プログラミング言語

Python 入門

2020-10-27 – 2020-10-29

農研機構・東北農業情報研究センター（盛岡）

農研機構・農業情報研究センター
孫 建強

10/27	13:00-17:00	基本文法
10/28	09:00-12:00	テキスト処理
	13:00-17:00	NumPy
		Pandas
10/29	09:00-12:00	可視化

Python 文法（基礎）

- プログラミング言語
- 基本オブジェクト
- 基本文法

プログラミング言語

- プログラミング言語
- データ構造とアルゴリズム

プログラミング言語

特定のタスクを達成させるための一連の操作を、コンピューターに理解可能な形で記述するための言語。

プログラミング言語とゲーム

プログラミング言語をゲームに例えると理解しやすい場合がある。例えば、ゲームではコントローラーを通じて入力し、上に進む、左に進む、右に進む、下に進むなど基本操作（命令）を入力し、キャラクターを制御し、ミッションをクリアする。これらの一連の入力（上下左右の順序や回数）がプログラミング言語である。

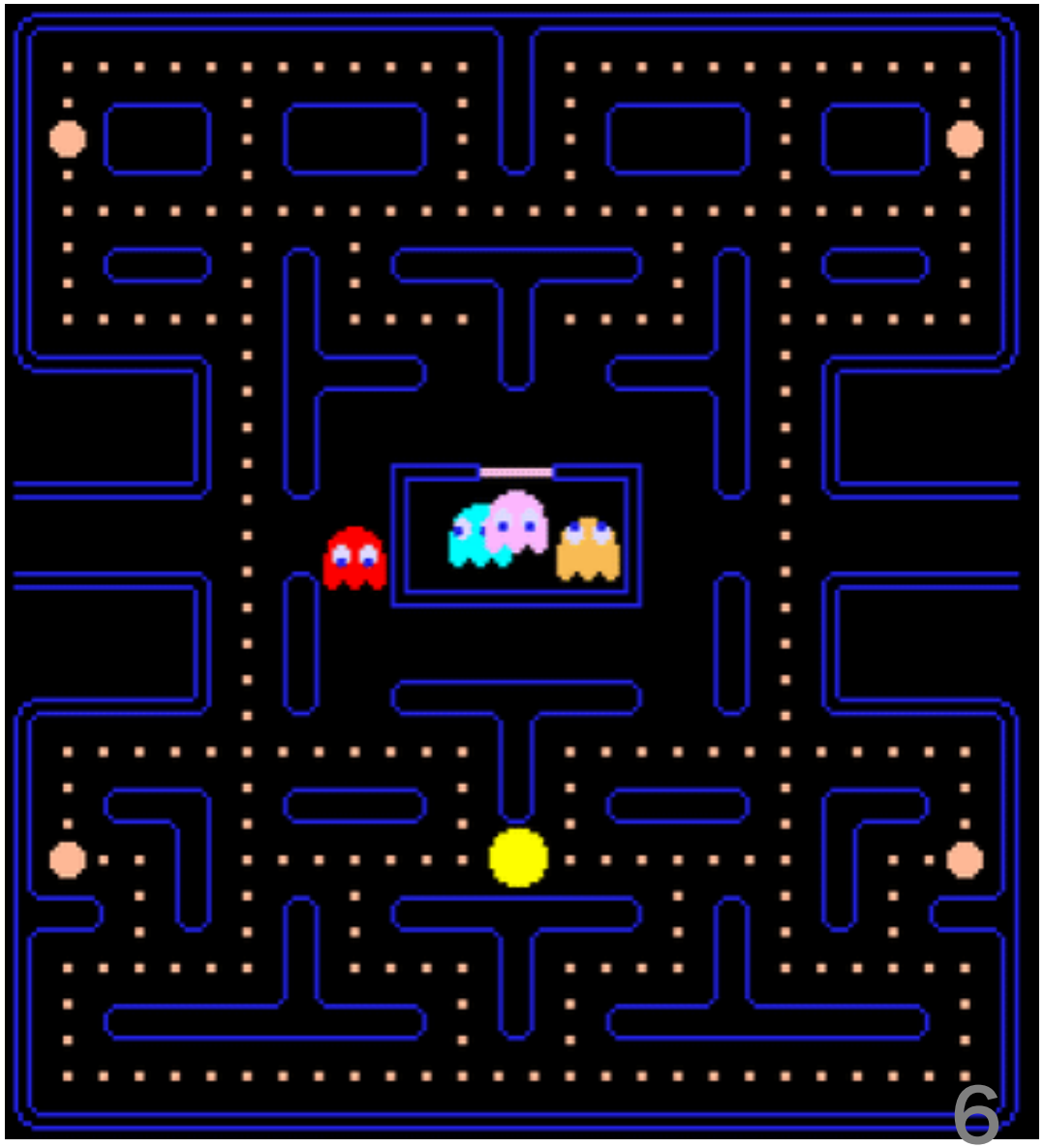
言語仕様

U	上へ進む
D	下へ進む
L	左へ進む
R	右へ進む
0-9	進むマス数

→

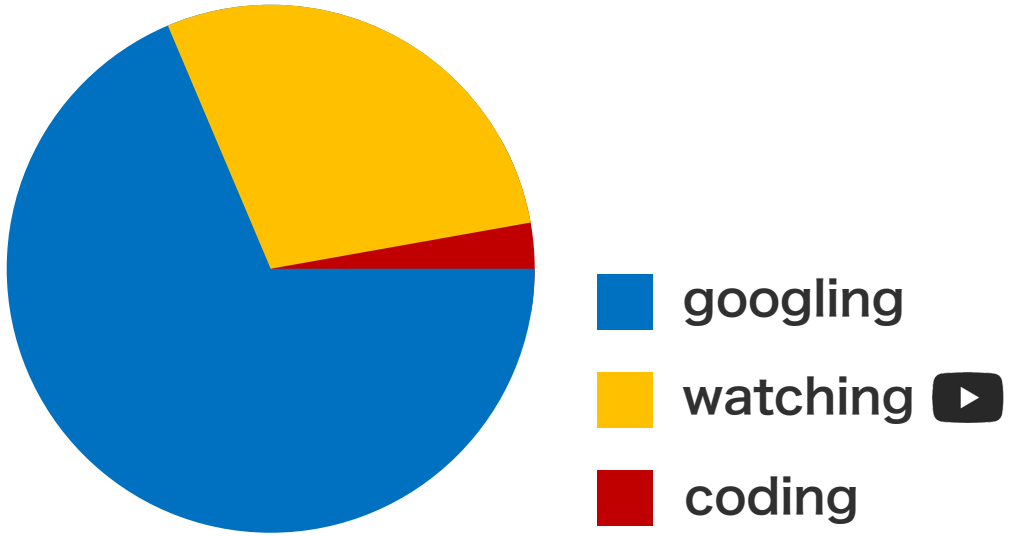
```
L4
D3
R3
D3
R3
U3
R3
U3
:
```

→

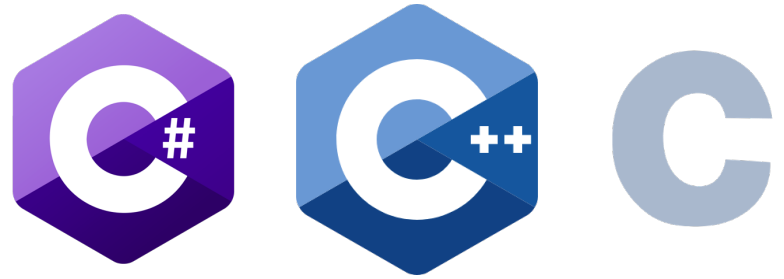


プログラミング言語

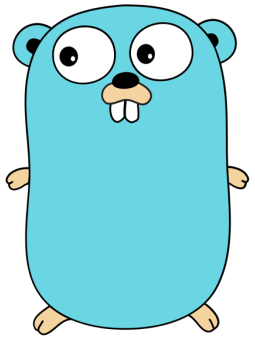
プログラミング言語を学ぶ上で、重要なのは理論的な思考力を身につけることである。ある作業をコンピューターに処理させたいとき、その作業を基本的な操作に分解し、それらを正しい順序で組み合わせ、コンピューターが理解できる形に置き換える必要がある。このような作業の分解や組み合わせなどが大切である。一方で、プログラミングの文法や常用関数の名前や使い方は、あまり重要ではなく、都度に調べればよい。

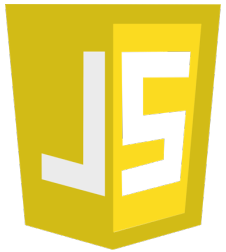


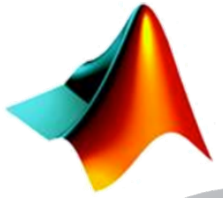
プログラミング言語



julia







MATLAB

現在、数百種類以上のプログラミング言語が使われている。その機能に着目すると、様々な目的に使用できる汎用型言語と統計やウェブページ作成などの特定の目的で使用できる専用型言語に分けることができる。プログラミング言語同士に優劣はなく、目的に応じて使い分けされている。

Mother Tongues

Tracing the roots of computer languages through the ages

Just like half of the world's spoken tongues, most of the 2,300-plus computer programming languages are either endangered or extinct. As powerhouses C/C++, Visual Basic, Cobol, Java and other modern source codes dominate our systems, hundreds of older languages are running out of life.

An ad hoc collection of engineers-electronic lexicographers, if you will-aim to save, or at least document the lingo of classic software. They're combing the globe's 9 million developers in search of coders still fluent in these nearly forgotten lingua frangas. Among the most endangered are Ada, APL, B (the predecessor of C), Lsp, Oberon, Smalltalk, and Simula.

Code-raker Grady Booch, Rational Software's chief scientist, is working with the Computer History Musuem in Silicon Valley to record and, in some cases, maintain languages by writing new compilers so our ever-changing hardware can grok the code. Why bother? "They tell us about the state of software practice, the minds of their inventors, and the technical, social, and economic forces that shaped history at the time," Booch explains. "They'll provide the raw material for software archaeologists, historians, and developers to learn what worked, what was brilliant, and what was an utter failure." Here's a peek at the strongest branches of programming's family tree. For a nearly exhaustive rundown, check out the Language List at *HTTP://www.informatik.uni-freiburg.de/Java/misc/lang_list.html.* - Michael Mendeno

Key

- 1954 Year Introduced
- **Active:** thousands of users
- **Protected:** taught at universities; compilers available
- **Endangered:** usage dropping off
- **Extinct:** no known active users or up-to-date compilers
- **Lineage continues**

1954 1955 1956 1957 1958 1959 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974 1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001

Fortran — Once one of the most widely used languages in science and engineering.
Fortran IV — Created for the IBM 7090/94
Fortran 90 ISO/IEC
Flow-Matic — Originally B-O. One of the first English-like data-processing languages.
Cobol — The culprit behind many Y2K bugs. Also known as Common Business-Oriented Language.
Cobol 61
ANSI Cobol 74
ANSI Cobol 85
OO Cobol
IPL — Possibly the first list-processing language used in AI guru Ed Feigenbaum's PhD thesis.
Lisp — Invented by John McCarthy at MIT. Lisp has an unusual syntax made up of lots of nested parantheses. Still popular with AI researchers.
Scheme — A Lisp-like language designed by MIT researcher Seymour Papert to make programming accessible to children through graphics and simple syntax.
A Lisp offshoot that's still used in introductory programming classes at universities.
Scheme 84
Scheme IEEE
Scheme R5RS
Haskell 1.0
Haskell 1.1
Logo
Object Logo
Clos — Common Lisp Object System.
Common Lisp
ANSI Common Lisp
George Radin's attempt to combine the best features of Fortran, Cobol, and Algol 60.
PL/1 — Algorithmic Language, designed as a portable language for scientific computations. Algol 68's complexity was either ahead of its time or rightfully doomed, depending on whom you ask.
ANSI PL/1
PL/M — Programming Language/Microcomputers
Self
Self 4.0
A simple object-oriented language.
NetRexx
Initially called Oak. Still one of the fastest-growing languages around, despite the standards fued between Sun and Microsoft. Somewhat like C++, Java allows for "write once, run anywhere" portability across the Net.
Smalltalk — The first language to show off the power of object-oriented code. Developed by Software Concepts Group, Xerox PARC, led by Alan Kay.
The version Xerox released to the public.
Smalltalk-80
Algol 58 — The hoped-for Esperanto of the computing world. Designed by an international Zurich-based committee as a universal language, it was one of the first attempts at making software more portable. Initially called International Algebraic Language.
Algol 68
Rex 1.00
Rex 2.00 — Restructured Extended Executor
An object-oriented language.
Object Rexx
Simula — Popular in Europe during the '70s, Simula introduced the now-standard concept of object-oriented, rather than procedural, programming. Desgined by Kristen Nygaard and Ole-Johan Dahl.
Simula 67 — All-purpose version of Simula
A simple language (A = argument, B = basic value, C = ?).
Interactive object-oriented language.
Java
Java 2(v1.3)
Prolog — Programming en Logique for natural-language processing. Popular for AI programs.
Clu — Called Cluster, it's an object-oriented academic language created to teach rigid engineering skills.
Marco and text-processing language used by Unix admins to manipulate large config files programmatically, rather than editing them by hand.
Objective-C
Eiffel
Eiffel 3
Microsoft's answer to Java, it is a key component of the Microsoft.Net platform for Web services.
ABC
Sather 0.1
Ruby
An object oriented language.
Python
Python 1.6
AWK
NAWK — New AWK, a pattern scanning and processing language.
A popular language among Web site builders, it includes features missing from Perl.
ECMA Script
Possibly the most common language today. Adds object-oriented features to C.
Tool Command Language, "tickle." The duct tape of programming (a scripting language for patching together different languages).
Cmm
JavaScript
Microsoft's version of JavaScript. Can't they share anything? :-)
C--, a scripting language.
JScript
C#
One of the most widely deployed languages today. Windows and Unix OSes are written mostly in C and its descendants.
C with Classes
C++
Internet C++
Concurrent C
Tcl
Tcl/Tk
Tcl/Tk 8.1
BCPL — Basic Combined Programming Language
B — Used primarily for non-numeric programming.
C
ANSI C (C89)
ISO C (C95)
Updates C++ for the Net with a Java-like virtual machine, so code can run on any kind of computer.
ADA
ADA 83
ADA 95
The US Department of Defense's effort to craft a standard object-oriented language for its work. Named after Ada Lovelace, arguably the world's first computer programmer, and created by Jean Ichbiah's team of Honeywell.
Modula 3
Found in millions of Web pages. Originally dubbed LiveScript, it was renamed by Netscape marketers who licensed the name to ride Java's buzz. It has little in common with that language.
A teaching language named for French mathematician Blaise Pascal. Designed by legendary Swiss programmer Niklaus Wirth for simplicity, in reaction to the complexity of Algol 68.
Modula — Modular Language
Modula 2 ISO
Object-oriented Pascal, designed for simplicity.
Object Pascal
Oberon
Another creation of Niklaus Wirth. Later modified by Robert Griesemer for numerical apps on supercomputers.
Delphi
Pascal
String-processing language for text and formula manipulation, common in text processers. Invented by David Farber, Ralph Griswold, and Ivan Polonsky at Bell Labs.
Csh — The swiss Army Knife of programming (a k a Practical Extraction and Report Language), used for patching together different languages. Spawned a quasi-literary culture that writes Perl haiku
Perl 1.000
Perl 4.000
Programmer's easy street - with this descendant of Snobol, there's no need to know the underlying OS.
sh
Ksh
Frame-based language.
King of the one-liners, also referred to as A Programming Language. An algebra-like language with a special set of characters for writing complex programs as short formulas. Designed by Kenneth Iverson at Harvard.
A general-purpose language known for its beauty and grace. Designed by Ralph Griswold to be successor to Snobol4.
C-Shell, a scripting language and command-shell interpreter. Written by C programmers to make Unix command lines more like C's.
Standard ML, a general-purpose language.
The kitchen sink of command-line programming features for Unix, based on the original Bourne shell (a k a Bourne Again Shell).
Bash
Snobol
Snobol4
Icon
ML — Meta Language
SML
SML 90
SML 97
Word-based language first used to guide the National Radio Astronomy Observatory telescope at Kitt Peak in Arizone.
Page-description language for printers and graphics sistems; 75 percent of all commercial documents are produced on PostScript printers.
Caml
Caml 2-6.1
Objective Caml
O Caml 2
O Caml 3.00
Forth
PostScript
PostScript Level 2
J
Although mocked by "real" programmers for its limited abilities, Basic has outlived many more advanced languages, as well as the RadioShack TRS-80 computers that made it a household word. Stands for Bigenner's All-Purpose Symbolic Instruction Code.
APL
Categorical Abstract Machine Language
APL 96
Microsoft Basic
Basic
MS Basic 2.0
Visual Basic
The Rodney Dangerfield of programming languages. Popular for building Web sites with Microsoft Visual Studio tools.

Survival of the Fittest

Reasons a language endures, with examples of some classic tongues

- **Appeals to a wide audience** C (bolstered by the popularity of Unix)
- **Gets a job done** Cobol (designed for business-report writing)
- **Delivers new functionality** Java (runs on any hardware platform)
- **Fills a niche** Mathematica (speeds up complex computations)
- **Offers a modicum of elegance** Icon (has friendly, line-oriented syntax)
- **Has a powerful user base or backer** C# (developed by Microsoft for .Net)
- **Has a charismatic leader** Perl (programmer-author Larry Wall)

Sources: Paul Boutin; Brent Hailpern, associate director of computer science at IBM Research; The Retrocomputing Museum; Todd Proebsting, senior researcher at Microsoft; Gio Wiederhold, computer scientist, Stanford University

アプリ開発 ⟶ C/C++, Java, Python, ...

ウェブ開発 ⟶ Python, Ruby, JavaScript, ...

科学計算 ⟶ Python, R, FORTRAN, ...

1990 2000 2010

Python 0.9

Python 1.0

Guido van Rossum が C 言語のように機能が多くて、shell のように簡単に描けるプログラミング言語を開発した。プログラミング言語の名前はBBC コメディ番組『空飛ぶモンティ・パイソン』にちなんで Python と名付けられた。Python は性能重視に作られた言語ではないため、性能を高めたい場合は C 言語でライブラリー .so を作成して Python に読み込むことで対応する。

Python 2.0

Python の基本的な特徴が決定される。

- オブジェクト指向型
- 動的型付け言語
- 他言語との互換性
- パッケージ

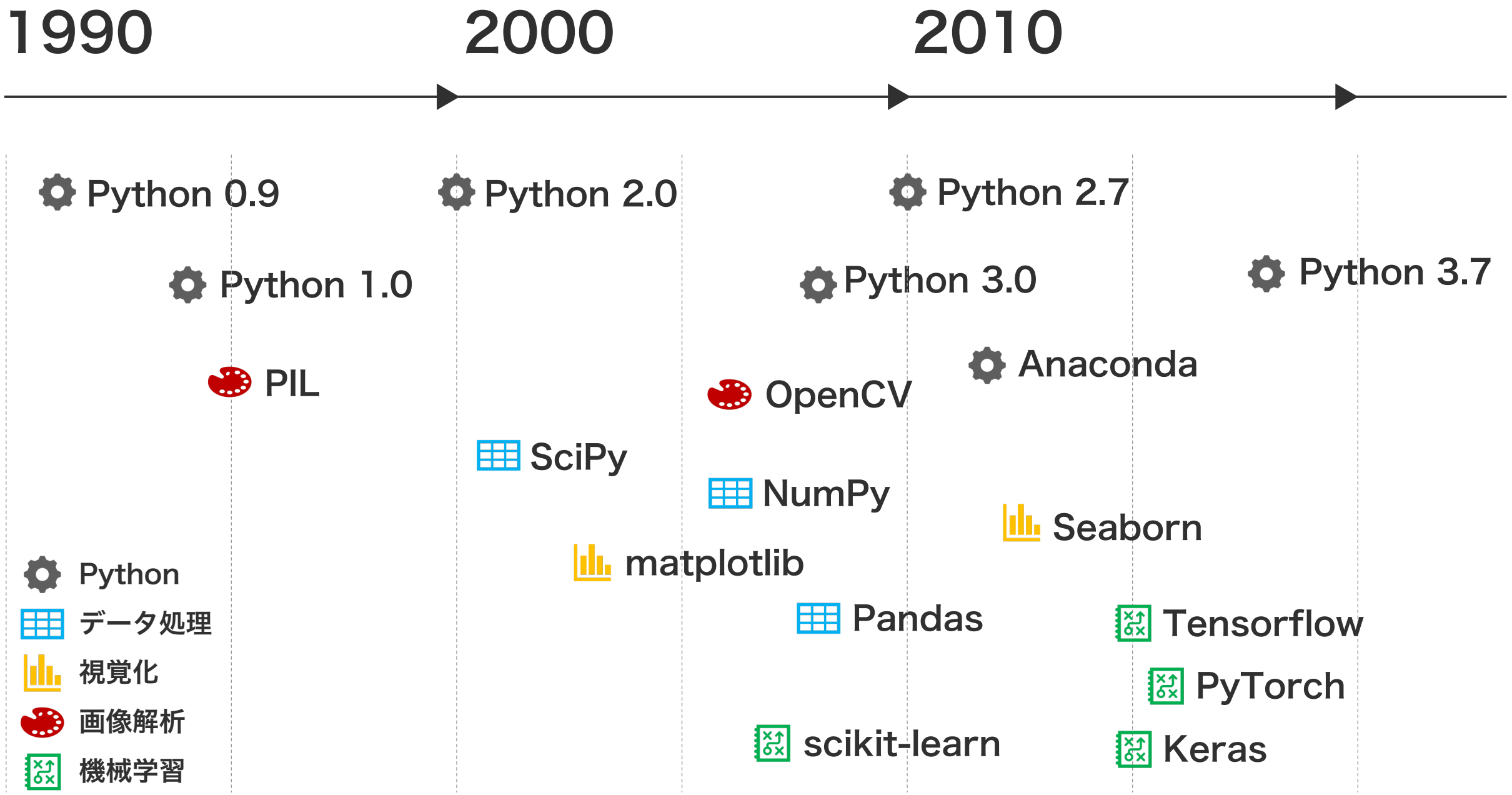
また、2004 年に ウェブ開発フレームワーク Django のリリースにより、Dropbox 、 YouTube 、CIA などのウェブに Python が使われるようになる。

Python 2.7

Python 3.0

Python 2.x が盛んに使われる中、Python 3.0 がリリースされる。2008 年以降に、2.x と 3.0 系が並行して使われる時代に入る。機械学習、人工知能などパッケージが開発され、Python ユーザー数がさらに増える。現在では 3.x が主流となり、2.x は 2020 年を持ってサポート終了。これから Python を始めるとき 3.7 以降を使うことをお薦めする。

Python 3.7


1990
2000
2010
Python 0.9
Python 1.0
PIL
Python 2.0
SciPy
NumPy
matplotlib
OpenCV
Python 2.7
Python 3.0
Pandas
scikit-learn
Anaconda
Seaborn
Tensorflow
PyTorch
Keras
Python 3.7
Python
データ処理
視覚化
画像解析
機械学習

YouTube

Instagram

django

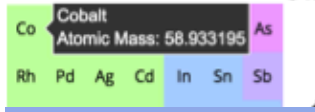
Dropbox

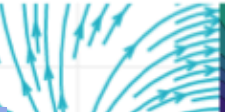
NumPy

pandas

matplotlib

plotly

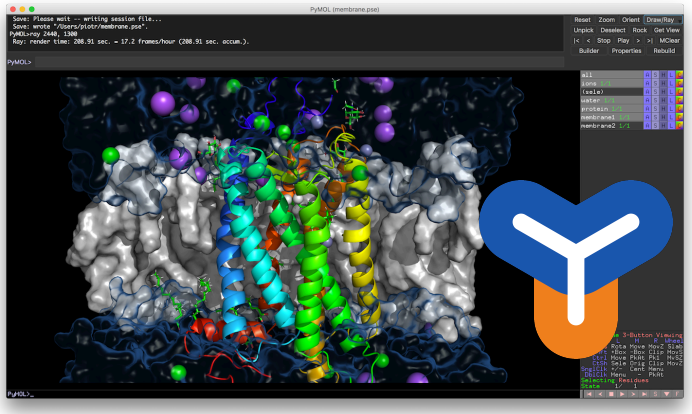


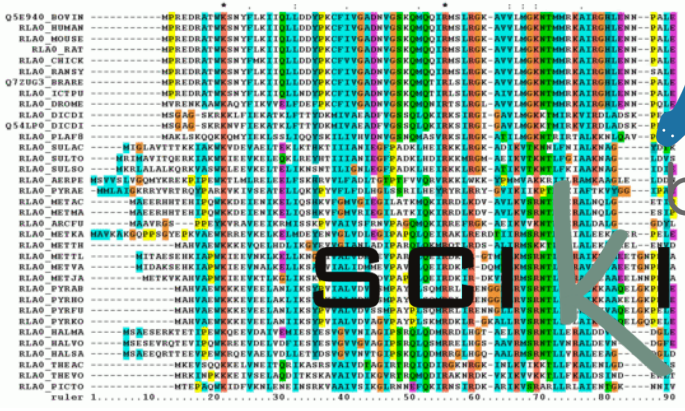


SciPy.org

Apache OpenOffice™

scikit learn





biopython

BIO

PyTorch

TensorFlow

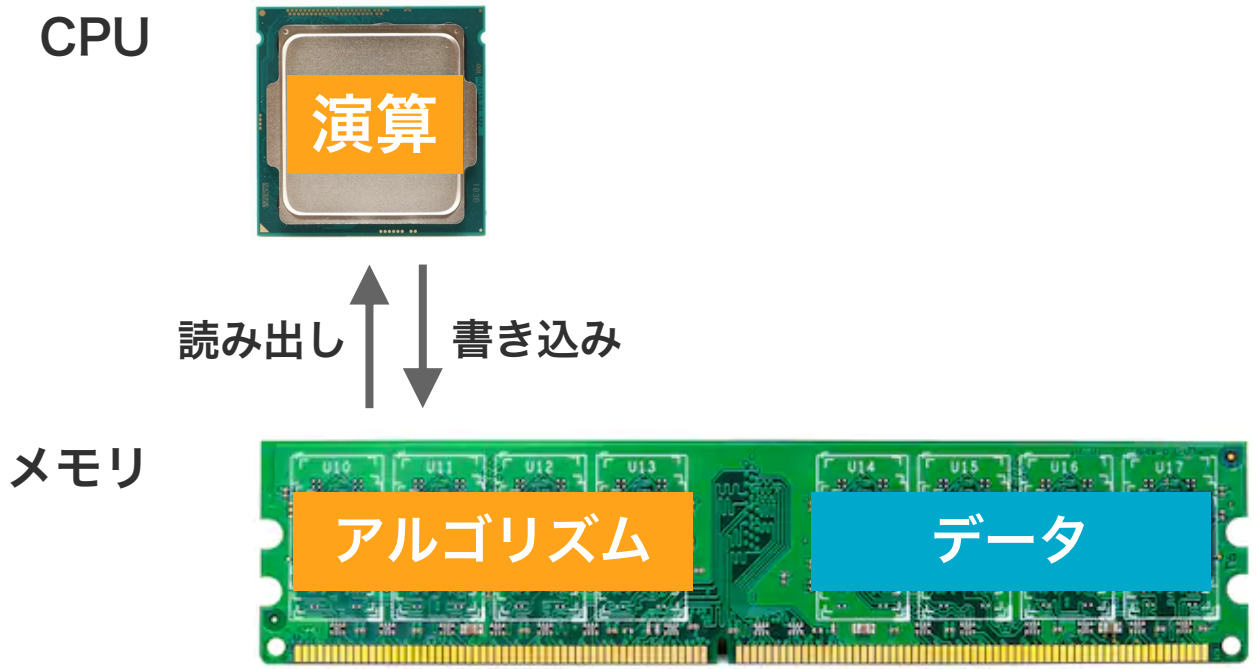
python™

プログラミング言語

- プログラミング言語
- データ構造とアルゴリズム

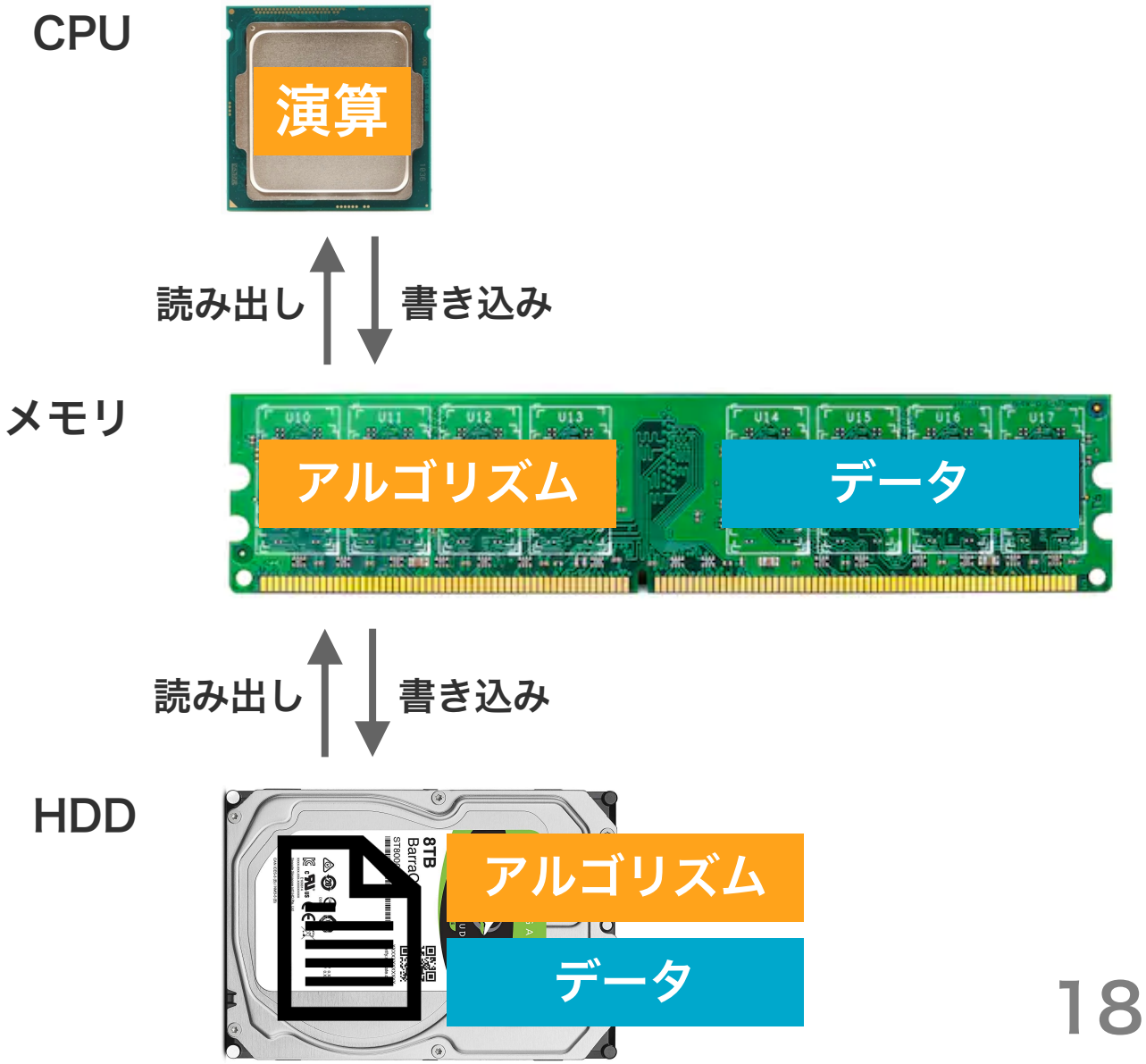
データ構造とアルゴリズム

コンピュータでデータを処理するためには、データや処理手順（アルゴリズム）をコンピュータのメモリ上に保持する必要がある。CPU はメモリ上に保持されたアルゴリズムに従い、メモリ上から指定された位置にアクセスしデータを読み込み、計算を行い、その計算結果をメモリ上の指定された位置に保存する。



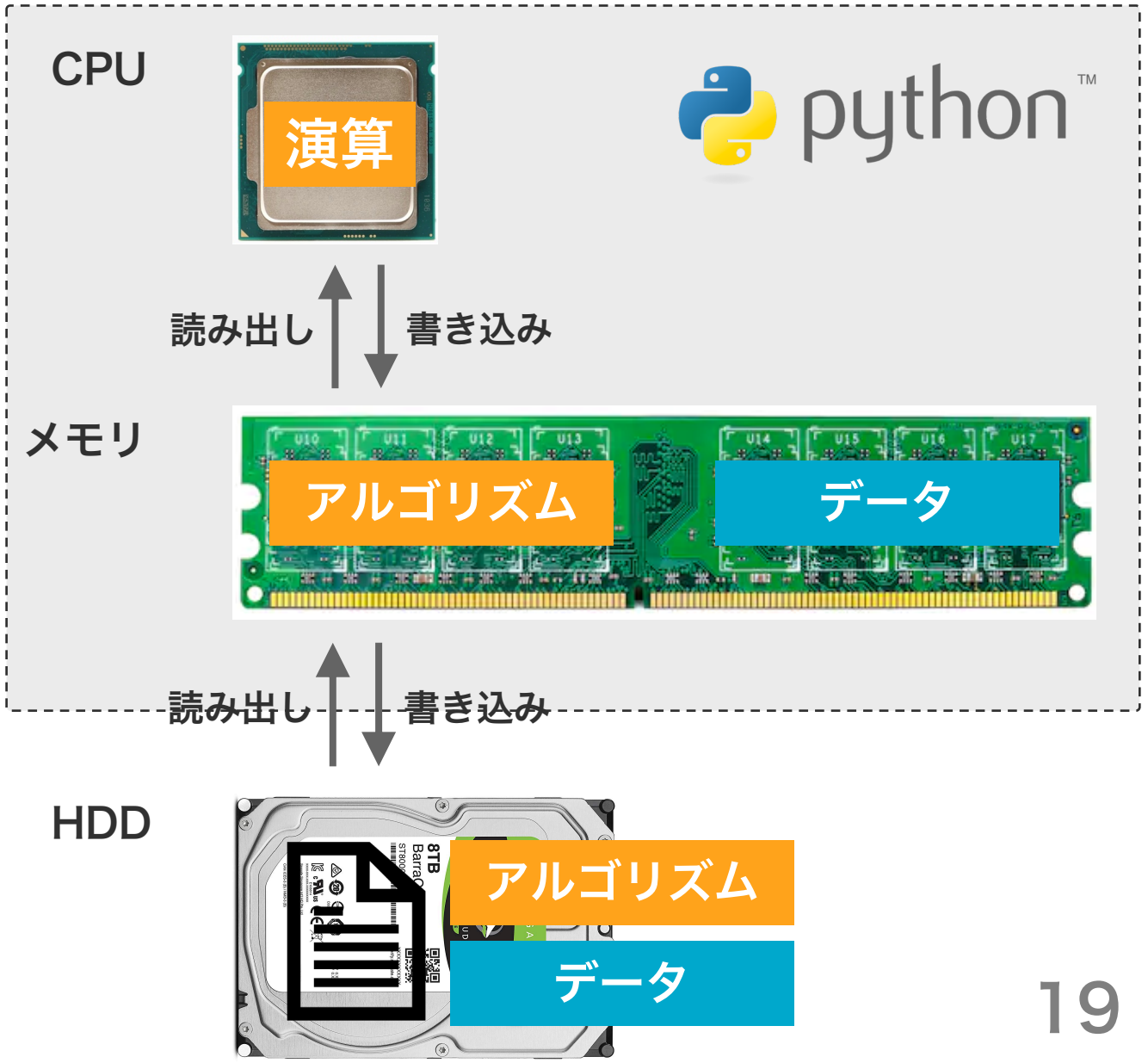
データ構造とアルゴリズム

コンピュータの電源を切ると、メモリ上に保持された内容は消える。そのため、解析する度に、アルゴリズムやデータを直接メモリに書き込むよりも、それらを不揮発性メモリであるハードディスクに保存し、必要に応じてハードディスクからメモリに自動的にコピーしてから解析した方が効率がいい。



データ構造とアルゴリズム

アルゴリズムおよびデータを Python 文法に従ってファイルに記述すれば、Python エンジンが、そのファイルの情報をメモリ上に展開し、CPU やメモリ間の制御を行いながら演算を行う。

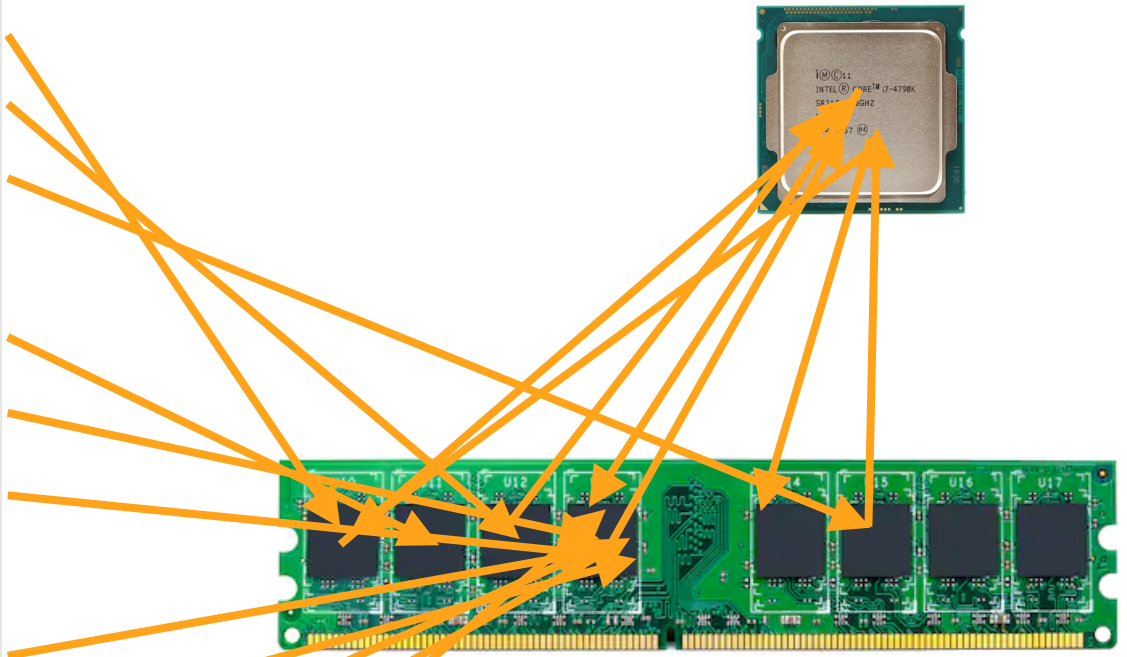


データ構造とアルゴリズム

データ

アルゴリズム

```
a = 120
takeout = True
has_coupon = True


s = 0
if has_coupon:
    s = a * 0.95

if takeout:
    s = a * 1.08
else:
    s = a * 1.10

print(s)
```



python™

データとアルゴリズムを 1 つのファイルに記述し、それを Python に実行させることで、Python がメモリや CPU 間の様々な制御を行う。

データ構造とアルゴリズム

データ →

- 整数
- 小数
- 文字
- ベクトル
- 行列

→

- スカラー
- リスト
- ディクショナリ
- セット

アルゴリズム →

- 探索
- 並べ替え
- 線型計画法
- 待ち行列理論
- 動的計画法

→

- 条件構文
- 繰り返し構文
- 関数

アルゴリズム – 探索

次の数列の中に 9 が存在するかどうかを調べたよ。ただし、コンピュータは、複数の作業を同時に実行できないことに注意せよ。

2	6	4	9	3	8	0	5	1	7

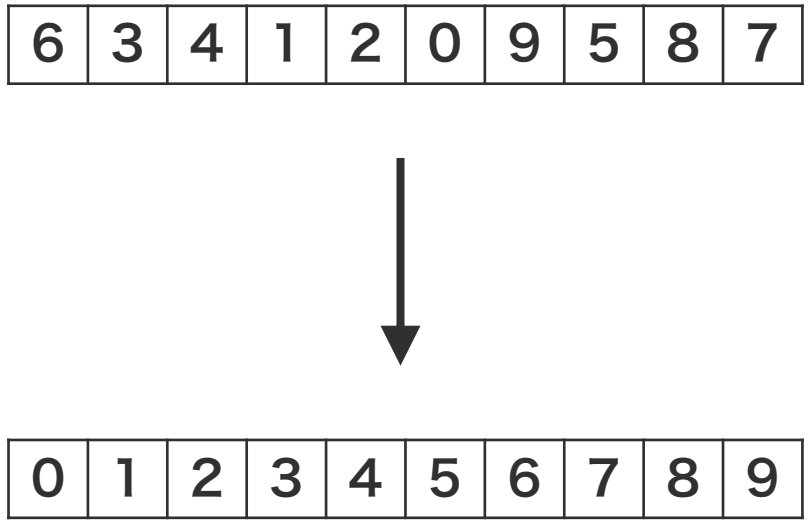
アルゴリズム – 探索

次の数列の中に 9 が存在するかどうかを調べたよ。ただし、コンピュータは、複数の作業を同時に実行できないことに注意せよ。

0	1	2	3	4	5	6	7	8	9

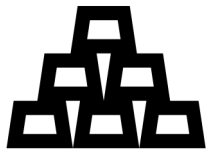
アルゴリズム – 並べ替え

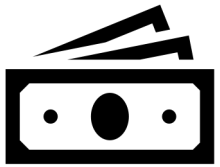
次の数列の各要素を昇順に並べ替えよ。ただし、コンピュータは、複数の作業を同時に実行できないことに注意せよ。



アルゴリズム – 線形計画法

リュックサックに入る範囲で、できるだけ高額な物を盗め。


ゴールド


札束

カード類

油絵

パソコン

タブレット

指輪

テレビ

デスク

Python 文法（基礎）

- プログラミング言語
- 基本オブジェクト
- 基本文法

基本オブジェクト

- スカラー
- リスト
- ディクショナリ
- セット
- タプル

オブジェクト

乗り物

車

電車

船

飛行機

⋮
⋮

オブジェクト

スカラー

リスト

ディクショナリ

セット

タプル

⋮

⋮

スカラー

オブジェクトには様々な種類がある。そのうち、ある値を 1 つだけ保持できるオブジェクトをスカラーとよぶ。スカラーに値を保持させるためには、そのスカラーに値を付与する必要がある。この付与作業を代入という。Python では、オブジェクトに値を代入するとき、代入演算子 "=" を使う。代入演算子 "=" は、右辺の値を、左辺のオブジェクトに代入する機能を持つ。代入演算子 "=" は、数学における等しいという意味を持たない。

右のコードは、1 という値を、a という名前のオブジェクトに代入することを表している。このコードを実行することによって、プログラムが終了するまで、a は 1 を保持している状態になる。

```
a = 1
```

※数値（整数、小数）以外に、複素数や文字、文字列を代入することもできる。

スカラー

代入演算子 "=" の右辺が計算式の場合は、その計算結果が左辺に代入される。また、すでに値を保持している変数に、他の値を代入すると、既存の値が上書きされる。

```
a = 1
b = 1

c = a + b
# 2

d = c - 2
# 0

e = a + 2
# 3

f = d + e
# 3

a = f - 1
# 2
```

標準出力

オブジェクトに代入された値は、コンピューターのメモリ上に保存される。出力命令を出さない限り、ディスプレイ（標準出力）に出力されない。出力命令をだすとき、print 関数を使用する。

※この資料では、紙面の関係上、これ以降 print 関数を省略する。

```
a = 1
b = 1

c = a + b
print(c)
# 2

d = c - 2
print(d)
# 0

e = a + 2
print(e)
# 3

f = d + e
print(f)
# 3
```

算術演算子

Python で四則演算を行うのに次のような演算子を使用する。割り算に関して、余と商を求める演算子もある。

演算子	意味	例
+	加	7 + 4 = 11
-	減	7 - 4 = 3
*	乗	7 * 4 = 28
/	除	7 / 4 = 1.75
%	余	7 % 4 = 3
//	商	7 // 4 = 1
**	累乗	2 ** 10 = 1024

※小数に対しても余と商を求めることができる。a//b は、a を b で割った後に整数部分を返す。また、a%b は、a - a//b*b で計算された結果が返される。

```
a = 11
b = 3

a + b


a - b

a * b

a / b

a % b

a // b
```

算術演算子

Python で四則演算を行うのに次のような演算子を使用する。割り算に関して、余と商を求める演算子もある。

演算子	意味	例
+	加	7 + 4 = 11
-	減	7 - 4 = 3
*	乗	7 * 4 = 28
/	除	7 / 4 = 1.75
%	余	7 % 4 = 3
//	商	7 // 4 = 1
**	累乗	2 ** 10 = 1024

※小数に対しても余と商を求めることができる。a//b は、a を b で割った後に整数部分を返す。また、a%b は、a - a//b*b で計算された結果が返される。

```
a = 11
b = 3

a + b
# 14

a - b
# 8

a * b
# 33

a / b
# 3.6666666666665

a % b
# 2

a // b
# 3
```

問題 O1-1

コシヒカリは複数の県で植えられている。各県におけるコシヒカリの 10a あたりの収量を次の表にまとめた。

コシヒカリの 10a あたりの平均収量を計算せよ。

都道府県名	収量 (kg/10a)
新潟	528.3
茨城	520.7
福島	538.8
栃木	535.2

```
niigata = 528.3

ibaraki = 520.7

fukushima = 538.8

tochigi = 535.2
```

本来はこれらのデータを CSV ファイルから直接読み込むが、初めは練習のために直打ちで！

問題 O1-2

各県における３品種の米の収穫量を次の表にまとめた。

各品種に対して、10a あたりの平均収量を計算せよ。

都道府県名	品種	収量 (kg/10a)
新潟	コシヒカリ	528.3
茨城	コシヒカリ	520.7
福島	コシヒカリ	538.8
栃木	コシヒカリ	535.2
宮城	ひとめぼれ	519.4
岩手	ひとめぼれ	520.5
福島	ひとめぼれ	561.6
秋田	あきたこまち	567.5
岩手	あきたこまち	538.3
岡山	あきたこまち	501.9

オブジェクト名（変数名）の命名規則

変数の名前に使える文字は、数字・アルファベット・アンダーバーである。ただし、変数名の最初の文字を数値以外の文字にする必要がある。また、予約語（if, for, def など）を変数名にすることができない。この規則を守れば、変数名を自由につけることができるが、実際には、この規則のほかに慣用的なルール（PEP8）に従って変数を命名することが一般的である。

- 変数名、関数名をその作用がわかるように小文字の英単語で命名する。単語が複数ある場合、アンダーバーで繋げる。
- クラス名を CapitalizedWords 形式で命名する。
- 定数をその意味がわかるように大文字の英単語で命名する。単語が複数ある場合、アンダーバーで繋げる。

https://www.python.org/dev/peps/pep-0008/

```
def calc_odd_sum(x):
    x = np.array(x)
    s = np.sum(x[x % 2 == 1])
    return s


fib_n = [1, 1, 2, 3, 5, 8, 13, 21, 34]

fib_odd_sum = calc_odd_sum(fib_n)



SIMULATION_TRY = 100

sim_results = []

for (i in range(SIMULATION_TRY)) {
    _result = simulate_calc_pi()
    sim_results.append(_result)
}
```

基本オブジェクト

- スカラー
- リスト
- ディクショナリ
- セット
- タプル

リスト

同じ属性をもつ複数の値をまとめて扱うときにリストを使う。たとえば、ある種のどんぐりの重さの分布を調べたいとき、どんぐりを 3 つ拾い、それぞれの重さを測ったのち、3 つの重さ全体に対して平均や分散を求めていくことになる。この際に、3 つのどんぐりの重さをそれぞれ別々のオブジェクトに保存するよりも、これらをまとめて 1 つのオブジェクトに保存した方がわかりやすい。このような場合において、リストが使われる。

```
a = [2, 3, 2]
```

1. 複数の値をカンマ区切りで並べる

2. 角括弧 (square bracket) で全体をまとめる

リスト

複数のスカラーの平均を求める場合、すべてのオブジェクトの合計値を計算し、その合計値をオブジェクトの個数で割る計算を行う。

```
weight_1 = 2
weight_2 = 3
weight_3 = 2

s = weight_1 + weihgt_2 + weight_3
n = 3

mean = s / n
```

リストの全要素に対して平均値を求めたいとき、全要素を束ねて一括で計算することができる。

```
weights = [2, 3, 2]

n = len(weights)
s = sum(weights)


mean = s / n
```

あるオブジェクトを代入すると、他のオブジェクトが返ってくるオブジェクトを関数という。

関数	機能
len(x)	リスト x の要素数を調べる
sum(x)	リスト x の全要素の合計を求める

問題 O2-1

コシヒカリは複数の県で植えられている。各県におけるコシヒカリの 10a あたりの収量を次の表にまとめた。コシヒカリの 10a あたりの平均収量を計算せよ。

都道府県名	収量 (kg/10a)
新潟	528.3
茨城	520.7
福島	538.8
栃木	535.2
千葉	512.0

```
koshihikari = [528.3, 520.7, 538.8, 535.2]
```

問題 O2-2

各県における３品種の米の収穫量を次の表にまとめた。各品種に対して、10a あたりの平均収量を計算せよ。

都道府県名	品種	収量 (kg/10a)
新潟	コシヒカリ	528.3
茨城	コシヒカリ	520.7
福島	コシヒカリ	538.8
栃木	コシヒカリ	535.2
宮城	ひとめぼれ	519.4
岩手	ひとめぼれ	520.5
福島	ひとめぼれ	561.6
秋田	あきたこまち	567.5
岩手	あきたこまち	538.3
岡山	あきたこまち	501.9

リスト

リストは、同じ属性の要素が複数あったとき、それらの要素を束ねて扱うときに利用される。これらの要素には、添字（index）とよばれる番号が振り分けられている。添字を使用することで、特定の位置にある要素を取り出して、表示したり、再代入したりすることができる。リストの添字は、リストの先頭から 0、1、2、などのように整数が振り当てられている。添字を使って特定の要素を取り出すとき、w[0] のように、変数名のすぐ後に角括弧で添字を囲むように使用する。この使用方法は Python を使用する上での定義であり、括弧の形を変えることはできない。

```
w = [12, 10, 11, 13, 11]


w[0]


w[1]


w[5]
```

リスト

リストは、同じ属性の要素が複数あったとき、それらの要素を束ねて扱うときに利用される。これらの要素には、添字（index）とよばれる番号が振り分けられている。添字を使用することで、特定の位置にある要素を取り出して、表示したり、再代入したりすることができる。リストの添字は、リストの先頭から 0、1、2、などのように整数が振り当てられている。添字を使って特定の要素を取り出すとき、w[0] のように、変数名のすぐ後に角括弧で添字を囲むように使用する。この使用方法は Python を使用する上での定義であり、括弧の形を変えることはできない。

```
w = [12, 10, 11, 13, 11]


w[0]
# 12

w[1]
# 10

w[5]
# IndexError: list index out of range
```

リストのサイズを超えた添字を与えると、添字が範囲外にあるという IndexError が発生する。

リスト

リストは、同じ属性の要素が複数あったとき、それらの要素を束ねて扱うときに利用される。これらの要素には、添字（index）とよばれる番号が振り分けられている。添字を使用することで、特定の位置にある要素を取り出して、表示したり、再代入したりすることができる。リストの添字は、リストの先頭から 0、1、2、などのように整数が振り当てられている。添字を使って特定の要素を取り出すとき、w[0] のように、変数名のすぐ後に角括弧で添字を囲むように使用する。この使用方法は Python を使用する上での定義であり、括弧の形を変えることはできない。

```
w = [12, 10, 11, 13, 11]


w[0]
# 12

w[1]
# 10

w[5]
# IndexError: list index out of range

w[2] = 9


w
```

リスト

リストは、同じ属性の要素が複数あったとき、それらの要素を束ねて扱うときに利用される。これらの要素には、添字（index）とよばれる番号が振り分けられている。添字を使用することで、特定の位置にある要素を取り出して、表示したり、再代入したりすることができる。リストの添字は、リストの先頭から 0、1、2、などのように整数が振り当てられている。添字を使って特定の要素を取り出すとき、w[0] のように、変数名のすぐ後に角括弧で添字を囲むように使用する。この使用方法は Python を使用する上での定義であり、括弧の形を変えることはできない。

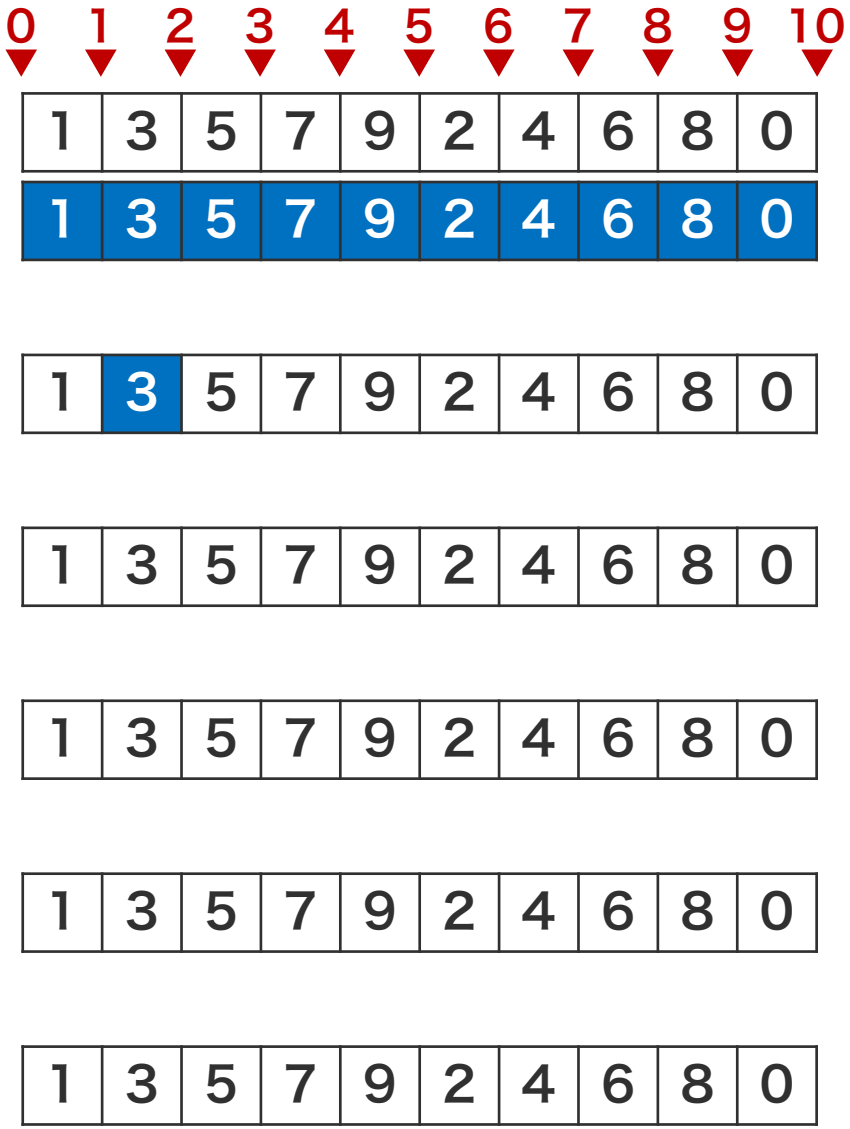
```
w = [12, 10, 11, 13, 11]



w[0]
# 12

w[1]
# 10

w[5]
# IndexError: list index out of range

w[2] = 9



w
# [12, 10, 9, 13, 11]
```

リスト

リストの中から、連続した要素をまとめて取り出す（スライス）とき、":" を使うと便利である。

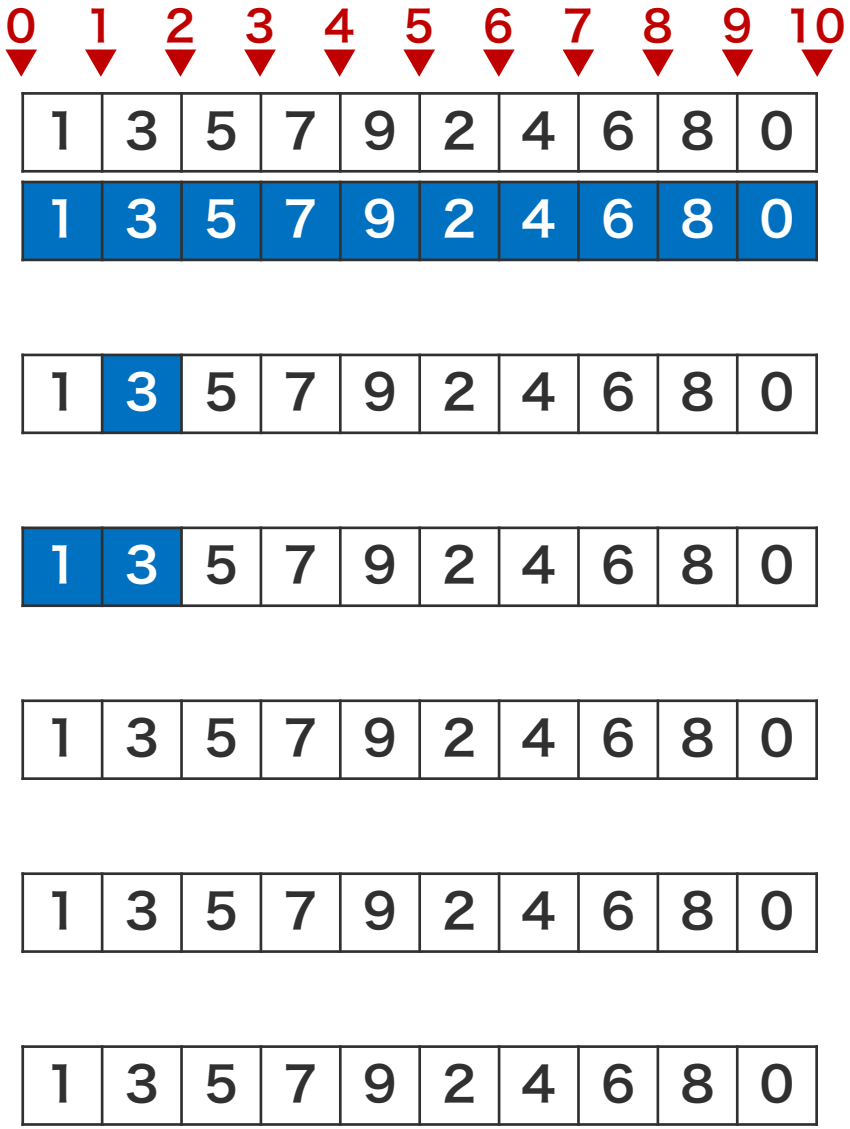
```
a = [1, 3, 5, 7, 9, 2, 4, 6, 8, 0]



a
# [1, 3, 5, 7, 9, 2, 4, 6, 8, 0]

a[1]
# 3

a[0:2]


a[2:6]


a[:4]


a[5:]
```

リスト



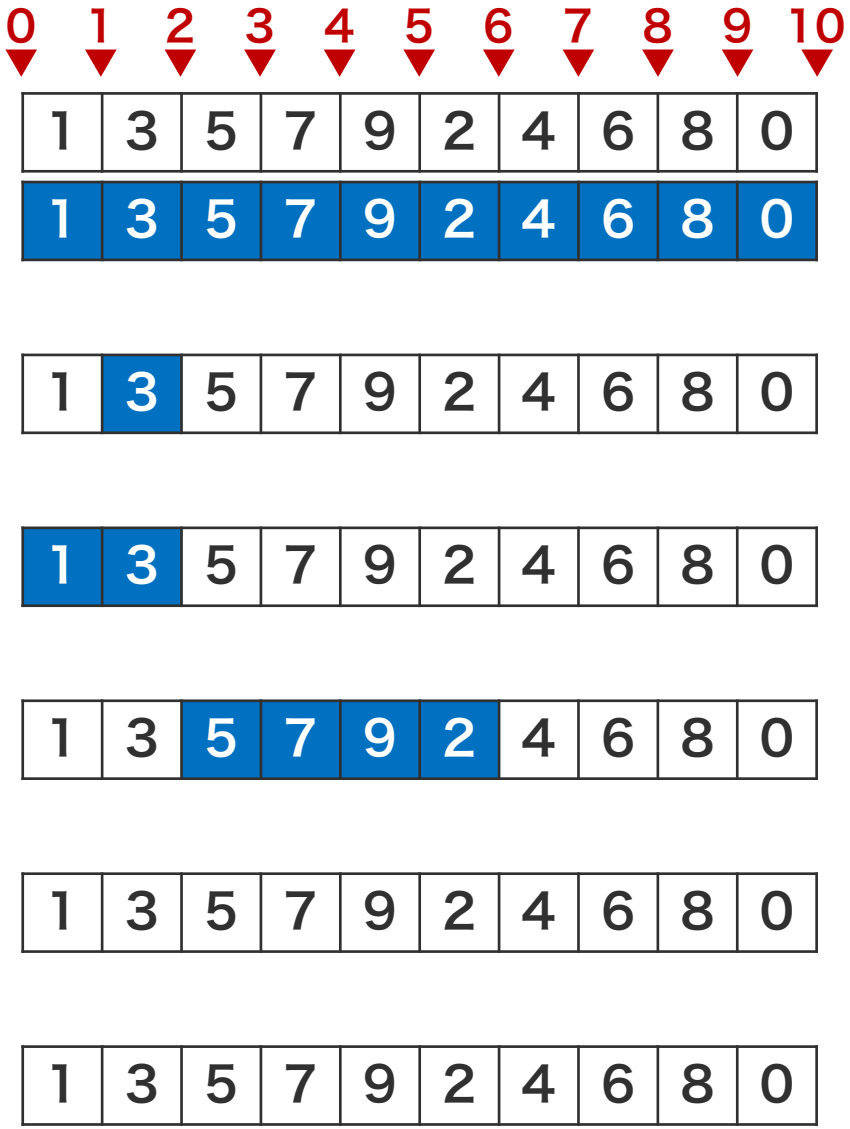

```
a = [1, 3, 5, 7, 9, 2, 4, 6, 8, 0]


a
# [1, 3, 5, 7, 9, 2, 4, 6, 8, 0]

a[1]
# 3

a[0:2]


a[2:6]


a[:4]


a[5:]
```

リスト



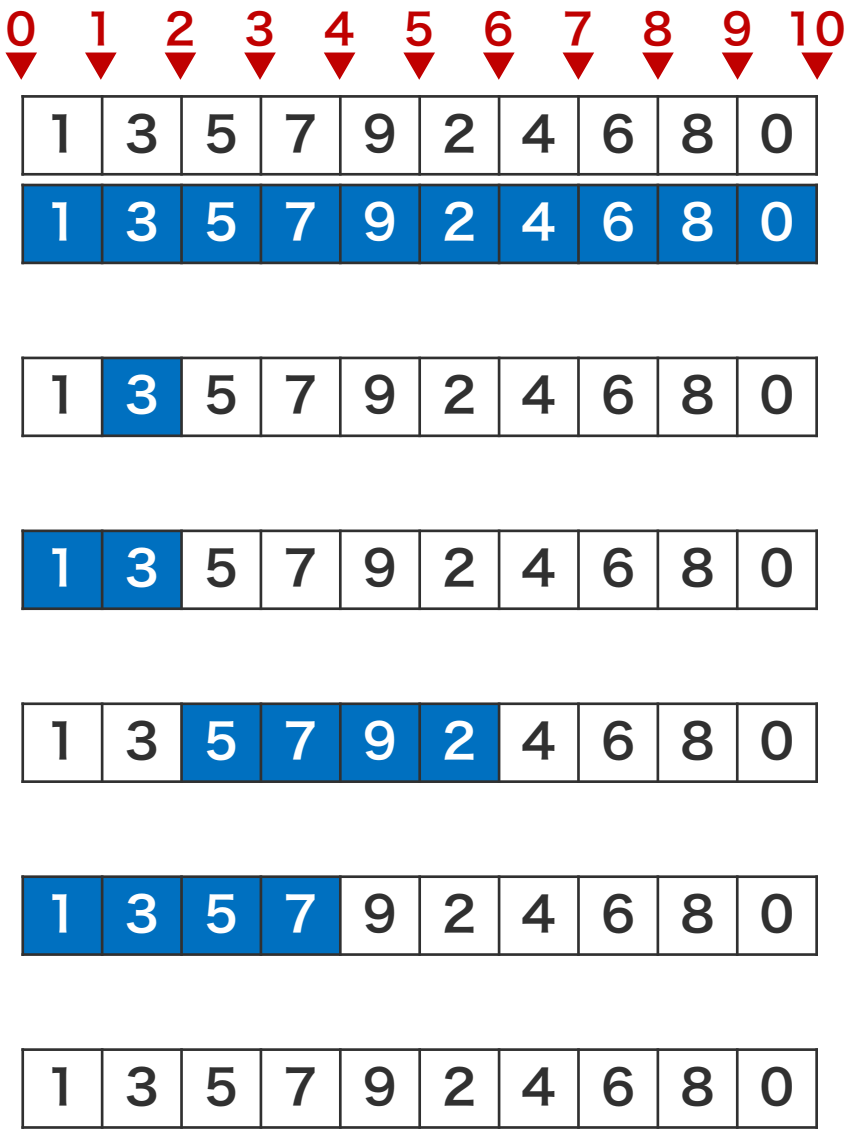

```
a = [1, 3, 5, 7, 9, 2, 4, 6, 8, 0]


a
# [1, 3, 5, 7, 9, 2, 4, 6, 8, 0]

a[1]
# 3

a[0:2]
# [1, 3]

a[2:6]


a[:4]


a[5:]
```

リスト



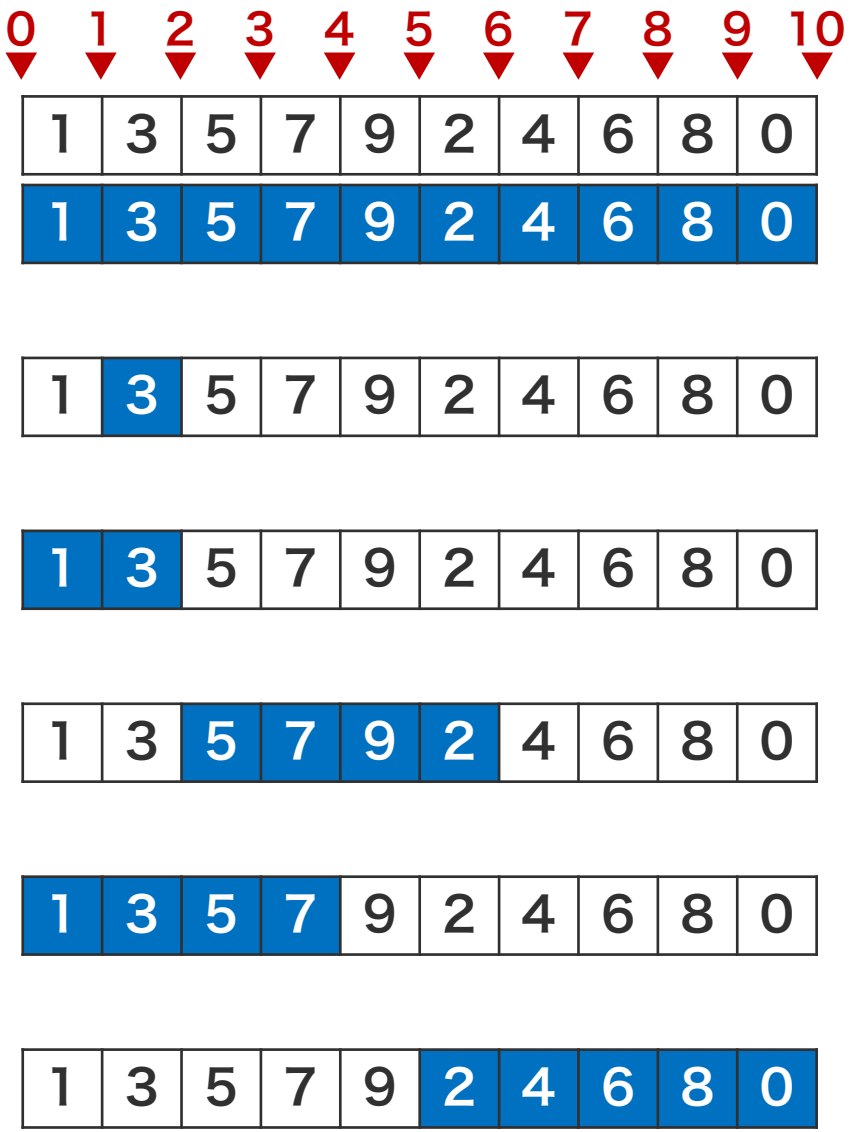

```
a = [1, 3, 5, 7, 9, 2, 4, 6, 8, 0]


a
# [1, 3, 5, 7, 9, 2, 4, 6, 8, 0]

a[1]
# 3

a[0:2]
# [1, 3]

a[2:6]
# [5, 7, 9, 2]

a[:4]


a[5:]
```

リスト




```
a = [1, 3, 5, 7, 9, 2, 4, 6, 8, 0]


a
# [1, 3, 5, 7, 9, 2, 4, 6, 8, 0]

a[1]
# 3

a[0:2]
# [1, 3]

a[2:6]
# [5, 7, 9, 2]

a[:4]
# [1, 3, 5, 7]

a[5:]
```

リスト




```
a = [1, 3, 5, 7, 9, 2, 4, 6, 8, 0]


a
# [1, 3, 5, 7, 9, 2, 4, 6, 8, 0]

a[1]
# 3

a[0:2]
# [1, 3]

a[2:6]
# [5, 7, 9, 2]

a[:4]
# [1, 3, 5, 7]

a[5:]
# [2, 4, 6, 8, 0]
```

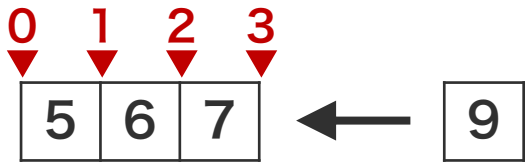
リスト操作

すでに作られたリストに新しい要素を追加することができる。リストの後尾に要素を追加するには append 関数（メソッド）を使用する。また、リストの先頭あるいは指定した位置に要素を挿入するには insert 関数（メソッド）を使用する。

関数	機能
append	リストの後尾に要素を 1 つだけ追加する。
extend	リストの後尾に要素を複数個追加する。
insert	リストの位置 i に要素を 1 つだけ挿入する。
pop	リストの位置 i にある要素を削除する。

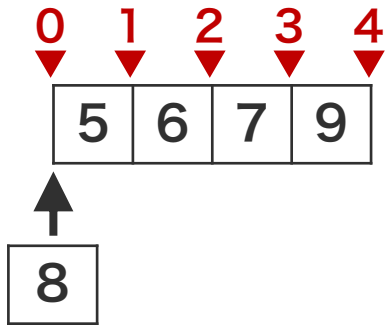
```
a = [5, 6, 7]


a.append(9)
a


a.insert(0, 8)
a


a.insert(2, 1)
a


a.append(4)
a
```

リスト操作



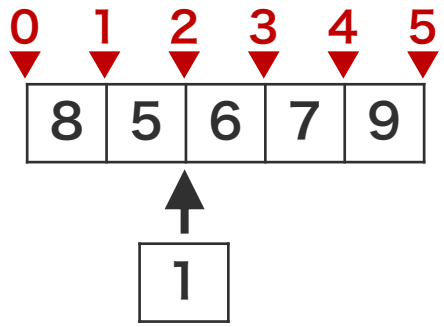

```
a = [5, 6, 7]

a.append(9)
a
# [5, 6, 7, 9]


a.insert(0, 8)
a



a.insert(2, 1)
a



a.append(4)
a
```

リスト操作



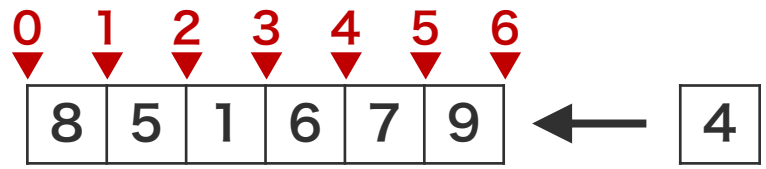

```
a = [5, 6, 7]

a.append(9)
a
# [5, 6, 7, 9]


a.insert(0, 8)
a
# [8, 5, 6, 7, 9]


a.insert(2, 1)
a



a.append(4)
a
```

リスト操作




```
a = [5, 6, 7]

a.append(9)
a
# [5, 6, 7, 9]


a.insert(0, 8)
a
# [8, 5, 6, 7, 9]


a.insert(2, 1)
a
# [8, 5, 1, 6, 7, 9]


a.append(4)
a
```

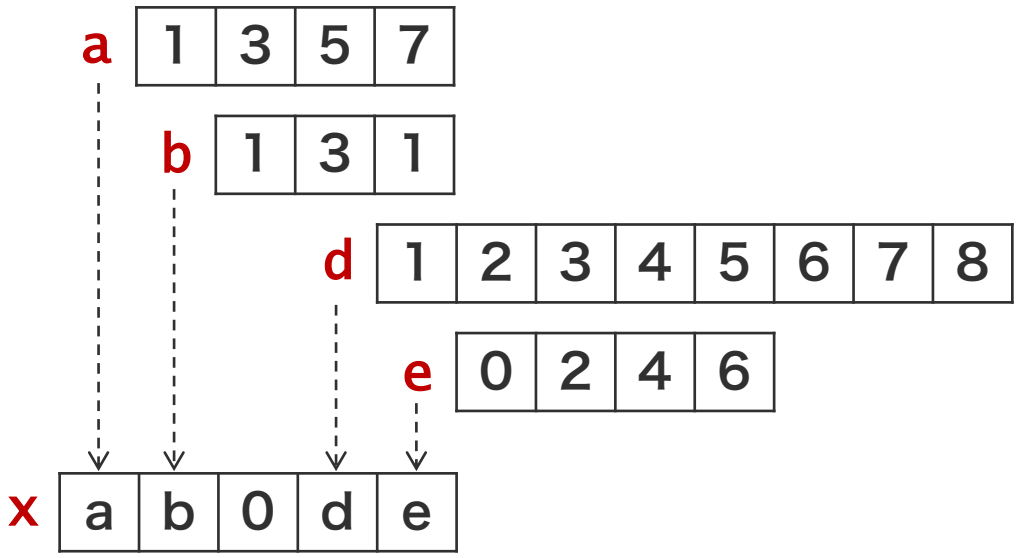
リスト操作




```
a = [5, 6, 7]

a.append(9)
a
# [5, 6, 7, 9]


a.insert(0, 8)
a
# [8, 5, 6, 7, 9]


a.insert(2, 1)
a
# [8, 5, 1, 6, 7, 9]


a.append(4)
a
# [8, 5, 1, 6, 7, 9, 4]
```

二次元リスト

リストは複数の要素を持つことができる。これまでに見てきた各要素は、一つ一つの数値であった。実は、リストに代入できる要素は、Python のオブジェクトであればよい。つまり、リストの中に、リストを代入することもできる。このようなリストを二次元リスト、多次元リストと呼んだりする。

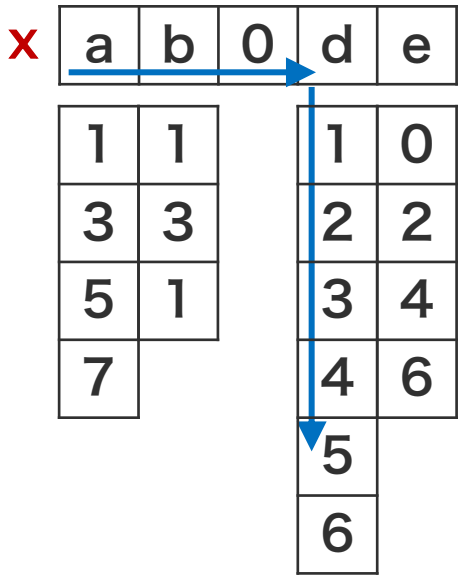



```
a = [1, 3, 5, 7]
b = [1, 3, 1]
d = [1, 2, 3, 4, 5, 6, 7, 8]
e = [0, 2, 4, 6]

x = [a, b, 0, d, e]
```

二次元リスト

リストは複数の要素を持つことができる。これまでに見てきた各要素は、一つ一つの数値であった。実は、リストに代入できる要素は、Python のオブジェクトであればよい。つまり、リストの中に、リストを代入することもできる。このようなリストを二次元リスト、多次元リストと呼んだりする。

```
a = [1, 3, 5, 7]
b = [1, 3, 1]
d = [1, 2, 3, 4, 5, 6, 7, 8]
e = [0, 2, 4, 6]

x = [a, b, 0, d, e]

x[3][5]
# 6
```



問題 O2-3

米３品種の栽培状況を調べた結果を次の表のようにまとめた。これらの情報を一つのオブジェクト w に保存せよ。ただし、必要に応じて二次元リストを使うこと。

都道府県名	品種	収量 (kg/10a)
新潟	コシヒカリ	528.3
茨城	コシヒカリ	520.7
福島	コシヒカリ	538.8
栃木	コシヒカリ	535.2
宮城	ひとめぼれ	519.4
岩手	ひとめぼれ	520.5
福島	ひとめぼれ	561.6
秋田	あきたこまち	567.5
岩手	あきたこまち	538.3
岡山	あきたこまち	501.9

```
koshihikari = [528.3, 520.7, 538.8,
                535.2, 512.0]
hitomebore = [519.4, 520.5, 561.6]
akitakomachi = [567.5, 538.3, 501.9]
```

w

基本オブジェクト

- スカラー
- リスト
- ディクショナリ
- セット
- タプル

ディクショナリ

ディクショナリは、リストと同じく、複数の要素を束ねて保存できるオブジェクトである。リストは各要素を index と呼ばれる添字で管理しているのに対して、ディクショナリは各要素を名前（キー）で管理している。キーは文字列であるため、そのまま入力するとオブジェクト名に間違えられる。そのため、キーを " または ' で囲む必要がある。

樹木	櫟	樫	楢
どんぐり			
重さ	2.0	1.3	2.8

1. キーと値をペアで与える
2. 複数のペアをカンマ区切りで並べる
3. 波括弧 (curly bracket) で全体をまとめる

```
a = {
'buna': 2.0,
'kashi': 1.3,
'nara': 2.8
}
```

ディクショナリ

ディクショナリの要素を取り出すときは、変数名に続けて、角括弧を書き、角括弧の中にキー入れる。括弧の形を変更することはできない。

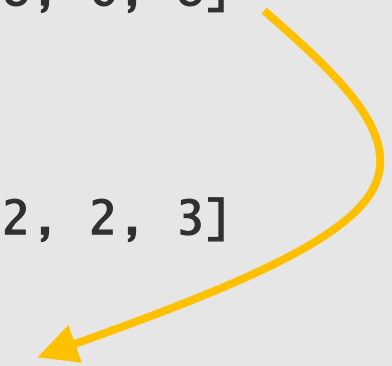
```
weights = {
    'buna': 2.0,
    'kashi': 1.3,
    'nara': 2.8
}


weights['buna']
# 2.0


weights['kashi']
# 1.3


weights['shii']
# KeyError: 'shii'
```

存在しないキーを与えると、キーが見つからないというKeyError が起こる。

ディクショナリ

既存のディクショナリの値を更新するときは、キーを指定し、代入演算子で新しい値を代入する。

```
weights = {
    'buna': [12, 11, 13, 14, 13],
    'kashi': [9, 9, 8, 9],
    'nara': [6, 7, 8, 6, 8]
}


weights['buna'] = [2, 2, 3]

weights
# {'buna': [2, 2, 3], 'kashi': [9, 9, 8,
9], 'nara': [6, 7, 8, 6, 8]}
```



ディクショナリ

既存のディクショナリの値を更新するときは、キーを指定し、代入演算子で新しい値を代入する。また、ディクショナリに存在していないキーに対して、値を代入すると、そのキーと値のペアは、ディクショナリに新規追加される。

```
weights = {
    'buna': [12, 11, 13, 14, 13],
    'kashi': [9, 9, 8, 9],
    'nara': [6, 7, 8, 6, 8]
}


weights['shii'] = [2, 2, 3]

weights
# {'buna': [12, 11, 13, 14, 13],
'kashi': [9, 9, 8, 9], 'nara': [6, 7, 8,
6, 8], 'shii': [2, 2, 3]}
```

問題 O3-1

赤色で書かれたオブジェクトが保持している値を答えよ。

```
d = {'store': 'shop',
     'color': 'colour',
     'gray': 'grey'}


d['color']



d['movie']
```

曜日の略語を入力すると、その正式な単語を返すディクショナリを作成せよ。

```
d = {









}

d['FRI']
# Friday

d['MON']
# Monday
```

基本オブジェクト

- スカラー
- リスト
- ディクショナリ
- セット
- タプル

セット

集合は、リストと同様に複数の要素を保持できる。しかし、同じ値を持つ要素は 1 つしか持てない。また、集合には順序関係が存在しないため、位置番号で要素を取得しようとするとエラーが起こる。

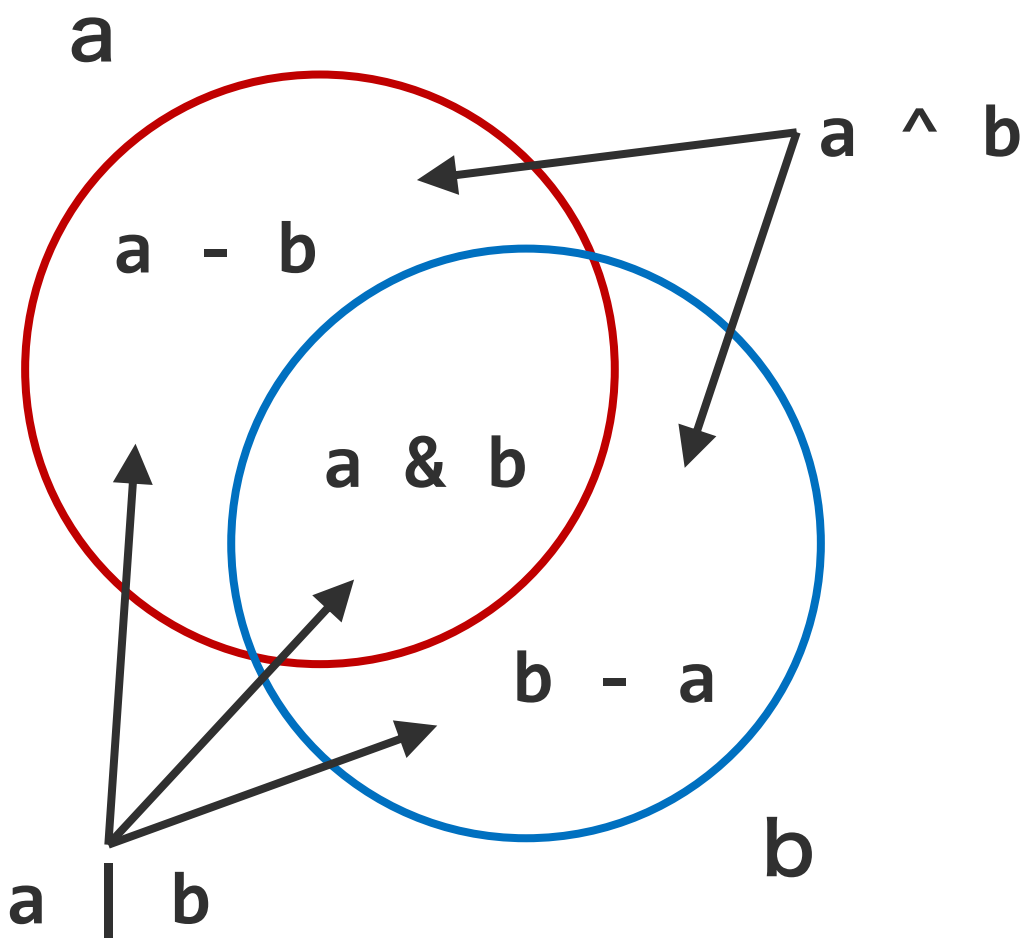
順序集合を取り扱う場合は、collections パッケージ中の OrderedDict 型のオブジェクトを使用する。この資料では順序集合を取り上げない。

```
a = {1, 3, 1, 5, 9, 7}
a
# {1, 3, 5, 7, 9}


a[1]
# TypeError


for i in a:
  print(i)
# 1
# 3
# 5
# 7
# 9
```

集合演算

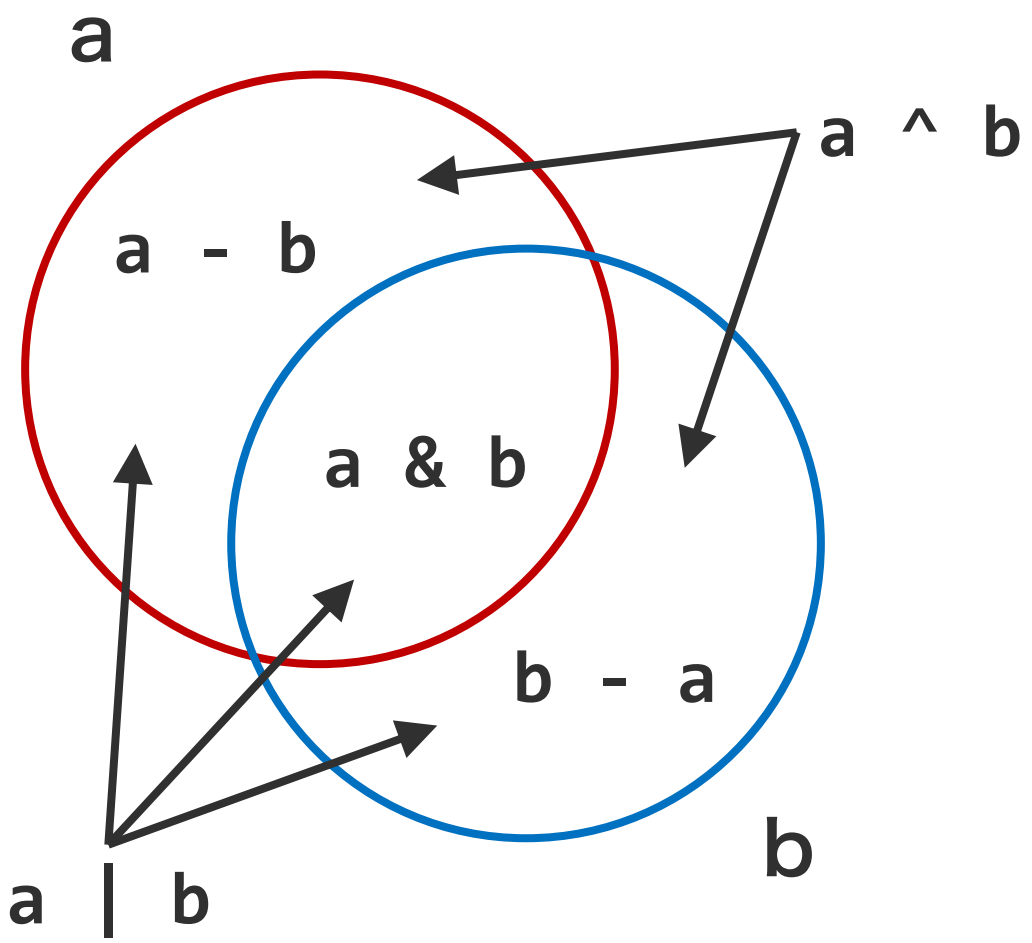
集合は、リストに含まれる重複要素を取り除いたり、２つのリストを比較して共通要素を取り出したりする際に使われる。ただし、リストを集合に変更した時に順位情報が失われるので、集合を再びリストに戻した時は元の順序が失われる。

演算子	動作
a | b	集合 a と集合 b の和集合
a & b	集合 a と集合 b の積集合
a - b	集合 a と集合 b の差集合
a ^ b	集合 a と集合 b の対象差集合
a <= b	集合 a は集合 b に含まれているかどうかを判定



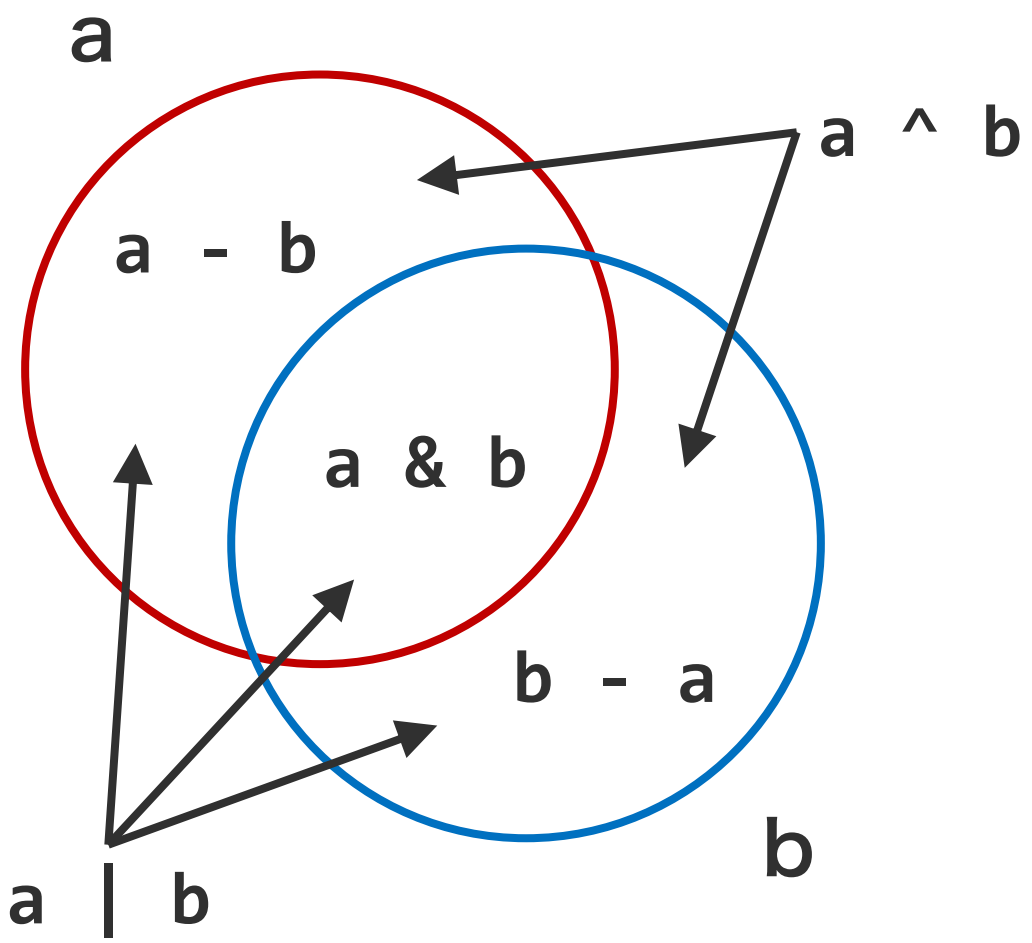
集合演算

```
a = [1, 1, 2, 1, 3, 1, 4, 5]
b = [2, 4, 6, 8, 10, 12, 14]
c = [3, 6, 9, 12, 15, 18, 21]

x = list(set(a))
x
# [1, 2, 3, 4, 5]


x = list(set(b) | set(c))
x
# [2, 3, 4, 6, 8, 9, 10, 12, 14, 15, 18, 21]


x = list(set(b) & set(c))
x
# [12, 6]
```



集合演算

```
a = [1, 1, 2, 1, 3, 1, 4, 5]
b = [2, 4, 6, 8, 10, 12, 14]
c = [3, 6, 9, 12, 15, 18, 21]

x = list(set(b) - set(c))
x
# [2, 4, 8, 10, 14]


x = list(set(c) - set(b))
x
# [3, 9, 15, 18, 21]


x = list(set(b) ^ set(c))
x
# [2, 3, 4, 8, 9, 10, 14, 15, 18, 21]
```



基本オブジェクト

- スカラー
- リスト
- ディクショナリ
- セット
- タプル

タプル

タプルは、リストとほぼ同じようなオブジェクトである。リストは、一度作成したあとに、値を追加したり、更新したりすることができる。これに対して、タプルは、一度作成すると、あとで値の追加や更新ができない。そのため、タプルを読み取り専用のリスト見なせる。

```
a = (1, 1, 2, 3, 5, 8, 12)


a[4]
# 5


a[6] = 13
# TypeError: 'tuple' object does not
support item assignment
```

順序あり

リスト・タプル

複数の要素を、ゼロから数え始める位置番号で管理するオブジェクト。

順序なし

ディクショナリ

複数の要素をキーで管理している。キーの重複は許容されない。

セット

複数の要素を順序情報なしで管理している。重複した要素がある場合、1 つだけ残り、ほかはすべて削除される。

Python 文法（基礎）

- プログラミング言語
- 基本オブジェクト
- 基本文法

基本文法

- 予約語
- 条件構文
- 繰り返し構文
- 関数
- パッケージ

予約語

条件判定		条件構文	繰り返し構文	関数		例外処理	いろいろ	
True	is	if	for	def	import	try	lambda	async
False	not	elif	while	return	from	except	del	await
None	in	else	pass	yield	as	finally	zip	global
	or		break		class	raise	assert	with
	and		continue				nonlocal	
			enumerate					

基本文法

- 予約語
- 条件構文
- 繰り返し構文
- 関数
- パッケージ

イートインスペースのあるパン屋さんで、アップルパイを 1 つとメロンパンを 1 つ購入した。合計いくら支払えばよいか。ただし、店内で食べる場合の消費税率を 10% とし、持ち帰りの場合の消費税率を 8% とする。

アップルパイ	180
メロンパン	120
小計	300

takeout

300 × 1.08 = 324

eatin

300 × 1.10 = 330

if 構文

Python の条件構文は if、elif、else などの単語を使う。Python の条件構文は必ず if から始まる。条件構文の 1 行目には、if とともに条件を書く。条件判定後の処理は、2 行目以降に書く。ただし、条件判定後の処理は、条件構文の一部であることを明示するために、行の先頭にインデントを入れる。

もし持ち帰りならば ▶

もし店内で食事するならば ▶

```
a = 180
b = 120
s = 0


takeout = True

eatin = False


if takeout is True :
    s = (a + b) * 1.08


if eatin is True:
    s = (a + b) * 1.10


s
# 324
```

if 構文

```
a = 180
b = 120
s = 0
takeout = True

if takeout is True :
    s = (a + b) * 1.08

print(s)
```

判定条件

条件構文の開始 ▶

インデント

条件構文の終了 ▶

条件構文ブロック

インデントの数で、条件構文ブロックが終了しているかどうかを判定している。

インデント

インデントは、タブまたはスペースキーで入力する。

- タブ 1 個分
- スペース 4 個分 ✓
- スペース 2 個分

英語キーボード

日本語キーボード

if 構文

```
a = 180
b = 120
s = 0
takeout = True

if takeout is True :
    s = (a + b) * 1.08

print(s)
```

判定条件

条件構文の開始 ▶

インデント

条件構文の終了 ▶

真判定の場合は、「is True」 を 省略するのが 正しい書き方である。

```
a = 180
b = 120
s = 0
takeout = True

if takeout:
    s = (a + b) * 1.08

print(s)
```

論理演算子

条件構文は、与えられた条件が真（True）であるかどうかを判定して、分岐処理を行う構文である。条件は不等式で与えるのが一般的である。

論理演算子	処理
a == b	a と b が等しいならば True
a != b	a と b が等しくなければ True
a > b	a が b よりも大きければ True
a >= b	a が b 以上ならば True
a < b	a が b よりも小さければ True
a <= b	a が b 以下ならば True

※表中の論理演算子のほか、is と is not も理論演算に使われる。"a is b" は a と b が同じ属性（かつ同じ値）を持ったオブジェクトである場合に True を返す演算子である。気になる方は、各自で調べてください。

```
a = 4
b = 3
c = 0
d = 0



if a > 3:
    c = 1
c
# 1



if b > 10:
    d = 1
d
# 0
```

問題 S1-1

イートインスペースのあるパン屋さんで、アップルパイを 1 つとメロンパンを 1 つ、購入して持ち帰った。商品購入時に、可能であればクーポン券を使う。このとき、合計金額を計算せよ。

ただし、

- アップルパイは 1 つ 180 円、メロンパンは 1 つ 120 円である。
- クーポン券は、合計金額が 1000 円を超えた場合のみに、5% OFF される。
- 店内で食べる場合の消費税率を 10% で、持ち帰りの場合の消費税率を 8% とする。

```
apple = 180
melon = 120

takeout = True
eatin = False


# total amount


# can use a coupon or not?


# eat-in or takeout ?
```

論理演算

条件構文は、与えられた条件が真（True）であるかどうかを判定して、分岐処理を行う構文である。複数の条件を同時に判断することもできる。論理演算で最もよく使われる演算として論理積と論理和である。論理積は、英語の AND に相当するものである。論理和は、英語の OR に相当するものである。

論理演算子	演算
and	論理積
&	論理積
or	論理和
|	論理和
^	排他的論理和

```
apple = 180 * 1
melon = 120 * 1
plain = 400 * 2
takeout = True
eatin = False
coupon = True
s = apple + melon + plain

if (coupon) and (s > 1000):
    s = s * 0.95

if takeout:
    s = s * 1.08

if eatin:
    s = s * 1.10

s
# 1128.6
```

問題 S1-2

下に示した 2 種類の割引処理の違いを、論理演算子に注意しながら、説明してみてください。それぞれがどのような条件で割引されるのか。

```
coupon = True

s = 1100




if (coupon)  and  (s > 1000) :

    s = (a + b) * 0.95

s
```

```
coupon = True

s = 1100




if (coupon)  or  (s > 1000) :

    s = (a + b) * 0.95

s
```

論理演算

条件 1	条件 2	論理積	論理和	排他的論理和
x	y	x and y	x or y	x ^ y
真	真	真	真	偽
真	偽	偽	真	真
偽	真	偽	真	真
偽	偽	偽	偽	偽

問題 S1-3

赤色で書かれたオブジェクトが保持している値を答えよ。

```
a = 4
b = 3
c = 0

if (a > 0) and (b > 0):
    c = 1

c
```

```
a = 1
b = 4
c = 0

if (a > 0) or (b < 2):
    c = 1

c
```

if-else 構文

これまでに見てきた条件構文は「もし〜ならば・・・をする」のように、ある条件を満たせば、特定の 1 つの処理を行うものであった。これに対して、ある条件を満たしたときに処理 A を、満たさなかった時に処理 B を行いたい場合は、if 構文を続けて 2 つ書けば実現できる。

```
a = 180
b = 120
s = 0


takeout = True
eatin = False



if takeout:
    s = (a + b) * 1.08


if eatin:
    s = (a + b) * 1.10

s
# 324
```

if-else 構文

ある条件の真偽判定に基づいて処理を分けたい場合は、if 構文を 2 つ書くことで対応できる。しかし、if 構文を 2 つ続けて書くことで、同じ条件を 2 回判断していることになる。理論的にも、処理的にも煩雑である。このような場合は、if と else を用いて条件構文を作成すると、理論的にも処理的にもシンプルになる。

```
a = 180
b = 120
s = 0


takeout = True
eatin = False


if takeout:
    s = (a + b) * 1.08


if eatin:
    s = (a + b) * 1.10


s
# 324
```

```
a = 180
b = 120
s = 0


takeout = True



if takeout:
    s = (a + b) * 1.08


else:
    s = (a + b) * 1.10


s
# 324
```

if-else 構文

ある条件の真偽判定に基づいて処理を分けたい場合は、if 構文を 2 つ書くことで対応できる。しかし、if 構文を 2 つ続けて書くことで、同じ条件を 2 回判断していることになる。理論的にも、処理的にも煩雑である。このような場合は、if と else を用いて条件構文を作成すると、理論的にも処理的にもシンプルになる。

```
a = 180
b = 120
s = 0


takeout = True



if takeout:
    s = (a + b) * 1.08

else:
    s = (a + b) * 1.10

s
# 324
```

条件判定 ▶

条件成立時に実行

条件不成立時に実行

問題 S1-4

イートインスペースのあるパン屋さんで、アップルパイを 1 つとメロンパンを 1 つ、食パンを 2 袋購入した。このとき、合計金額を計算せよ。なお、商品購入時に、可能であればクーポン券を使用する。

ただし、

- アップルパイは 1 つ 180 円、メロンパンは 1 つ 120 円、食パンは 1 袋 400 円である。
- クーポン券は、合計金額が 1000 円を超えた場合のみに、5% OFF される。
- 店内で食べる場合の消費税率を 10% で、持ち帰りの場合の消費税率を 8% とする。
- 店内で食べるか持ち帰るかの判断を if-else 文で判断し、また、クーポンが使えるかどうかの判断を if 文で判断せよ。

```
apple = 180 * 1
melon = 120 * 1
plain = 400 * 2
```

if-elif-else 構文

1 つのオブジェクトに対して複数の閾値を設けて条件判定したい場合は、複数の if 文を使って書くことができる。しかし、この場合、ロジックも複雑になることに注意する必要がある。

得点	成績
90 点超え	excellent
80 点超え 90 以下	good
60 点超え 80 以下	passing
60 以下	failing

```
a = 81
b = ''

if a <= 60:
    b = 'failing'

if a > 60:
    b = 'passing'

if a > 80:
    b = 'good'

if a > 90 :
    b = 'excellent'



b
```

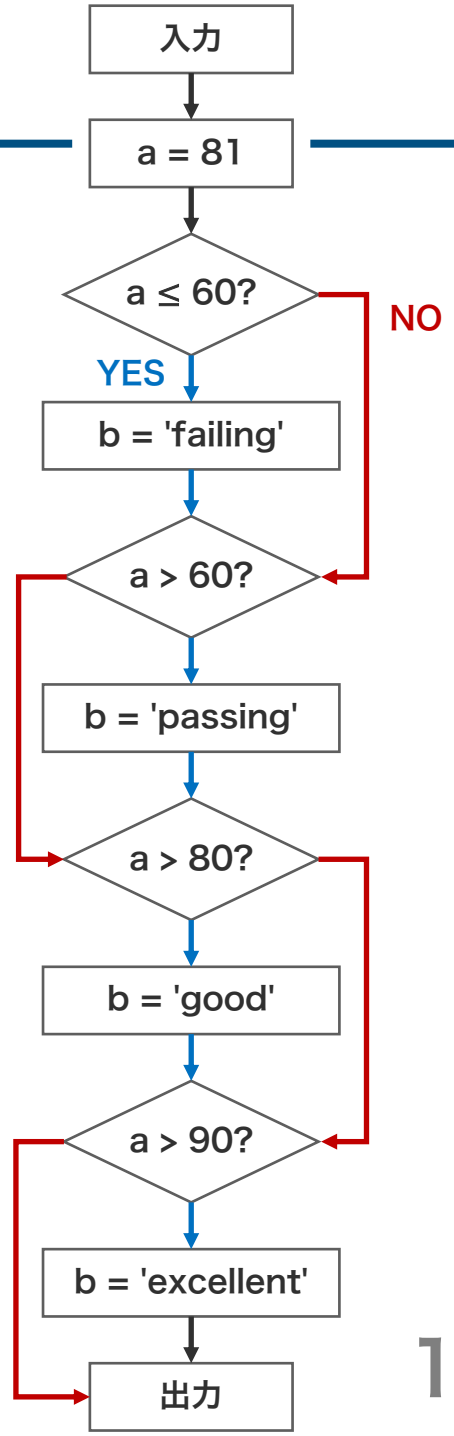
```
a = 81
b = ''

if a > 90:
    b = 'excellent'

elif a > 80:
    b = 'good'

elif a > 60:
    b = 'passing'

else:
    b = 'failing'



b
```

if-elif-else 構文

1 つのオブジェクトに対して複数の閾値を設けて条件判定したい場合は、複数の if 文を使って書くことができる。しかし、この場合、ロジックも複雑になることに注意する必要がある。

得点	成績
90 点超え	excellent
80 点超え 90 以下	good
60 点超え 80 以下	passing
60 以下	failing

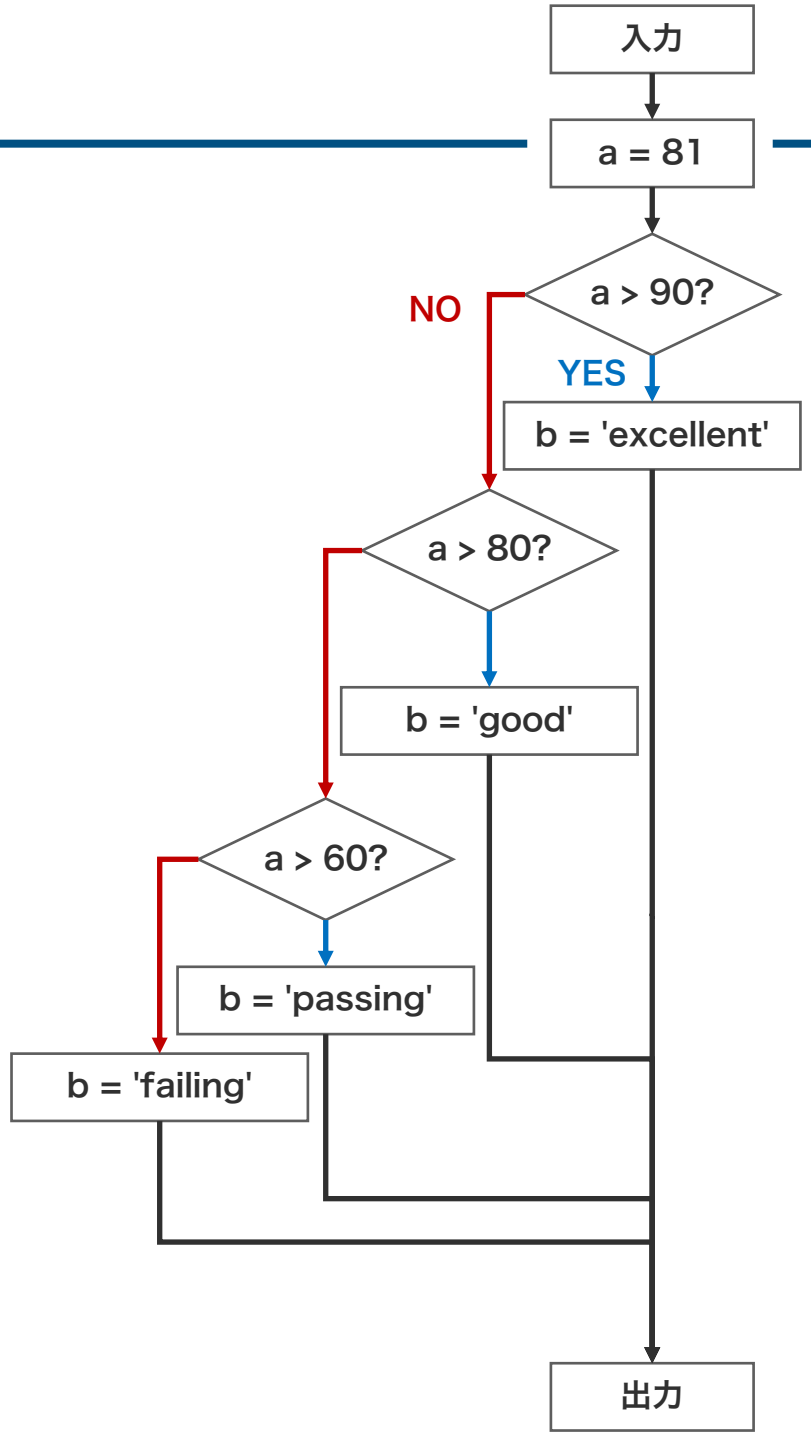
```
a = 81
b = ''

if a <= 60:
    b = 'failing'

if a > 60:
    b = 'passing'

if a > 80:
    b = 'good'

if a > 90 :
    b = 'excellent'



b
```



if-elif-else 構文

1 つのオブジェクトに対して複数の閾値を設けて条件判定したい場合は、複数の if 文を使って書くことができる。しかし、この場合、ロジックも複雑になることに注意する必要がある。

得点	成績
90 点超え	excellent
80 点超え 90 以下	good
60 点超え 80 以下	passing
60 以下	failing




```
a = 81
b = ''

if a > 90:
    b = 'excellent'

elif a > 80:
    b = 'good'

elif a > 60:
    b = 'passing'

else:
    b = 'failing'



b
```

if-elif-else 構文

分岐条件が複数あるとき、if 構文で処理するのか、あるいは if-elif-else 構文で処理するのかについて、初めは特に拘る必要はない。使いやすい方を利用すれば良い。

得点	成績
90 点超え	excellent
80 点超え 90 以下	good
60 点超え 80 以下	passing
60 以下	failing

```
a = 81
b = ''

if a <= 60:
    b = 'failing'

if a > 60:
    b = 'passing'

if a > 80:
    b = 'good'

if a > 90 :
    b = 'excellent'



b
```

```
a = 81
b = ''

if a > 90:
    b = 'excellent'

elif a > 80:
    b = 'good'

elif a > 60:
    b = 'passing'

else:
    b = 'failing'



b
```

if-elif-else 構文

分岐条件が複数あるとき、if 構文で処理するのか、あるいは if-elif-else 構文で処理するのかについて、初めは特に拘る必要はない。使いやすい方を利用すれば良い。ただし、いずれの方法でも判定条件の順序に注意を払う必要がある。

得点	成績
90 点超え	excellent
80 点超え 90 以下	good
60 点超え 80 以下	passing
60 以下	failing

```
a = 81
b = ''

if a > 90 :
    b = 'excellent'

if a > 80:
    b = 'good'

if a > 60:
    b = 'passing'

if a <= 60:
    b = 'failing'



b
# passing
```

```
a = 81
b = ''

if a > 60:
    b = 'passing'

elif a > 80:
    b = 'good'

elif a > 90:
    b = 'excellent'

else:
    b = 'failing'



b
# passing
```

問題 S1-4

赤色で書かれたオブジェクトが保持している値を答えよ。

```
a = 11
s = ''
if (3 <= a) and (a <= 5):
    s = 'spring'

elif (6 <= a) and (a <= 8):
    s = 'summer'

elif (9 <= a) and (a <= 11):
    s = 'autumn'

else:
    s = 'winter'



s
```

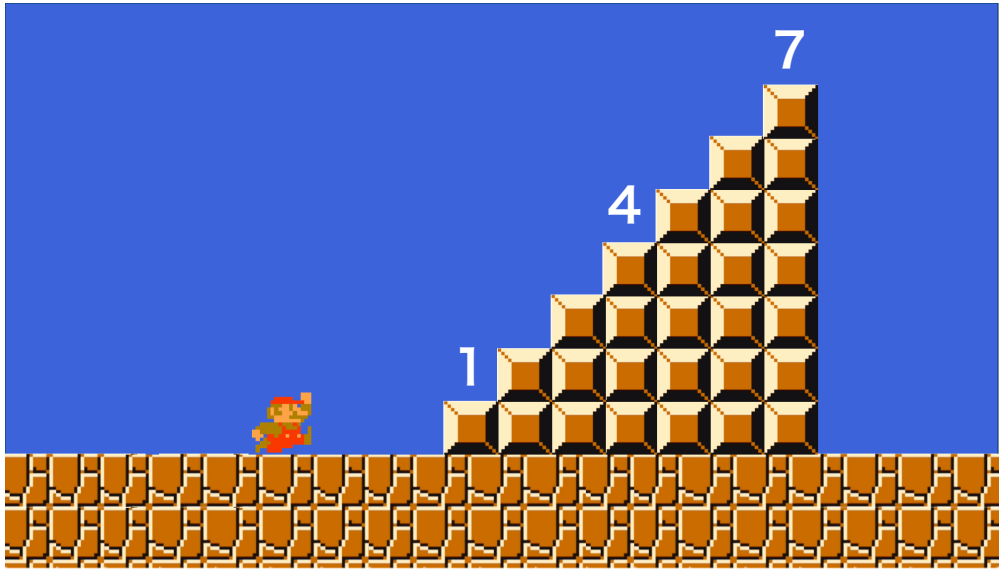
```
a = 4
d = ''
if a == 1:
    d = 'Mon'
elif a == 2:
    d = 'Tue'
elif a = 3:
    d = 'Wed'
elif a == 4:
    d = 'Thu'
ellf a == 5:
    d = 'Fri'
elif a == 6:
    d = Sat'
elif a == 7:
    d = 'Sun'
d
```

基本文法

- ○ 予約語
- ○ 条件構文
- ○ 繰り返し構文
- ○ 関数
- ○ パッケージ

while 構文

繰り返し構文には while 構文と for 構文の二種類がある。このうち、while 構文は、「与えた条件を満たす限り同じ作業を繰り返す」命令文である。例えば、「マリオが最上階に達するまで階段を登る」ことは、「《いまマリオが最上階よりも下にいる》という条件を満たす限り、階段を登る動作を続ける」に言い換えできる。




```
current_stage = 0


while current_stage < 7:
    goup(1)
    current_stage = current_stage + 1



print(current_stage)
```

while 構文

```
current_stage = 0

while current_stage < 7 :
    goup(1)
    current_stage = current_stage + 1

print(current_stage)
# 7
```

判定条件

繰り返し構文の開始 ▶

インデント

繰り返し構文の終了 ▶

繰り返し構文ブロック

インデントの数で、構文ブロックが終了しているかどうかを判定している。

while 構文

リストは複数の値を保持している。リスト中の各値に対して、同じ操作を繰り返したいときに while 構文が用いられる。例えば、リスト中の各要素を 1000 倍（kg 単位を g 単位に変更など）する場合は右のようにするこのとき、判定条件を「現在の位置がリストの末端でない」とすればよい。

現在処理している値の位置番号 ▶ が 6 未満ならば、下記の処理を続ける。

```
a = [1.2, 1.3, 2.1, 3.2, 1.9, 3.2]
len_a = 6



i = 0


while i < len_a:

    a[i] = a[i] * 1000

    i = i + 1


a
# [1200, 1300, 2100, 3200, 1900, 3200]
```

問題 S2-1

while 構文を使用して、リスト a の全要素の合計を求めよ。

```
a = [5, 2, 6, 3, 7, 5]
len_a = 6

s = 0

i = 0

while ??? :
    s = ???



s
```

問題 S2-2

リスト z の位置 i の要素が、リスト a の位置 i の要素とリスト b の位置 i の要素の和となるようなリスト z を作成せよ。

```
a = [1, 1, 2, 3, 4, 5]
b = [1, 3, 5, 7, 9, 10]
z = [0, 0, 0, 0, 0, 0]









z
# [2, 4, 7, 10, 13, 15]
```

for 構文

繰り返し構文には while 構文と for 構文の 2 種類がある。このうち、while 構文は、「与えた条件を満たす限り同じ作業を繰り返す」命令文であるのに対して、for 構文は「n 回繰り返す」命令文である。for 文を使用するとき、繰り返し回数 n をはじめに指定する必要がある。

```
a = [9, 12, 10, 11, 13]
len_a = 5


i = 0
while i < len_a:
    a[i] = a[i] * 1000




b = [9, 12, 10, 11, 13]
len_b = 5

for i in range(len_b):
    b[i] = b[i] * 1000
```

for 構文

```
b = [9, 12, 10, 11, 13]
len_b = 5

for i in range(len_b) :
    b[i] = b[i] * 1000

b
# [9000, 12000, 10000, 11000, 13000]
```

繰り返し回数　繰り返す範囲

繰り返し構文の開始 ▶

インデント

繰り返し構文の終了 ▶

繰り返し構文ブロック

インデントの数で、構文ブロックが終了しているかどうかを判定している。

for 構文

```
b = [9, 12, 10, 11, 13]
c = []

for x in b :
    c.append(x * 1000)

c
# [9000, 12000, 10000, 11000, 13000]
```

一時変数　繰り返す対象

繰り返し構文の開始 ▶

インデント

繰り返し構文の終了 ▶

リストの各要素に対して繰り返し処理を行う場合、繰り返す対象としてリスト名をそのまま与えることが一般的である。

繰り返し構文ブロック

インデントの数で、構文ブロックが終了しているかどうかを判定している。

問題 S2-3

for 構文を使用して、リスト a の全要素の合計を求めよ。

```
a = [5, 2, 6, 3, 7, 5]
n_len = 6

s = 0

for i in ??? :
    s = ???



s
```

問題 S2-4

リスト z の位置 i の要素が、リスト a の位置 i の要素とリスト b の位置 i の要素の和となるようなリスト z を作成せよ。

```
a = [1, 1, 2, 3, 4, 5]
b = [1, 3, 5, 7, 9, 10]
z = [0, 0, 0, 0, 0, 0]




z
# [2, 4, 7, 10, 13, 15]
```

問題 S2-5

リスト a の中の最大値を M に代入するプログラムを書け。

```
a = [1, 9, 2, 0, 4, 7]
M = None
```

代入する値はまだ決まっていないが、とりあえずこの名前のオブジェクトを用意しておく。

```
M
```

問題 S2-6

リスト a の中の偶数要素の個数を n に代入するプログラムを書け。

```
a = [1, 9, 2, 2, 4, 8, 7, 3]
n = None




n
```

基本文法

- ○ 予約語
- ○ 条件構文
- ○ 繰り返し構文
- ○ 関数
- ○ パッケージ

関数

同じ処理を繰り返すとき、その処理をまとめてパッケージ化することができる。このパッケージ化された処理群を関数という。お金のいらない自動販売機に例えるとわかりやすい。例えば、お茶ボタンを押したときにお茶が出て、コーヒーボタンを押したときにコーヒーが出てくる。このとき、自動販売機は、ボタンを通して入力値（引数）を受け取ると、それに対応する飲み物（戻り値）を出力している関数と言える。

関数

関数を使用しないで、支払い金額を計算するプログラムについて考えてみる。180 円のパンを 1 つ買ったときに支払うべき金額を、持ち帰りの有無・クーポンの使用の有無を考慮して、計算すると、右のようになる。

＊ クーポン券使用で、持ち帰りの場合

```
a = 180
takeout = True
use_coupon = True

s = 0

if use_coupon:
    s = a * 0.95

if takeout:
    s = s * 1.08
else:
    s = s * 1.10

s
```

関数

関数を使用しないで、支払い金額を計算するプログラムについて考えてみる。180 円のパンを 1 つ買ったときに支払うべき金額を、持ち帰りの有無・クーポンの使用の有無を考慮して、計算すると、右のようになる。

＊クーポン券使用で、店内で食べる場合

```
a = 180
takeout = False
use_coupon = True

s = 0

if use_coupon:
    s = a * 0.95

if takeout:
    s = s * 1.08
else:
    s = s * 1.10

s
```

関数

関数を使用しないで、支払い金額を計算するプログラムについて考えてみる。180 円のパンを 1 つ買ったときに支払うべき金額を、持ち帰りの有無・クーポンの使用の有無を考慮して、計算すると、右のようになる。

＊クーポン券使用しないで、店内で食べる場合

```
a = 180
takeout = False
use_coupon = False

s = 0

if use_coupon:
    s = a * 0.95

if takeout:
    s = s * 1.08
else:
    s = s * 1.10

s
```

関数

関数を使用しないで、支払い金額を計算するプログラムについて考えてみる。180 円のパンを 1 つ買ったときに支払うべき金額を、持ち帰りの有無・クーポンの使用の有無を考慮して、計算すると、右のようになる。

＊クーポン券使用しないで、持ち帰りの場合

```
a = 180
takeout = True
use_coupon = False

s = 0

if use_coupon:
    s = a * 0.95

if takeout:
    s = s * 1.08
else:
    s = s * 1.10

s
```

関数

関数を使用しないで、支払い金額を計算するプログラムについて考えてみる。140 円のパンを 1 つ買ったときに支払うべき金額を、持ち帰りの有無・クーポンの使用の有無を考慮して、計算すると、右のようになる。

＊クーポン券使用しないで、持ち帰りの場合

```
a = 140
takeout = True
use_coupon = False

s = 0

if use_coupon:
    s = a * 0.95

if takeout:
    s = s * 1.08
else:
    s = s * 1.10

s
```

関数

関数を使用しないで、支払い金額を計算するプログラムについて考えてみる。120 円のパンを 1 つ買ったときに支払うべき金額を、持ち帰りの有無・クーポンの使用の有無を考慮して、計算すると、右のようになる。

＊クーポン券使用で、持ち帰りの場合

```
a = 120
takeout = True
use_coupon = True

s = 0

if use_coupon:
    s = a * 0.95

if takeout:
    s = s * 1.08
else:
    s = s * 1.10

s
```

関数

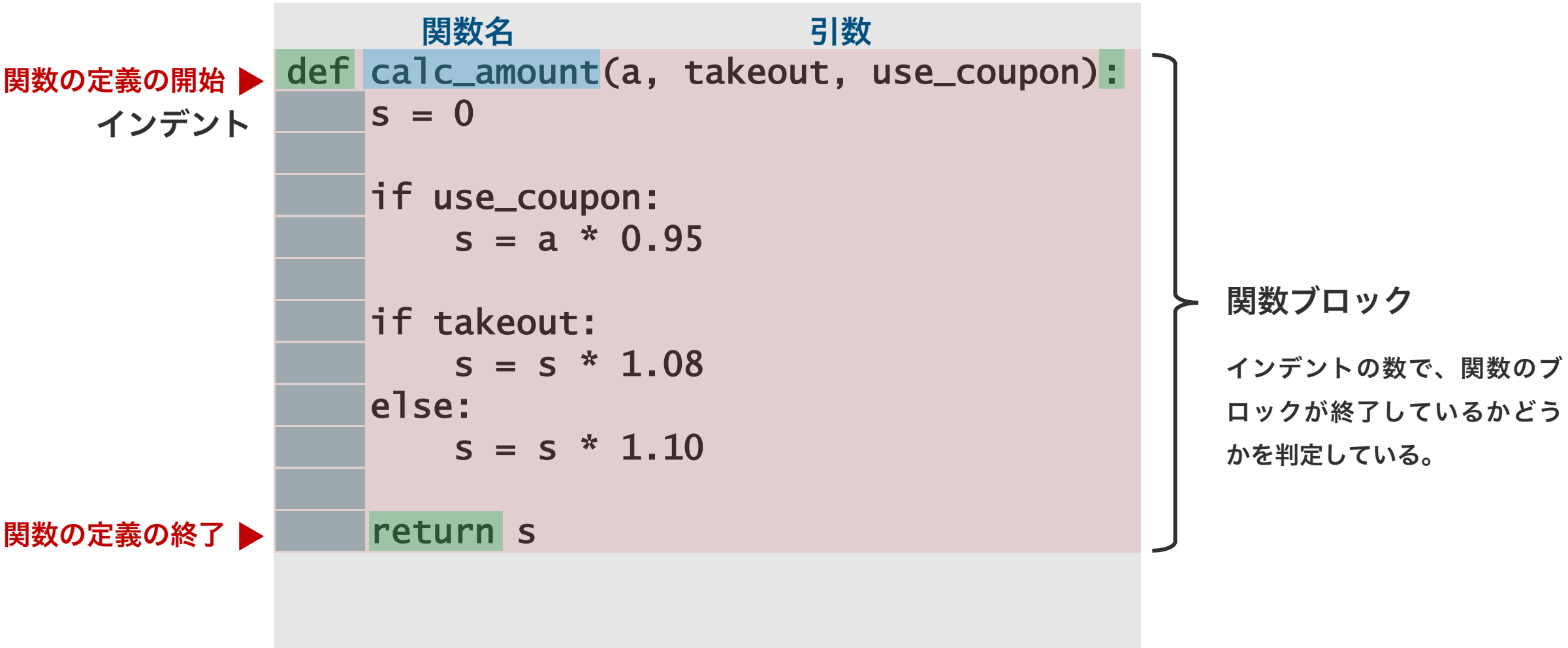
```
a = 120
takeout = True
use_coupon = True

s = 0

if use_coupon:
    s = a * 0.95

if takeout:
    s = s * 1.08
else:
    s = s * 1.10

s
```

実行するたびに変化する値（変数）を入力値として定義。

共通する処理をそのまま移植。

```
def calc_amount(a, takeout, use_coupon):
    s = 0

    if use_coupon:
        s = a * 0.95

    if takeout:
        s = s * 1.08
    else:
        s = s * 1.10

    return s
```

関数


関数名
引数
関数の定義の開始
インデント
def calc_amount(a, takeout, use_coupon):
s = 0
if use_coupon:
s = a * 0.95
if takeout:
s = s * 1.08
else:
s = s * 1.10
関数の定義の終了
return s
関数ブロック
インデントの数で、関数のブロックが終了しているかどうかを判定している。

関数

関数を使うとき、その関数の名前を呼び出せばよい。その際に、関数に入力したい値を、定義した順番通りにカンマ区切りで並べて与える。関数での処理結果を変数に代入したい場合は、代入演算子を使用する。

```
def calc_amount(a, takeout, use_coupon):
    s = a
    if use_coupon:
        s = s * 0.95
    if takeout:
        s = s * 1.08
    else:
        s = s * 1.10
    return s


total_1 = calc_amount(120, True, True)
print(total_1)

total_2 = calc_amount(140, True, False)
print(total_2)

total_3 = calc_amount(160, False, True)
print(total_3)
```

問題 S3-1

リスト中の要素のうち、奇数の要素の個数を求める関数を作成せよ。

```
a = [1, 7, 4, 8, 3]

def getnodd(x):




getnodd(a)
# 3
```

リスト中の要素のうち、奇数の要素の和を求める関数を作成せよ。

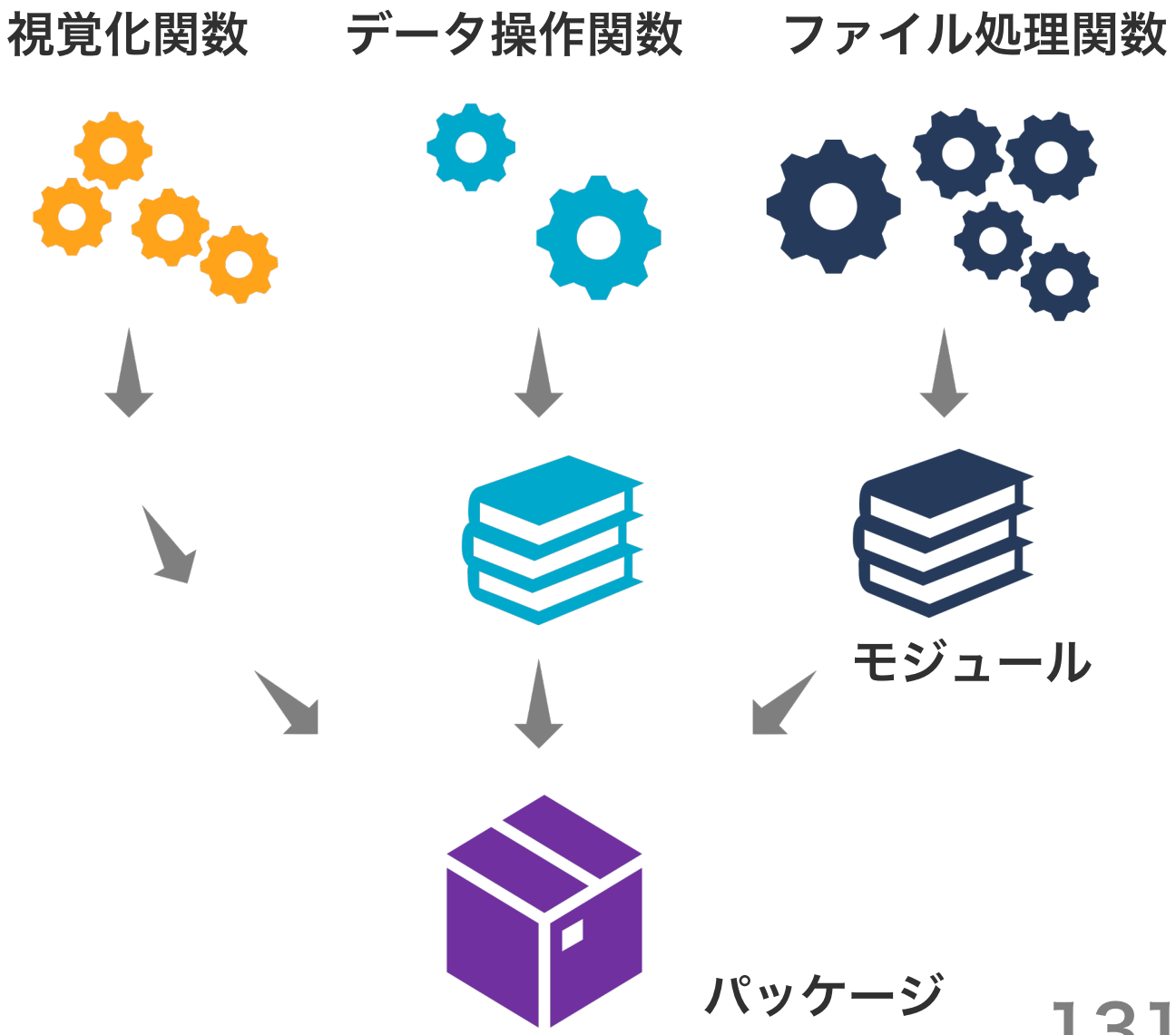
```
a = [1, 7, 4, 8, 3]

def sumodd(x):




sumodd(a)
# 11
```

基本文法

- ○ 予約語
- ○ 条件構文
- ○ 繰り返し構文
- ○ 関数
- ○ パッケージ

パッケージ

いくつかの機能の似た関数をまとめたものをライブラリー（モジュール）と呼ぶ。さらに複数のモジュールをまとめたものがパッケージと呼ぶ。



パッケージ

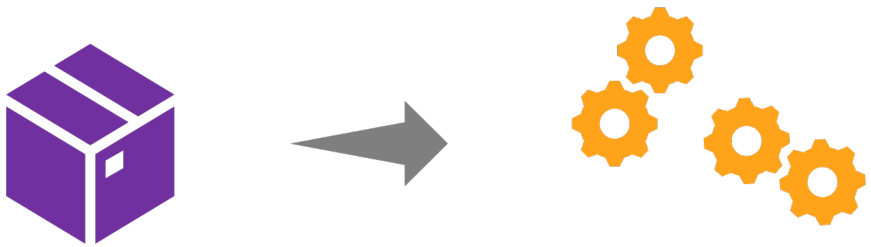
パッケージは、各開発者や開発団体の公式ウェブサイトあるいは GitHub や PyPI などのレポジトリで配布されている。例えば、PyPI には約 17 万のパッケージが登録されている。PyPI への登録は個人でも簡単に行えるので、研究目的の場合は、あまりマイナーなパッケージを使わないようにすること。

パッケージ

- NumPy
  - ベクトルの高速演算
  - 線形代数
- Pandas
  - データ操作
- SciPy
  - 統計処理
  - 微分方程式・積分
  - パラメーター最適化
- matplotlib
  - 可視化
- Seaborn
  - 可視化
- OpenCV
  - 画像解析
  - 動画解析
  - 機械学習
- scikit-image
  - 画像処理
  - 機械学習
- PIL / Pillow
  - 簡易画像処理
- scikit-learn
  - モデリング
  - 統計的機械学習
- tensorflow
  - 深層学習
- PyTorch
  - 深層学習

パッケージ

パッケージを使うとき、import を使ってパッケージの全機能を読み込むことをできる。例えば、データ処理用ライブラリーの NumPy の全機能を読み込むとき、"import numpy" のようにする。パッケージ中の関数（メソッド）を使うとき、パッケージ名の後に . をつけて、必要な関数を呼び出す。例えば、NumPy 中の array 関数を使いたい場合は、右のようにする。また、パッケージ名が長くて、読み込み後に略名を与えることができる。この際に、右のように as を使用する。略名を与えた以降、略名を用いて関数などの呼び出しが可能になる。



```
import numpy
a = numpy.array([1, 3, 5])
b = numpy.array([2, 4, 6])


import numpy as np
a = np.array([1, 3, 5])
b = np.array([2, 4, 6])


import matplotlib.pyplot as plt
fig = plt.figure()
```

テキスト処理

- テキスト処理
- ファイル処理

テキスト処理

- テキスト処理
- ファイル処理

テキストデータ

テキストデータはアルファベットや仮名などの文字からなるデータを指す。Python のオブジェクトには、数値の他にアルファベット・漢字・仮名などの文字を代入することもできる。文字をオブジェクトに代入するとき、その文字が、オブジェクトの名前ではなく、文字のデータであることを明示するために、文字データの両側を引用符で囲む。

```
s = 'a'


t = 'B'


u = 't'
u
# 't'


v = t
v
# 'B'
```

t が引用符で囲まれているため、文字データとしての t がオブジェクト u に代入される。

t が引用符で囲まれていないため、t はオブジェクトである。オブジェクト t の内容がオブジェクト v に代入される。

テキストデータ

単語や文章などのような文字列も、1 つのオブジェクトに代入できる。この場合、1 つのオブジェクトに、複数の文字データが保存されている、と捉えることができるため、このオブジェクトをリストのように扱うことができる。添字を指定して、特定の位置の文字を取り出したり、スライスして部分文字列を取り出したりすることができる。また、for 構文を使用して、文字列中の文字を 1 つずつ順に取り出すこともできる。

```
s = 'Smile. Tomorrow will be worse.'

s[0]
# 'S'



s[7:15]
# 'Tommorow'



for t in s:
    print(t)
# 'S'
# 'm'
# 'i'
# 'l'
# 'e'
# '.'
```

テキストデータ

文字列を保持しているオブジェクトに対して、足算が定義されている。そのため、+ 演算子を使用することで、複数の文字列を連結することができる。

```
s1 = 'Smile.'
s2 = 'Tomorrow will be worse.'
s3 = ' '

s = s1 + s2
s
# 'Smile.Tomorrow will be worse.'


s = s1 + s3 + s2
s
# 'Smile. Tomorrow will be worse.'


s = s1 + ' ' + s2
s
# 'Smile. Tomorrow will be worse.'
```

問題 T1-1

赤色で書かれたオブジェクトが保持している値を答えよ。

```
s1 = 'anything that can go wrong'
s2 = 'it will go wrong'



s1[:8]



s2[3:]



s3 = s1 + ',' + s2[3:]
s3
```

```
s1 = 'left to themselves'
s2 = 'things tend to go'
s3 = 'from bad to better'



s = s1 + ', ' + s2
s = s + ' ' + s3[:12] + 'worse.'
s
```

問題 T1-2

与えられた塩基配列の中から ATG を検索し、ATG が見つかればその位置を、そうでなければ -1 を出力するプログラムを for/while 構文や if 構文で作成せよ。

```
s = 'CCACAGTCATGTGTCAGTCGTAAGT'



# 8
```

```
s = 'CATTGTGTCACAGCCAGTCGTAAGT'



# -1
```

文字列操作関数

文字列の操作には、連結、分割、検索、置換などがある。これらの操作は、次の関数（文字列メソッド）で行える。

`a + b`	文字列 a の後ろに文字列 b を連結する。
`a.split(',')`	コンマ , を区切り文字として、文字列を分割して、部分文字列のリストに変換する。
`a.find('ATG')`	文字列 a の先頭から ATG を探す。見つかればその位置の添字を返し、そうでなければ -1 を返す。
`a.replace('TAG', '*')`	文字列 a 中ににある部分文字列 TAG を * に置き換える。

```
s = '21,31,41,51,61'


s.split(',')
# ['21', '31', '41', '51', '61']



s.find('41')
# 6


s.find('71')
# -1


s.replace(',', ';')
# '21;31;41;51;61'
```

文字と数値の交互変換

Python の世界で、文字と数値の扱い方が異なる。文字と文字の足し算では文字同士が連結され、数値と数値の足し算では数学的に和が計算される。文字と数値の足し算は定義されておらず、エラーが出る。

```
a = '12'
b = 21


a + b
# TypeError: can only concatenate str
(not "int") to str
```

文字列 a に、整数 b を連結できないことを示すエラー。

```
b + a
# TypeError: unsupported operand type(s)
for +: 'int' and 'str'
```

整数 b に文字列 a を足せないことを示すエラー。

文字から数値への変換

文字列を整数または小数に変換することができる。整数への変換は int 関数を利用し、小数への変換は float 関数を利用する。これらの関数は、すべての文字列を整数・小数に変換できるわけではない。変換できない場合は、エラーが起こる。

```
a1 = '12'
a2 = '12.345'
a3 = '1.23e6'

int(a1)
# 12

int(a2)
# ValueError: invalid literal for int()
with base 10: '12.345'

float(a1)
# 12.0

float(a2)
# 12.345

float(a3)
# 1230000.0
```

数値から文字への変換

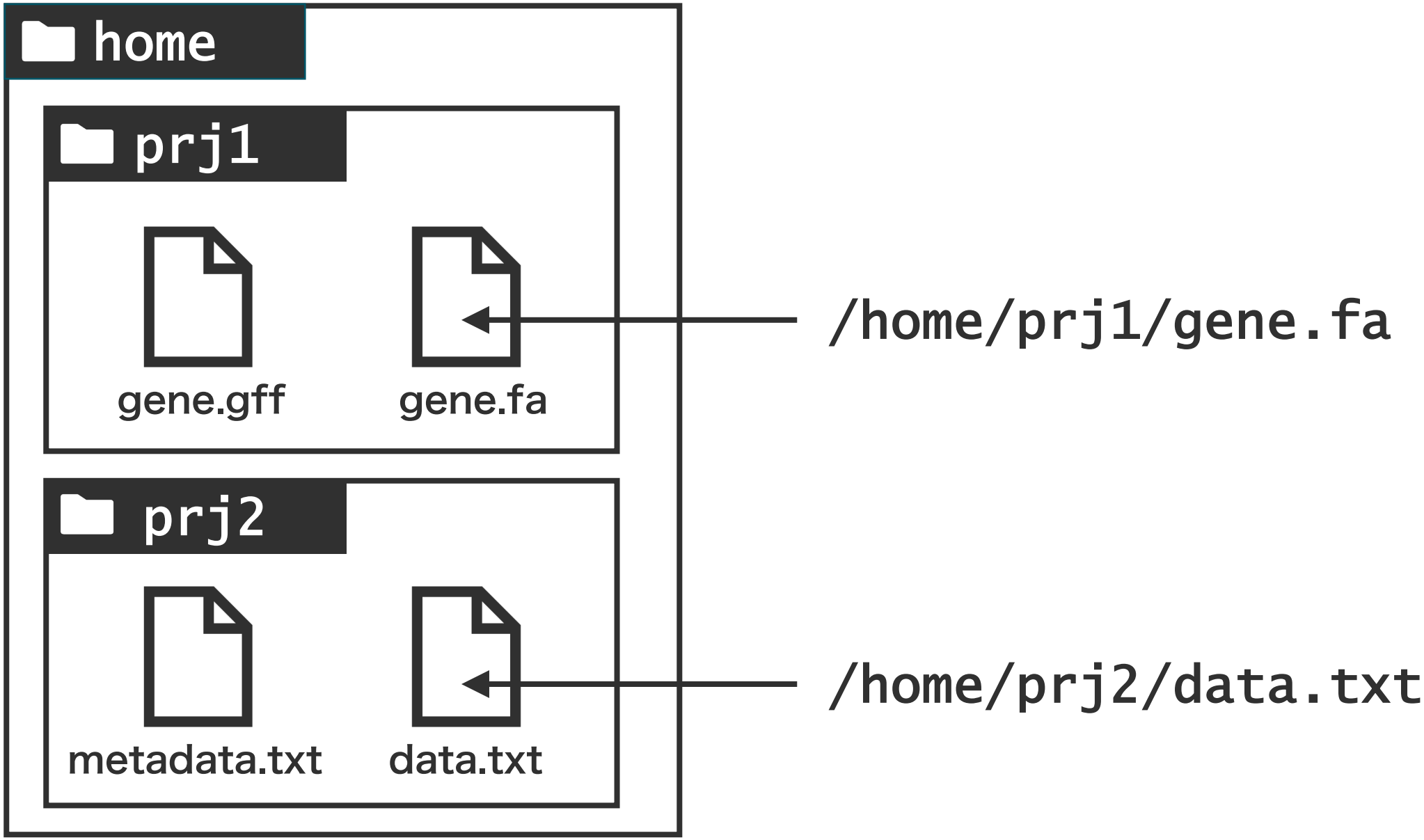
数値から文字への変換は str 関数を使用する。基本的に、すべての数値を文字に変換できる。桁数の多い小数を文字に変換するとき、桁落ちが発生する場合がある。

```
b1 = 12
b2 = 12.345
b3 = 12.34567890123456789 01
b4 = 1.23e6


str(b1)
# '12'


str(b2)
# '12.345'


str(b3)
# '12.345678901234567'


str(b4)
# '1230000.0'
```

テキスト処理

- テキスト処理
- ファイル処理

パス

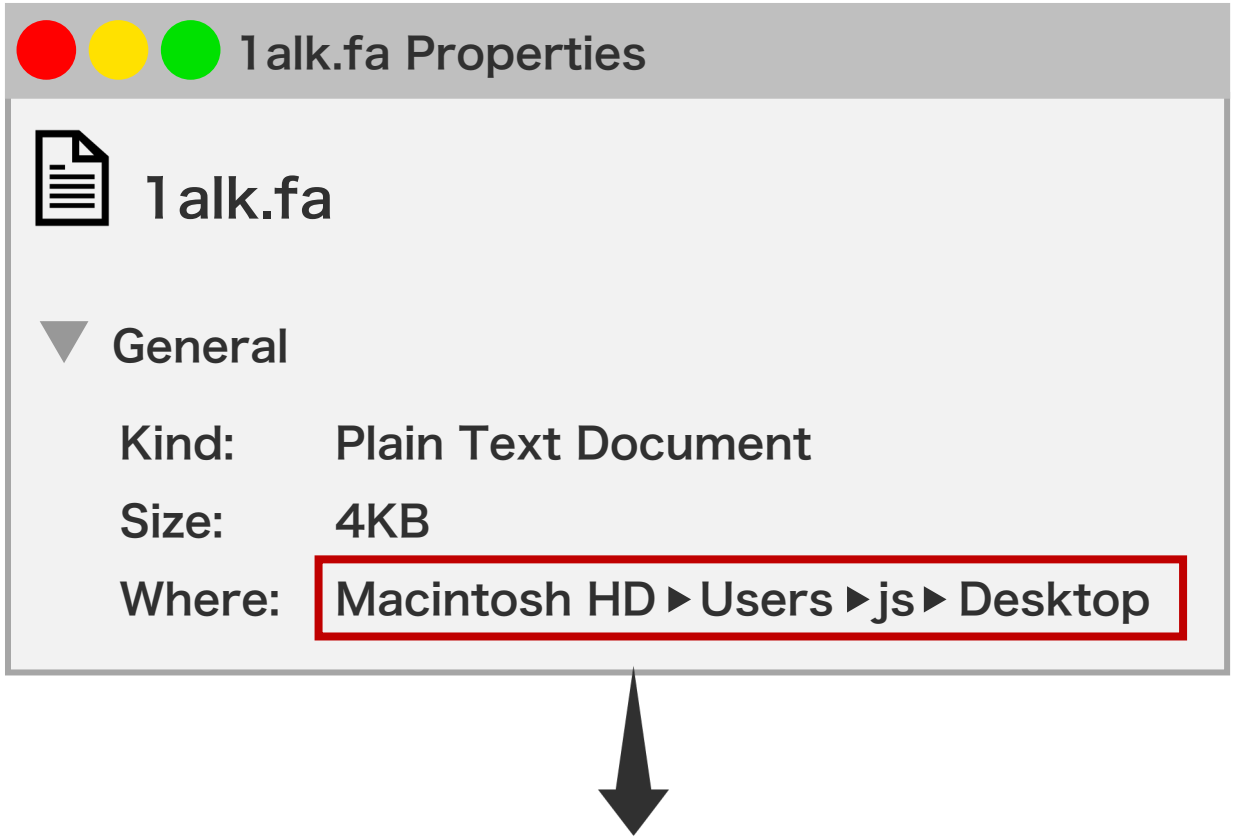

home
prj1
gene.gff
gene.fa
prj2
metadata.txt
data.txt
/home/prj1/gene.fa
/home/prj2/data.txt

ファイルパス（Macintosh）

1. パスを調べたいファイルを右クリックし、「Get Info」を選ぶ。

2. 「General」タブの「Where」項目にファイルへのパスが記載されている。

    - ファイルパスの「Macintosh HD」は最上層を表し、パスを書くとき「/」と書く。
    - 小さい三角形はフォルダの包含関係を表すので、パスを書くときは「/」と書く。



/Users/js/Desktop/1alk.fa

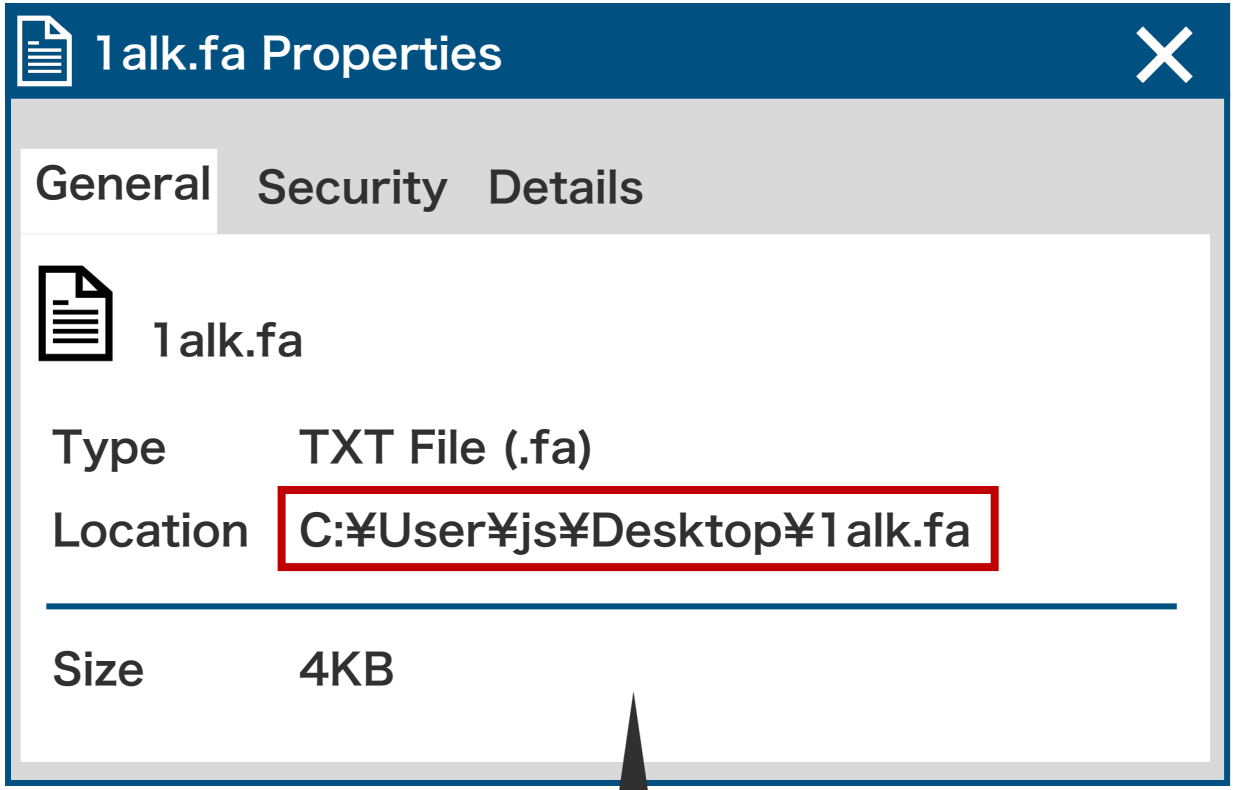
ファイルパス（Windows）

1. パスを調べたいファイルを右クリックし、「Properties」を選ぶ。

2. 「General」タブの「Location」項目にファイルへのパスが記載されている。

   - **日本語環境の場合、「＼」が「¥」として表示される。**どちらもコンピューター内部では 0x5C として認識されている。

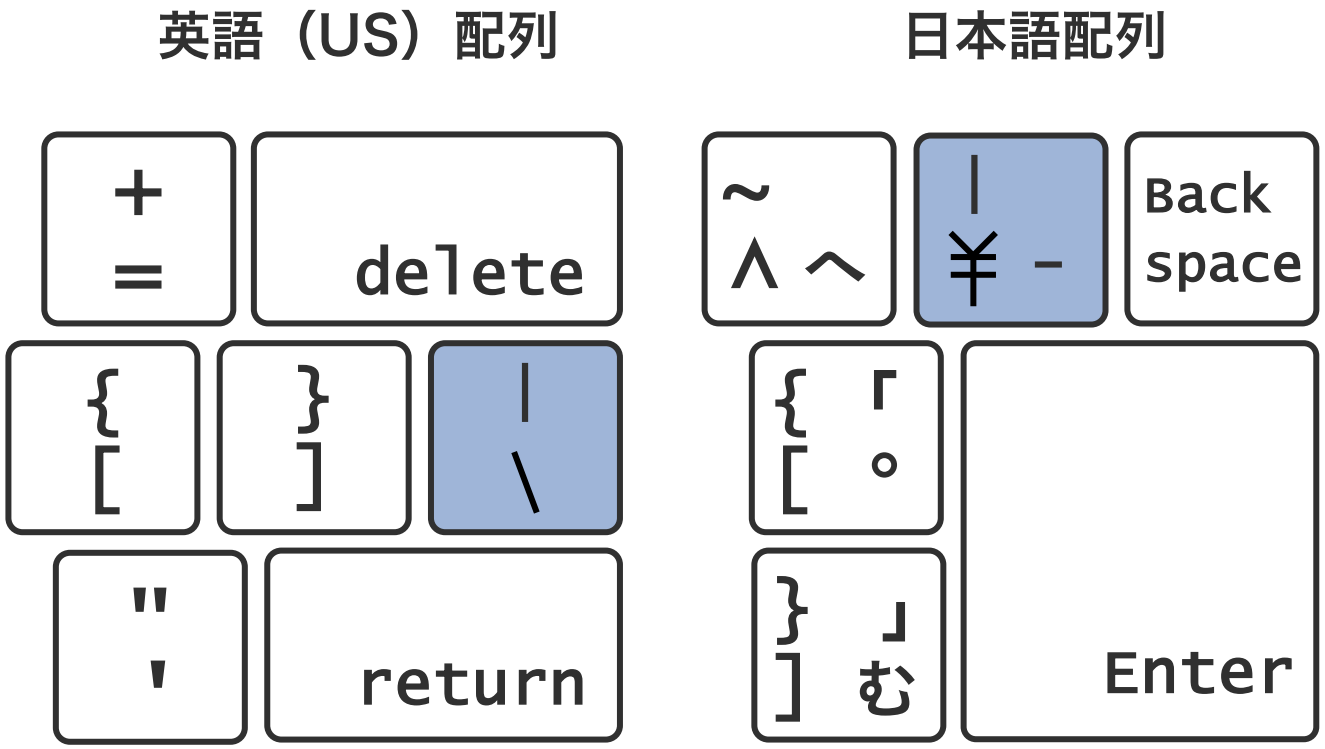
   - パスとして使用するとき、Location 欄に書かれているパス中の**「¥」を「/」に置き換える。**



C:/User/js/Desktop/1alk.fa

バックスラッシュ（＼）・円マーク（¥）

バックスラッシュは、OS の種類、言語環境や文字コードによって、ディスプレイでの表示が異なる。日本語環境のシステムであれば円マーク「¥」として表示され、それ以外の言語環境では「＼」として表示される。バックスラッシュと円マークは、表示が異なるものの、コンピューター上では同一のコード 0x5C として扱われる。また、バックスラッシュは特別な制御を行うための文字であり、一般的な文字として使用されることは少ない。



macOS 日本語環境を使用している場合は、＼ と ¥ の両方を入力できる。Option を押しながら＼または¥キーを押すことで、 ＼と¥の入力の切り替えができる。

ファイル

Python でファイルを読み込むには、open 関数を使用する。open 関数に、ファイルへのパスとともにオープン・モードを与えて使う。

モード	意味
r	読み込みモード。ファイルが存在しない場合はエラーになる。
w	書き込みモード。ファイルが存在しない場合は新規作成される。ファイルが存在する場合は、既存のファイルが削除されたうえで新規作成される。
a	追記モード。既存のファイルの最後に追記する。ファイルが存在しない場合は新規作成される。

https://aabbdd.jp/notes/data/1alk.fa

```
f = '1alk.fa'


with open(f, 'r') as fh:

    for line in fh:

        print(line)


# >1ALK:A|PDBID|CHAI...
# TPEMPVLENRAAQGNITA...
# TGQYTHYALNKKTGKPDY...
# AHVTSRKCYGPSATSQKC...
```

ファイル読み込み

パス f に保存されているファイルを、r モードで開き、ファイルハンドルにセットアップする。

```
f = '1alk.fa'


with open(f, 'r') as fh:

    for line in fh:

        print(line)


# >1ALK:A|PDBID|CHAI...
# TPEMPVLENRAAQGNITA...
# TGQYTHYALNKKTGKPDY...
# AHVTSRKCYGPSATSQKC...
```

https://aabbdd.jp/notes/data/1alk.fa

ファイル読み込み

ファイルの内容がリストに変換されてファイルハンドルに代入される※。ファイルの i 行目の文字列情報が、リストの i 番目の要素に代入されている。そのため、for 構文を利用して、リストの先頭から要素を順に取り出せば、ファイルの内容を 1 行目から順に取り出せるようになる。

このように見えるが、・・・

```
>1ALK
TPEMPV
TGQYTH
AHVTSR
```

```
fh = [
  '>1ALK',
  'TPEMPV',
  'TGQYTH',
  'AHVTSR'
]
```

▶

```
f = '1alk.fa'

with open(f, 'r') as fh:
    for line in fh:
        print(line)

# >1ALK:A|PDBID|CHAI...
# TPEMPVLENRAAQGNITA...
# TGQYTHYALNKKTGKPDY...
# AHVTSRKCYGPSATSQKC...
```

※ 正確には、ファイルの何行の何文字目まで読み込んだかの位置情報（ファイルポインター）がファイルハンドルに保存される。ファイルハンドルに対して for 構文を適用すると、ポインターを自動的に 1 行ずつ進めさせることができる。また、read メソッドと for 構文を組み合わせることで、ポインターを 1 文字ずつ進めさせることもできる。

ファイル読み込み

ファイルの内容がリストに変換されてファイルハンドルに代入される※。ファイルの i 行目の文字列情報が、リストの i 番目の要素に代入されている。そのため、for 構文を利用して、リストの先頭から要素を順に取り出せば、ファイルの内容を 1 行目から順に取り出せるようになる。

```
f = '1alk.fa'

with open(f, 'r') as fh:
    for line in fh:
        print(line)

# >1ALK:A|PDBID|CHAI...
# TPEMPVLENRAAQGNITA...
# TGQYTHYALNKKTGKPDY...
# AHVTSRKCYGPSATSQKC...
```

このように見えるが、実際は特殊文字が存在している！

```
>1ALK\n
TPEMPV\n
TGQYTH\n
AHVTSR\n
```

特殊文字

```
fh = [
    '>1ALK\n',
    'TPEMPV\n',
    'TGQYTH\n',
    'AHVTSR\n'
]
```

特殊文字

※ 正確には、ファイルの何行の何文字目まで読み込んだかの位置情報（ファイルポインター）がファイルハンドルに保存される。ファイルハンドルに対して for 構文を適用すると、ポインターを自動的に 1 行ずつ進めさせることができる。また、read メソッドと for 構文を組み合わせることで、ポインターを 1 文字ずつ進めさせることもできる。

ファイル読み込み

```
f = '1alk.fa'


with open(f, 'r') as fh:

    for line in fh:

        print(line)


# >1ALK:A|PDBID|CHAI...
# TPEMPVLENRAAQGNITA...
# TGQYTHYALNKKTGKPDY...
# AHVTSRKCYGPSATSQKC...
```

すべての行の読み込みが終わり、ファイルが閉じられる。 ▶

エスケープシーケンス

Python では、特殊な用途向けに、いくつかの特殊文字が定義されている。例えば、改行とタブがその特殊文字にあたる。改行文字は、人には見えないが、コンピュータが改行として認識するために必要な特殊文字である。限られたアルファベットと数字の中で、このような特殊文字を表すためには、既存文字を組み合わせて表現する。一般に、バックスラッシュに 1 文字を足して、特殊文字を表現していることが多い。例えば、改行ならば \n、タブならば \t のように表現する。

※ ＼ と ￥ は同じ文字コードであることに注意！

```
s = 'Smile. nTomorrow will be worse.'
s
# 'Smile. nTomorrow will be worse.'



s = 'Smile. \nTomorrow will be worse.'
s
# 'Smile.'
# 'Tomorrow will be worse.'



s = 'Smile. \\nTomorrow will be worse.'
s
# 'Smile. \nTomorrow will be worse.'
```

問題 F1-1

リスト fh の要素のうち、「>」から始まる要素の個数を求めよ。

```
fh = ['>1ALK:A',
      'TPEMPVL',
      'TGQYTHA',
      '>1ALK:B',
      'TPEMPVL',
      'TGQYTHA']
```

問題 F1-2

1alk.fa ファイルを読み込んで、「>」から始まる行の行数を求めよ。

```
f = '1alk.fa'
n = 0
```

https://aabbdd.jp/notes/data/1alk.fa

問題 F1-3

ft.fa に記載された塩基配列について、A の出現確率を求めよ。

```
f = 'ft.fa'
pA = 0
```

https://aabbdd.jp/notes/data/ft.fa

問題 F1-4

diversity_galapagos.txt は、タブ区切りのテキストファイルであり、ガラパゴス島における種の多様性データが記載されている。このファイルを読み込み、面積（Area 列）の最も大きい島の名前（Island 列）を答えよ。ただし、このテキストファイルには、「#」から始まるコメント行とデータの属性を示すヘッダー行が含まれていることに注意。

```
f = 'diversity_galapagos.fa'
max_area_island_name = 0
```

https://aabbdd.jp/notes/data/diversity_galapagos.txt

ファイル書き込み

ファイルの書き込みには w および a モードが定義されている。このほかにも w+ や a+ などのモードも定義されているが、初めは w モードのみ使えればよい。

モード	意味
r	読み込みモード。ファイルが存在しない場合はエラーになる。
w	書き込みモード。ファイルが存在しない場合は新規作成される。ファイルが存在する場合は、既存のファイルが削除されたうえで新規作成される。
a	追記モード。既存のファイルの最後に追記する。ファイルが存在しない場合は新規作成される。

```
f = 'output.txt'


with open(f, 'w') as outfh:

    outfh.write('abcdefg')
    outfh.write('1234567')
```

ファイル書き込み

パス f を書き込みモードで開き、ファイルハンドル ▶ にセットアップする。

```
f = 'output.txt'


with open(f, 'w') as outfh:

    outfh.write('abcdefg')
    outfh.write('1234567')
```

ファイル書き込み

abcdefg をファイルに書き込む ▶

```
f = 'output.txt'


with open(f, 'w') as outfh:

    outfh.write('abcdefg')
    outfh.write('1234567')
```

ファイル書き込み

1234567 をファイルに書き込む ▶

```
f = 'output.txt'


with open(f, 'w') as outfh:

    outfh.write('abcdefg')
    outfh.write('1234567')
```

ファイル書き込み

書き込みが終了し、ファイルが閉じられる。 ▶

```
f = 'output.txt'


with open(f, 'w') as outfh:

    outfh.write('abcdefg')
    outfh.write('1234567')
```

output.txt

```
abcdefg1234567
```

ファイル書き込み

```
f = 'output.txt'


with open(f, 'w') as outfh:

    outfh.write('abcdefg\n')
    outfh.write('1234567\n')
```

書き込みが終了し、ファイルが閉じられる。 ▶

output.txt

```
abcdefg
1234567
```

NumPy

- １ 次元配列
- ２ 次元配列
- ３ 次元配列
- データの読み書き

python™

pandas

NumPy

アプリ開発やシステム管理など様々な用途で利用できるような汎用機能を提供する。

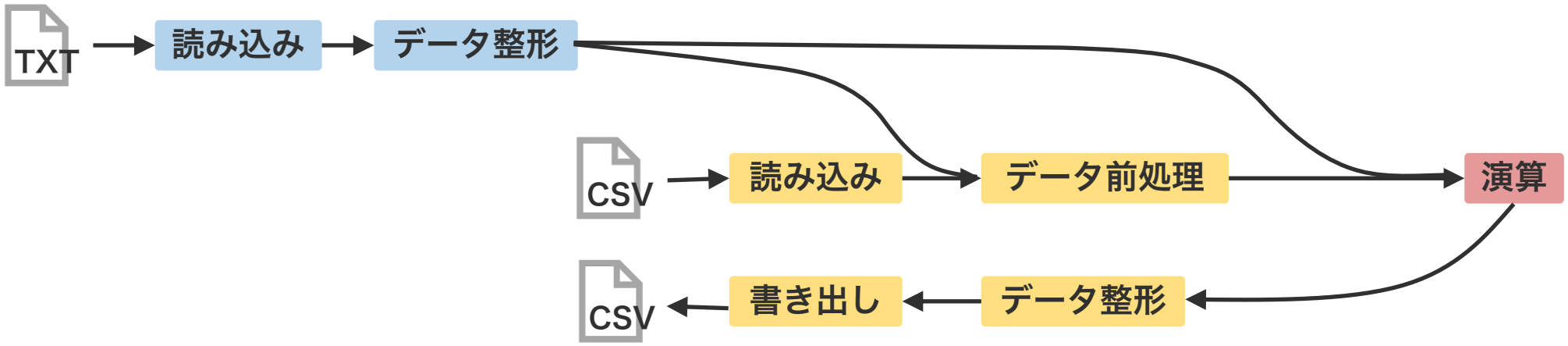
- リスト
- 2 次元リスト

CSV ファイルの読み書きや表データの操作や整形などに特化した機能を提供する。

- シリーズ
- データフレーム

整形されたデータに対して、情報量をできるだけ落とさずに、高速に演算を行う機能を提供する。

- 配列
- 2 次元配列



NumPy

- １ 次元配列
- ２ 次元配列
- ３ 次元配列
- データの読み書き

1 次元配列 / np.array

NumPy の 1 次元配列は、Python の 1 次元リストを拡張したようなオブジェクトである。NumPy 配列は、NumPy で用意された関数を使用して作成する。例えば、np.array 関数を使えば、任意の Python のリストを配列に変換できる。また、np.full や np.linspace などの関数を使えば、すべての要素が同じ値であるような配列や等差数列のような配列を作ることもできる。

- np.array
- np.zeros
- np.ones
- np.full
- np.arange
- np.linspace

```
import numpy as np


a = [1, 1, 2, 3, 5, 8]
a
# [1, 1, 2, 3, 5, 8]


b = np.array(a)
b
# array([1, 1, 2, 3, 5, 8])


c = np.array([1, 1, 2, 3, 5, 8])
c
# array([1, 1, 2, 3, 5, 8])
```

1 次元配列 / np.full

すべての要素が同じ値であるような配列を作るとき、np.zeros, np.ones, np.full 関数を使用すると便利である。np.zeros と np.ones 関数は、関数名の通り、すべての要素が 0 または 1 であるような配列を作るときに使用する。また、また、np.full 関数は、任意の値で初期化された配列を作るときに使用する。

np.array

np.zeros　np.ones　np.full

np.arange

np.linspace

```
import numpy as np


a = np.zeros(5)
a
# array([0, 0, 0, 0, 0])


b = np.ones(8)
b
# array([1, 1, 1, 1, 1, 1, 1, 1])


c = np.full(5, np.nan)
c
# array([nan, nan, nan, nan, nan])
```

1 次元配列 / np.arange

等差数列からなる配列を作るとき、np.arange 関数を使用する。np.arange 関数は、start (数列の最初の値), stop (数列の範囲), step (間隔) の 3 つの引数を受け取り、それらに基づいて等差数列からなる配列を作る。start を省略すると start=0 となり、step を省略すると step=1 となる。

```
import numpy as np

a = np.arange(5)
a
# array([0, 1, 2, 3, 4])


b = np.arange(1, 6)
b
# array([1, 2, 3, 4, 5])


c = np.arange(1, 6, 2)
c
# array([1, 3, 5])


d = np.arange(10, 0, -2)
d
# array([10,  8,  6,  4,  2])
```

np.array

np.zeros　np.ones　np.full

np.arange

np.linspace

関数名に注意。np.arrange ではない。NumPy 前身であった Numeric 時代で使われた arrayrange 関数の省略形として arange が使われいたことに由来する。

1 次元配列 / np.linspace

等差数列からなる配列を作るとき、np.linspace 関数も利用できる。この関数は、start (数列の最初の値), stop (数列の最後の値), num (要素数) の 3 つの引数を受け取り、それらに基づいて等差数列からなる配列を作る。num を省略すると num=50 となる。

np.array

np.zeros　np.ones　np.full

np.arange

np.linspace

⚠ 関数名に注意。np.linespace ではない。Linearly space vectors を生成する関数のため、linspace である。

```
import numpy as np


a = np.linspace(1, 9, 3)
a
# array([1., 5., 9.])


b = np.linspace(1, 9, 5)
b
# array([1., 3., 5., 7., 9.])


c = np.linspace(1, 0, 5)
c
# array([1.  , 0.75, 0.5 , 0.25, 0.  ])
```

1 次元配列 / 数値計算

NumPy には、切り上げや切り捨てを行う np.ceil や np.floor、対数化を行う np.log、三角関数の値を計算する np.sin, np.cos や np.tan など、平均や分散を計算する np.mean や np.std など、多くの関数が用意されている。これらの関数名をすべて覚える必要はなく、使いたいときにその使い方を調べればよい。

```
import numpy as np

a = np.array([1, 5, 10, 50, 100])


np.log10(a)
# array([0., 0.69897, 1., 1.69897, 2.])


np.sqrt(a)
# array([1.,2.2360,3.1622,7.0710,10.])


np.mean(a)
# 33.2


np.std(a)
# 37.722142038860945
```

https://docs.scipy.org/doc/numpy/reference/ufuncs.html

1 次元配列

NumPy の 1 次元配列は、リストほぼ同じように取り扱うことができる。配列は基本的に全要素を束ねて使うことが多いが、一部の要素だけを取り出したり、変更したりすることもできる。その際、配列の先頭から数えて何番目の要素を取り出したいのかを、整数で指定する必要がある。

```
import numpy as np

w = np.array([12, 10, 11, 13, 11])

w[0]
# 12

w[1]
# 10

w[2]
# 11

w[2] = 9

w
# array([12, 10, 9, 13, 11])
```

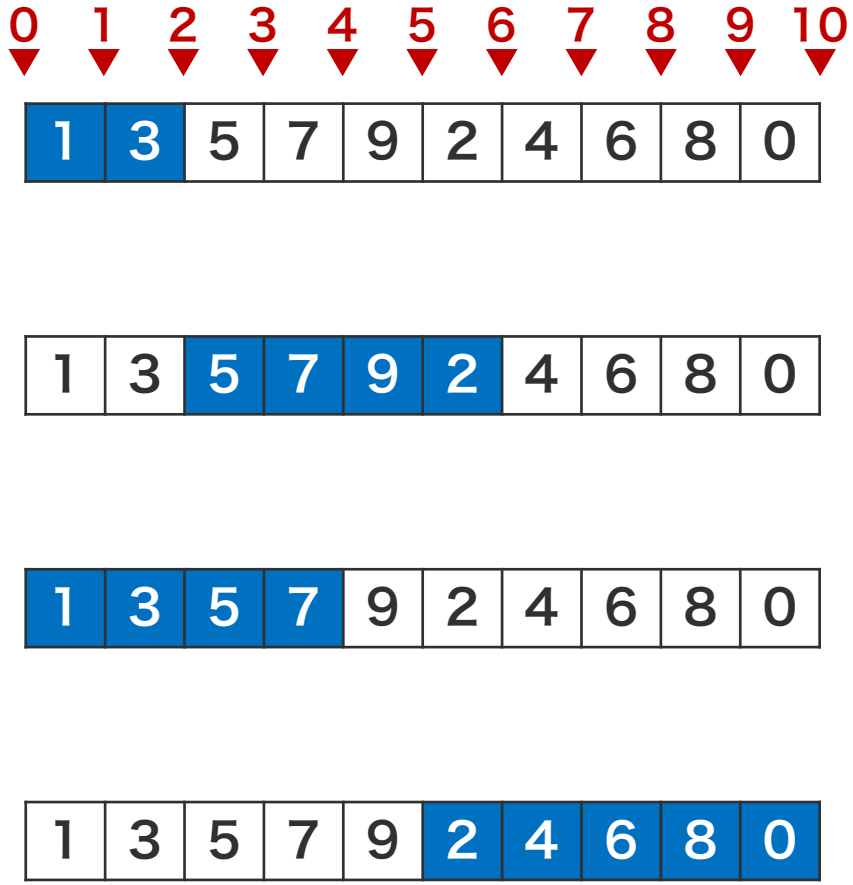
1 次元配列 / スライス

Python のリストと同様に、隣り合う要素をスライスして取得することもできる。スライスを行うとき、スライスの開始位置と終了（手前の）位置をコロン（:）で区切って指定する。

```
a = np.array([1, 3, 5, 7, 9,
              2, 4, 6, 8, 0])



b = a[3:6]
b
# np.array([7, 9, 2])

b[0] = 0
b[1] = 0
b[2] = 0

b
# np.array([0, 0, 0])



a
# np.array([1, 3, 5, 0, 0,
#           0, 4, 6, 8, 0])
```

スライスにより取り出された部分配列は、元の配列への参照となっている。したがって、スライスで得られた部分配列の値を変更すると、元の配列の値も変化してしまう。

1 次元配列 / スライス




```
a = np.array([1, 3, 5, 7, 9,
              2, 4, 6, 8, 0])
```

```
a[0:2]
# np.array([1, 3])
```

```
a[2:6]
# np.array([5, 7, 9, 2])
```

```
a[:4]
# np.array([1, 3, 5, 7])
```

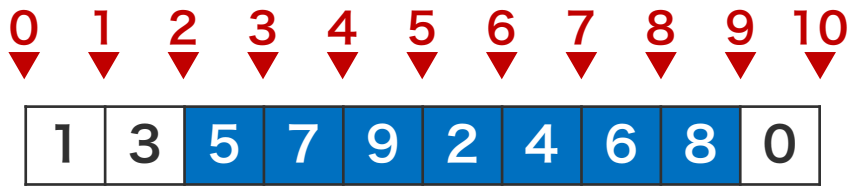
```
a[5:]
# np.array([2, 4, 6, 8, 0])
```

1 次元配列 / スライス

スライスを行うとき、開始位置と終了位置の他に、ステップ数を与えることもできる。1 要素おきに値を 1 つ取り出す場合などに便利である。また、配列では、連続していない要素を同時に取り出すこともできる。

```
import numpy as np

a = np.array([1, 3, 5, 7, 9,
              2, 4, 6, 8, 0])


a[2:9]



a[2:9:1]



a[2:9:3]



a[[0, 2, 4, 6]]
```

1 次元配列 / スライス




```
import numpy as np

a = np.array([1, 3, 5, 7, 9,
              2, 4, 6, 8, 0])


a[2:9]
# array([5, 7, 9, 2, 4, 6, 8])
```

```
a[2:9:1]
# array([5, 7, 9, 2, 4, 6, 8])
```

```
a[2:9:3]
# array([5, 2, 8])
```

```
a[[0, 2, 4, 6]]
# array([1, 5, 9, 4])
```

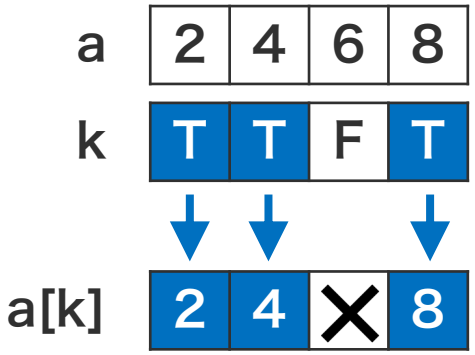
1 次元配列 / フィルター

配列から要素を取り出すとき、位置番号で指定するほか、True または False からなる配列（ブーリアンベクトル）で指定することもできる。このとき、ブーリアンベクトルの長さは、操作対象となる配列の長さと同じでなければならない。

```
import numpy as np

a = np.array([   2,    4,     6,    8])

k = np.array([True, True, False, True])

a[k]



a = np.array([   2,    4,     6,    8])

k = np.array([False, True, True, True])

a[k]
```

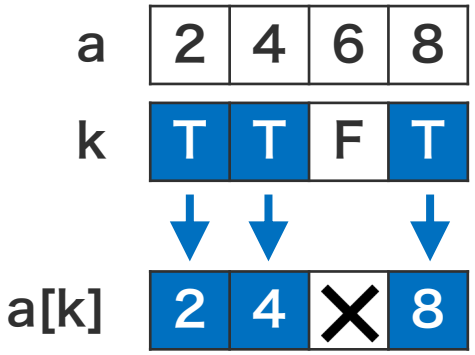
1 次元配列 / フィルター

フィルター k を用いて、ベクトル a の要素をフィルタリングしているイメージ。

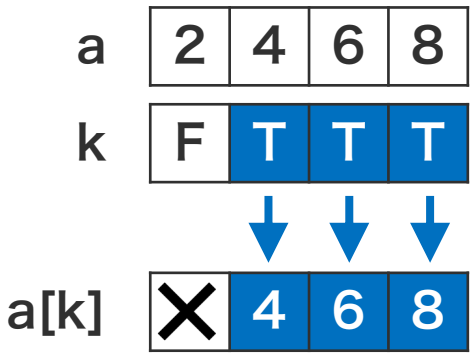



```
import numpy as np


a = np.array([   2,    4,     6,    8])

k = np.array([True, True, False, True])


a[k]
# array([2, 4, 8])



a = np.array([   2,    4,     6,    8])

k = np.array([False, True, True, True])


a[k]
```

1 次元配列 / フィルター




```
import numpy as np


a = np.array([   2,    4,     6,    8])

k = np.array([True, True, False, True])


a[k]
# array([2, 4, 8])
```




```
a = np.array([   2,    4,     6,    8])

k = np.array([False, True, True, True])


a[k]
# array([4, 6, 8])
```

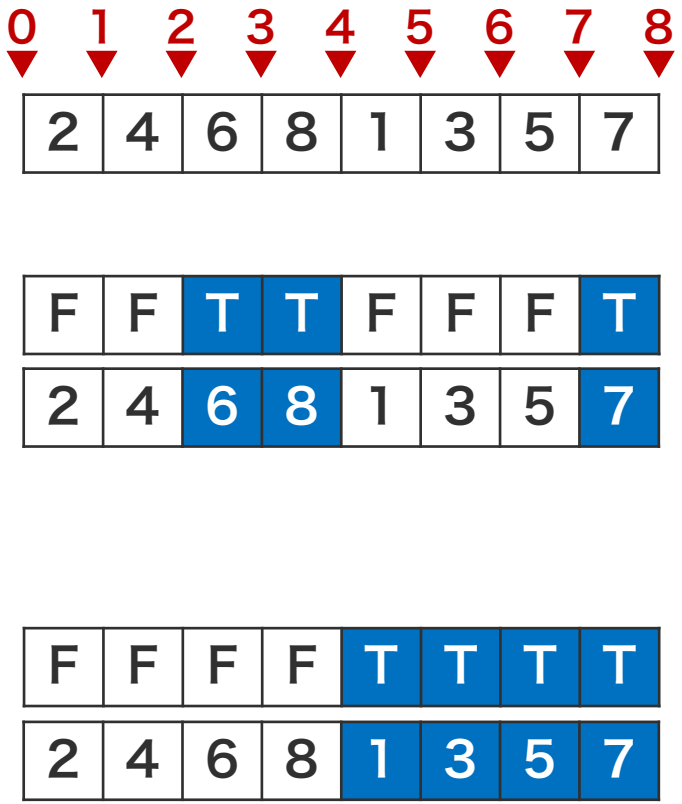
フィルター k を用いて、ベクトル a の要素をフィルタリングしているイメージ。

1 次元配列 / フィルター

フィルターは、True または False を組み合わせて直打ちで作成できるが、ある条件を制定して、その条件に基づいて作成することが一般的である。例えば、5 よりも大きい値を取り出すフィルターや奇数の値を取り出すフィルターは、右のように作成する。

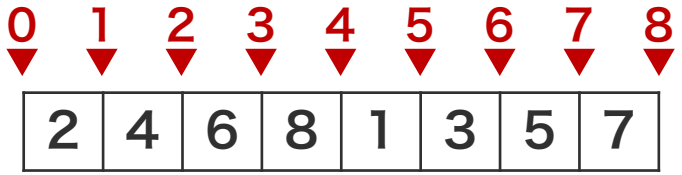
```
import numpy as np


a = np.array([2, 4, 6, 8, 1, 3, 5, 7])


f1 = (a > 5)
f1
# array([F, F, T, T, F, F, F, T])



f2 = (a % 2 == 1)
f2
# array([F, F, F, F, T, T, T, T])
```

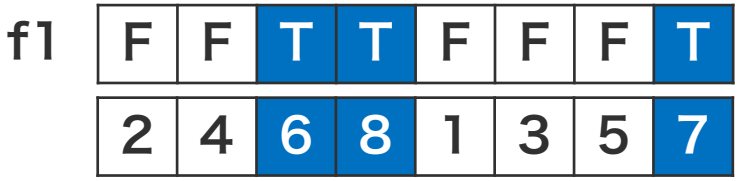
1 次元配列 / フィルター

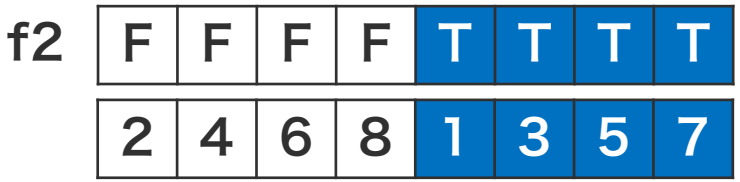


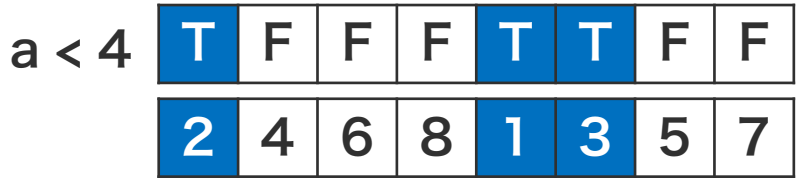

```
import numpy as np


a = np.array([2, 4, 6, 8, 1, 3, 5, 7])


f1 = (a > 5)
a[f1]
# array([6, 8, 7])



f2 = (a % 2 == 1)
a[f2]
# array([1, 3, 5, 7])
```

1 次元配列 / フィルター










```
import numpy as np


a = np.array([2, 4, 6, 8, 1, 3, 5, 7])


f1 = (a > 5)
a[f1]
# array([6, 8, 7])



f2 = (a % 2 == 1)
a[f2]
# array([1, 3, 5, 7])


a[(a < 4)]
# array([2, 1, 3])
```

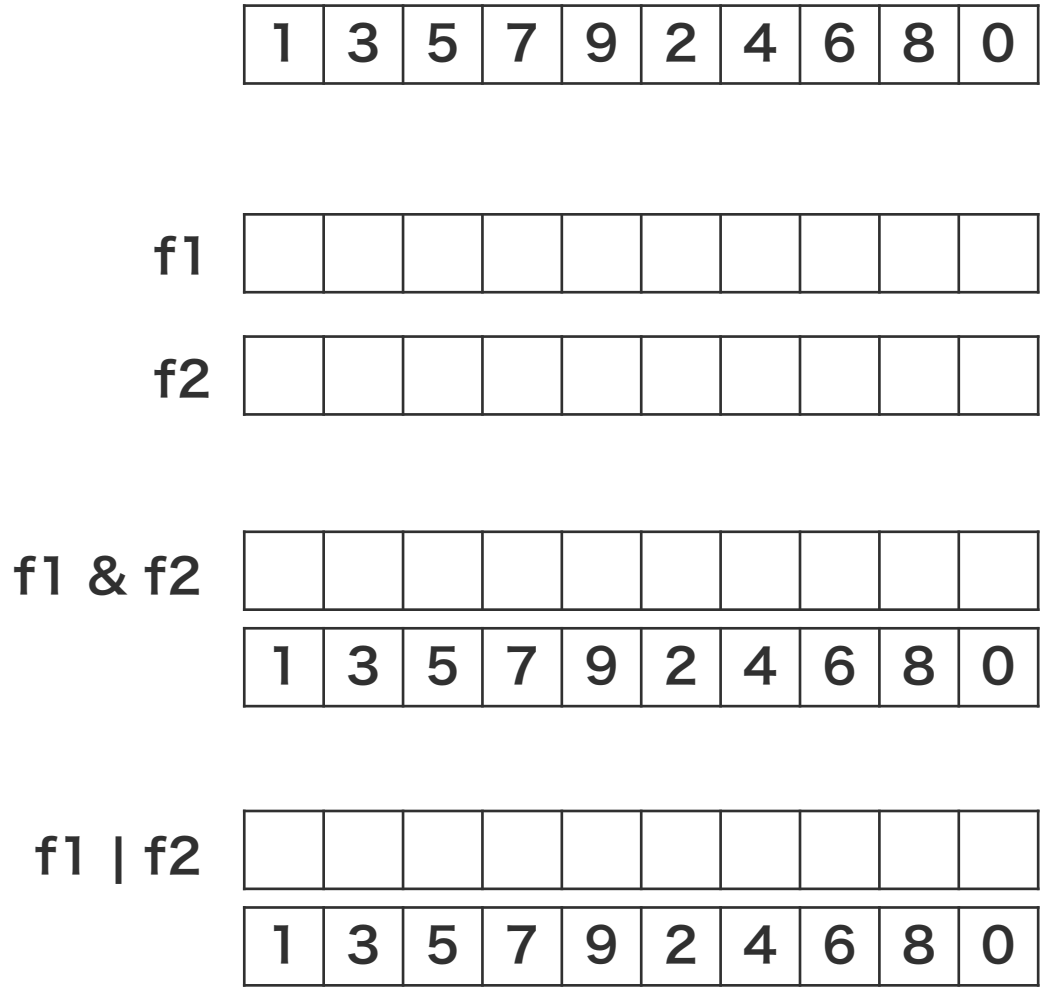
フィルターを一次変数に保存せずに、直接使用することもできる。

1 次元配列 / フィルター

複数のフィルターを重ねて使用することもできる。この場合、フィルターを重ねるときに AND 演算で重ねるか、OR 演算で重ねるかを指定する必要がある。

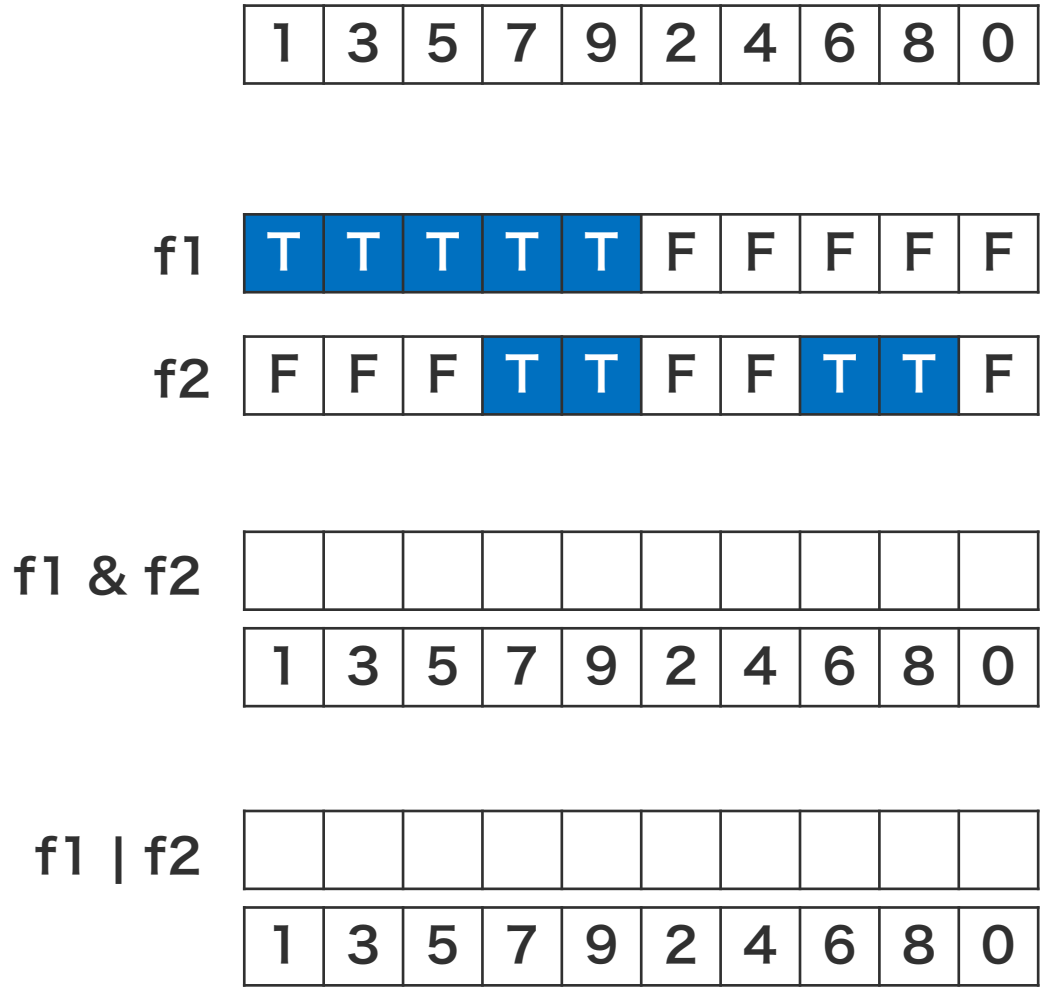
```
import numpy as np

a = np.array([1, 3, 5, 7, 9,
              2, 4, 6, 8, 0])


f1 = (a % 2 == 1)

f2 = (a > 5)




a[f1 & f2]




a[f1 | f2]
```

1 次元配列 / フィルター



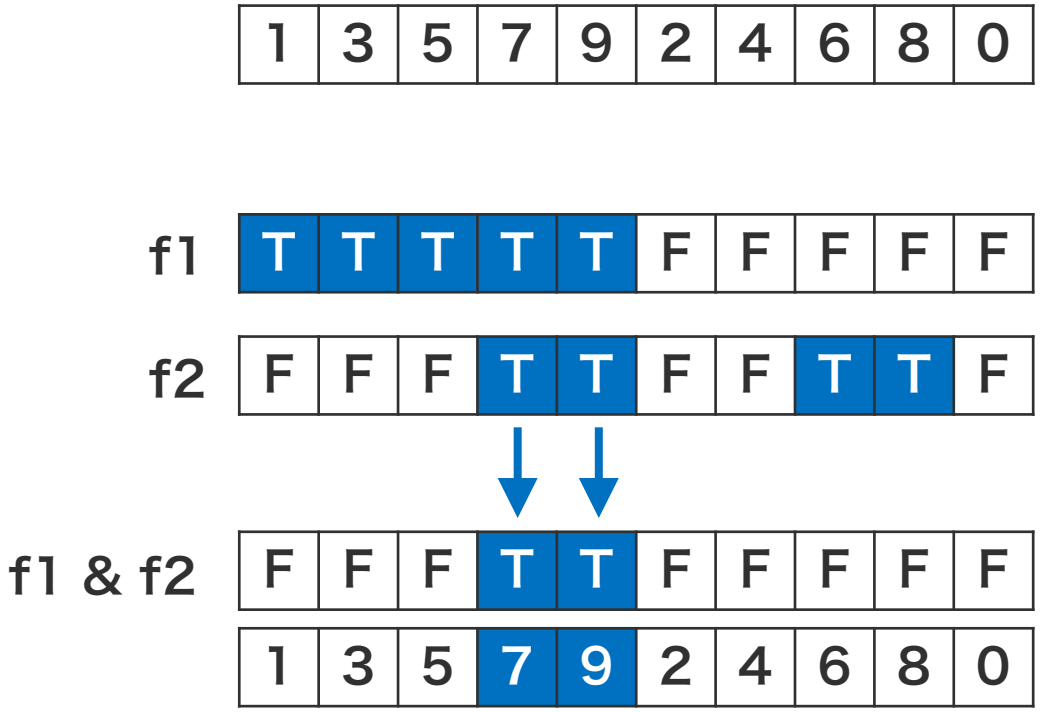

```
import numpy as np

a = np.array([1, 3, 5, 7, 9,
              2, 4, 6, 8, 0])


f1 = (a % 2 == 1)

f2 = (a > 5)




a[f1 & f2]




a[f1 | f2]
```

1 次元配列 / フィルター



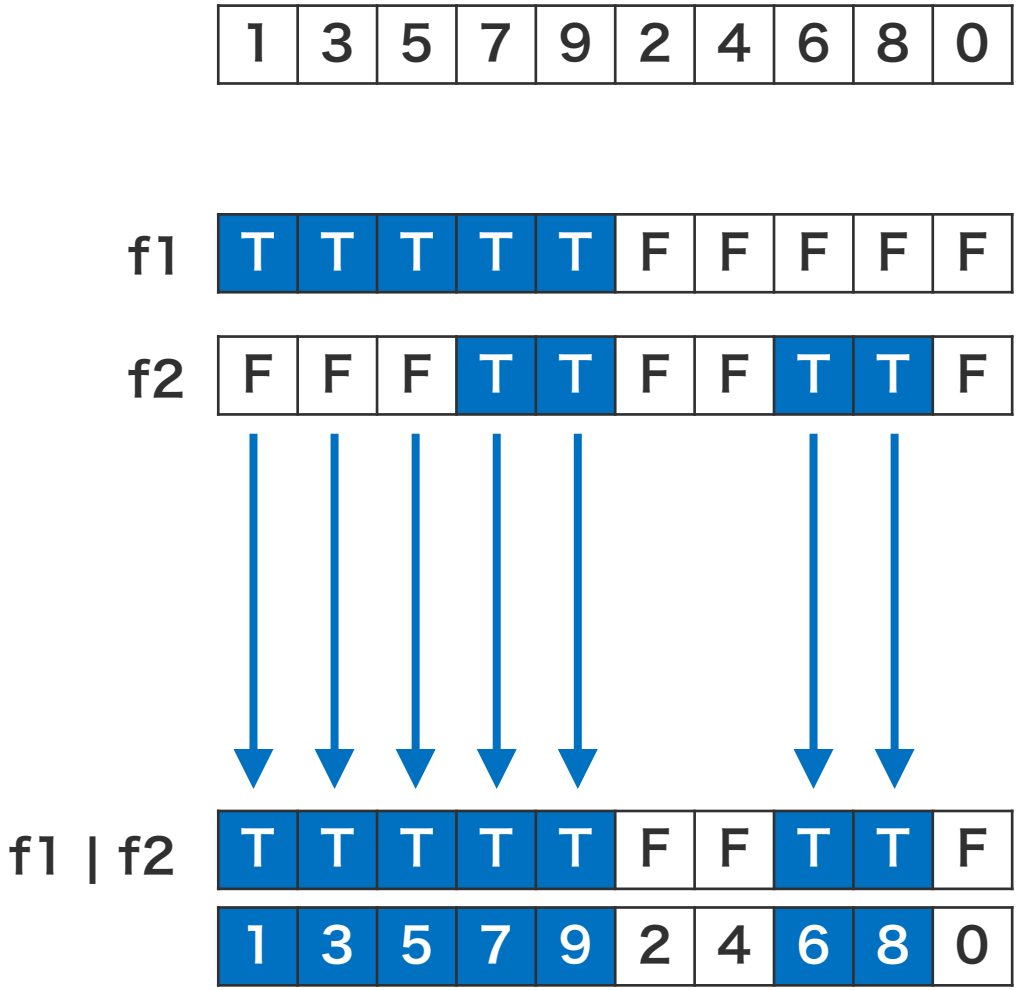

```
import numpy as np

a = np.array([1, 3, 5, 7, 9,
              2, 4, 6, 8, 0])


f1 = (a % 2 == 1)

f2 = (a > 5)




a[f1 & f2]




a[f1 | f2]
```

1 次元配列 / フィルター



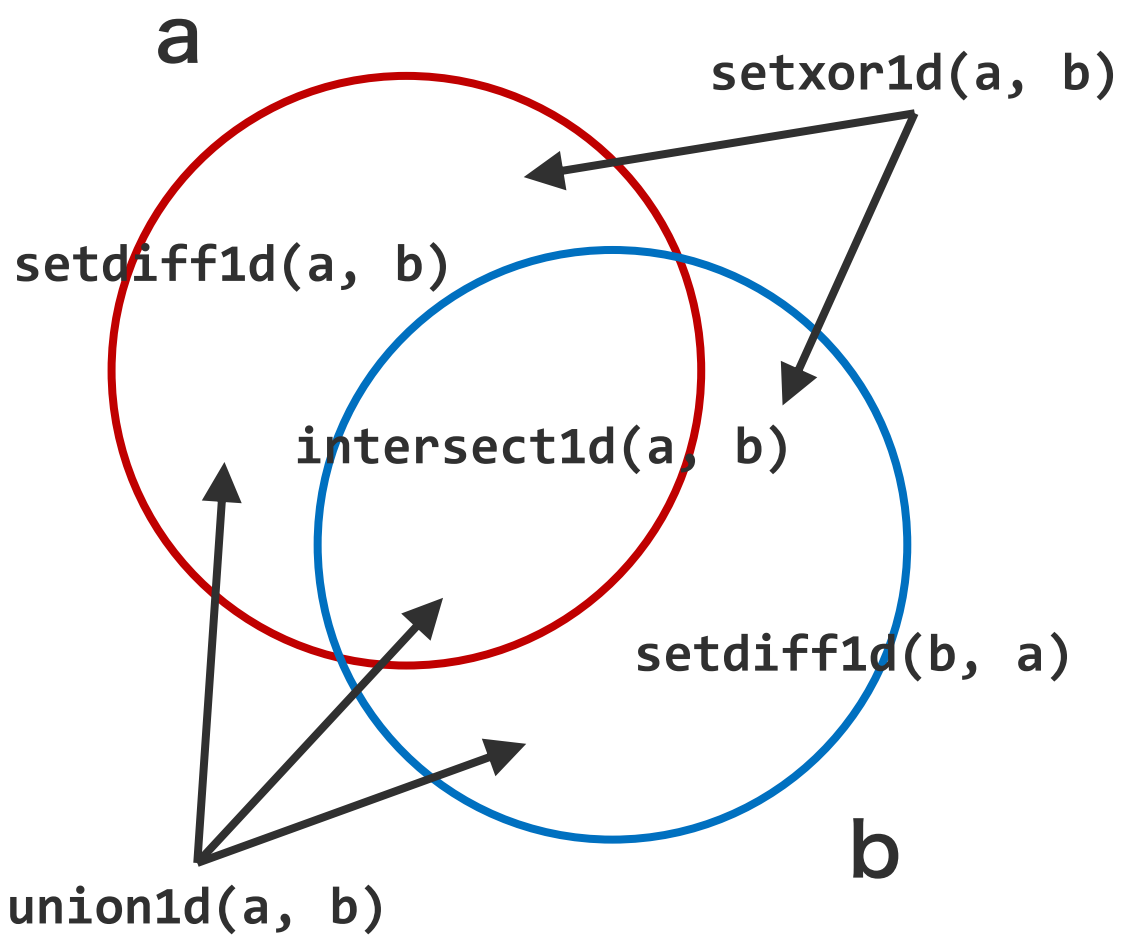

```
import numpy as np

a = np.array([1, 3, 5, 7, 9,
              2, 4, 6, 8, 0])


f1 = (a % 2 == 1)

f2 = (a > 5)




a[f1 & f2]
# array([7, 9])



a[f1 | f2]
```

1 次元配列 / フィルター




```
import numpy as np

a = np.array([1, 3, 5, 7, 9,
              2, 4, 6, 8, 0])


f1 = (a % 2 == 1)

f2 = (a > 5)




a[f1 & f2]
# array([7, 9])



a[f1 | f2]
# array([1, 3, 5, 7, 9, 6, 8])
```

集合演算




```
import numpy as np

a = np.array([1, 1, 2, 3, 5, 8])

b = np.array([2, 4, 6, 8])


np.union1d(a, b)
# array([1, 2, 3, 4, 5, 6, 8])


np.setdiff1d(a, b)
# array([1, 3, 5])


np.setdiff1d(b, a)
# array([4, 6])
```

擬似乱数生成

NumPy の random モジュールには擬似乱数を生成する機能が実装されている。次の表に、一様分布および正規分布から乱数を生成する関数を示した。これ以外にも、ポアソン分布やガンマ分布などの分布から乱数を生成する関数が多数用意されている。必要なときに調べて使うとよい。

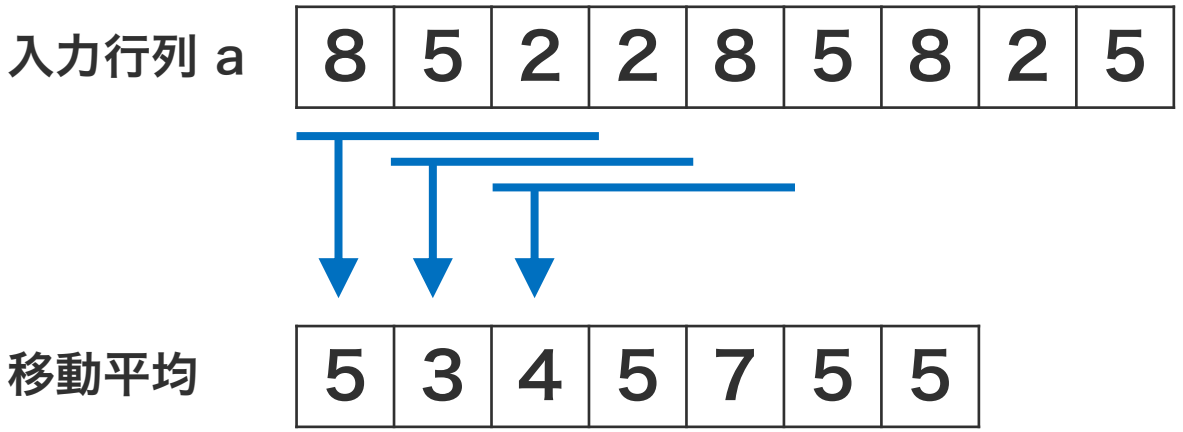
メソッド	動作
`.rand(n)`	範囲 [0, 1) の一様分布から n 個の乱数を生成。
`.normal(m, s, n)`	平均 m, 標準偏差 s の正規分布から n 個の乱数を生成。
`.randint(l, h, n)`	範囲 [l, h) から n 個の整数乱数を生成。
`.shuffle(arr)`	配列 arr の要素をシャッフルする。
`.seed(s)`	乱数シードを s に固定する。

```
import numpy as np

np.random.seed(2020)

np.random.rand(3)
# array([0.986276, 0.873391, 0.509745])

np.random.normal(5, 1, 3)
# array([4.115677, 3.762758, 4.291560])

np.random.randint(0, 2, 5)
# array([0, 1, 1, 1, 1])

a = np.array([1, 2, 3, 4, 5, 6, 7])
np.random.shuffle(a)
a
# array([2, 7, 6, 5, 1, 4, 3])
```

seed を削除すると実行する度じ異なる値が生成される。

問題 N1-1

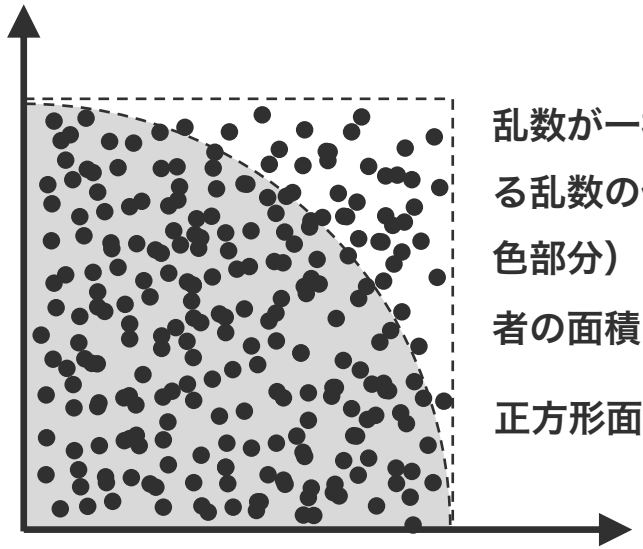
入力配列 a に対して 3 時点の移動平均を求めよ。




```
import numpy as np

a = np.array([8, 5, 2, 2, 8,
                5, 8, 2, 5])
```

問題 N1-2

np.random.rand(n) 関数は、0≦x<1 の一様分布から n 個の乱数を生成する関数である。0≦x<1 および 0≦y<1 の範囲で乱数を生成し、下図のように、正方形に含まれる乱数の個数と、第一象限にある単位円の内側に含まれる乱数の個数に着目して、円周率を小数 3 桁 (= 3.141...) まで正確に求めよ。



乱数が一様であれば、正方形内部に含まれる乱数の個数と第一象限にある単位円（灰色部分）に含まれる乱数の個数の比が、両者の面積の比に近似できる。

正方形面積：灰色部分面積 = 1 : $\pi/4$

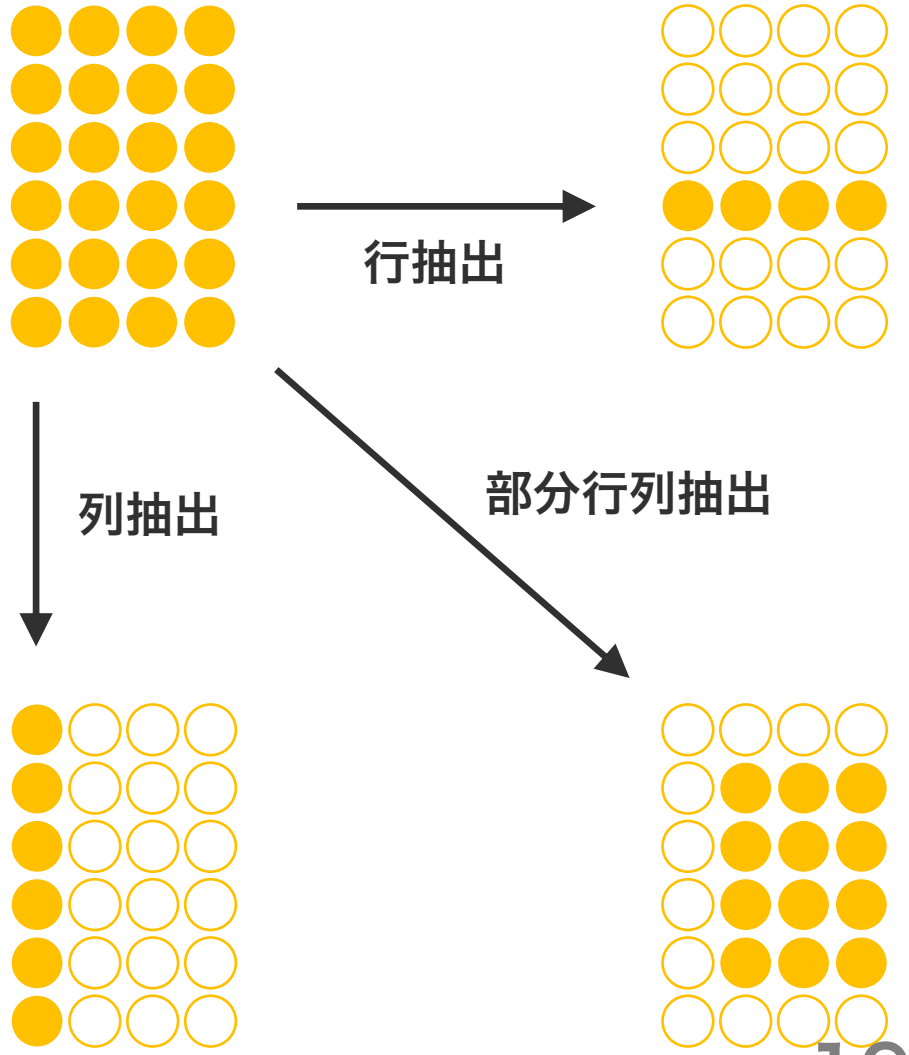
```
import numpy as np

n = 10000
x = np.random.rand(n)
y = np.random.rand(n)
```

NumPy

- 1 次元配列
- 2 次元配列
- 3 次元配列
- データの読み書き

2 次元配列

NumPy の２次元配列は、縦と横の構造を持ち、数学の行列のように、特定の行あるいは列を抽出したり、行列演算を行ったりすることができる。



2 次元配列 / np.array

2 次元配列を作成するには、2 次元のリストを作成して、
それを np.array 関数に代入することで作成する。

```
a = np.array([[11, 12, 13, 14, 15, 16],
              [21, 22, 23, 24, 25, 26],
              [31, 32, 33, 34, 35, 36],
              [41, 42, 43, 44, 45, 46],
              [51, 52, 53, 54, 55, 56],
              [61, 62, 63, 64, 65, 66]])
```

11	12	13	14	15	16
21	22	23	24	25	26
31	32	33	34	35	36
41	42	43	44	45	46
51	52	53	54	55	56
61	62	63	64	65	66

2 次元配列 / np.zeros, np.ones

2 次元配列も 1 次元配列と同様に、同じ値からなる配列を作る場合、np.zeros、np.ones、np.full などの関数を使う。なお、2 次元配列の場合、横と縦のサイズを指定する必要がある。

```
b = np.zeros((2, 5))
```

0	0	0	0	0
0	0	0	0	0

```
b = np.ones((5, 3))
```

1	1	1
1	1	1
1	1	1
1	1	1
1	1	1

2 次元配列 / np.full

2 次元配列も 1 次元配列と同様に、同じ値からなる配列を作る場合、np.zeros、np.ones、np.full などの関数を使う。なお、2 次元配列の場合、横と縦のサイズを指定する必要がある。

```
a = np.full((5, 3), np.nan)
```

nan	nan	nan
nan	nan	nan
nan	nan	nan
nan	nan	nan
nan	nan	nan

2 次元配列 / np.identity

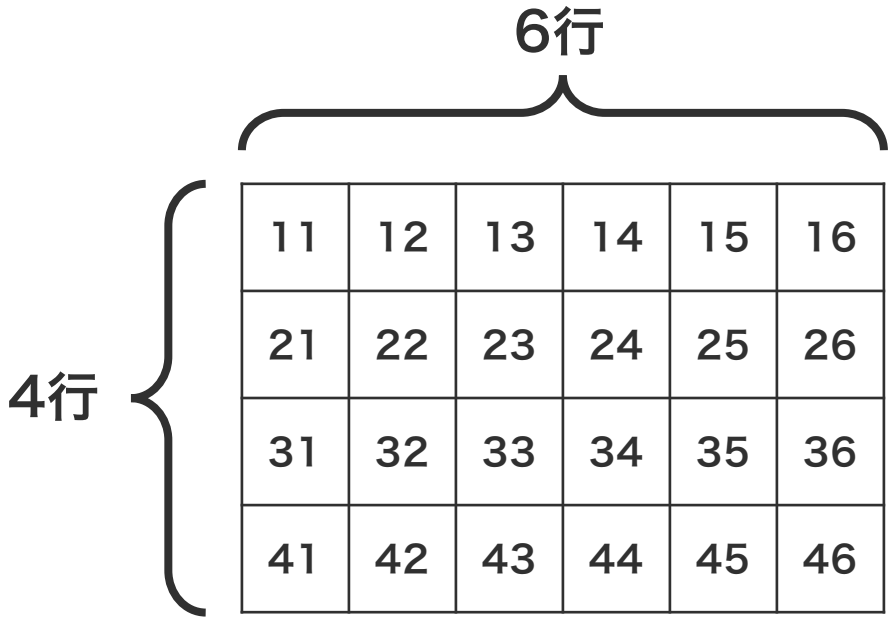
２次元配列の場合、np.identity 関数を使用して、単位行列を作成することができる。

```
a = np.identity(6)
```

1	0	0	0	0	0
0	1	0	0	0	0
0	0	1	0	0	0
0	0	0	1	0	0
0	0	0	0	1	0
0	0	0	0	0	1

2 次元配列

２次元行列の構造を調べるときは、ndim, shape, np.size などを利用する。ndim 属性には、配列の次元数が記録されている。shape 属性には、配列構造（サイズ）が記録されている。また、np.size 関数を使用することで、配列の特定の次元のサイズを取得することができる。



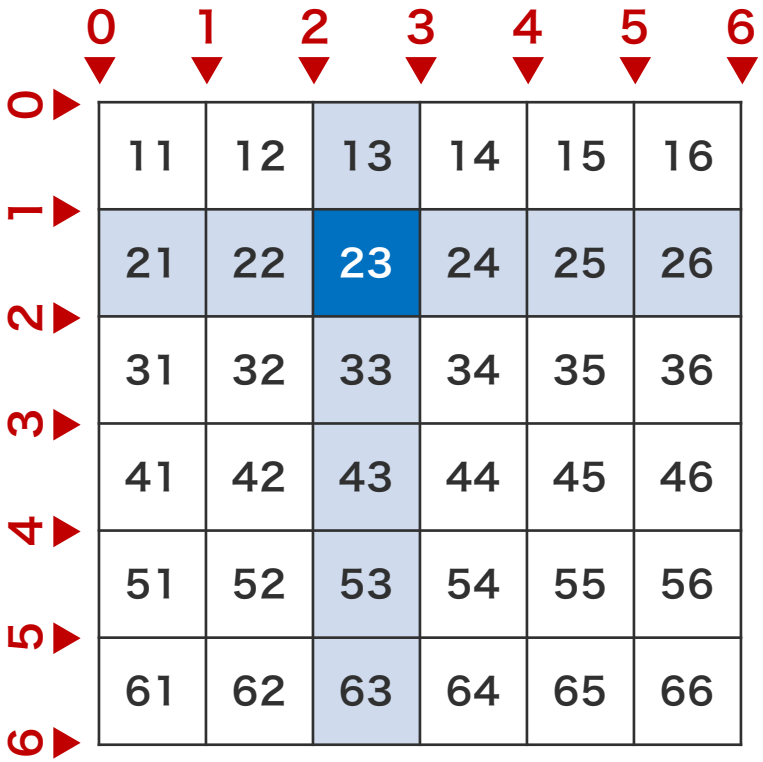

```
a = np.array([[11, 12, 13, 14, 15, 16],
              [21, 22, 23, 24, 25, 26],
              [31, 32, 33, 34, 35, 36],
              [41, 42, 43, 44, 45, 46]])


a.ndim
# 2


a.shape
# (4, 6)


np.size(a, axis=0)
# 4


np.size(a, axis=1)
# 6
```

2 次元配列

2 次元配列から要素を取り出すとき、位置番号を指定して取り出したり、スライスして取り出したりすることができる。

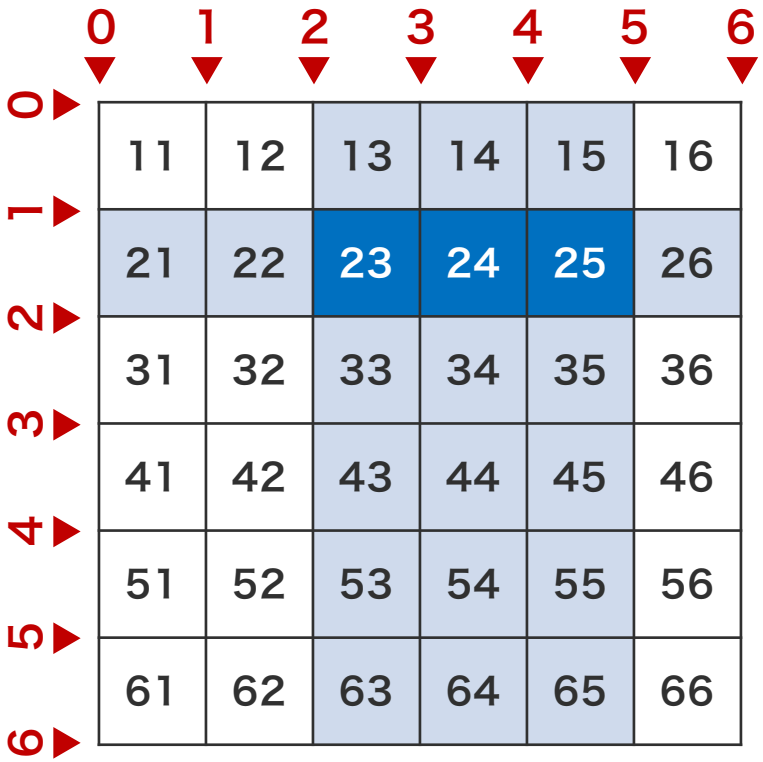
```
b = np.array([[11, 12, 13, 14, 15, 16],
              [21, 22, 23, 24, 25, 26],
              [31, 32, 33, 34, 35, 36],
              [41, 42, 43, 44, 45, 46],
              [51, 52, 53, 54, 55, 56],
              [61, 62, 63, 64, 65, 66]])

b[1, 2]


b[1, 2:5]


b[1:5, 3]


b[2:4, 1:6]
```

2 次元配列



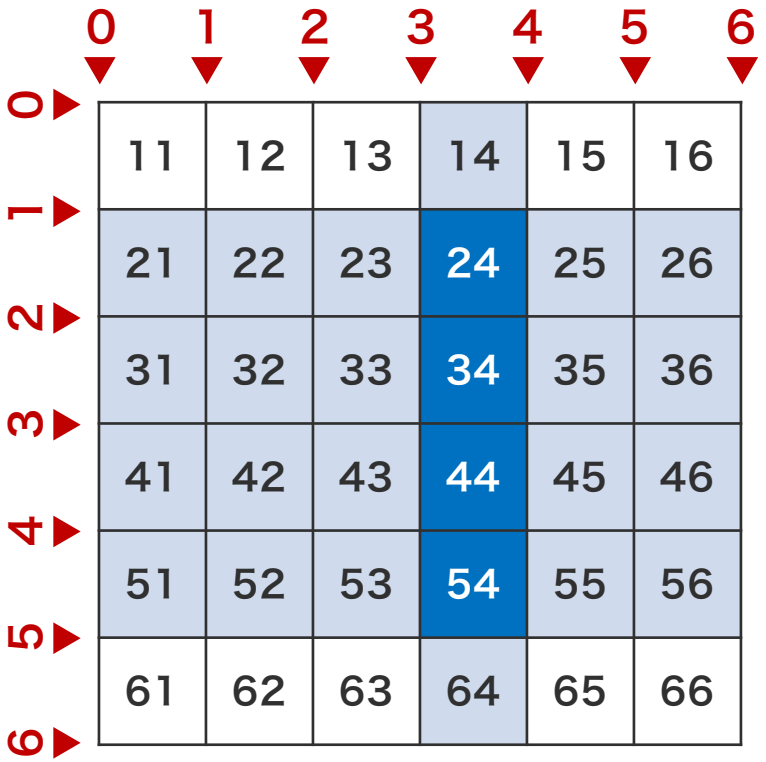

```
b = np.array([[11, 12, 13, 14, 15, 16],
              [21, 22, 23, 24, 25, 26],
              [31, 32, 33, 34, 35, 36],
              [41, 42, 43, 44, 45, 46],
              [51, 52, 53, 54, 55, 56],
              [61, 62, 63, 64, 65, 66]])

b[1, 2]
# 23

b[1, 2:5]


b[1:5, 3]


b[2:4, 1:6]
```

2 次元配列



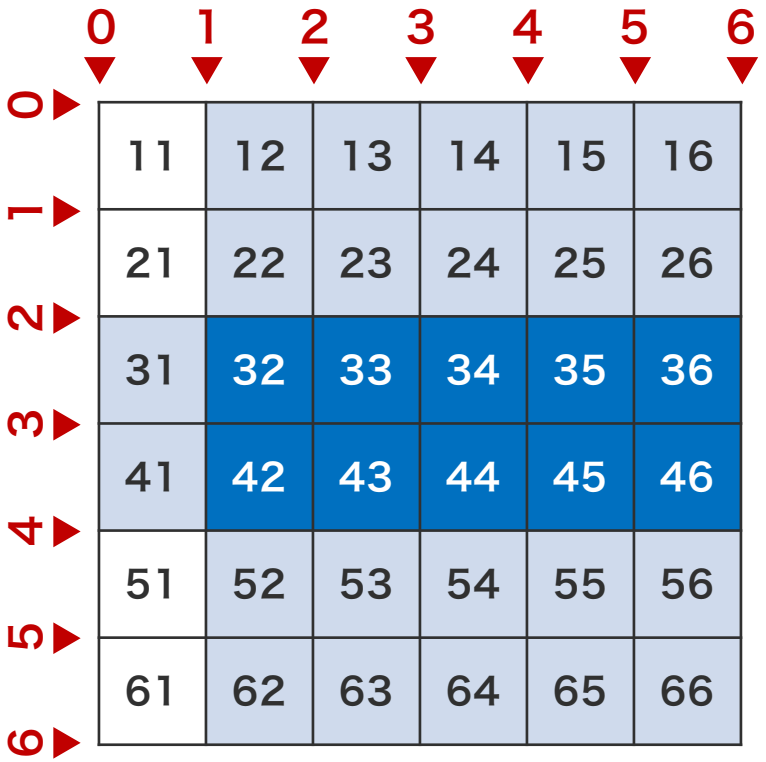

```
b = np.array([[11, 12, 13, 14, 15, 16],
              [21, 22, 23, 24, 25, 26],
              [31, 32, 33, 34, 35, 36],
              [41, 42, 43, 44, 45, 46],
              [51, 52, 53, 54, 55, 56],
              [61, 62, 63, 64, 65, 66]])

b[1, 2]
# 23

b[1, 2:5]
# array([23, 24, 25])

b[1:5, 3]


b[2:4, 1:6]
```

2 次元配列



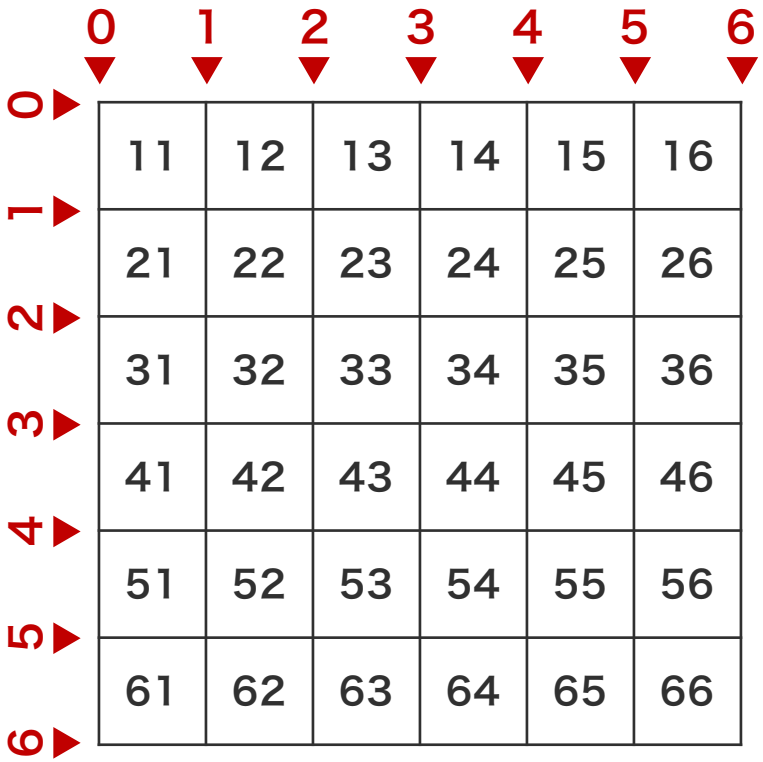

```
b = np.array([[11, 12, 13, 14, 15, 16],
              [21, 22, 23, 24, 25, 26],
              [31, 32, 33, 34, 35, 36],
              [41, 42, 43, 44, 45, 46],
              [51, 52, 53, 54, 55, 56],
              [61, 62, 63, 64, 65, 66]])

b[1, 2]
# 23

b[1, 2:5]
# array([23, 24, 25])

b[1:5, 3]
# array([24, 34, 44, 54])

b[2:4, 1:6]
```

2 次元配列



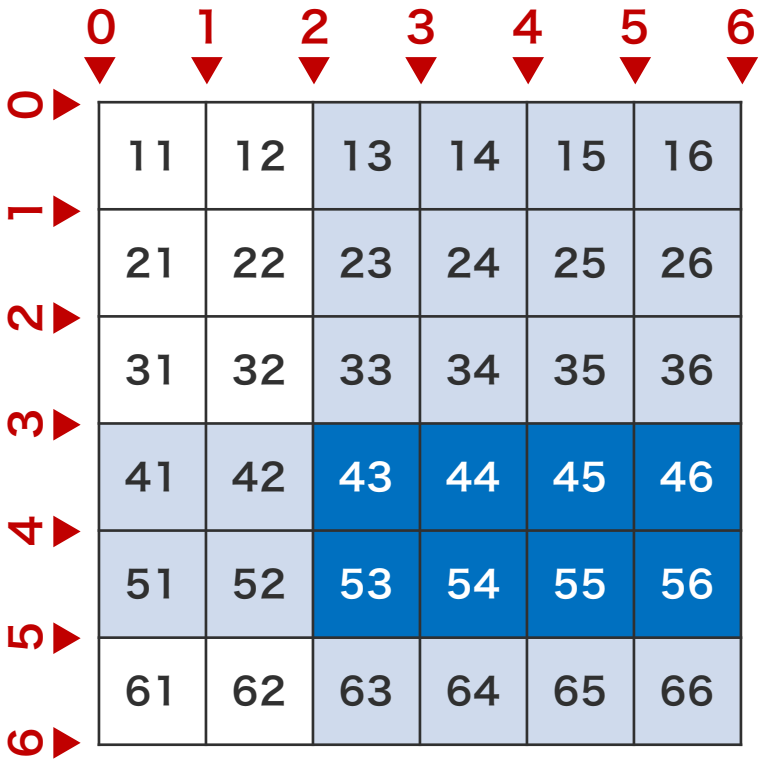

```
b = np.array([[11, 12, 13, 14, 15, 16],
              [21, 22, 23, 24, 25, 26],
              [31, 32, 33, 34, 35, 36],
              [41, 42, 43, 44, 45, 46],
              [51, 52, 53, 54, 55, 56],
              [61, 62, 63, 64, 65, 66]])

b[1, 2]
# 23

b[1, 2:5]
# array([23, 24, 25])

b[1:5, 3]
# array([24, 34, 44, 54])

b[2:4, 1:6]
# array([[32, 33, 34, 35, 36],
#        [42, 43, 44, 45, 46]])
```

2 次元配列



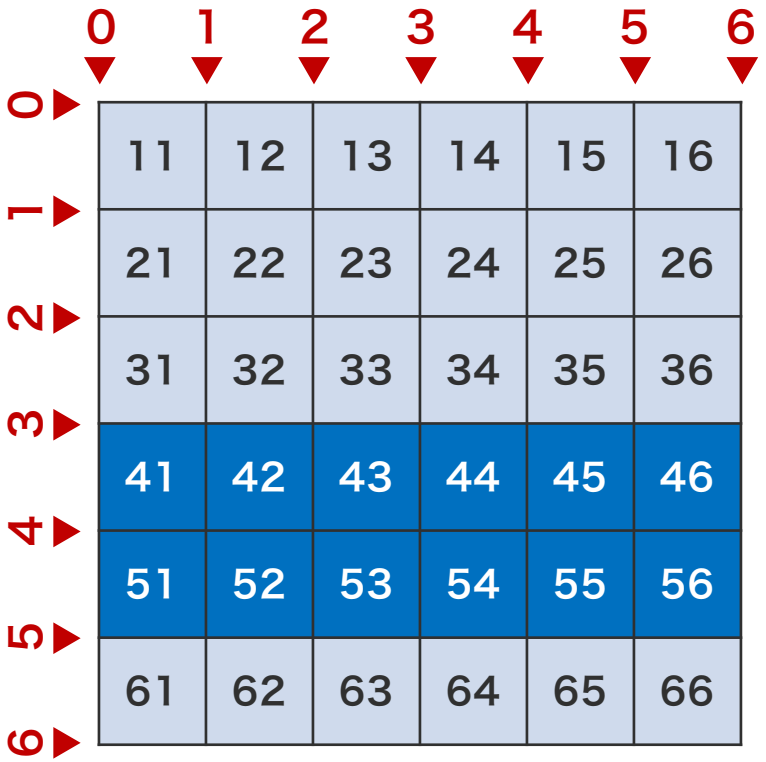

```
b = np.array([[11, 12, 13, 14, 15, 16],
              [21, 22, 23, 24, 25, 26],
              [31, 32, 33, 34, 35, 36],
              [41, 42, 43, 44, 45, 46],
              [51, 52, 53, 54, 55, 56],
              [61, 62, 63, 64, 65, 66]])

b[3:5, 2:]


b[3:5, :]


b[:, 2]


b[:, 1:3]
```

2 次元配列



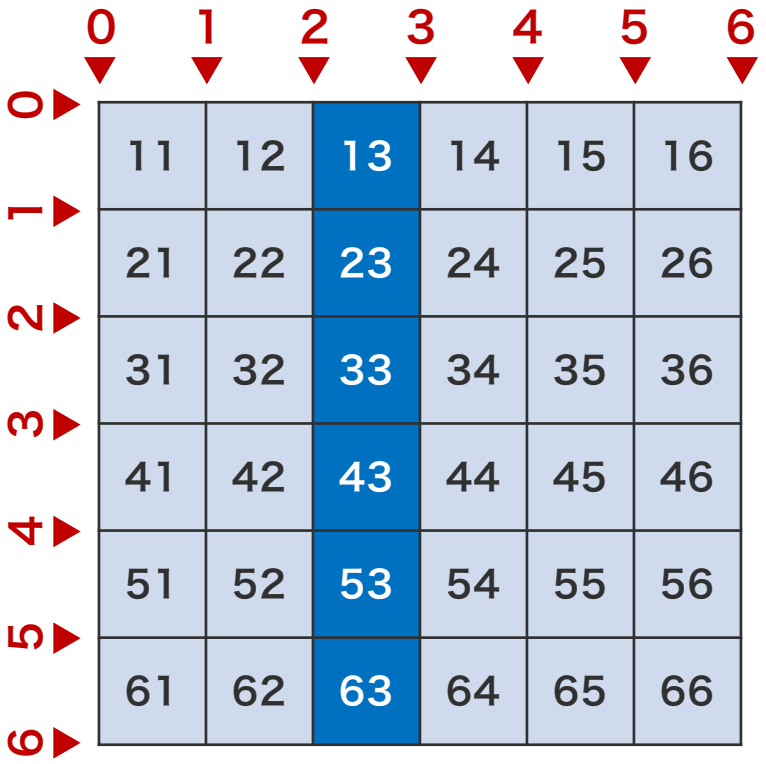

```
b = np.array([[11, 12, 13, 14, 15, 16],
              [21, 22, 23, 24, 25, 26],
              [31, 32, 33, 34, 35, 36],
              [41, 42, 43, 44, 45, 46],
              [51, 52, 53, 54, 55, 56],
              [61, 62, 63, 64, 65, 66]])

b[3:5, 2:]
# array([[43, 44, 45, 46],
#        [53, 54, 55, 56]])
b[3:5, :]


b[:, 2]


b[:, 1:3]
```

2 次元配列



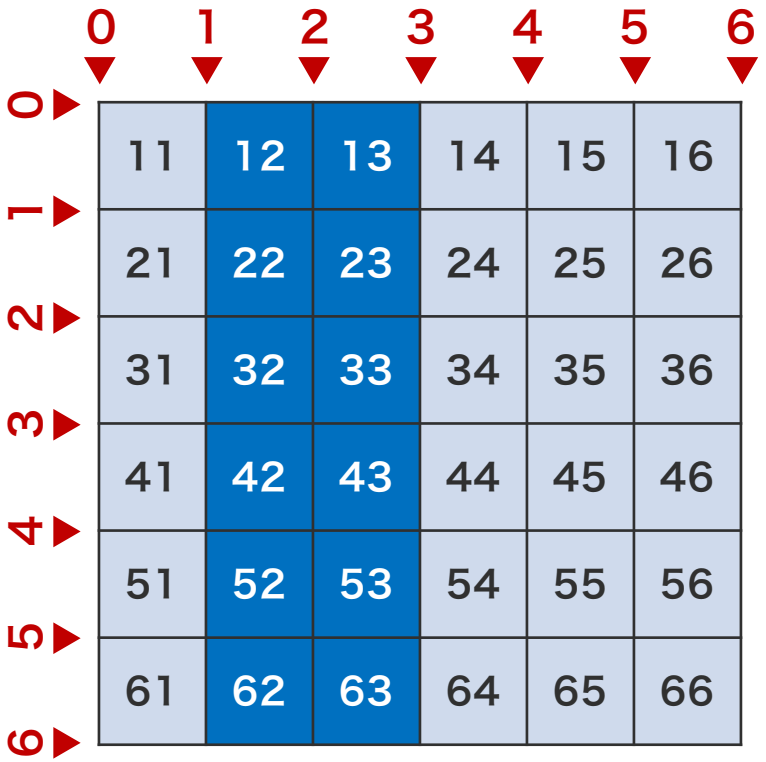

```
b = np.array([[11, 12, 13, 14, 15, 16],
              [21, 22, 23, 24, 25, 26],
              [31, 32, 33, 34, 35, 36],
              [41, 42, 43, 44, 45, 46],
              [51, 52, 53, 54, 55, 56],
              [61, 62, 63, 64, 65, 66]])

b[3:5, 2:]
# array([[43, 44, 45, 46],
#        [53, 54, 55, 56]])
b[3:5, :]
# array([[41, 42, 43, 44, 45, 46],
#        [51, 52, 53, 54, 55, 56]])
b[:, 2]


b[:, 1:3]
```

2 次元配列



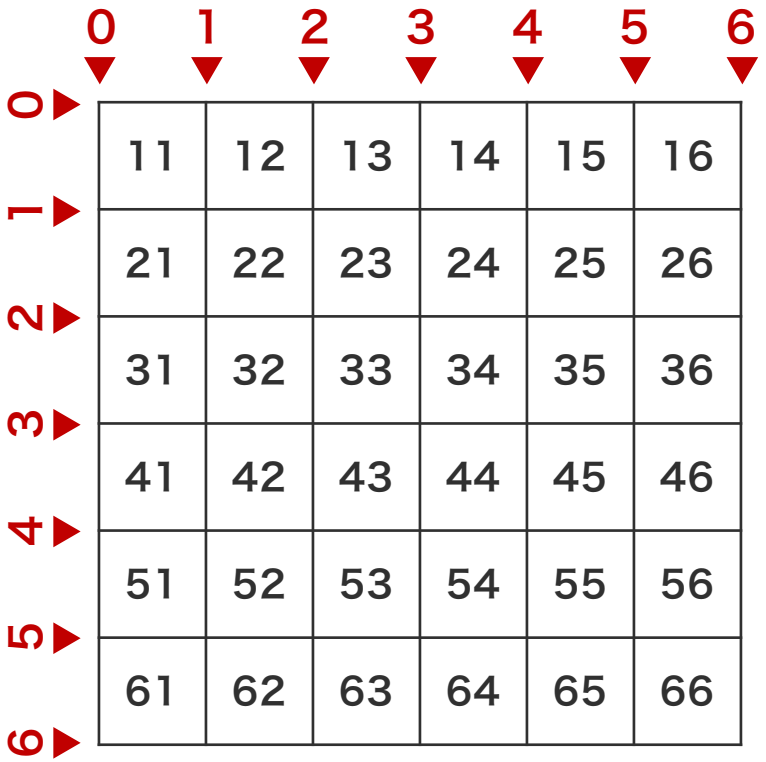

```
b = np.array([[11, 12, 13, 14, 15, 16],
              [21, 22, 23, 24, 25, 26],
              [31, 32, 33, 34, 35, 36],
              [41, 42, 43, 44, 45, 46],
              [51, 52, 53, 54, 55, 56],
              [61, 62, 63, 64, 65, 66]])

b[3:5, 2:]
# array([[43, 44, 45, 46],
#        [53, 54, 55, 56]])
b[3:5, :]
# array([[41, 42, 43, 44, 45, 46],
#        [51, 52, 53, 54, 55, 56]])
b[:, 2]
# array([13, 23, 33, 43, 53, 63])

b[:, 1:3]
```

2 次元配列



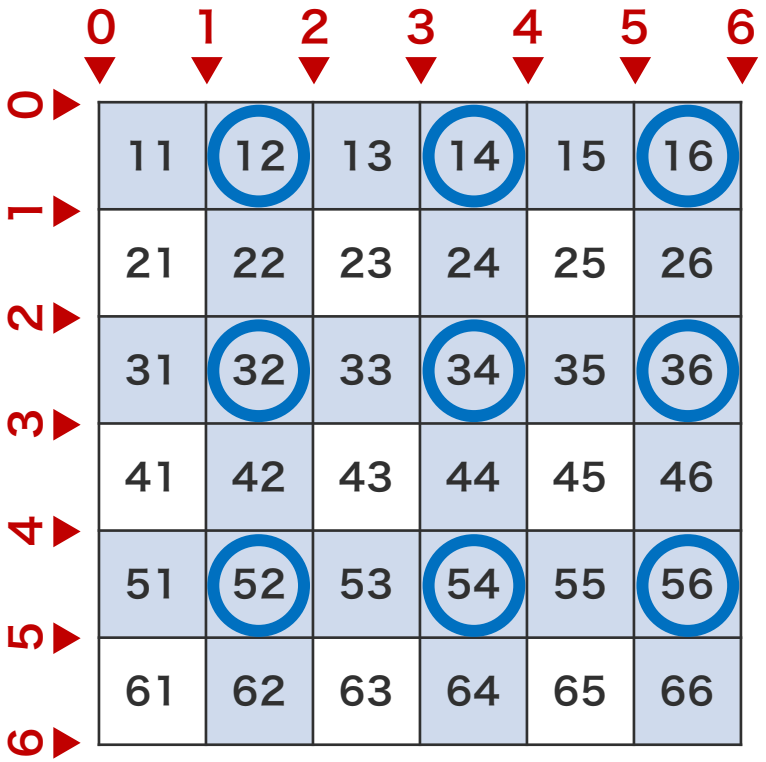

```
b = np.array([[11, 12, 13, 14, 15, 16],
              [21, 22, 23, 24, 25, 26],
              [31, 32, 33, 34, 35, 36],
              [41, 42, 43, 44, 45, 46],
              [51, 52, 53, 54, 55, 56],
              [61, 62, 63, 64, 65, 66]])

b[3:5, 2:]
# array([[43, 44, 45, 46],
#        [53, 54, 55, 56]])
b[3:5, :]
# array([[41, 42, 43, 44, 45, 46],
#        [51, 52, 53, 54, 55, 56]])
b[:, 2]
# array([13, 23, 33, 43, 53, 63])

b[:, 1:3]
# array([[12, 13], [22, 23], [32, 33],
#        [42, 43], [52, 53], [62, 63]])
```

2 次元配列 / 注意点



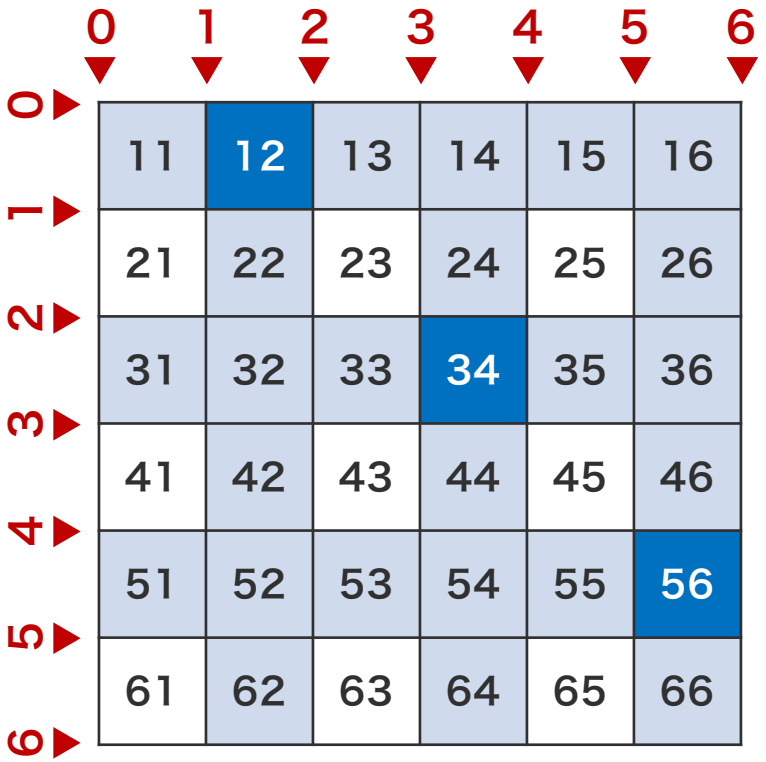

```
b = np.array([[11, 12, 13, 14, 15, 16],
              [21, 22, 23, 24, 25, 26],
              [31, 32, 33, 34, 35, 36],
              [41, 42, 43, 44, 45, 46],
              [51, 52, 53, 54, 55, 56],
              [61, 62, 63, 64, 65, 66]])

r = [0, 2, 4]
c = [1, 3, 5]

b[r, c]
```

2 次元配列 / 注意点



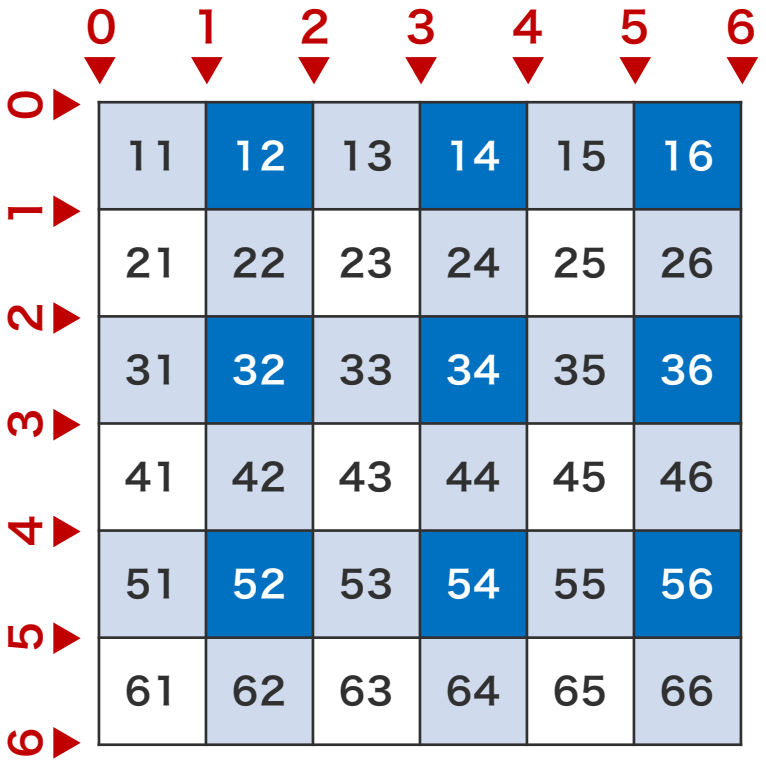

```
b = np.array([[11, 12, 13, 14, 15, 16],
              [21, 22, 23, 24, 25, 26],
              [31, 32, 33, 34, 35, 36],
              [41, 42, 43, 44, 45, 46],
              [51, 52, 53, 54, 55, 56],
              [61, 62, 63, 64, 65, 66]])

r = [0, 2, 4]
c = [1, 3, 5]

b[r, c]
```

2 次元配列 / 注意点




```
b = np.array([[11, 12, 13, 14, 15, 16],
              [21, 22, 23, 24, 25, 26],
              [31, 32, 33, 34, 35, 36],
              [41, 42, 43, 44, 45, 46],
              [51, 52, 53, 54, 55, 56],
              [61, 62, 63, 64, 65, 66]])

r = [0, 2, 4]
c = [1, 3, 5]

b[r, c]
# array([12, 34, 56])
```

2 次元配列 / 注意点




```
b = np.array([[11, 12, 13, 14, 15, 16],
              [21, 22, 23, 24, 25, 26],
              [31, 32, 33, 34, 35, 36],
              [41, 42, 43, 44, 45, 46],
              [51, 52, 53, 54, 55, 56],
              [61, 62, 63, 64, 65, 66]])

r = [0, 2, 4]
c = [1, 3, 5]

b[r, c]
# array([12, 34, 56])



b[np.ix_(r, c)]
# array([[12, 14, 16],
#        [32, 34, 36],
#        [52, 54, 56]])
```

行列計算

NumPy の 2 次元配列に、行列演算が定義されている。配列の各要素同士の加減乗除の他に、内積や外積も簡単に求めることができる。

計算式	計算内容
`a + b`	各要素の足し算
`a - b`	各要素の引き算
`a * b`	各要素の掛け算（アダマール積）
`a / b`	各要素の割り算
`np.dot(a, b)`	行列同士の内積
`np.outer(a, b)`	行列同士の外積（テンソル積）
`np.sum(a)`	全要素の和
`a.T`	行列 a の転置行列

```
import numpy as np

a = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]])

b = np.array([[1, 1, 1],
              [0, 1, 1],
              [0, 0, 1]])

a + b
# array([[ 2, 3, 4],
#        [ 4, 6, 7],
#        [ 7, 8, 10]])

np.dot(a, b)
# array([[ 1,  3,  6],
#        [ 4,  9, 15],
#        [ 7, 15, 24]])
```

問題 N2-1

カーネル行列 b を用いて、行列 a に対して畳み込み演算を行い、その結果を出力せよ。

```
a = np.array([[0, 2, 0, 1, 2, 0],
              [1, 0, 3, 0, 0, 2],
              [3, 1, 0, 2, 2, 1],
              [2, 2, 2, 1, 0, 1],
              [1, 0, 1, 3, 2, 2],
              [0, 3, 1, 0, 2, 0]])

b = np.array([[1, 0, 0],
              [0, 1, 0],
              [0, 0, 1]])
```

入力行列 a

0	2	0	1	2	0
1	0	3	0	0	2
3	1	0	2	2	1
2	2	2	1	0	1
1	0	1	3	2	2
0	3	1	0	2	0

カーネル行列 b

1	0	0
0	1	0
0	0	1

```
0*1 + 2*0 + 0*0 +
1*0 + 0*1 + 3*0 +
3*0 + 1*0 + 0*1 = 3
```

演算結果

0	7	2	2
4	1	5	3
6	6	3	4
3	3	7	3

問題 N2-1

カーネル行列 b を用いて、行列 a に対して畳み込み演算を行い、その結果を出力せよ。

```
a = np.array([[0, 2, 0, 1, 2, 0],
              [1, 0, 3, 0, 0, 2],
              [3, 1, 0, 2, 2, 1],
              [2, 2, 2, 1, 0, 1],
              [1, 0, 1, 3, 2, 2],
              [0, 3, 1, 0, 2, 0]])

b = np.array([[1, 0, 0]
              [0, 1, 0],
              [0, 0, 1]])
```

入力行列 a

0	2	0	1	2	0
1	0	3	0	0	2
3	1	0	2	2	1
2	2	2	1	0	1
1	0	1	3	2	2
0	3	1	0	2	0

カーネル行列 b

1	0	0
0	1	0
0	0	1

```
2*1 + 0*0 + 1*0 +
0*0 + 3*1 + 0*0 +
1*0 + 0*0 + 2*1 = 7
```

演算結果

0	7	2	2
4	1	5	3
6	6	3	4
3	3	7	3

問題 N2-1

カーネル行列 b を用いて、行列 a に対して畳み込み演算を行い、その結果を出力せよ。

```
a = np.array([[0, 2, 0, 1, 2, 0],
              [1, 0, 3, 0, 0, 2],
              [3, 1, 0, 2, 2, 1],
              [2, 2, 2, 1, 0, 1],
              [1, 0, 1, 3, 2, 2],
              [0, 3, 1, 0, 2, 0]])

b = np.array([[1, 0, 0],
              [0, 1, 0],
              [0, 0, 1]])
```

入力行列 a

0	2	0	1	2	0
1	0	3	0	0	2
3	1	0	2	2	1
2	2	2	1	0	1
1	0	1	3	2	2
0	3	1	0	2	0

カーネル行列 b

1	0	0
0	1	0
0	0	1

```
0*1 + 1*0 + 2*0 +
3*0 + 0*1 + 0*0 +
0*0 + 2*0 + 2*1 = 2
```

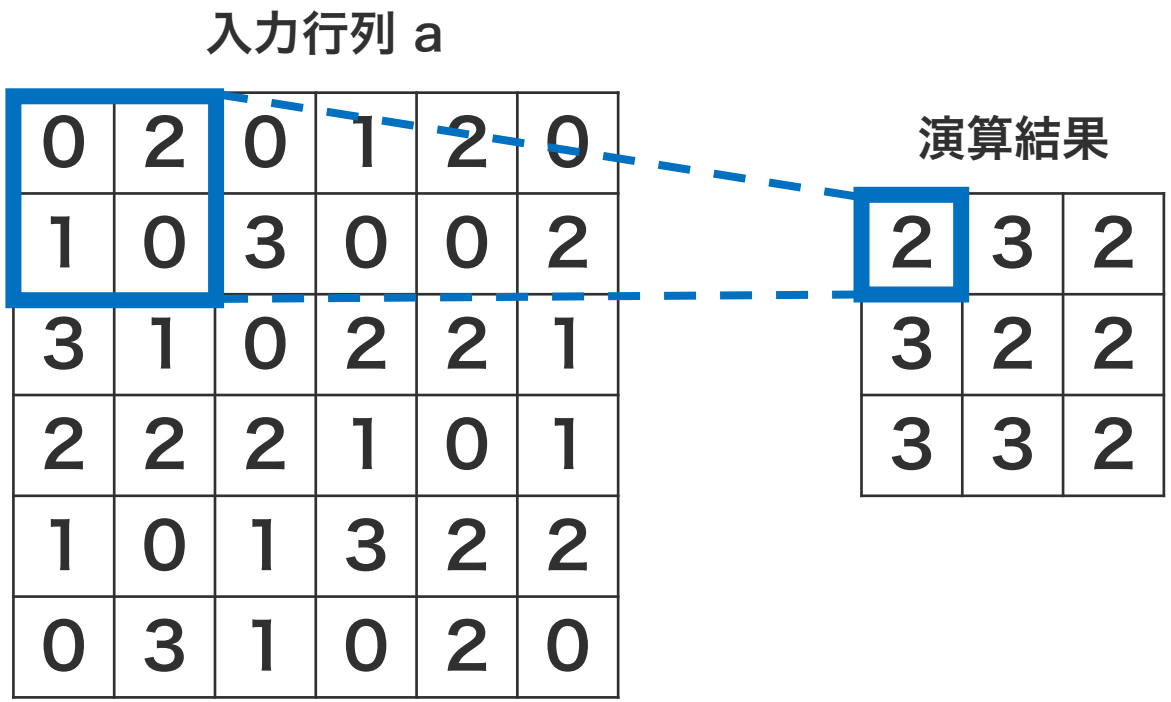
演算結果

0	7	2	2
4	1	5	3
6	6	3	4
3	3	7	3

問題 N2-2

行列 a に対して 2x2 範囲の最大プーリング演算を行い、
その結果を出力せよ。

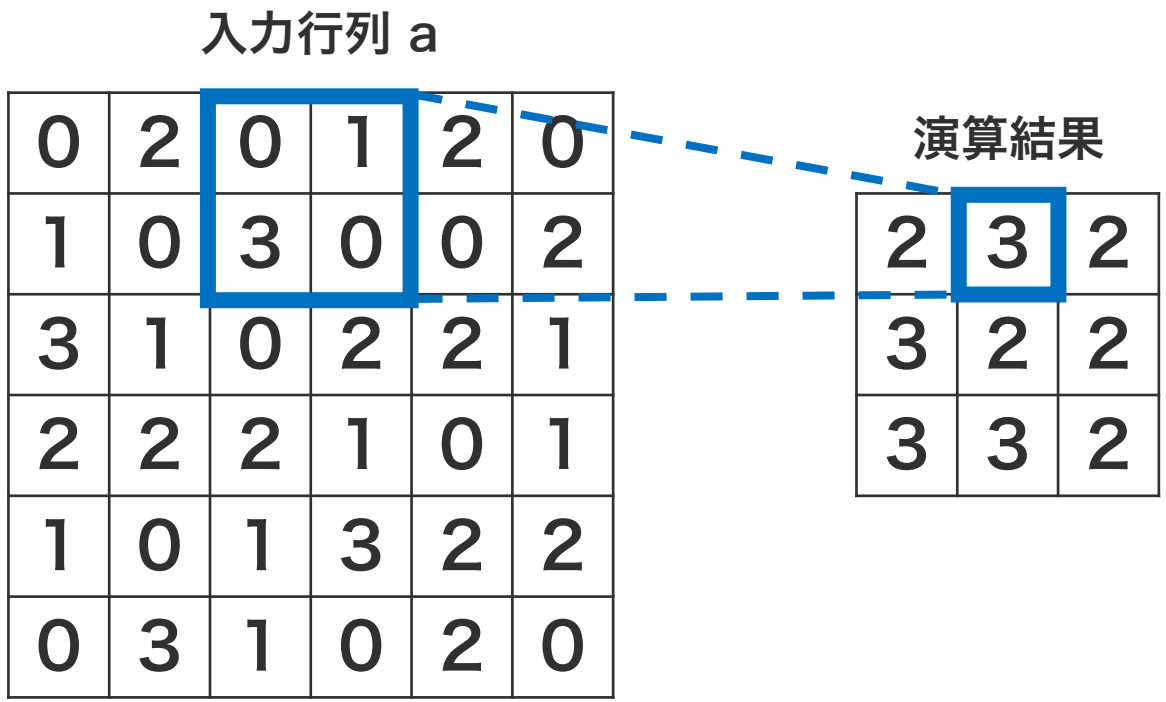
```
a = np.array([[0, 2, 0, 1, 2, 0],
              [1, 0, 3, 0, 0, 2],
              [3, 1, 0, 2, 2, 1],
              [2, 2, 2, 1, 0, 1],
              [1, 0, 1, 3, 2, 2],
              [0, 3, 1, 0, 2, 0]])
```



問題 N2-2

行列 a に対して 2x2 範囲の最大プーリング演算を行い、
その結果を出力せよ。

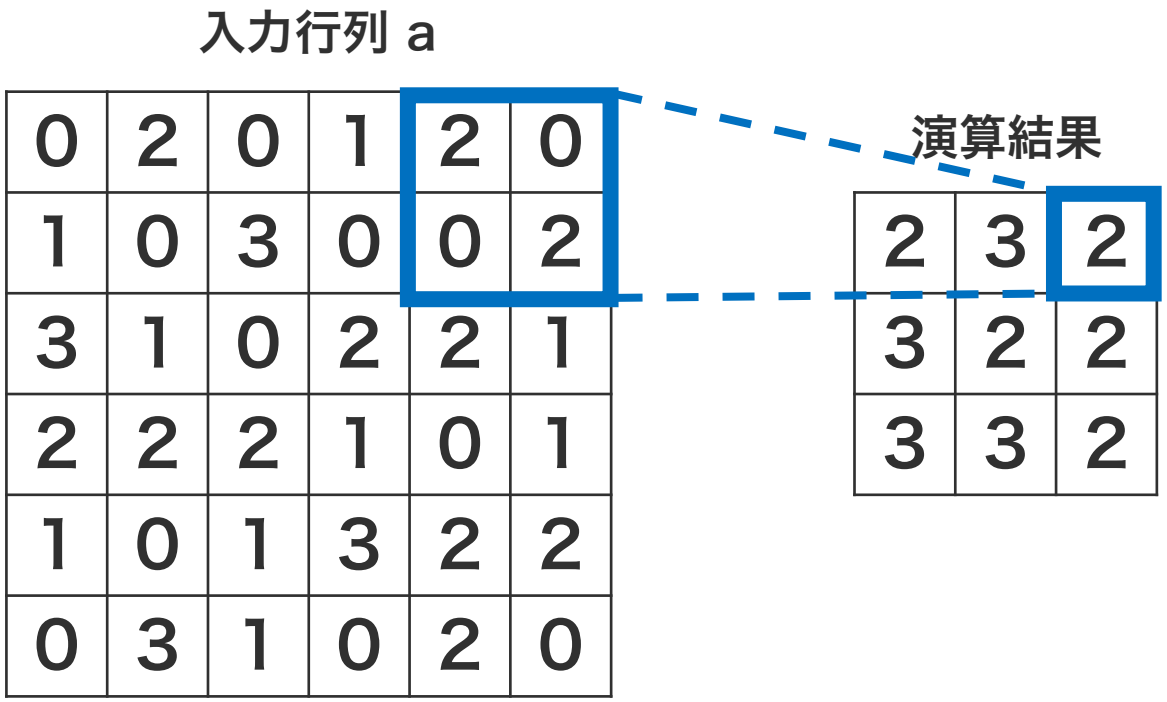
```
a = np.array([[0, 2, 0, 1, 2, 0],
              [1, 0, 3, 0, 0, 2],
              [3, 1, 0, 2, 2, 1],
              [2, 2, 2, 1, 0, 1],
              [1, 0, 1, 3, 2, 2],
              [0, 3, 1, 0, 2, 0]])
```



問題 N2-2

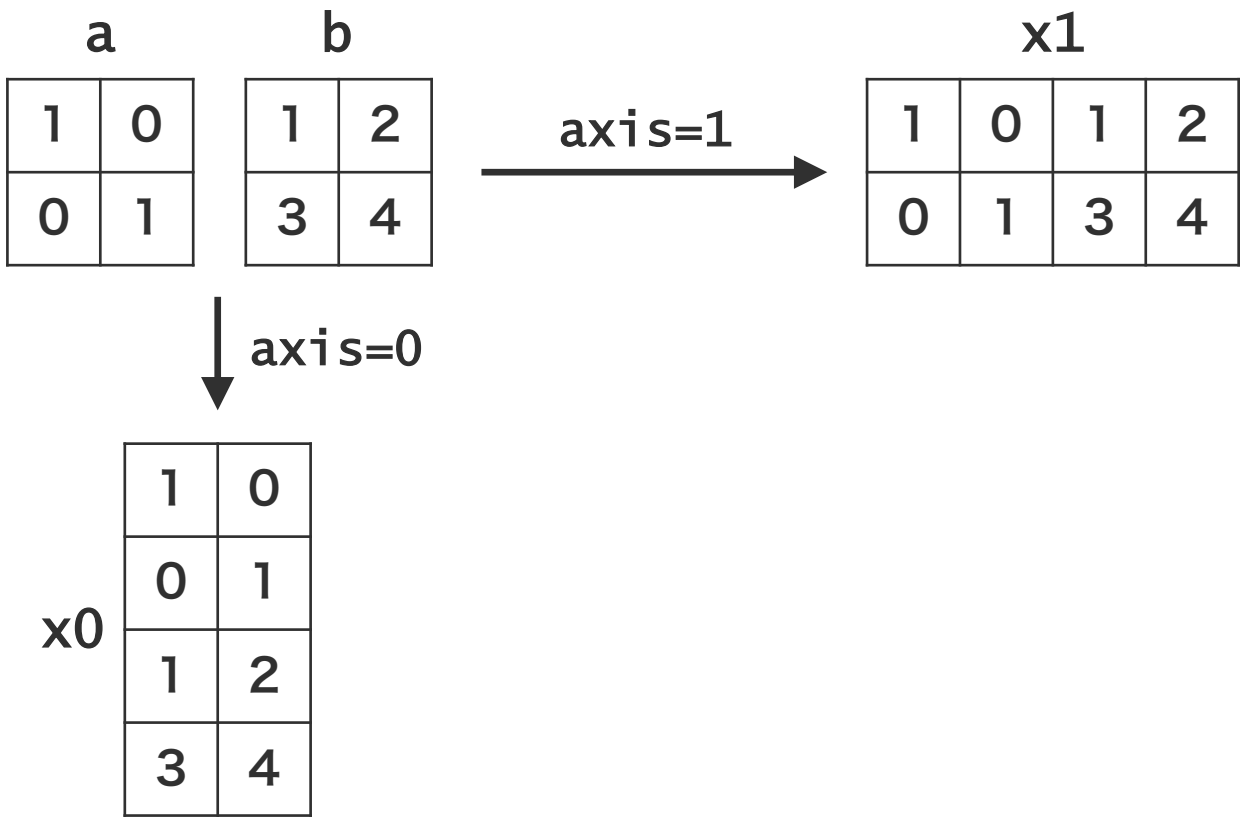
行列 a に対して 2x2 範囲の最大プーリング演算を行い、
その結果を出力せよ。

```
a = np.array([[0, 2, 0, 1, 2, 0],
              [1, 0, 3, 0, 0, 2],
              [3, 1, 0, 2, 2, 1],
              [2, 2, 2, 1, 0, 1],
              [1, 0, 1, 3, 2, 2],
              [0, 3, 1, 0, 2, 0]])
```



行列操作 / np.concatenate

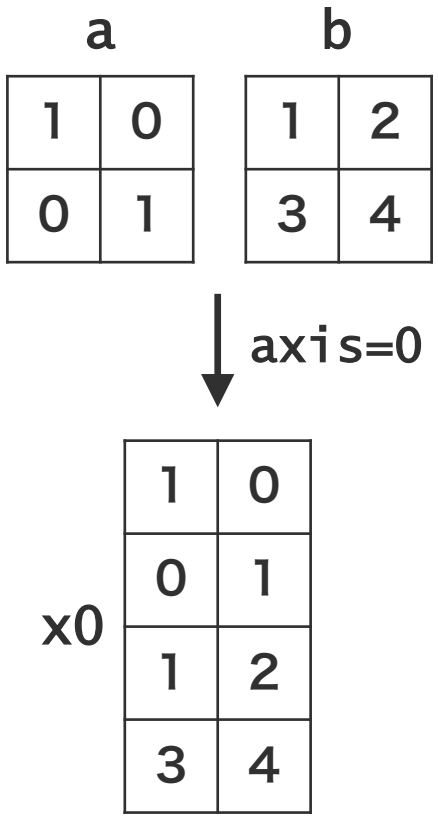
np.concatenate 関数は、複数のNumPy 配列を結合する機能を持つ。この関数を使うとき、結合する軸（次元）方向を axis 引数で指定。省略された場合は axis=0 となる。




```
a = [[1, 0], [0, 1]]
b = [[1, 2], [3, 4]]

x0 = np.concatenate([a, b], axis=0)
x0
# array([[1, 0],
#        [0, 1],
#        [1, 2],
#        [3, 4]])

x1 = np.concatenate([a, b], axis=1)
x1
# array([[1, 0, 1, 2],
#        [0, 1, 3, 4]])

c = [9, 9]

y0 = np.concatenate([a, c], axis=0)
# ValueError: all the input arrays must
# have same number of dimensions, ...
```

行列操作 / np.vstack

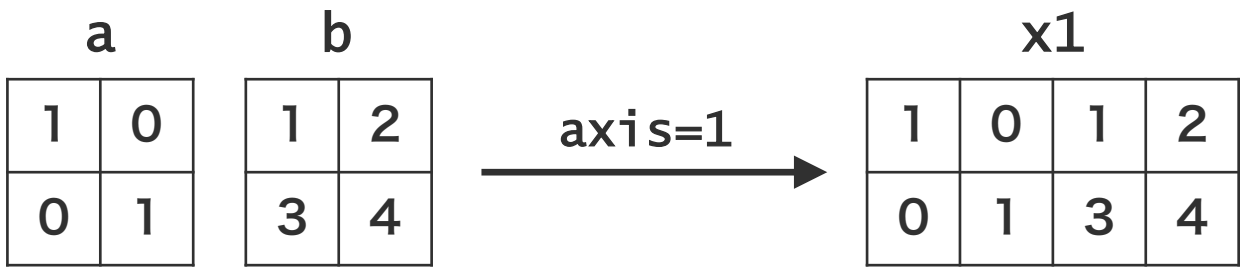
np.vstack 関数は複数の配列を第 1 次元方向に結合する機能を持つ。axis=0 のときの np.concatenate 関数の機能と同じ。




```
a = [[1, 0], [0, 1]]
b = [[1, 2], [3, 4]]


x = np.concatenate([a, b], axis=0)
x
# array([[1, 0],
#        [0, 1],
#        [1, 2],
#        [3, 4]])


y = np.vstack([a, b])
y
# array([[1, 0],
#        [0, 1],
#        [1, 2],
#        [3, 4]])
```

行列操作 / np.hstack

np.hstack 関数は複数の配列を第 2 次元方向に結合する機能を持つ。axis=1 のときの np.concatenate 関数の機能と同じ。




```
a = [[1, 0], [0, 1]]
b = [[1, 2], [3, 4]]


x = np.concatenate([a, b], axis=1)
x
# array([[1, 0, 1, 2],
#        [0, 1, 3, 4]])


y = np.hstack([a, b])
y
# array([[1, 0, 1, 2],
#        [0, 1, 3, 4]])
```

行列操作 / flatten

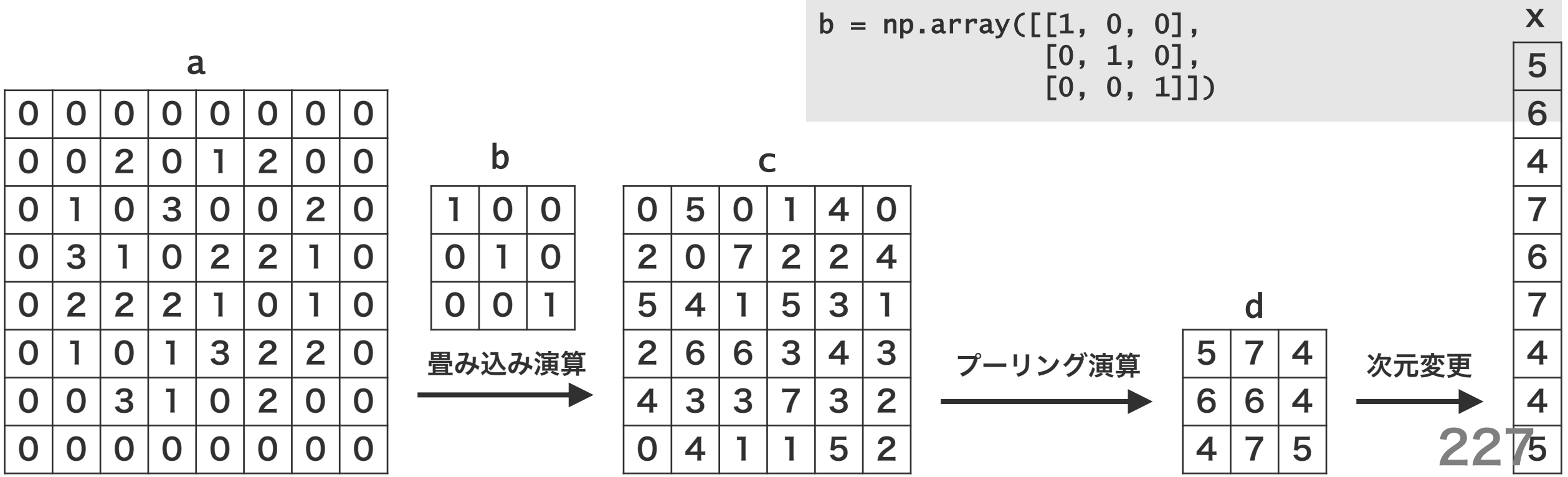
flatten 関数は、多次元配列を 1 次元配列に変更する関数である。多次元配列を崩すときに、横方向から崩すのか（order='C'）、縦方向から崩すのか（order='F'）を指定することもできる。

```
a = np.array([[1, 0, 2, 4],
              [0, 1, 3, 9]])


x = a.flatten()
x
# array([1, 0, 2, 4, 0, 1, 3, 9])


y = a.flatten(order='F')
y
# array([1, 0, 0, 1, 2, 3, 4, 9])
```

問題 N2-3

下図のように入力配列 a に対して、畳み込み演算と最大プーリング演算を行い、最後にその結果を 1 次元配列に変更せよ。なお、演算方法については問題 N2-1, N2-2 を参照のこと。

```
a = np.array([[0, 0, 0, 0, 0, 0, 0, 0],
              [0, 0, 2, 0, 1, 2, 0, 0],
              [0, 1, 0, 3, 0, 0, 2, 0],
              [0, 3, 1, 0, 2, 2, 1, 0],
              [0, 2, 2, 2, 1, 0, 1, 0],
              [0, 1, 0, 1, 3, 2, 2, 0],
              [0, 0, 3, 1, 0, 2, 0, 0],
              [0, 0, 0, 0, 0, 0, 0, 0]])

b = np.array([[1, 0, 0],
              [0, 1, 0],
              [0, 0, 1]])
```

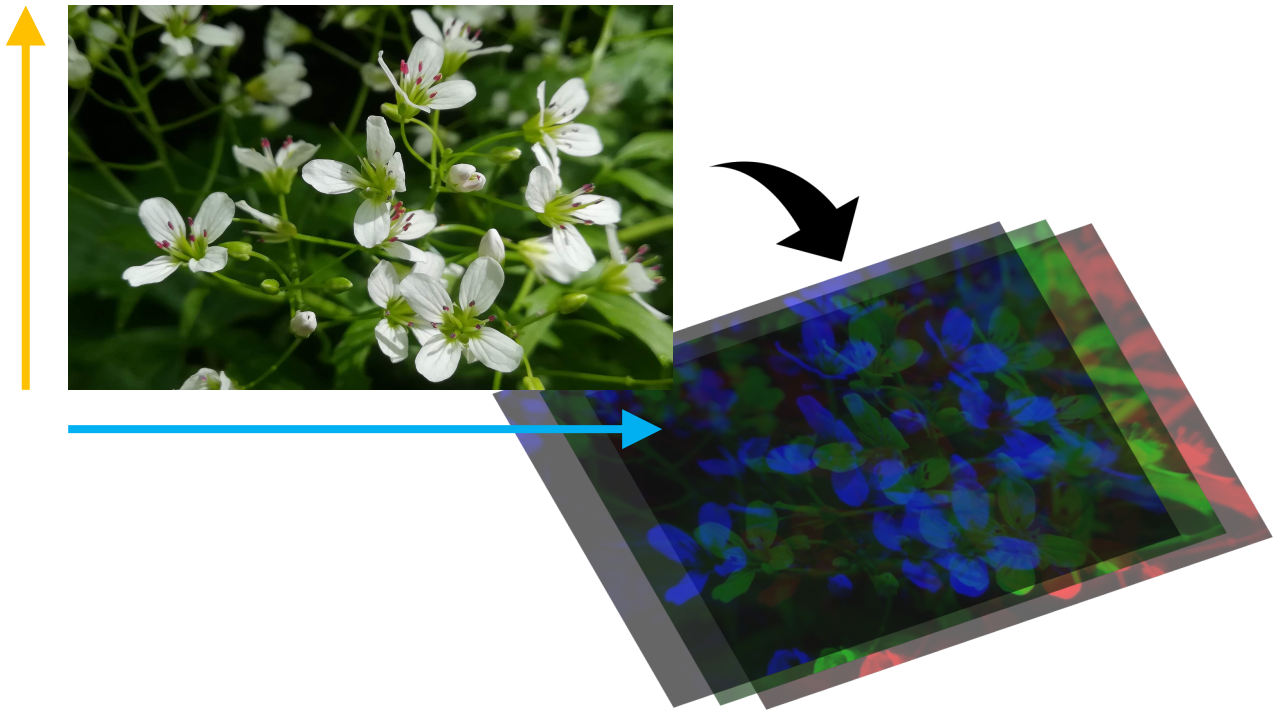


NumPy

- １ 次元配列
- ２ 次元配列
- ３ 次元配列
- データの読み書き

3 次元配列

3 次元配列は、画像解析のときによく使われる。デジタルカメラで撮られている画像の色は、赤・緑・青の 3 原色によって表される。そのため、縦 n 横 m の画像は、n x m の配列がまずあり、その各 (n, m) 要素には RGB の 3 つの要素が含まれているようになる。



```
b = np.array([
    [[1, 0, 1], [1, 1, 0], [1, 1, 1]],
    [[1, 0, 1], [1, 1, 0], [1, 1, 1]],
    [[1, 0, 1], [1, 1, 0], [1, 1, 1]],
    [[1, 0, 1], [1, 1, 0], [1, 1, 1]],
    [[1, 0, 1], [1, 1, 0], [1, 1, 1]]
])
b
```

```
array([[[1 0 1]
        [1 1 0]
        [1 1 1]]
       [[1 0 1]
        [1 1 0]
        [1 1 1]]
       [[1 0 1]
        [1 1 0]
        [1 1 1]]
       [[1 0 1]
        [1 1 0]
        [1 1 1]]
       [[1 0 1]
        [1 1 0]
        [1 1 1]]])
```

多次元配列

多次元配列は、深層学習による画像分類のときに使われることが多い。深層学習では、一度に画像を 16〜64 枚ずつまとめて学習するので、3 次元の画像を複数枚束ねたときに、4 次元の配列が使われる。

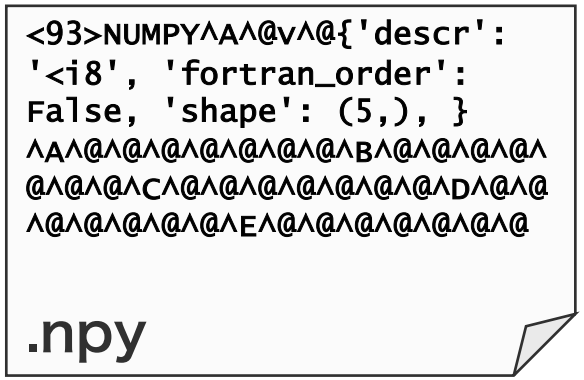
NumPy

- １ 次元配列
- ２ 次元配列
- ３ 次元配列
- データの読み書き

データの書き出し / バイナリー

NumPy の配列データをテキストファイルに書き出して保存する方法は、2 通りある。1 つは、配列データをバイナリ形式で書き出す方法である。この場合、データの構造やメタ情報も同時に書き出されるため、情報の欠損が少ない。1 つの配列を 1 つのファイルに保存するとき、save 関数を使う。複数個の配列を 1 つのファイルに保存するとき、savez を使う。

```
# binary (.npy)

a = np.array([1, 2, 3, 4, 5])

np.save('data.npy', a)



# binary (.npz)

a = np.array([1, 2, 3, 4, 5])
b = np.array([0, 0, 1, 0, 1])

np.savez('objects.npz', a = a, b = b)
```



データの読み込み / バイナリー

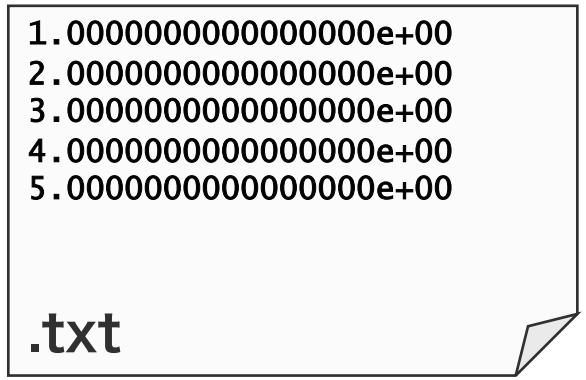
NumPy のバイナリデータを保存しているファイルからデータを読み込むときに load 関数を使用する。複数の配列を含むファイルを読み込むときも load 関数を使用するが、その際、複数個の配列はディクショナリに似たデータ構造でオブジェクトに代入される。そのオブジェクトの .files 属性には、配列の名前の一覧が保存されている。

```
# binary (.npy)

a = np.load('data.npy')
a
# array([1, 2, 3, 4, 5])



# binary (.npz)

x = np.load('objects.npz')
x.files
# ['a', 'b']

x['a']
# array([1, 2, 3, 4, 5])

x['b']
# array([0, 0, 1, 0, 1])
```

データの書き出し / テキスト

NumPy の配列データをファイルに保存するもう 1 つの方法は、配列データをテキストデータとして書き出す方法である。可読性はあるものの、桁数の多い小数などを正確に保存できないことがある。テキストファイルとして書き出すとき、savetxt 関数を使用する。この関数のオプションを指定することで、データ間の区切り文字や小数の有効桁数を指定することができる。

```
# text (.txt)

a = np.array([1, 2, 3, 4, 5])


np.savetxt('data.tsv', a)


np.savetxt('data.csv', a,
           fmt='%.18e',
           delimiter=',')
```



データの読み込み / テキスト

テキストファイルを読み込むときに loadtxt 関数または genfromtxt 関数を使用する。loadtxt 関数を使用するとき、読み込み時にエラーが発生した場合、ファイルの区切り文字やヘッダーに合わせて、loadtxt 関数のオプションを調整することで対処できる。また、loadtxt 関数は欠損値を含むファイルを処理できないため、欠損値を含む場合、genefromtxt 関数を使用する。

```
# text (.txt)

a = np.loadtxt('data.txt')
a
# array([1., 2., 3., 4., 5.])
```

Pandas

- シリーズ
- データフレーム
- 表データ処理

python™

アプリ開発やシステム管理など様々な用途で利用できるような汎用機能を提供する。

- リスト
- 2 次元リスト

pandas

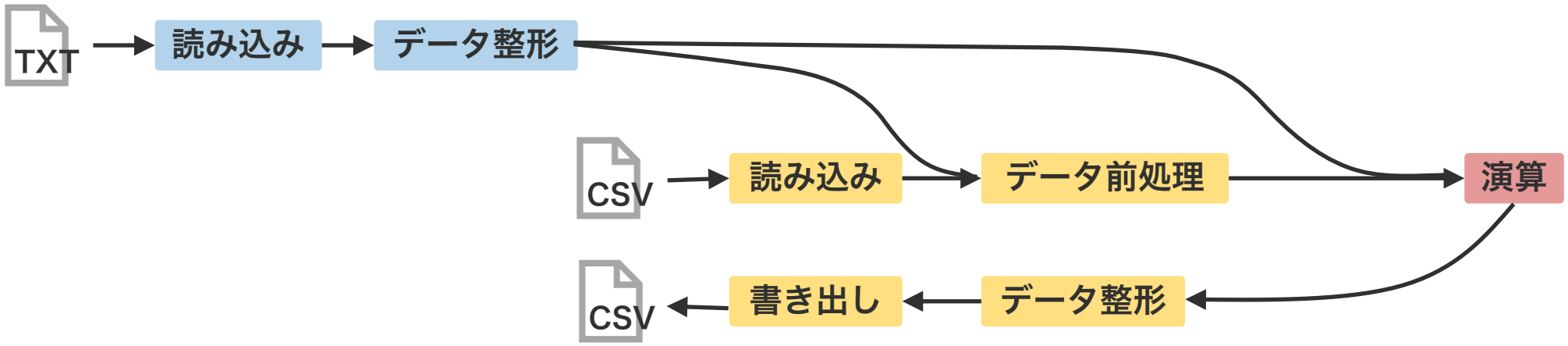
CSV ファイルの読み書きや表データの操作や整形などに特化した機能を提供する。

- シリーズ
- データフレーム

NumPy

整形されたデータに対して、情報量をできるだけ落とさずに、高速に演算を行う機能を提供する。

- 配列
- 2 次元配列

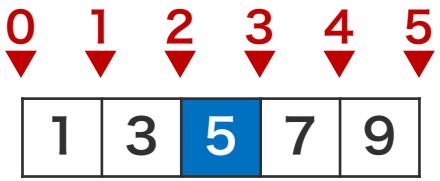


Pandas

- シリーズ
- データフレーム
- 表データ処理

シリーズ

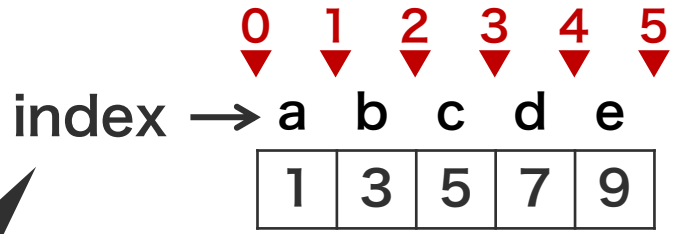
Pandas のシリーズは、1 次元の配列データを扱うときに使用する。pd.Series 関数にリストを代入して作成する。シリーズから要素を取り出すときは、位置番号を指定して取り出す。




```
import pandas as pd

x = pd.Series([1, 3, 5, 7, 9])



x[2]
# 5
```

シリーズ

シリーズは、リストと異なり、各値を位置番号と index の両方で管理している。シリーズを pd.Series 関数で作成するときに、index が自動的に作られるが、自ら指定して作ることもできる。

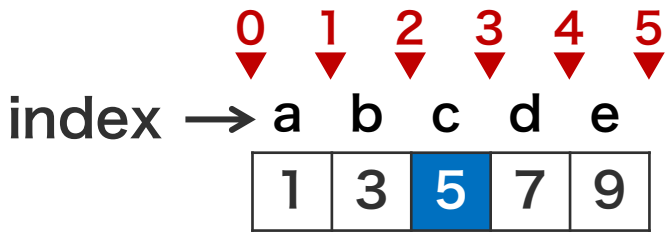


シリーズの各要素に位置番号の他に index と呼ばれる索引が付けられる。

```
import pandas as pd

x = pd.Series([1, 3, 5, 7, 9])
x
# 0    1
# 1    3
# 2    5
# 3    7
# 4    9
# dtype: int64

x = pd.Series([1, 3, 5, 7, 9],
              index=['a', 'b', 'c', 'd', 'e'])
x
# a    1
# b    3
# c    5
# d    7
# e    9
# dtype: int64
```

シリーズ

シリーズを作成するとき、文字列を index に指定した場合、その文字列でシリーズの各要素を取得できるようになる。




```
import pandas as pd

x = pd.Series([1, 3, 5, 7, 9],
              index=['a', 'b', 'c', 'd', 'e'])
x
# a    1
# b    3
# c    5
# d    7
# e    9
# dtype: int64


x[2]
# 5


x['c']
# 5
```

シリーズ

シリーズは、値と index の両方のデータを保持している。シリーズから値だけを NumPy の 1 次元配列として取得したい場合は、x.values のように取得する。また、シリーズから index だけを取得したい場合は、x.index を使用する。

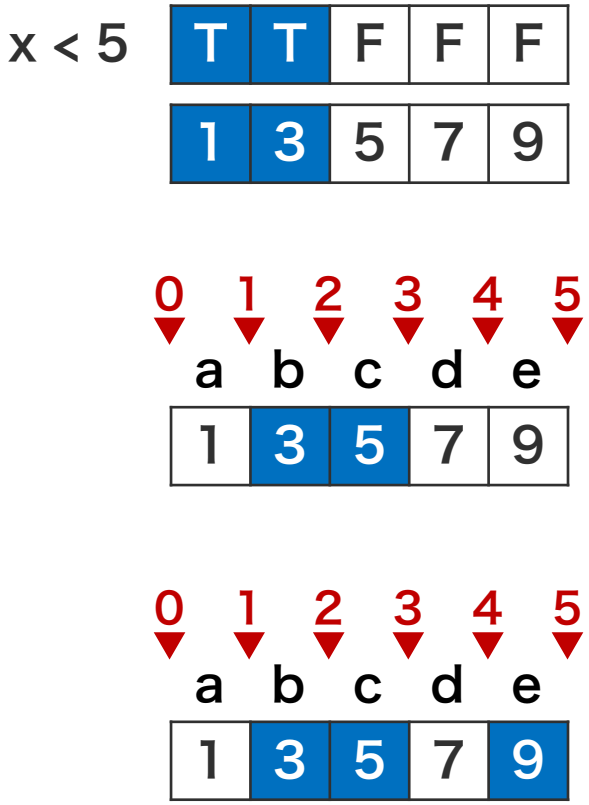
```
import pandas as pd

x = pd.Series([1, 3, 5, 7, 9],
      index=['a', 'b', 'c', 'd', 'e'])

x
# a    1
# b    3
# c    5
# d    7
# e    9
# dtype: int64

x.values
# array([1, 3, 5, 7, 9])

x.index
# Index(['a', 'b', 'c', 'd', 'e'],
#       dtype='object')
```

シリーズ

シリーズから要素を取得するとき、位置番号と index で取得できるほか、NumPy のようにスライスしたり、フィルター（ブーリアンベクトル）を使用して取得したりすることもできる。

```
import pandas as pd
x = pd.Series([1, 3, 5, 7, 9],
              index=['a', 'b', 'c', 'd', 'e'])

x[x < 5]




x[1:3]




k = ['b', 'c', 'e']
x[k]
```

シリーズ




```
import pandas as pd
x = pd.Series([1, 3, 5, 7, 9],
              index=['a', 'b', 'c', 'd', 'e'])

x[x < 5]
# a    1
# b    3
# dtype: int64

x[1:3]
# b    3
# c    5
# dtype: int64

k = ['b', 'c', 'e']
x[k]
# b    3
# c    5
# e    9
# dtype: int64
```

問題 P1-1

赤色で書かれているオブジェクトが保持している値を答えよ。

```
import pandas as pd
x = pd.Series([1, 3, 5, 7, 9],
              index=['a', 'b', 'c', 'd', 'e'])


x[['a', 'c', 'e']]




keep1 = (1 < x)
keep2 = (x < 7)
x[keep1 & keep2]
```

```
import pandas as pd
x = pd.Series([1, 3, 5, 7, 9],
              index=['a', 'b', 'c', 'd', 'e'])

x['a'] = 2
x[0] = 4
x.values


x[x > 4] = 6
x.values


x[x % 2 == 0] = 1
x.values
```

シリーズ同士の計算

シリーズ同士は NumPy の配列と同様に四則演算が可能である。計算対象のシリーズの長さが同一でない場合は、見かけ上、短い方に欠損値を埋め込んだ上で計算される。また、シリーズとスカラーの計算も定義されている。この場合、スカラーが自動的に、シリーズと同じ長さに展開されて、計算が行われる（ブロードキャスト）。

```
import pandas as pd

x = pd.Series([1, 3, 5, 9])
y = pd.Series([2, 4, 6, 8])
z = pd.Series([1, 1])
w = 2

a = x + y
a.values
#array([ 3,  7, 11, 17])


b = x - z
b.values
# array([ 0.,  2., nan, nan])

c = x * w
c.values
# array([ 2,  6, 10, 18])
```

シリーズ同士の計算

シリーズに index がついている場合、計算は index に基づいて計算される。どちらか一方のシリーズにしか存在しない index の場合、存在しない方を欠損値として扱う。

```
import pandas as pd


x = pd.Series([1, 3, 5, 7],
              index=['a', 'b', 'c', 'd'])
y = pd.Series([2, 4, 6, 8],
              index=['d', 'c', 'b', 'a'])
a = x + y
a.values
# array([9, 9, 9, 9])


x = pd.Series([1, 3, 5, 7],
              index=['a', 'b', 'c', 'd'])
y = pd.Series([2, 4, 6, 8],
              index=['a', 'b', 'd', 'c'])
a = x * y
a.values
# array([ 2,  12, 40, 42)
```

計算後の結果は index 順に並べ替えられる。x * y と y * x の計算結果は同じである。

シリーズ同士の計算

シリーズに index がついている場合、計算は index に基づいて計算される。どちらか一方のシリーズにしか存在しない index の場合、存在しない方を欠損値として扱う。

```
import pandas as pd


x = pd.Series([1, 3, 5, 7],
              index=['a', 'b', 'd', 'e'])

y = pd.Series([2, 4, 6, 8],
              index=['a', 'b', 'c', 'e'])


a = x + y
a.values
# array([ 3.,  7., nan, nan, 15.])


b = x * y
b.values
# array([ 2., 12., nan, nan, 56.])
```

要約統計量

Pandas のシリーズに対して、平均、分散、中央値などの要約統計量を計算するメソッドが多く用意されている。

```
import numpy as np
import pandas as pd
x = pd.Series([1, 2, 3, 4, 5])

x.count()
# 5

x.min()
# 1

x.max()
# 5

x.idxmax()
# 4

x.quantile(0.25)
# 2.0
```

```
x.sum()
# 15

x.mean()
# 3.0

x.median()
# 3.0

x.var()
# 2.5

x.std()
# 1.5811388300841898

x.cumsum().values
# array([ 1,  3,  6, 10, 15])
```

欠損値 / dropna

Pandas のシリーズに欠損値・非数値（np.nan）を含めることができる。欠損値は、Pandas で用意されたメソッドを使って取り除いたり、その位置を調べたりすることができる。

メソッド	動作
dropna	シリーズ中の np.nan を取り除く。
fillna	シリーズ中の np.nan を指定した値で置き換える。
isnull	シリーズ中の各要素が np.nan かどうかを True と False で表す。
notnull	is.null と反対の動作を行う。

欠損値除去後のシリーズの要素数が変化するので、複数のシリーズを同時に解析するとき、要素数の違いによりブロードキャスト機能が働き、想定外の計算が行われる可能があることに注意。

```
import numpy as np
import pandas as pd

x = pd.Series([1, 3, np.nan, 7, np.nan])
x
# a     3.0
# b     7.0
# c     NaN
# d     NaN
# e    15.0
# dtype: float64

y = x.dropna()
y
# 0 1.0
# 1 3.0
# 3 7.0
# dtype: float64
```

欠損値 / fillna

Pandas のシリーズに欠損値・非数値（np.nan）を含めることができる。欠損値は、Pandas で用意されたメソッドを使って取り除いたり、その位置を調べたりすることができる。

メソッド	動作
dropna	シリーズ中の np.nan を取り除く。
fillna	シリーズ中の np.nan を指定した値で置き換える。
isnull	シリーズ中の各要素が np.nan かどうかを True と False で表す。
notnull	is.null と反対の動作を行う。

合理的な根拠なしに、すべての欠損値を特定の値に置き換えてはならない。fillna メソッドを使用するときは十分に注意すること。

```
import numpy as np
import pandas as pd

x = pd.Series([1, 3, np.nan, 7, np.nan])


y = x.fillna(0)
y
# 0 1.0
# 1 3.0
# 2 0.0
# 3 7.0
# 4 0.0
# dtype: float64
```

欠損値 / isnull

Pandas のシリーズに欠損値・非数値（np.nan）を含めることができる。欠損値は、Pandas で用意されたメソッドを使って取り除いたり、その位置を調べたりすることができる。

メソッド	動作
dropna	シリーズ中の np.nan を取り除く。
fillna	シリーズ中の np.nan を指定した値で置き換える。
isnull	シリーズ中の各要素が np.nan かどうかを True と False で表す。
notnull	is.null と反対の動作を行う。

```
import numpy as np
import pandas as pd

x = pd.Series([1, 3, np.nan, 7, np.nan])


y = x.isnull()
y
# 0 False
# 1 False
# 2 True
# 3 False
# 4 True
# dtype: bool
```

欠損値 / notnull

Pandas のシリーズに欠損値・非数値（np.nan）を含めることができる。欠損値は、Pandas で用意されたメソッドを使って取り除いたり、その位置を調べたりすることができる。

メソッド	動作
dropna	シリーズ中の np.nan を取り除く。
fillna	シリーズ中の np.nan を指定した値で置き換える。
isnull	シリーズ中の各要素が np.nan かどうかを True と False で表す。
notnull	is.null と反対の動作を行う。

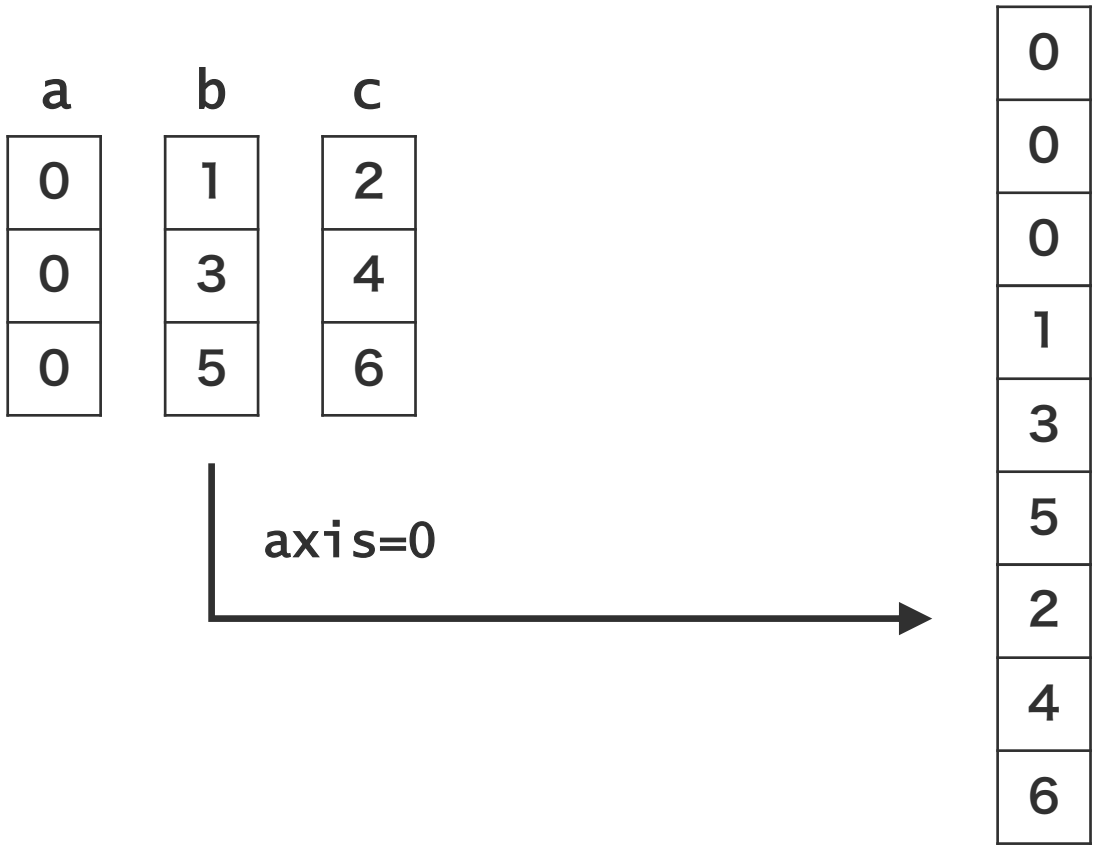
```
import numpy as np
import pandas as pd

x = pd.Series([1, 3, np.nan, 7, np.nan])


y = x.notnull()
y
# 0 True
# 1 True
# 2 False
# 3 True
# 4 False
# dtype: bool
```

Pandas

- シリーズ
- データフレーム
- 表データ処理

データフレーム / pd.concat

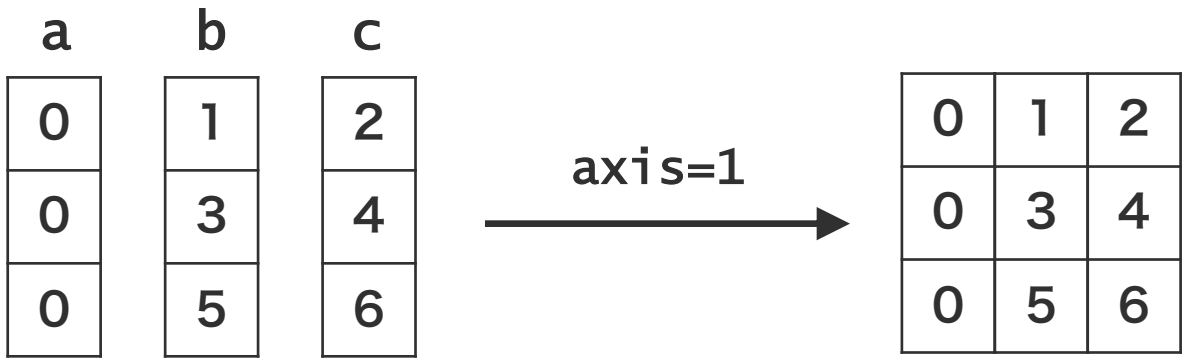
Pandas では、行列型のデータをデータフレームと呼ぶ。データフレームは、シリーズを行方向あるいは列方向に束ねることで作成される。シリーズ同士を束ねるとき pd.concat 関数を使用する。




```
import pandas as pd
a = pd.Series([0, 0, 0])
b = pd.Series([1, 3, 5])
c = pd.Series([2, 4, 6])

x = pd.concat([a, b, c])
x
# 0    0
# 1    0
# 2    0
# 0    1
# 1    3
# 2    5
# 0    2
# 1    4
# 2    6
# dtype: int64

x.__class__.__name__
# 'Series'
```

データフレーム / pd.concat

pd.concat 関数の axis 引数を指定することで、複数のシリーズを列方向に束ねることもできる。




```
import pandas as pd
a = pd.Series([0, 0, 0])
b = pd.Series([1, 3, 5])
c = pd.Series([2, 4, 6])

y = pd.concat([a, b, c], axis=1)
y
#    0  1  2
# 0  0  1  2
# 1  0  3  4
# 2  0  5  6

y.__class__.__name__
# 'DataFrame'
```

データフレーム / ディクショナリ

リストを値として保存しているディクショナリを、Pandas のデータフレームに変換することもできる。このとき、ディクショナリのキーは、データフレームの列名に変換される。

```
import pandas as pd


d = {'buna' : [1, 0, 1, 0, 1],
     'kashi': [1, 3, 5, 7, 9],
     'nara' : [0, 2, 4, 6, 8]}


df = pd.DataFrame(d)
df
#    buna kashi  nara
# 0     1     1     0
# 1     0     3     2
# 2     1     5     4
# 3     0     7     6
# 4     1     9     8
```

データフレーム / ディクショナリ

シリーズを値として保存しているディクショナリも Pandas のデータフレームに変換することもできる。ディクショナリのキーは、データフレームの列名に変換される。また、シリーズの index は、データフレームの index （行名）に変換される。

```
import pandas as pd

w1 = pd.Series([1, 0, 1, 0, 1],
         index=['a', 'b', 'c', 'd', 'e'])
w2 = pd.Series([1, 3, 5, 7, 9],
         index=['a', 'b', 'c', 'd', 'e'])
w3 = pd.Series([0, 2, 4, 6, 8],
         index=['a', 'b', 'c', 'd', 'e'])

d = {'buna': w1, 'kashi': w2,
     'nara': w3}
df = pd.DataFrame(d)
df
#    buna kashi  nara
# a     1     1     0
# b     0     3     2
# c     1     5     4
# d     0     7     6
# e     1     9     8
```

データフレーム / ディクショナリ

index を含むシリーズの場合、データフレームはそれらの index に基づいて作られる。データフレームは index で並べ替えられるため、その順番は必ずしもシリーズの順番を反映しないことに注意。

```
import pandas as pd

w1 = pd.Series([1, 0, 1, 0, 1],
         index=['b', 'a', 'c', 'd', 'f'])
w2 = pd.Series([1, 3, 5, 7, 9],
         index=['a', 'b', 'c', 'f', 'e'])
w3 = pd.Series([0, 2, 4, 6, 8],
         index=['c', 'b', 'a', 'd', 'e'])

d = {'buna': w1, 'kashi': w2,
     'nara': w3}
df = pd.DataFrame(d)
df
#    buna  kashi  nara
# a   0.0    1.0   4.0
# b   1.0    3.0   2.0
# c   1.0    5.0   0.0
# d   0.0    NaN   6.0
# e   NaN    9.0   8.0
# f   1.0    7.0   NaN
```

データフレーム / 二次元リスト

二次元リストを作成し、それをデータフレームに変換することもできる。この際に、データフレームの列名（columns）と行名（index）が自動的に振られる。なお、データフレームを作成するときに、columns と index 引数を利用することで、列名と行名を自由につけることができる。

```
import pandas as pd
d = pd.DataFrame([[11, 12, 13, 14],
                  [21, 22, 23, 24],
                  [31, 32, 33, 34]])
d
#    0  1  2  3
# 0 11 12 13 14
# 1 21 22 23 24
# 2 31 32 33 34

d = pd.DataFrame([[11, 12, 13, 14],
                  [21, 22, 23, 24],
                  [31, 32, 33, 34]],
        index=['R1', 'R2', 'R3'],
      columns=['C1', 'C2', 'C3', 'C4'])
d
#    C1 C2 C3 C4
# R1 11 12 13 14
# R2 21 22 23 24
# R3 31 32 33 34
```

データフレーム / 行名と列名

データフレームの index と columns は、あとから変更することができる。データフレームの index と columns 属性に直接新しい名前を代入することで変更できる。

```
import pandas as pd

d = pd.DataFrame([[11, 12, 13, 14],
                  [21, 22, 23, 24],
                  [31, 32, 33, 34]],
        index=['R1', 'R2', 'R3'],
      columns=['C1', 'C2', 'C3', 'C4'])
d
#     C1  C2  C3  C4
# R1  11  12  13  14
# R2  21  22  23  24
# R3  31  32  33  34

d.index = ['x', 'y', 'z']
d.columns = ['h', 'i', 'j', 'k']
d
#    h  i  j  k
# x 11 12 13 14
# y 21 22 23 24
# z 31 32 33 34
```

特定の列または行の名前だけを変更したいとき、Pandas の rename メソッドを使用すると便利である。

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.rename.html

データフレーム要素参照 / iloc

データフレームから要素を取得する方法として、位置番号を利用しても、行名・列名を利用しても取得できる。位置番号で取得する場合は、iloc を使用し、0 から始まる位置番号を与える。

	C1	C2	C3	C4
R1	11	12	13	14
R2	21	22	23	24
R3	31	32	33	34

R1	11	12	13	14
R2	21	22	23	24
R3	31	32	33	34

```
import pandas as pd

d = pd.DataFrame([[11, 12, 13, 14],
                  [21, 22, 23, 24],
                  [31, 32, 33, 34]],
        index=['R1', 'R2', 'R3'],
      columns=['C1', 'C2', 'C3', 'C4'])


d.iloc[0, :]
# C1 11
# C2 12
# C3 13
# C4 14

d.iloc[:, 2]
# R1 13
# R2 23
# R3 33
```

データフレーム要素参照 / iloc

	C1	C2	C3	C4
R1	11	12	13	14
R2	21	22	23	24
R3	31	32	33	34

R1	11	12	13	14
R2	21	22	23	24
R3	31	32	33	34

```
import pandas as pd

d = pd.DataFrame([[11, 12, 13, 14],
                  [21, 22, 23, 24],
                  [31, 32, 33, 34]],
        index=['R1', 'R2', 'R3'],
      columns=['C1', 'C2', 'C3', 'C4'])


d.iloc[0:2, 1:4]
#     C2 C3 C4
# R1 12 13 14
# R2 22 23 24



d.iloc[[0, 2], [1, 3]]
#     C2 C4
# R1 12 14
# R3 32 34
```

NumPy の動作と異なるので、両者を混同しないように。

データフレーム要素参照 / loc

データフレームから要素を行名または列名で取得するときは、loc を使用する。

	C1	C2	C3	C4
R1	11	12	13	14
R2	21	22	23	24
R3	31	32	33	34

R1	11	12	13	14
R2	21	22	23	24
R3	31	32	33	34

```
import pandas as pd

d = pd.DataFrame([[11, 12, 13, 14],
                  [21, 22, 23, 24],
                  [31, 32, 33, 34]],
        index=['R1', 'R2', 'R3'],
      columns=['C1', 'C2', 'C3', 'C4'])


d.loc['R1', :]
# C1 11
# C2 12
# C3 13
# C4 14

d.loc[:, 'C3']
# R1 13
# R2 23
# R3 33
```

データフレーム要素参照 / loc

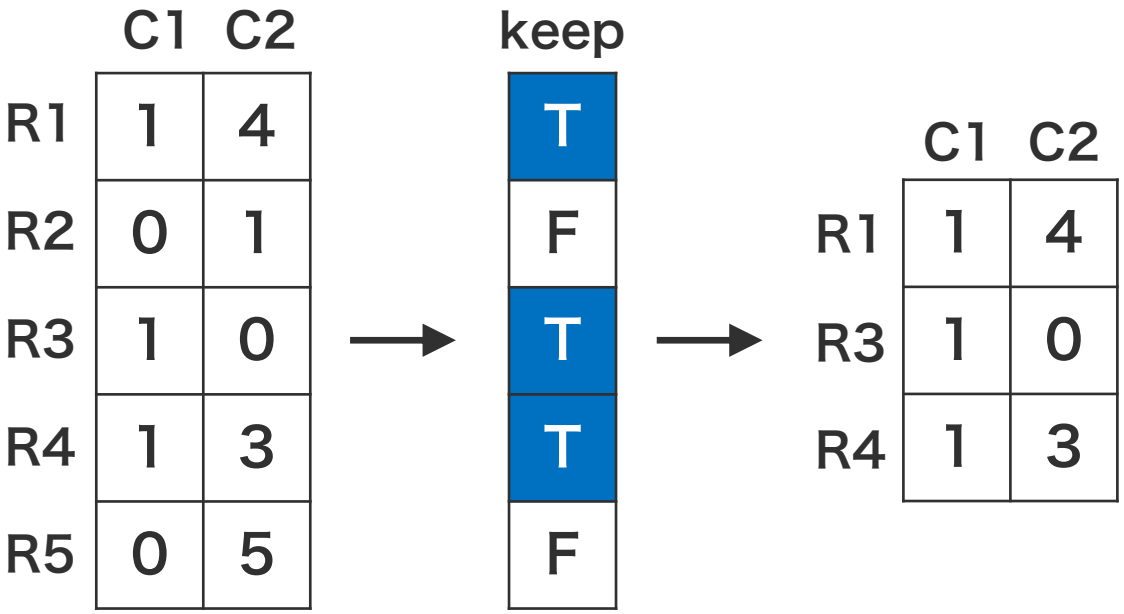
	C1	C2	C3	C4
R1	11	12	13	14
R2	21	22	23	24
R3	31	32	33	34

R1	11	12	13	14
R2	21	22	23	24
R3	31	32	33	34

```
import pandas as pd

d = pd.DataFrame([[11, 12, 13, 14],
                  [21, 22, 23, 24],
                  [31, 32, 33, 34]],
        index=['R1', 'R2', 'R3'],
      columns=['C1', 'C2', 'C3', 'C4'])


d.loc[['R1', 'R2'], 'C2':'C4']
#     C2 C3 C4
# R1 12 13 14
# R2 22 23 24



d.loc[['R1', 'R3'], ['C2', 'C4']]
#     C2 C4
# R1 12 14
# R3 32 34
```

データフレーム要素参照

データフレームもシリーズと同様に、True と False からなるフィルター（ブーリアンベクトル）を使って要素を取得することができる。フィルターを使用する場合は、loc または iloc の両方を使用することができる。



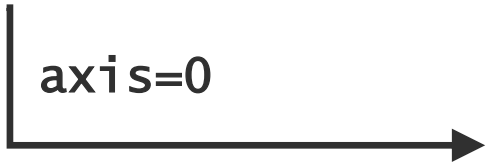

```
import pandas as pd

d = pd.DataFrame([[1, 4],
                  [0, 1],
                  [1, 0],
                  [1, 3],
                  [0, 5]],
   index=['R1', 'R2', 'R3', 'R4', 'R5'],
 columns=['C1', 'C2'])


keep = (d.loc[:, 'C1'] > 0)

d.loc[keep, :]
#     C1 C2
# R1  1  4
# R3  1  0
# R4  1  3
```

データフレーム / pd.concat

pd.concat 関数を使用することで、複数のデータフレームを結合させて 1 つのデータフレームにまとめることができる。pd.concat 関数の axis 引数を通して結合する次元方向を指定できる。

11	12	13	14
21	22	23	24

31	32	33	34
41	42	43	44



11	12	13	14
21	22	23	24
31	32	33	34
41	42	43	44

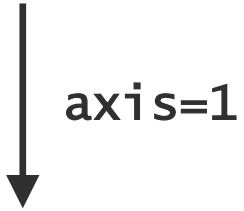
```
import pandas as pd

d1 = pd.DataFrame([[11, 12, 13, 14],
                   [21, 22, 23, 24]])
d2 = pd.DataFrame([[31, 32, 33, 34],
                   [41, 42, 43, 44]])

df = pd.concat([d1, d2])
df
#    0  1  2  3
# 0 11 12 13 14
# 1 21 22 23 24
# 2 31 32 33 34
# 3 41 42 43 44
```

データフレーム / pd.concat

pd.concat 関数を使用することで、複数のデータフレームを結合させて 1 つのデータフレームにまとめることができる。pd.concat 関数の axis 引数を通して結合する次元方向を指定できる。

11	12	13	14
21	22	23	24

31	32	33	34
41	42	43	44



11	12	13	14	31	32	33	34
21	22	23	24	41	42	43	44

```
import pandas as pd

d1 = pd.DataFrame([[11, 12, 13, 14],
                   [21, 22, 23, 24]])
d2 = pd.DataFrame([[31, 32, 33, 34],
                   [41, 42, 43, 44]])

df = pd.concat([d1, d2], axis=1)
df
#    0  1  2  3  0  1  2  3
# 0 11 12 13 14 31 32 33 34
# 1 21 22 23 24 41 42 43 44
```

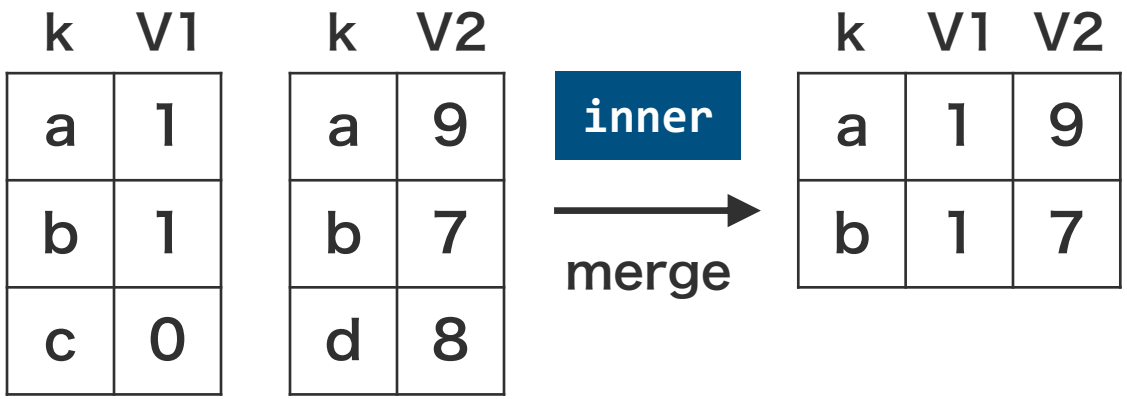
データフレーム / pd.concat

データフレームに index または列名が存在するとき、データフレーム同士が index と列名に基づいて結合される。結合後のデータフレームの行と列の並び順に十分に注意すること。

```
import pandas as pd

d1 = pd.DataFrame([[11, 12, 13, 14],
                   [21, 22, 23, 24]],
        index=['A', 'B'],
        columns=['a', 'b', 'c', 'd'])

d2 = pd.DataFrame([[31, 32, 33, 34],
                   [41, 42, 43, 44]],
        index=['X', 'Y'],
        columns=['a', 'b', 'c', 'e'])


df = pd.concat([d1, d2])
df
#     a  b  c    d    e
# A 11 12 13 14.0  NaN
# B 21 22 23 24.0  NaN
# X 31 32 33  NaN 34.0
# Y 41 42 43  NaN 44.0
```

データフレーム / pd.merge

複数のデータフレームを、特定の列の値に基づいて、マージするすことができる。このとき pd.merge 関数を使用する。



データフレームの結合を行うとき、キーとなる列に重複要素が含まれると、予期しない挙動になる場合があるため、十分に注意すること。

```
import pandas as pd

d1 = pd.DataFrame([['a', 1],
                   ['b', 1],
                   ['c', 0]],
                  columns=['k', 'V1'])

d2 = pd.DataFrame([['a', 9],
                   ['b', 7],
                   ['d', 8]],
                  columns=['k', 'V2'])


d = pd.merge(d1, d2)   # how='inner'
d
#   k V1 V2
# 0 a  1  9
# 1 b  1  7
```

データフレーム / merge

k	V1
a	1
b	1
c	0

k	V2
a	9
b	7
d	8

outer

merge

k	V1	V2
a	1	9
b	1	7
c	0	NaN
d	NaN	8

```
import pandas as pd

d1 = pd.DataFrame([['a', 1],
                   ['b', 1],
                   ['c', 0]],
                  columns=['k', 'V1'])

d2 = pd.DataFrame([['a', 9],
                   ['b', 7],
                   ['d', 8]],
                  columns=['k', 'V2'])


d = pd.merge(d1, d2, how='outer')
d
#    k   V1   V2
# 0  a  1.0  9.0
# 1  b  1.0  7.0
# 2  c  0.0  NaN
# 3  d  NaN  8.0
```

データフレーム / merge

k	V1
a	1
b	1
c	0

k	V2
a	9
b	7
d	8

left

merge

k	V1	V2
a	1	9
b	1	7
c	0	NaN

```
import pandas as pd

d1 = pd.DataFrame([['a', 1],
                   ['b', 1],
                   ['c', 0]],
                  columns=['k', 'V1'])

d2 = pd.DataFrame([['a', 9],
                   ['b', 7],
                   ['d', 8]],
                  columns=['k', 'V2'])


d = pd.merge(d1, d2, how='left')
d
#    k  V1   V2
# 0  a   1  9.0
# 1  b   1  7.0
# 2  c   0  NaN
```

データフレーム / merge

k	V1
a	1
b	1
c	0

k	V2
a	9
b	7
d	8

right

merge

k	V1	V2
a	1	9
b	1	7
d	NaN	8

```
import pandas as pd

d1 = pd.DataFrame([['a', 1],
                   ['b', 1],
                   ['c', 0]],
                  columns=['k', 'V1'])

d2 = pd.DataFrame([['a', 9],
                   ['b', 7],
                   ['d', 8]],
                  columns=['k', 'V2'])


d = pd.merge(d1, d2, how='right')
d
#    k   V1  V2
# 0  a  1.0   9
# 1  b  1.0   7
# 2  d  NaN   8
```

データフレーム / merge

マージ対象のデータフレームの基準列の列名が異なる場合は、pd.merge 関数の left_on および right_on 引数で指定する。

k	V1
a	1
b	1
c	0

f	V2
a	9
b	7
d	8

merge

k	V1	f	V2
a	1	a	9
b	1	b	7
c	0	NaN	NaN
NaN	NaN	d	8

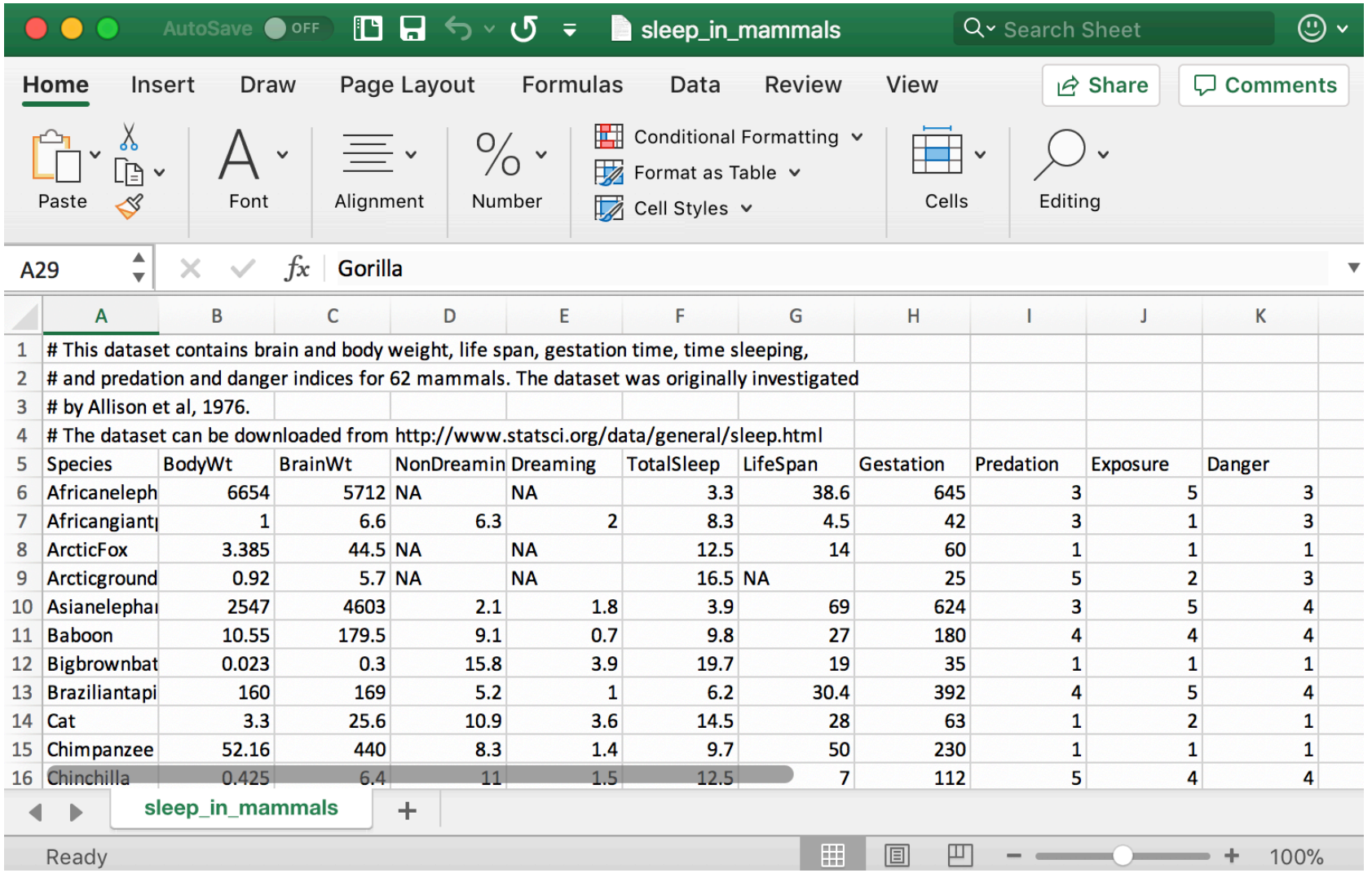
```
import pandas as pd

d1 = pd.DataFrame([['a', 1],
                   ['b', 1],
                   ['c', 0]],
                  columns=['k', 'V1'])

d2 = pd.DataFrame([['a', 9],
                   ['b', 7],
                   ['d', 8]],
                  columns=['f', 'V2'])

d = pd.merge(d1, d2, how='outer',
             left_on='k', right_on='f')
d
#      k   V1    f   V2
# 0    a  1.0    a  9.0
# 1    b  1.0    b  7.0
# 2    c  0.0  NaN  NaN
# 3  NaN  NaN    d  8.0
```

Pandas

- シリーズ
- データフレーム
- **表データ処理**

表データ

生物学で取り扱うデータは、一般的に、1 サンプル 1 行で記載されている。また、各行は複数の項目からなり、サンプルの属性が記載されいてる。このようなデータは、一般的にタブ区切りファイル（TSV）またはカンマ区切りファイル（CSV）のテキストファイルで保存される。



表データ

コメント

データ

```
# This dataset contains brain and body weight, life span, gestation time,
# and predation and danger indices for 62 mammals. The dataset was origin
# by Allison et al, 1976.
# The dataset can be downloaded from http://www.statsci.org/data/general/
Species BodyWt  BrainWt NonDreaming     Dreaming         TotalSleep       L
Africanelephant 6654    5712    NA      NA      3.3      38.6    645      3
Africangiantpouchedrat  1       6.6     6.3     2        8.3     4.5      4
ArcticFox       3.385   44.5    NA      NA      12.5     14      60       1
Arcticgroundsquirrel    0.92    5.7     NA      NA       16.5    NA       2
Asianelephant   2547    4603    2.1     1.8     3.9      69      624      3
Baboon  10.55   179.5   9.1     0.7     9.8      27      180     4        4
Bigbrownbat     0.023   0.3     15.8    3.9     19.7     19      35       1
Braziliantapir  160     169     5.2     1       6.2      30.4    392      4
Cat     3.3     25.6    10.9    3.6     14.5     28      63      1        2
Chimpanzee      52.16   440     8.3     1.4     9.7      50      230      1
Chinchilla      0.425   6.4     11      1.5     12.5     7       112      5
Cow     465     423     3.2     0.7     3.9      30      281     5        5
```

https://aabbdd.jp/notes/data/sleep_in_mammals.txt

表データ

属性（特徴量）

ヘッダー →

サンプル

Species	BodyWt	BrainWt	NonDreaming		Dreaming		TotalSleep		L
Africanelephant	6654	5712	NA		NA	3.3	38.6	645	3
Africangiantpouchedrat	1	欠損値	6.6		6.3	2	8.3	4.5	4
ArcticFox	3.385	44.5	NA		NA	12.5	14	60	1
Arcticgroundsquirrel		0.92	5.7		NA	NA	16.5	NA	2
Asianelephant	2547	4603	2.1		1.8	3.9	69	624	3
Baboon	10.55	179.5	9.1		0.7	9.8	27	180	4
Bigbrownbat		0.023	0.3		15.8	3.9	19.7	19	1
Braziliantapir		160	169		5.2	1	6.2	30.4	392
Cat	3.3	25.6	10.9		3.6	14.5	28	63	1
Chimpanzee		52.16	440		8.3	1.4	9.7	50	230
Chinchilla		0.425	6.4		11	1.5	12.5	7	112
Cow	465	423	3.2		0.7	3.9	30	281	5

https://aabbdd.jp/notes/data/sleep_in_mammals.txt

表データ読み込み

Pandas の read_table 関数を使うことで、CSV または TSV データを読み込むことができる。

```
import pandas as pd

f = 'sleep_in_mammals.txt'

d = pd.read_csv(f)
```

```
pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader.read()

pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._read_low_memory()

pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._read_rows()

pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._tokenize_rows()

pandas/_libs/parsers.pyx in pandas._libs.parsers.raise_parser_error()

ParserError: Error tokenizing data. C error: Expected 1 fields in line 5, saw 11
```

sleep_in_mammals.txt ファイルは、read_table 関数が想定しているフォーマットに従っていないため、エラーになった。

https://aabbdd.jp/notes/data/sleep_in_mammals.txt

表データ読み込み

Pandas の read_table 関数を使うことで、CSV または TSV データを読み込むことができる。データを正しく読み込むには、read_table 関数に、コメント行を明示し、ヘッダー行の有無、区切り文字の種類を正しく指定する必要がある。

```
import pandas as pd

f = 'sleep_in_mammals.txt'

d = pd.read_csv(f,
                comment='#',
                header=0,
                sep='\t')
```

バックスラッシュは特別な意味を持つ文字である。バックスラッシュの後に続く1文字は、特別な意味を持つ。例えば 't' は英文字の t を表すが、'\t' はタブを表す。

https://aabbdd.jp/notes/data/sleep_in_mammals.txt

表データ読み込み

```
import pandas as pd

f = 'sleep_in_mammals.txt'

d = pd.read_csv(f, comment='#', header=0, sep='\t')

d.shape
# (62, 11)

d.head()
#                  Species    BodyWt  BrainWt  ...  Predation  Exposure  Danger
# 0         Africanelephant  6654.000   5712.0  ...          3         5       3
# 1  Africangiantpouchedrat     1.000      6.6  ...          3         1       3
# 2                ArcticFox     3.385     44.5  ...          1         1       1
# 3    Arcticgroundsquirrel     0.920      5.7  ...          5         2       3
# 4           Asianelephant  2547.000   4603.0  ...          3         5       4
# [5 rows x 11 columns]
```

https://aabbdd.jp/notes/data/sleep_in_mammals.txt

表データ読み込み

```
import pandas as pd

f = 'sleep_in_mammals.txt'

d = pd.read_csv(f, comment='#', header=0, sep='\t', index_col=0)

d.shape
# (62, 10)

d.head()
#                          BodyWt  BrainWt  ...  Exposure  Danger
# Species                                   ...
# Africanelephant        6654.000   5712.0  ...         5       3
# Africangiantpouchedrat    1.000      6.6  ...         1       3
# ArcticFox                 3.385     44.5  ...         1       1
# Arcticgroundsquirrel      0.920      5.7  ...         2       3
# Asianelephant          2547.000   4603.0  ...         5       4
# [5 rows x 10 columns]
```

特定の列をデータの一部ではなくて、行名として取り組むこともできる。

https://aabbdd.jp/notes/data/sleep_in_mammals.txt

表データ / 行名と列名

```
import pandas as pd

f = 'sleep_in_mammals.txt'

d = pd.read_csv(f, comment='#', header=0, sep='\t', index_col=0)

d.index
# Index(['Africanelephant', 'Africangiantpouchedrat', 'ArcticFox',
#        'Arcticgroundsquirrel', 'Asianelephant', 'Baboon', 'Bigbrownbat',
#        'Braziliantapir', 'Cat', 'Chimpanzee', 'Chinchilla', 'Cow',
#        ...
#        'Treeshrew', 'Vervet', 'Wateropossum', 'Yellow-belliedmarmot'],
#       dtype='object', name='Species')

d.columns
# Index(['BodyWt', 'BrainWt', 'NonDreaming', 'Dreaming', 'TotalSleep',
#        'LifeSpan', 'Gestation', 'Predation', 'Exposure', 'Danger'],
#       dtype='object')
```

https://aabbdd.jp/notes/data/sleep_in_mammals.txt

表データ / 要素の取得

```
import pandas as pd

f = 'sleep_in_mammals.txt'

d = pd.read_csv(f, comment='#', header=0, sep='\t', index_col=0)

d.iloc[0:3, 0:5]
#                          BodyWt  BrainWt  NonDreaming  Dreaming  TotalSleep
# Species
# Africanelephant         6654.000   5712.0          NaN       NaN         3.3
# Africangiantpouchedrat     1.000      6.6          6.3       2.0         8.3
# ArcticFox                  3.385     44.5          NaN       NaN        12.5

d.iloc[0:2, :]
#                          BodyWt  BrainWt  NonDreaming  ... Exposure  Danger
# Species                                                ...
# Africanelephant         6654.000   5712.0          NaN  ...        5       3
# Africangiantpouchedrat     1.000      6.6          6.3  ...        1       3
```

https://aabbdd.jp/notes/data/sleep_in_mammals.txt

表データ / 要素の取得

```
import pandas as pd

f = 'sleep_in_mammals.txt'

d = pd.read_csv(f, comment='#', header=0, sep='\t', index_col=0)

species = ['Cat', 'Rat', 'Cow', 'Pig']
features = ['BodyWt', 'BrainWt', 'TotalSleep', 'LifeSpan']

d.loc[species, features]
#          BodyWt  BrainWt  TotalSleep  LifeSpan
# Species
# Cat        3.30     25.6        14.5      28.0
# Rat        0.28      1.9        13.2       4.7
# Cow      465.00    423.0         3.9      30.0
# Pig      192.00    180.0         8.4      27.0
```

https://aabbdd.jp/notes/data/sleep_in_mammals.txt

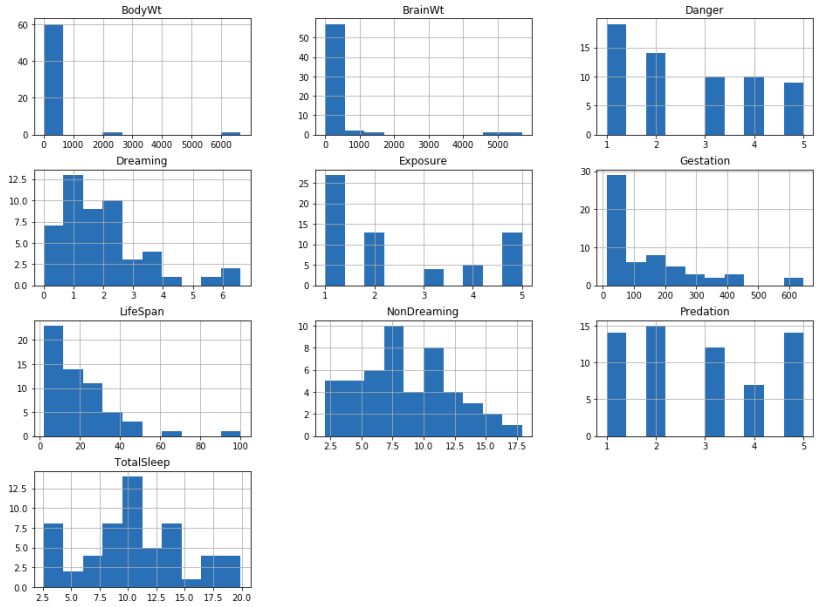
表データ読み込み / データ要約

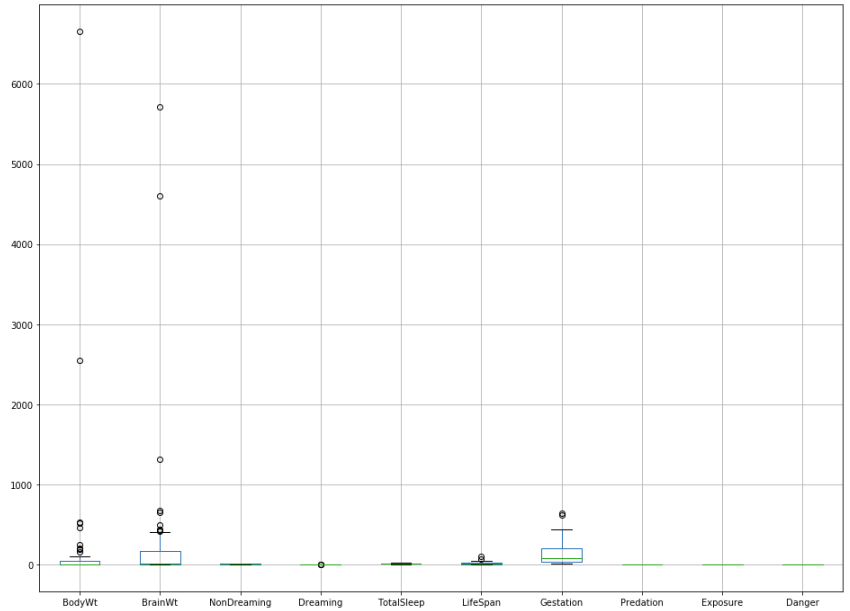
```
import pandas as pd

f = 'sleep_in_mammals.txt'

d = pd.read_csv(f, comment='#', header=0, sep='\t', index_col=0)

d.describe()
#              BodyWt      BrainWt  NonDreaming  ...  Predation   Exposure      Danger
# count    62.000000    62.000000    48.000000  ...  62.000000  62.000000  62.000000
# mean    198.789984   283.134194     8.672917  ...   2.870968   2.419355   2.612903
# std     899.158011   930.278942     3.666452  ...   1.476414   1.604792   1.441252
# min       0.005000     0.140000     2.100000  ...   1.000000   1.000000   1.000000
# 25%       0.600000     4.250000     6.250000  ...   2.000000   1.000000   1.000000
# 50%       3.342500    17.250000     8.350000  ...   3.000000   2.000000   2.000000
# 75%      48.202500   166.000000    11.000000  ...   4.000000   4.000000   4.000000
# max    6654.000000  5712.000000    17.900000  ...   5.000000   5.000000   5.000000
# [8 rows x 10 columns]
```

https://aabbdd.jp/notes/data/sleep_in_mammals.txt

表データ読み込み / データ可視化

```
import pandas as pd

f = 'sleep_in_mammals.txt'

d = pd.read_csv(f, comment='#', header=0, sep='\t', index_col=0)

d.hist()
d.boxplot()
```





※ 一部の属性の値の範囲が大きいので、対数化してからグラフに描くと全体の傾向を掴めやすくなる。

※ matplotlib でグラフを描いた方が一般的であるので、ここでは Pandas の視覚化機能を取り上げない。

https://aabbdd.jp/notes/data/sleep_in_mammals.txt

問題 P2-1

diversity_galapagos.txt を Pandas で読み込み、面積（Area）が最も大きい島の名前とその面積を答えよ。

```
import pandas as pd

f = 'diversity_galapagos.txt'
d = pd.read_csv(f, comment='#', header=0, sep='\t', index_col=0)
```

https://aabbdd.jp/notes/data/diversity_galapagos.txt

ヒント

シリーズ s の最大値は max(s) で求めることができる。

問題 P2-2

diversity_galapagos.txt を Pandas で読み込み、 各島における面積あたりの種数を求めよ。

```
import pandas as pd

f = 'diversity_galapagos.txt'
d = pd.read_csv(f, comment='#', header=0, sep='\t', index_col=0)
```

https://aabbdd.jp/notes/data/diversity_galapagos.txt

ファイル書き出し

Pandas では、データを CSV または TSV ファイルに書き出すとき、to_csv 関数（メソッド）を使う。この際に、区切り文字、行名の有無、列名の有無を指定することができる。また、欠損値を特定の文字に変換することもできる。

```
import pandas as pd
f = 'sleep_in_mammals.txt'

d = pd.read_csv(f, comment='#', header=0, sep='\t',
                index_col=0)

d.to_csv('o.txt', sep=',', header=False, index=False)
```

```
6654.0,5712.0,,,3.3,38.6,645.0,3,5,3
1.0,6.6,6.3,2.0,8.3,4.5,42.0,3,1,3
3.385,44.5,,,12.5,14.0,60.0,1,1,1
0.92,5.7,,,16.5,,25.0,5,2,3
2547.0,4603.0,2.1,1.8,3.9,69.0,624.0,3,5,4
10.55,179.5,9.1,0.7,9.8,27.0,180.0,4,4,4
0.023,0.3,15.8,3.9,19.7,19.0,35.0,1,1,1
160.0,169.0,5.2,1.0,6.2,30.4,392.0,4,5,4
3.3,25.6,10.9,3.6,14.5,28.0,63.0,1,2,1
52.16,440.0,8.3,1.4,9.7,50.0,230.0,1,1,1
```

ファイル書き出し

Pandas では、データを CSV または TSV ファイルに書き出すとき、to_csv 関数（メソッド）を使う。この際に、区切り文字、行名の有無、列名の有無を指定することができる。また、欠損値を特定の文字に変換することもできる。

```
import pandas as pd
f = 'sleep_in_mammals.txt'

d = pd.read_csv(f, comment='#', header=0, sep='\t',
                   index_col=0)

d.to_csv('o.txt', sep=',', header=True, index=True)
```

```
Species,BodyWt,BrainWt,NonDreaming,Dreaming,TotalSleep,LifeSpan,Gestation,Predation,Exposure
Africanelephant,6654.0,5712.0,,,3.3,38.6,645.0,3,5,3
Africangiantpouchedrat,1.0,6.6,6.3,2.0,8.3,4.5,42.0,3,1,3
ArcticFox,3.385,44.5,,,12.5,14.0,60.0,1,1,1
Arcticgroundsquirrel,0.92,5.7,,,16.5,,25.0,5,2,3
Asianelephant,2547.0,4603.0,2.1,1.8,3.9,69.0,624.0,3,5,4
Baboon,10.55,179.5,9.1,0.7,9.8,27.0,180.0,4,4,4
Bigbrownbat,0.023,0.3,15.8,3.9,19.7,19.0,35.0,1,1,1
Braziliantapir,160.0,169.0,5.2,1.0,6.2,30.4,392.0,4,5,4
Cat,3.3,25.6,10.9,3.6,14.5,28.0,63.0,1,2,1
```

ファイル書き出し

Pandas では、データを CSV または TSV ファイルに書き出すとき、to_csv 関数（メソッド）を使う。この際に、区切り文字、行名の有無、列名の有無を指定することができる。また、欠損値を特定の文字に変換することもできる。

```
import pandas as pd
f = 'sleep_in_mammals.txt'

d = pd.read_csv(f, comment='#', header=0, sep='\t',
                index_col=0)

d.to_csv('o.txt', sep='\t', header=True, index=True)
```

```
Species BodyWt  BrainWt NonDreaming     Dreaming        TotalSleep      LifeSpan        Gestati
Africanelephant 6654.0  5712.0                  3.3     38.6    645.0   3       5       3
Africangiantpouchedrat  1.0     6.6     6.3     2.0     8.3     4.5     42.0    3       1
ArcticFox       3.385   44.5                    12.5    14.0    60.0    1       1       1
Arcticgroundsquirrel    0.92    5.7                     16.5            25.0    5       2
Asianelephant   2547.0  4603.0  2.1     1.8     3.9     69.0    624.0   3       5       4
Baboon  10.55   179.5   9.1     0.7     9.8     27.0    180.0   4       4       4
Bigbrownbat     0.023   0.3     15.8    3.9     19.7    19.0    35.0    1       1       1
Braziliantapir  160.0   169.0   5.2     1.0     6.2     30.4    392.0   4       5       4
Cat     3.3     25.6    10.9    3.6     14.5    28.0    63.0    1       2       1
```

ファイル書き出し

Pandas では、データを CSV または TSV ファイルに書き出すとき、to_csv 関数（メソッド）を使う。この際に、区切り文字、行名の有無、列名の有無を指定することができる。また、欠損値を特定の文字に変換することもできる。

```
import pandas as pd
f = 'sleep_in_mammals.txt'

d = pd.read_csv(f, comment='#', header=0, sep='\t',
                index_col=0)

d.to_csv('o.txt', sep='\t', header=True, index=True)
         na_rep= 'NA')
```

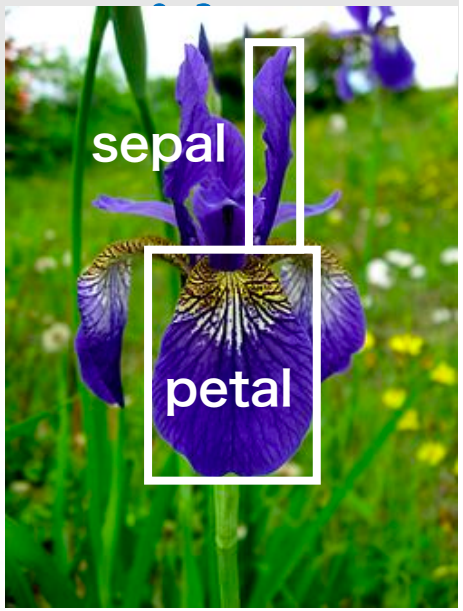
```
Species BodyWt  BrainWt NonDreaming     Dreaming        TotalSleep      LifeSpan        Gestati
Africanelephant 6654.0  5712.0  NA      NA      3.3     38.6    645.0   3       5       3
Africangiantpouchedrat  1.0     6.6     6.3     2.0     8.3     4.5     42.0    3       1
ArcticFox       3.385   44.5    NA      NA      12.5    14.0    60.0    1       1       1
Arcticgroundsquirrel    0.92    5.7     NA      NA      16.5    NA      25.0    5       2
Asianelephant   2547.0  4603.0  2.1     1.8     3.9     69.0    624.0   3       5       4
Baboon  10.55   179.5   9.1     0.7     9.8     27.0    180.0   4       4       4
Bigbrownbat     0.023   0.3     15.8    3.9     19.7    19.0    35.0    1       1       1
Braziliantapir  160.0   169.0   5.2     1.0     6.2     30.4    392.0   4       5       4
Cat     3.3     25.6    10.9    3.6     14.5    28.0    63.0    1       2       1
```

空白がNAに置換される

グループ演算

Pandas のデータフレームでは、ある列の値に基づいて、全データをいくつかのサブセット分け、それぞれのサブセットに対して平均や分散などを計算する機能が提供されている。

```
import pandas as pd
f = 'iris.txt'
d = pd.read_csv(f, header=0, sep='\t')

d.head()
#    ID  Sepal.Length  Sepal.Width  Petal.Length  Petal.Width Species
# 0   1           5.1          3.5           1.4          0.2  setosa
# 1   2           4.9          3.0           1.4          0.2  setosa
# 2   3           4.7          3.2           1.3          0.2  setosa
# 3   4           4.6          3.1           1.5
# 4   5           5.0          3.6           1.4
```

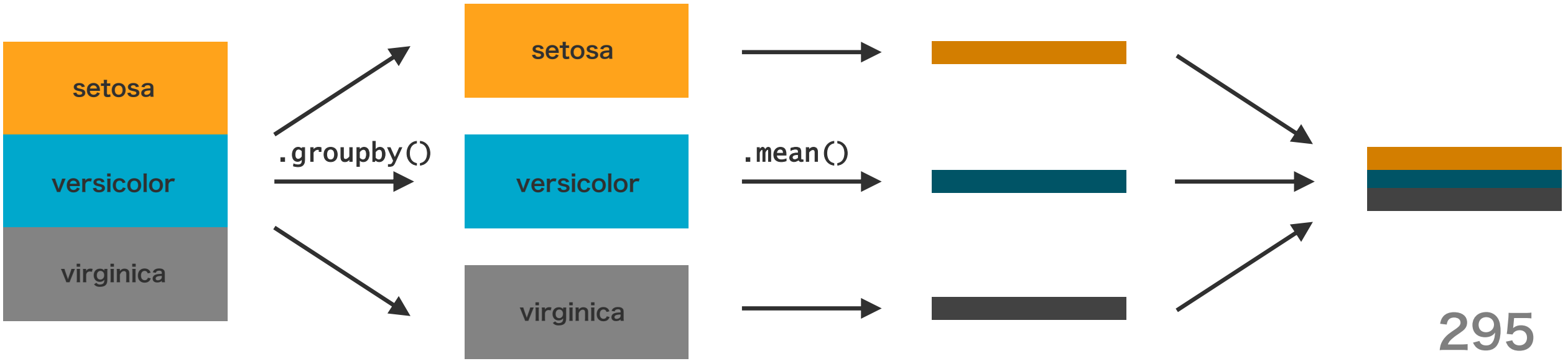


iris データセットは、3種のアヤメ(setosa・versicolor・virginica)に対して、萼 sepal の長さ length と幅 width、花弁 petal の長さ length と幅 width を測定したデータである。各種には 50 個体のデータ含まれ、3種で計150個体のデータが含まれている。

https://aabbdd.jp/notes/data/iris.txt

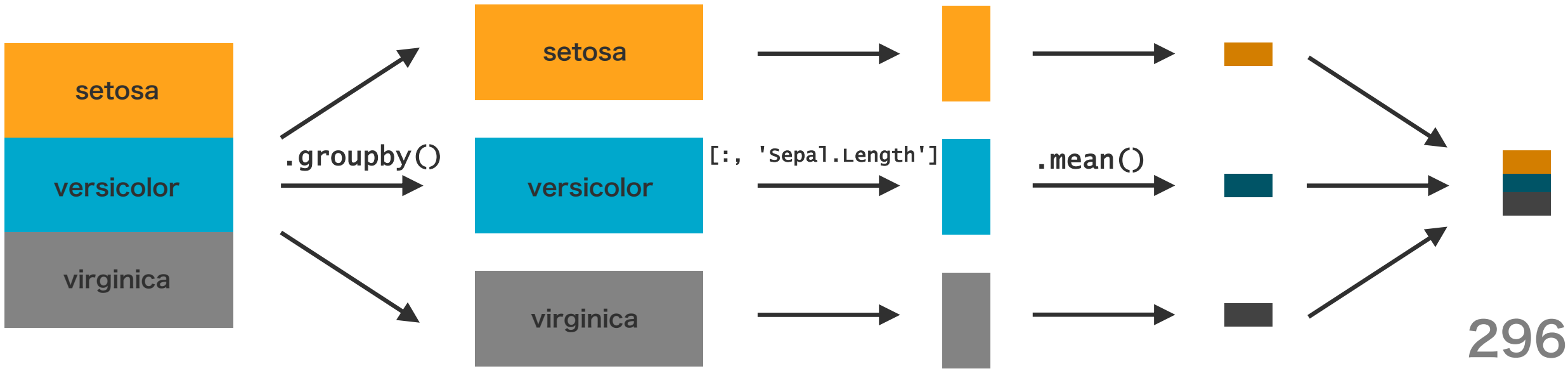
グループ演算 / groupby

```
import pandas as pd
f = 'iris.txt'
d = pd.read_csv(f, header=0, sep='\t')

d.groupby('Species').mean()
#                ID  Sepal.Length  Sepal.Width  Petal.Length  Petal.Width
# Species
# setosa       25.5         5.006        3.428         1.462        0.246
# versicolor   75.5         5.936        2.770         4.260        1.326
# virginica   125.5         6.588        2.974         5.552        2.026
```



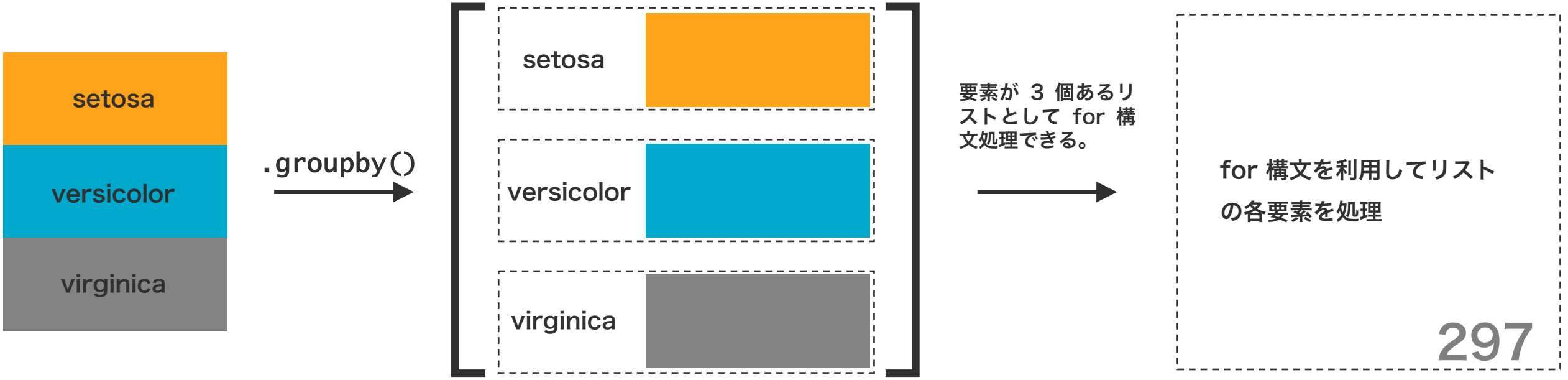
グループ演算 / groupby

```
import pandas as pd
f = 'iris.txt'
d = pd.read_csv(f, header=0, sep='\t')

d.groupby('Species').loc[:, 'Sepal.Length'].mean()
# Species
# setosa          5.006
# versicolor      5.936
# virginica       6.588
# Name: Sepal.Length, dtype: float64
```



グループ演算 / groupby

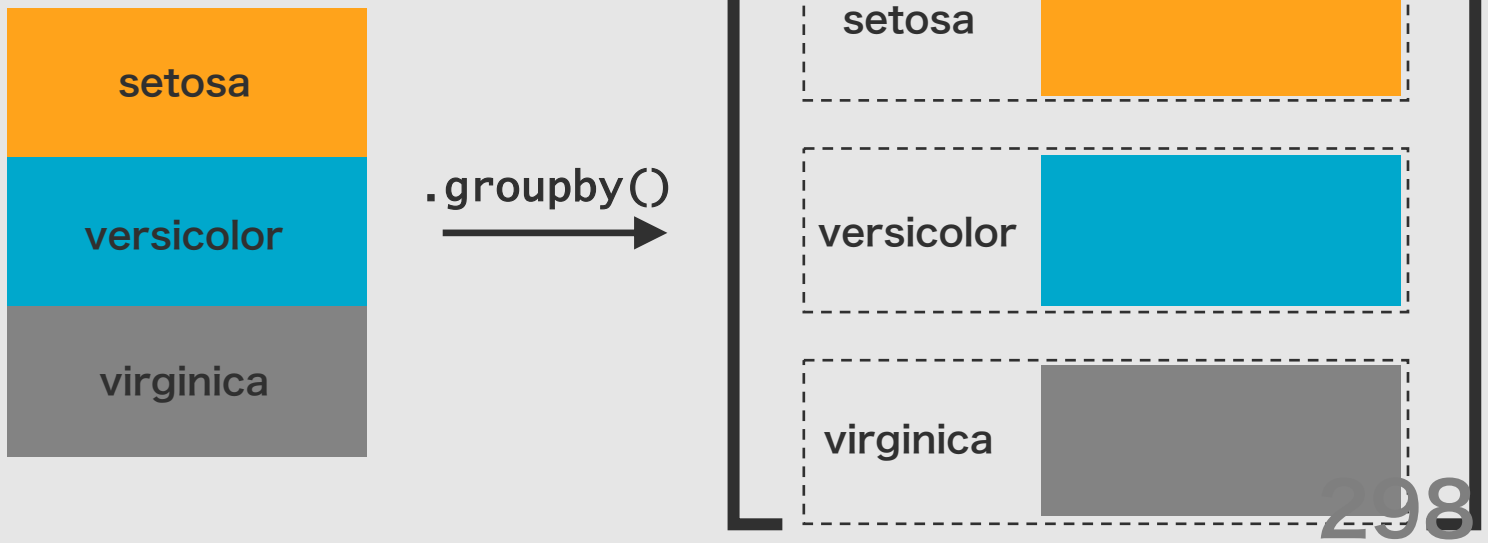
```
import pandas as pd
f = 'iris.txt'
d = pd.read_csv(f, header=0, sep='\t')

for gname, subset in d.groupby('Species'):
    print(gname)
    print(subset.head())
```



グループ演算 / groupby

```
import pandas as pd
f = 'iris.txt'
d = pd.read_csv(f, header=0, sep='\t')

for gname, subset in d.groupby('Species'):
    M = subset.iloc[:, 1:5].max(axis=0)
    m = subset.iloc[:, 1:5].min(axis=0)
    print(gname)
    print(M - m)

# setosa
# Sepal.Length 1.5
# Sepal.Width 2.1
# Petal.Length 0.9
# Petal.Width 0.5
# versicolor
# Sepal.Length 2.1
# Sepal.Width 1.4
# ...
# ...
```

2 列目から 6 列目の部分列を抽出し、各列の最大値を計算する。



グループ演算 / apply

```
import pandas as pd
f = 'iris.txt'
d = pd.read_csv(f, header=0, sep='\t')


def calc_diff(df):
    M = df.iloc[:, 1:5].max(axis=0)
    m = df.iloc[:, 1:5].min(axis=0)
    return M - m


d.groupby('Species').apply(calc_diff)
#             Sepal.Length  Sepal.Width  Petal.Length  Petal.Width
# Species
# setosa               1.5          2.1           0.9          0.5
# versicolor           2.1          1.4           2.1          0.8
# virginica            3.0          1.6           2.4          1.1
```

for 構文を利用したグループ演算を関数化することで、高速な apply 関数を利用できるようにある。

問題 P3-1

iris.txtを読み込み、各種の花弁 petal の長さ length と幅 width の比率の平均値を求めよ。

```
import pandas as pd

f = 'iris.txt'
d = pd.read_csv(f, header=0, sep='\t')
```

https://aabbdd.jp/notes/data/iris.txt

問題 P3-2

rice.txt には、野生型イネ wt と実験処理後のイネ ANU842 を 3 つの環境（F10, NH4Cl, NH4NO3）で栽培し、その収穫後のバイオマスの乾燥重量（shoot_dry_mass, root_dry_mass）を測定したデータが記録されている。F10 栽培環境で栽培したイネ wt の shoot_dry_mass と root_dry_mass の平均値をそれぞれ求めよ。

```
import pandas as pd

f = 'rice.txt'
d = pd.read_csv(f, header=0, sep='\t')
d.head()
#    replicate  block  root_dry_mass  shoot_dry_mass  trt  fert  variety
# 0          1      1             56             132  F10   F10       wt
# 1          2      1             66             120  F10   F10       wt
# 2          3      1             40             108  F10   F10       wt
# 3          4      1             43             134  F10   F10       wt
# 4          5      1             55             119  F10   F10       wt
```

https://aabbdd.jp/notes/data/rice.txt

可視化

- matplotlib 基本
- 基本グラフ
- プロット領域の分割

pandas

$y_i t = \beta' x_{it} + \mu_i + \epsilon_{it}$

Pandas はデータ分析用のパッケージであり、可視化機能も実装されている。シリーズやデータフレーム等を手軽に可視化できる。

ggplot

R / ggplot2 とほぼ同じような使い方で、ほぼ同じような仕上がりとなる。The grammar of graphics と呼ばれる文法に従って記述する。

matplotlib

matplotlib は初期から存在する可視化パッケージである。ユーザーが多いため、情報量も多い。細かな調整が効き、複雑なグラフも描ける。

plotly

ウェブベースのインタラクティブなグラフを作成できる。解析結果をリアルタイムに表示させたいときに利用する。

seaborn

matplotlib をベースとしている。matplotlib を補完する位置付けである。ペアプロットやヒートマップなどの応用グラフも関数一つで描ける。

Bokeh

ウェブベースのインタラクティブなグラフを作成できる。The grammar of graphics と呼ばれる文法に従って記述する。

matplotlib 基本

- object-oriented interface
- state-based interface

matplotlib / Application Programming Interface

matplotlib には 2 種類の可視化 API が用意されている。1 つはオブジェクト指向型プログラミング言語を意識した object-oriented interface である。もう 1 つは、MATLAB の使い方を踏襲した state-based interface である。

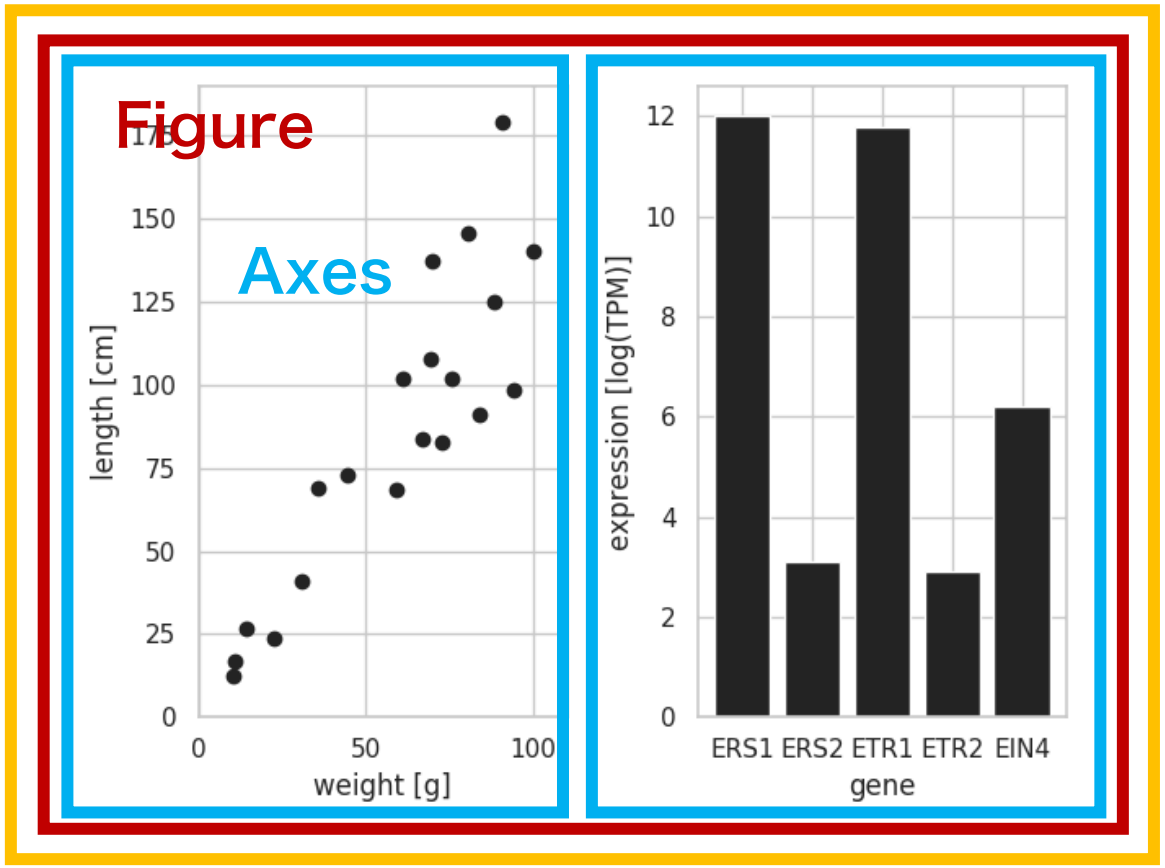
object-oriented interface

プロット領域をいくつかのクラス（サブ領域）に分割し、そのクラスで定義されたメソッド（関数）を使用して、グラフを作成していくインタフェースである。

state-based interface

クラスを意識せずに、あらかじめ用意された関数を使用してグラフを描いていくインタフェースである。

Pyplot



matplotlib API

object-oriented interface

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
y = np.array([1.2, 2.5, 3.4, 3.3, 2.8])

fig = plt.figure()
ax = fig.add_subplot()


ax.plot(x, y)

fig.show()
```

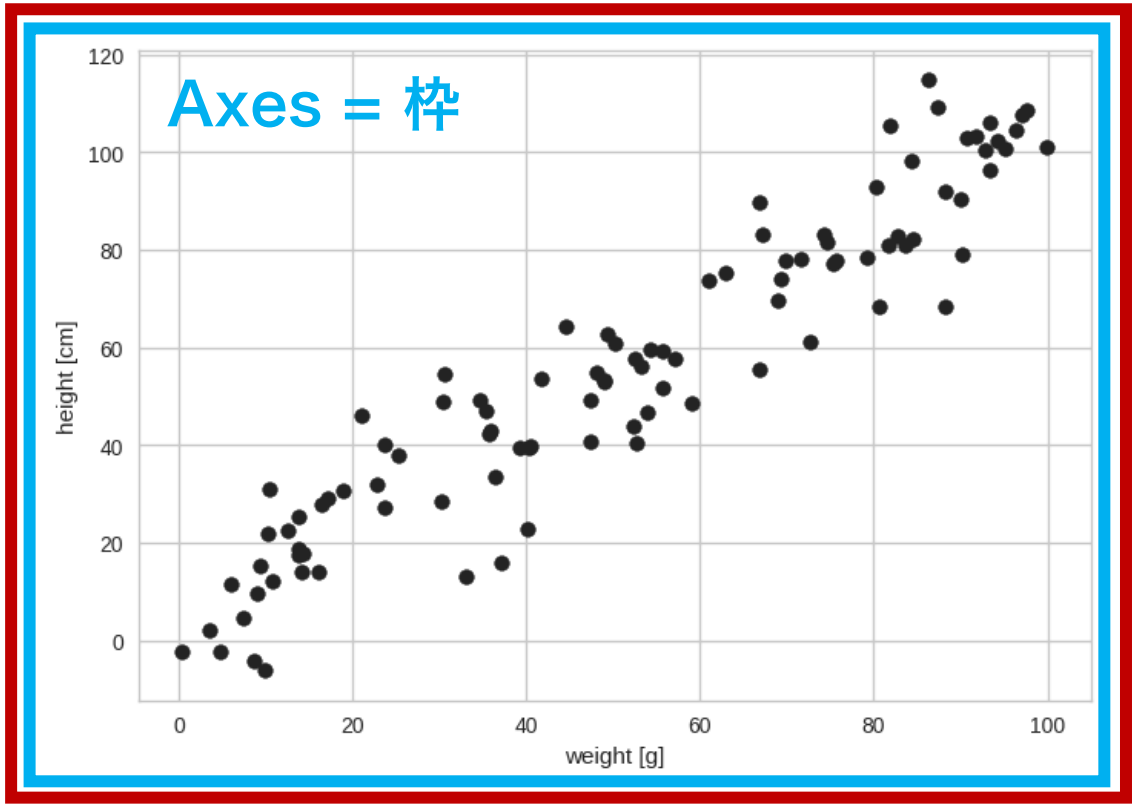
state-based interface

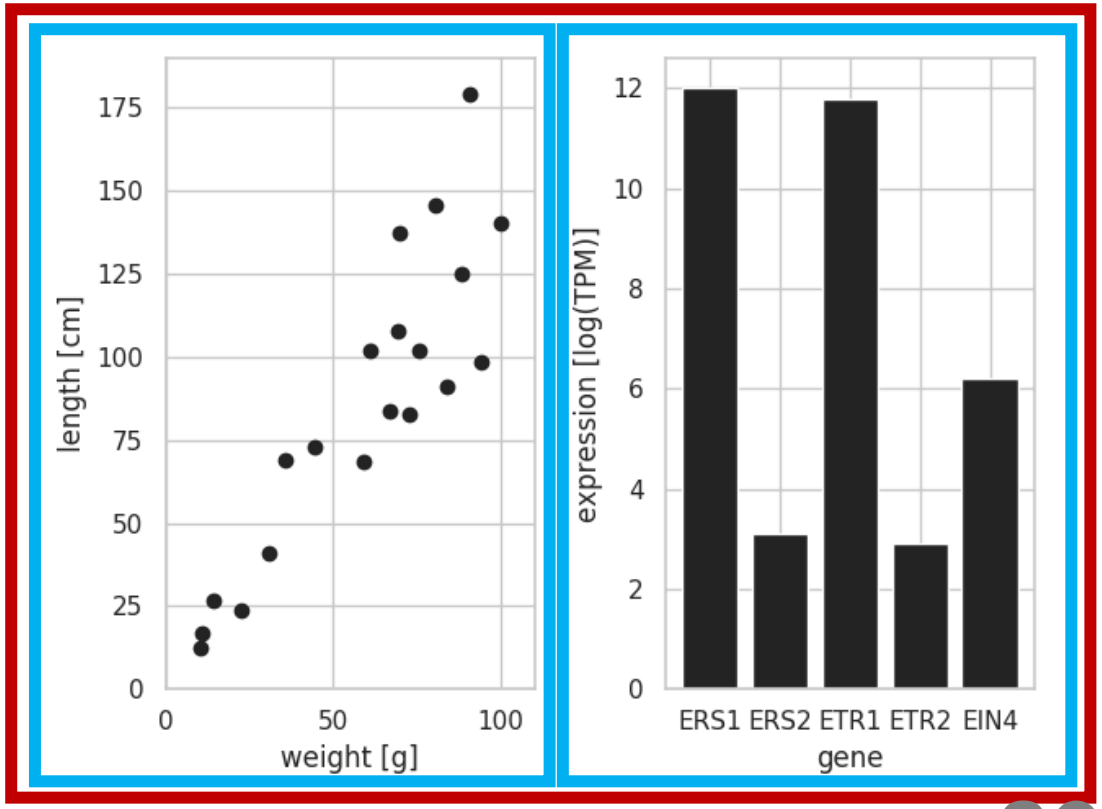
```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
y = np.array([1.2, 2.5, 3.4, 3.3, 2.8])




plt.plot(x, y)

plt.show()
```

object-oriented interface

Pyplot = 落書き帳

Figure = ページ

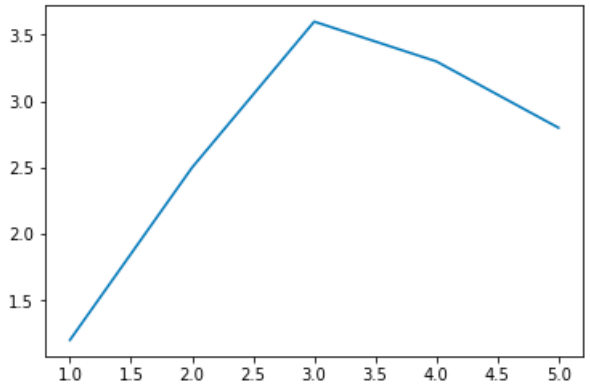




object-oriented interface

matplotlib を利用してグラフを作成するには pyplot モジュール中のメソッドを使用する。

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
y = np.array([1.2, 2.5, 3.4, 3.3, 2.8])

fig = plt.figure()
ax = fig.add_subplot()
ax.plot(x, y)
fig.show()
```



object-oriented interface

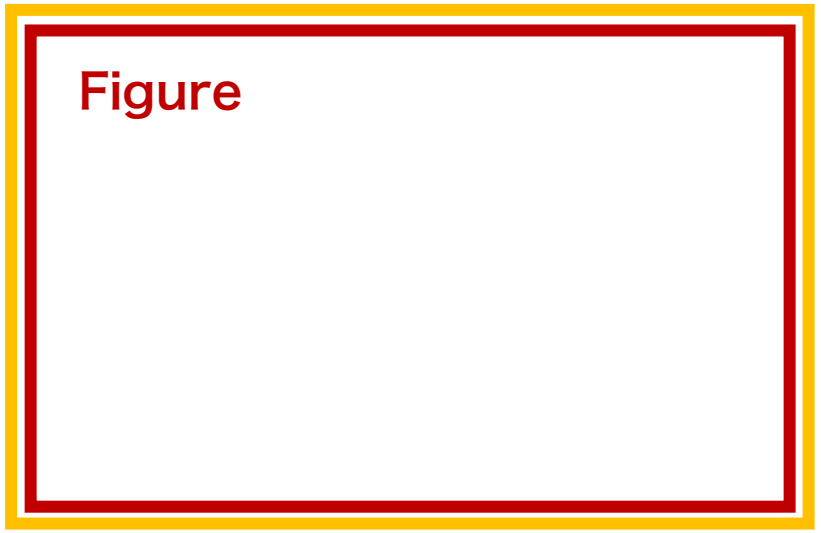
1. matplotlib.pyplot モジュールの機能を呼び出す。pyplot 描画デバイスが用意される。

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
y = np.array([1.2, 2.5, 3.4, 3.3, 2.8])

fig = plt.figure()
ax = fig.add_subplot()
ax.plot(x, y)
fig.show()
```

object-oriented interface

1. matplotlib.pyplot モジュールの機能を呼び出す。pyplot 描画デバイスが用意される。
2. Figure クラスのオブジェクト（領域）を用意する。

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
y = np.array([1.2, 2.5, 3.4, 3.3, 2.8])

fig = plt.figure()
ax = fig.add_subplot()
ax.plot(x, y)
fig.show()
```



object-oriented interface

1. matplotlib.pyplot モジュールの機能を呼び出す。pyplot 描画デバイスが用意される。
2. Figure クラスのオブジェクト（領域）を用意する。
3. Figure 領域の中に、さらに Axes クラスのオブジェクト（領域）を作成する。

▶

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
y = np.array([1.2, 2.5, 3.4, 3.3, 2.8])

fig = plt.figure()
ax = fig.add_subplot()
ax.plot(x, y)
fig.show()
```

object-oriented interface

1. matplotlib.pyplot モジュールの機能を呼び出す。pyplot 描画デバイスが用意される。
2. Figure クラスのオブジェクト（領域）を用意する。
3. Figure 領域の中に、さらに Axes クラスのオブジェクト（領域）を作成する。
4. Axes 領域に線グラフを描く。

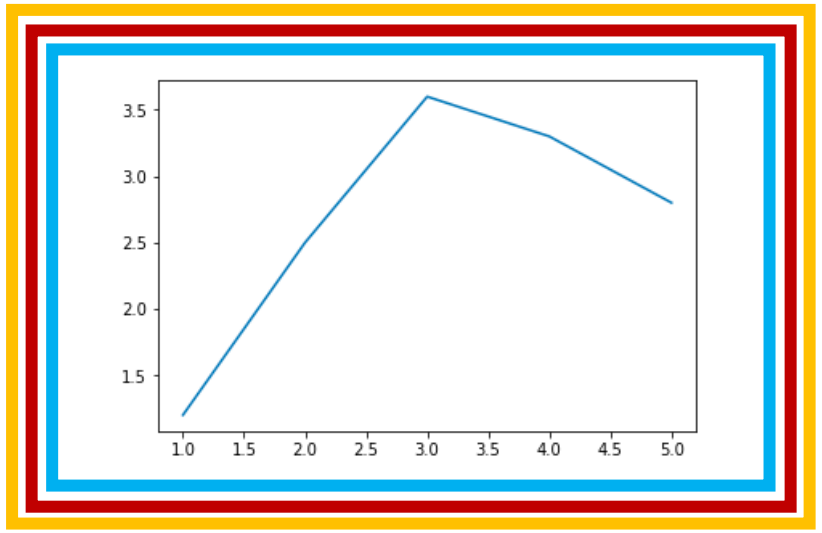
```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
y = np.array([1.2, 2.5, 3.4, 3.3, 2.8])

fig = plt.figure()
ax = fig.add_subplot()
ax.plot(x, y)
fig.show()
```

object-oriented interface

1. matplotlib.pyplot モジュールの機能を呼び出す。pyplot 描画デバイスが用意される。
2. Figure クラスのオブジェクト（領域）を用意する。
3. Figure 領域の中に、さらに Axes クラスのオブジェクト（領域）を作成する。
4. Axes 領域に線グラフを描く。
5. グラフを表示する。

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
y = np.array([1.2, 2.5, 3.4, 3.3, 2.8])

fig = plt.figure()
ax = fig.add_subplot()
ax.plot(x, y)
fig.show()
```

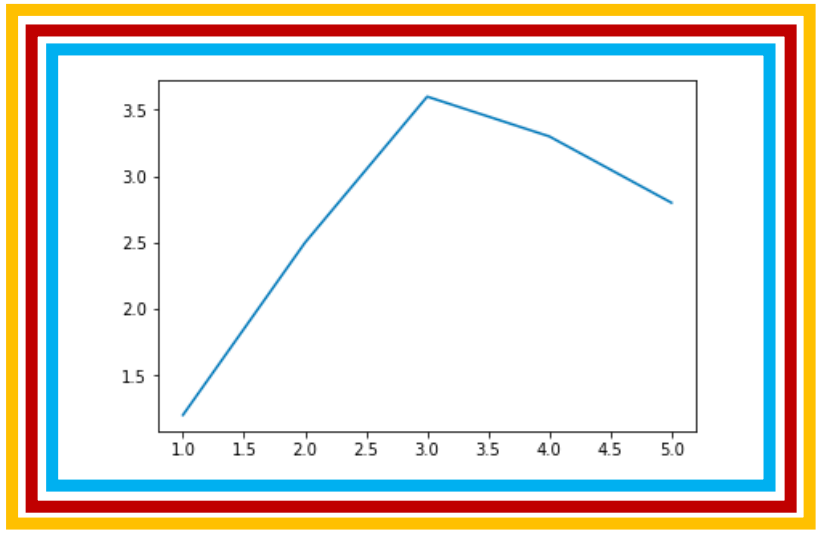
▶



object-oriented interface / グラフ保存

グラフをファイルに保存するとき、show メソッドの代わりに savefig メソッドを使用する。ファイルのフォーマットは format 引数で指定する。png の他に pdf、ps、eps、svg を指定できる。

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
y = np.array([1.2, 2.5, 3.4, 3.3, 2.8])

fig = plt.figure()
ax = fig.add_subplot()
ax.plot(x, y)
fig.savefig('fig1.png', format='png')
```



object-oriented interface / グラフ保存

グラフの軸目盛りなどに使う文字の大きさを調整したい場合は、set_xlabel などのメソッドを使う。また、グラフ画像のサイズや解像度などを調整したい場合は、Figure 領域を呼び出す際に行う。

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
y = np.array([1.2, 2.5, 3.4, 3.3, 2.8])

fig = plt.figure(figsize=[8, 6],
                 dpi=300)
ax = fig.add_subplot()
ax.plot(x, y)

ax.set_xlabel("xlabel", fontsize=18)
ax.set_ylabel("ylabel", fontsize=18)
ax.tick_params(labelsize=18)

fig.savefig('fig1.png', format='png')
```

matplotlib 基本

- object-oriented interface
- state-based interface

state-based interface

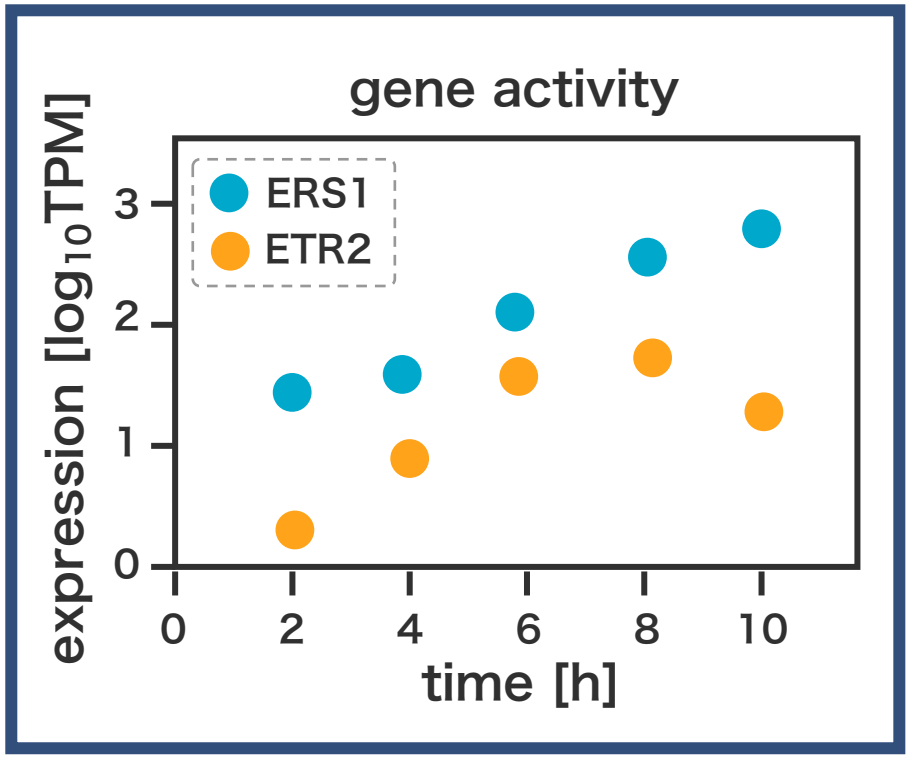
State-based interface は、すべての操作を pyplot のメソッドとして行うインタフェースである。pyplot が現在操作中の Figure クラスや Axes クラスを自動的に識別して操作し、グラフを作成する。State-based interface を用いて散布図を描くとき、右のようなコードを書く。

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
y = np.array([1.2, 2.5, 3.4, 3.3, 2.8])

plt.plot(x, y)
plt.show()
```

matplotlib API 可視化関数

object-oriented interface と state-based interface で利用できる可視化関数は次のように対応している。

グラフ	object-oriented	state-based
線グラフ	`ax.plot`	`plt.plot`
散布図	`ax.scatter`	`plt.scatter`
棒グラフ	`ax.bar`	`plt.bar`
ヒストグラム	`ax.hist`	`plt.hist`
ボックスプロット	`ax.boxplot`	`plt.boxplot`



matplotlib API 可視化関数

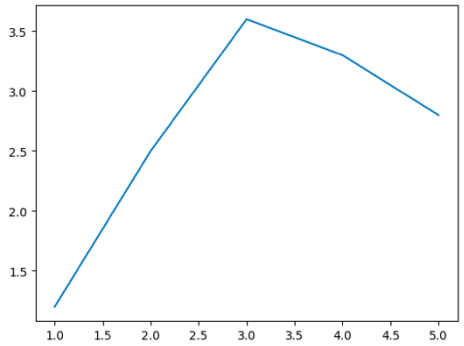
object-oriented interface と state-based interface で利用できる可視化関数は次のように対応している。

グラフ	object-oriented	state-based
グラフのタイトル	`ax.set_title`	`plt.title`
目盛りの比率	`ax.set_aspect`	`plt.axes().set_aspect`
グラフの凡例	`ax.legend`	`plt.legend`
x 軸の表示範囲	`ax.set_xlim`	`plt.xlim`
x 軸のラベル	`ax.set_xlabel`	`plt.xlabel`
x 軸の目盛りの表示位置	`ax.set_xticks`	`plt.xticks`
x 軸の目盛りの値	`ax.set_xticklabels`	
x 軸の目盛りのスケール	`ax.set_xscale`	`plt.xscale`

可視化

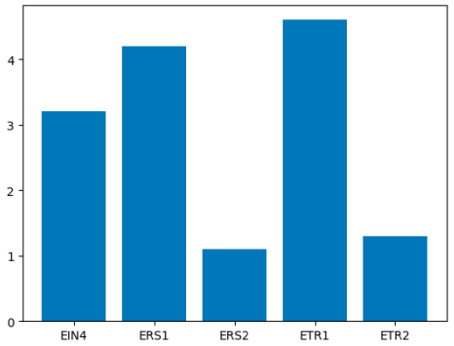
- matplotlib 基本
- 基本グラフ
- プロット領域の分割

基本グラフ



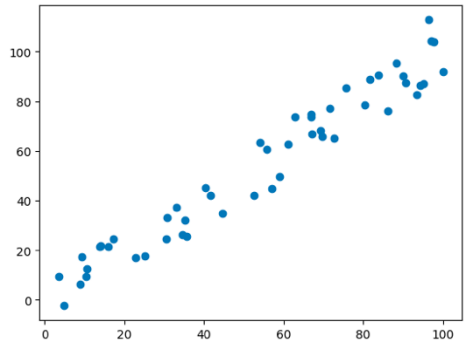
line chart

折れ線グラフは、系列データの変化を捉えるために使われる。



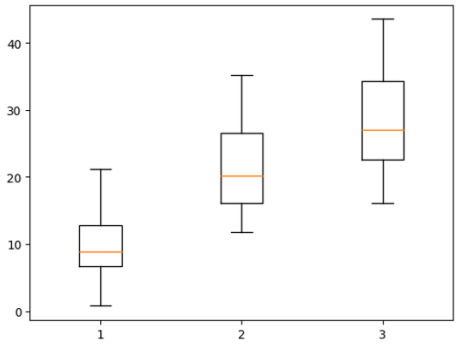
bar chart

棒グラフは、カテゴリカルデータを可視化する目的で使われる。なお、人を騙す目的で使用する場合は縦軸の起点を 0 以外にすると効果的である。



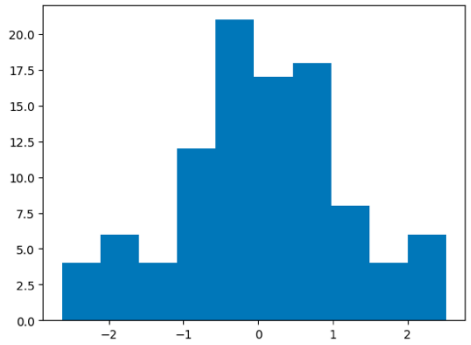
scatter chart

散布図は、2 変量の連続値データ同士の相関や分布などを可視化する目的で使われる。



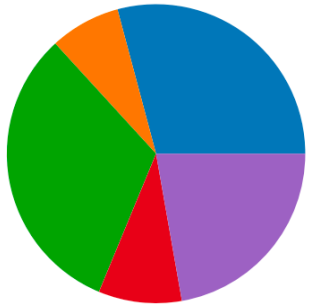
boxplot

ボックスプロットは、複数の連続量データの分布の特徴を可視化する目的で使われる。



histogram

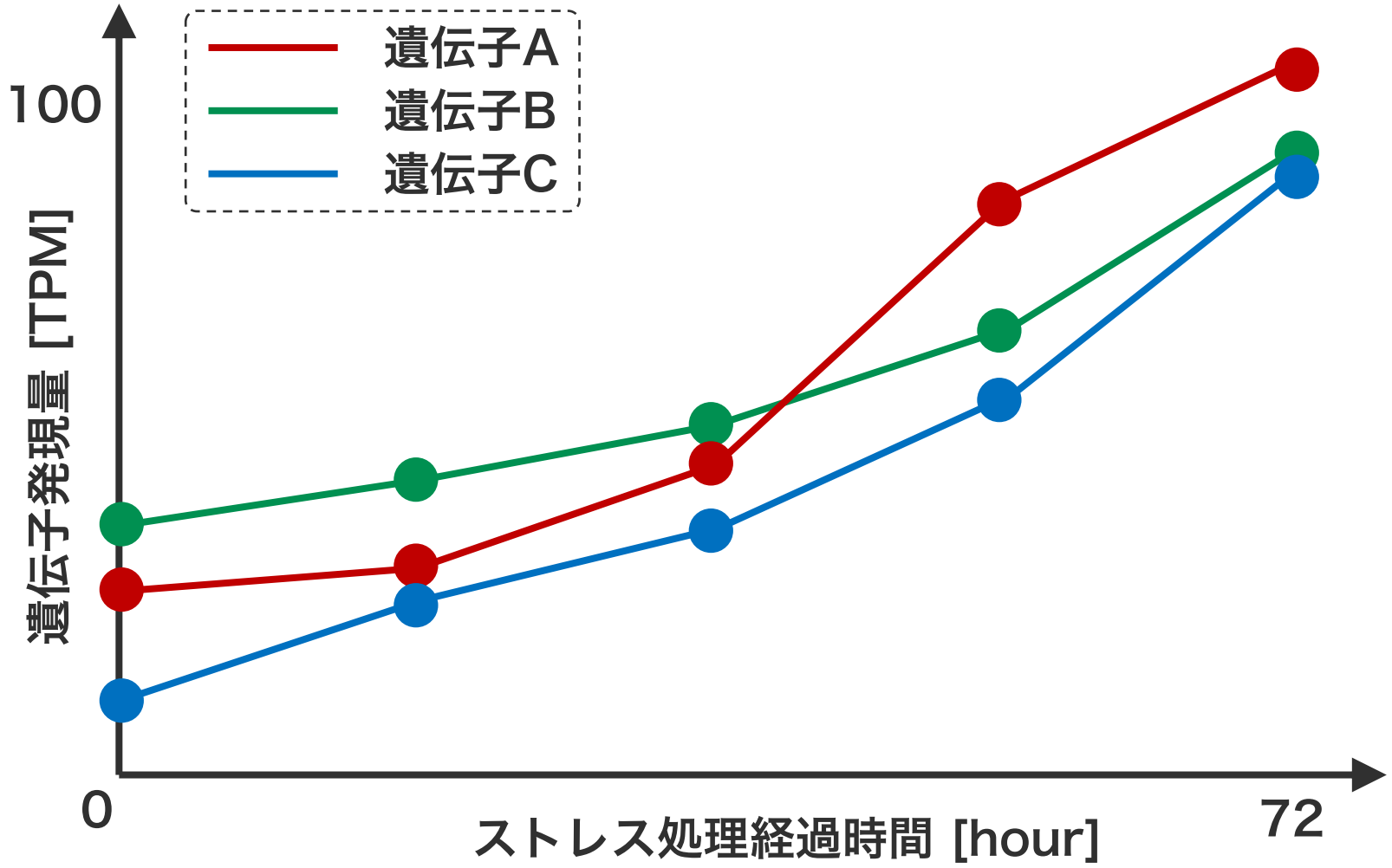
ヒストグラムは、1 変量の連続値データの分布を可視化する目的で使われる。



pie chart

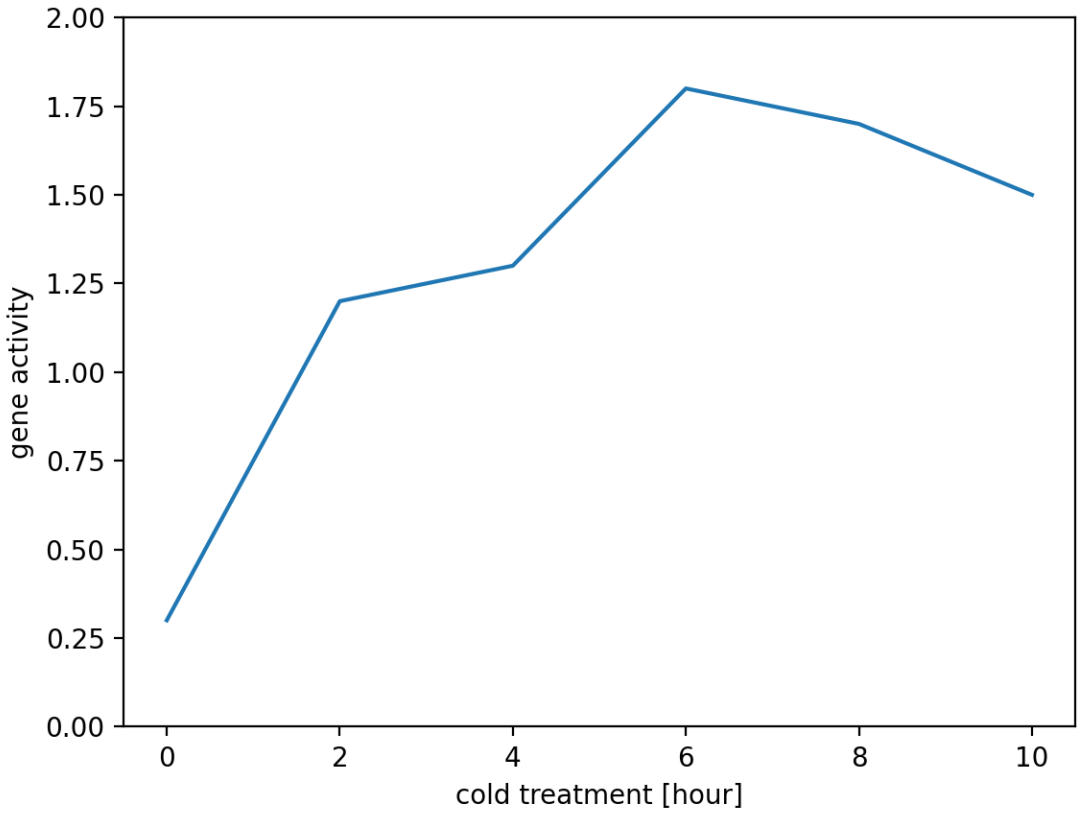
円グラフは、人を騙す目的で多用される。なお、騙し効果を上げるために、3D 円グラフや歪んだ円グラフを用いると良い。

線グラフ



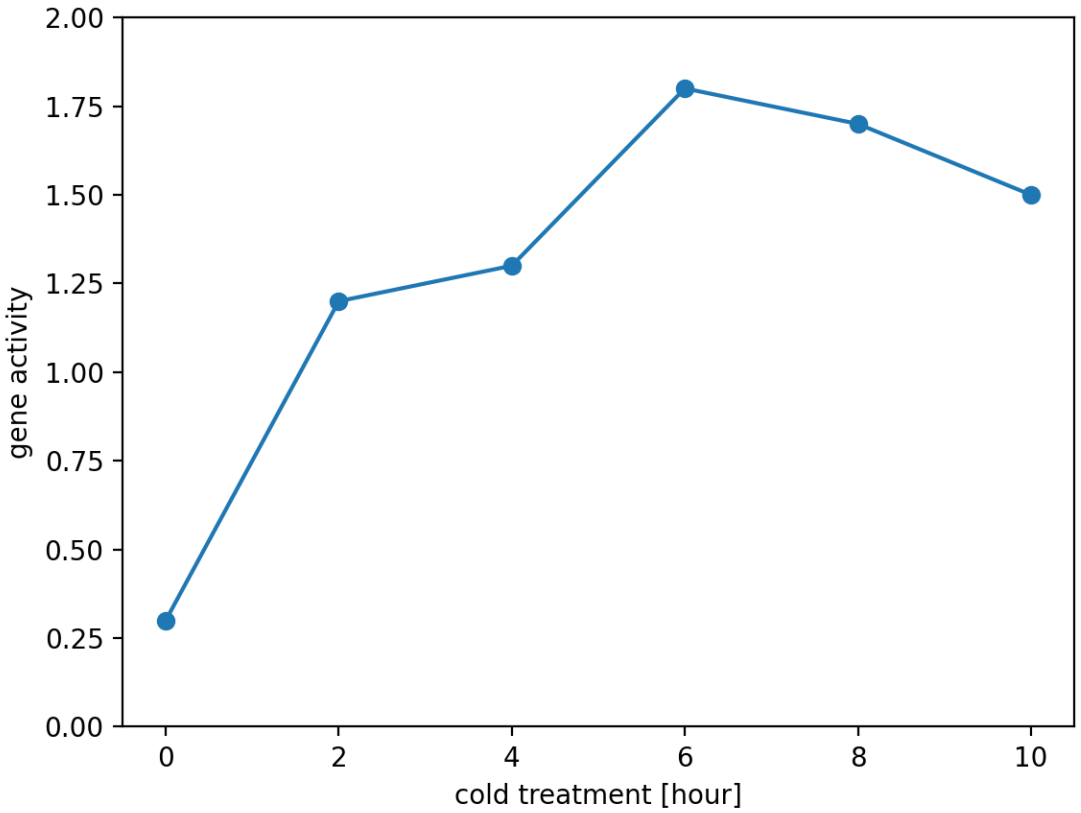
- 線グラフはデータの系列的な変化を見るためのグラフ
- 縦軸および横軸は連続量
- 原点 (0, 0) が省略されることがある
- 観測値を強調するために、観測値に点を明示することもある

線グラフ



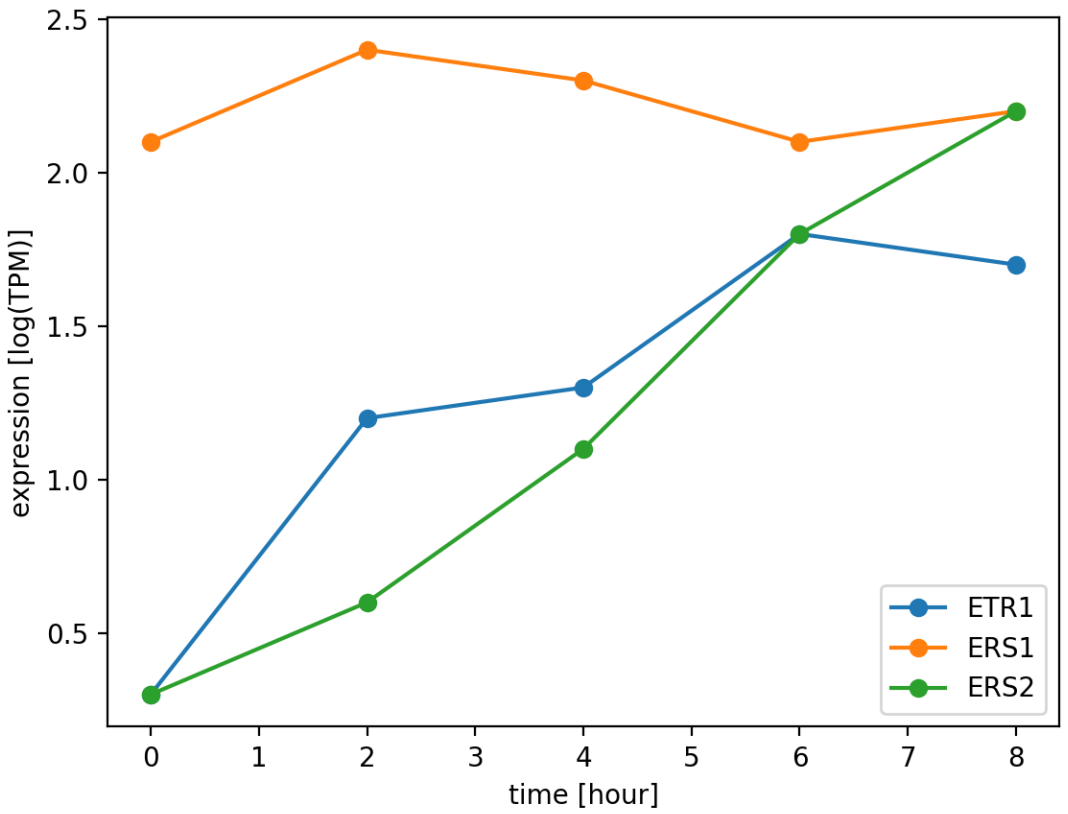

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([0, 2, 4, 6, 8, 10])
y = np.array([0.3, 1.2, 1.3,
              1.8, 1.7, 1.5])

fig = plt.figure()

ax = fig.add_subplot()

ax.plot(x, y)

ax.set_xlabel('cold treatment [hour]')
ax.set_ylabel('gene activity')
ax.set_ylim(0, 2)

fig.show()
```

線グラフ




```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([0, 2, 4, 6, 8, 10])
y = np.array([0.3, 1.2, 1.3,
              1.8, 1.7, 1.5])

fig = plt.figure()

ax = fig.add_subplot()

ax.plot(x, y, marker='o')

ax.set_xlabel('cold treatment [hour]')
ax.set_ylabel('gene activity')
ax.set_ylim(0, 2)

fig.show()
```

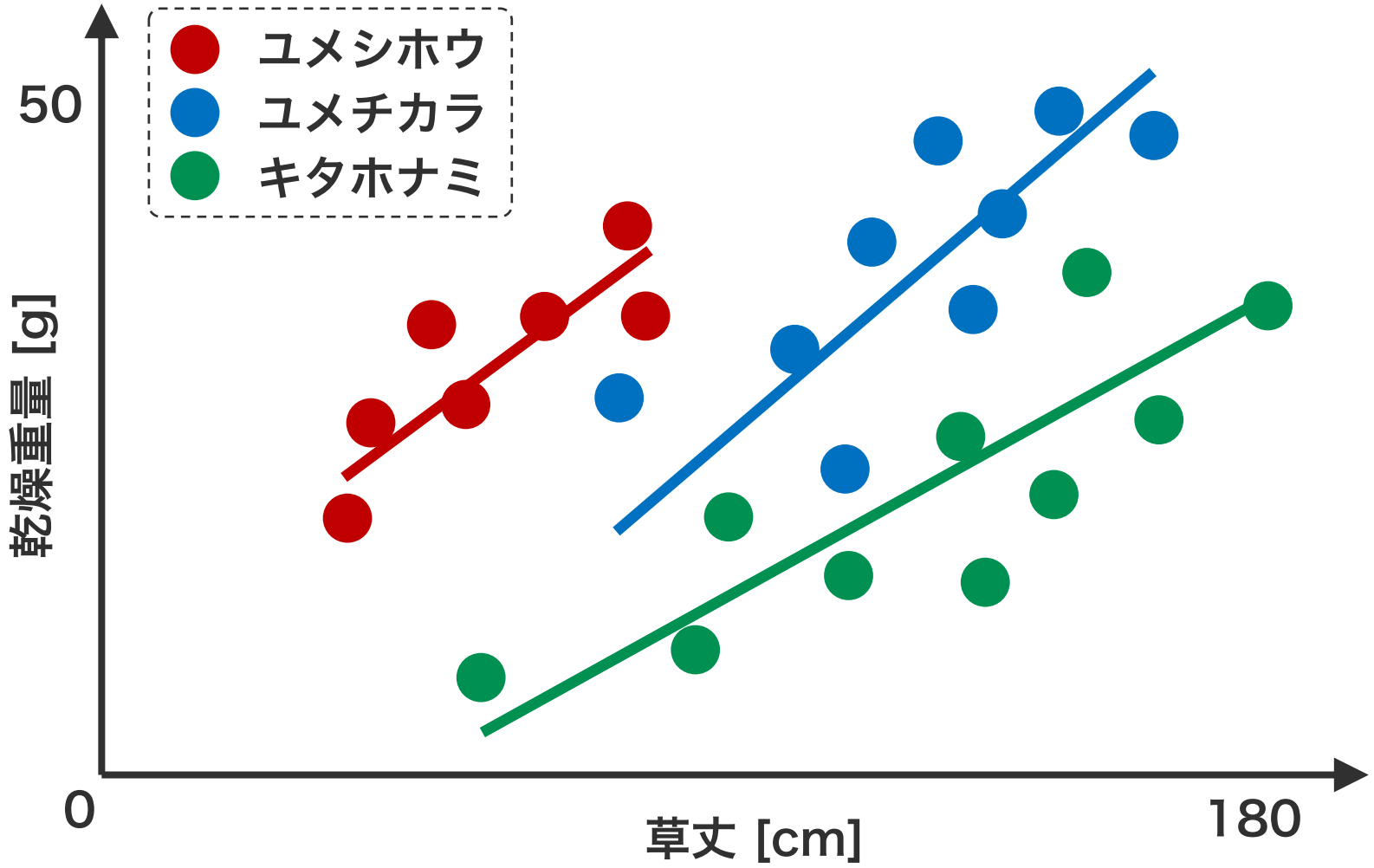
線グラフ




```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([0, 2, 4, 6, 8])
g1 = np.array([0.3, 1.2, 1.3, 1.8, 1.7])
g2 = np.array([2.1, 2.4, 2.3, 2.1, 2.2])
g3 = np.array([0.3, 0.6, 1.1, 1.8, 2.2])

fig = plt.figure()

ax = fig.add_subplot()

ax.plot(x, g1, label='ETR1', marker='o')
ax.plot(x, g2, label='ERS1', marker='o')
ax.plot(x, g3, label='ERS2', marker='o')

ax.legend()
ax.set_xlabel('time [hour]')
ax.set_ylabel('expression [log(TPM)]')
fig.show()
```

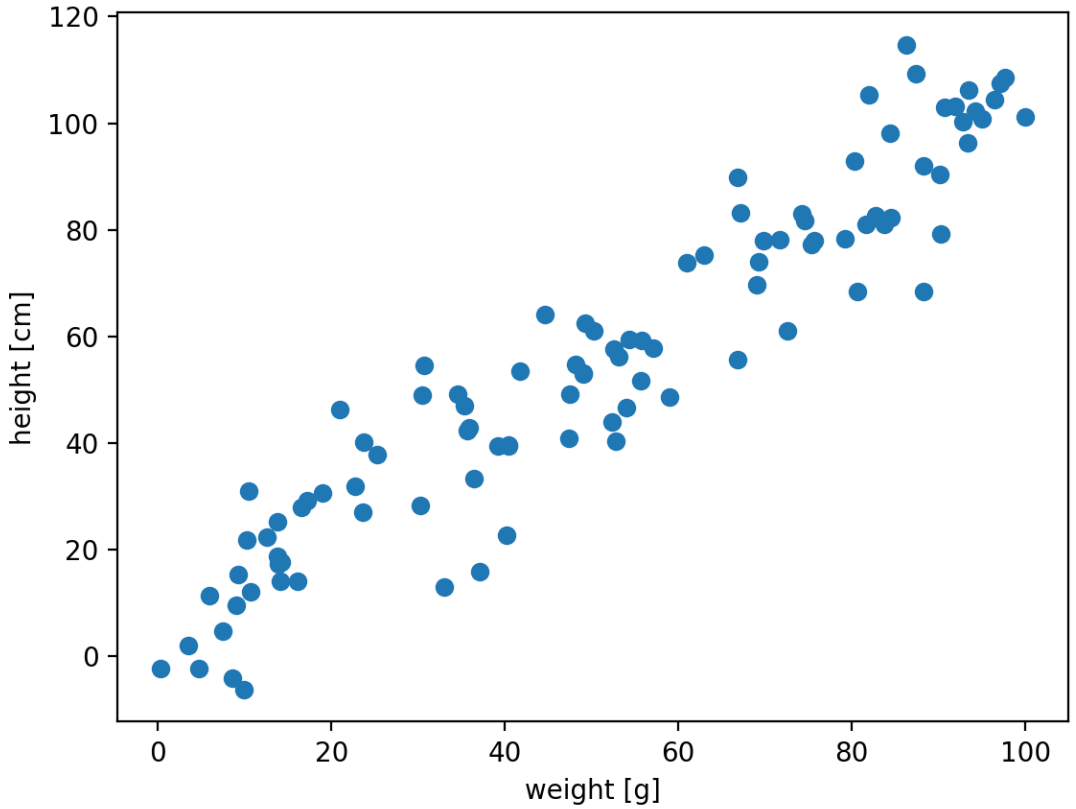
plot メソッドを複数回使うことで、複数の線グラフを描くことができる。グラフを描く際に色を指定しない場合は、自動的に配色される。

散布図



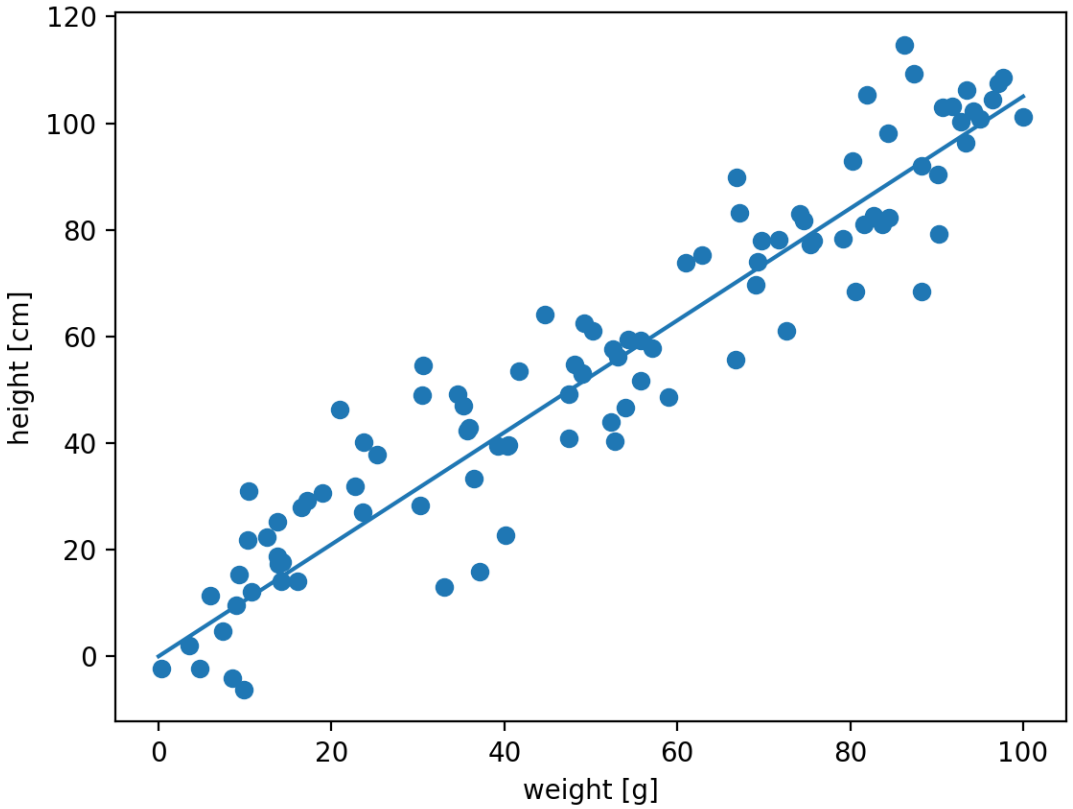
- 多変量データ同士の相関や分布などを見るためのグラフ
- 縦軸および横軸は連続量
- 原点（0, 0）が省略されることがある
- 回帰直線との併用もよく見られる

散布図




```
import matplotlib.pyplot as plt
import numpy as np

np.random.seed(2018)
x = np.random.uniform(0, 100, 100)
y = x + np.random.normal(5, 10, 100)

fig = plt.figure()

ax = fig.add_subplot()

ax.scatter(x, y)


ax.set_xlabel('weight [g]')
ax.set_ylabel('height [cm]')

fig.show()
```

散布図




```
import matplotlib.pyplot as plt
import numpy as np

np.random.seed(2018)
x = np.random.uniform(0, 100, 100)
y = x + np.random.normal(5, 10, 100)

fig = plt.figure()

ax = fig.add_subplot()

ax.scatter(x, y)
ax.plot([0, 100], [0, 100 + 5])

ax.set_xlabel('weight [g]')
ax.set_ylabel('height [cm]')

fig.show()
```

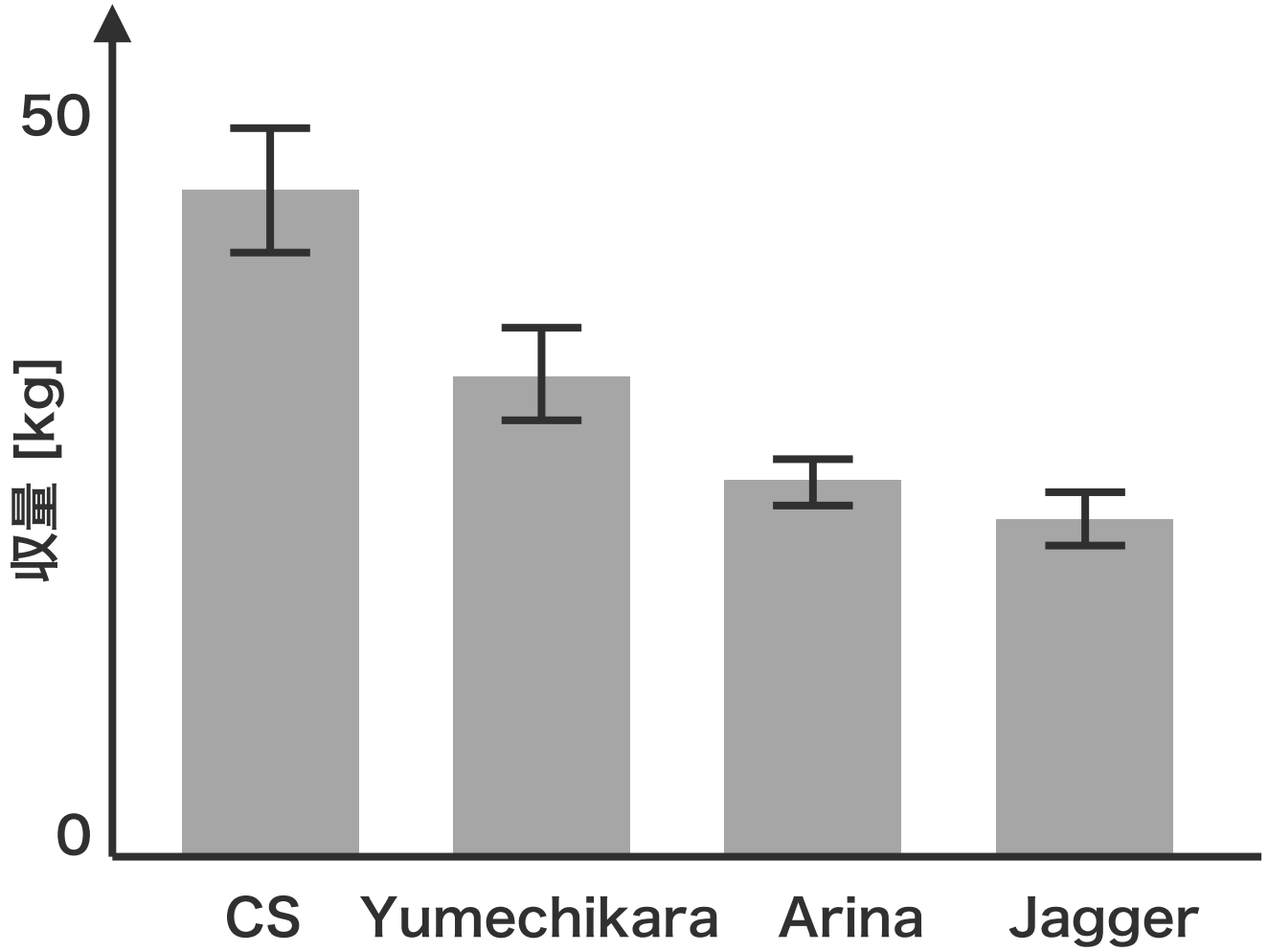
問題 M1-1

diversity_galapagos.txt には、ガラパゴス島における種の多様性データが記載されている。このデータを読み込み、島の面積（Area）と種数（Species）の関係を散布図で描け。

```
import pandas as pd

f = 'diversity_galapagos.txt'
d = pd.read_csv(f, comment='#', header=0, sep='\t', index_col=0)
```

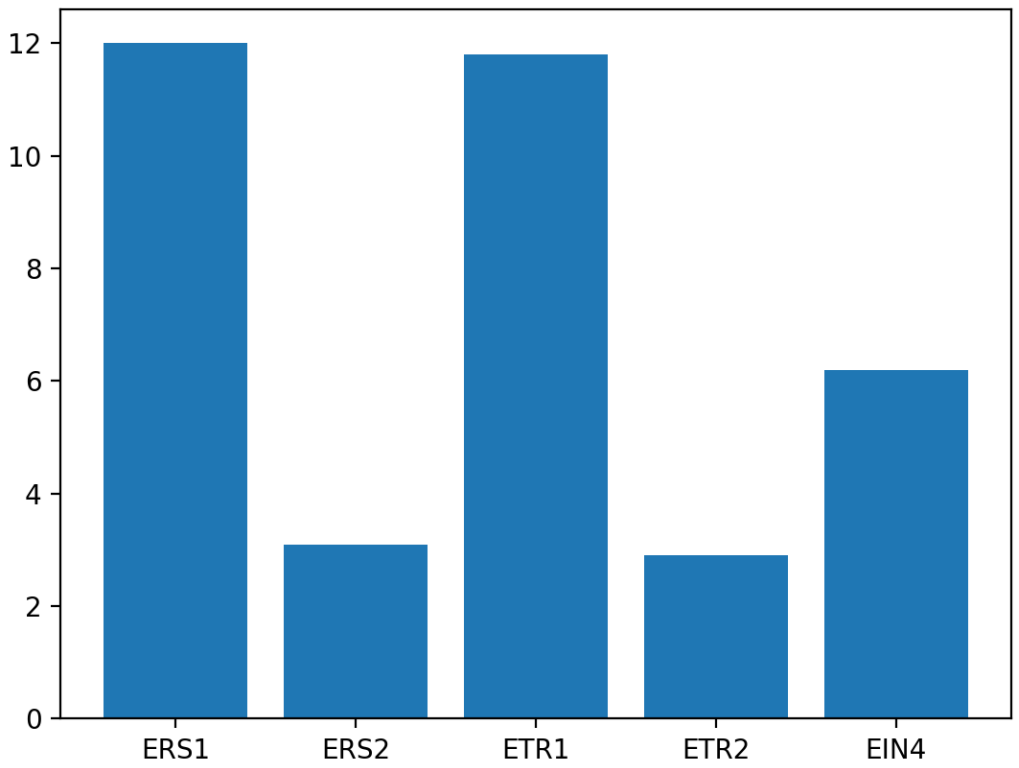
https://aabbdd.jp/notes/data/diversity_galapagos.txt

棒グラフ



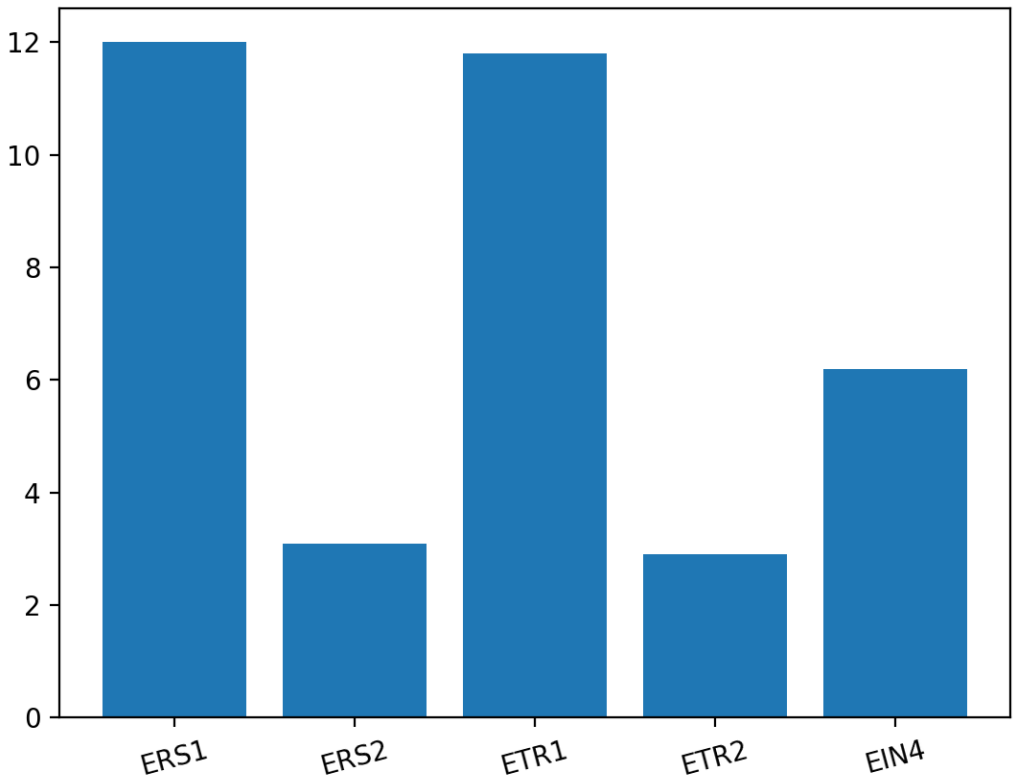
- 複数のカテゴリに属している値同士の大小を可視化するためのグラフ
- 縦軸が連続量、横軸がカテゴリ、あるいはその逆
- 人を騙す目的で使用するとき、原点をゼロ以外の値にしたり、縦軸の目盛り間隔を操作すると効果的である
- エラーバーとともに用いられることがある

棒グラフ



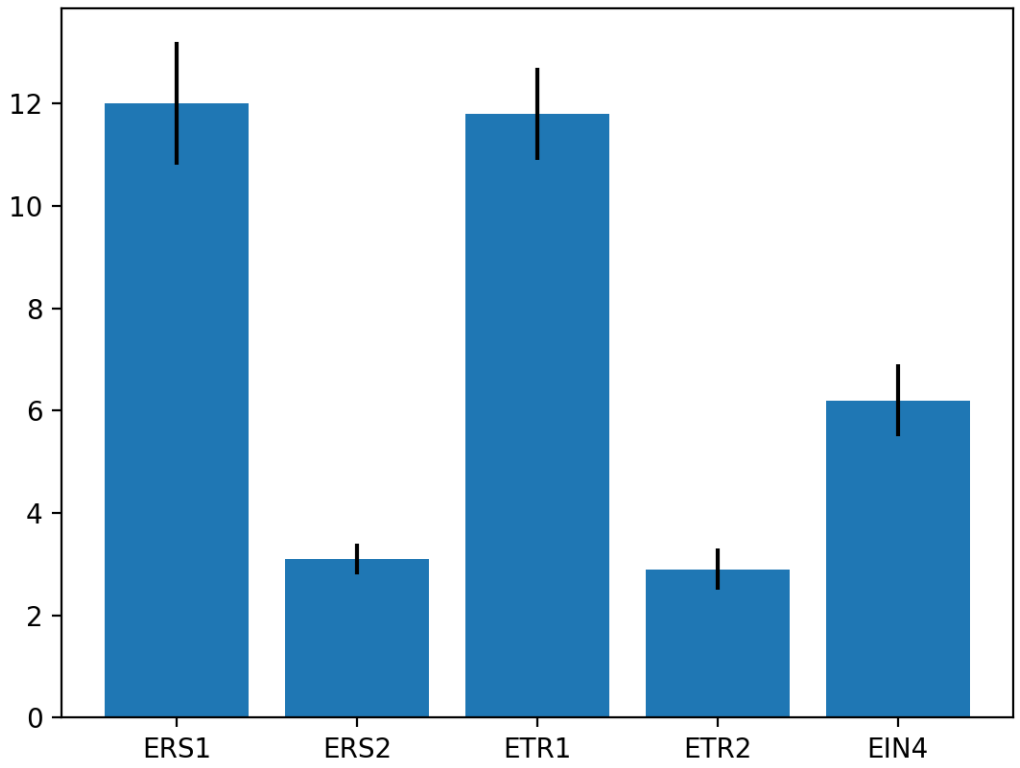

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(['ERS1', 'ERS2', 'ETR1',
              'ETR2', 'EIN4'])
y = np.array([12.0, 3.1, 11.8, 2.9, 6.2])

fig = plt.figure()
ax = fig.add_subplot()
ax.bar(x, y)

fig.show()
```

棒グラフ



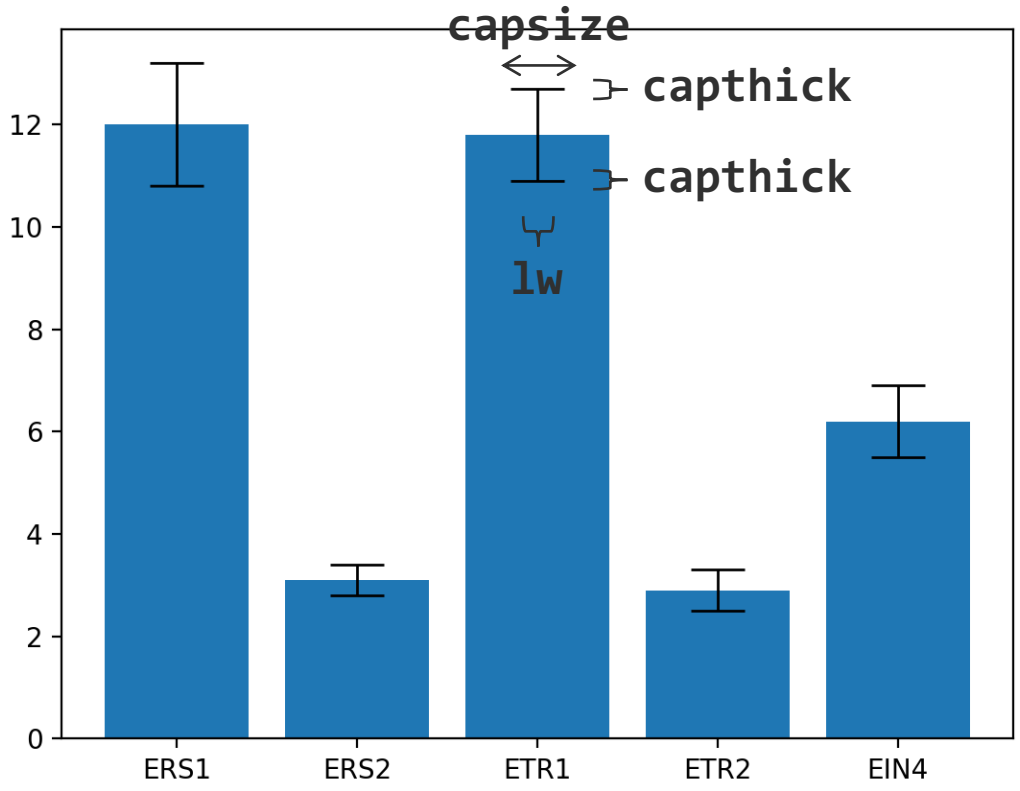

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(['ERS1', 'ERS2', 'ETR1',
              'ETR2', 'EIN4'])
y = np.array([12.0, 3.1, 11.8, 2.9, 6.2])

fig = plt.figure()
ax = fig.add_subplot()
ax.bar(x, y)
ax.set_xticklabels(x, rotation=15)

fig.show()
```

棒グラフ



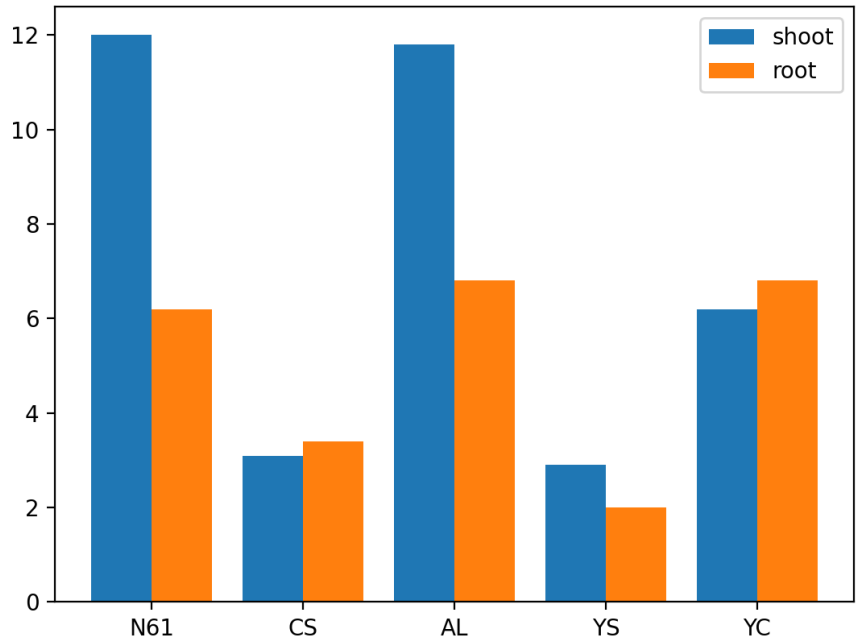

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(['ERS1', 'ERS2', 'ETR1',
              'ETR2', 'EIN4'])
y = np.array([12.0, 3.1, 11.8, 2.9, 6.2])
e = np.array([1.2, 0.3, 0.9, 0.4, 0.7])

fig = plt.figure()
ax = fig.add_subplot()


ax.bar(x, y, yerr = e)

fig.show()
```

棒グラフ




```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(['ERS1', 'ERS2', 'ETR1',
              'ETR2', 'EIN4'])
y = np.array([12.0, 3.1, 11.8, 2.9, 6.2])
e = np.array([1.2, 0.3, 0.9, 0.4, 0.7])

fig = plt.figure()
ax = fig.add_subplot()
error_bar_set = dict(lw = 1, capthick = 1,
                     capsize = 10)
ax.bar(x, y, yerr = e,
       error_kw=error_bar_set)
fig.show()
```

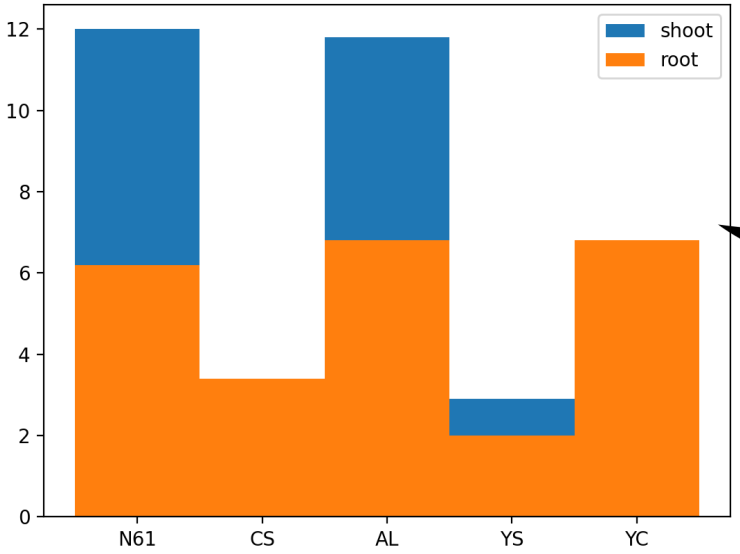
棒グラフ



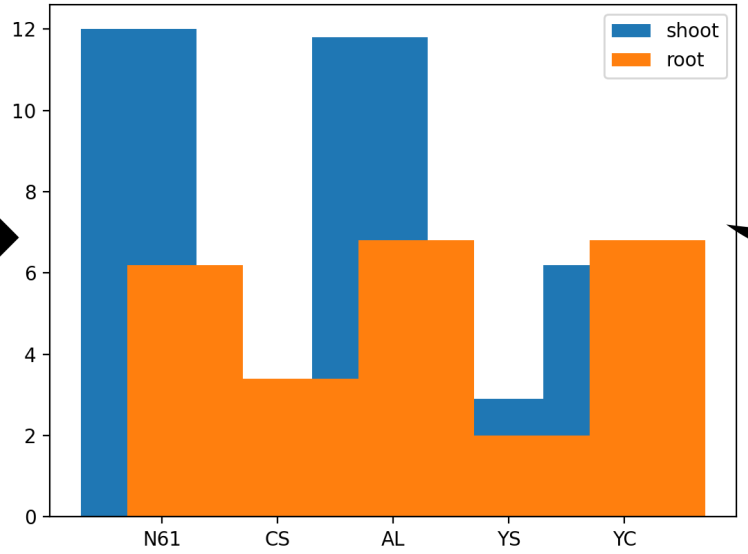

```
import matplotlib.pyplot as plt
import numpy as np

xlabel = np.array(['N61', 'CS', 'AL', 'YS',  'YC'])

x = np.array([0, 1, 2, 3, 4])
y_shoot = np.array([12.0, 3.1, 11.8, 2.9, 6.2])
y_root = np.array([6.2, 3.4, 6.8, 2.0, 6.8])

fig = plt.figure()
ax = fig.add_subplot()

ax.bar(x - 0.2, y_shoot, width=0.4,label='shoot')
ax.bar(x + 0.2, y_root, width=0.4, label='root')

ax.legend()
ax.set_xticks(x)
ax.set_xticklabels(xlabel)
fig.show()
```

棒グラフ

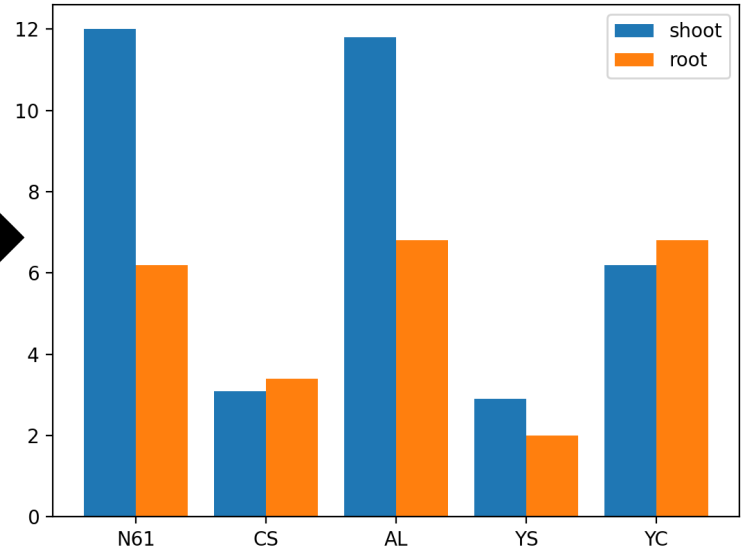
```
ax.bar(x, shoot)
ax.bar(x, root)
```



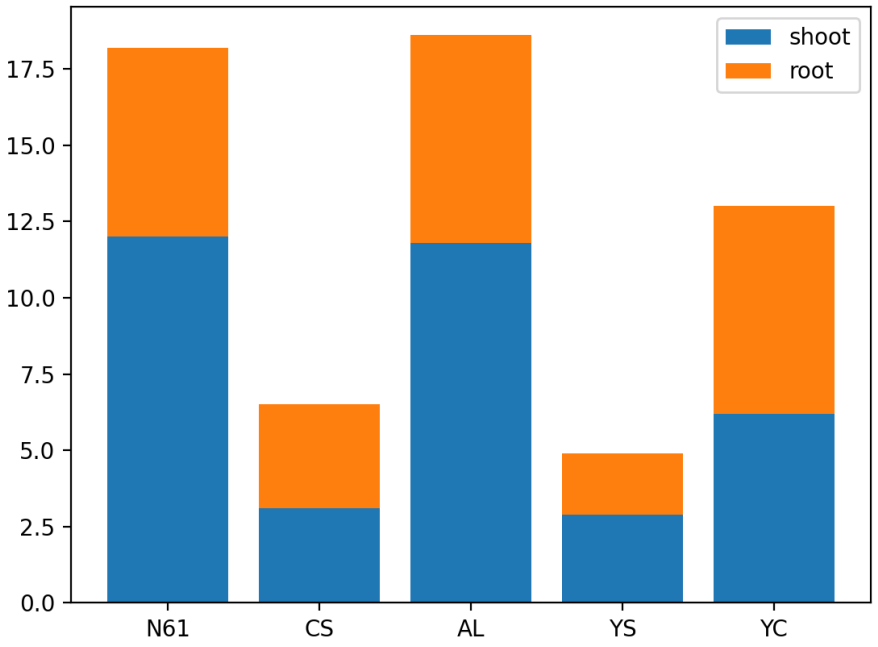

```
ax.bar(x - 0.2, shoot)
ax.bar(x + 0.2, root)
```




```
ax.bar(x - 0.2, shoot, width=0.4)
ax.bar(x + 0.2, root, width=0.4)
```



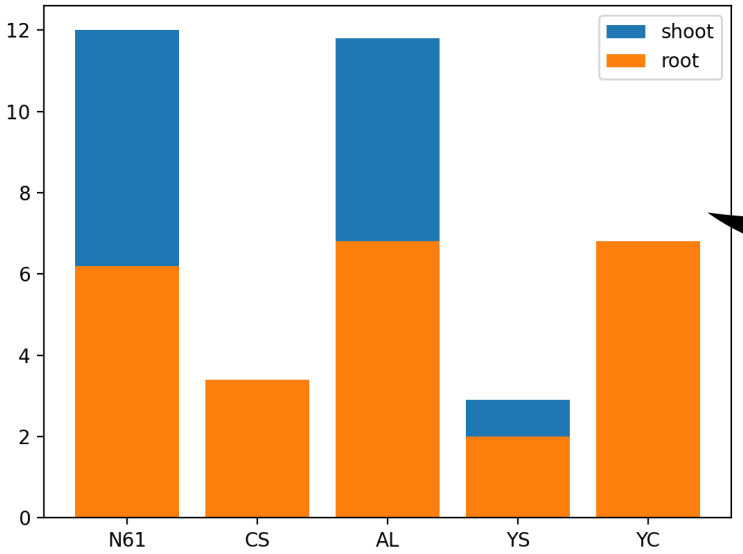
棒グラフ



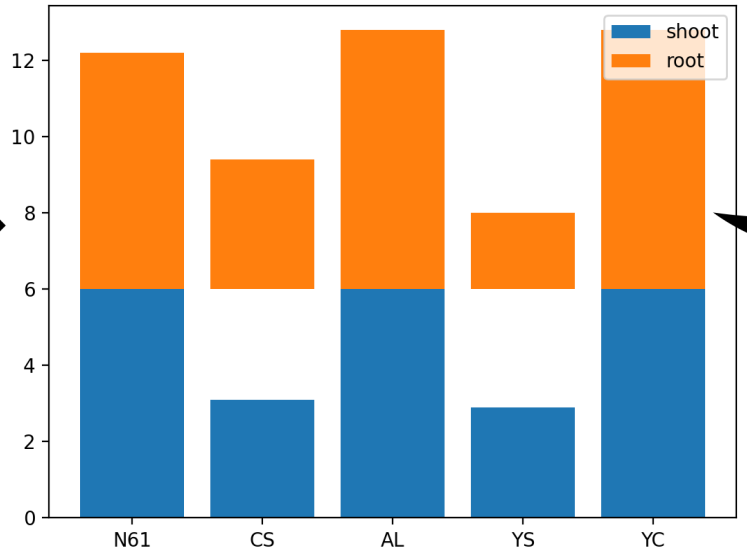

```
import matplotlib.pyplot as plt
import numpy as np

xlabel = np.array(['N61', 'CS', 'AL', 'YS', 'YC'])

x = np.array([0, 1, 2, 3, 4])
y_shoot = np.array([12.0, 3.1, 11.8, 2.9, 6.2])
y_root = np.array([6.2, 3.4, 6.8, 2.0, 6.8])

fig = plt.figure()
ax = fig.add_subplot()

ax.bar(x, y_shoot, label='shoot')
ax.bar(x, y_root, label='root', bottom=y_shoot)

ax.legend()
ax.set_xticks(x)
ax.set_xticklabels(xlabel)
fig.show()
```

棒グラフ

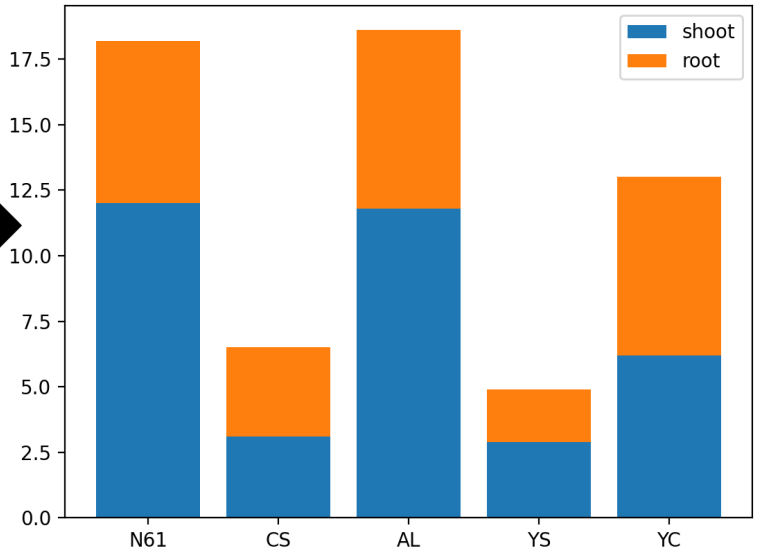
```
ax.bar(x, shoot)
ax.bar(x, root)
```



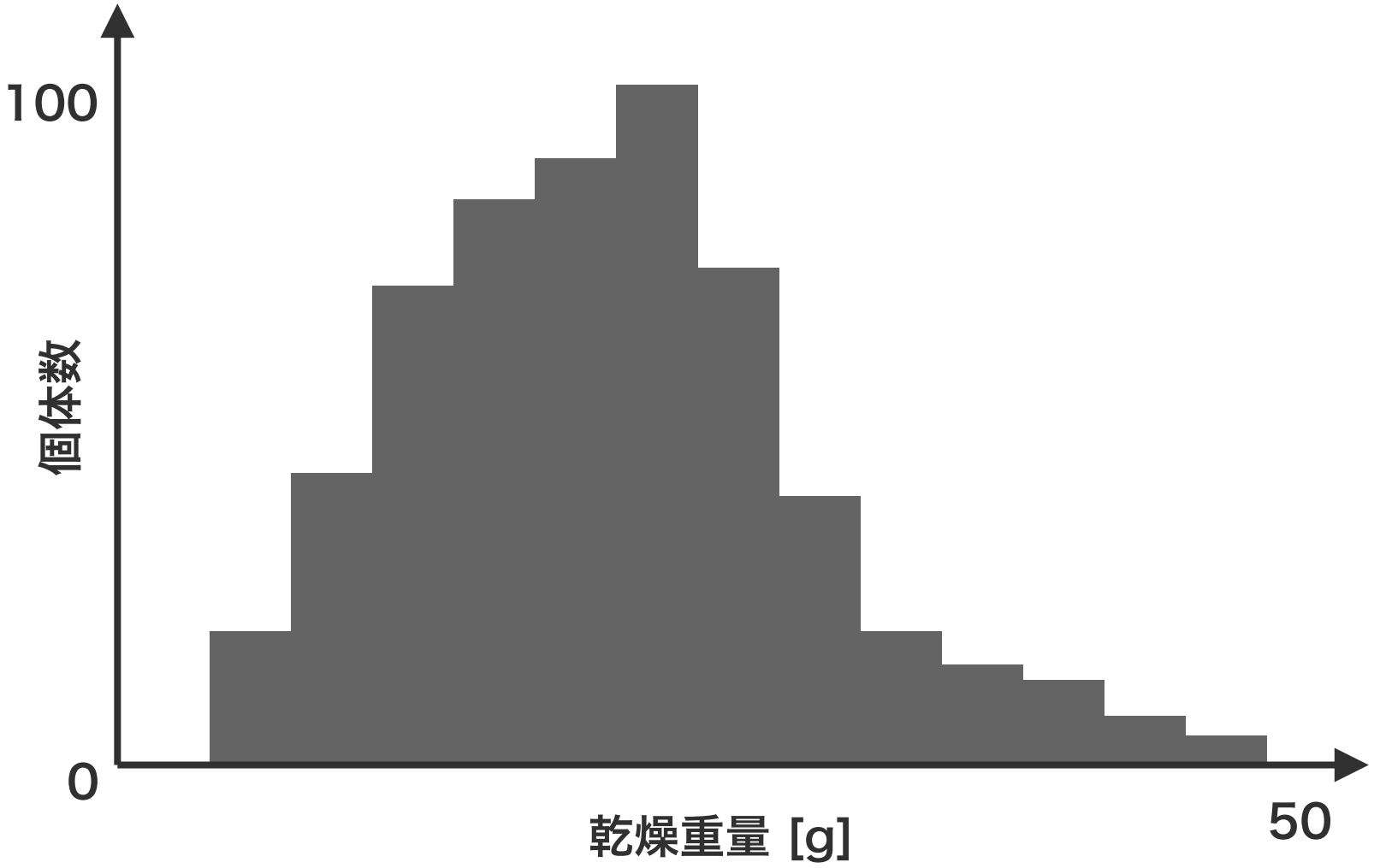

```
ax.bar(x, shoot)
ax.bar(x, root, bottom=6)
```




```
ax.bar(x, shoot)
ax.bar(x, root, bottom=shoot)
```

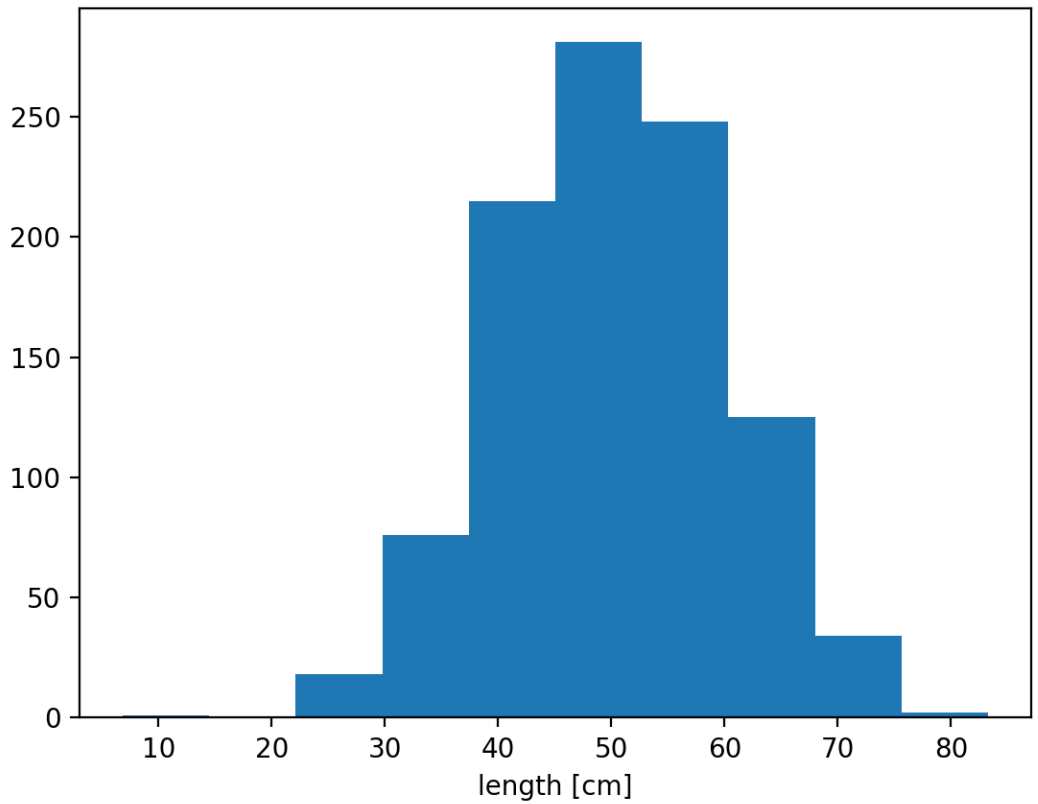


ヒストグラム



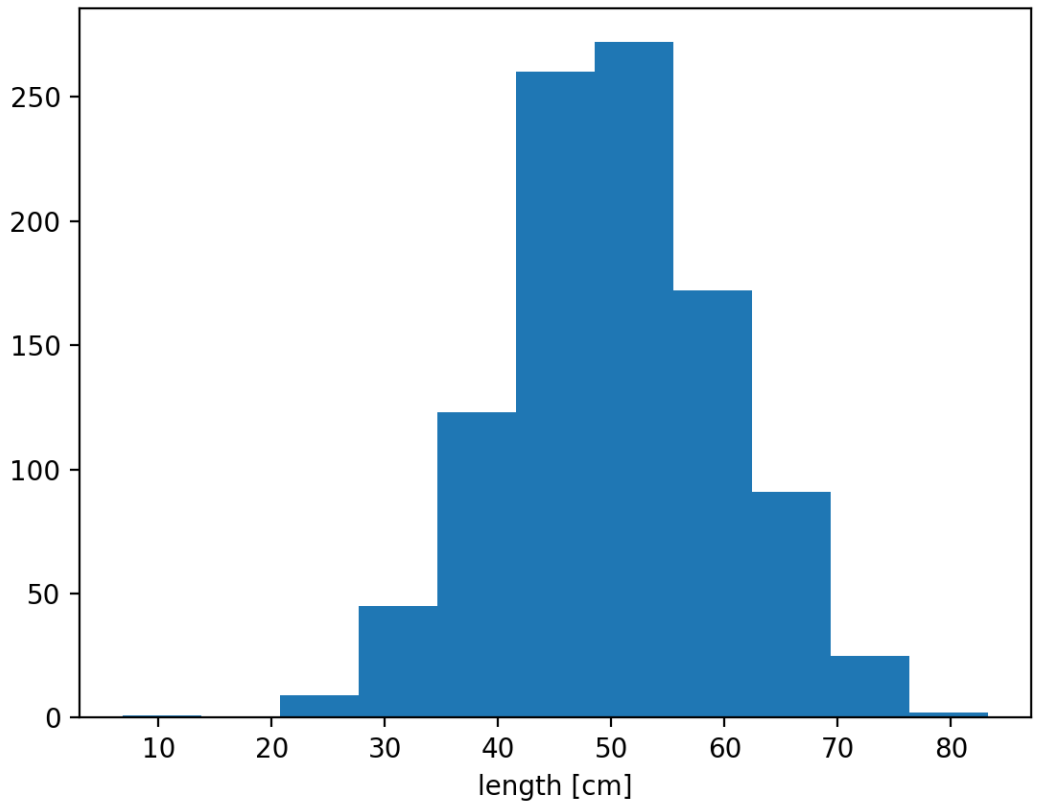
- １変量の連続値データを可視化するためのグラフ
- 横軸が連続量であり、縦軸は頻度・個数または確率である
- 横幅は恣意的（経験的）に決めることもあれば、スタージェスの公式などで決めることもある

ヒストグラム



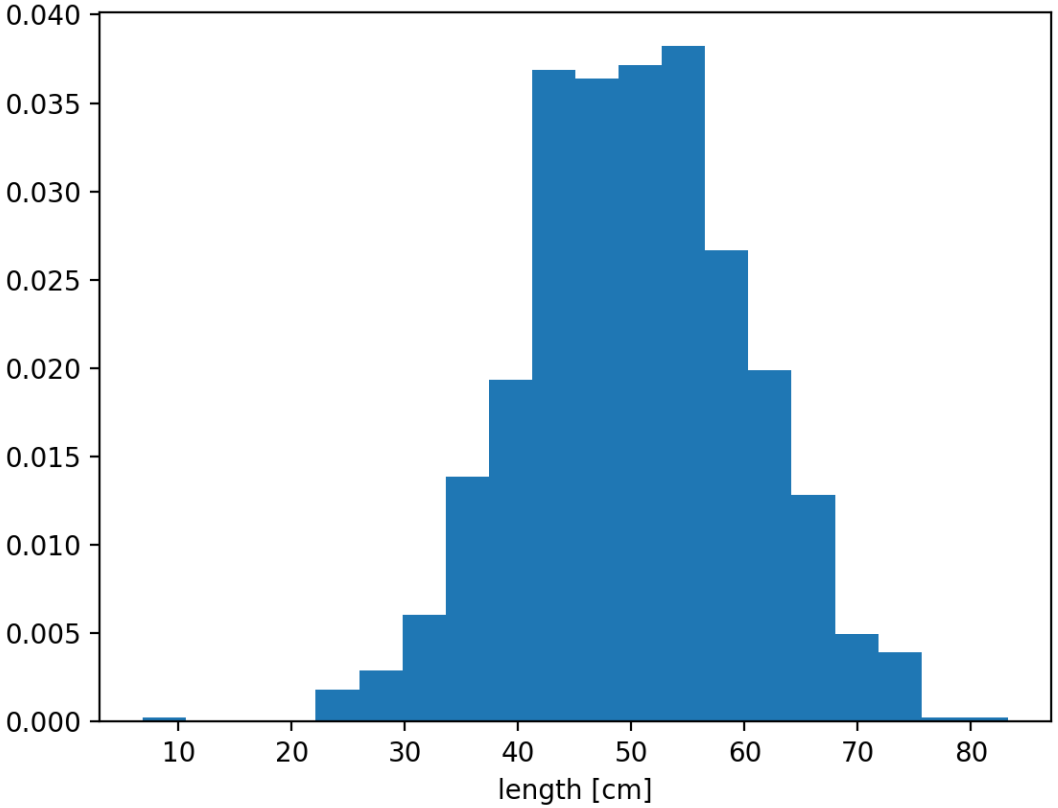

```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(2018)

x = np.random.normal(50, 10, 1000)

fig = plt.figure()
ax = fig.add_subplot()
ax.hist(x)

ax.set_xlabel('length [cm]')

fig.show()
```

ヒストグラム




```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(2018)

x = np.random.normal(50, 10, 1000)

fig = plt.figure()
ax = fig.add_subplot()
ax.hist(x, bins='sturges')

ax.set_xlabel('length [cm]')

fig.show()
```

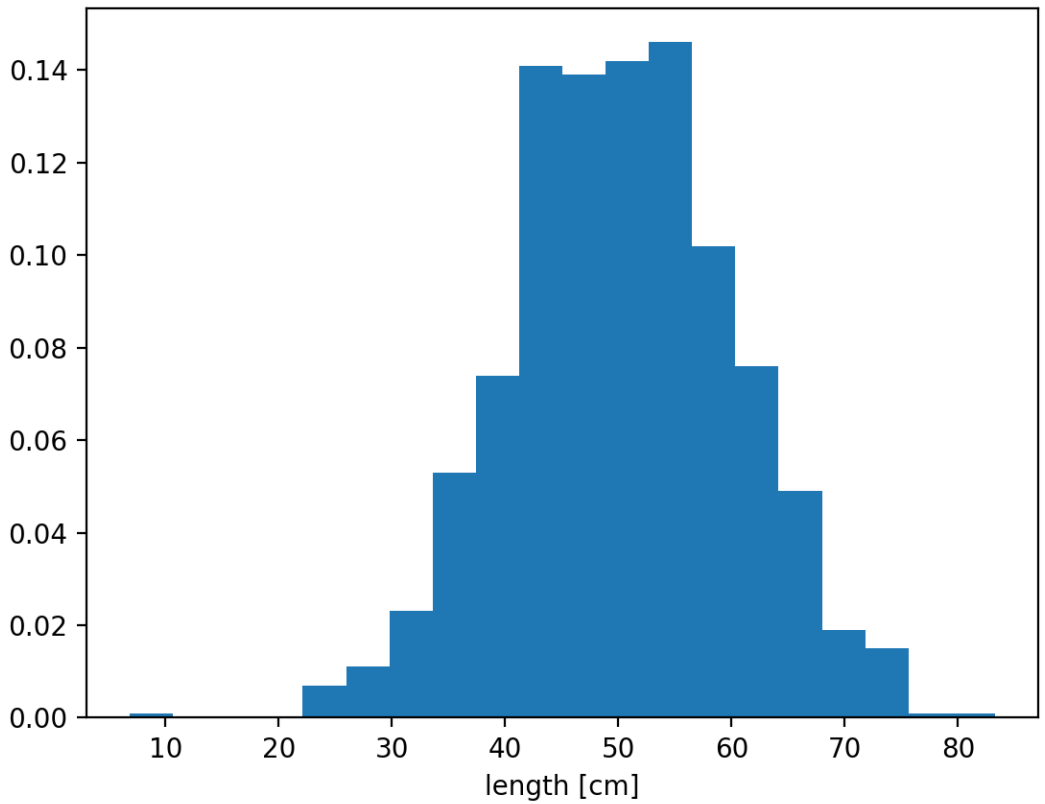
ヒストグラム




```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(2018)

x = np.random.normal(50, 10, 1000)

fig = plt.figure()
ax = fig.add_subplot()
ax.hist(x, bins=20, density=True)

ax.set_xlabel('length [cm]')

fig.show()
```

density=True を指定したとき、すべてのビンの面積を足すと 1 になる。棒の高さを足して 1 になるわけではないことに注意。

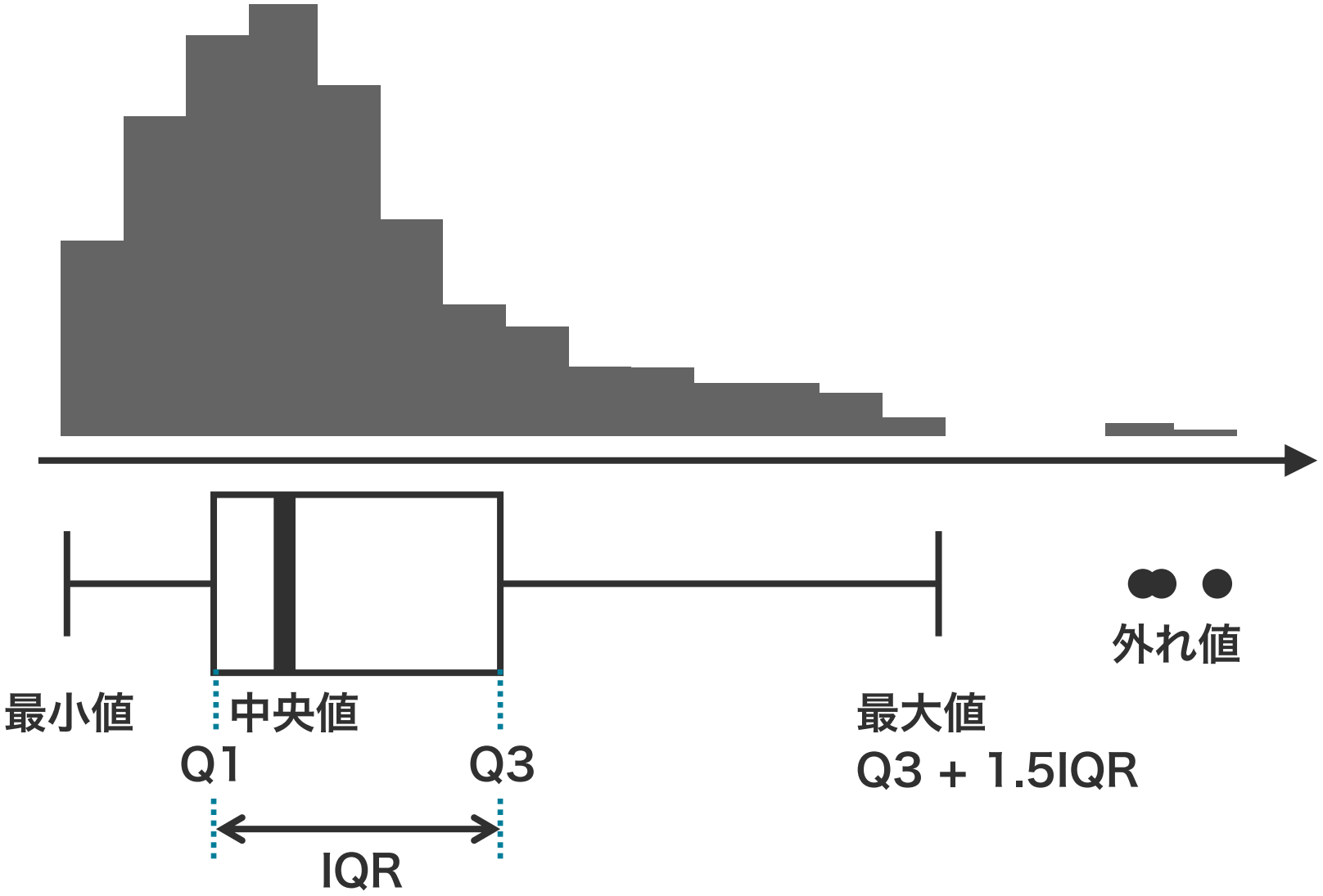
ヒストグラム




```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(2018)

x = np.random.normal(50, 10, 1000)
w = np.ones_like(x)/float(len(x))

fig = plt.figure()
ax = fig.add_subplot()
ax.hist(x, bins=20, weights=w)
ax.set_xlabel('length [cm]')

fig.show()
```

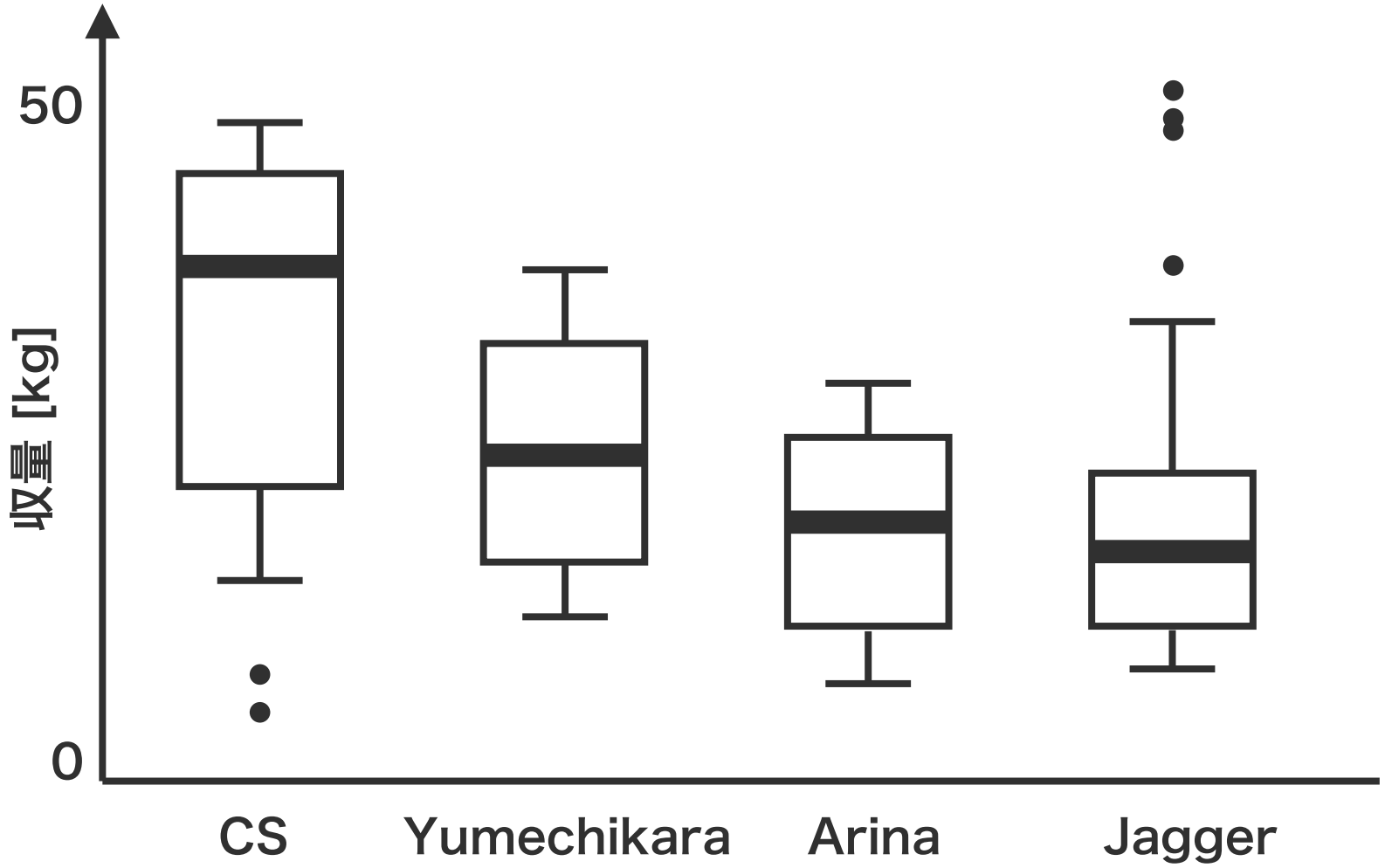
ビンの高さの合計値が 1 となるように、データに重みをおかけてヒストグラム描く。

ボックスプロット



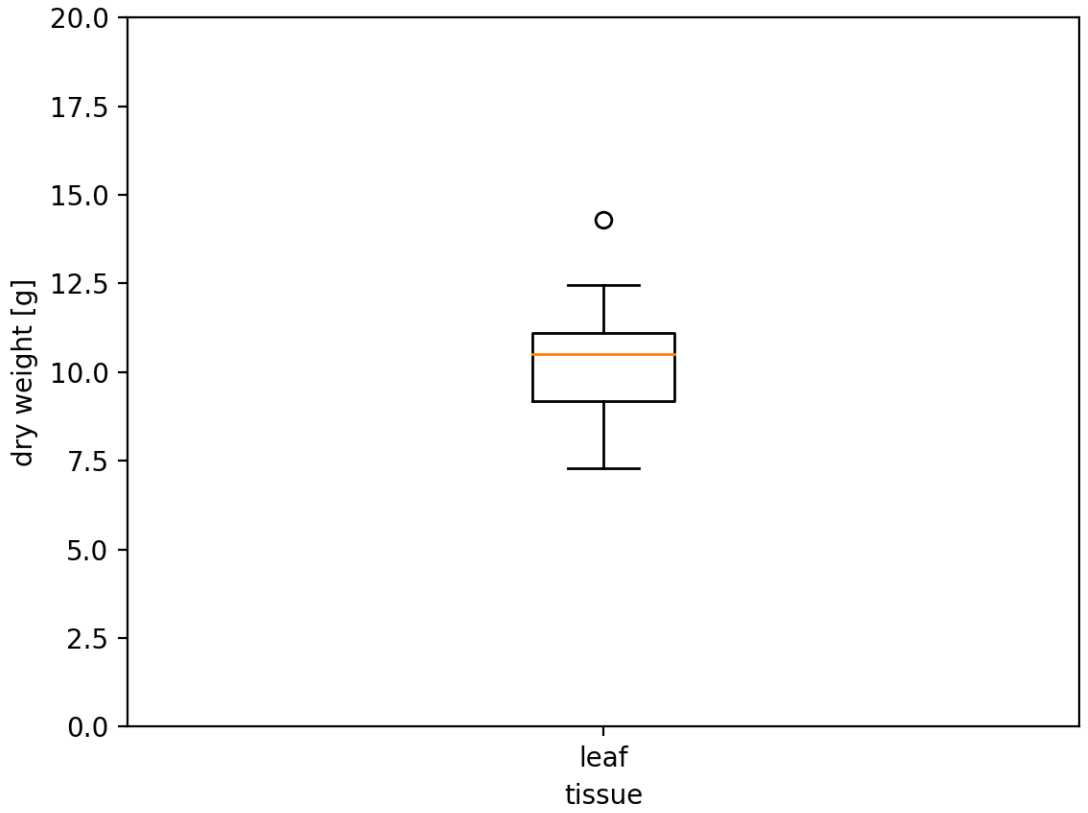
- 複数の 1 変量の連続値データを可視化するためのグラフ
- 最大値、最小値、第 1 四分位数 Q1、中央値、第 3 四分位数 Q3 などを簡単に確認できる
- [Q1 − 1.5IQR, Q3 + 1.5IQR] の外側にあるデータは外れ値

ボックスプロット



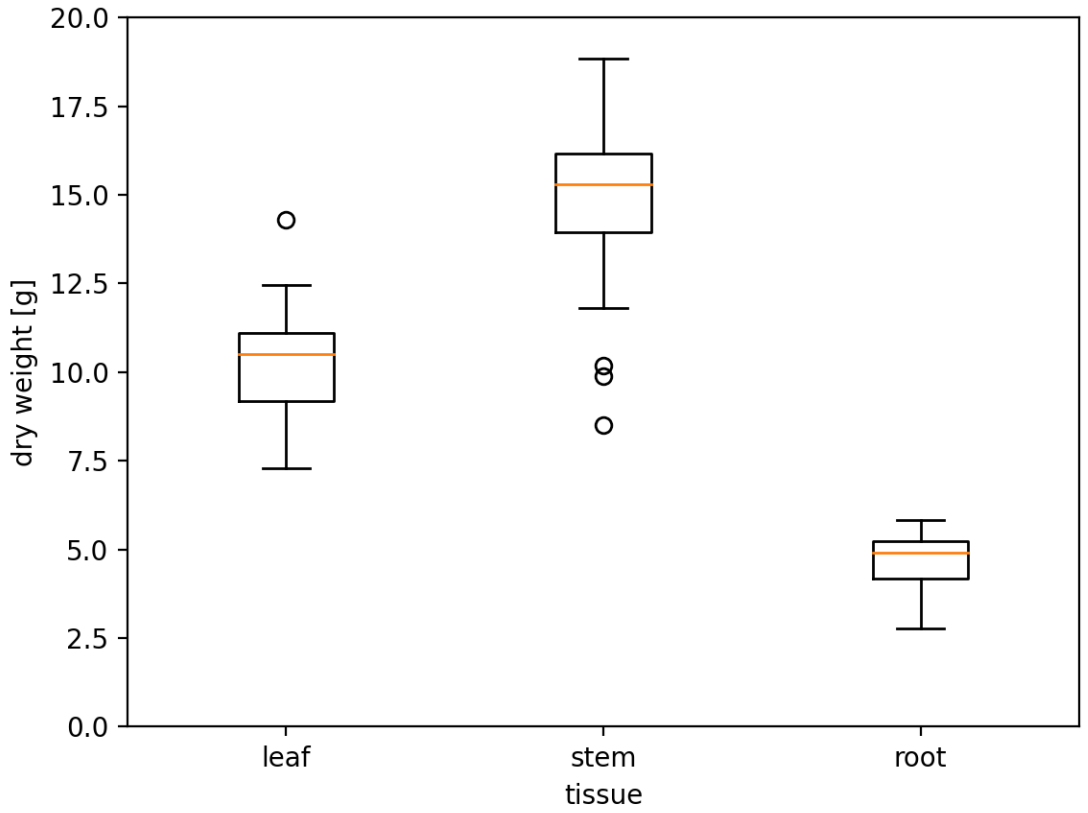
- 複数の 1 変量の連続値データを可視化するためのグラフ
- 最大値、最小値、第 1 四分位数 Q1、中央値、第 3 四分位数 Q3 などを簡単に確認できる
- [Q1 – 1.5IQR, Q3 + 1.5IQR] の外側にあるデータは外れ値
- 複数の変量の分布を同時に比べるときに便利

ボックスプロット



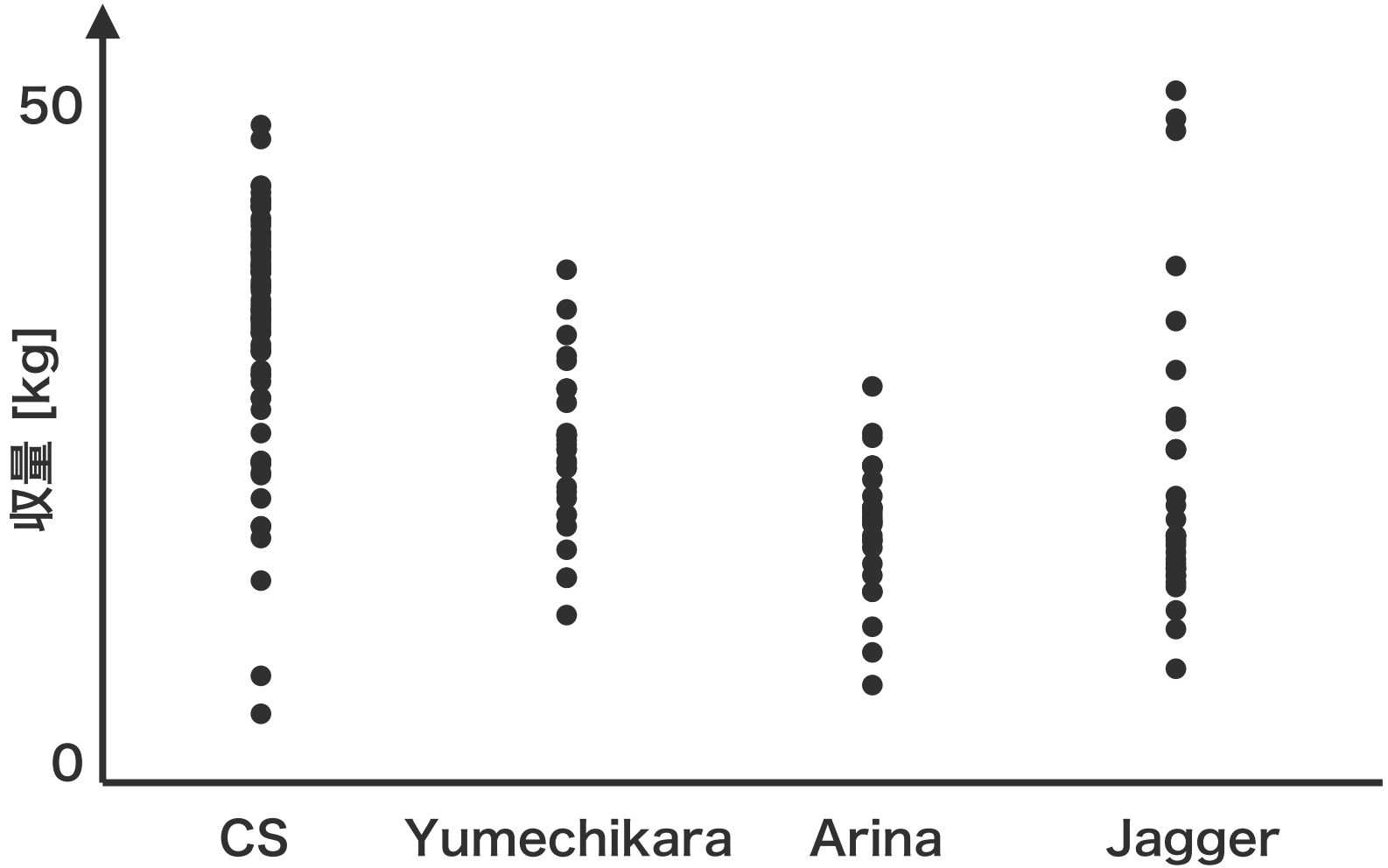

```
import numpy as  np
import matplotlib.pyplot as plt
np.random.seed(2018)

x1 = np.random.normal(10, 2, 20)



fig = plt.figure()
ax = fig.add_subplot()


ax.boxplot([x1], labels=['leaf'])



ax.set_xlabel('tissue')
ax.set_ylabel('dry weight [g]')
ax.set_ylim(0, 20)
fig.show()
```

ボックスプロット



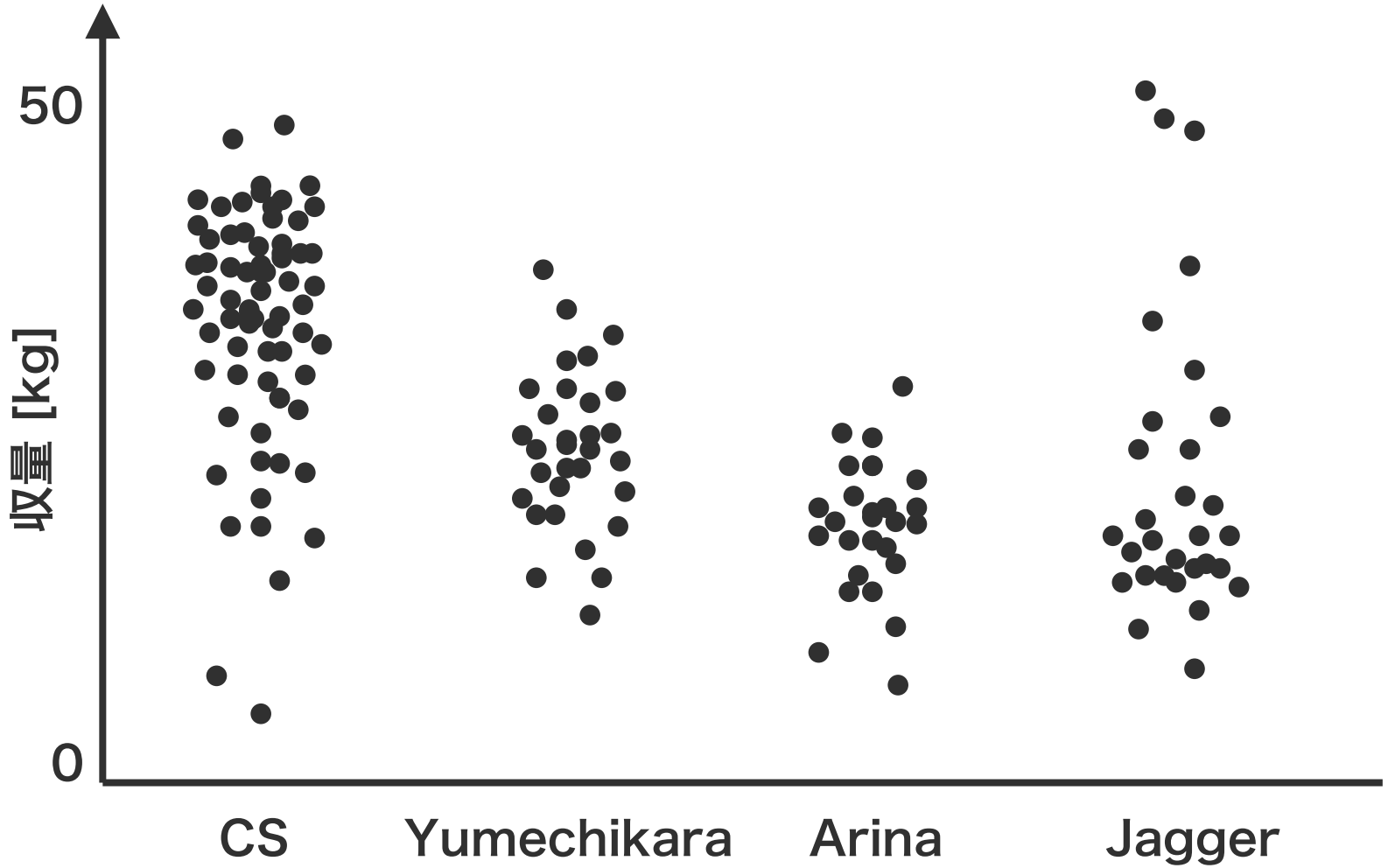

```
import numpy as  np
import matplotlib.pyplot as plt
np.random.seed(2018)

x1 = np.random.normal(10, 2, 20)
x2 = np.random.normal(15, 3, 20)
x3 = np.random.normal(5, 1, 20)

fig = plt.figure()
ax = fig.add_subplot()


ax.boxplot([x1, x2, x3],
           labels=['leaf', 'stem', 'root'])


ax.set_xlabel('tissue')
ax.set_ylabel('dry weight [g]')
ax.set_ylim(0, 20)
fig.show()
```

jitter



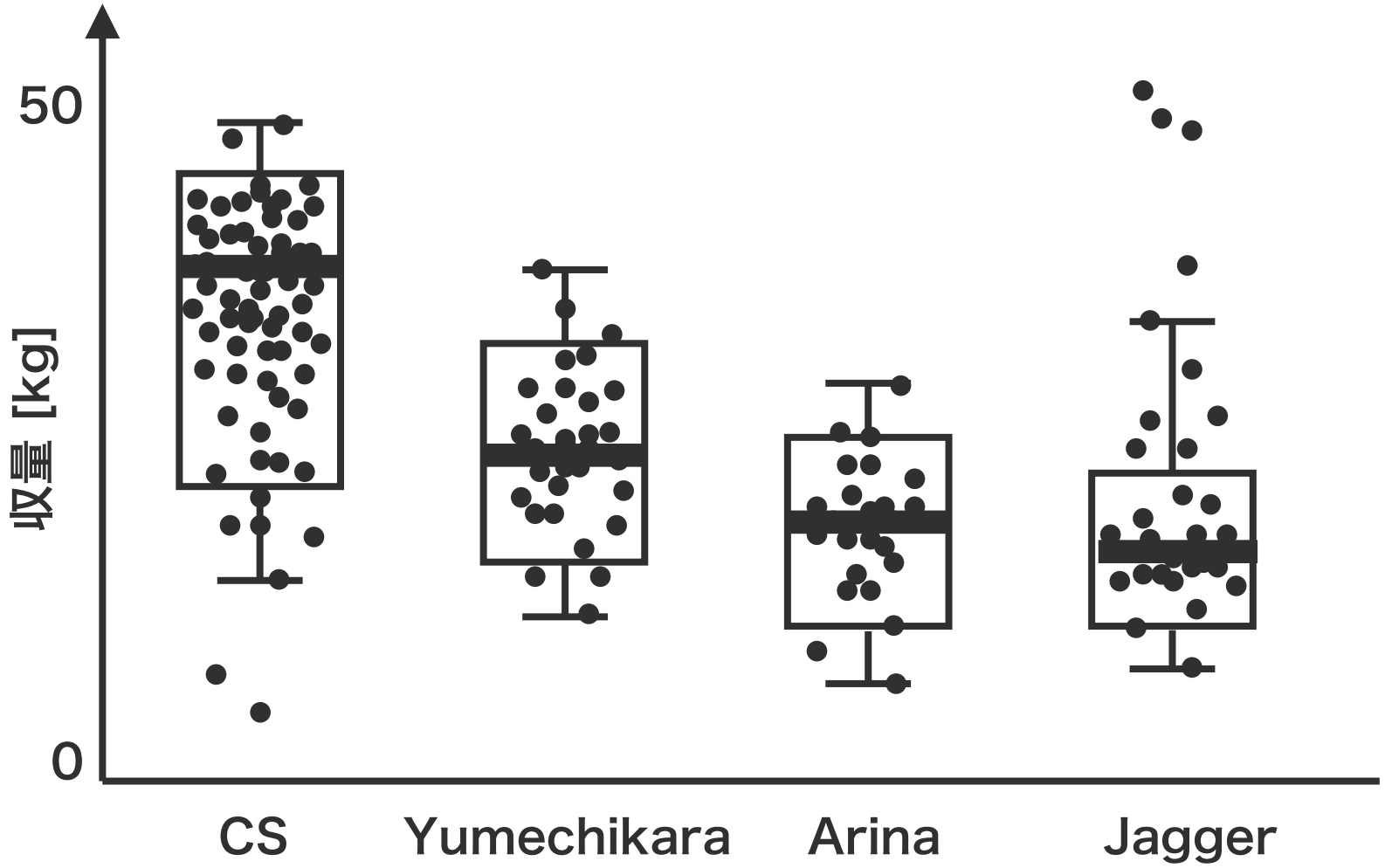
- 連続値データを可視化するためのグラフ
- 実際の値をプロットすると、データが多いところが重なるため、点を左右にランダムにずらしてプロットする
- ボックスプロットと合わせて使われる場合もある

jitter



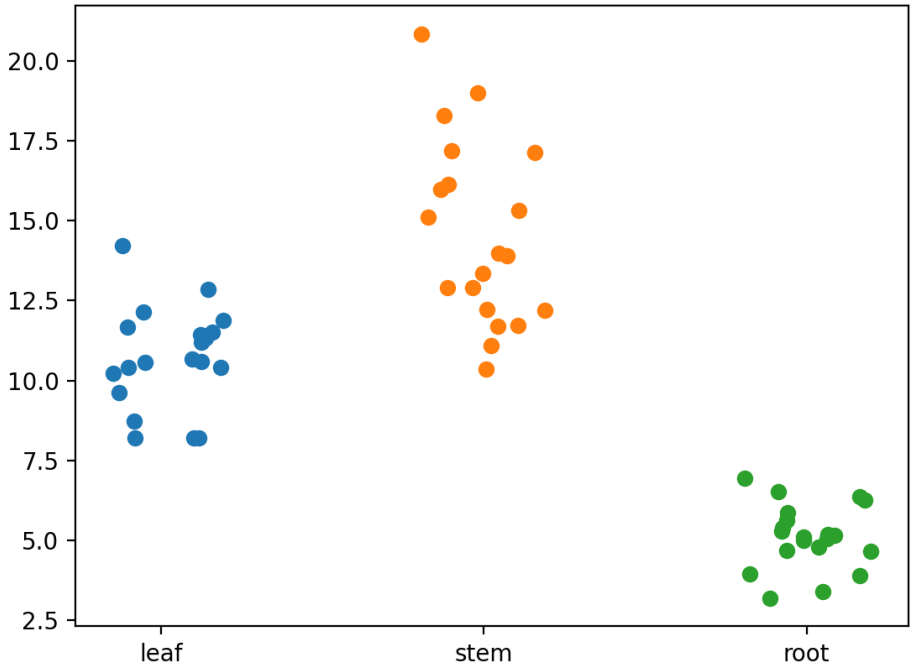
- 連続値データを可視化するためのグラフ
- 実際の値をプロットすると、データが多いところが重なるため、点を左右にランダムにずらしてプロットする
- ボックスプロットと合わせて使われる場合もある

jitter



- 連続値データを可視化するためのグラフ
- 実際の値をプロットすると、データが多いところが重なるため、点を左右にランダムにずらしてプロットする
- ボックスプロットと合わせて使われる場合もある

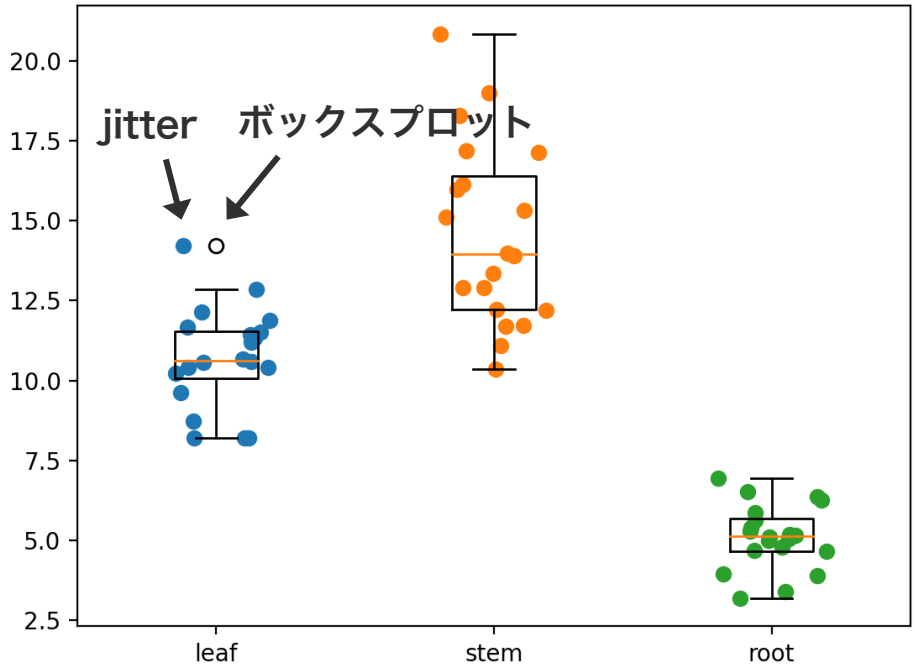
ボックスプロット




```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(112358)

y1 = np.random.normal(10, 2, 20)
y2 = np.random.normal(15, 3, 20)
y3 = np.random.normal(5, 1, 20)

x1 = 1 + np.random.uniform(-0.2, 0.2, len(y1))
x2 = 2 + np.random.uniform(-0.2, 0.2, len(y2))
x3 = 3 + np.random.uniform(-0.2, 0.2, len(y3))

fig = plt.figure()
ax = fig.add_subplot()
ax.scatter(x1, y1)
ax.scatter(x2, y2)
ax.scatter(x3, y3)
ax.set_xticks([1, 2, 3])
ax.set_xticklabels(['leaf', 'stem', 'root'])
fig.show()
```

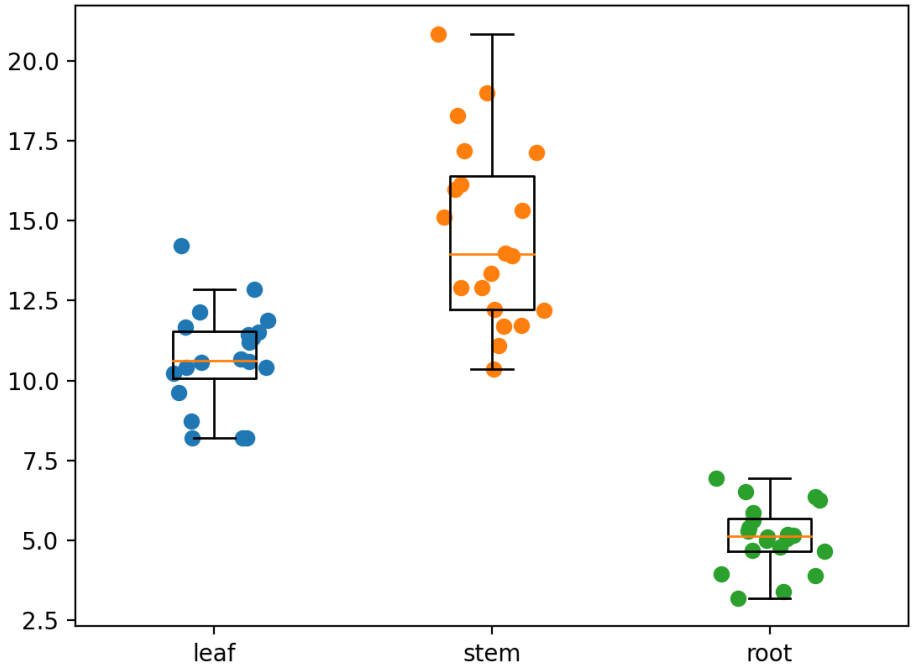
y1、y2、y3 のデータの y 座標をそのままにして、x 座標に乱数を加えて左右にずらし。

ボックスプロット




```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(112358)

y1 = np.random.normal(10, 2, 20)
y2 = np.random.normal(15, 3, 20)
y3 = np.random.normal(5, 1, 20)

x1 = 1 + np.random.uniform(-0.2, 0.2, len(y1))
x2 = 2 + np.random.uniform(-0.2, 0.2, len(y2))
x3 = 3 + np.random.uniform(-0.2, 0.2, len(y3))

fig = plt.figure()
ax = fig.add_subplot()
ax.scatter(x1, y1)
ax.scatter(x2, y2)
ax.scatter(x3, y3)
ax.boxplot([y1, y2, y3],
           labels=['leaf', 'stem', 'root'])
fig.show()
```

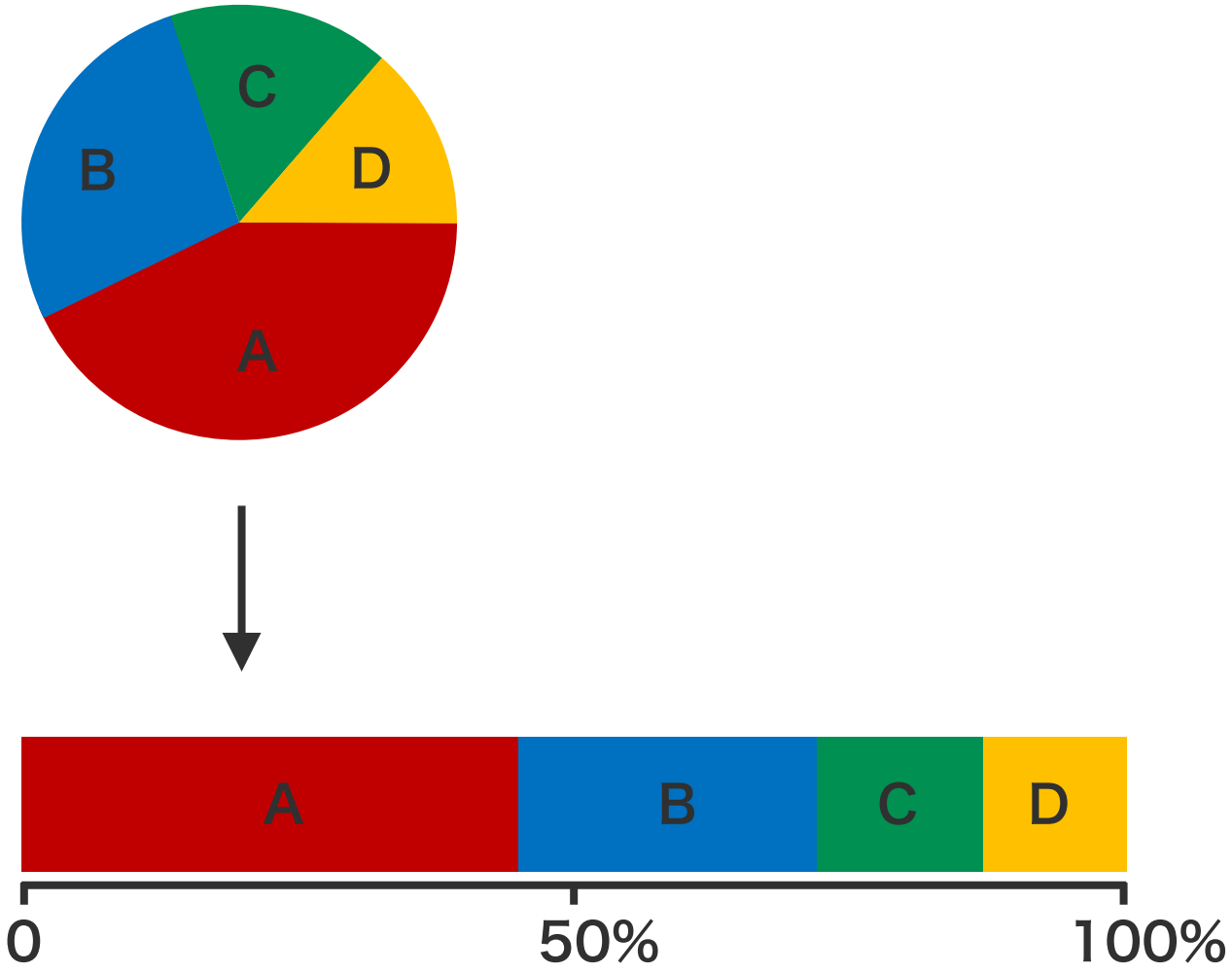
ボックスプロット




```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(112358)

y1 = np.random.normal(10, 2, 20)
y2 = np.random.normal(15, 3, 20)
y3 = np.random.normal(5, 1, 20)

x1 = 1 + np.random.uniform(-0.2, 0.2, len(y1))
x2 = 2 + np.random.uniform(-0.2, 0.2, len(y2))
x3 = 3 + np.random.uniform(-0.2, 0.2, len(y3))

fig = plt.figure()
ax = fig.add_subplot()
ax.scatter(x1, y1)
ax.scatter(x2, y2)
ax.scatter(x3, y3)
ax.boxplot([y1, y2, y3], showfliers=False,
           labels=['leaf', 'stem', 'root'])
fig.show()
```

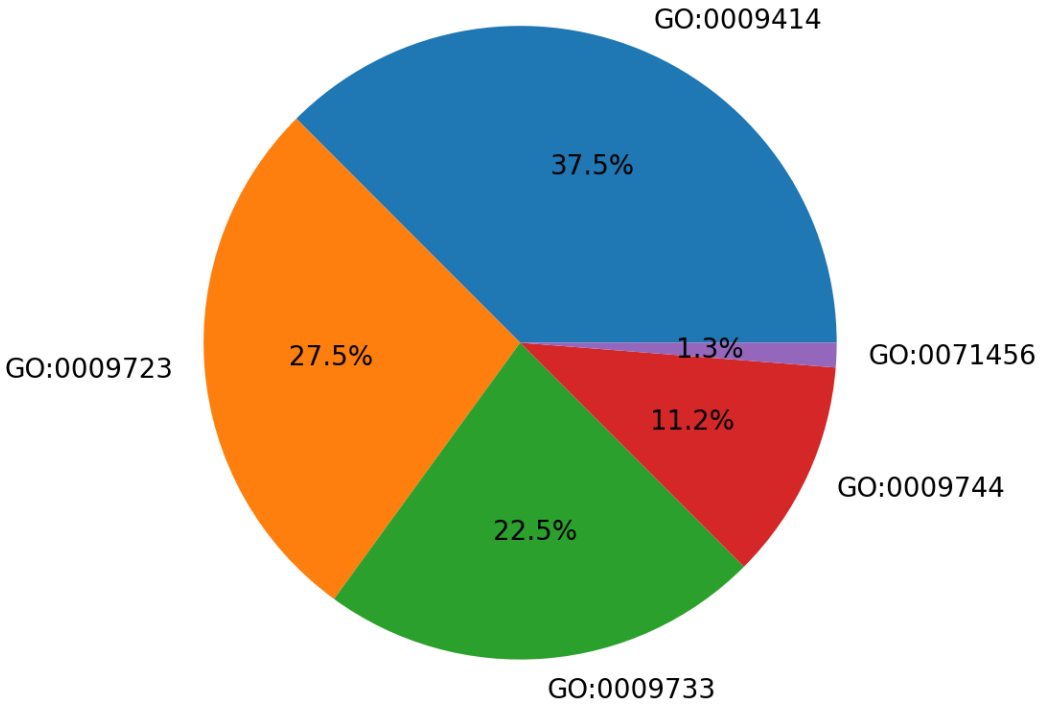
showfliers=False を指定することで、boxplot メソッドは外れ値を描かなくなる。

円グラフ



- 円グラフは誤解されやすいため、他のグラフで代替できる場合は、なるべく円グラフを使わない方が無難
- 人を騙す目的であれば、積極的に円グラフを用いるとよい
- さらに騙し効果を上げるために、3D 円グラフ、歪んだ円グラフあるいは合計が 100% にならない円グラフなどを用いるとよい

円グラフ

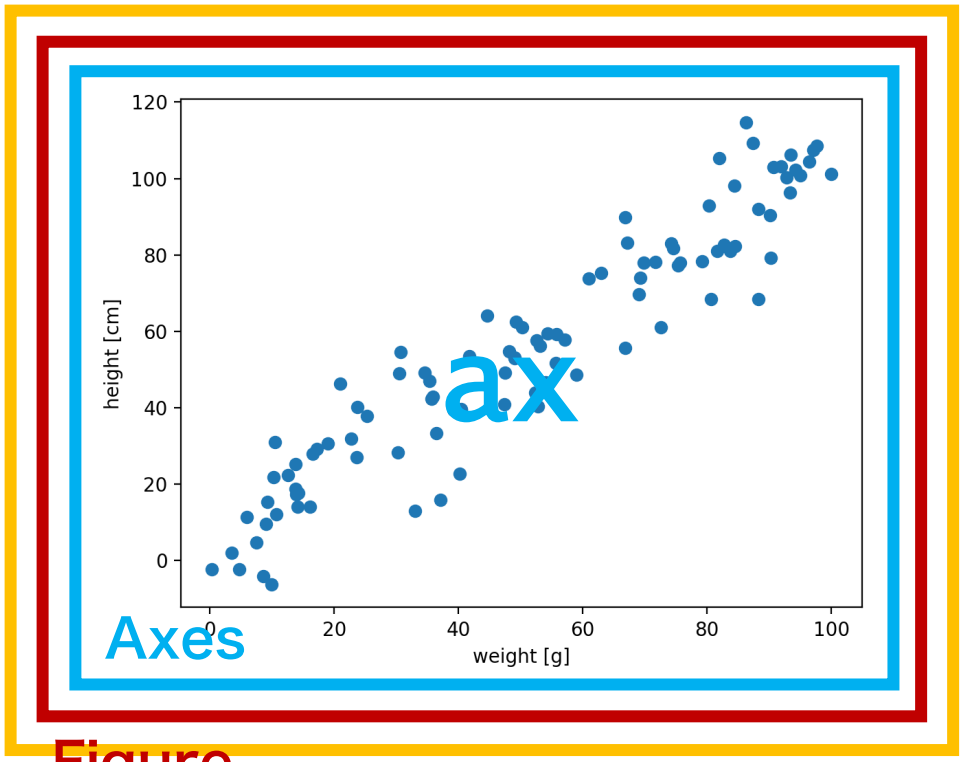



```
import matplotlib.pyplot as plt
import numpy as np

x = ['GO:0009414', 'GO:0009723',
     'GO:0009733', 'GO:0009744',
     'GO:0071456']

y = np.array([300, 220, 180, 90, 10])


fig = plt.figure()

ax = fig.add_subplot()

ax.pie(y, labels=x, autopct="%1.1f%%")

ax.axis('equal')

fig.show()
```

可視化

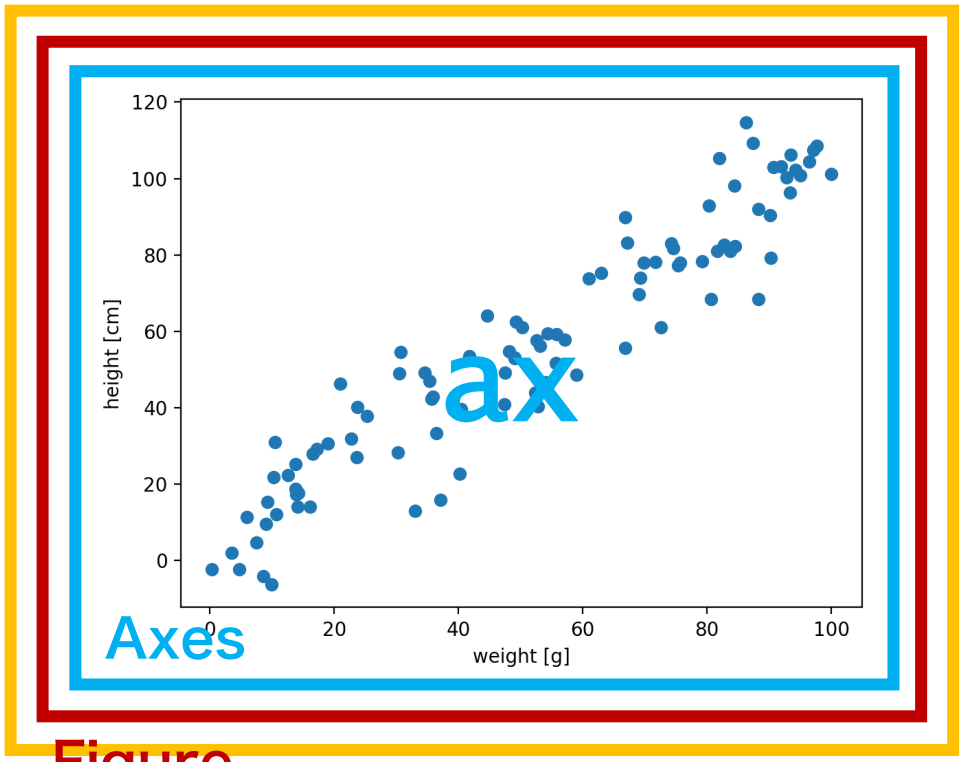
- matplotlib 基本
- 基本グラフ
- プロット領域の分割

subplot



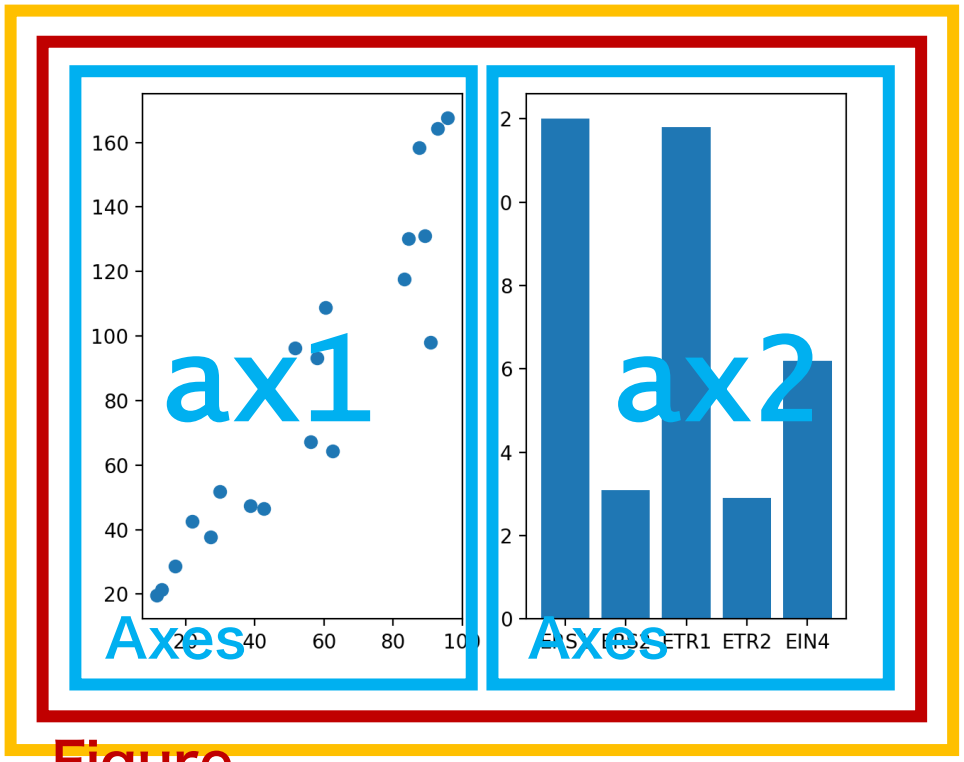

```
import matplotlib.pyplot as plt
import numpy as np

np.random.seed(2018)
x = np.random.uniform(0, 100, 100)
y = x + np.random.normal(5, 10, 100)
fig = plt.figure()

ax = fig.add_subplot()

ax.scatter(x, y)
ax.set_xlabel('weight [g]')
ax.set_ylabel('height [cm]')
fig.show()
```

subplot



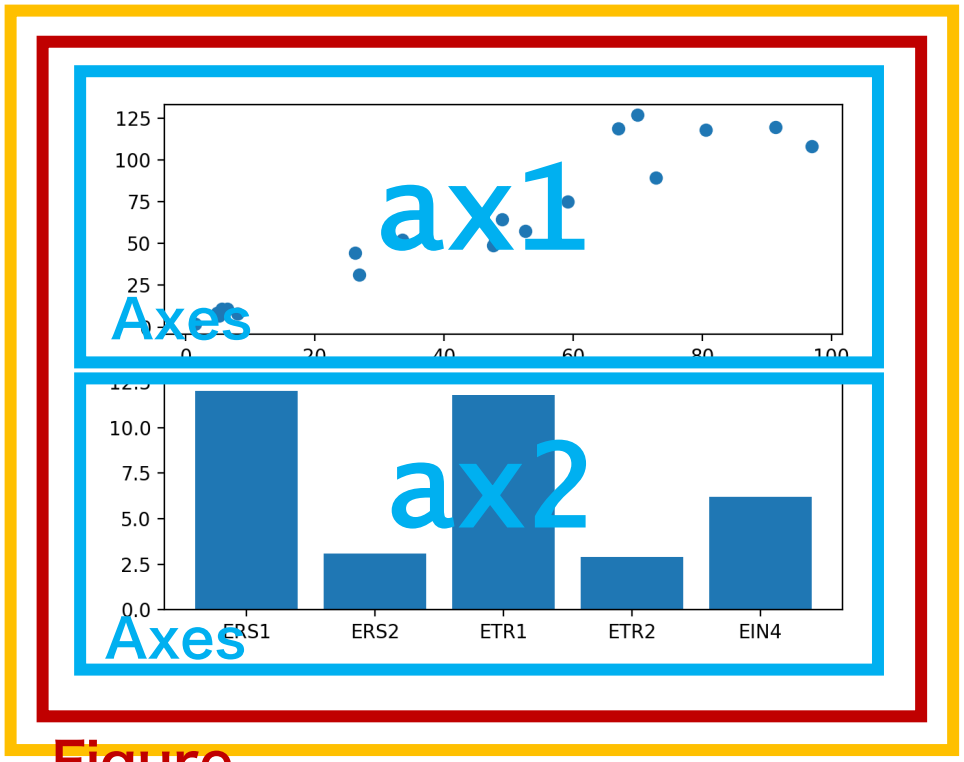

```
import matplotlib.pyplot as plt
import numpy as np

np.random.seed(2018)
x = np.random.uniform(0, 100, 100)
y = x + np.random.normal(5, 10, 100)
fig = plt.figure()

ax = fig.add_subplot(1, 1, 1)

ax.scatter(x, y)
ax.set_xlabel('weight [g]')
ax.set_ylabel('height [cm]')
fig.show()
```

subplot



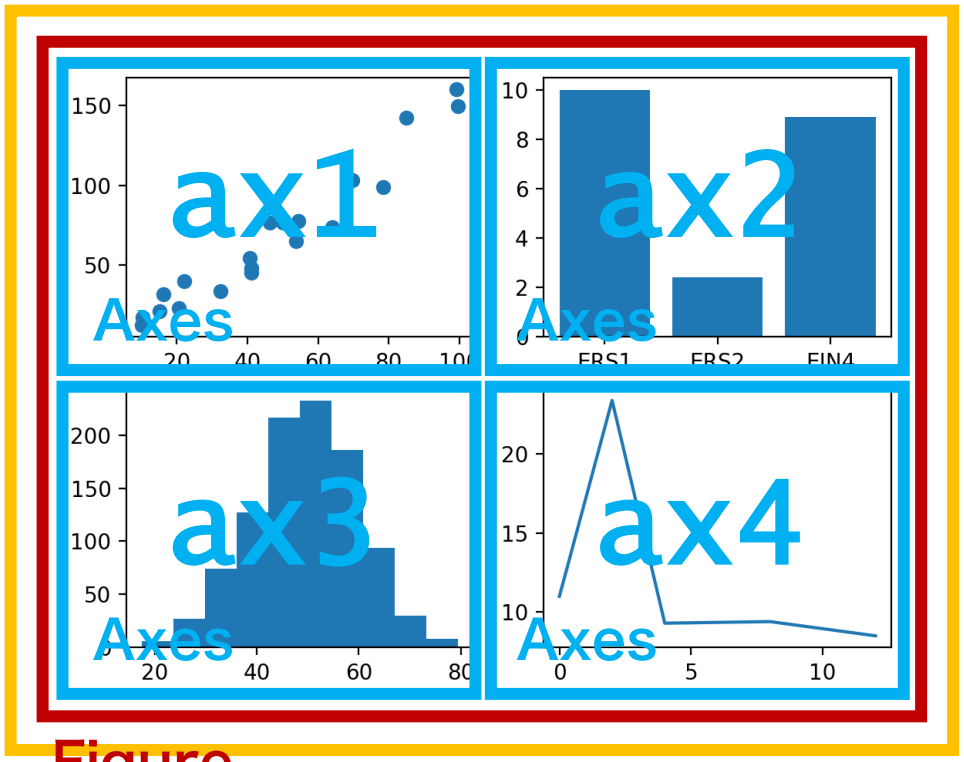

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
fig = plt.figure()
x1 = np.random.uniform(0, 100, 20)
y1 = x1 * np.random.uniform(1, 2, 20)

ax1 = fig.add_subplot(1, 2, 1)
ax1.scatter(x1, y1)
x2 = np.array(['ERS1', 'ERS2', 'ETR1',
               'ETR2', 'EIN4'])
y2 = np.array([12.0, 3.1, 11.8, 2.9, 6.2])
x2_position = np.arange(len(x2))

ax2 = fig.add_subplot(1, 2, 2)
ax2.bar(x2_position, y2, tick_label=x2)
fig.show()
```

subplot



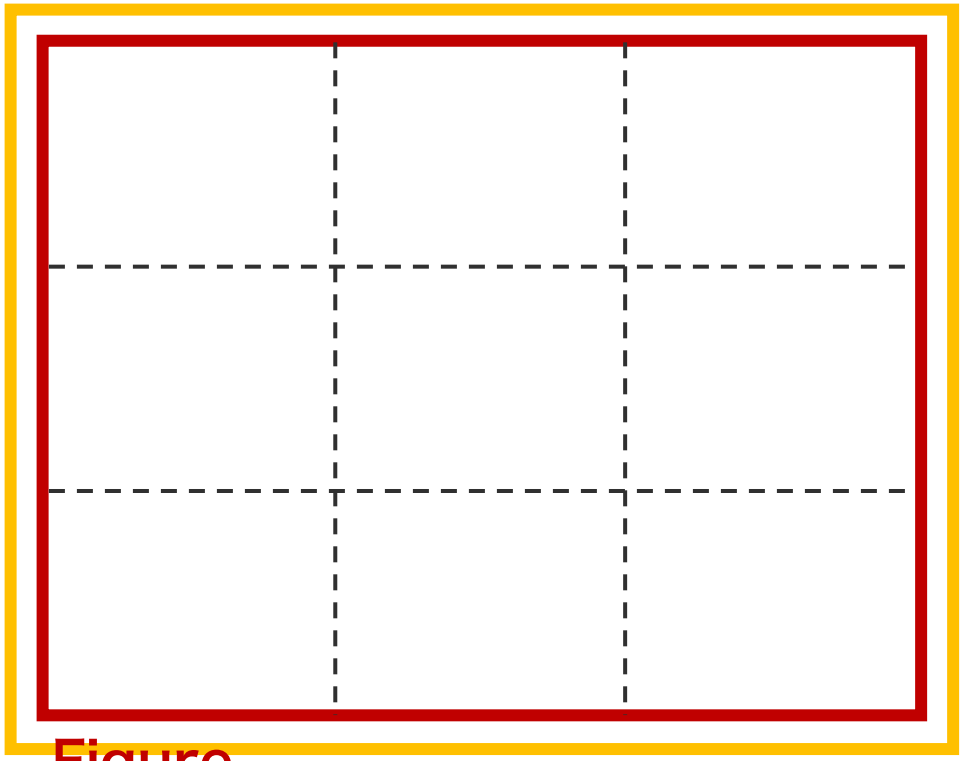

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
fig = plt.figure()
x1 = np.random.uniform(0, 100, 20)
y1 = x1 * np.random.uniform(1, 2, 20)

ax1 = fig.add_subplot(2, 1, 1)
ax1.scatter(x1, y1)
x2 = np.array(['ERS1', 'ERS2', 'ETR1',
               'ETR2', 'EIN4'])
y2 = np.array([12.0, 3.1, 11.8, 2.9, 6.2])
x2_position = np.arange(len(x2))

ax2 = fig.add_subplot(2, 1, 2)
ax2.bar(x2_position, y2, tick_label=x2)
fig.show()
```

subplot




```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
x1 = np.random.uniform(0, 100, 20)
y1 = x1 * np.random.uniform(1, 2, 20)
x2 = np.array(['ERS1', 'ERS2', 'EIN4'])
y2 = np.array([10.0, 2.4, 8.9])
x3 = np.random.normal(50, 10, 1000)
x4 = np.array([0, 2, 4, 8, 12])
y4 = np.array([11.0, 23.4, 9.3, 9.4, 8.5])
fig = plt.figure()
ax1 = fig.add_subplot(2, 2, 1)
ax1.scatter(x1, y1)
ax2 = fig.add_subplot(2, 2, 2)
ax2.bar(x2, y2)
ax3 = fig.add_subplot(2, 2, 3)
ax3.hist(x3)
ax4 = fig.add_subplot(2, 2, 4)
ax4.plot(x4, y4)
fig.show()
```

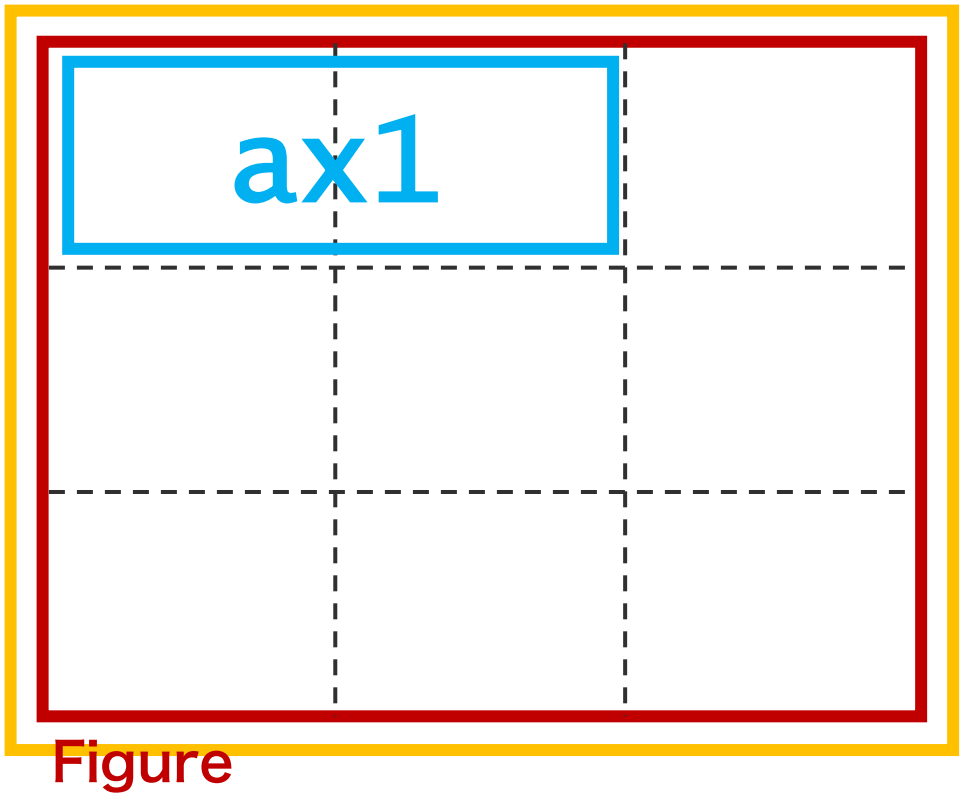
GridSpec



GridSpec を使用すると、描画領域をグリッド状に分けたのちに、隣接するグリッド同士を結合させることによって、描画領域を任意の形に分割できる。

```
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec

gs = gridspec.GridSpec(3, 3)
```

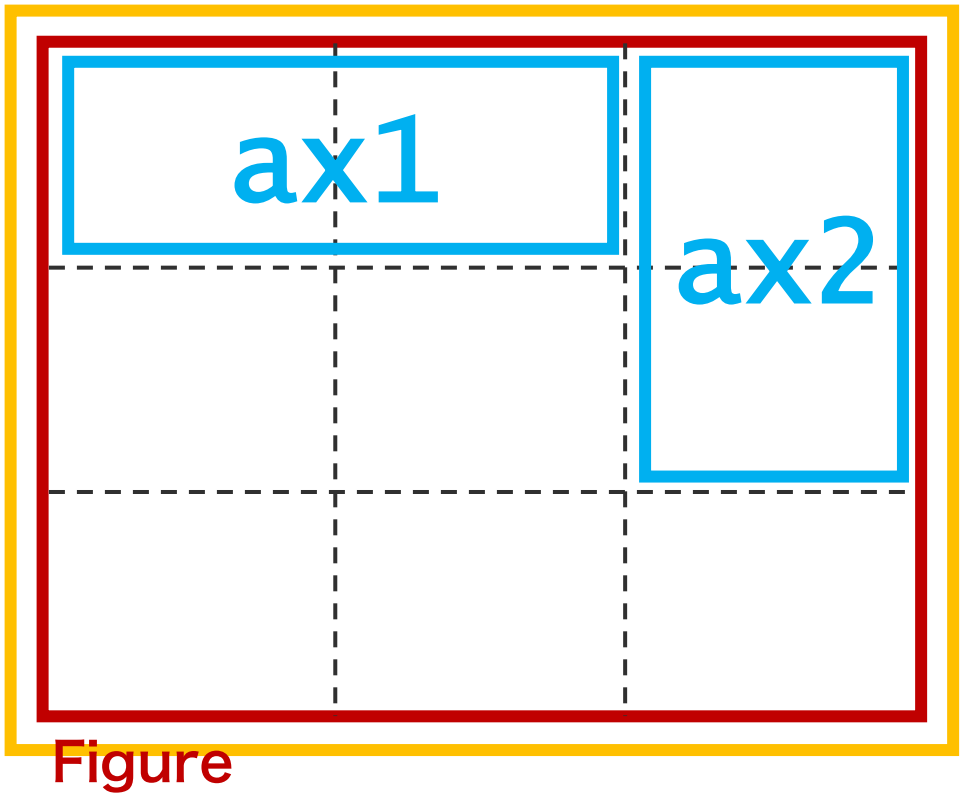
GridSpec



GridSpec を使用すると、描画領域をグリッド状に分けたのちに、隣接するグリッド同士を結合させることによって、描画領域を任意の形に分割できる。

```
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec

gs = gridspec.GridSpec(3, 3)

ax1 = plt.subplot(gs[0, 0:2])
```

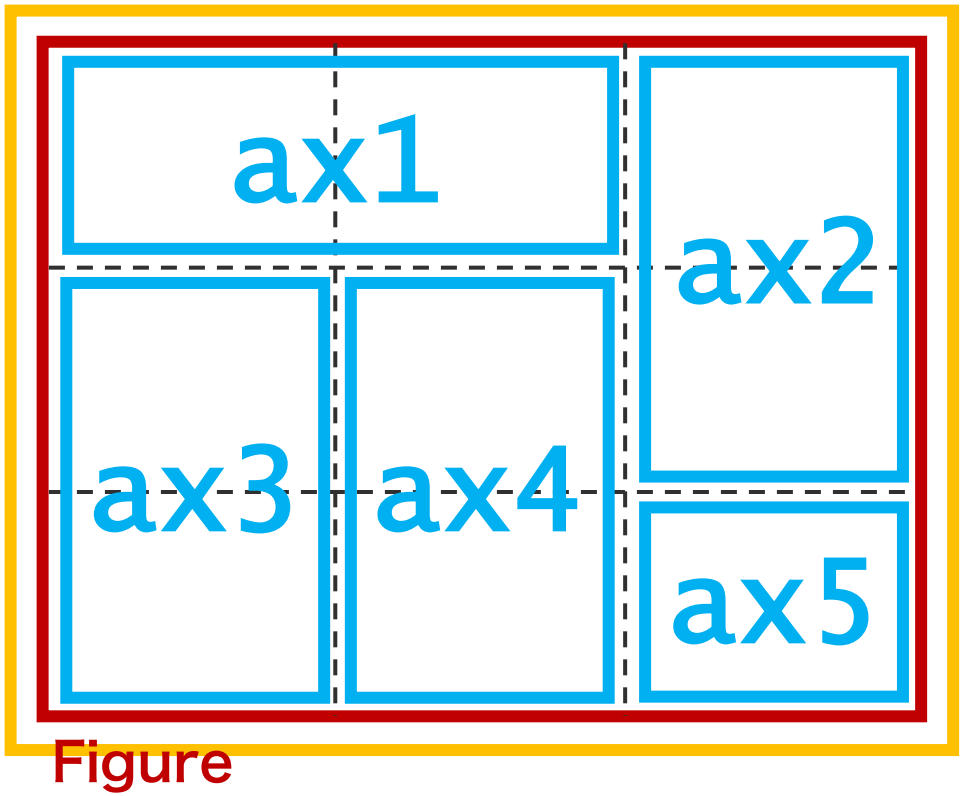
GridSpec



GridSpec を使用すると、描画領域をグリッド状に分けたのちに、隣接するグリッド同士を結合させることによって、描画領域を任意の形に分割できる。

```
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec

gs = gridspec.GridSpec(3, 3)

ax1 = plt.subplot(gs[0, 0:2])
ax2 = plt.subplot(gs[0:2, 2])
```

GridSpec



GridSpec を使用すると、描画領域をグリッド状に分けたのちに、隣接するグリッド同士を結合させることによって、描画領域を任意の形に分割できる。

```
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec

gs = gridspec.GridSpec(3, 3)

ax1 = plt.subplot(gs[0, 0:2])
ax2 = plt.subplot(gs[0:2, 2])
ax3 = plt.subplot(gs[1:3, 0])
ax4 = plt.subplot(gs[1:3, 1])
ax5 = plt.subplot(gs[2, 2])

ax1.plot()
ax2.plot()
ax3.plot()
ax4.plot()
ax5.plot()

plt.tight_layout()
plt.show()
```