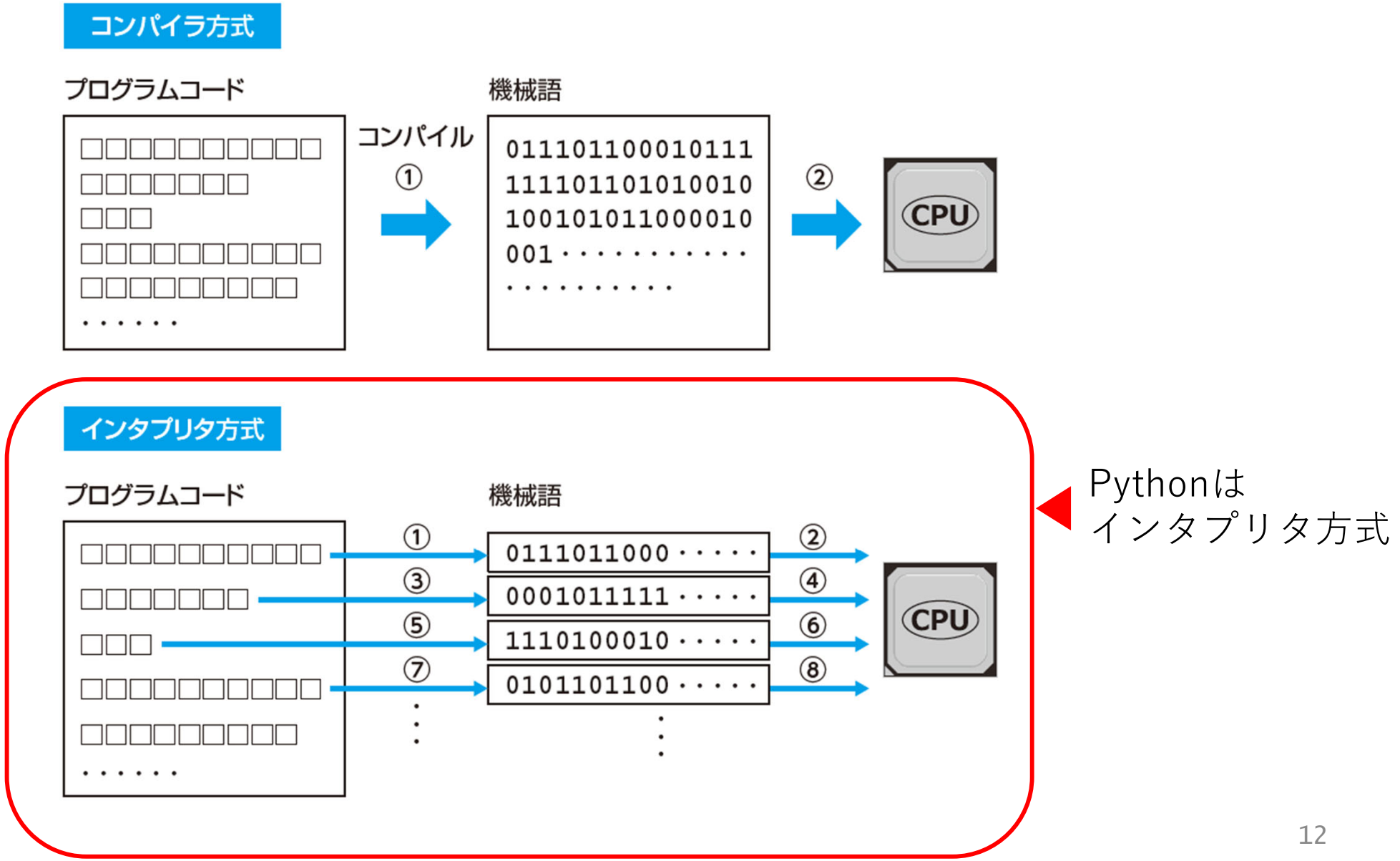
- コンピュータに命令を与えるものが「プログラム」
- プログラムを作成するための専用言語が「プログラミング言語」
- その中の1つに「Python」がある



さまざまなプログラミング言語

- C　歴史のある言語、組み込みプログラム
- C++　C言語の後継、オブジェクト指向
- C#　C++言語の後継、米マイクソロソフト
- Perl　スクリプト言語、手軽な開発
- PHP　サーバサイド、Webページ生成
- Java　オブジェクト指向、大規模システム
- JavaScript　ブラウザで動作、動的なWebページ
- Python　修得が容易、機械学習分野で普及

プログラムコードが実行されるまで



Pythonは
インタプリタ方式

Pythonのプログラムコード

```
total = 0
for i in range(1, 101):
    total += i
print(total)
```

↑
1から100までの整数を順番に足しあげて、その結果を画面に表示するプログラムのプログラムコード

- 半角英数と記号で記述する
- 人が読んで理解できるテキスト形式

2通りの実行方法

1. プログラムコードを1行ずつ与え、そのつど実行させる

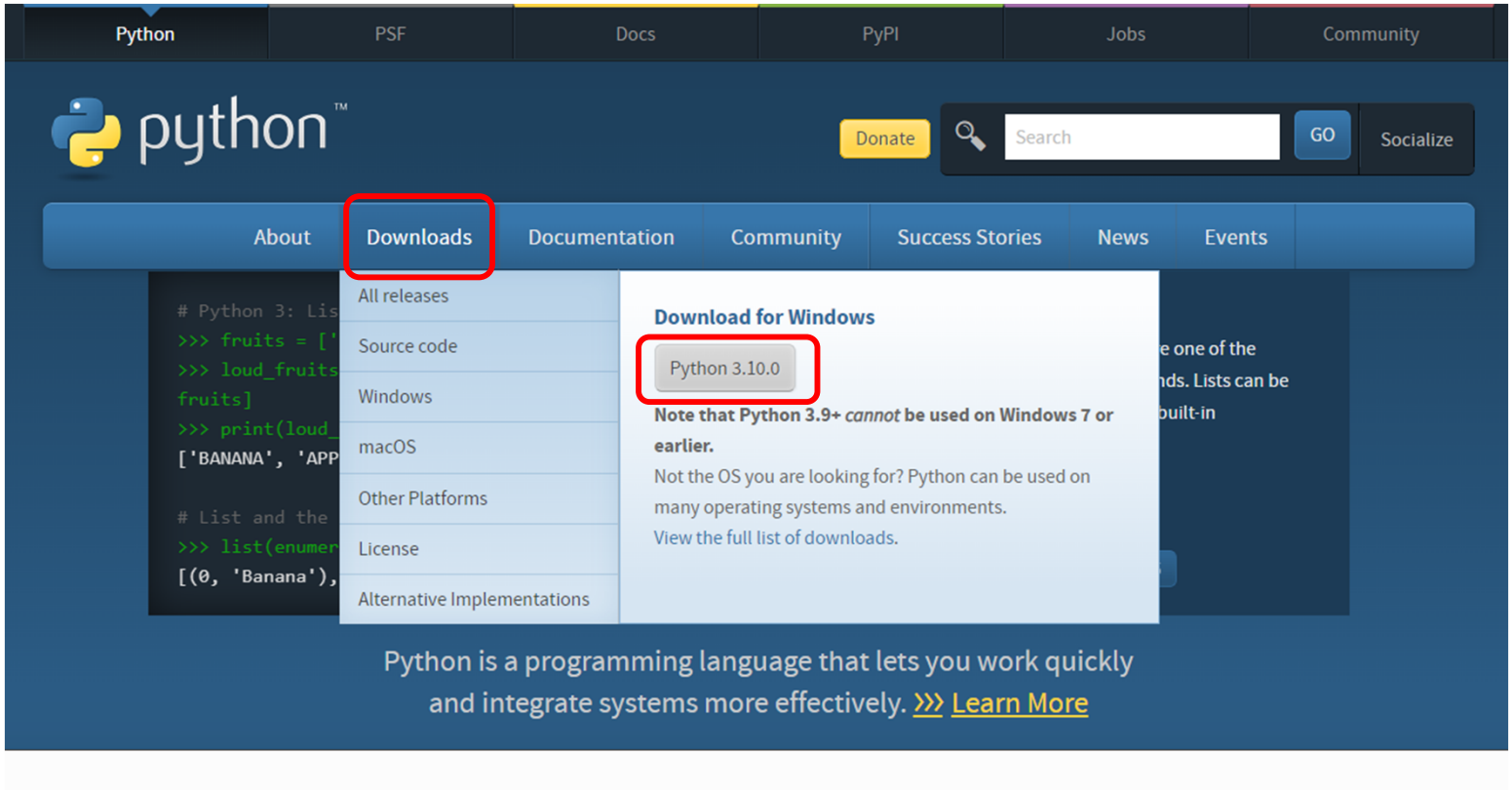
対話モード

2. プログラムコードを記述したファイルを作成し、そのファイルを与えて実行させる

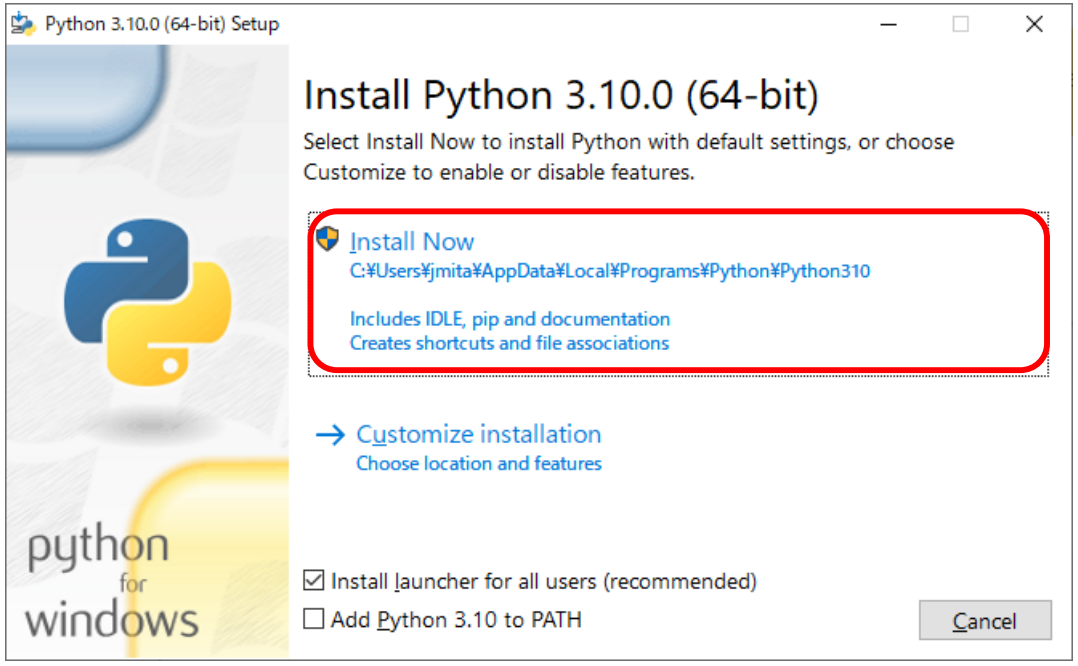
Pythonに触れてみる

Pythonのインストール

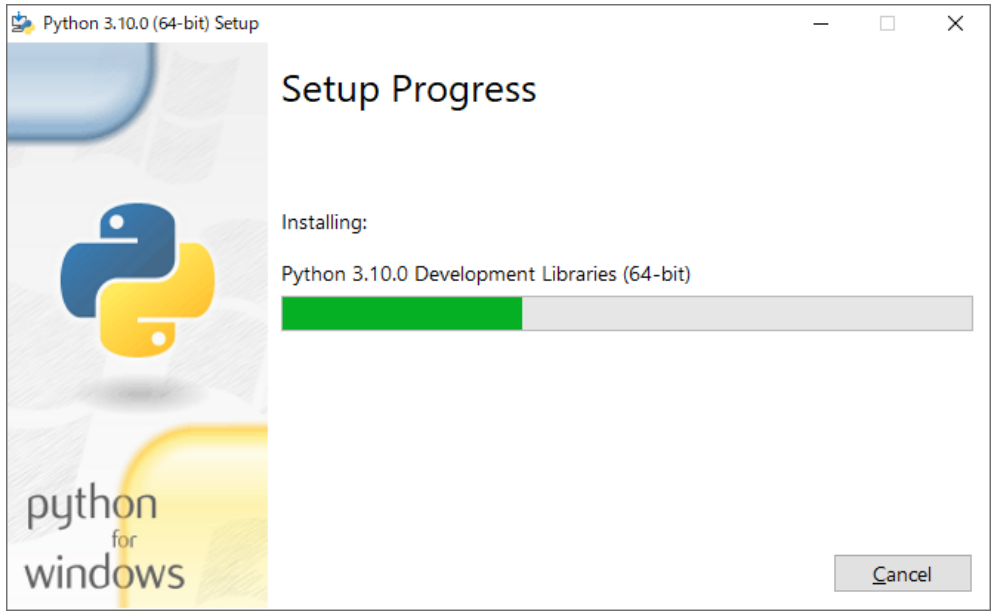
- Pythonの公式ページ
  https://www.python.org/

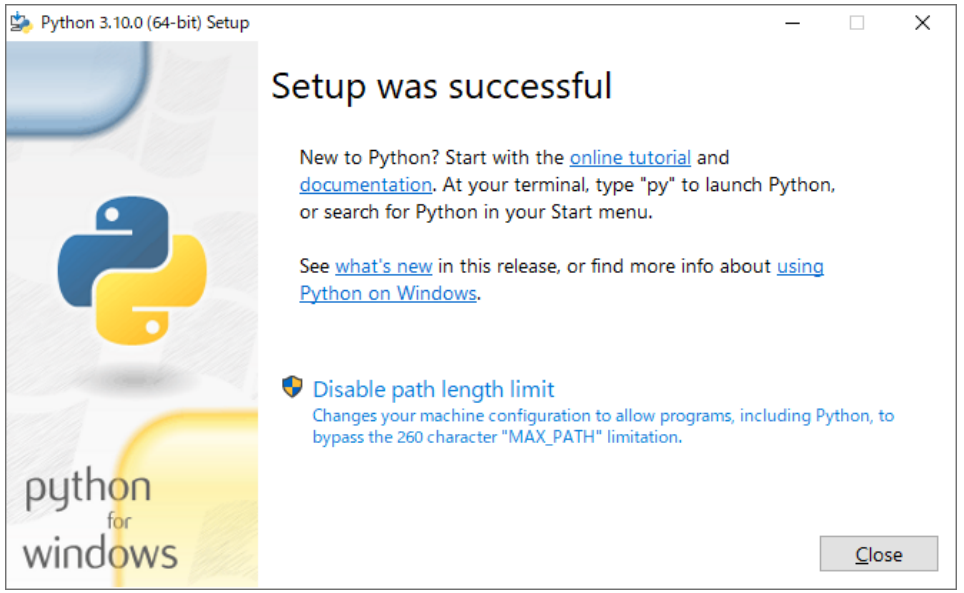


Pythonのインストール



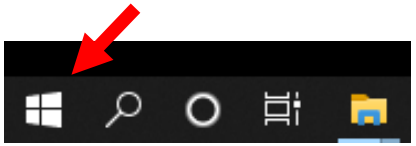
Pythonのインストール

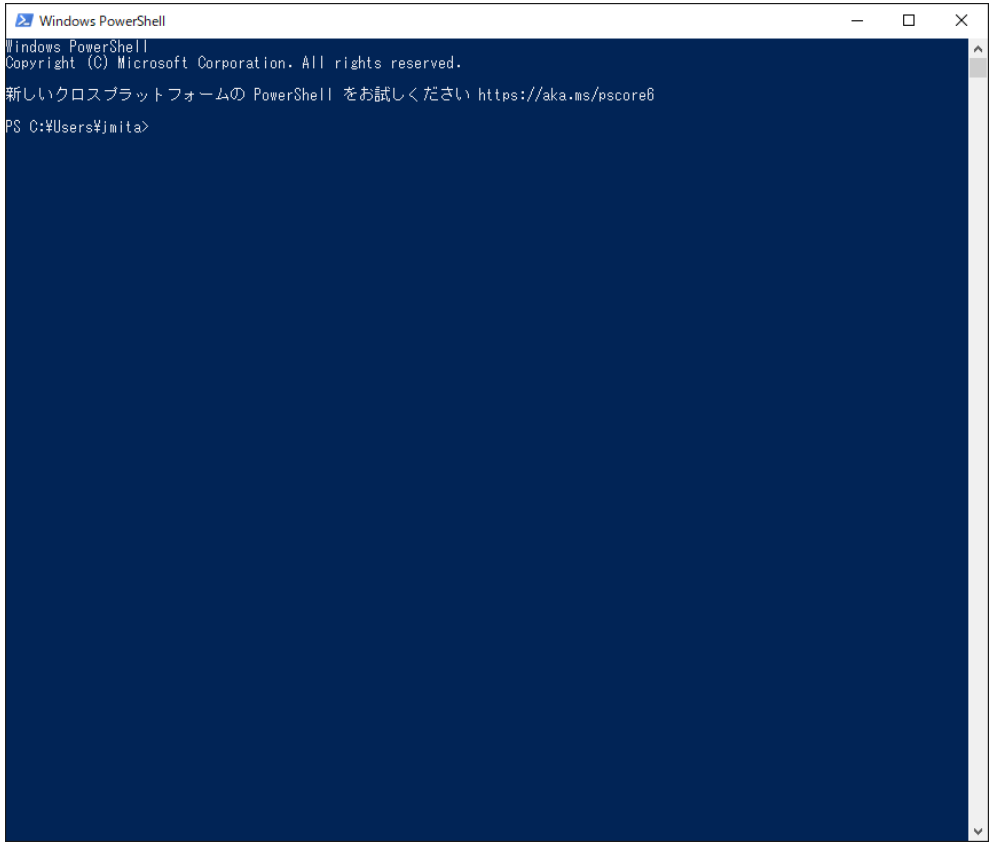




Windows PowerShell の起動

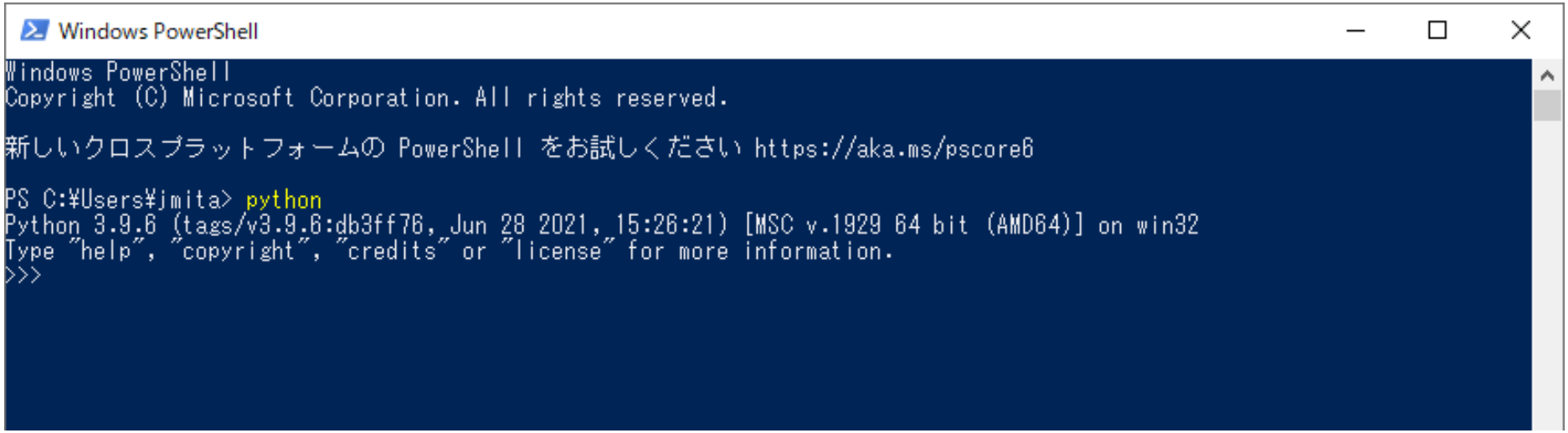
右クリック

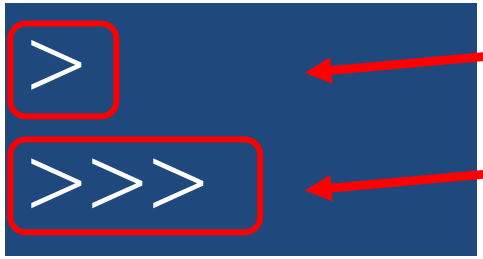




Python の起動

- シェルで「python」と入力



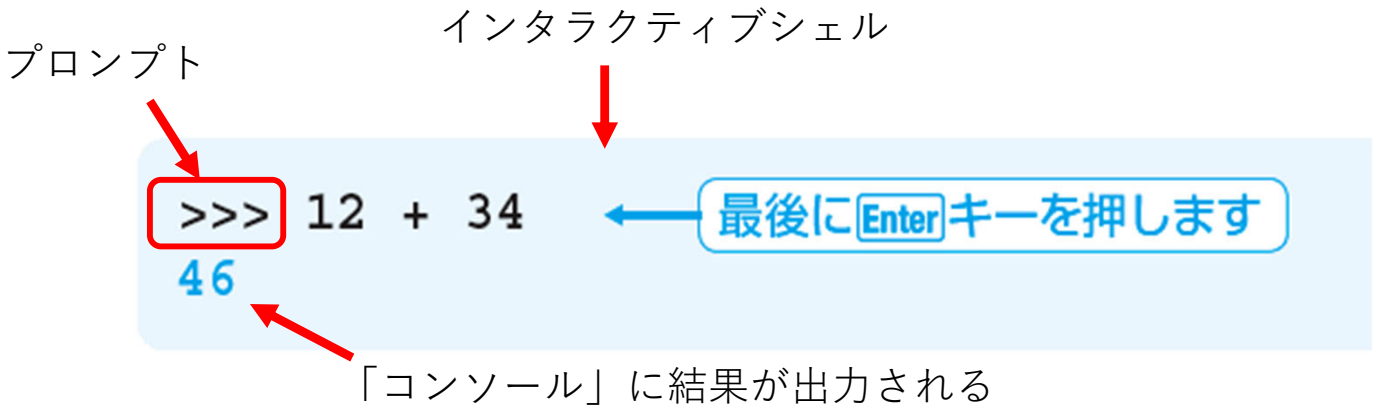


Windows PowerShell のプロンプト

対話モードのPythonのプロンプト

※ プロンプト：命令を待っている状態を示す記号

対話モードを電卓のように使う



掛け算

```
>>> 15 * 10
150
```

割り算

```
>>> 90 / 2
45.0
```

括弧を使った計算

```
>>> 10 * (7 - 2) + 5
55
```

いろいろ試してみよう！

文字列を扱う

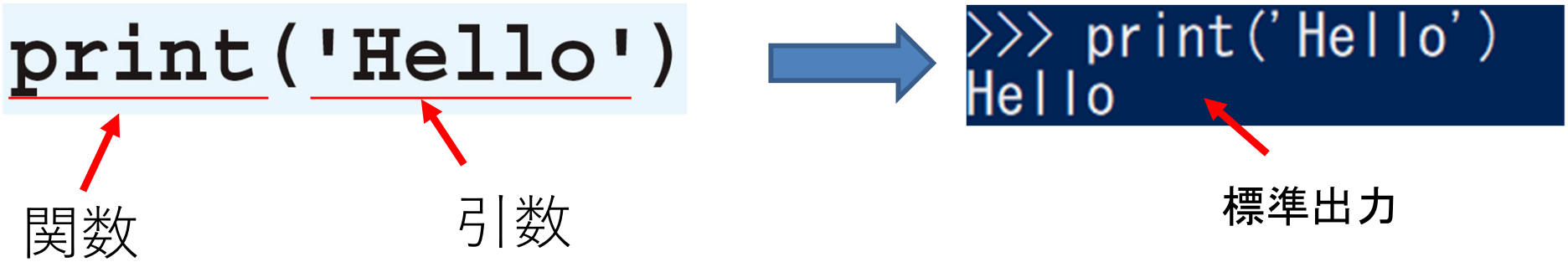
```
>>> print('Hello, Python.')
Hello, Python.
```

`print('出力する内容')`
として、文字列を出力できる。

文字列の前後をシングルクォーテーション（'）、または、
ダブルクォーテーション（"）で囲む

いろいろ試してみよう！

print関数



print 関数は、引数で与えられたものを
標準出力に出力する働きをする組み込み関数

※ print 関数のように、はじめから使える関数を組み込み関数という（自分で作った関数はユーザ定義関数）

※ 標準出力はプログラム実行環境によって異なる

プログラムコードのルール

- 半角英数字と半角記号を使用して記述する（クオーテーションで囲んだ文字列は例外）

- 大文字と小文字は区別される
  - ○ `print('Hello')`
  - × `Print('Hello')`

- 単語や数字、記号の前後には、半角の空白文字を入れても入れなくてもよい
  - ○ `3+4`
  - ○ `3 + 4`

エラー

```
>>> Print('Hello')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'Print' is not defined
```

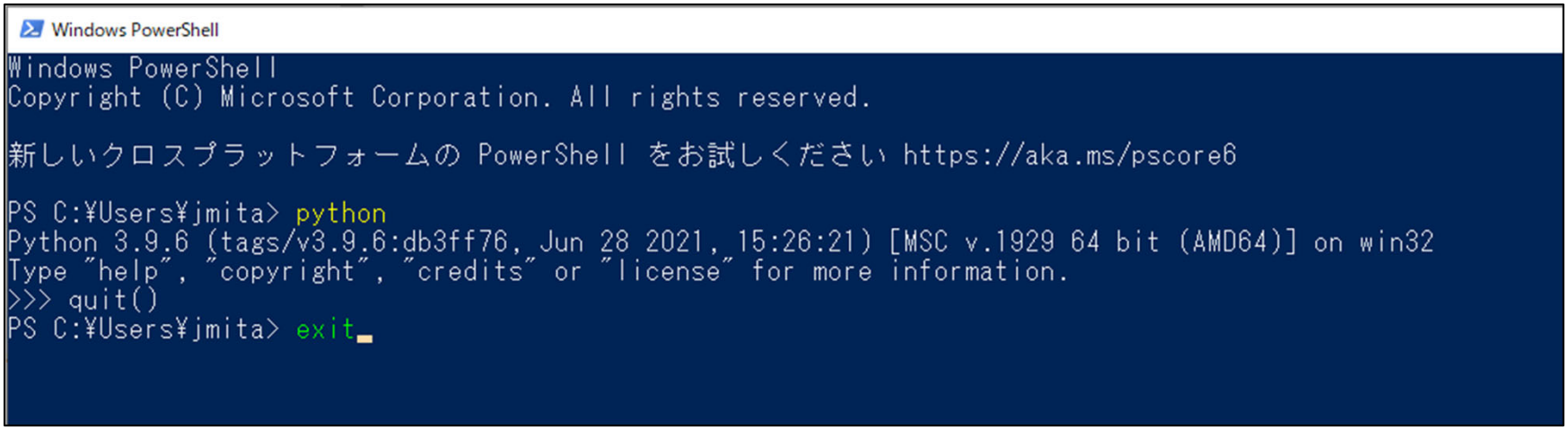
```
>>> print('Hello)
  File "<stdin>", line 1
    print('Hello)
                 ^
SyntaxError: EOL while scanning string literal
```

プログラムコードを適切に実行できない場合にエラーが発生し、エラーメッセージが表示される。

※ エラーメッセージには、発生したエラーに関する説明が表示されるので、がんばって読むようにする。

インタラクティブシェルの終了

`quit()` または `exit()` で終了する。



※ Windows PowerShell を終了するには exit

変数

変数

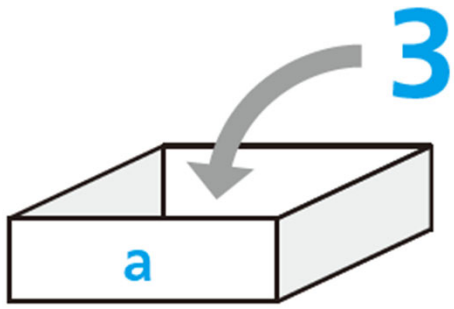
「**変数**」とは、値を入れておく入れ物

```
a = 3
```

←「変数aに3を**代入**する」

変数名

「代入」の一般的なイメージ



a という名前のついた箱に 3 を入れる

代入の正確なイメージ

```
a = 3
```

←「変数aに3を**代入**する」

「3という値を表すオブジェクトがコンピュータのメモリ上のどこかに保管される。その保管場所を示す所在地情報が、aという名前の箱に入れられる」

代入した値を確認する

print 関数で、変数に代入されている値を出力できる。

```
>>> a = 3
>>> print(a)
3
```

変数aに3を代入します

変数aに代入されている値を出力します

※ 変数の値を見ることを「参照する」という

複数の変数の値をいっぺんに出力できる

```
>>> a = 10
>>> b = 123
>>> print(a, b)
10 123
```

print関数に2つの変数を渡しています

それぞれの変数の値が出力されました

インタラクティブシェルでは print 関数を省略できる

```
>>> a = 3
>>> a
3
```

print() を使わずに、変数名だけを記述しています

変数の値が出力されます

いろいろ試してみよう！

3つのキーワード

- 変数
- 代入
- 参照

値を変更する

変数に、あとから別の値を代入できる

```
>>> a = 3
>>> print(a)
3
>>> a = 5
>>> print(a)
5
>>> a = 'hello'
>>> print(a)
hello
```

変数aに3を代入します

変数aに5を代入します

変数aに文字列'hello'を代入します

いろいろ試してみよう！

練習問題

問題 1

次の文章のうち正しいものには○を、正しくないものには×をつけてください。

（1） コンピュータは、Pythonのプログラムコードを直接理解して処理を行う。

（2） Pythonのプログラムコードは、大文字と小文字の違いを区別しない。

（3） Pythonには、1行ずつプログラムコードを入力して、そのつど実行する方法がある。

（4） 「 print(こんにちは)」と記述すると、「こんにちは」という文字列が出力される。

（5） 変数には後から異なる値を代入できる。

問題１（解答）

次の文章のうち正しいものには○を、正しくないものには×をつけてください。

× （1） コンピュータは、Pythonのプログラムコードを直接理解して処理を行う。

× （2） Pythonのプログラムコードは、大文字と小文字の違いを区別しない。

○ （3） Pythonには、1行ずつプログラムコードを入力して、そのつど実行する方法がある。

× （4） 「 print(こんにちは)」と記述すると、「こんにちは」という文字列が出力される。

○ （5） 変数には後から異なる値を代入できる。

問題 2

次の文章の空欄に入れるべき語句を、選択肢から選んでください。

コンピュータが値を記憶しておくための入れ物のことを[（1）]という。
[（1）] に値を格納することを[（2）]という。
[（2）] を行うには、記号[（3）]を使用する。
[（1）]に[（2）]された値は print 関数を用いてコンソールに[（4）] できる。

【選択肢】

・代入　・変数　・オブジェクト　・出力
・>>>　・＝

問題 2（解答）

次の文章の空欄に入れるべき語句を、選択肢から選んでください。

コンピュータが値を記憶しておくための入れ物のことを[変数]という。
[変数] に値を格納することを[代入]という。
[代入] を行うには、記号[=]を使用する。
[変数]に[代入]された値は print 関数を用いてコンソールに[出力] できる。

【選択肢】

・代入　・変数　・オブジェクト　・出力
・>>>　・＝

問題 3

対話モードで、次の計算を実行して結果を確認しましょう。

(1)　1 + 2 + 3 + 4
(2)　2 + 3 * 2
(3)　(2 + 3) * 2
(4)　10 / 2.5
(5)　3 / 0

問題３（解答）

対話モードで、次の計算を実行して結果を確認しましょう。

(1)　1 + 2 + 3 + 4
(2)　2 + 3 * 2
(3)　(2 + 3) * 2
(4)　10 / 2.5
(5)　3 / 0

```
>>> 1+2+3+4
10
>>> 2+3*2
8
>>> (2+3)*2
10
>>> 10/2.5
4.0
>>> 3/0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>>
```

問題 4

次のようにして、インタラクティブシェルで変数aに10という値を代入し、print関数で値を出力できます。

```
>>> a = 10
>>> print(a)
10
```

（1） 変数bに5という値を代入してから、print関数で変数bに代入された値を出力してください。

（2） 変数cに「Python」という文字列を代入してから、print関数で変数cに代入された値を出力してください。

問題４（解答）

次のようにして、インタラクティブシェルで変数aに10という値を代入し、print関数で値を出力できます。

```
>>> a = 10
>>> print(a)
10
```

（1） 変数bに5という値を代入してから、print関数で変数bに代入された値を出力してください。

```
>>> b = 5
>>> print(b)
5
```

（2） 変数cに「Python」という文字列を代入してから、print関数で変数cに代入された値を出力してください。

```
>>> c = 'Python'
>>> print(c)
Python
```

第2章 Pythonの基本

型と算術演算

型

- データの種類のことを「型」とよぶ

はじめから準備されている型「組み込み型」

型	型名（日本語表記）	値の例
`int`	整数型	−1, 0, 1, 2, 10, 100
`float`	浮動小数点数型	小数点を含む数 −0.12, 0.0, 0.5, 2.34
`str`	文字列型	'hello', 'こんにちは'
`bool`	真偽値型	True, False

type関数による型の確認

type(値) または type(変数名) で型を確認できる

```
>>> type(10)
<class 'int'>
>>> type(0.5)
<class 'float'>
>>> type('こんにちは')
<class 'str'>
>>> type(True)
<class 'bool'>
```

10はint型であることがわかります

0.5はfloat型であることがわかります

'こんにちは'はstr型であることがわかります

Trueはbool型であることがわかります

```
>>> a = 10
>>> type(a)
<class 'int'>
>>> b = 'こんにちは'
>>> type(b)
<class 'str'>
```

変数aに10を代入します

aの値 (注❷-5) はint型であることがわかります

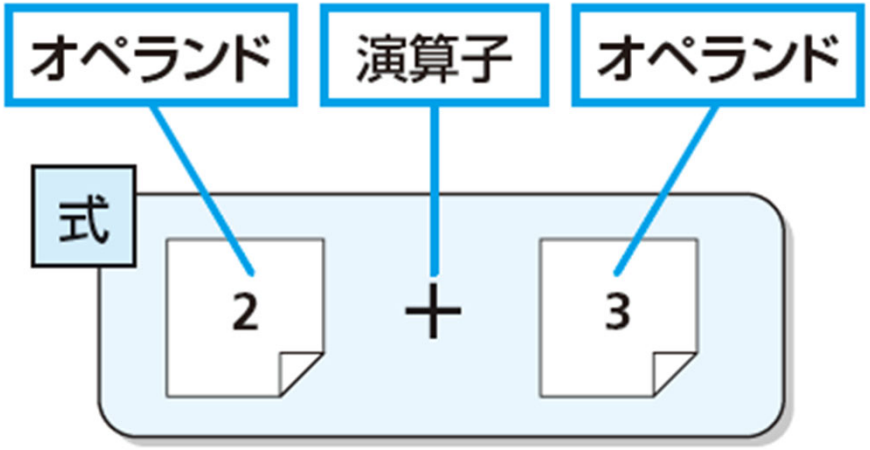
変数bに'こんにちは'という文字列を代入します

bの値はstr型であることがわかります

いろいろ試してみよう！

算術演算

- 覚えるべき用語



式の値は5

文(代入文)

```
a = 2 + 3
```

リテラル
(プログラムコード中に記述される数値)

演算子の種類

演算子	演算の内容
+	加算(足し算)
-	減算(引き算)
*	乗算(掛け算)
/	除算(割り算)
//	商
%	剰余
**	べき乗
:=	代入

変数を含む算術演算

式に変数名が含まれる場合は、
変数に代入されている値が使用される

```
>>> a = 10
>>> print(a + 3)
13
```

変数aに10を代入します

式a + 3の値を出力します

```
b = a + 3
```

(変数aの値)+3 が変数bに代入される

```
a = a + 3
```

(変数aの値)+3 が変数aに代入される
つまりaの値が3増える

算術演算の短縮表現

```
a = a + 3
```

短縮

```
a += 3
```

加算代入

演算子	使用例	説明
+=	a += b	a = a + b と同じ
-=	a -= b	a = a - b と同じ
*=	a *= b	a = a * b と同じ
/=	a /= b	a = a / b と同じ
%=	a %= b	a = a % b と同じ
//=	a //= b	a = a // b と同じ
**=	a **= b	a = a ** b と同じ

いろいろ試してみよう！

数値の型

プログラムコードへの記述のしかたで、型が異なる。
・小数点を含まない → int型
・小数点を含む → float型

```
>>> type(0)
<class 'int'>
>>> type(0.0)
<class 'float'>
```

0という表記はint型になります

0.0という表記はfloat型になります

いろいろ試してみよう！

演算と数値の型

int型どうしの加算・減算・乗算 → int型
Int型どうしの除算 → float型
float型を含む演算 → float型

```
>>> type(3 + 2)
<class 'int'>
>>> type(3 - 2)
<class 'int'>
>>> type(3 * 2)
<class 'int'>
>>> type(3 / 2)
<class 'float'>
>>> type(4 / 2)
<class 'float'>
```

int型どうしを加算、減算、乗算した結果はint型です

int型どうしで除算した結果はfloat型になります

割り切れる場合でもfloat型になります

いろいろ試してみよう！

数値の型変換

- int 型 → float 型

```
a = float(100)
```

← 変数aは float 型になる

- float 型 → int 型

```
a = int(9.6)
```

← 変数aは int 型になる
小数点以下は切り捨て

いろいろ試してみよう！

文字列とリストの扱い

文字列の扱い

- +演算子による文字列の連結

```
a = 'AAA' + 'BBB'
```

```
a = 'AAA'
b = 'BBB'
c = a + b
```

- *演算子による連結の繰り返し

```
a = 'ABC' * 3
```

← 'ABCABCABC'になる

いろいろ試してみよう！

数値→文字列の変換

数値を文字列のようには扱えない

✕
```
a = 500 + '円'
```

str(数値)で文字列に変換する

○
```
a = str(500) + '円'
```

```
>>> year = 2021
>>> print(str(year) + '年')
2021年
```

yearの値（int型）を文字列に変換してから連結しています

いろいろ試してみよう！

変数の値の埋め込み

数値を文字列に変換してから連結

```
>>> price = 550
>>> print('この商品は' + str(price) + '円です')
この商品は550円です
```

フォーマット文字列の使用して簡潔に記述できる

```
>>> price = 550
>>> print(f'この商品は{price}円です')
この商品は550円です
```

f'文字列' とすると、文字列に含まれる **{変数名}** 部分が変数の値に置き換わる

いろいろ試してみよう！

フォーマット文字列の活用

フォーマット文字列

f'文字列' とすると、文字列に含まれる **{変数名}** 部分が変数の値に置き換わる

{変数名} 部分に**式**を入れることもできる

```
a = 5
b = 550
print(f'1つ{a}円です。{b}個で{a * b}円です')
```

いろいろ試してみよう！

文字列→数値の変換

文字列を数値のようには扱えない

```
a = '500'
b = a * 2
```

← 文字列

← bの値は'500500'になる

int(文字列)で整数に変換する

```
a = '500'
b = int(a) * 2
```

← bの値は1000になる

※ 小数点を含む数値に変換するときは float(文字列)

いろいろ試してみよう！

len関数による文字列の長さの取得

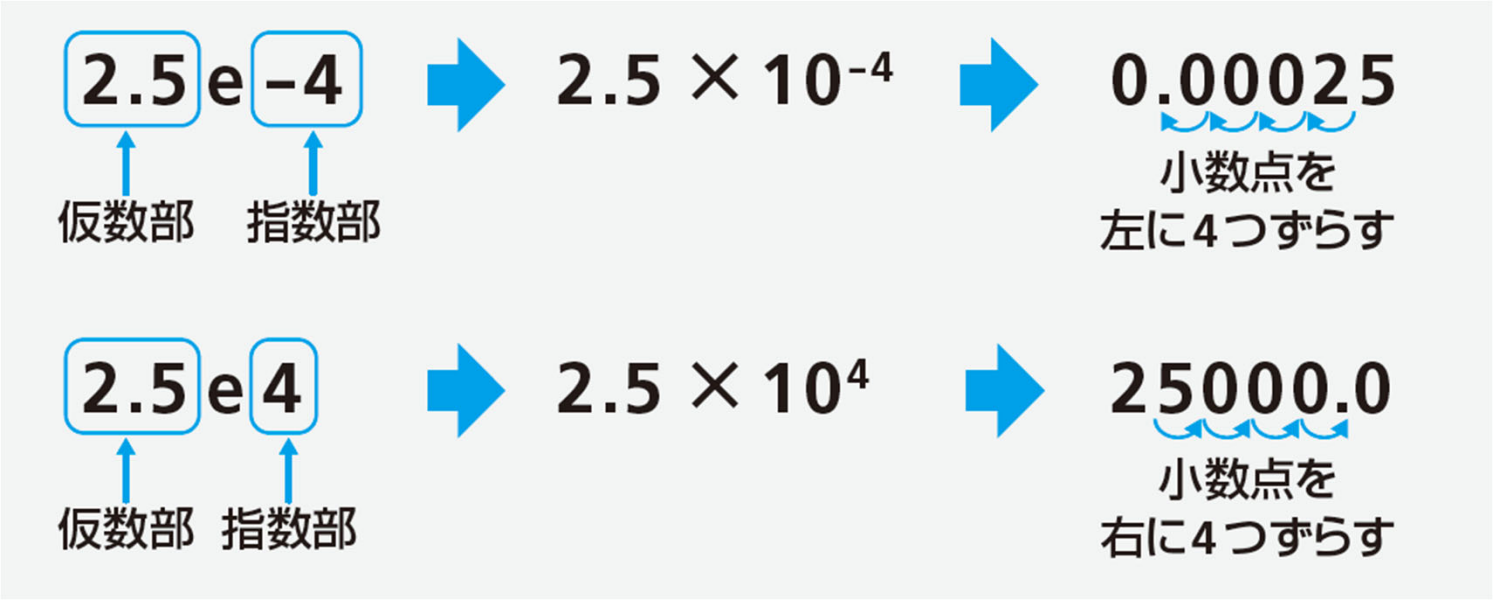
len(文字列)　で文字列の長さ（文字数）を取得できる

```
>>> len('Hello')
5
```

```
>>> a = 'Python'
>>> len(a)
6
```

いろいろ試してみよう！

数値の指数表現




```
>>> print(2.5e-4)
0.00025
>>> print(2.5e4)
25000.0
```

リスト

リストを使って、複数の値をまとめて管理できる

```
a = [10, 20, 30, 40, 50]
```

リストに格納された**要素**には、**インデックス**を使ってアクセスする

```
>>> print(a[0])   ← 先頭の要素
10
>>> print(a[1])   ← 2番目の要素
20
>>> print(a[4])   ← 5番目の要素
50
```

※ インデックスは0から始まる

いろいろ試してみよう！

マイナスのインデックス

```
a = [10, 20, 30, 40, 50]
```

インデックスにマイナスの値も使える

```
>>> print(a[-1])
50
>>> print(a[-2])
40
>>> print(a[-5])
10
```

← 末尾の要素

← 後ろから2番目の要素

← 後ろから5番目の要素

いろいろ試してみよう！

リスト内の値の変更

インデックスを指定して値を変更できる

```
>>> a = [10, 20, 30, 40, 50]
>>> print(a)
[10, 20, 30, 40, 50]

>>> a[0] = 99
>>> print(a)
[99, 20, 30, 40, 50]

>>> a[-1] = 'A'
>>> print(a)
[99, 20, 30, 40, 'A']
```

← リストの全要素を出力

← 先頭の要素を99に変更

← 文字列にもできる

いろいろ試してみよう！

リストの要素数の確認

len関数でリストの要素数を取得できる

```
>>> a = [10, 20, 30, 40, 50]
>>> len(a)
5
```

いろいろ試してみよう！

モジュールの利用

モジュールの使用

- モジュールとは各種の機能を管理する単位
- 必要に応じてモジュールを読み込んでプログラムを作る
  （「モジュールを**インポートする**」という）

```
import モジュール名
```

- **math**モジュールをインポートすると、sin, cos などの関数を使用できるようになる

```
import math
```

mathモジュールに含まれる関数・定数

関数	説明
ceil(x)	xの値以上の最小の整数を返す
cos(x)	xの余弦（コサイン）を返す。xの単位はラジアン
floor(x)	xの値以下の最大の整数を返す
exp(x)	e（自然対数の底。ネイピア数）のx乗を返す
log(x)	xの自然対数を返す
sqrt(x)	xの平方根を返す
sin(x)	xの正弦（サイン）を返す。xの単位はラジアン
tan(x)	xの正接（タンジェント）を返す。xの単位はラジアン
radians(x)	角度xをラジアンに変換した値を返す
atan(x)	xの逆正接（アークタンジェント）を返す

定数名	値
pi	円周率　3.141592653589793
e	自然対数の底　2.718281828459045

mathモジュールの利用

mathモジュールのインポート

```
import math
```

mathモジュールに含まれる関数の利用

```
math.関数名(引数)
```

```
>>> import math
>>> print(math.sqrt(2))
1.4142135623730951
>>> print(math.floor(12.345))
12
```

← mathモジュールをインポートします

← sqrt関数で2の平方根の計算をします

← floor関数で小数点以下を切り捨てます

いろいろ試してみよう！

mathモジュールの利用

mathモジュールに含まれる定数の利用

```
math.定数名
```

```
>>> import math
>>> print(math.pi)
3.141592653589793
```

いろいろ試してみよう！

randomモジュールの利用

- random モジュールに含まれる関数

関数	説明
`random()`	0以上1未満の浮動小数点数を返す
`randrange(x)`	0から(x-1)までの整数を返す
`choice(list)`	listからランダムに1つ選んだ要素を返す
`randint(a, b)`	a以上b以下のランダムな整数を返す

```
>>> import random
>>> random.randint(1, 6)
4
```

← 実行のたびに異なる値が出力される

randomモジュールの利用

```
>>> import random
>>> janken = ['グー', 'チョキ', 'パー']
>>> random.choice(janken)
'グー'
```

すでにrandomモジュールをインポートした後であれば記述は不要です

3つの要素を持つリストです

リストjankenに含まれる要素からランダムに1つ選びます

毎回結果が異なります

いろいろ試してみよう！

モジュールに別名をつけて使う

```
import モジュール名 as 別名
```

```
>>> import math as m
>>> print(m.pi)
3.141592653589793
```

← mathモジュールの別名をmとする

← 別名を使って記述

モジュール名が長いときに便利

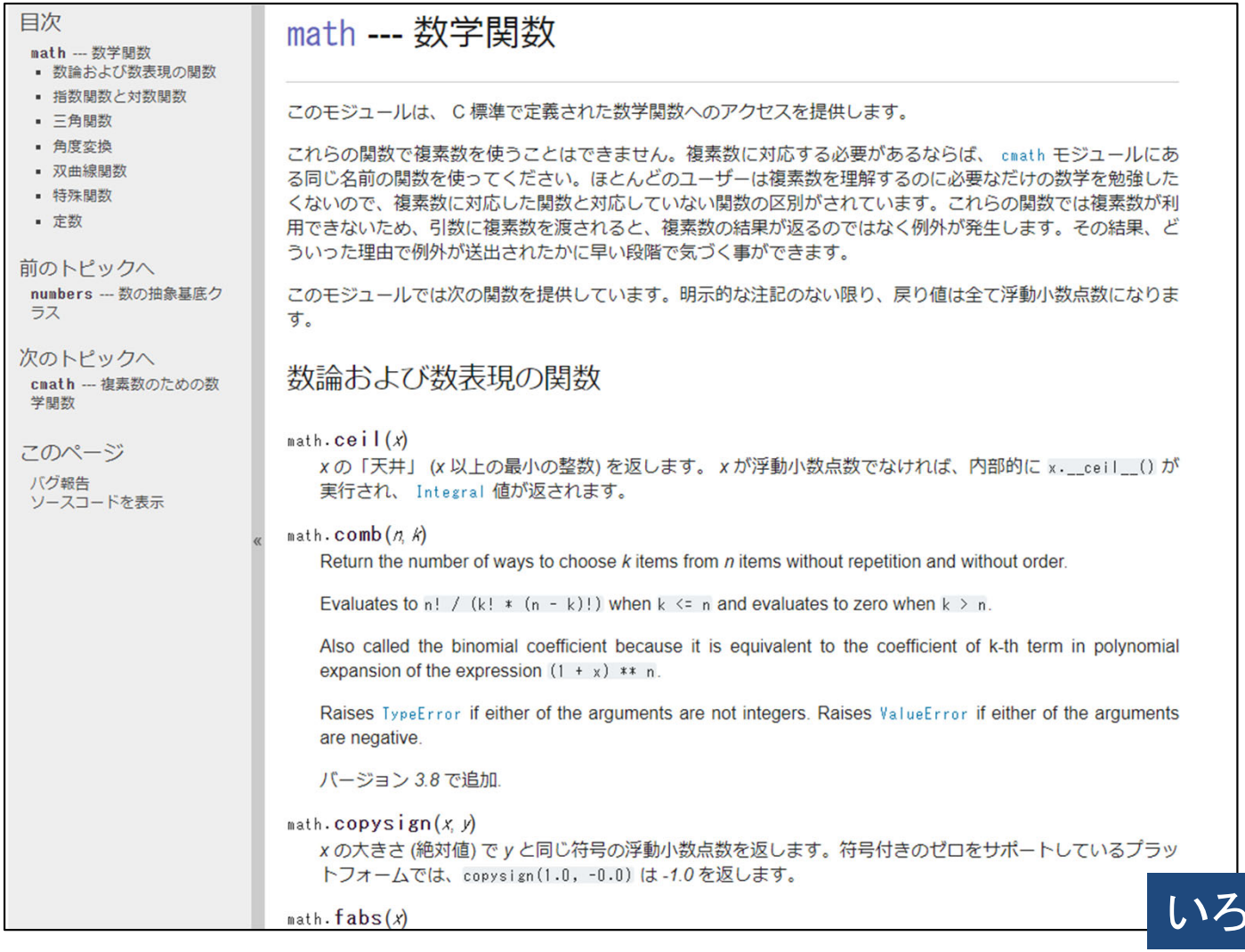
ドキュメントを読む

- Pythonの標準ライブラリのドキュメント

https://docs.python.org/ja/3/library/index.html

モジュールに含まれる関数を調べる

「標準ライブラリ」のページ右上の「モジュール」のリンクから「math」を探してみよう。
ブラウザの「検索」機能も使ってみよう（[Ctrl]+[F]）



目次

math --- 数学関数
- 数論および数表現の関数
- 指数関数と対数関数
- 三角関数
- 角度変換
- 双曲線関数
- 特殊関数
- 定数

前のトピックへ
numbers --- 数の抽象基底クラス

次のトピックへ
cmath --- 複素数のための数学関数

このページ
バグ報告
ソースコードを表示

math --- 数学関数

このモジュールは、 C 標準で定義された数学関数へのアクセスを提供します。

これらの関数で複素数を使うことはできません。複素数に対応する必要があるならば、 cmath モジュールにある同じ名前の関数を使ってください。ほとんどのユーザーは複素数を理解するのに必要なだけの数学を勉強したくないので、複素数に対応した関数と対応していない関数の区別がされています。これらの関数では複素数が利用できないため、引数に複素数を渡されると、複素数の結果が返るのではなく例外が発生します。その結果、どういった理由で例外が送出されたかに早い段階で気づく事ができます。

このモジュールでは次の関数を提供しています。明示的な注記のない限り、戻り値は全て浮動小数点数になります。

数論および数表現の関数

math.ceil(*x*)
x の「天井」 (*x* 以上の最小の整数) を返します。 *x* が浮動小数点数でなければ、内部的に x.__ceil__() が実行され、 Integral 値が返されます。

math.comb(*n*, *k*)
Return the number of ways to choose *k* items from *n* items without repetition and without order.

Evaluates to n! / (k! * (n - k)!) when k <= n and evaluates to zero when k > n.

Also called the binomial coefficient because it is equivalent to the coefficient of k-th term in polynomial expansion of the expression (1 + x) ** n.

Raises TypeError if either of the arguments are not integers. Raises ValueError if either of the arguments are negative.

バージョン *3.8* で追加.

math.copysign(*x*, *y*)
x の大きさ (絶対値) で *y* と同じ符号の浮動小数点数を返します。符号付きのゼロをサポートしているプラットフォームでは、copysign(1.0, -0.0) は *-1.0* を返します。

math.fabs(*x*)

いろいろ試してみよう！

練習問題

問題 1

以下の記述について、正しいものには○を、誤りのあるものには×をつけてください。

（1） 一度int型の値を代入した変数aに対して、後から文字列を代入することはできない。

（2） int型の値とfloat型の値を加算するときには、その前にint型の値をfloat型に型変換しておく必要がある。

（3） int型とfloat型の値を含む算術演算の結果はfloat型になる。

（4） a = int(3.8)と記述した場合、変数aの値は4になる。

問題１（解答）

以下の記述について、正しいものには○を、誤りのあるものには×をつけてください。

× （1） 一度int型の値を代入した変数aに対して、後から文字列を代入することはできない。

× （2） int型の値とfloat型の値を加算するときには、その前にint型の値をfloat型に型変換しておく必要がある。

○ （3） int型とfloat型の値を含む算術演算の結果はfloat型になる。

× （4） a = int(3.8)と記述した場合、変数aの値は4になる。

（3になる）

問題 2

次の値を求める式を書いてください。

（1） 100を9で割った商

（2） 1000を7で割った余り

（3） 3の5乗

問題２（解答）

次の値を求める式を書いてください。

（1） 100を9で割った商

```
100 // 9
```

（2） 1000を7で割った余り

```
1000 % 7
```

（3） 3の5乗

```
3 ** 5
```

問題 3

次の命令文を、加算代入（+=）、減算代入（-=）、乗算代入（*=）、除算代入（/=）、剰余代入（%=）の演算子を使って、短い表現に書き換えてください。

（1） `a = a + 5`

（2） `b = b - 6`

（3） `c = c * a`

（4） `d = d / 3`

（5） `e = e % 2`

問題３（解答）

次の命令文を、加算代入（+=）、減算代入（-=）、乗算代入（*=）、除算代入（/=）、剰余代入（%=）の演算子を使って、短い表現に書き換えてください。

(1) `a = a + 5`

`a += 5`

(2) `b = b - 6`

`b -= 6`

(3) `c = c * a`

`c *= a`

(4) `d = d / 3`

`d /= 3`

(5) `e = e % 2`

`e %= 2`

問題 4

次のプログラムコードを実行した後の変数aの値を答えてください（対話モードで実行するときに表示されるプロンプト「>>>」は省略しています）。

(1)
```
a = 3
a *= 3
```

(2)
```
b = 2
a = b * b
```

(3)
```
a = int(1.9)
```

(4)
```
x = 'XXX'
y = 'YYY'
a = x + y
```

問題４（解答）

次のプログラムコードを実行した後の変数aの値を答えてください（対話モードで実行するときに表示されるプロンプト「>>>」は省略しています）。

(1)
```
a = 3
a *= 3
```
9

(2)
```
b = 2
a = b * b
```
4

(3)
```
a = int(1.9)
```
1

(4)
```
x = 'XXX'
y = 'YYY'
a = x + y
```
XXXYYY

問題 5

「私は21歳です。」という文字列が出力されるように作成した次のプログラムコードは、実行するとエラーが発生します。適切に動作するように修正してください。

```
age = 21
print('私は' + age + '歳です。')
```

問題５（解答）

「私は21歳です。」という文字列が出力されるように作成した次のプログラムコードは、実行するとエラーが発生します。適切に動作するように修正してください。

```
age = 21
print('私は' + age + '歳です。')
```

```
age = 21
print('私は' + str(age) + '歳です。')
```

※ フォーマット文字列を使う場合

```
age = 21
print(f'私は{age}歳です。')
```

問題 6

mathモジュールを利用して、cos(120°)の値を求めてください。

問題 6 （解答）

mathモジュールを利用して、cos(120°)の値を求めてください。

```
import math
print(math.cos(math.radians(120)))
```

※ 別解

```
import math
print(math.cos(2 / 3 * math.pi))
```

第3章 条件分岐と繰り返し

一歩前に進むための準備

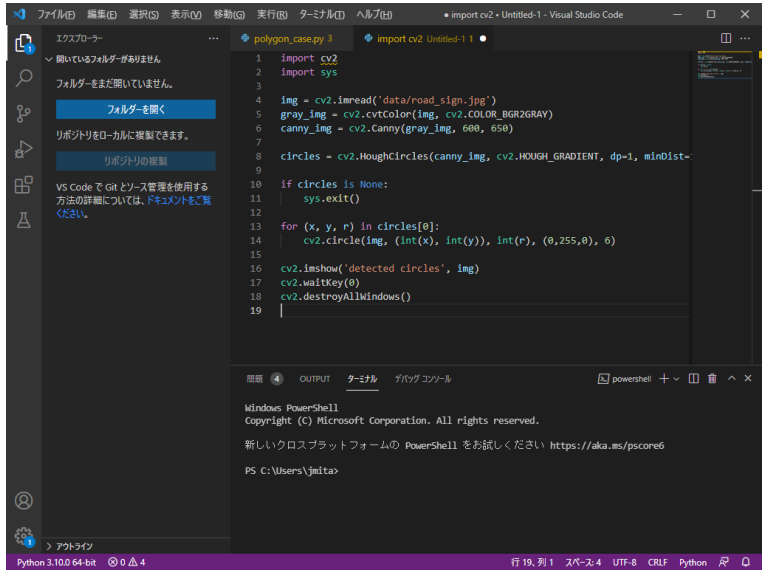
ファイルに保存したプログラムコードの実行(1/3)

- プログラムコードをファイルに保存して実行する利点
  - 一度作成したプログラムコードを繰り返し実行できる
  - 後から一部分だけ修正できる
  - プログラムコードを配布できる
  - 大きなプログラムを作成できる

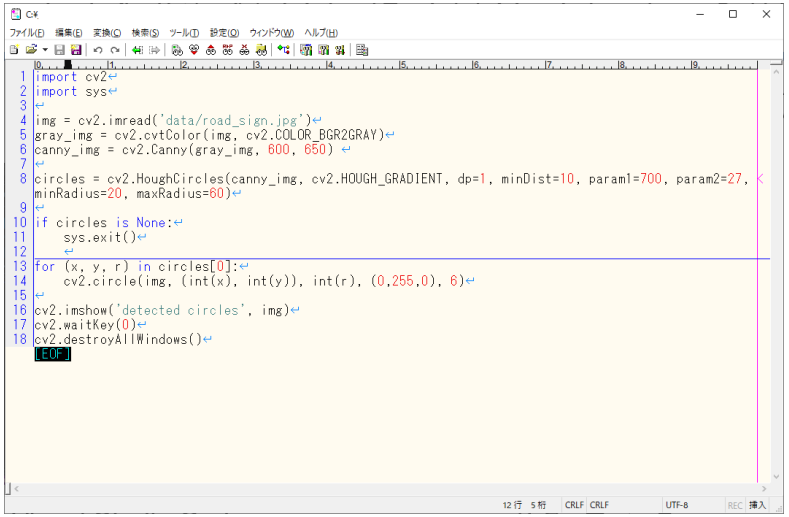
- 作成するファイル
  - テキストファイル
    - 文字コード：UTF-8
    - 拡張子：.py

ファイルに保存したプログラムコードの実行(2/3)

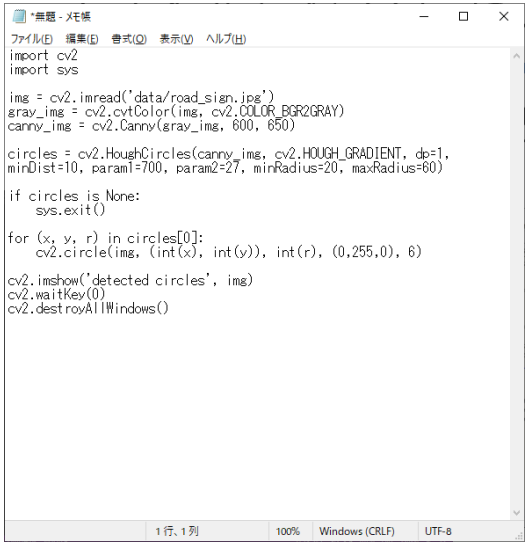
- プログラムコードを記述したファイルの作成



Visual Studio Code



サクラエディタ



メモ帳

ファイルに保存したプログラムコードの実行(3/3)

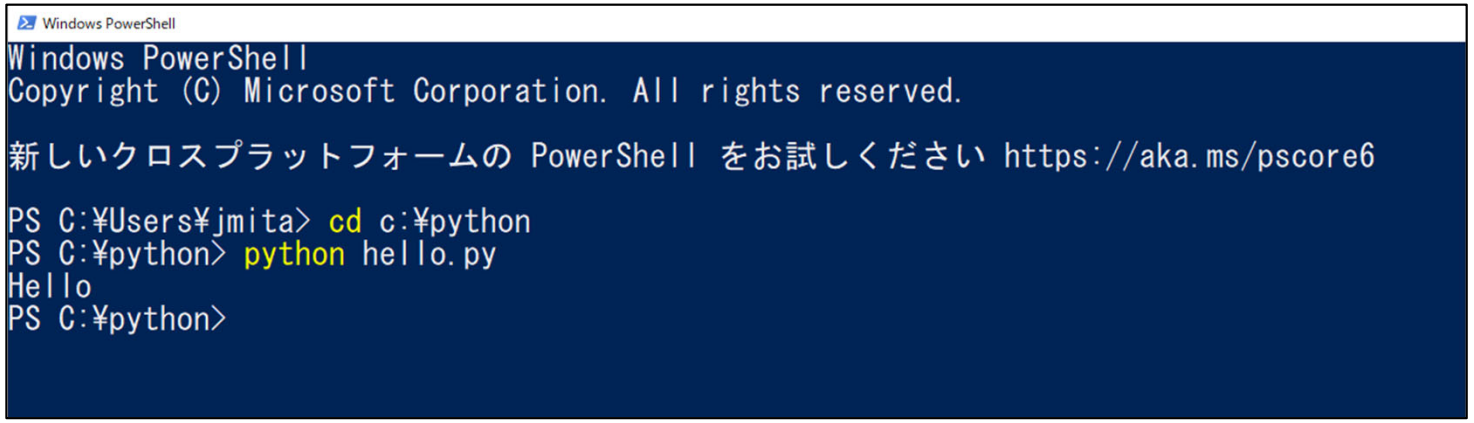
1. Windows PowerShell で保存したフォルダへ移動

```
PS C:¥Users¥py> cd C:¥python
PS C:¥python>
```

← ファイルを保存したフォルダのパスを入力します

2. プログラムの実行

   ＞ **python *ファイル名*** と入力



input関数を使ったキーボードからの入力の受け取り

```
1: name = input('名前を入力してください¥n')
2: print(name + 'さん、こんにちは。')
```

キーボードからの入力を変数nameで受け取ります（注❸-10）

受け取った文字列を使ってメッセージを出力します

実行結果

```
名前を入力してください
田中たかし
田中たかしさん、こんにちは。
```

プログラムからの出力です

キーボードで入力した文字列です

入力された文字列を使ったメッセージが出力されました

いろいろ試してみよう！

コメント文

記号に続けてコメント（メモ）を記述できる。
記号の後ろはプログラムに影響を与えない

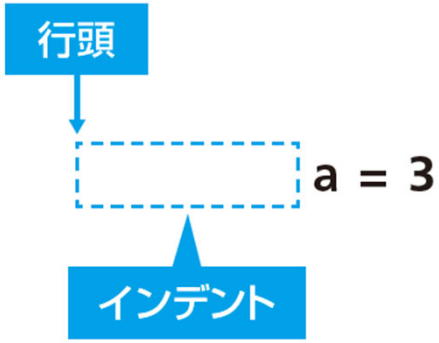
```
# 高さを受け取る
height = float(input('高さを入力してください¥n'))

# 幅を受け取る
width = float(input('幅を入力してください¥n'))

# 面積（高さ×幅）を計算して出力する
print(f'面積は{height * width}です')
```

※ プログラムコードの一部を一時的に無効にする用途でも使用できる

インデントとブロック

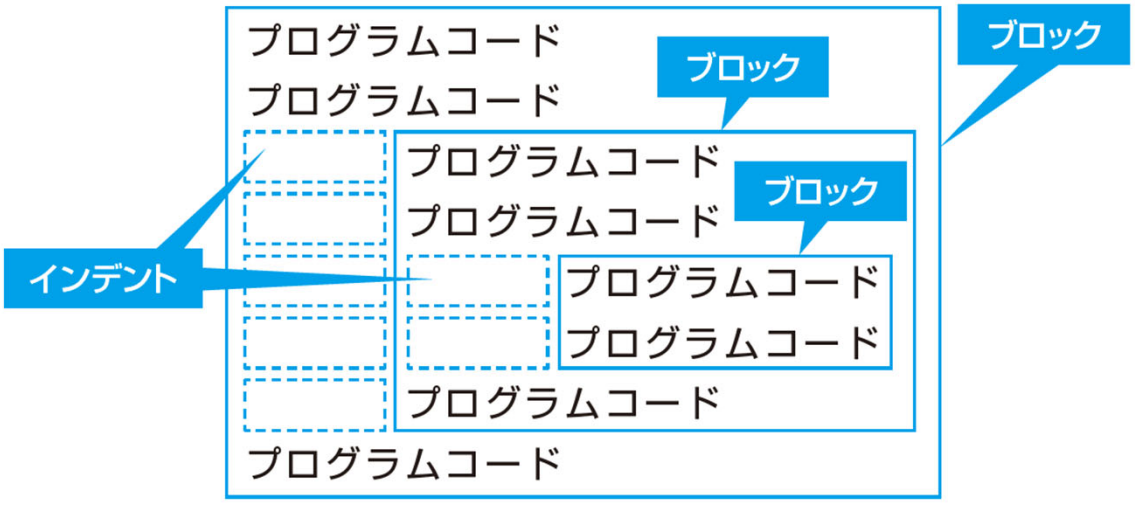
- **インデント**

行頭から最初の文字までの空白
空白文字4つ分を1つの単位とする



- **ブロック**

インデントによってプログラムコードを1つのまとまりにしたもの



条件分岐と論理演算子

if文による条件分岐

「もしも○○ならば××を実行する」

構文

コロンを付ける

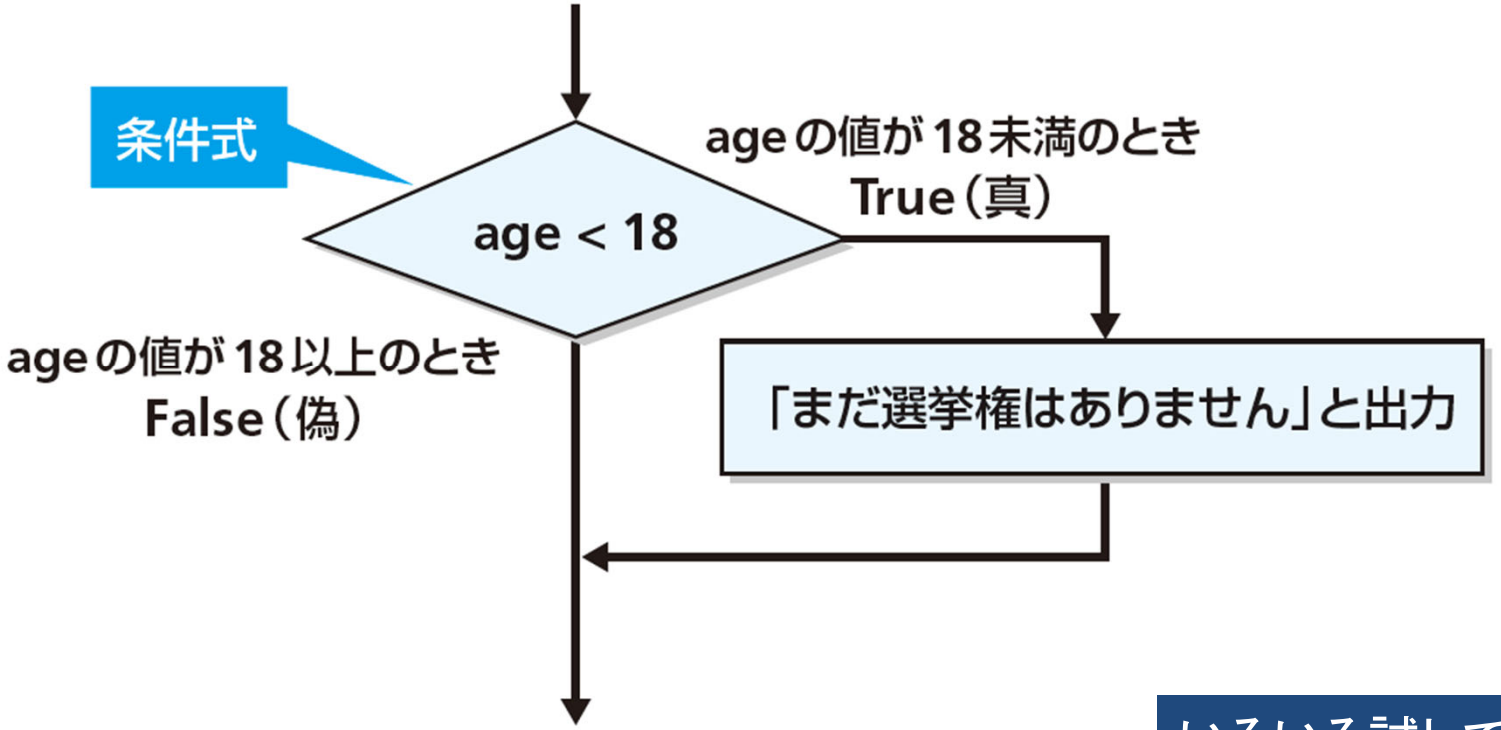
```
if 条件式:
    処理内容
```

インデント

条件を満たすとき（**条件式**の値が真（True）のとき）
処理内容が実行される。
そうでないときは、実行されない。

if文による条件分岐

```
age = 
if age < 18:
    print('まだ選挙権はありません')
```



いろいろ試してみよう！

if文による条件分岐

```
age = int(input('年齢を教えてください: '))
if age < 18:
    print('まだ選挙権はありません')
    print('18歳になったら投票に行きましょう')

print('処理を終わります')
```

キーボードから入力された数字を
int型にして、変数ageに代入します

条件式と関係演算子

演算子	説明	例
==	左辺と右辺が等しい	a == 1（変数aが1のときにTrue）
!=	左辺と右辺が等しくない	a != 1（変数aが1でないときにTrue）
>	左辺が右辺より大きい	a > 1（変数aが1より大きいときにTrue）
<	左辺が右辺より小さい	a < 1（変数aが1より小さいときにTrue）
>=	左辺が右辺より大きいか等しい	a >= 1（変数aが1以上のときにTrue）
<=	左辺が右辺より小さいか等しい	a <= 1（変数aが1以下のときにTrue）

```
if age == 18:
    print('18歳ですね。投票に行けますよ。')
```

if～else文による条件分岐

「もしも○○ならば××を実行し、そうでなければ△△を実行する」

構文

コロンを付ける

```
if 条件式:
    処理内容1
else:
    処理内容2
```

インデント

コロンを付ける

インデント

条件を満たすとき（**条件式**の値が真（True）のとき）
処理内容1が実行される。
そうでないときは、
処理内容2が実行される。

if～else文による条件分岐

```
age = 
if age < 18:
    print('まだ選挙権はありません')
else:
    print('投票に行きましょう')
```

age < 18がTrueのときに実行されます

age < 18がFalseのときに実行されます

いろいろ試してみよう！

if～elif～else文による条件分岐

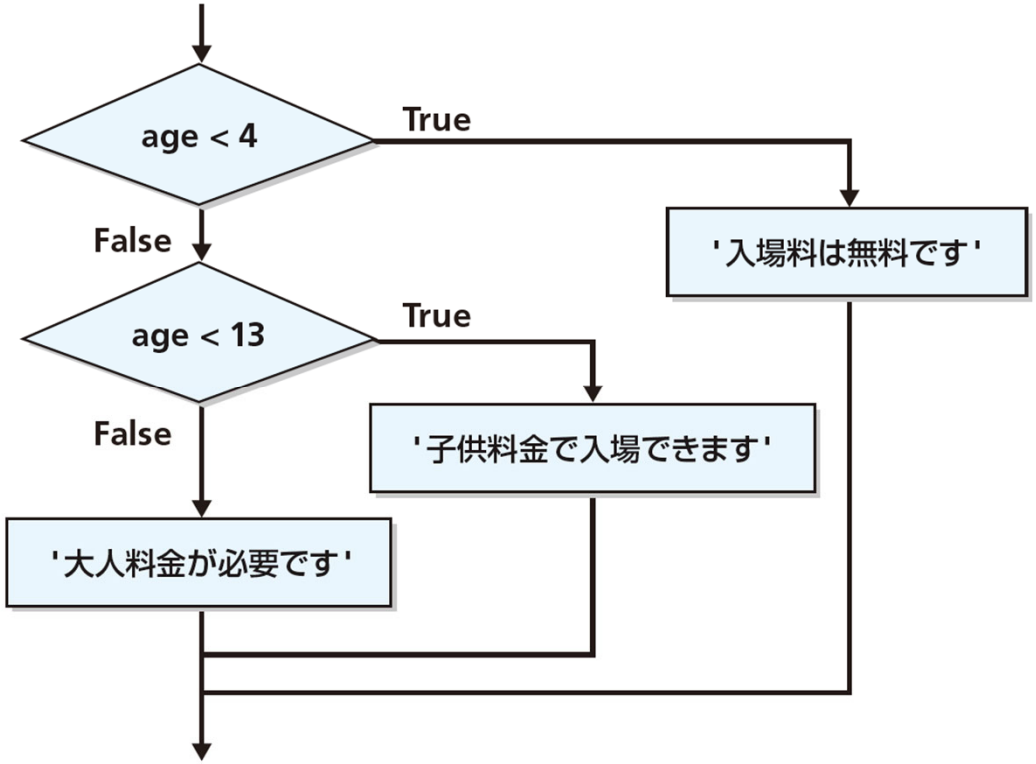
構文

```
if 条件式1:
    処理内容1
elif 条件式2:
    処理内容2
else:
    処理内容3
```

条件式1 が **True** のとき **処理内容1** が実行される。
そうでないとき、**条件式2** が **True** のとき **処理内容2** が実行される。
それ以外の時には、**処理内容2**が実行される。

if～elif～else文による条件分岐

```
age = 
if age < 4:
    print('入場料は無料です')
elif age < 13:
    print('子供料金で入場できます')
else:
    print('大人料金が必要です')
```



いろいろ試してみよう！

三項演算子

```
値1 if 条件式 else 値2
```

条件式 が **True** のとき、**値1** になる。
そうでないとき、**値2** になる。

例

```
c = a if a > b else b
```

c の値は a になる（もし a > b なら）。そうでなければ b

※ 英語の文章のようにして左から順番に読んでいくとわかりやすい

論理演算子による条件の組み合わせ

「変数aが10で、かつ変数bが5である」

```
a == 10 and b == 5
```

「変数aが10である、または変数bが5である」

```
a == 10 or b == 5
```

論理演算子

演算子	動作	式がTrueになる条件
`and`	論理積	左辺と右辺の両方が`True`のとき
`or`	論理和	左辺と右辺の少なくともどちらかが`True`のとき
`not`	否定	右辺が`False`のとき（左辺はなし）

論理演算子による条件の組み合わせ

```
age = □
if age < 13 or age >= 65:
    print('入場料は無料です。')
else:
    print('料金が必要です。')
```

いろいろ試してみよう！

演算子の優先度とカッコ

```
a + 10 > b * 5
```

```
(a + 10) > (b * 5)
```

```
a > 10 and b < 3
```

```
(a > 10) and (b < 3)
```

優先順位	演算子
高い	**
	* / % //
	+ -
	< > <= >= == !=
	not
	and
	or
低い	:=

比較演算子の連結

```
(a > 5) and (a < 10)
```

```
5 < a < 10
```

※ 左から順番に評価される
（5 < a が評価されてから a < 10 が評価される）

if文と真偽値

```
if a == True:
    処理内容
```

← aの値をTrueと比較しています

※ a が True のときだけ処理内容が実行される

```
if a:
    処理内容
```

← 条件式の代わりに変数aの値を用います

```
if not a:
    処理内容
```

※ a が False のときだけ処理内容が実行される

処理の繰り返し

while文による処理の繰り返し

構文

```
while 条件式:
    処理内容
```

※ 条件式の値が True の間、処理内容を繰り返す

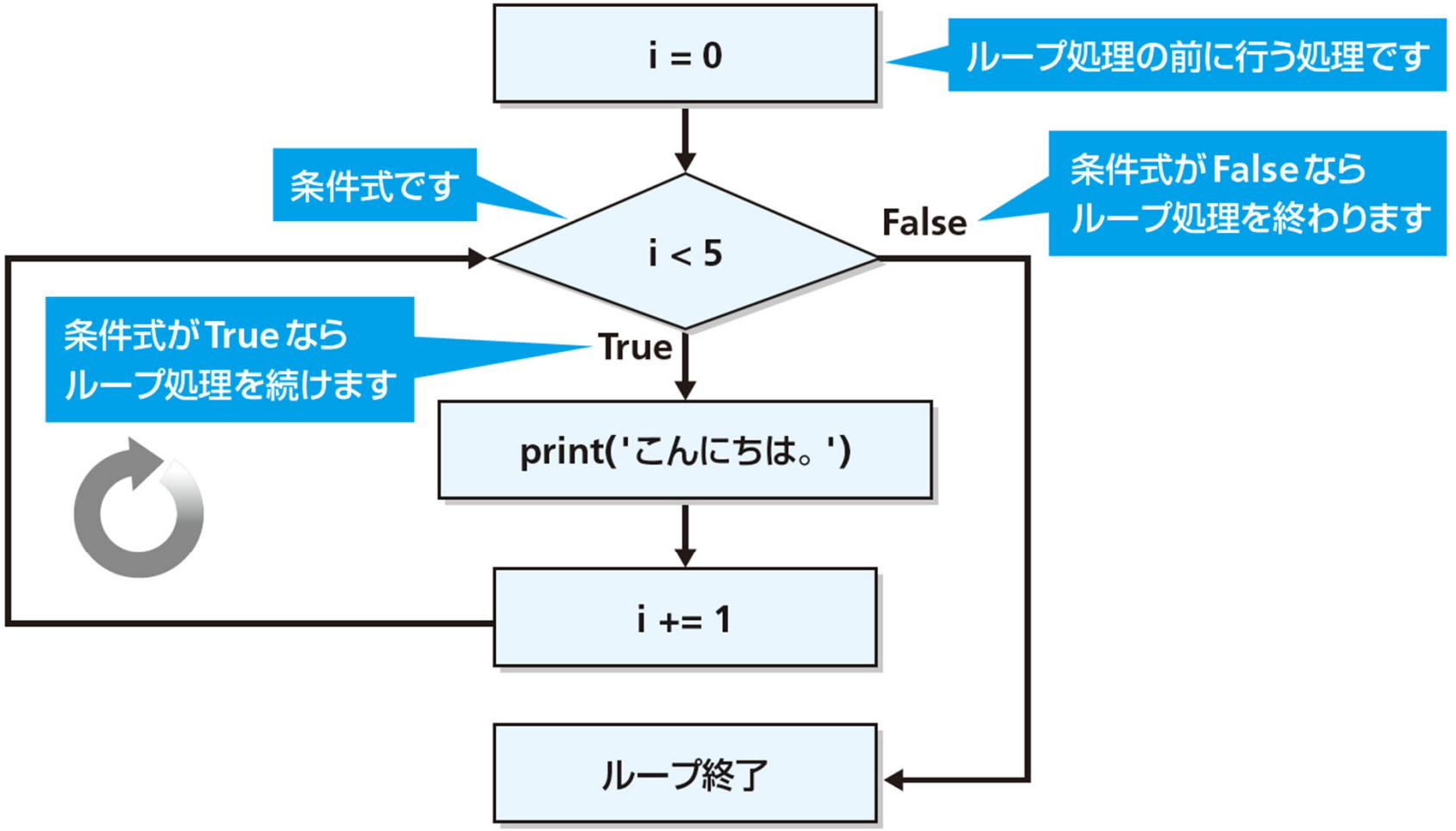
```
i = 0
while i < 5:
    print('こんにちは。')
    i += 1
```

実行結果

```
こんにちは。
こんにちは。
こんにちは。
こんにちは。
こんにちは。
```

処理の流れ

```
i = 0
while i < 5:
    print('こんにちは。')
    i += 1
```



while文の例

```
i = 20
while i > 0:
    print(i)
    i -= 5
```

← 変数iの値が0より大きければ次のブロックの処理を繰り返します

← 変数iの値を出力します

← 変数iの値を5だけ減らします

実行結果

```
20
15
10
5
```

20から5ずつ小さくなる値が出力されています

いろいろ試してみよう！

for文による処理の繰り返し

構文

```
for 変数 in 反復可能オブジェクト:
    処理内容
```

※「反復可能オブジェクト」から1つずつ要素をとりだして「変数」に代入。処理内容を繰り返す。

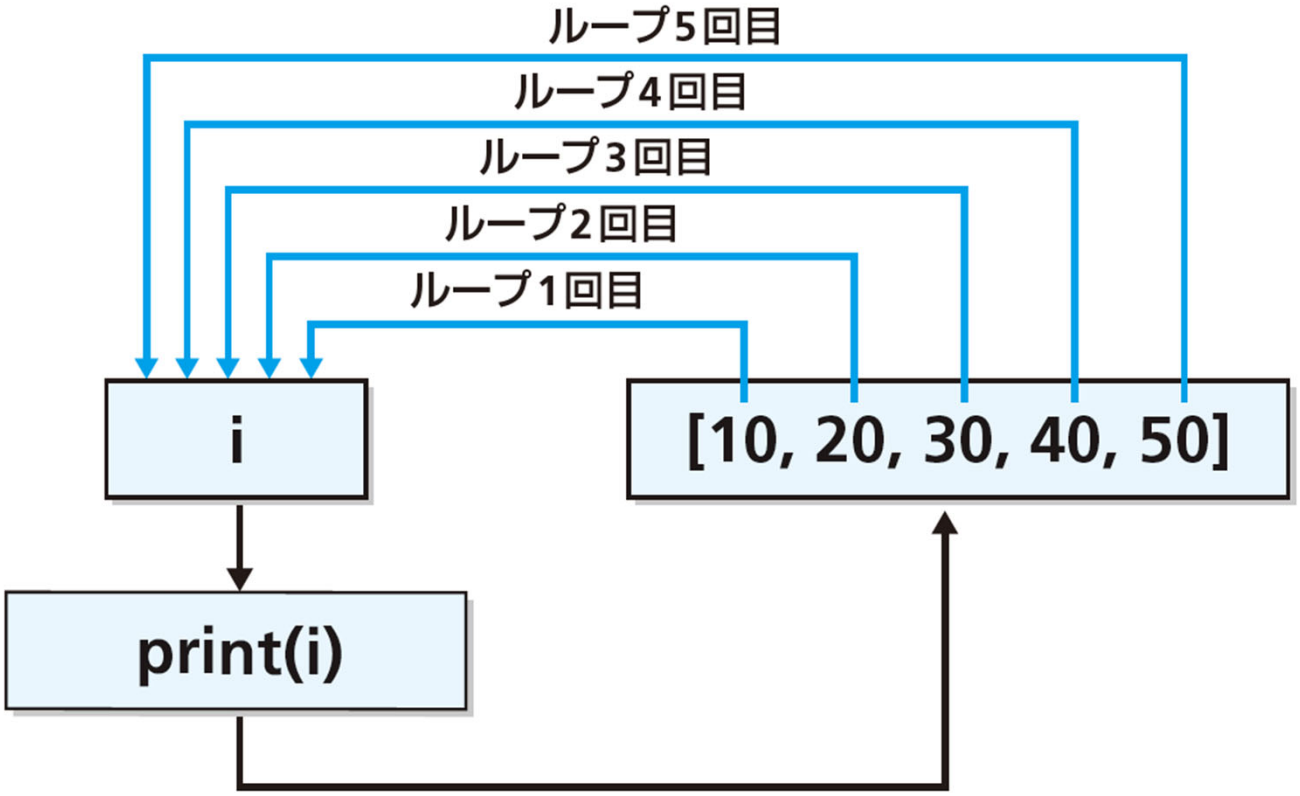
```
for i in [10, 20, 30, 40, 50]:
    print(i)
```

実行結果

```
10
20
30
40
50
```

for 文の処理の流れ

```
for i in [10, 20, 30, 40, 50]:
    print(i)
```



rangeオブジェクト

```
for i in range(10):
    print(i)
```

変数iに0から9の値が順番に代入されます

実行結果

```
0
1
2
(略)
9
```

0から9の値が1ずつ順番に出力されます

rangeオブジェクトの生成方法	得られる整数の列
range(stop)	0から「stopの値−1」までの整数
range(start, stop)	startの値から「stopの値−1」までの整数
range(start, stop, step)	startの値から「stopの値−1」までの整数。ただし、増分はstepの値

range オブジェクト

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(3, 10))
[3, 4, 5, 6, 7, 8, 9]
>>> list(range(1, 30, 10))
[1, 11, 21]
```

0から9までの数字が並びます

3から9までの数字が並びます

29を超えない範囲で1から10ずつ値が増えます

```
for i in range(100, 201, 5):
    print(i)
```

100から始まり5ずつ増える値が、200に達するまで順番に代入されます

実行結果

```
100
105
110
(略)
200
```

いろいろ試してみよう！

ループ処理の流れの変更

break ループの処理を中断する

```
total = 0
for i in range(10):
    total += i
    if total > 20:
        break

print(i, total)
```

ループ処理の流れの変更

continue ループ内の処理をスキップする

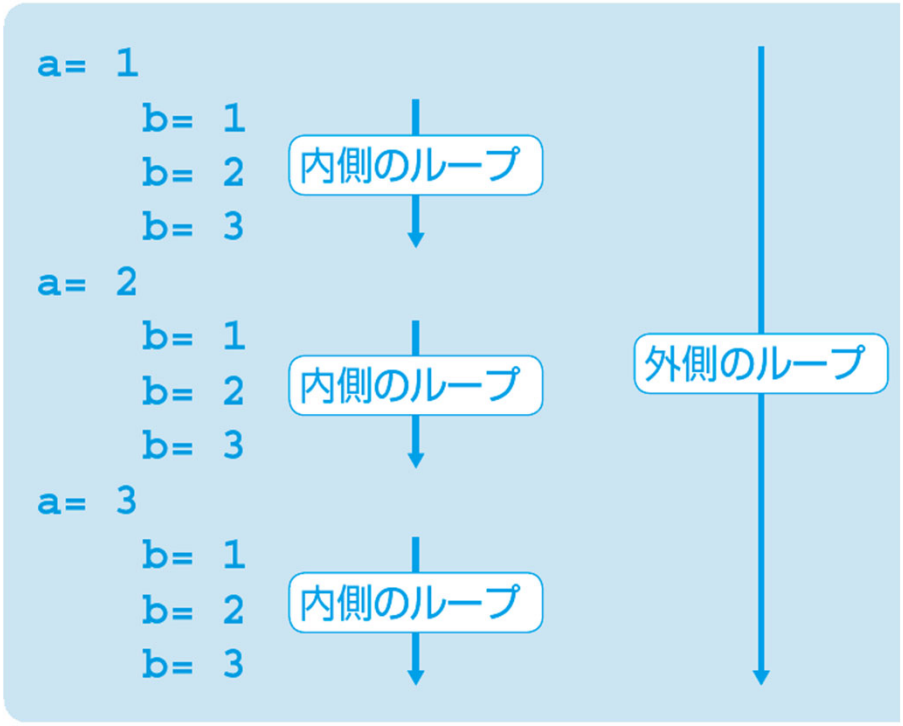
```
total = 0
for i in range(100):
    if i % 3 == 0:
        continue
    print(i)
    total += i

print('合計は', total)
```

ループ処理のネスト

```
for a in range(1, 4):
    print('a=', a)
    for b in range(1, 4):
        print('    b=', b)
```

forループの中にfor文があります

実行結果



いろいろ試してみよう！

練習問題

問題 1

次の条件を、関係演算子を使って記述してください。

問題例　aはbより大きい

解答例　**a ＞ b**

（1） aはbと等しい

（2） aはbと等しくない

（3） bはcより小さい

（4） aはb以下である

（5） cはb以上である

問題１（解答）

次の条件を、関係演算子を使って記述してください。

問題例　aはbより大きい

解答例　a ＞ b

（1）　aはbと等しい　　a == b

（2）　aはbと等しくない　　a != b

（3）　bはcより小さい　　b < c

（4）　aはb以下である　　a <= b

（5）　cはb以上である　　c >= b

問題 2

次のプログラムコードにある空欄を埋めて、変数aの値が3で割り切れるときには「3で割り切れます」、そうでないときには「3で割り切れません」とコンソールに出力するプログラムを完成させてください。

```
a = 2021
```

空欄

問題２（解答）

次のプログラムコードにある空欄を埋めて、変数aの値が3で割り切れるときには「3で割り切れます」、そうでないときには「3で割り切れません」とコンソールに出力するプログラムを完成させてください。

```
a = 2021
```

空欄

```
if a % 3 == 0:
    print('3で割り切れます')
else:
    print('3で割り切れません')
```

問題 3-1

10から20までの整数を順番に足し合わせて、その結果を出力するプログラムを作ってください。

ただし、while文を使った場合とfor文を使った場合の2つのプログラムコードを作成してください。

問題 3-1（解答）

10から20までの整数を順番に足し合わせて、その結果を出力するプログラムを作ってください。

ただし、while文を使った場合とfor文を使った場合の2つのプログラムコードを作成してください。

while 文

```
total = 0
i = 10
while i < 21:
    total += i
    i += 1
print(total)
```

for 文

```
total = 0
for i in range(10, 21):
    total += i
print(total)
```

問題 3-2

問題3-1で作成したfor文を使ったプログラムコードに対して、15だけは足し合わせしないように、変更してください。ただし、continue 命令を使ってください。

for 文

```
total = 0
for i in range(10, 21):
    total += i
print(total)
```

問題 3-2（解答）

問題3-1で作成したfor文を使ったプログラムコードに対して、15だけは足し合わせしないように、変更してください。ただし、continue 命令を使ってください。

for 文

```
total = 0
for i in range(10, 21):
    if i == 15:
        continue
    total += i
print(total)
```

問題 4

次の条件を、論理演算子と関係演算子を使って記述してください。

（1） aは5または8と等しい

（2） aとcは両方ともb以下

（3） aは1より大きくて10より小さいが、5ではない

（4） aはbまたはcと等しいが、aとdは等しくない

問題４（解答）

次の条件を、論理演算子と関係演算子を使って記述してください。

（1） aは5または8と等しい

```
a == 5 or a == 8
```

（2） aとcは両方ともb以下

```
a <= b and c <= b
```

（3） aは1より大きくて10より小さいが、5ではない

```
a > 1 and a < 10 and a != 5
```

（4） aはbまたはcと等しいが、aとdは等しくない

```
(a == b or a == c) and a != d
```

問題 5

次に示すものは、scoresという変数名のリストに格納されている要素のうち、値が60より大きいものの数をカウントするプログラムコードです。空欄を埋めて、完成させてください。

```
scores = [65, 80, 40, 92, 76, 52]
count = 0    # 値が60よりも大きな要素の数
for i in scores:
        空欄
print(count)    # 結果を出力
```

問題５（解答）

次に示すものは、scoresという変数名のリストに格納されている要素のうち、値が60より大きいものの数をカウントするプログラムコードです。空欄を埋めて、完成させてください。

```
scores = [65, 80, 40, 92, 76, 52]
count = 0    # 値が60よりも大きな要素の数
for i in scores:
    if i > 60:
        count += 1
print(count)   # 結果を出力
```

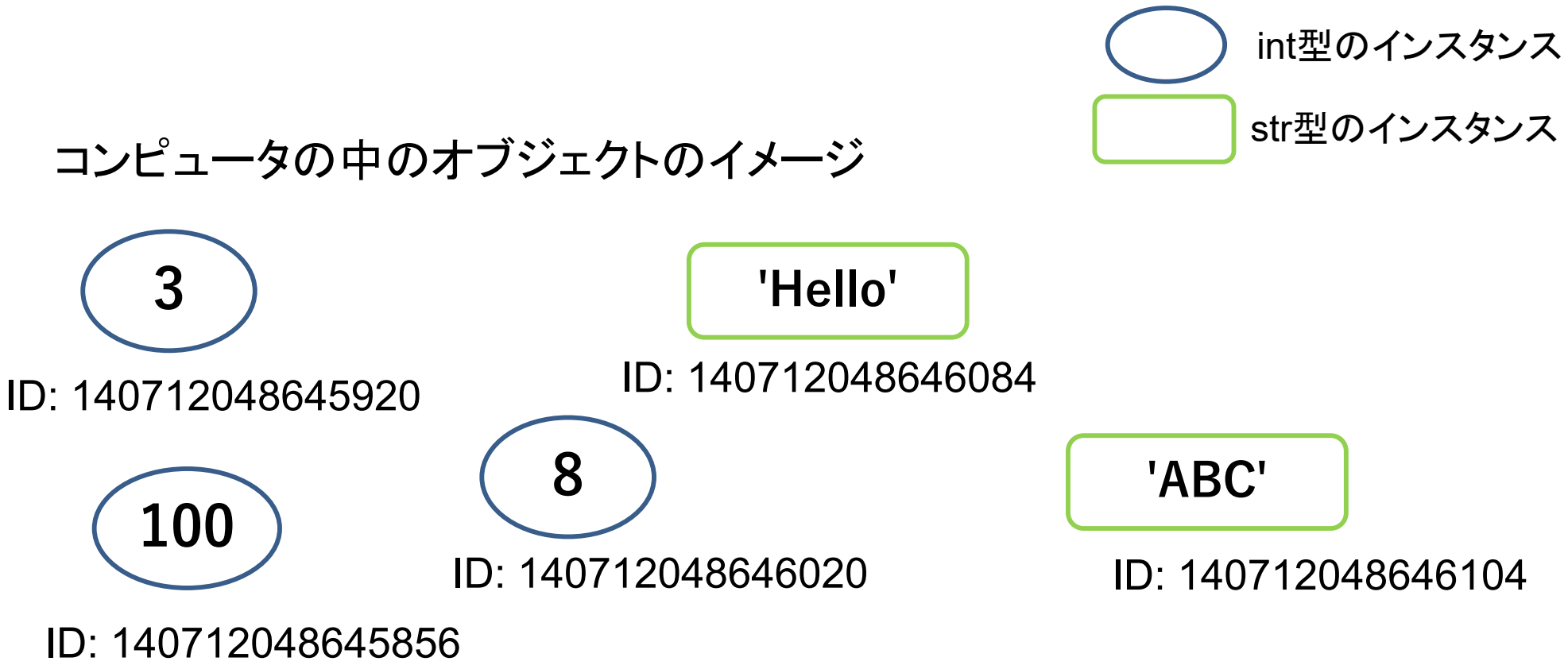
第4章 組み込み型とオブジェクト

オブジェクト指向

インスタンスの管理とID

- Python はオブジェクト指向言語

  1つ1つの**オブジェクト**にIDを割り当てて管理



代入の正確なイメージ

```
a = 3
```

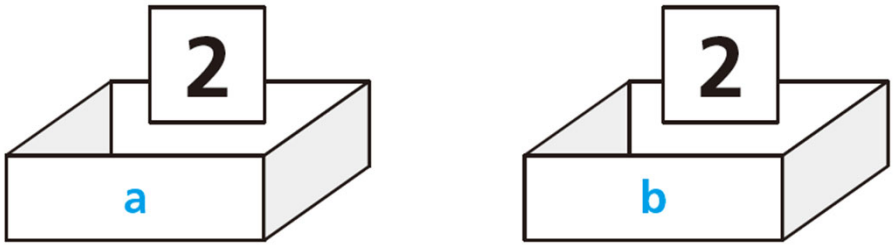
←「変数aに3を**代入**する」

「3という値を表すオブジェクトがコンピュータのメモリ上のどこかに保管される。その保管場所を示す所在地情報が、aという名前の箱に入れられる」

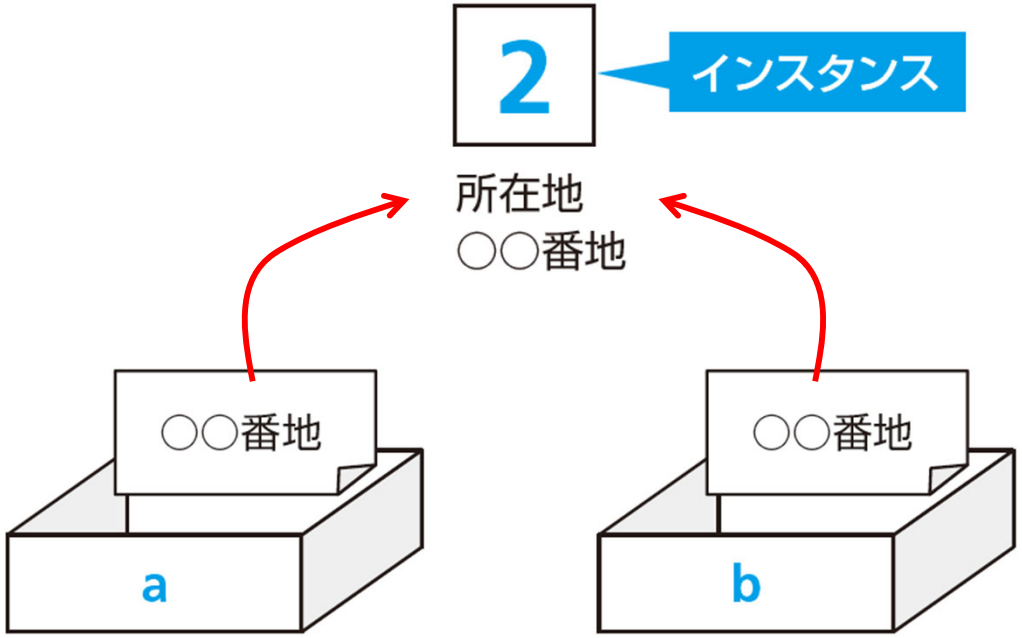
代入の正確なイメージ

```
a = 2
b = a
```

✕ まちがったイメージ



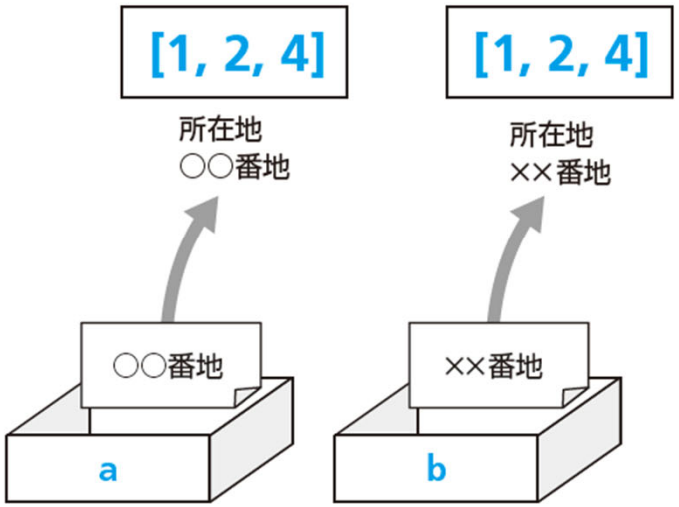
○ 正しいイメージ



インスタンスの同値性と同一性

「同値」と「同一」は異なる概念

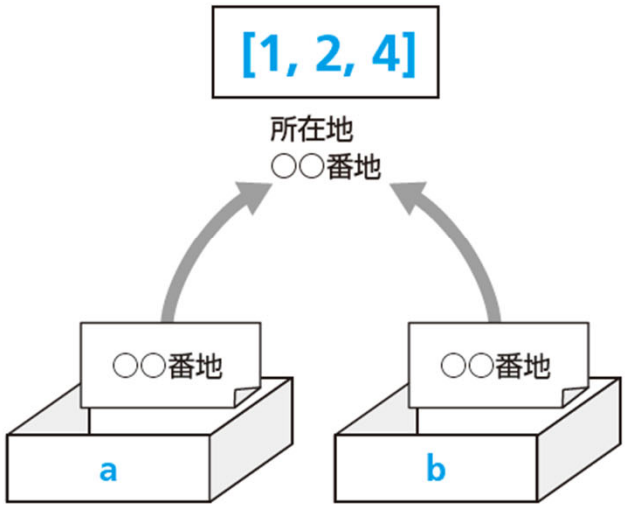
変数aとbは異なるインスタンスを参照している



同値だけど同一でない

式	値
a == b	True
a is b	False

変数aとbは同じインスタンスを参照している



同値かつ同一

式	値
a == b	True
a is b	True

※ 「==演算子」は同値性を、「is 演算子」は同一性を判定する

クラスとインスタンスとメソッド

- **インスタンス**
  オブジェクトのこと
- **クラス**
  インスタンスの種類（型）のこと
- **メソッド**
  クラスごとに定義された機能
  （関数のようなもの）

例1：**'Hello'** というオブジェクト（文字列）はstrクラスのインスタンスで、**lower()**や**count()**などのメソッドを持つ

例2：**[10,20,30]**というオブジェクト（リスト）はlistクラスのインスタンスで、**reverse()**や**pop()**などのメソッドを持つ

組み込み型

- Pythonに最初から準備されている型（クラス）

- これまでに学習したもの
  int型、float型、str型、list型

- これから学習するもの
  タプル、辞書、セット
  ※ いずれも複数のオブジェクトを格納するために使用する

メソッドの呼び出し

プログラムコードでの表記

オブジェクト.**メソッド名( )**

例

```
>>> 'Hello, Python'.count('o')
2
```

オブジェクト　　メソッド名

strクラスのメソッド

strクラスには文字列を操作するメソッドが定義されている

`count(x)`	xが文字列にいくつ含まれるかを返す
`lower()`	すべての文字を小文字にした文字列を返す
`upper()`	すべての文字を大文字にした文字列を返す
`split(x)`	文字列をxで分割した結果をリストで返す
`replace(x, y)`	文字列に含まれるxをyに置換した結果を返す

```
>>> 'Hello, Python'.count('o')
2
>>> 'PYTHON'.lower()
python
>>> 'python'.upper()
PYTHON
>>> '2021-12-31'.split('-')
['2021', '12', '31']
>>> 'Java言語'.replace('Java', 'Python')
'Python言語'
```

strクラスのformatメソッド

文字列に含まれる{} への変数の値の埋め込み

```
>>> year = 2021
>>> month = 4
>>> day = 10
>>> message = '今日は{}年{}月{}日です'.format(year, month, day)
>>> print(message)
今日は2021年4月10日です
```

← 文字列の中の{}に数字が埋め込まれました

{}の中でのインデックスの指定

```
>>> year = 2021
>>> month = 4
>>> day = 10
>>> message = '今日は{1}月{2}日です。西暦{0}年です。'.format(year,
month, day)
>>> print(message)
今日は4月10日です。西暦2021年です。
```

変数monthの値が埋め込まれます

変数yearの値が埋め込まれます

変数dayの値が埋め込まれます

strクラスのformatメソッド

{}の中での書式を指定できる

{インデックス番号:書式} のように記述する
（インデックス番号は省略可）

書式の例
.3f　小数点以下3桁まで出力
,　　3桁ごとにカンマで区切る
>5　5文字分の幅に右寄せする

```
>>> '1/3={:.3f}'.format(1/3)
1/3=0.333
>>> '{:,}円'.format(1000000)
1,000,000円
>>> '{:>5}'.format(123)
  123
```

いろいろ試してみよう！

in 演算子

```
文字列1 in 文字列2
```

文字列1 が **文字列2** に含まれると **True**、
そうでないなら **False** になる

```
>>> 'cat' in 'catch'
True
>>> 'Z' in 'ABCDE'
False
```

```
>>> s1 = 'py'
>>> s2 = 'python'
>>> s1 in s2
True
```

いろいろ試してみよう！

リストとタプル

listクラスの主なメソッド

```
a = [10, 20, 30, 40]
```
← a はListクラスのインスタンス

メソッド	説明
append(x)	末尾にxを追加する
insert(i, x)	インデックスiの位置にxを挿入する
remove(x)	最初に見つかった、値がxの要素を削除する
pop()	末尾にある要素を返し、削除する
clear()	全要素を削除する
index(x)	最初に見つかった、値がxの要素のインデックスを返す
count(x)	要素にxが出現する回数を返す
reverse()	要素の並び順を反転する

メソッドを使ったリストの操作

```
>>> a = [10, 20, 30, 40]
>>> a.append(100)
>>> a
[10, 20, 30, 40, 100]
>>> a.pop()
100
>>> a
[10, 20, 30, 40]
>>> a.reverse()
>>> a
[40, 30, 20, 10]
>>> a.index(20)
2
```

いろいろ試してみよう！

メソッド以外のリストの操作

```
>>> a = [10, 20, 30, 40]
>>> del a[1]
>>> a
[10, 30, 40]
>>> a = a + [0, 1]
>>> a
[10, 30, 40, 0, 1]
>>> 1 in a
True
>>> 2 in a
False
>>> len(a)
5
>>> sorted(a)
[0, 1, 10, 30, 40]
>>> sorted(a, reverse=True)
[40, 30, 10, 1, 0]
```

いろいろ試してみよう！

内包表記

for文を用いてリストの要素を作成することができる

構文

```
[式 for 変数 in 反復可能なオブジェクト]
```

例

```
>>> data = [2**n for n in range(1, 11)]
>>> data
[2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]
```

```
>>> [str(n)+'月' for n in range(1, 13)]
['1月', '2月', '3月', '4月', '5月', '6月', '7
月', '8月', '9月', '10月', '11月', '12月']
```

いろいろ試してみよう！

if 構文を含む内包表記

構文

```
[式 for 変数 in 反復可能なオブジェクト if 条件式]
```

例

```
>>> data0 = ['apple', 'orange', 'banana', 'avocado']
>>> data1 = [s for s in data0 if s[0] == 'a']
>>> print(data1)
['apple', 'avocado']
```

← aで始まる単語だけが格納されています

いろいろ試してみよう！

リストを含むリスト

```
>>> data = [[1, 2], [3, 4, 5]]
>>> data[0][0]
1
>>> data[0][1]
2
>>> data[1][0]
3
>>> data[1][1]
4
>>> data[1][2]
5
```

先頭のリストの先頭の要素を参照します

先頭のリストの2番目の要素を参照します

2番目のリストの先頭の要素を参照します

2番目のリストの2番目の要素を参照します

2番目のリストの3番目の要素を参照します

タプル

- リスト同様に複数の要素を格納する（**コレクション**とよぶ）

リスト

```
>>> a = [1, 2, 3, 4]
>>> a
[1, 2, 3, 4]
>>> a[0]
1
>>> a[0] = 99
>>> a
[99, 2, 3, 4]
```

タプル

```
>>> a = (1, 2, 3, 4)
>>> a
(1, 2, 3, 4)
>>> a[0]
1
>>> a[0] = 99
エラー
```

タプルは要素の値を変更できない

( )の省略とアンパック代入

タプルの作成時に ( ) を省略できる

```
>>> a = 1, 2, 3, 4
>>> a
(1, 2, 3, 4)
```

複数の変数へ1つずつ値を代入できる
（**アンパック代入とよぶ**）

```
>>> a, b, c = (1, 2, 3)
>>> a
1
>>> b
2
>>> c
3
```

複数の変数への一括代入

```
>>> x, y, z = 1, 3, 9
>>> x
1
>>> y
3
>>> z
9
```

いろいろ試してみよう！

辞書と集合（セット）

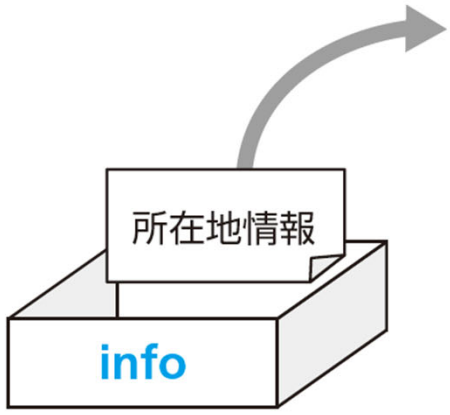
辞書

- **キー**と**値**のペアを格納
- インデックスの代わりにキーを使って値を取得できる

構文

```
{キー1:値1, キー2:値2, キー3:値3, … }
```

```
>>> info = {'firstname':'太郎', 'lastname':'山田', 'address': '茨
城県つくば市 99-99'}
>>> info['firstname']
太郎
```



キー(key)	値(value)
'firstname'	'太郎'
'lastname'	'山田'
'address'	'茨城県つくば市99-99'

辞書の要素の取得

keysメソッドによる
すべてのキーの取得

```
>>> for k in info.keys():
...     print(k)
...
firstname
lastname
address
```

valuesメソッドによる
すべての値の取得

```
>>> for v in info.values():
...     print(v)
...
太郎
山田
茨城県つくば市 99-99
```

itemsメソッドによる
すべての要素（キーと値のペアを格納したタプル）の取得

```
>>> for i in info.items():
...     print(i)
...
('firstname', '太郎')
('lastname', '山田')
('address', '茨城県つくば市 99-99')
```

いろいろ試してみよう！

辞書の操作

```
>>> info = {'firstname':'太郎', 'lastname':'山田',
'address': '茨城県つくば市 99-99'}

>>> info.get('firstname')
太郎
>>> info.get('tel')
None
>>> info['tel'] = '090-000-0000'
>>> info.get('tel')
'090-000-0000'
>>> del info['tel']
>>> len(info)
3
>>> 'age' in info
False
>>> 'firstname' in info
True
```

いろいろ試してみよう！

辞書の要素の並べ替え

sorted 関数を使うと、
辞書の要素を昇順に取得できる（**キー**を基準）

```
>>> data = {'b':5, 'c':2, 'a':8, 'd':7}
>>> print(sorted(data.items()))
[('a', 8), ('b', 5), ('c', 2), ('d', 7)]
```

sorted 関数に **key=lambda x:x[1]** を指定すると、
値を基準とした昇順に取得できる

```
>>> data = {'b':5, 'c':2, 'a':8, 'd':7}
>>> print(sorted(data.items(), key=lambda x:x[1]))
 [('c', 2), ('b', 5), ('d', 7), ('a', 8)]
```

セット

- **値**を格納
- 要素の重複を許さない。順序が無い。

構文 `{値1, 値2, …}`

リストは **[1, 2, 3]**
タプルは **(1, 2, 3)**
セットは **{1, 2, 3}**

```
>>> a = {'A', 'B', 'C', 'D'}
>>> a
{'A', 'B', 'C', 'D'}
```

```
>>> a = {'A', 'B', 'B', 'C', 'C', 'C'}
>>> a
{'A', 'B', 'C'}  ←それぞれの値が1つずつ含まれる
```

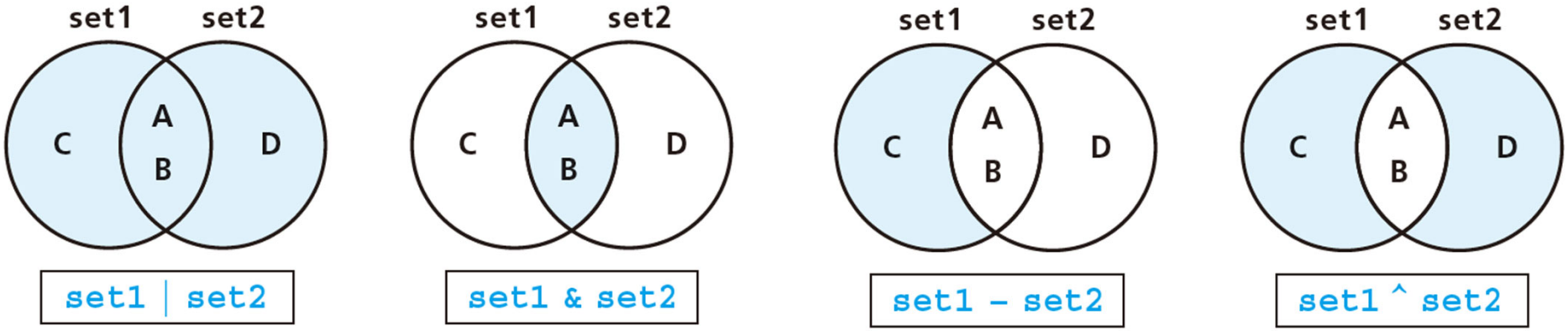
セットの操作

```
>>> data = [1, 2, 3]
>>> a = set(data)
>>> a
{1, 2, 3}
>>> set1 = {'A', 'B', 'C'}
>>> set2 = {'B', 'C'}
>>> set1.issubset(set2)
False
>>> set2.issubset(set1)
True
```

いろいろ試してみよう！

セットどうしの演算（集合演算）

演算子	演算	例
|	和集合	set1 | set2（set1とset2の少なくともどちらか一方に含まれる要素からなるセットを得る）
&	積集合	set1 & set2（set1とset2の両方に含まれる要素からなるセットを得る）
-	差集合	set1 - set2（set1から、set2に含まれる要素を除いたセットを得る）
^	排他的論理和	set1 ^ set2（set1とset2のどちらか一方にだけ含まれる要素からなるセットを得る）



set1 = { 'A', 'B', 'C' }　　set2 = { 'A', 'B', 'D' }

セットどうしの演算（集合演算）

```
>>> set1 = {'A', 'B', 'C'}
>>> set2 = {'A', 'B', 'D'}
>>> set1 | set2
{'D', 'B', 'C', 'A'}
>>> set1 & set2
{'A', 'B'}
>>> set1 - set2
{'C'}
>>> set1 ^ set2
{'C', 'D'}
```

いろいろ試してみよう！

基本型の性質

- 変更可能（ミュータブル）
- 反復可能（イテラブル）
- 順序を持つ（シーケンス）

基本型の性質

		変更可能 （ミュータブル）	反復可能 （イテラブル）	順序を持つ （シーケンス型）
int float bool	数値、真偽値	×	×	×
str	文字列	×	○	○
list	リスト	○	○	○
tuple	タプル	×	○	○
dict	辞書	○	○	×
set	集合	○	○	×

for文で要素を1つ1つ参照できる

インデックスで要素を参照できる

ミュータブルな型とイミュータブルな型

- ミュータブルな型：値を変更できる

  list型、**dict**型、**set**型
  （リスト、辞書、セット）

- イミュータブルな型：値を変更できない

  int型、**float**型、**bool**型、**str**型、**tuple**型
  （数値、真偽値、文字列、タプル）

  ※ インスタンス（オブジェクト）そのものを取り換えないと値を変えられない

ミュータブルな型とイミュータブルな型

```
>>> a = 2
>>> id(a)
140735461447360
>>> a = 3
>>> id(a)
140735461447392
```

変数**a**の値を変える

↓

変数**a**が参照するオブジェクトが別のものに替わる

```
>>> a = [1, 2, 3]
>>> id(a)
2103730545344
>>> a[0] = 3
>>> id(a)
2103730545344
```

変数**a**の要素の値を変える

↓

変数**a**が参照するオブジェクトは変わらない

反復可能なオブジェクト

```
for a in        :
    print(a)
```

反復可能なオブジェクト
（**イテラブル**なオブジェクト）

※ for 構文で1つ1つの要素を参照できる
文字列、セット、タプル、リスト、辞書
（str型、tuple型、list型、dict型、set型）

```
>>> for a in 'Python':
...     print(a)

P
y
t
h
o
n
```

```
>>> for a in {1, 2, 3}:
...     print(a)

1
2
3
```

順序を持つオブジェクト

a[0]，a[1]のようにインデックスで要素を参照できる
（**シーケンス型**のオブジェクト）

文字列、リスト、タプル
（str型、list型、 tuple型）

```
>>> a = 'Python'
>>> a[2]
't'
```

```
>>> a = (1, 2, 3)
>>> a[2]
3
```

スライス式

スライス式でインデックスの範囲を指定できる
[start:end] ← start ～ (end-1)の範囲

```
>>> a = 'Python'
>>> a[1:3]
'yt'
>>> a = (0, 1, 2, 3, 4)
>>> a[2:5]
(2, 3, 4)
```

[start:end] start または end を省略できる。
省略した場合は、それぞれ0と（要素数）を指定したものとみなされる。

```
>>> a = 'Python'
>>> a[:3]
'Pyt'
>>> a = (0, 1, 2, 3, 4)
>>> a[2:]
(2, 3, 4)
```

基本型の性質

		変更可能 （ミュータブル）	反復可能 （イテラブル）	順序を持つ （シーケンス型）
int float bool	数値、真偽値	×	×	×
str	文字列	×	○	○
list	リスト	○	○	○
tuple	タプル	×	○	○
dict	辞書	○	○	×
set	集合	○	○	×

for文で要素を1つ1つ参照できる

インデックスで要素を参照できる

練習問題

問題 1

次の文章の空欄に入れるべき語句を答えてください。

- Pythonは [(1)] 指向型の言語であり、クラスは[(1)]の属性や機能を定義したものです。
- 'Hello'や'Python'といった文字列はstr型のオブジェクトですが、別の表現をすると、strクラスの [(2)] である、ということができます。
- strクラスには、文字列に含まれる文字を小文字に変換するlowerという[(3)]があります。
- a == bという式の値がTrueであったとき、aとbは [(4)] であるといい、a is bという式の値がTrueであったとき、aとbは [(5)] であるといいます。

問題１（解答）

次の文章の空欄に入れるべき語句を答えてください。

- Pythonは [**オブジェクト**] 指向型の言語であり、クラスは[**オブジェクト**]の属性や機能を定義したものです。

- 'Hello'や'Python'といった文字列はstr型のオブジェクトですが、別の表現をすると、strクラスの [**インスタンス**] である、ということができます。

- strクラスには、文字列に含まれる文字を小文字に変換するlowerという[**メソッド**]があります。

- a == bという式の値がTrueであったとき、aとbは [**同値**] であるといい、a is bという式の値がTrueであったとき、aとbは [**同一**] であるといいます。

問題 2

n 番目の要素の値がn * nであるようなリストを、内包表記を使って作成してください。ただし、要素数は10とします。

問題２（解答）

n 番目の要素の値がn * nであるようなリストを、内包表記を使って作成してください。ただし、要素数は10とします。

```
[n * n for n in range(1, 11)]
```

問題 3

次の説明文が、リスト、タプル、辞書、セットのいずれに該当するか、答えてください（複数が該当する場合もあります）。

(1) キーと値のペアを格納する
(2) 要素の値の重複が許されない
(3) 要素の値を変更できない
(4) インデックスで要素にアクセスできる

問題３（解答）

次の説明文が、リスト、タプル、辞書、セットのいずれに該当するか、答えてください（複数が該当する場合もあります）。

(1) キーと値のペアを格納する　　**辞書**
(2) 要素の値の重複が許されない　　**セット**
(3) 要素の値を変更できない　　**タプル**
(4) インデックスで要素にアクセスできる

リストとタプル

問題 4

下記の表は、基本型が、どのような性質を持つかをまとめたものです。空欄に対して、○か×を入れてください。

		変更可能 （ミュータブル）	反復可能 （イテラブル）	順序を持つ （シーケンス型）

`int`
`float`
`bool`
	数値、真偽値			

`str`	文字列			
`list`	リスト			
`tuple`	タプル			
`dict`	辞書			
`set`	セット			

問題 4（解答）

下記の表は、基本型が、どのような性質を持つかをまとめたものです。空欄に対して、○か×を入れてください。

		変更可能（ミュータブル）	反復可能（イテラブル）	順序を持つ（シーケンス型）
int float bool	数値、真偽値	×	×	×
str	文字列	×	○	○
list	リスト	○	○	○
tuple	タプル	×	○	○
dict	辞書	○	○	×
set	セット	○	○	×

for文で使用可

インデックスで参照可

第5章 ユーザ定義関数

関数

関数とは

- 複数の命令文を1つにまとめて名前を付けたもの
- 何度も使う命令文の集まりを管理したり、プログラムコードの見通しをよくするのに便利な仕組み

- 組み込み関数：はじめから準備されている関数
  例： **print**関数、 **id**関数、**len**関数、**del**関数

- ユーザー定義関数：自分で新しく定義する関数

構文:関数の定義

```
def 関数名():
    処理内容
```

ユーザー定義関数

```
def say_hello():
    print('こんにちは')
    print('今日はよい天気ですね')
```

← say_hello 関数の定義

```
say_hello()
say_hello()
```

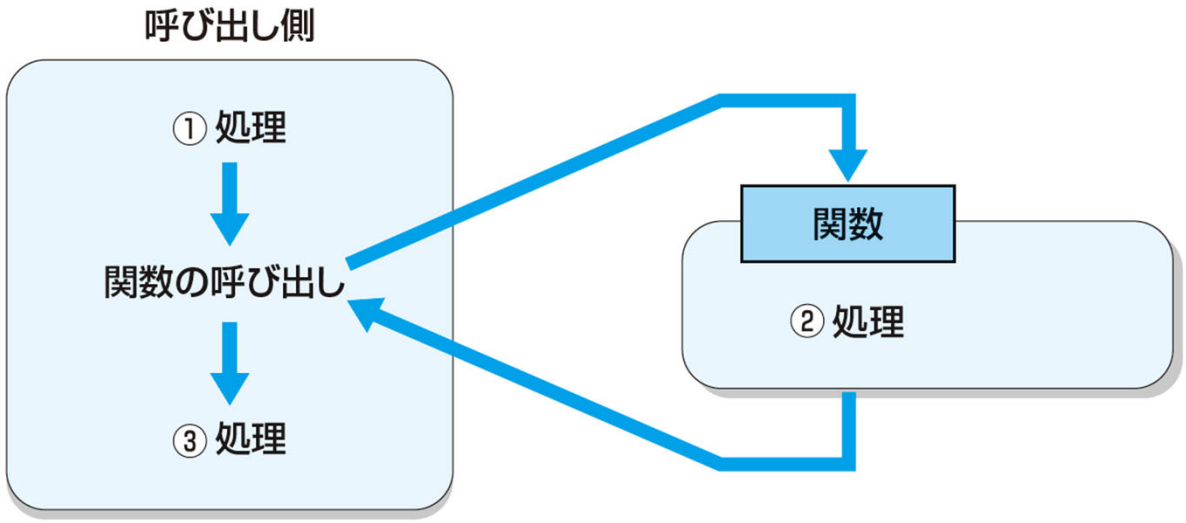
← say_hello 関数の呼び出し

← say_hello 関数の呼び出し

実行結果

```
こんにちは
今日はよい天気ですね
こんにちは
今日はよい天気ですね
```

関数呼び出しの処理の流れ




```
def function_a():
    print('function_aの処理です')

def function_b():
    print('function_bの処理です')

function_a()
function_b()
```

関数function_aの定義です

関数function_bの定義です

関数function_aを呼び出します

関数function_bを呼び出します

```
function_aの処理です
function_bの処理です
```

関数の定義位置

- 関数の定義は、関数の呼び出しの前に行われている必要がある

```
def function_a():
    print('function_aの処理です')

function_a()
function_b()

def function_b():
    print('function_bの処理です')
```

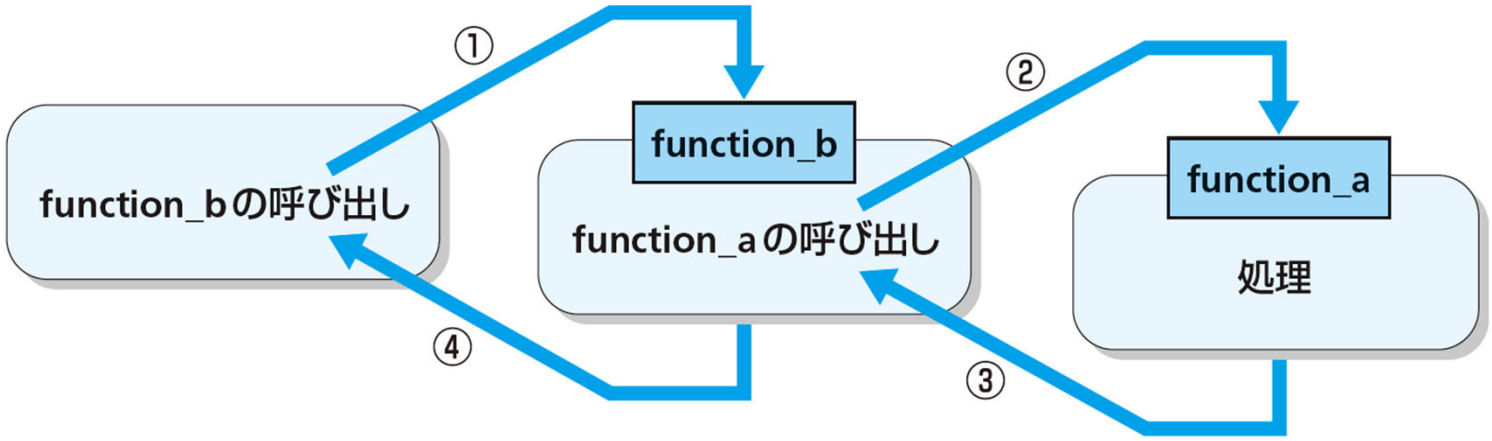
この時点ではfunction_bが定義されていないためエラーになります

関数の呼び出しの階層

```
def function_a():
    print('function_aの処理です')

def function_b():
    print('function_bの処理開始')
    function_a()
    print('function_bの処理終了')

print('function_bを呼び出します')
function_b()
```

実行結果

```
function_bを呼び出します
function_bの処理開始
function_aの処理です
function_bの処理終了
```



変数のスコープ

変数には参照できる範囲（スコープ）がある

ローカル変数：関数の中でのみ使用できる

```
def function_a():
    a = 10  ← 関数の中でのみ参照できる
    print(a)

print(a) ← エラーになる
```

グローバル変数：プログラムコード全体で使用できる

```
a = 10  ← プログラムコード全体で参照できる

def function_a():
    print(a)
```

変数のスコープ

- 関数の中でグローバル変数の値を変更したい場合は global キーワードを使用する

global宣言がある場合

```
a = 10

def function_a():
    global a
    a = 5  ←グローバル変数

function_a()
print(a)
```

実行結果

```
5
```

global宣言がない場合

```
a = 10

def function_a():
    a = 5 ←ローカル変数

function_a()
print(a)
```

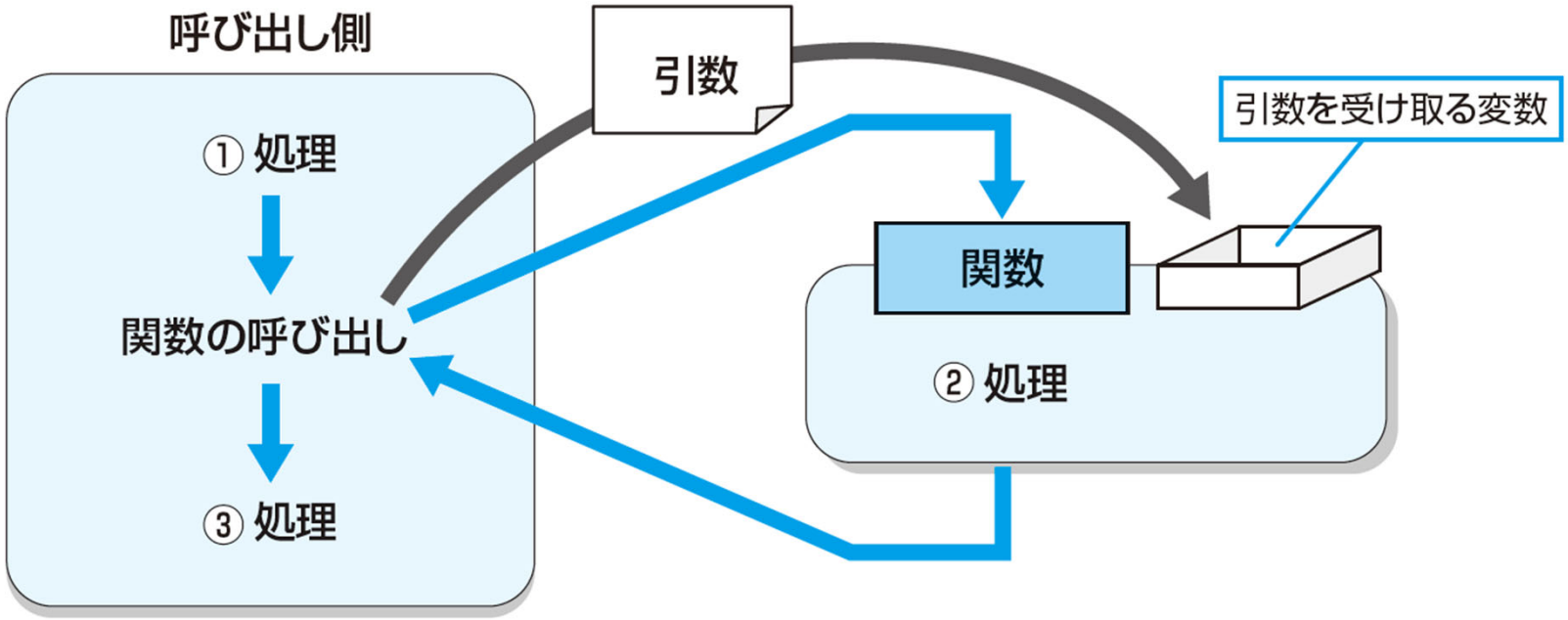
実行結果

```
10
```

関数の引数

引数

- 関数を呼び出すときに、関数に対して値を渡すことができる。
- 渡される値を**引数**という。



引数のある関数

関数の後ろのカッコ()
の中の変数で値を受け取る

```
def countdown(start):
    print('関数が受け取った値:', start)
    print('カウントダウンをします')
    counter = start
    while counter >= 0:
        print(counter)
        counter -= 1

countdown(3)
countdown(10)
```

受け取った値を使用する

値3を引数にして関数を呼び出す

値10を 〃

実行結果

```
関数が受け取った値： 3
カウントダウンをします
3
2
1
0
関数が受け取った値： 10
カウントダウンをします
10
9
(中略)
1
0
```

引数が2つある関数

```
def countdown(start, end):
    print('1つ目の引数で受け取った値:', start)
    print('2つ目の引数で受け取った値:', end)
    print('カウントダウンをします')
    counter = start
    while counter >= end:
        print(counter)
        counter -= 1

countdown(7, 3)
```

引数をカンマ区切りで並べる
（引数列とよぶ）

2つの値を引数にして関数を呼び出す

変数名を指定した呼び出し
（キーワード引数）

```
countdown(start=7, end=3)

countdown(end=3, start=7)
```

デフォルト引数

関数の定義

```
def countdown(start, end=0):
    処理内容
```

※ デフォルト値のある引数は、そうでない引数よりも引数列の後方に書く必要がある。

引数列に「**変数名=デフォルト値**」を記述しておくと、
関数を呼び出すときに、引数を省略できる

関数の呼び出し例（2番目の引数を省略できる）

```
countdown(10)
```

関数の呼び出し例（2つの引数を指定することもできる）

```
countdown(10, 0)
```

可変長引数

- 可変長引数：引数の数に制限がない

タプルで複数の値を受け取る

```
def 関数名(*変数名):
    処理内容
```

辞書で複数のキーと値のペアを受け取る

```
def 関数名(**変数名):
    処理内容
```

```
def func_a(*args):
    for a in args:
        print(a)

func_a(1, 2)
func_a(1, 2, 3, 4)
```

```
def func_b(**kwargs):
    for k, v in kwargs:
        print(k, v)

func_b(a=1, b=2)
func_b(c=3, d=4, e=5, f=6)
```

ドキュメントでの関数の書式

※ Python標準ライブラリのドキュメントで使用されている表記

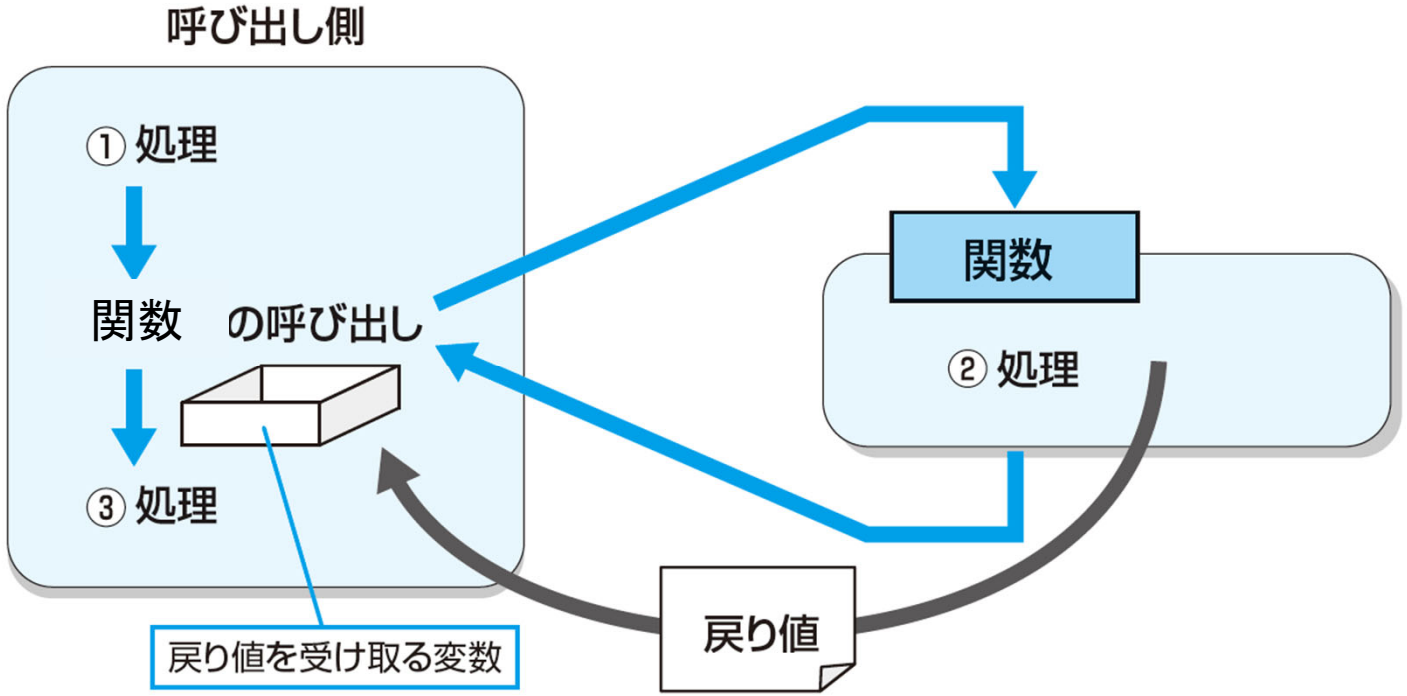
ドキュメントでの表記例	引数の数	説明
`func()`	0	引数なし
`func(x)`	1	引数xが必要
`func([x])`	0,1	引数xを省略可
`func(*x)`	0～	可変長引数。カンマ区切りでいくつでも指定できる
`func(x, y)`	2	引数xとyが必要
`func(x[, y])`	1,2	引数xが必要。引数yは省略可
`func([x[, y]])`	0,1,2	引数xとyが省略可 （xのみを省略することはできない）（注❺-6）
`func(x, y=1)`	1,2	引数xが必要。引数yは省略可 （yはデフォルト引数）
`func(x=1, y=2)`	0,1,2	引数xと引数yを省略可 （x、yの両方がデフォルト引数）
`func(x, *y)`	1～	引数xが必要。引数yは可変長引数
`func(x, y, z)`	3	引数xとyとzが必要
`func(x, y[, z])`	2,3	引数xとyが必要。引数zは省略可
`func(x[, y[, z]])`	1,2,3	引数xが必要。引数yとzは省略可 （yのみを省略することはできない）

[ ] 内の引数は省略できる

関数の戻り値

戻り値

- 関数で処理した結果の値を、呼び出し元に戻すことができる。
- 戻される値のことを「戻り値」という。



戻り値のある関数

- 値を戻すには、関数の中に

```
return 値
```

と記述する

```
def circle_area(r):
    return r * r * 3.14

a = circle_area(2.5)
print('半径2.5の円の面積は', a)
```

真偽値を戻り値とする関数

```
def is_positive(i):
    if i > 0:
        return True
    else:
        return False

a = -10;
if is_positive(a) == True:
    print('aの値は正です')
else:
    print('aの値は負またはゼロです')
```

```
def is_positive(i):
    return i > 0

a = -10;
if is_positive(a):
    print('aの値は正です')
else:
    print('aの値は負またはゼロです')
```

複数の値を戻す

- 複数の値を戻したい場合は、タプルを使用する

```
def get_two_numbers():
    return (2, 3)

a, b = get_two_numbers()
print(a, b)
```

高階関数とラムダ式

高階関数

- 関数もオブジェクト
- **高階関数**とは、関数を引数で受け取る関数

関数を受け取る

```
def print_price(price, func):
    print('価格は' + func(price))
```

受け取った関数を使う

```
def price_without_tax(price):
    return f'税抜き{price}円'

def price_with_tax(price):
    return f'税込み{int(price*1.1)}円'

print_price(800, price_without_tax)
print_price(800, price_with_tax)
```

関数を引数にしている

ラムダ式（lambda式）

- 関数の表記をシンプルにしたもの

構文

```
lambda 引数列： 戻り値
```

関数

```
def price_without_tax(price):
    return f'税抜き{price}円'
```

ラムダ式

```
lambda price: f'税抜き{price}円'
```

ラムダ式（lambda式）

```
def print_price(price, func):
    print('価格は' + func(price))

def price_without_tax(price):
    return f'税抜き{price}円'

def price_with_tax(price):
    return f'税込み{int(price*1.1)}円'

print_price(800, price_without_tax)
print_price(800, price_with_tax)
```

ラムダ式を使うことで簡潔に記述できる

```
def print_price(price, func):
    print('価格は' + func(price))

print_price(800, lambda price: f'税抜き{price}円')
print_price(800, lambda price: f'税込み{int(price*1.1)}円')
```

練習問題

問題 1

次の文章の空欄に入れるべき語句を、選択肢から選んでください。

- 関数を呼び出すときに関数に渡す値のことを[（1）]といい、関数から戻される値のことを[（2）]という。
- 関数に複数の値を渡すために、[（1）]をカンマ区切りで並べたものを[（3）]と呼ぶ。
- [（1）]のうち、変数の名前を指定したものを[（4）]、値が渡されなかった場合に設定されるデフォルト値が決まっているものを[（5）]、個数が決まっていないものを[（6）]という。

【選択肢】

・戻り値　・可変長引数　・引数　・キーワード引数

・デフォルト引数　・引数列

問題１（解答）

次の文章の空欄に入れるべき語句を、選択肢から選んでください。

- 関数を呼び出すときに関数に渡す値のことを[**引数**]といい、関数から戻される値のことを[**戻り値**]という。
- 関数に複数の値を渡すために、[**引数**]をカンマ区切りで並べたものを[**引数列**]と呼ぶ。
- [**引数**]のうち、変数の名前を指定したものを[**キーワード引数**]、値が渡されなかった場合に設定されるデフォルト値が決まっているものを[**デフォルト引数**]、個数が決まっていないものを[**可変長引数**]という。

【選択肢】

・戻り値　・可変長引数　・引数　・キーワード引数

・デフォルト引数　・引数列

問題 2

次のようなfunc関数が定義されています。

```
def func(a, b = 5):
    print(a, b)
```

次のうち、func関数を正しく呼び出せるものを選んでください。

（1） func()
（2） func(5)
（3） func(5, 10)
（4） func(a = 5)
（5） func(b = 10)
（6） func(5, b = 10)
（7） func(b = 10, a = 5)

問題２（解答）

次のようなfunc関数が定義されています。

```
def func(a, b = 5):
    print(a, b)
```

次のうち、func関数を正しく呼び出せるものを選んでください。

（1） func()
○（2） func(5)
○（3） func(5, 10)
○（4） func(a = 5)
（5） func(b = 10)
○（6） func(5, b = 10)
○（7） func(b = 10, a = 5)

問題 3

各問いの条件に合う関数を作成してください。プログラムコードには、その関数を呼び出す命令文を含めてください。関数に戻り値がある場合は、受け取った戻り値を出力するようにしてください。引数の値は自由に決めてかまいません。

例題

関数名： get_triangle_area

引数列： base, height

処理の内容： 底辺の長さがbase、高さがheightで表される三角形の面積を返す

解答例

```
def get_triangle_area(base, height):
    return base * height / 2

print(get_triangle_area(10.0, 3.0))
```

問題 3-1

関数名：　　print_hello

引数列：　　count

処理の内容：引数で渡されたcountの回数だけ、Helloという文字列を出力する。

※引数の値が3の場合はHelloという文字列を3回出力するようにする。

問題 3-1（解答）

関数名：　　print_hello

引数列：　　count

処理の内容：引数で渡されたcountの回数だけ、Helloという文字列を出力する。

※引数の値が3の場合はHelloという文字列を3回出力するようにする。

```
def print_hello(count):
    for i in range(0, count):
        print('Hello')

print_hello(3)
```

問題 3-2

関数名：　`get_rectangle_area`

引数列：　`width, height`

処理の内容：　引数で渡された幅（`width`）と高さ（`height`）の値を持つ長方形の面積を返す。

問題 3-2 （解答）

関数名：　　get_rectangle_area

引数列：　　width, height

処理の内容：　引数で渡された幅（width）と高さ（height）の値を持つ長方形の面積を返す。

```
def get_rectangle_area(width, height):
    return width * height

print(get_rectangle_area(10, 5))
```

問題 3-3

関数名：　`get_message`

引数列：　`name`

処理の内容：'こんにちは〇〇さん'という文字列を返す。〇〇には引数で渡された`name`の値を入れる。引数が指定されなかった場合は、'名無し'という文字列を`name`のデフォルト値とする。

問題 3-3 （解答）

関数名：　get_message

引数列：　name

処理の内容：'こんにちは○○さん'という文字列を返す。○○には引数で渡されたnameの値を入れる。引数が指定されなかった場合は、'名無し'という文字列をnameのデフォルト値とする。

```
def get_message(name='名無し'):
    return f'こんにちは{name}さん'

print(get_message())
print(get_message('山田'))
```

問題 3-4

関数名：　　`get_absolute_value`

引数列：　　`value`

処理の内容：　引数で渡された`value`の値の絶対値を返す。

※`5.2`の絶対値は`5.2`、`-3.3`の絶対値は`3.3`。

問題 3-4 （解答）

関数名：　　get_absolute_value

引数列：　　value

処理の内容：　引数で渡されたvalueの値の絶対値を返す。

※5.2の絶対値は5.2、-3.3の絶対値は3.3。

```
def get_absolute_value(value):
    if value < 0:
        return -value
    return value

print(get_absolute_value(5.2))
print(get_absolute_value(-3.3))
```

問題 3-5

関数名：　　`get_tail`

引数列：　　`*args`

処理の内容：　可変長引数で渡された複数の引数の中で、末尾の引数の値を返す。

※`get_tail(3, 5, 9, 2)` とすると、値2が返されるようにする。

問題 3-5 （解答）

関数名：　　get_tail

引数列：　　*args

処理の内容：　可変長引数で渡された複数の引数の中で、末尾の引数の値を返す。

※get_tail(3, 5, 9, 2)とすると、値2が返されるようにする。

```
def get_tail(*args):
    return args[-1]

print(get_tail(3, 5, 9, 2))
```

第6章 クラスの基本

新しいクラスを作る

クラスとは

- クラスとは、オブジェクトの種類のこと

  例： strクラス、listクラス、dictクラス

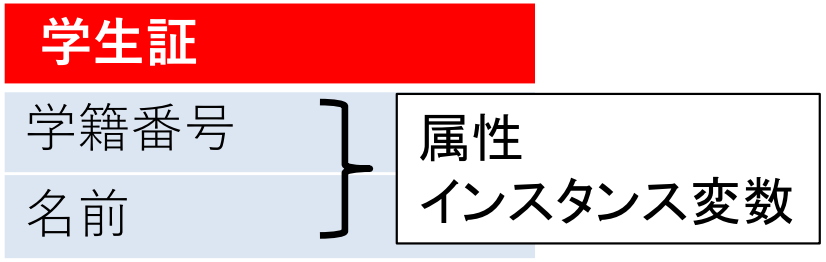
クラスはオブジェクトがどのような情報を持つか
どのような機能を持つかを定義する

メソッド

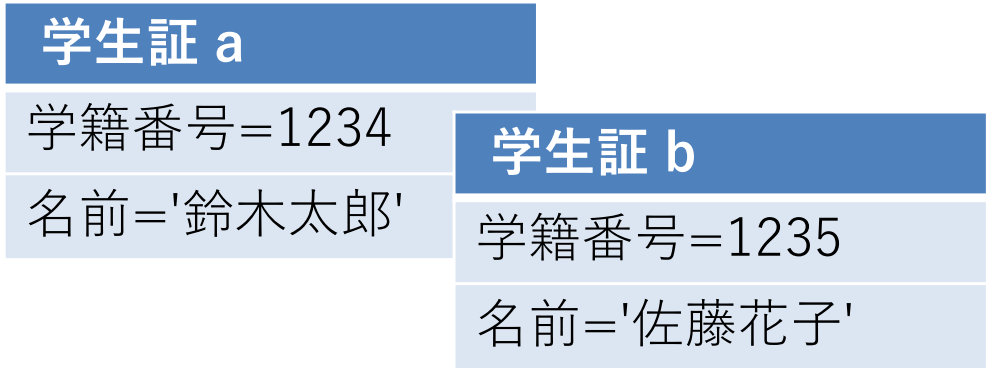
インスタンス変数

これから扱うクラスとインスタンスの例

クラス



インスタンス



クラスの定義

構文

```
class クラス名:
    初期化メソッドなどの定義
```

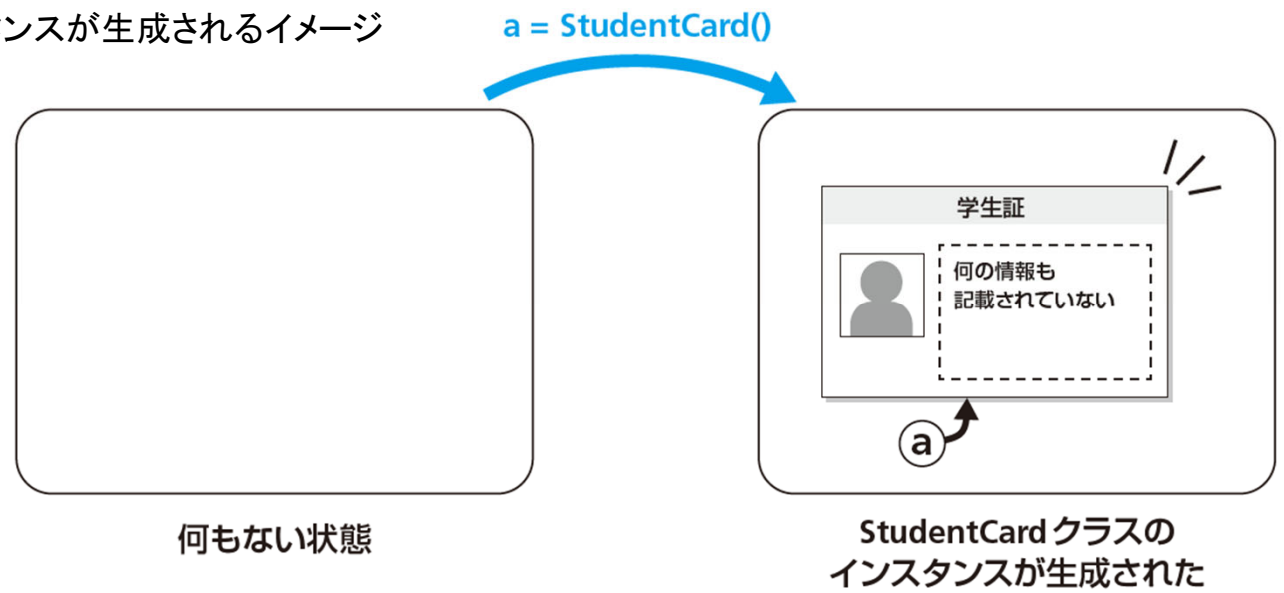
```
class StudentCard:
    pass

a = StudentCard()
```

StudentCardクラスの定義です

StudentCardクラスのインスタンスを生成して変数aに代入します

※ pass は何もしない命令

インスタンスが生成されるイメージ



初期化メソッド

- 初期化メソッド

  インスタンスが生成されるときに自動的に実行されるメソッド
  （これまでに学習した関数のようなもの）

構文

```
def __init__(self):
    処理内容
```

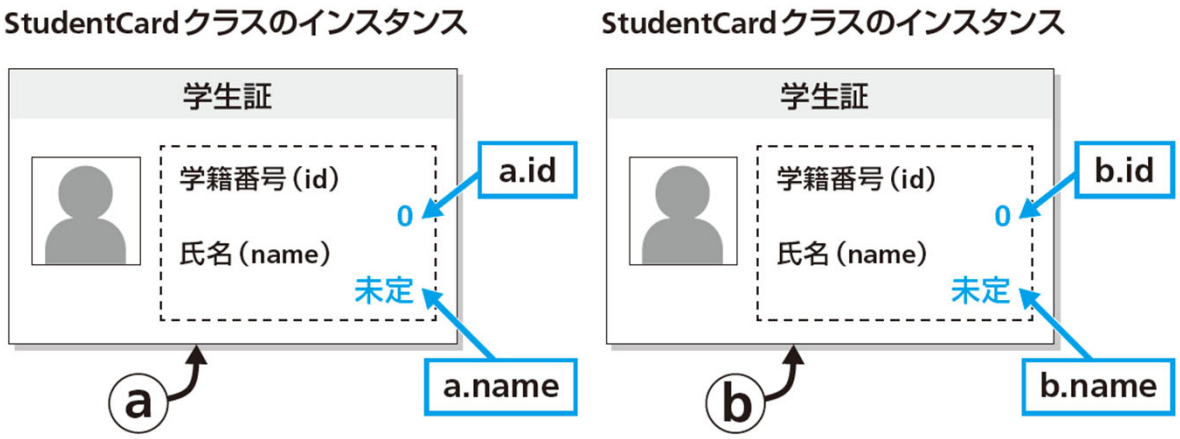
```
class StudentCard:
    def __init__(self):
        print('初期化メソッド内の処理です')
```

インスタンス変数

- インスタンス変数
  個々のインスタンスごとの値を持つ変数

```
class StudentCard:
    def __init__(self):
        print('初期化メソッド内の処理です')
        self.id = 0
        self.name = '未定'

a = StudentCard()
b = StudentCard()
```

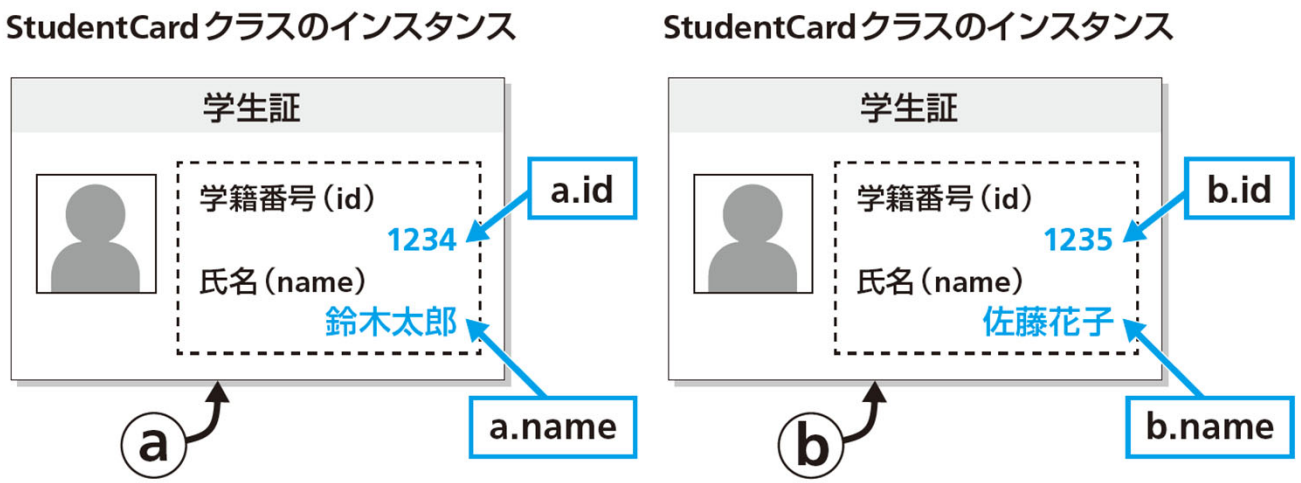
インスタンス変数



インスタンス変数の参照

```
class StudentCard:
    def __init__(self):
        print('初期化メソッド内の処理です')
        self.id = 0
        self.name = '未定'

a = StudentCard()
b = StudentCard()

a.id = 1234
a.name = '鈴木太郎'
b.id = 1235
b.name = '佐藤花子'
```

インスタンス変数の値を変更しています



初期化メソッドの引数

```
class StudentCard:
    def __init__(self, id, name):
        self.id = id
        self.name = name

a = StudentCard(1234, '鈴木太郎')
b = StudentCard(1245, '佐藤花子')
```

インスタンスを生成するときに、
2つの値を引数にしています

クラス変数とインスタンス変数

クラスの定義

クラス変数

```
class StudentCard:
    school_name = 'Python 大学'
    def __init__(self, id, name):
        self.id = id
        self.name = name
```

インスタンス変数

- クラス変数は、インスタンスの有無にかかわらず存在する
- インスタンス変数は、個々のインスタンスが所有する

クラス変数

StudentCard.school_name
= 'Python 大学'

インスタンス

id = 1234
name = '鈴木太郎'

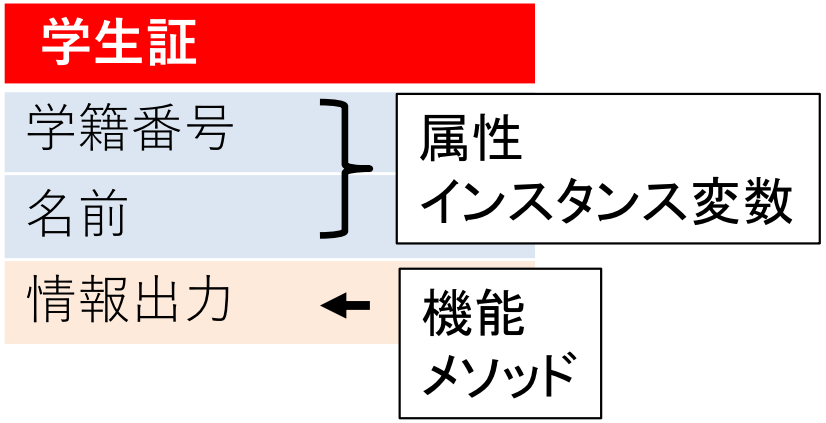
インスタンス

id = 1235
name = '佐藤花子'

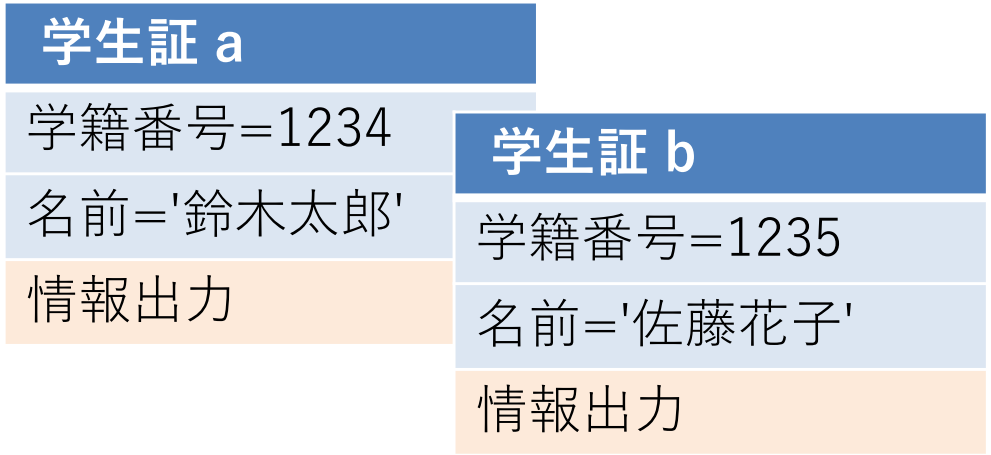
メソッドの定義

これから扱うクラスとインスタンスの例

クラス



インスタンス



メソッド

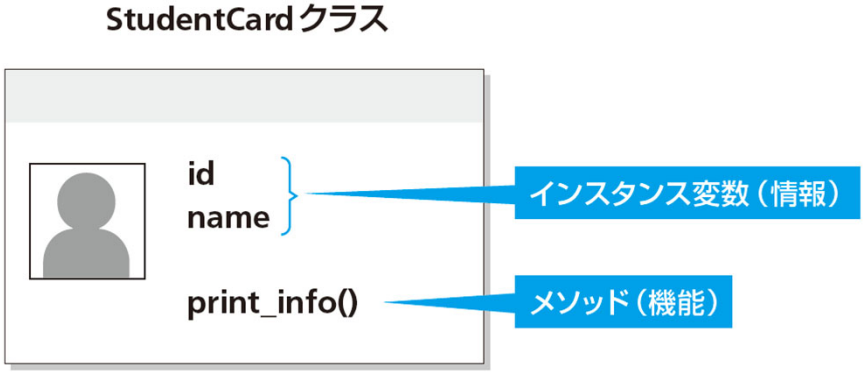
- メソッドとは

  インスタンスが行う命令文をまとめたもの（関数のようなもの）

  第一引数は自分自身を表す self

構文

```
def メソッド名(self, その他の引数):
    処理内容
```




```
class StudentCard:
    def __init__(self, id, name):
        self.id = id
        self.name = name

    def print_info(self):
        print('学籍番号:', self.id)
        print('氏名:', self.name)

a = StudentCard(1234, '鈴木太郎')
b = StudentCard(1235, '佐藤花子')
a.print_info()
b.print_info()
```

クラスメソッド

- クラスメソッドとは

  インスタンスを生成しなくても

  クラス名.メソッド名() で実行できるメソッド

  第一引数は自分のクラスを表す cls

```
class SimpleCalc:
    @classmethod
    def get_triangle_area(cls, base, height):
        return base * height / 2

print(SimpleCalc.get_triangle_area(10, 5))
```

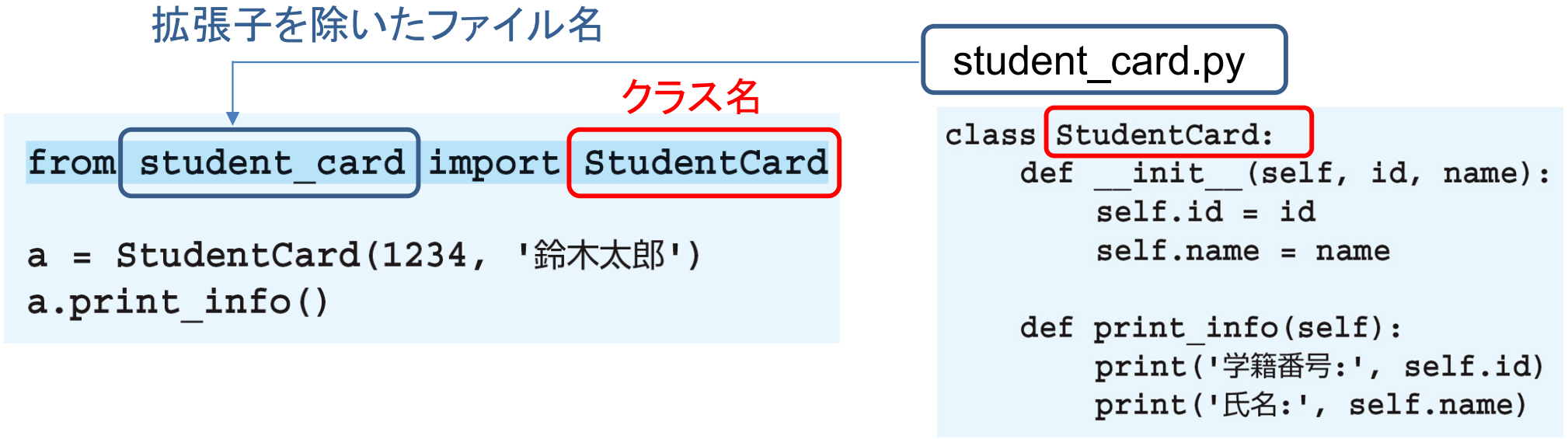
続くメソッドの定義がクラスメソッドであることを知らせるためのデコレーターです

クラスメソッドの定義です

クラスメソッドを呼び出しています

オリジナルのクラスをモジュールとして利用する

クラスを定義したファイルをモジュールとして利用できる



__name__変数

name_card.py

```
class NameCard:
    def __init__(self, name):
        self.name = name

a = NameCard('鈴木太郎')
print(a.name)
```


このファイルをインポートすると
実行されてしまう

name_card.py

```
class NameCard:
    def __init__(self, name):
        self.name = name

if __name__ == '__main__':
    a = NameCard('鈴木太郎')
    print(a.name)
```

インポート時には実行されなくなる

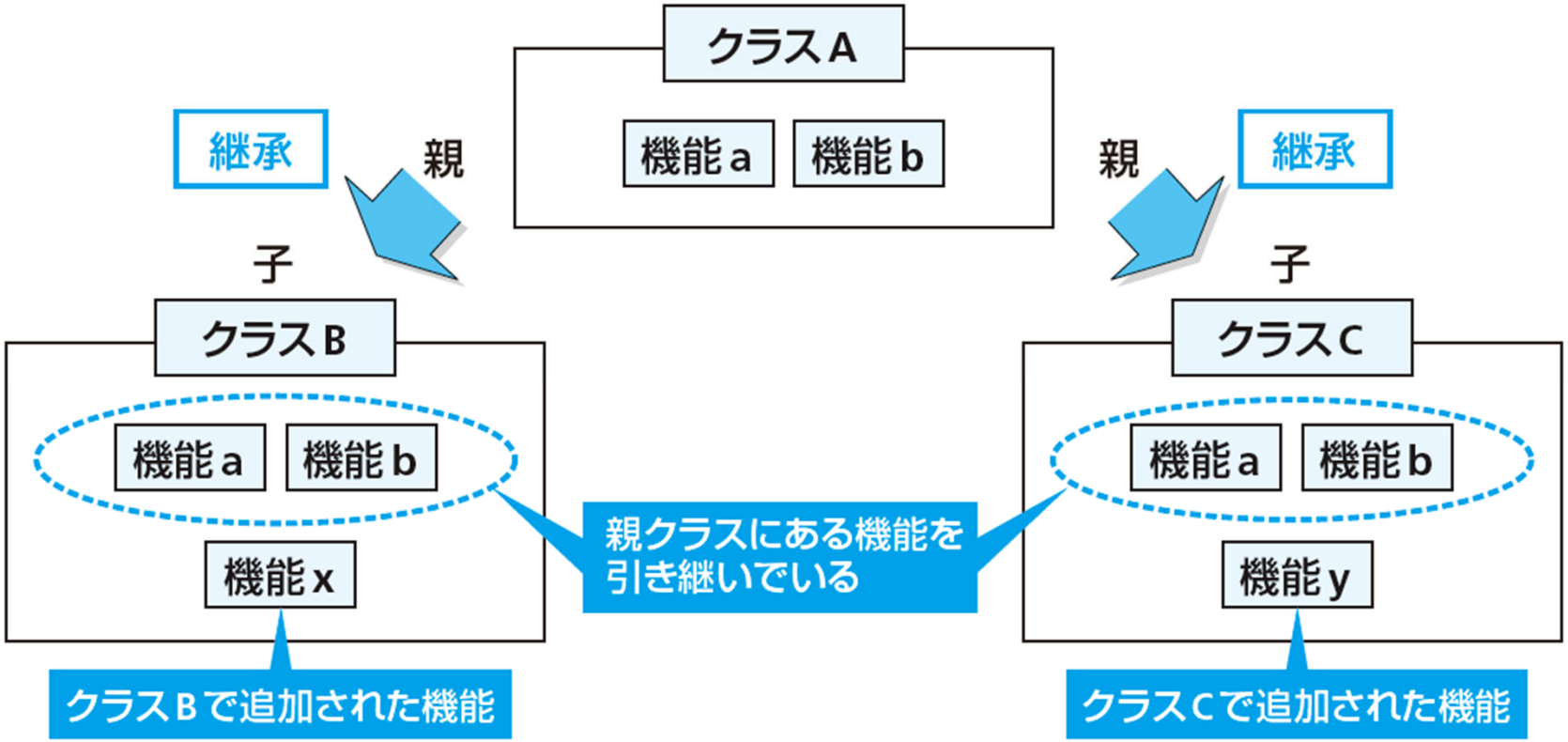
__name__変数の値
- プログラムコードが直接実行されたとき → **'__main__'**
- インポートされたとき → モジュール名 （この場合は'name_card'）

継承

継承

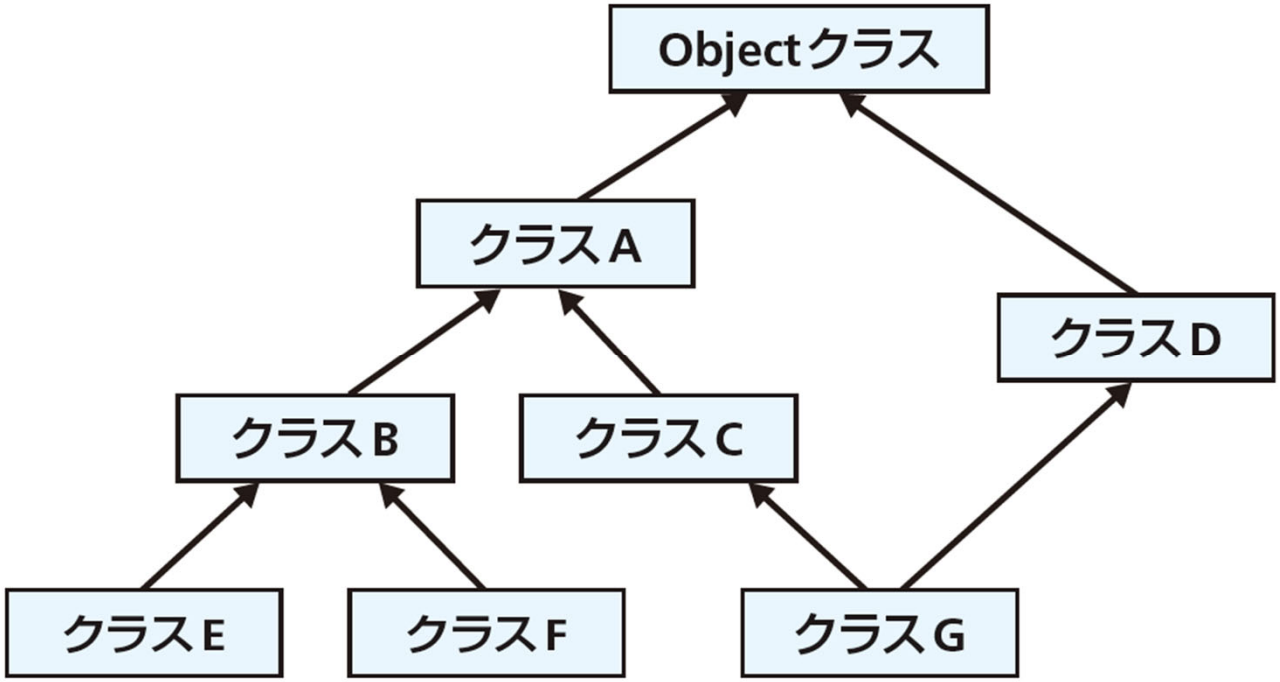
- 継承とは

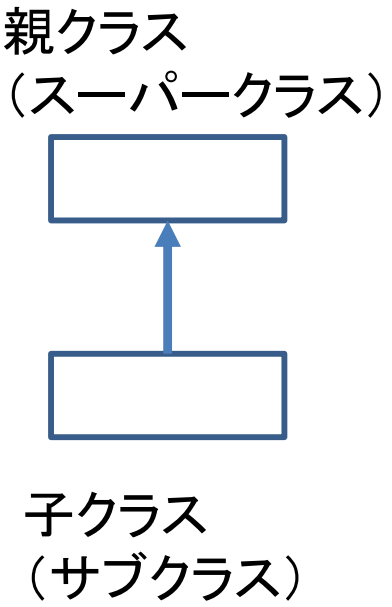
  既存のクラスの機能を再利用し、それを拡張することで新しいクラスを作成すること



Python の継承

- 全てのクラスがObjectクラスを直接または間接的に継承する
- 1つのクラスが2つのクラスを継承できる（**多重継承**）



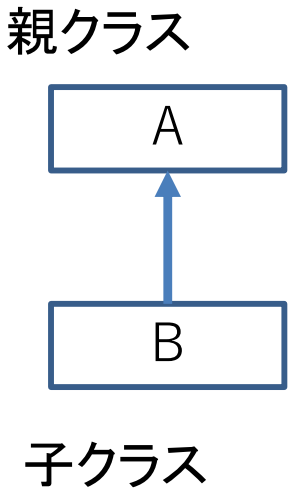


継承のしかた

構文

```
class クラス名(親クラス名):
    クラスの定義
```

```
class A:
    クラスの定義

calss B(A):
    クラスの定義
```

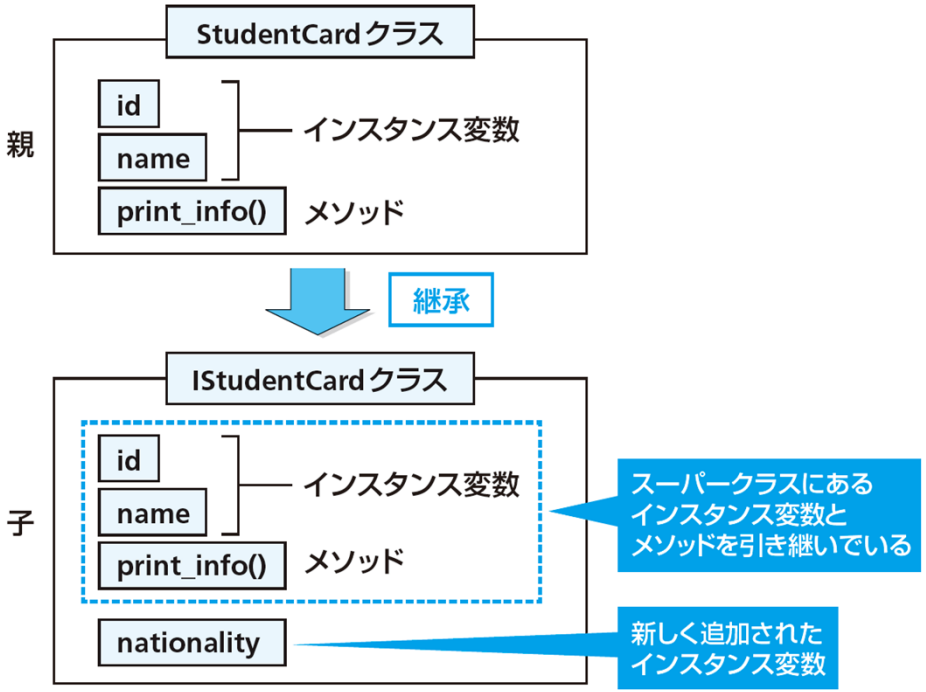


インスタンス変数とメソッドの継承

student_card.py

```
class StudentCard:
    def __init__(self, id, name):
        self.id = id
        self.name = name

    def print_info(self):
        print('学籍番号:', self.id)
        print('氏名:', self.name)
```

```
from student_card import StudentCard

class IStudentCard(StudentCard):
    def __init__(self, id, name, nationality):
        self.nationality = nationality
        super().__init__(id, name)
```



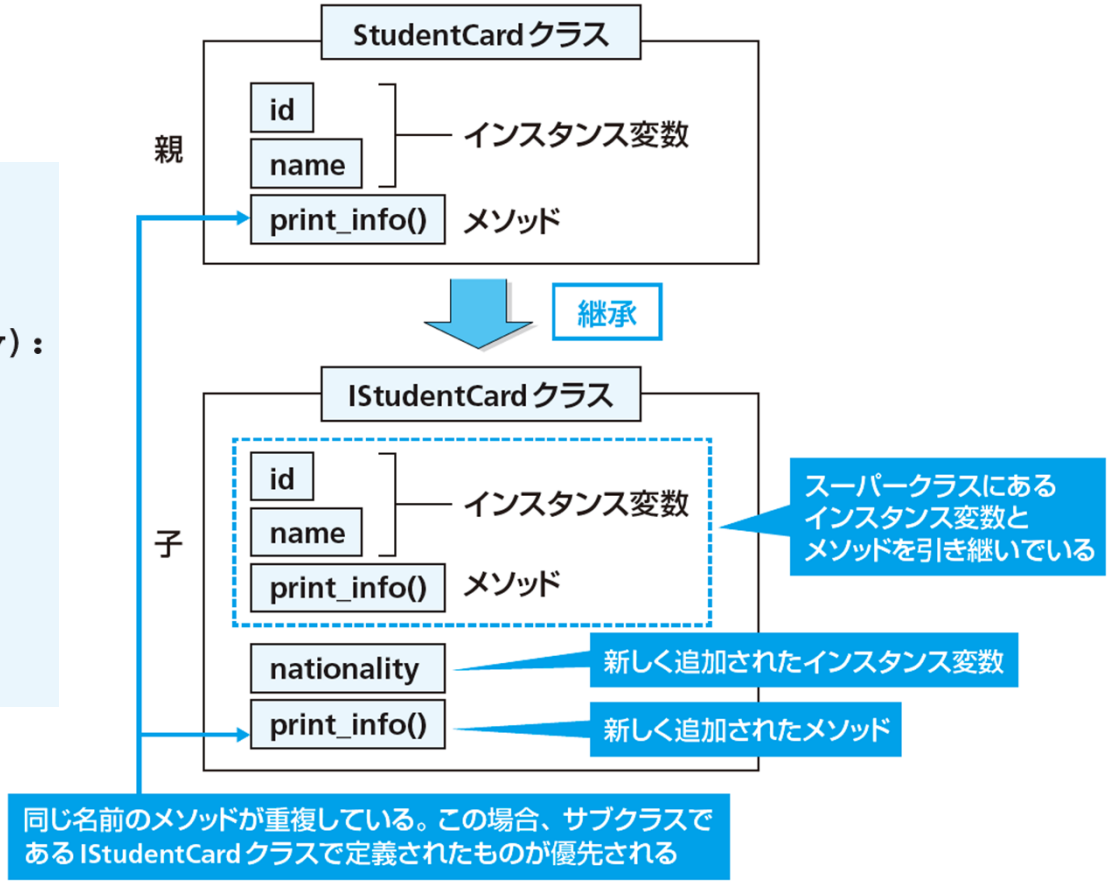
メソッドのオーバーライド

- オーバーライドとは

  スーパークラスと同じ名前のメソッドをサブクラスで定義すること。スーパークラスのメソッドはサブクラスのメソッドで上書きされる。

```
from student_card import StudentCard

class IStudentCard(StudentCard):
    def __init__(self, id, name, nationality):
        self.nationality = nationality
        super().__init__(id, name)

    def print_info(self):
        print(f'国籍: {self.nationality}')
        print(f'学籍番号: {self.id}')
        print(f'氏名: {self.name}')
```

親クラスも同じ名前のメソッドを持っている



親クラスのメソッドの呼び出し

super().メソッド名()
とすることで、子クラスの中から、親クラスのメソッドを呼び出すことができる

```
from student_card import StudentCard

class IStudentCard(StudentCard):
    def __init__(self, id, name, nationality):
        self.nationality = nationality
        super().__init__(id, name)

    def print_info(self):
        print(f'国籍: {self.nationality}')
        super().print_info()
```

練習問題

問題 1

次の文章の空欄に入れるべき語句を、選択肢から選んでください。

- Pythonは[（1）]指向型の言語といわれ、クラスは[（1）]の属性や機能を定義したものです。
- クラスの定義の中で[（1）]の持つ情報は[（2）]に持たせることができ、機能は[（3）]に持たせることができます。
- インスタンスが生成されるときに自動的に呼び出されるメソッドを [（4）]またはコンストラクタと呼びます。

【選択肢】

・クラス変数　・インスタンス変数　・関数

・オブジェクト　・メソッド　・初期化メソッド

問題１（解答）

次の文章の空欄に入れるべき語句を、選択肢から選んでください。

- Pythonは[**オブジェクト**]指向型の言語といわれ、クラスは[**オブジェクト**]の属性や機能を定義したものです。
- クラスの定義の中で[**オブジェクト**]の持つ情報は[**インスタンス変数**]に持たせることができ、機能は[**メソッド**]に持たせることができます。
- インスタンスが生成されるときに自動的に呼び出されるメソッドを[**初期化メソッド**]またはコンストラクタと呼びます。

【選択肢】

・クラス変数　・インスタンス変数　・関数

・オブジェクト　・メソッド　・初期化メソッド

問題 2

空欄（A）に、問いの条件に合う関数を追加してください。空欄（B）には、その関数を呼び出す命令文を記述してください。なお、関数に戻り値がある場合は、受け取った戻り値を出力するようにしてください。引数の値は自由に決めてかまいません。

```
# 個人の情報を表すクラス
class Person:
    def __init__(self, name, age):
        self.name = name   # 名前
        self.age = age     # 年齢

# ここに設問の関数を追加する
(A)

a = Person("高橋太郎", 19)
b = Person("小林花子", 18)

# 追加した関数を呼び出す。戻り値がある場合は出力する
(B)
```

問題 2-1

```
class Person:
    def __init__(self, name, age):
        self.name = name   # 名前
        self.age = age     # 年齢

# ここに設問の関数を追加する
```
(A)

```
a = Person("高橋太郎", 19)
b = Person("小林花子", 18)

# 追加した関数を呼び出す。戻り値がある場合は出力する
```
(B)

関数名：　print_info

引数列：　p（Personオブジェクト）

処理の内容：引数で受け取るPersonオブジェクトの、名前と年齢の情報を出力する。

問題 2-1

（解答）

```
class Person:
    def __init__(self, name, age):
        self.name = name   # 名前
        self.age = age     # 年齢

def print_info(p):
    print('名前', p.name)
    print('年齢', p.age)

a = Person("高橋太郎", 19)
b = Person("小林花子", 18)

# 追加した関数を呼び出す。戻り値がある場合は出力する
print_info(a)
```

関数名：　print_info

引数列：　p（Personオブジェクト）

処理の内容：引数で受け取るPersonオブジェクトの、名前と年齢の情報を出力する。

問題 2-2

```
class Person:
    def __init__(self, name, age):
        self.name = name   # 名前
        self.age = age     # 年齢

# ここに設問の関数を追加する
(A)

a = Person("高橋太郎", 19)
b = Person("小林花子", 18)

# 追加した関数を呼び出す。戻り値がある場合は出力する
(B)
```

関数名：　age_check

引数列：　p（Personオブジェクト）、i（整数値）

処理の内容：引数で受け取るPersonオブジェクトの年齢が、引数iの値を超えている場合はTrueを、そうでない場合はFalseを返す。

問題 2-2

（解答）

```
class Person:
    def __init__(self, name, age):
        self.name = name   # 名前
        self.age = age     # 年齢

def age_check(p, i):
    return p.age > I


a = Person("高橋太郎", 19)
b = Person("小林花子", 18)

# 追加した関数を呼び出す。戻り値がある場合は出力する
print(age_check(a, 20))
```

関数名：　age_check

引数列：　p（Personオブジェクト）、i（整数値）

処理の内容：引数で受け取るPersonオブジェクトの年齢が、引数iの値を超えている場合はTrueを、そうでない場合はFalseを返す。

問題 2-3

```
class Person:
    def __init__(self, name, age):
        self.name = name   # 名前
        self.age = age     # 年齢

# ここに設問の関数を追加する
(A)

a = Person("高橋太郎", 19)
b = Person("小林花子", 18)

# 追加した関数を呼び出す。戻り値がある場合は出力する
(B)
```

関数名：　print_younger_person_name

引数列：　p1（Personオブジェクト）、p2（Personオブジェクト）

処理の内容：引数で受け取る2つのPersonオブジェクトのうち、年齢の若いほうの名前を出力する。ただし、同じ年齢の場合はp1の名前を出力する。

問題 2-3

（解答）

```
class Person:
    def __init__(self, name, age):
        self.name = name   # 名前
        self.age = age     # 年齢
def print_younger_person_name(p1, p2):
    if p1.age <= p2.age:
        print(p1.name)
    else:
        print(p2.name)
a = Person("高橋太郎", 19)
b = Person("小林花子", 18)

# 追加した関数を呼び出す。戻り値がある場合は出力する
print_younger_person_name(a, b)
```

関数名：　print_younger_person_name

引数列：　p1（Personオブジェクト）、p2（Personオブジェクト）

処理の内容：引数で受け取る2つのPersonオブジェクトのうち、年齢の若いほうの名前を出力する。ただし、同じ年齢の場合はp1の名前を出力する。

問題 2-4

```
class Person:
    def __init__(self, name, age):
        self.name = name   # 名前
        self.age = age     # 年齢

# ここに設問の関数を追加する
(A)

a = Person("高橋太郎", 19)
b = Person("小林花子", 18)

# 追加した関数を呼び出す。戻り値がある場合は出力する
(B)
```

関数名：　get_total_age

引数列：　p1（Personオブジェクト）、p2（Personオブジェクト）

処理の内容：引数で受け取る2つのPersonオブジェクトの、年齢の合計を返す。

問題 2-4

（解答）

```
class Person:
    def __init__(self, name, age):
        self.name = name   # 名前
        self.age = age     # 年齢

def get_total_age(p1, p2):
    return p1.age + p2.age


a = Person("高橋太郎", 19)
b = Person("小林花子", 18)

# 追加した関数を呼び出す。戻り値がある場合は出力する
print(get_total_age(a, b))
```

関数名：　get_total_age

引数列：　p1（Personオブジェクト）、p2（Personオブジェクト）

処理の内容：引数で受け取る2つのPersonオブジェクトの、年齢の合計を返す。

問題 3

空欄（A）に、次の条件に合うメソッドを追加してください。

```
# 長方形を表すクラス
class Rectangle:
    def __init__(self, width, height):
        self.width = width       # 幅
        self.height = height  # 高さ

    # ここに設問のメソッドを追加する
    (A)

rec0 = Rectangle(5, 8)
rec1 = Rectangle(4, 6)

print('rec0の面積', rec0.get_area())
print('rec1の面積', rec1.get_area())
```

条件

メソッド名： `get_area`

引数列： なし

処理の内容： 面積（幅×高さ）を返す

問題３（解答）

空欄（A）に、次の条件に合うメソッドを追加してください。

```
# 長方形を表すクラス
class Rectangle:
    def __init__(self, width, height):
        self.width = width       # 幅
        self.height = height  # 高さ

    def get_area(self):
        return self.width * self.height

rec0 = Rectangle(5, 8)
rec1 = Rectangle(4, 6)

print('rec0の面積', rec0.get_area())
print('rec1の面積', rec1.get_area())
```

条件

メソッド名：　get_area

引数列：　なし

処理の内容：面積（幅×高さ）を返す

問題 4

```
class X:
    def __init__(self):
        print('[x]')

    def a(self):
        print('[x.a]')

    def b(self):
        print('[x.b]')

class Y(X):
    def __init__(self):
        super().__init__()
        print('[y]')

    def a(self):
        print('[y.a]')
        super().a()
```

以下の処理で、それぞれどのような出力が得られるか答えてください。

```
x = X()      # (1)
x.a()        # (2)
x.b()        # (3)
y = Y()      # (4)
y.a()        # (5)
y.b()        # (6)
```

問題 4（解答）

```
class X:
    def __init__(self):
        print('[x]')

    def a(self):
        print('[x.a]')

    def b(self):
        print('[x.b]')

class Y(X):
    def __init__(self):
        super().__init__()
        print('[y]')

    def a(self):
        print('[y.a]')
        super().a()
```

以下の処理で、それぞれどのような出力が得られるか答えてください。

```
x = X()         # (1)  [x]
x.a()           # (2)  [x.a]
x.b()           # (3)  [x.b]
y = Y()         # (4)  [x]
                       [y]
y.a()           # (5)  [y.a]
                       [x.a]
y.b()           # (6)  [x.b]
```

第7章 発展と応用

例外処理

例外

- 例外とは
  プログラム実行時に発生したトラブルやそれを表すもの。

```
print('a ÷ b の計算をします')
a = input('aの値を入力してください: ')
b = input('bの値を入力してください: ')
c = float(a) / float(b)
print('答えは', c)
```

入力された値が、数字ではない場合、bの値がゼロの場合、エラーが発生する

```
a ÷ b の計算をします
aの値を入力してください: abc
bの値を入力してください: def
Traceback (most recent call last):
  File "divide.py", line 4, in <module>
    c = float(a) / float(b)
ValueError: could not convert string to float: 'abc'
```

数字でないものを入力しています

例外処理

構文：try～except 文

```
try:
    本来実行したい処理（例外が発生する可能性がある）
except:
    例外が発生したときの処理
```

```
print('a ÷ b の計算をします')
try:
    a = input('aの値を入力してください： ')
    b = input('bの値を入力してください： ')
    c = float(a) / float(b)
    print('答えは', c)
except:
    print('入力が適切ではありません')

print('処理を終わります')
```

例外の種類による処理の切り替え

構文：try～except 文

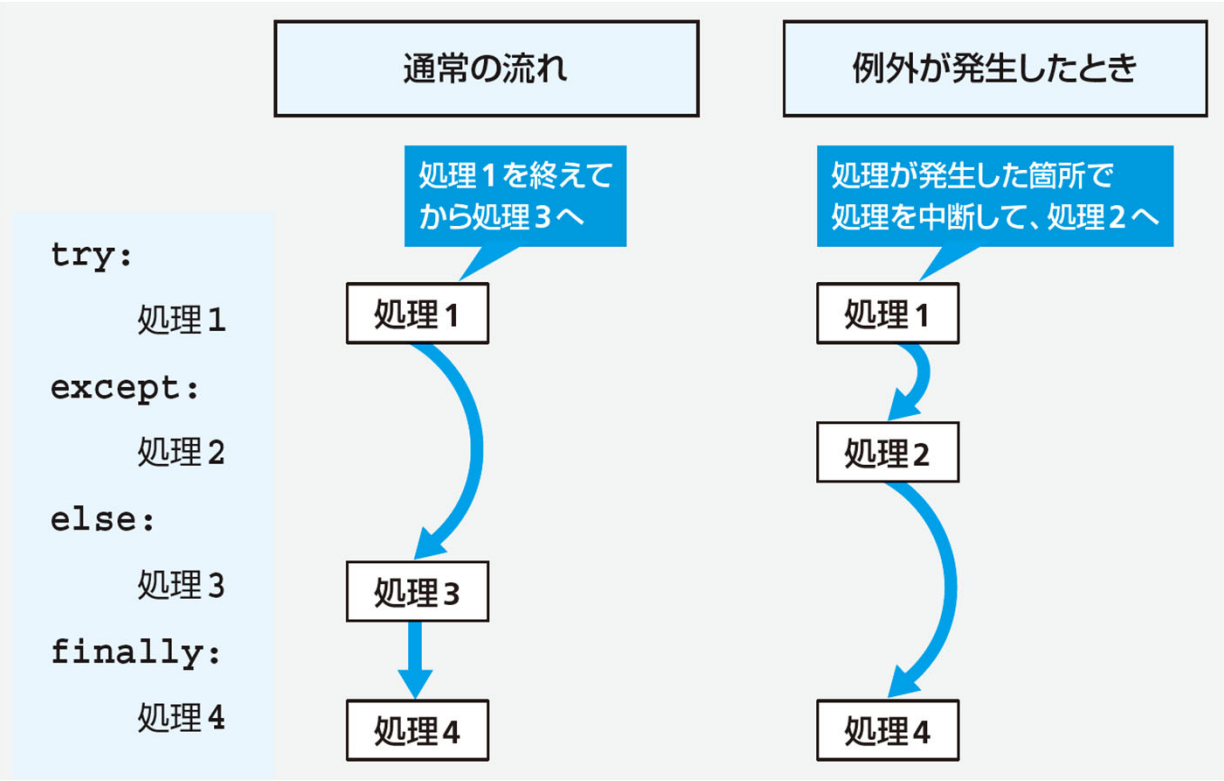
```
try:
    本来実行したい処理（例外が発生する可能性がある）
except 例外の種類1:
    例外（種類1）が発生したときの処理
except 例外の種類2:
    例外（種類2）が発生したときの処理
```

```
print('a ÷ b の計算をします')
try:
    a = input('aの値を入力してください: ')
    b = input('bの値を入力してください: ')
    c = float(a) / float(b)
    print('答えは', c)
except ValueError:
    print('入力が数字ではありません')
except ZeroDivisionError:
    print('ゼロで割ることはできません')

print('処理を終わります')
```

例外が発生しなかった場合の処理

構文：try～except～else～finally 文

```
try:
    例外が発生するかもしれない処理
except:
    例外が発生したときの処理
else:
    例外が発生しなかったときの処理
finally:
    例外の有無にかかわらず実行される処理
```



テキストファイルの読み書き

テキストファイル

テキストファイルとは

人が読んで理解することができる文字の集まり
（テキストファイルではないものを**バイナリファイル**という）
各種データを保存するためのファイルで広く使われている

テキストファイルの例

visitor_record.txt

```
2021-07-01,東京都,1,0
2021-07-01,千葉県,2,1
2021-07-01,千葉県,2,2
2021-07-01,神奈川県,4,2
2021-07-02,福島県,2,0
2021-07-02,埼玉県,3,2
2021-07-02,埼玉県,4,2
（略）
```

データをカンマ区切りで記述する形式を**CSV形式**とよぶ

1行分のデータを1つの**レコード**とよぶ

テキストファイルの読み込み

open関数 :ファイルの読み書きをする

```
open(ファイルパス, mode='r', encoding=None)
```

モード	説明
r	読み込み用に開く（ファイルが存在しないときはエラー）
w	書き込み用に開く（ファイルが存在しないときは新規作成）
a	追記用に開く（ファイルの末尾に追記。ファイルが存在しないときは新規作成）
+	読み書き両用にする
b	バイナリモードで開く

ファイルを1行ずつ読む

```
f = open(ファイルパス, 'r', encoding='UTF-8')
lines = f.readlines()

    for l in lines:
        print(l)

f.close()
```

with文を使ったファイルの読み込み

構文

```
with open(引数列) as 変数名:
```

with文を使ってファイルを1行ずつ読み込む例

```
with open(ファイルパス, 'r', encoding='UTF-8') as f:

    for l in f:
        print(l)
```

一度に全部をメモリに取り込まないので、サイズが大きなファイルを扱える

テキストファイルの書き出し

```
f = open(ファイルパス, 'w', encoding='UTF-8')

for i in range(0,100):
    f.write(str(i) + '¥n')

close(f)
```

改行も書き出す必要がある

with文を使う場合

```
with open(ファイルパス, 'w', encoding='UTF-8') as f:

for i in range(0,100):
    f.write(str(i) + '¥n')
```

テキストファイルの書き出し

with文を使う場合

7-08/file_write.py

```
with open(ファイルパス, 'w', encoding='UTF-8') as f:

for i in range(0,100):
    f.write(str(i) + '¥n')
```

改行も書き出す必要がある

ファイルの読み書き

with文を使って、出力先のファイルと読み込むファイルの両方を開く
入力ファイルを1行ずつ読み込み、'東京都'という文字列が含まれる場合は出力ファイルに書き出す

```
with open(出力ファイルのパス, 'w', encoding='UTF-8') as out_f:
    with open(入力ファイルのパス, 'r', encoding='UTF-8') as in_f:
        for line in in_f:
            if '東京都' in line:
                out_f.write(line)
```

実行結果（出力ファイルの中身）

```
2021-07-01,東京都,1,0
2021-07-03,東京都,2,1
2021-07-03,東京都,4,2
(略)
```

データの集計とグラフ描画

データの読み込み

都道府県と来館者数の組を、来館者の多い順にソートして出力する

```
pref_count_dict = {}
with open('data/visitor_record.txt', 'r', encoding='UTF-8') as f:
    for line in f:
        date, pref, num_adult, num_children = line.split(',')
        num_all = int(num_adult) + int(num_children)
        if pref in pref_count_dict:
            pref_count_dict[pref] += num_all
        else:
            pref_count_dict[pref] = num_all

pref_count_sorted = sorted(pref_count_dict.items(), key=lambda x:x[1], reverse=True)

for i in pref_count_sorted:
    print(i)
```

7-10/pref_count.py

visitor_record.txt

```
2021-07-01,東京都,1,0
2021-07-01,千葉県,2,1
2021-07-01,千葉県,2,2
2021-07-01,神奈川県,4,2
2021-07-02,福島県,2,0
2021-07-02,埼玉県,3,2
2021-07-02,埼玉県,4,2
…
```

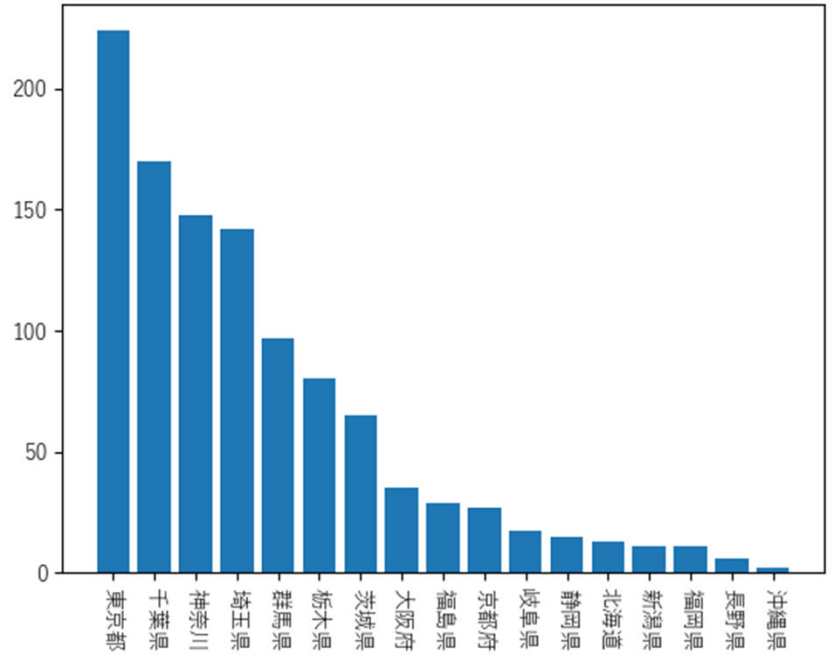
※ CSV形式（1行＝1レコード）

出力

```
('東京都', 224)
('千葉県', 170)
('神奈川県', 148)
('埼玉県', 142)
（略）
('長野県', 6)
('沖縄県', 2)
```

matplotlibライブラリを用いたグラフの作成

- matplotlibライブラリとは
  グラフ描画用の機能をまとめたライブラリ
- 公式サイト　https://matplotlib.org/



外部ライブラリ

- 外部ライブラリとは
  - Pythonに最初から備わっている「標準ライブラリ」とは別に、外部で開発されたライブラリで、必要に応じてインストールが必要
  - 画像処理、機械学習、グラフ生成、最適化計算など、特定の分野に特化した機能を提供
  - matplotlib は、外部ライブラリの1つ。グラフ描画用の機能が含まれる

pip コマンド

・外部ライブラリのインストール・管理を行うコマンド

・Windows PowerShell で「 pip list 」と入力することでインストール済みのライブラリ一覧を得られる

```
> pip list
Package          Version
---------------- ---------
beautifulsoup4   4.9.3
certifi          2020.11.8
chardet          3.0.4
cycler           0.10.0
idna             2.10
kiwisolver       1.3.1
matplotlib       3.3.3
(略)
```

・「 pip install *ライブラリ名* 」と入力することでライブラリをインストールできる

```
> pip install matplotlib
```

7-11/make_graph.py

```
 1: import matplotlib
 2: import matplotlib.pyplot as plt
 3:
 4: pref_count_dict = {}
 5: with open('data/visitor_record.txt', 'r', encoding='UTF-8'
) as f:
 6:     for line in f:
 7:         date, pref, num_adult, num_children = line.split(
',')
 8:         num_all = int(num_adult) + int(num_children)
 9:         if pref in pref_count_dict:
10:             pref_count_dict[pref] += num_all
11:         else:
12:             pref_count_dict[pref] = num_all
13:
14: pref_count_sorted = sorted(pref_count_
dict.items(), key=lambda x:x[1], reverse=True)
15:
16: labels = []
17: values = []
18: for l, v in pref_count_sorted:
19:     labels.append(l)
20:     values.append(v)
21:
22: matplotlib.use('Agg')
23: plt.rcParams['font.family'] = 'Yu Gothic'
24: plt.xticks(rotation=270)
25: plt.bar(range(0, len(pref_count_sorted)),
 values, tick_label=labels)
26: plt.savefig('graph.png')
```

①必要なライブラリをインポートします

都道府県名と訪問者数のタプルを、訪問者数でソートします

②グラフのラベル（都道府県名）を格納するためのリスト

③グラフの値（訪問者数）を格納するためのリスト

ソート順にラベルと値を取り出します

取り出したラベルと値をリストに格納します

④グラフをファイルに出力するために必要な記述です

⑤フォントの設定

⑥ラベルを縦書きにします

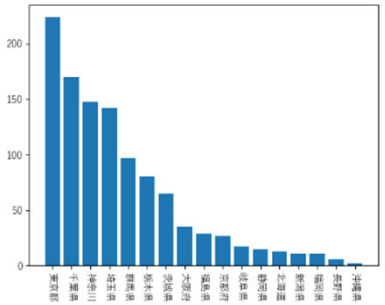
⑦グラフを作成します

⑧グラフを画像として保存します

データ読み込み

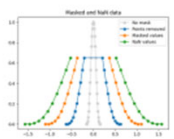
グラフ作成準備

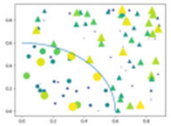
グラフ出力



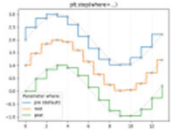
さまざまなグラフ

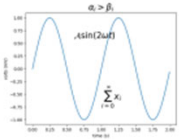
Examples using `matplotlib.pyplot.plot`

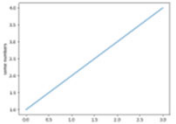

Plotting masked and NaN values

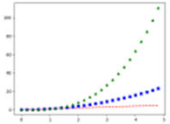

Scatter Masked

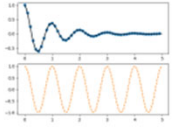

Stairs Demo

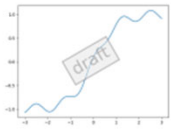

Step Demo


Pyplot Mathtext

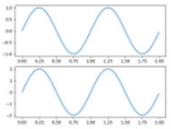

Pyplot Simple

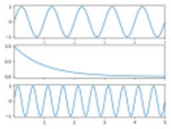

Pyplot Three

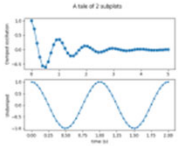

Pyplot Two Subplots

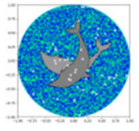


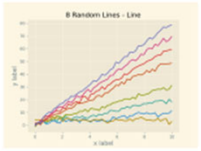
Custom Figure subclasses


Managing multiple figures in pyplot

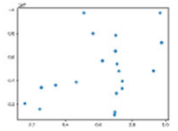

Shared Axis

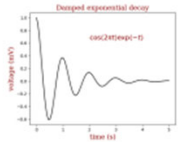

Multiple subplots

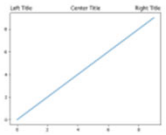

Dolphins

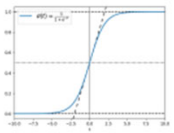

Solarized Light stylesheet

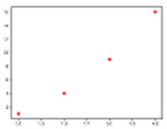
Frame grabbing


Coords Report


Controlling style of text and labels using a dictionary


Title positioning


Infinite lines


plot() format string

https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot

画像処理

OpenCVを用いた画像処理

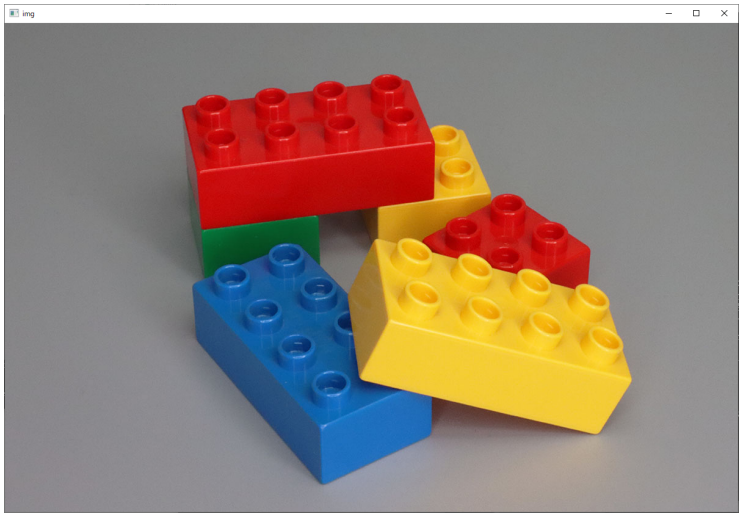
- OpenCVライブラリとは

  画像処理用の機能をまとめたライブラリ

```
> pip install opencv-python
```

OpenCVを用いた画像ファイルの表示

```
import cv2

img = cv2.imread('data/block.jpg')
cv2.imshow('img', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

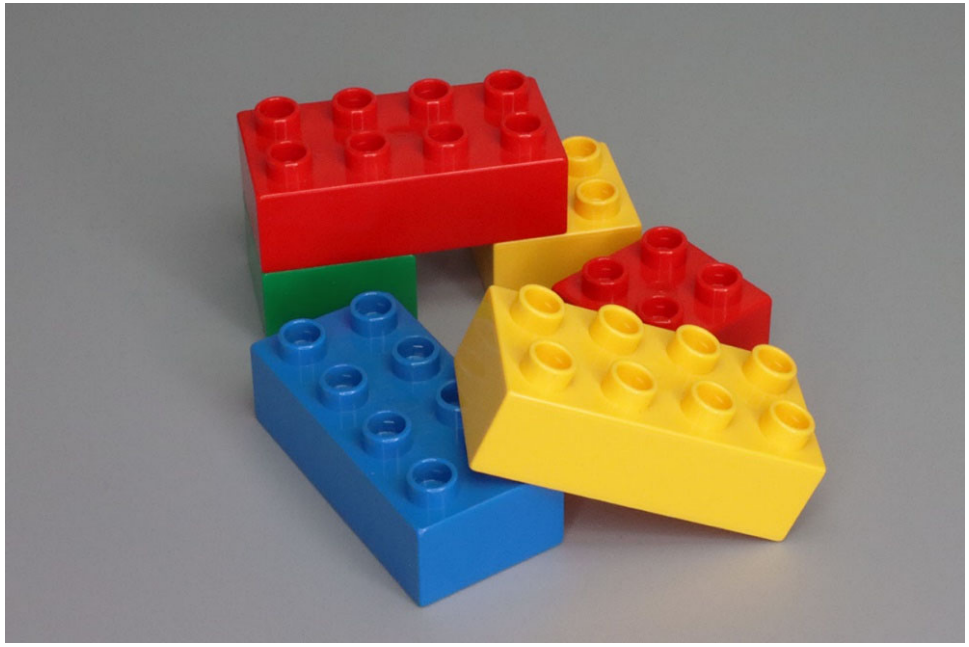


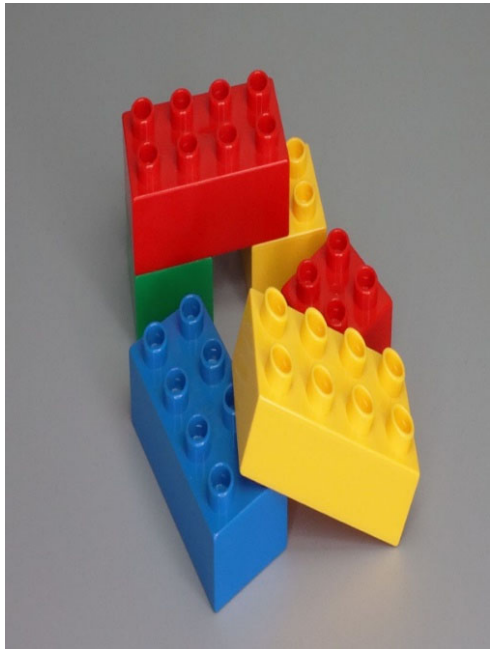
7-12/show_image.py

画像処理とファイルの書き出し

画像サイズの変更

```
height = img.shape[0]
width = img.shape[1]
resized_img = cv2.resize(img, (int(width/2), height))
cv2.imwrite('resized.jpg', resized_img)
```





画像処理とファイルの書き出し

色の変更（グレー画像）

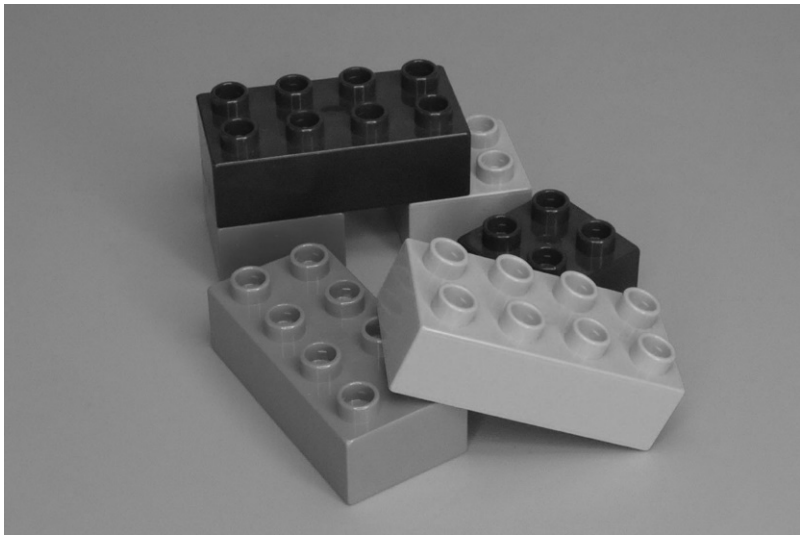
```
gray_img = cv2.cvtColor(img, cv2.COLOR_RGBA2GRAY)
cv2.imwrite('gray.jpg', gray_img)
```

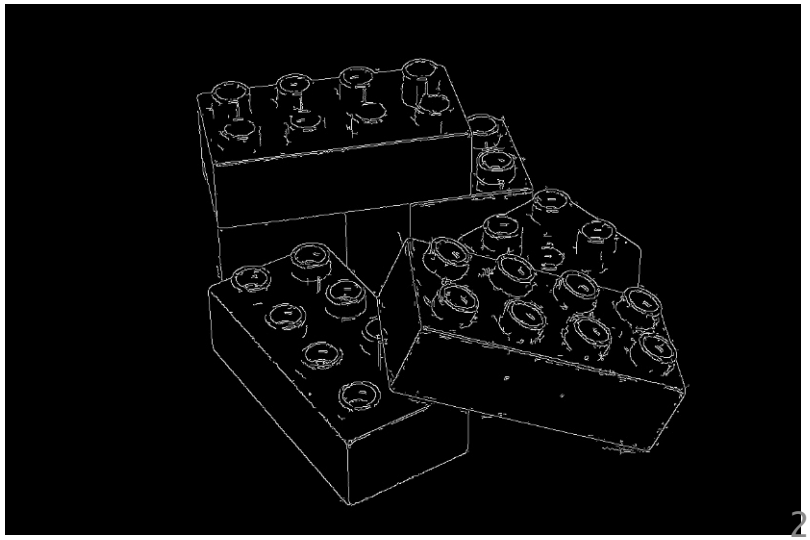
エッジ検出

エッジの長さを調整

エッジとみなす輝度の閾値

```
canny_img = cv2.Canny(img, 50, 100)
cv2.imwrite('canny.jpg', canny_img)
```





円の検出

写真から道路標識を検出

```
 1: import cv2
 2: import sys
 3:
 4: img = cv2.imread('road_sign.jpg')
 5: gray_img = cv2.cvtColor(img, cv2.COLOR_
BGR2GRAY)
 6: canny_img = cv2.Canny(gray_img, 600, 650)
 7:
 8:circles = cv2.HoughCircles(canny_img, cv2.
HOUGH_GRADIENT, dp=1, minDist=10, param1=700,
 param2=27, minRadius=20, maxRadius=60)
 9:
10: if circles is None:
11:     sys.exit()
12:
13: for (x, y, r) in circles[0]:
14:     cv2.circle(img, (int(x),
int(y)), int(r), (0,255,0), 6)
15:
16: cv2.imshow('detected circles', img)
17: cv2.waitKey(0)
18: cv2.destroyAllWindows()
```

処理を中断するexit関数を使用するためにインポートします

画像を読み込みエッジ検出までを済ませておきます

①円の検出を行います

エッジ検出用のパラメータ

円抽出の閾値

②何も検出されなければ処理を終えます

③円の情報を1つずつ取り出します

④検出された円を画像に上描きします

円を検出した結果の画像を表示します

実行結果

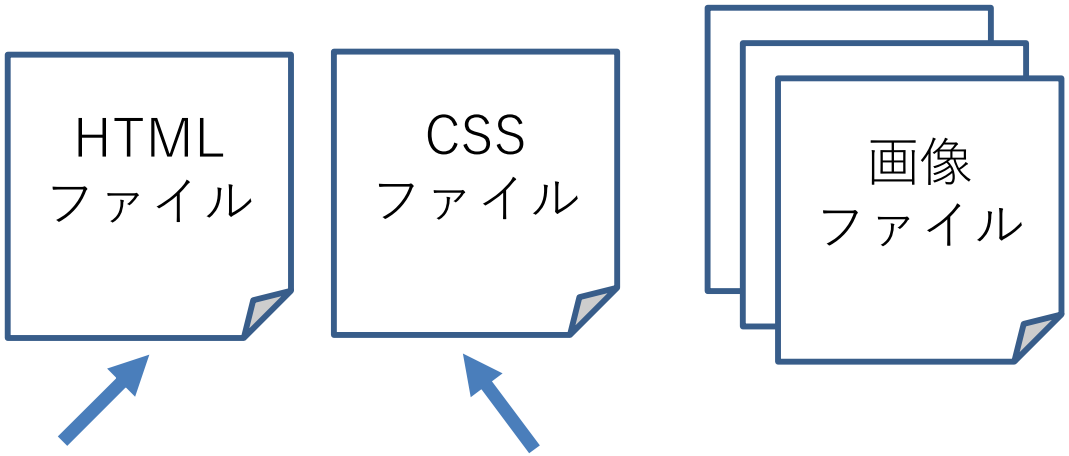
Webスクレイピング

Webスクレイピング

- Webスクレイピングとは

  インターネット上のWebページから、欲しい情報を取り出すこと。

Webページの構成



ページの構造と文書を記述したファイル

ページのデザインを記述したファイル

HTMLファイルの構造

- 文書の構造を**タグ**で定義

<タグ名>文章</タグ名>

```
<html>
<head>
<title>タイトル</title>
</head>
<body>
<h1>見出し</h1>
<p>段落1</p>
<p>段落2</p>
</body>
</html>
```

Webスクレイピングの流れ

1. インターネット上からHTMLファイルを取得する
2. HTMLファイルに含まれるタグの構造を見て、必要な情報が記述されている場所を特定する
3. HTMLファイルから必要な情報を取り出す

ライブラリの準備

- requestsライブラリ
  インターネット上のファイルを取得する

```
> pip install requests
```

- beautifulsoup4ライブラリ
  HTMLファイルの内容を解析する

```
> pip install beautifulsoup4
```

HTMLファイルの取得

- requestsモジュールのgetメソッドで HTMLファイルを取得できる

```
import requests

html = requests.get('https://www.shoeisha.co.jp/book
/ranking')

print(html.txt)
```

```
<!DOCTYPE HTML>
<html>
<head>
<script>
var dataLayer = dataLayer || [];
  dataLayer.push({
    'trackPageview':'SECOJP/book/ranking',
    'member' : 'nonmember',
  });
</script>
(略)
```

ブラウザで「ページのソースを表示」したときと同じ内容が得られる。

HTMLファイルの解析

```
import requests
from bs4 import BeautifulSoup

result = requests.get('取得したいURL')
soup = BeautifulSoup(result.text, 'html.parser')
print(soup.title)
```

インスタンス変数の title を参照すると、
titleタグに囲まれた文字列が得られる

HTMLファイルの解析（ファイルを読み込む場合）

- ファイルに保存済みのHTMLファイルを対象にする場合

```
from bs4 import BeautifulSoup

with open(ファイルパス, 'r', encoding='UTF-8') as f:
    soup = BeautifulSoup(f.read(), 'html.parser')

print(soup.title)
```

```
<section id="cate1">

  <h2>書籍ランキング</h2>

  <div class="newbooks">
      (略)
  </div>

  <div class="column">
    <ul class="list-unstyled">
      <li> <a href="xxx"><span class="date">1位</span>
タイトル1 </a></li>
      <li> <a href="xxx"><span class="date">2位</span>
タイトル2 </a></li>
          (略)
      <li> <a href="xxx"><span class="date">10位</span>
タイトル10 </a></li>
    </ul>
  </div>

</section>
```

```
 1: import requests
 2: from bs4 import BeautifulSoup
 3:
 4: result = requests.get('https://
www.shoeisha.co.jp/book/ranking')
 5: soup = BeautifulSoup(result.text, 'html.parser')
 6:
 7: for sec in soup.select('section'):
 8:     if(sec.select_one('h2')):
 9:         category = sec.select_one('h2').text
10:         title = sec.select_one('ul').select_one('li').text
11:
12:         print('カテゴリ：', category)
13:         print('書籍名：', title[3:])
```

①必要なライブラリをimportします

②BeautifulSoupのオブジェクトを生成します

③sectionタグの中身を1つずつ取り出します

④h2タグが存在するか調べます

⑤最初のh2タグの中身をカテゴリ名として取り出します

⑥最初のulタグの中の最初のliタグの中身を書籍のタイトルとして取り出します

⑦文字列titleの4文字目以降を出力します

BeautifulSoup オブジェクトのメソッド

- select メソッド： 指定したタグの要素をリストの形式で取得
- select_one メソッド： 指定したタグの先頭の要素を取得

実行結果

```
カテゴリ： 書籍ランキング
書籍名： ゲームメカニクス大全　ボードゲームに学ぶ「おもしろさ」の仕掛け
カテゴリ： 電子書籍ランキング
書籍名： 独習Python
カテゴリ： コンピュータ入門書ランキング
書籍名： Excelパワーピボット　7つのステップでデータ集計・分析を「自動化」する本
カテゴリ： コンピュータ専門書ランキング
書籍名： Python 1年生　体験してわかる！会話でまなべる！プログラミングのしくみ
カテゴリ： 資格書ランキング
書籍名： 深層学習教科書 ディープラーニング G検定（ジェネラリスト）公式テキスト
カテゴリ： ビジネス書ランキング
書籍名： THE MODEL（MarkeZine BOOKS）　マーケティング・インサイドセールス・➡
営業・カスタマーサクセスの共業プロセス
カテゴリ： 福祉書ランキング
書籍名： 福祉教科書 介護福祉士 完全合格過去＆模擬問題集 2021年版
カテゴリ： デザイン書ランキング
書籍名： やってはいけないデザイン
カテゴリ： 実用書ランキング
書籍名： 暮らしの図鑑 民藝と手仕事　長く使いたい暮らしの道具と郷土玩具61×基礎知識➡
×楽しむ旅
```

練習問題

問題 1

次のコードは、リストに含まれる文字列を1つだけ出力します。どの文字列を出力するかは、ユーザーが入力する値をインデックスに使用して決めます。入力した値が0、1、2のいずれかの場合には正しく動作しますが、そうでない場合にはエラーが発生してしまいます。

```
l = ['リンゴ', 'バナナ', 'オレンジ']
a = input('好きな整数を入力してください:')
print(l[int(a)])
```

try～except 構文を使用して、次のような例外処理を追加してください。

・ 入力した値が数字でない場合（**ValueError**が発生します）に「数字が入力されませんでした」と出力する

・ 数字であってもインデックスの範囲を超えている場合（**IndexError**が発生します）には、「範囲外の値が入力されました」と出力する

問題１（解答）

```
l = ['リンゴ', 'バナナ', 'オレンジ']
a = input('好きな整数を入力してください:')
print(l[int(a)])
```

・入力した値が数字でない場合に「数字が入力されませんでした」と出力する

・ 数字であってもインデックスの範囲を超えている場合には、「範囲外の値が入力されました」と出力する

```
l = ['リンゴ', 'バナナ', 'オレンジ']
a = input('好きな整数を入力してください:')
try:
    print(l[int(a)])
except ValueError:
    print('数字が入力されませんでした')
except IndexError:
    print('範囲外の値が入力されました')
```

問題 2

次の文章の空欄に入れるべき語句を答えてください。

- 人が読んで理解することができる文字の集まりで記述されたファイルをテキストファイルといい、テキストファイルでないものを [　　] という

- テキストファイルのうち、データをカンマ区切りで記述する形式を [　　]とよぶ。

- open関数の引数modeには、ファイルを読み込み用に開くときに [　] を、書き込み用に開くときに [　] を、追記用に開くときに [　] を指定する。

問題２（解答）

次の文章の空欄に入れるべき語句を答えてください。

- 人が読んで理解することができる文字の集まりで記述されたファイルをテキストファイルといい、テキストファイルでないものを [**バイナリファイル**] という

- テキストファイルのうち、データをカンマ区切りで記述する形式を [**CSV形式**]とよぶ。

- open関数の引数modeには、ファイルを読み込み用に開くときに [**r**] を、書き込み用に開くときに [**w**] を、追記用に開くときに [**a**] を指定する。

問題 3

次の文章の空欄に入れるべき語句を、選択肢から選んでください。

・Pythonに最初から備わっていない外部ライブラリは、[　　]コマンドでインストールする必要がある。

・[　　]ライブラリには、グラフ描画用の各種機能が含まれ、[　　]ライブラリは、画像処理用の各種機能が含まれる。

・インターネット上のWebページから、欲しい情報を取り出すことを [　　] という。

・一般的にWebページは、ページの構造と文書を [　　] を用いて記述した [　　]ファイルと、ページのデザインを記述した [　　]ファイル、および画像ファイルなどから構成される。

【選択肢】

・CSS　・HTML　・OpenCV　・pip　・matplotlib
・Webスクレイピング　・タグ

問題３（解答）

次の文章の空欄に入れるべき語句を、選択肢から選んでください。

・Pythonに最初から備わっていない外部ライブラリは、[**pip**]コマンドでインストールする必要がある。

・[**matplotlib**]ライブラリには、グラフ描画用の各種機能が含まれ、[**OpenCV**]ライブラリは、画像処理用の各種機能が含まれる。

・インターネット上のWebページから、欲しい情報を取り出すことを [**Webスクレイピング**] という。

・一般的にWebページは、ページの構造と文書を [**タグ**] を用いて記述した [**HTML**] ファイルと、ページのデザインを記述した [**CSS**] ファイル、および画像ファイルなどから構成される。

【選択肢】

・CSS　・HTML　・OpenCV　・pip　・matplotlib
・Webスクレイピング　・タグ

終