農学生命情報科学特論 I

ICT や IoT 等の先端技術を活用し、効率よく高品質生産を可能にするスマート農業への取り組みは世界的に進められています。その基礎を支えている技術の一つがプログラミング言語。なかでも、習得しやすくかつ応用範囲の広い Python がとくに注目されています。本科目では、農学や分子生物学などの分野で利用される Python の最新事例を紹介しながら、Python の基礎文法の講義を行います。

孫　建強　https://aabbdd.jp/
農研機構・農業情報研究センター

June
08 17:15–20:30

基本構文

第 1 回目の授業では、プログラミング言語の基本であるデータ構造とアルゴリズムを簡単に紹介してから、Python の基本構文を紹介する。Python のスカラー、リスト、ディクショナリ、条件構文と繰り返し構文を取り上げる。

June
15 17:15–20:30

文字列処理

バイオインフォマティックスの分野において、塩基配列やアミノ酸配列などの文字列からなるデータを扱うことが多い。第 2 回目の授業では、Python を利用した文字列処理を紹介し、FASTA や GFF などのファイルから情報を抽出する方法を取り上げる。

June
22 17:15–20:30

データ解析

Python で CSV や TSV ファイルに保存された数値データを扱うときに、NumPy や Pandas ライブラリーを使用する。第 3 回目の授業では、NumPy や Pandas の機能を紹介し、ベクトルや行列のデータ処理を中心に取り上げる。

June
29 17:15–20:30

データ可視化

Python で数値データを可視化するときに matplotlib ライブラリーを使用する。第 4 回目の授業では、matplotlib の基本的な使い方を紹介し、全回の授業を復習しながらデータの可視化を行う。

成績評価

- 授業の始めに出席確認を 1 回だけを行う。出席 1 回につき 1 点を与える。
- 出席確認後、レポート課題を示す。毎回 3 問出題し、授業全体を通して合計 12 問出題する。1 問 8 点とする。
- 授業を聞かなくても、レポート課題を解けそうであれば、遠慮なく退室していただいて構いません。

レポート課題の提出方法

- レポート課題を Jupyter Notebook / Jupyter Lab 上で解き、そのファイル（.ipynb）を メールで提出する。

  ✉ report＠aabbdd.jp

- 注意事項
  - メールの件名を「特論課題1」としてください。
  - メールの本文に何も書かないでください。
  - ファイルの名前を学生証番号（例：310000A.ipynb）としてください。
  - ファイル中に、名前・所属・研究内容などの個人情報・機密情報を書かないでください。個人情報・機密情報を含むレポートを零点とする。

レポート課題（第 1 回）

問題 1

リスト a = [1, 4, 9, 8, 3] の中の最大値を求めて変数 m に代入するプログラムを書け。ただし、sort 関数、max 関数、min 関数を使わないこと。

問題 2

リスト a = [1, 4, 0, 5, 2, 3] の中の奇数の和を求めて変数 s に代入するプログラムを書け。

問題 3

100 以下の素数の和を求めて変数 s に代入するプログラムを書け。

農学生命情報科学特論Ⅰ

1

- プログラミング言語
- 基本オブジェクト
- 基本文法

農学生命情報科学特論Ⅰ

1

- 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

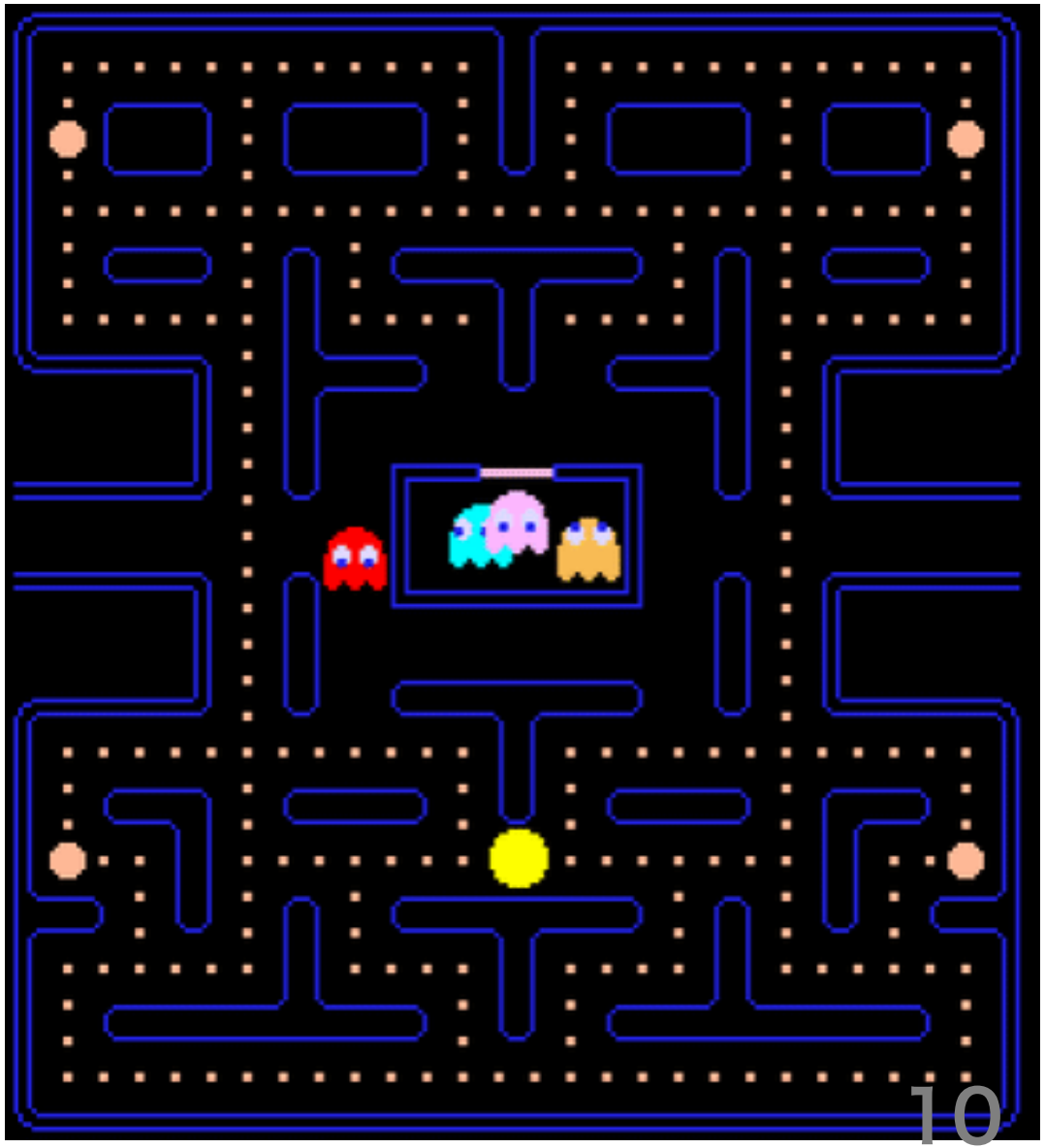
- プログラミング言語
- データ構造とアルゴリズム

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。

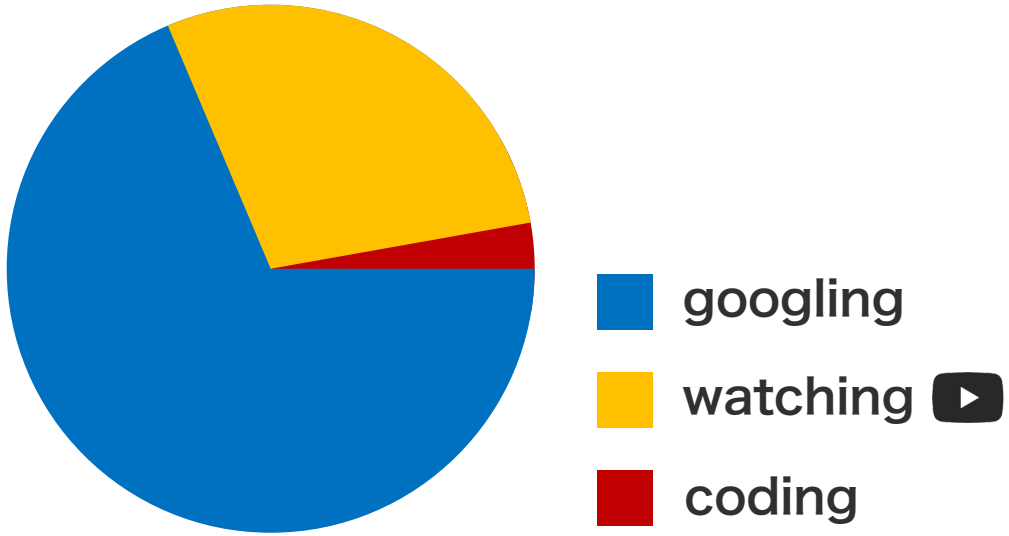
プログラミング言語とゲーム

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。

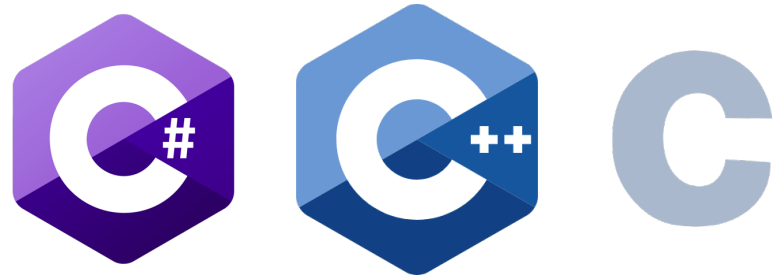


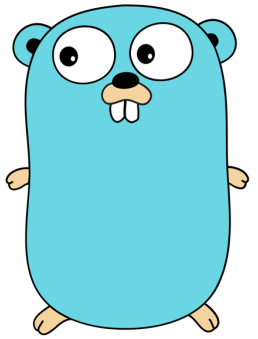
プログラミング言語

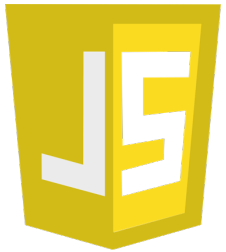
プログラミング言語を学ぶ上で、重要なのは理論的な思考力を身につけることである。ある作業をコンピューターに処理させたいとき、その作業を基本的な操作に分解し、それらを正しい順序で組み合わせ、コンピューターが理解できる形に置き換える必要がある。このような作業の分解や組み合わせなどが非常に大事である。一方で、プログラミングの文法や常用関数の名前や使い方は、あまり重要ではなく、都度に調べればよい。



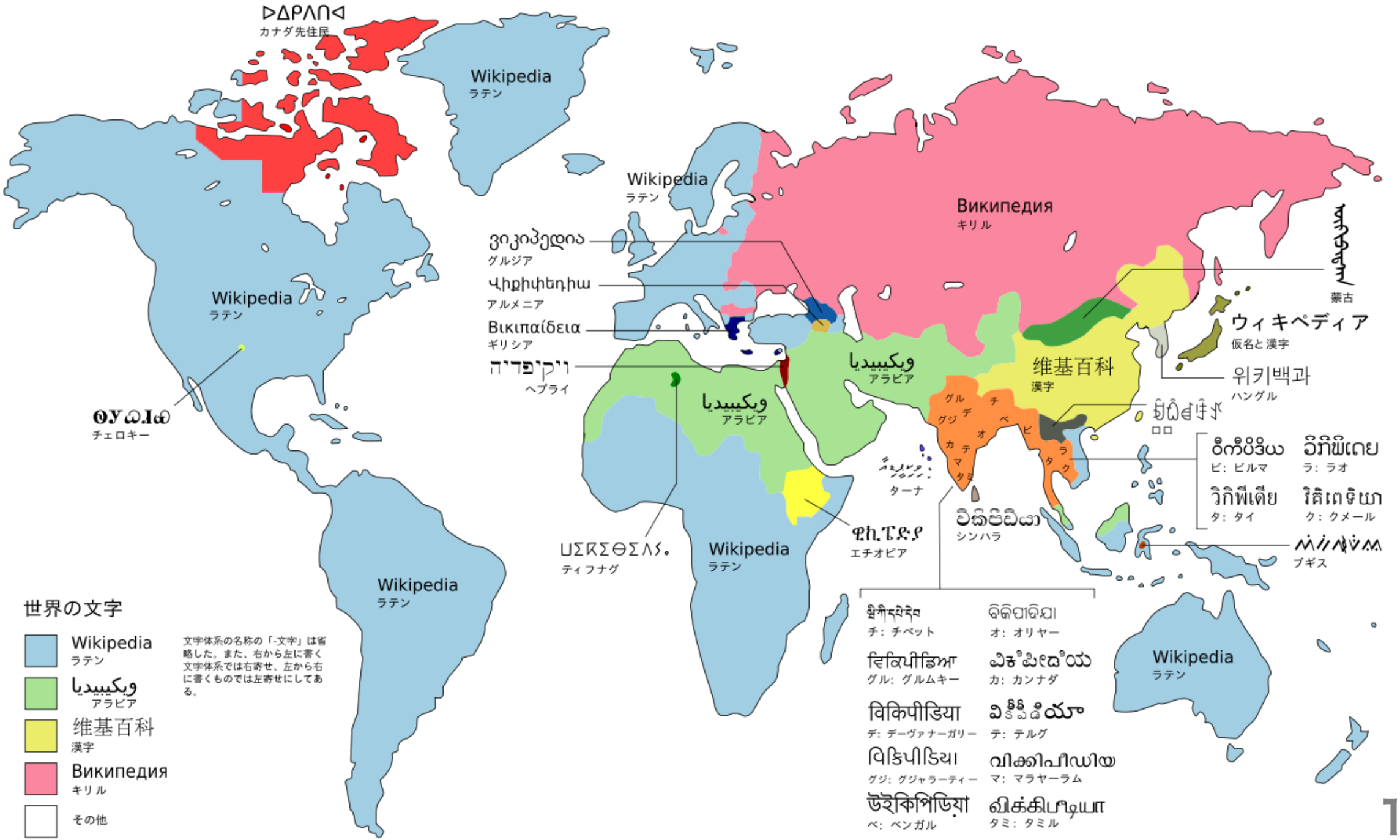
プログラミング言語

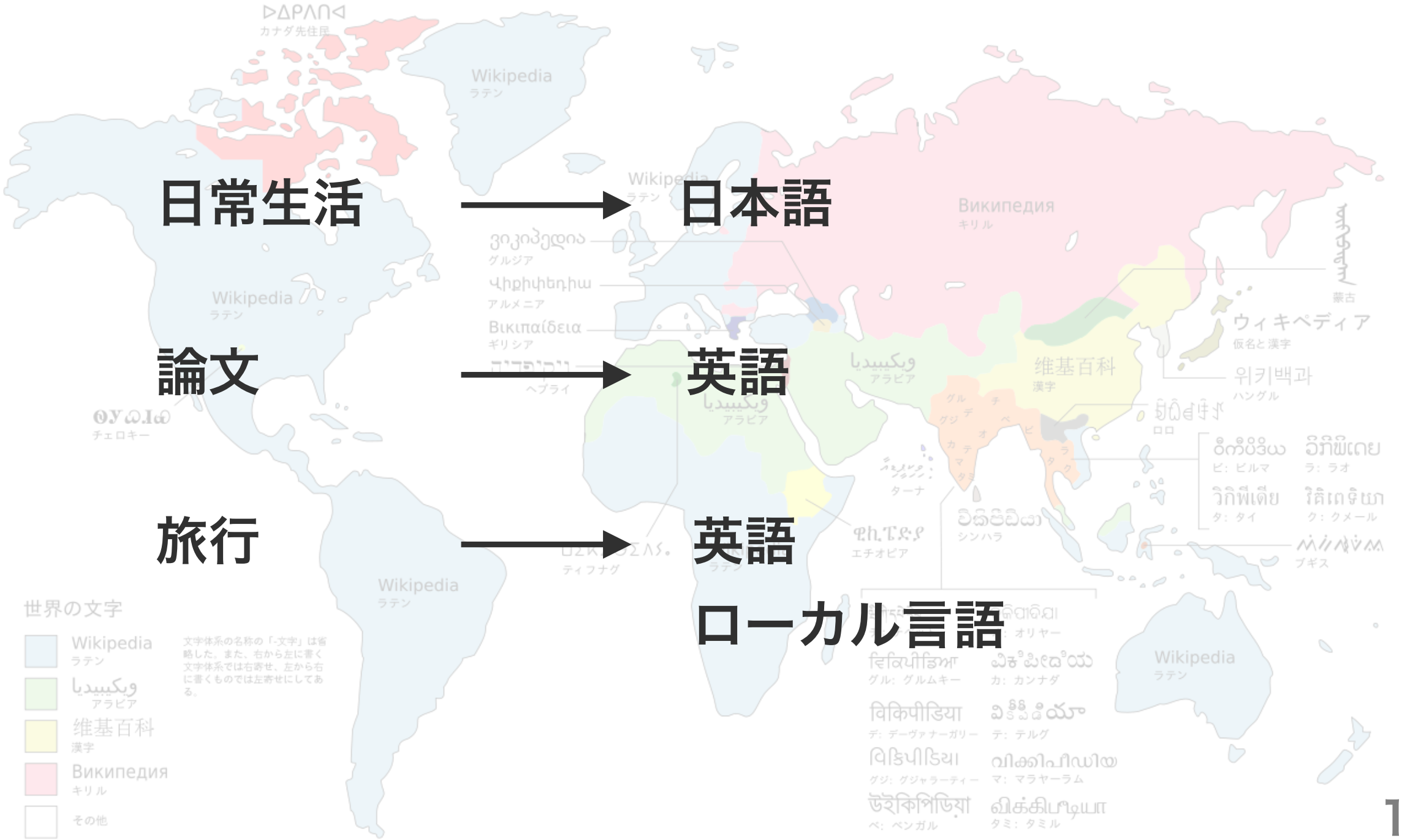






現在、数百種類以上のプログラミング言語が使われている。その機能に着目すると、様々な目的に使用できる汎用型言語と統計やウェブページ作成などの特定の目的で使用できる専用型言語に分けることができる。プログラミング言語同士に優劣はなく、目的に応じて使い分けされている。


ᐅᐃᑭᐱᓇᑦ
カナダ先住民
Wikipedia
ラテン
Wikipedia
ラテン
ᏫᎩᏪᎥᏯ
チェロキー
Wikipedia
ラテン
Wikipedia
ラテン
ვიკიპედია
グルジア
Վիքիփեդիա
アルメニア
Βικιπαίδεια
ギリシア
ויקיפדיה
ヘブライ
Википедия
キリル
ᠸᡳᡴᡳᡦᡝᡩᡳᠶᠠ
蒙古
ويكيبيديا
アラビア
维基百科
漢字
ウィキペディア
仮名と漢字
위키백과
ハングル
ويكيبيديا
アラビア
ᐃᑲᐱᑎᐃᐊ
ロロ
ဝီကီပီဒီယ
ビ：ビルマ
ᩅᩥᨠᩥᨻᩮᨯᩤ
タ：タイ
ᥕᥭᥒᥰᥖᥭᥑ
ブギス
ᐃᓕᕆᔨᒃ
エチオピア
Wikipedia
ラテン
ⵜⵉⴼⵉⵏⴰⵖ
ティフナグ
ދިވެހި
ターナ
විකිපීඩියා
シンハラ
Wikipedia
ラテン
ཆི་སྐད་དེབ་
チ：チベット
ଓଡ଼ିଆ
オ：オリヤー
ವಿಕಿಪೀಡಿಯ
カ：カンナダ
വിക്കിപീഡിയ
マ：マラヤーラム
ವಿಕಿಪೀಡಿಯಾ
タミ：タミル
世界の文字
Wikipedia
ラテン
ويكيبيديا
アラビア
维基百科
漢字
Википедия
キリル
その他
文字体系の名称の「-文字」は省略した。また、右から左に書く文字体系では右寄せ、左から右に書くものでは左寄せにしてある。


日常生活
日本語
論文
英語
旅行
英語
ローカル言語
世界の文字
Wikipedia
ラテン
Википедия
キリル
その他

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。

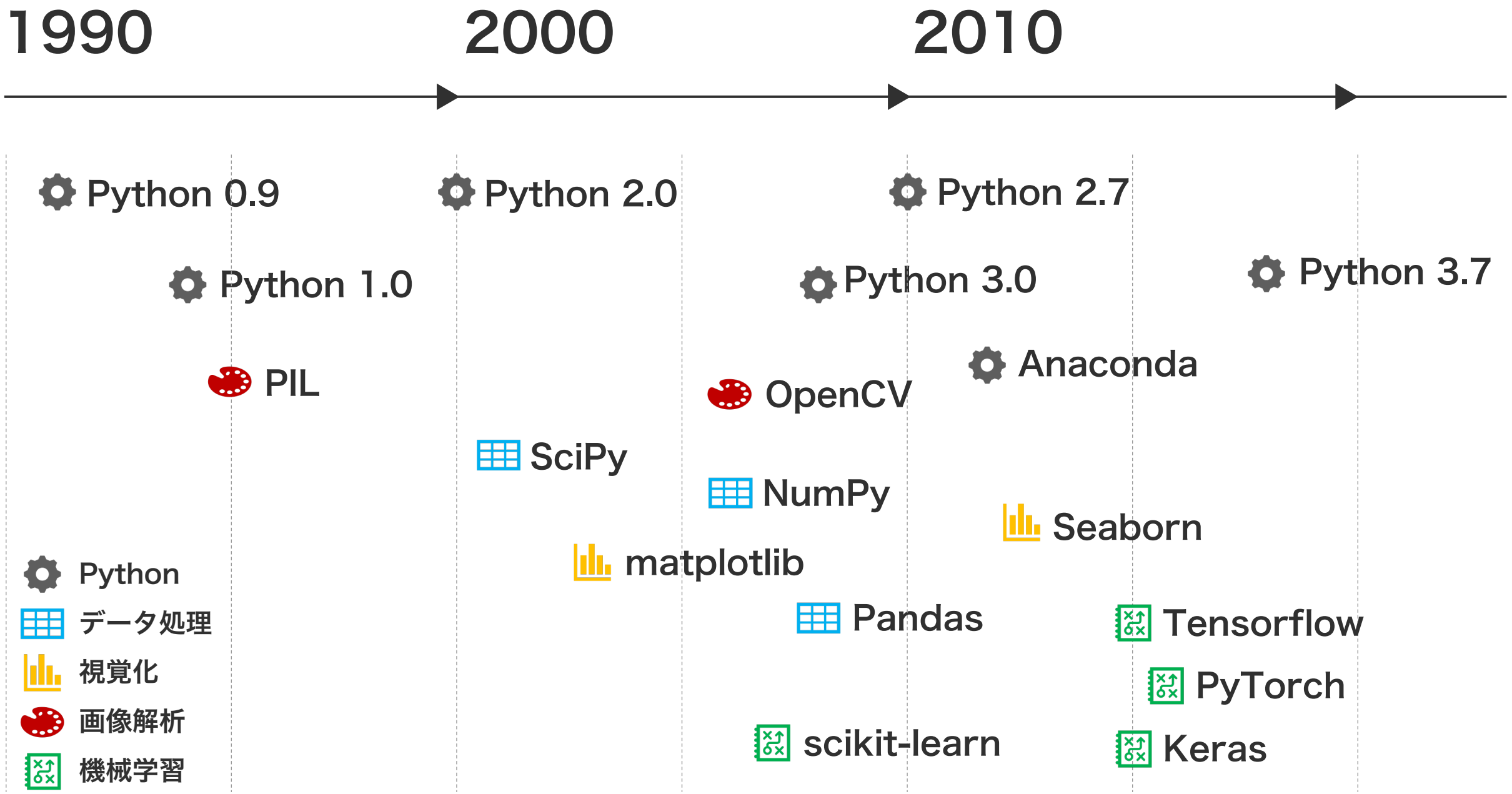
Python 2.7

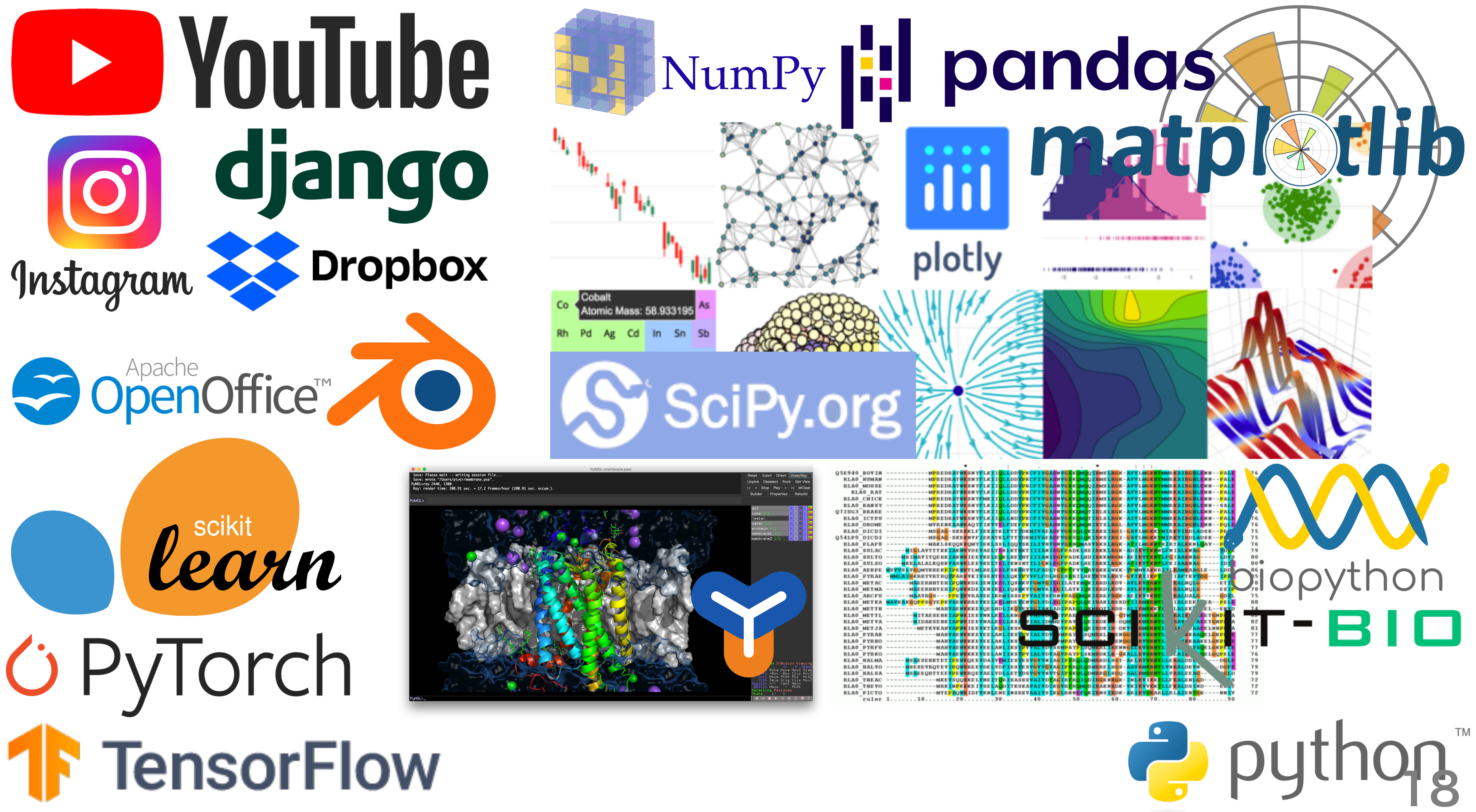
Python 3.0

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

Python 4.0 は 2023 年にリリース予定？


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

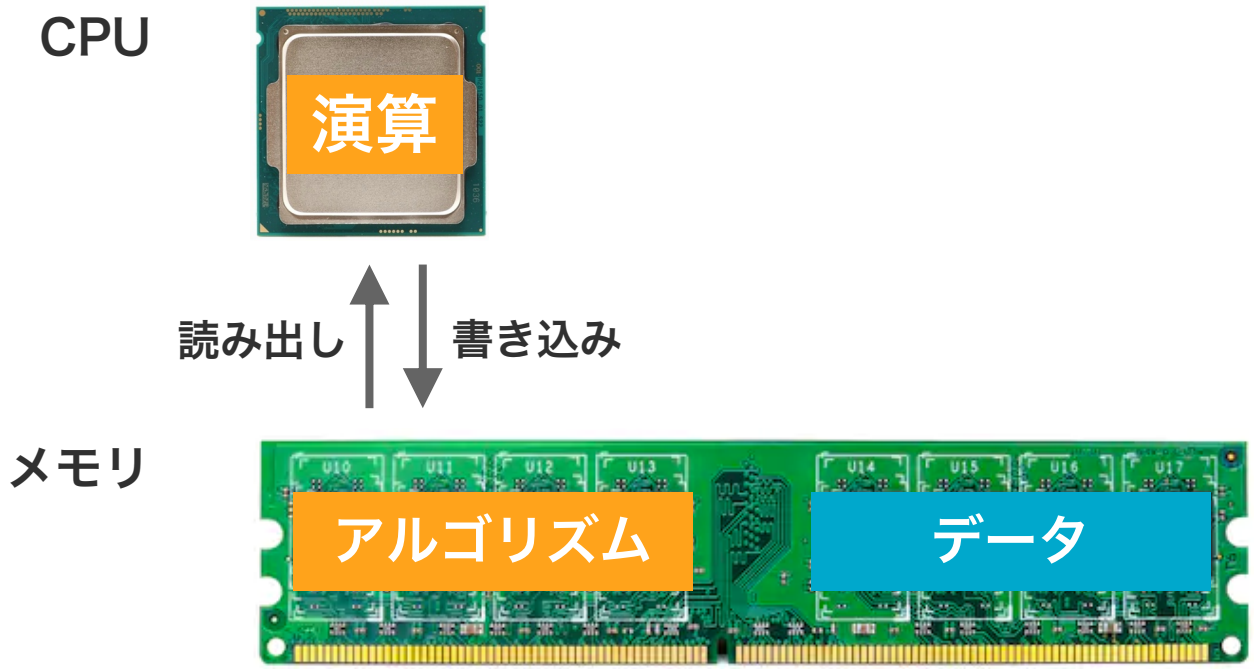

YouTube
NumPy
pandas
matplotlib
Instagram
django
Dropbox
plotly
Cobalt
Atomic Mass: 58.933195
Apache OpenOffice™
SciPy.org
scikit learn
PyTorch
TensorFlow
biopython
SCIKIT-BIO
python™

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

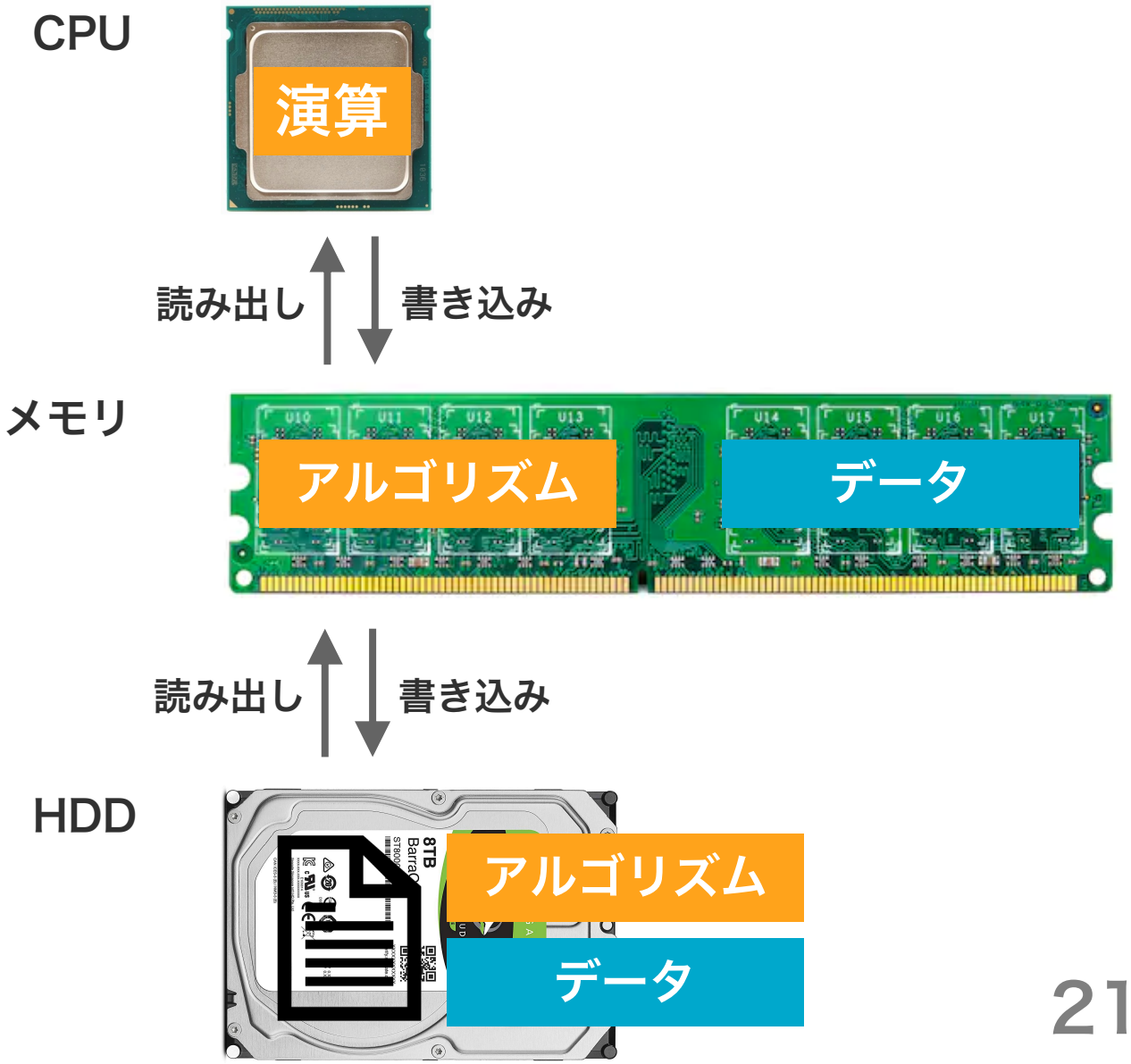
データ構造とアルゴリズム

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。



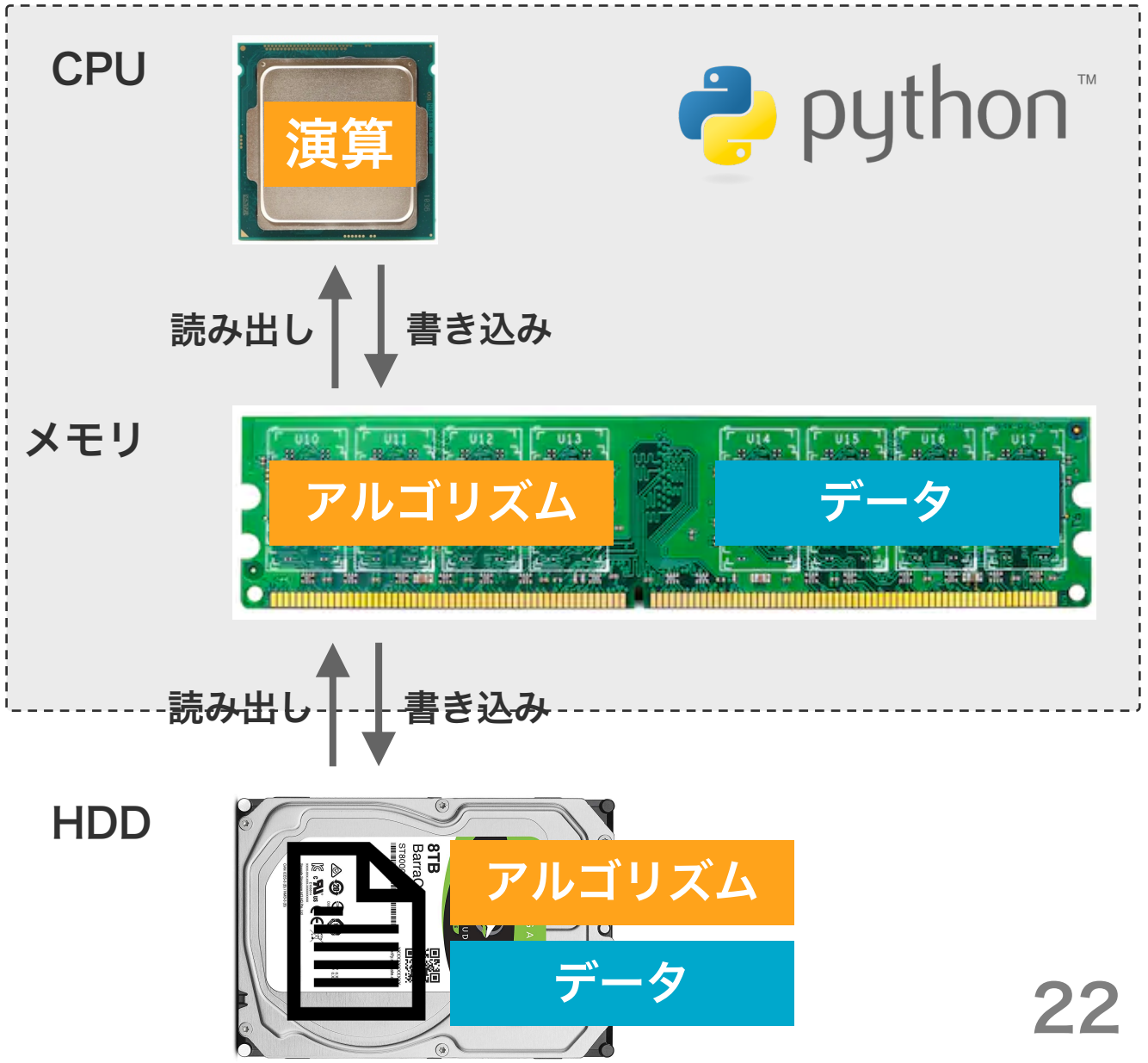
データ構造とアルゴリズム

メモリ上に保持された内容は、電源を切ると、すべて消える。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、アルゴリズムやデータ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。

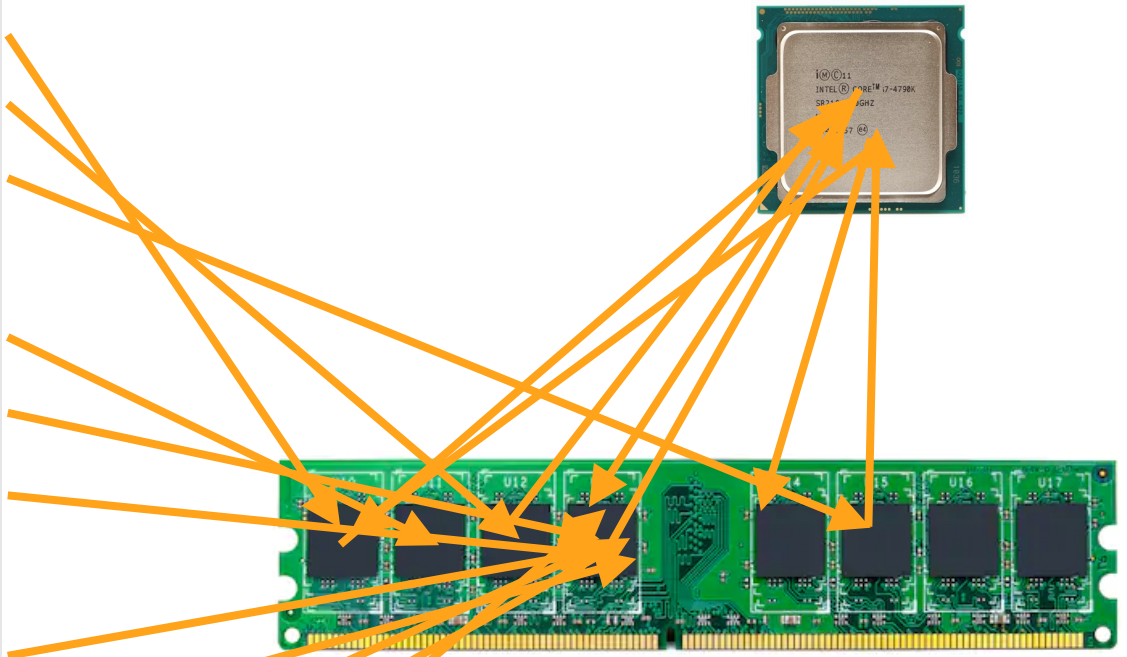


データ構造とアルゴリズム

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 →

- 並べ替え
- 探索
- 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

農学生命情報科学特論Ⅰ

1

- 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

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。

```
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。

各系統に対して、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

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

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。各系統に対して、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。

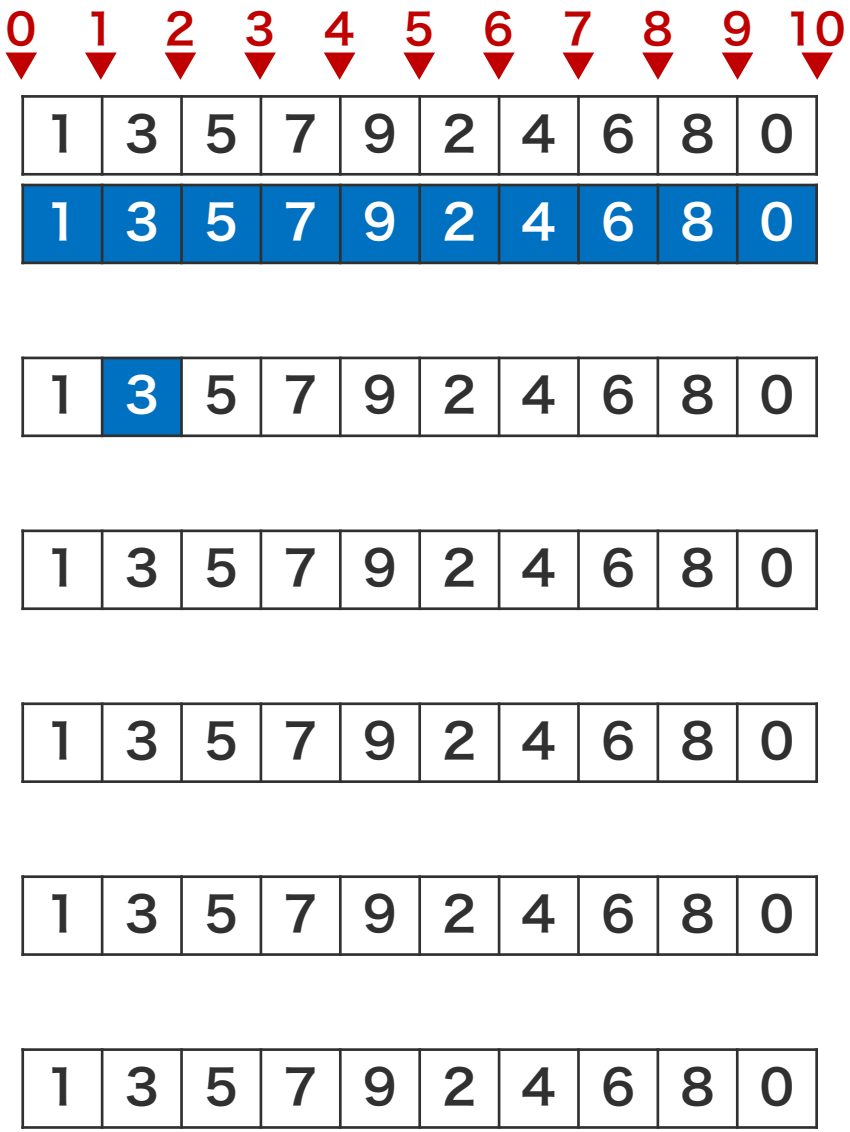
```
w = [12, 10, 11, 13, 11]


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。

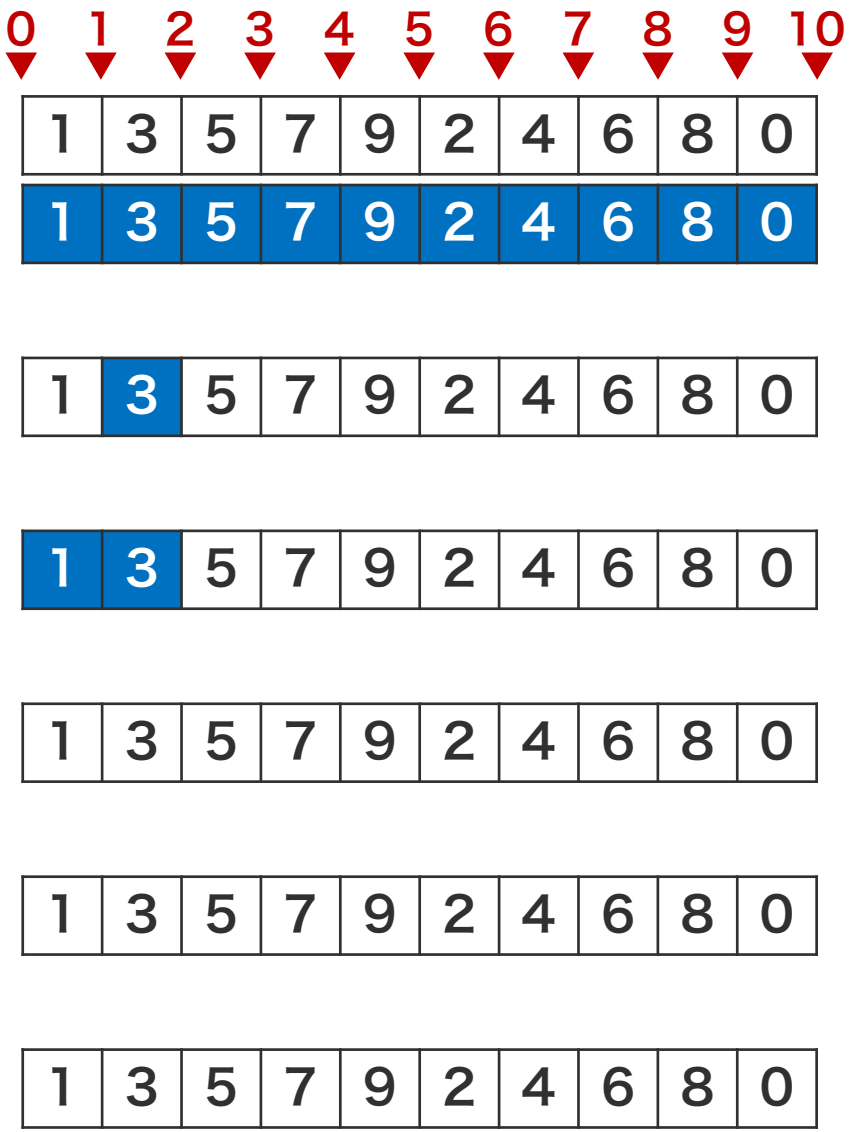
```
a = [1, 3, 5, 7, 9, 2, 4, 6, 8, 0]



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



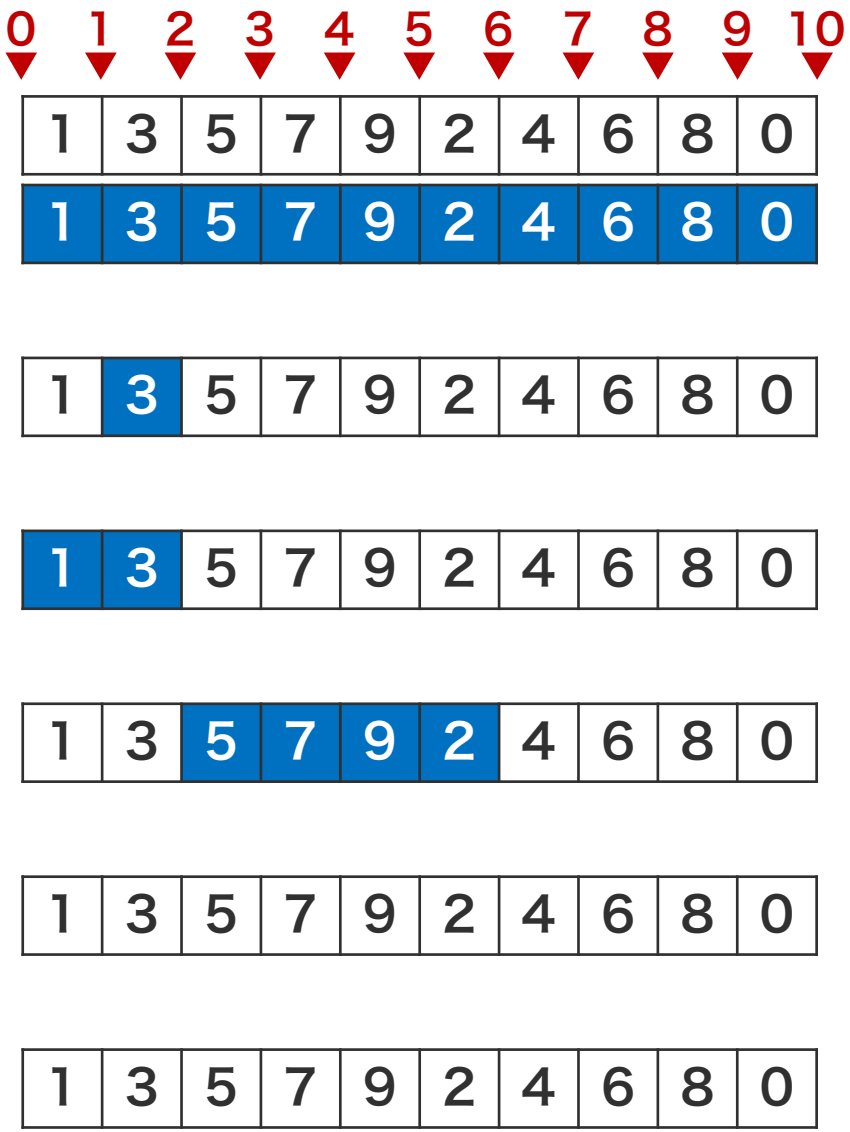

```
a = [1, 3, 5, 7, 9, 2, 4, 6, 8, 0]


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



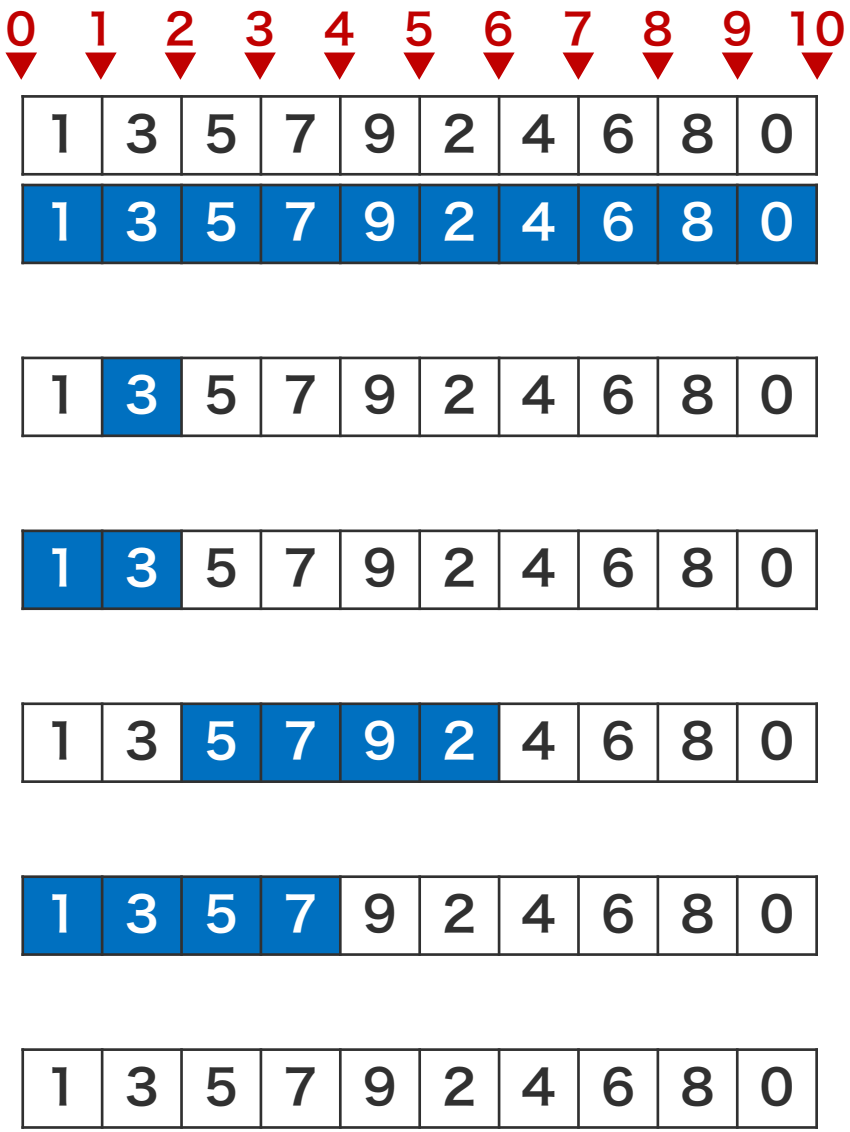

```
a = [1, 3, 5, 7, 9, 2, 4, 6, 8, 0]


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



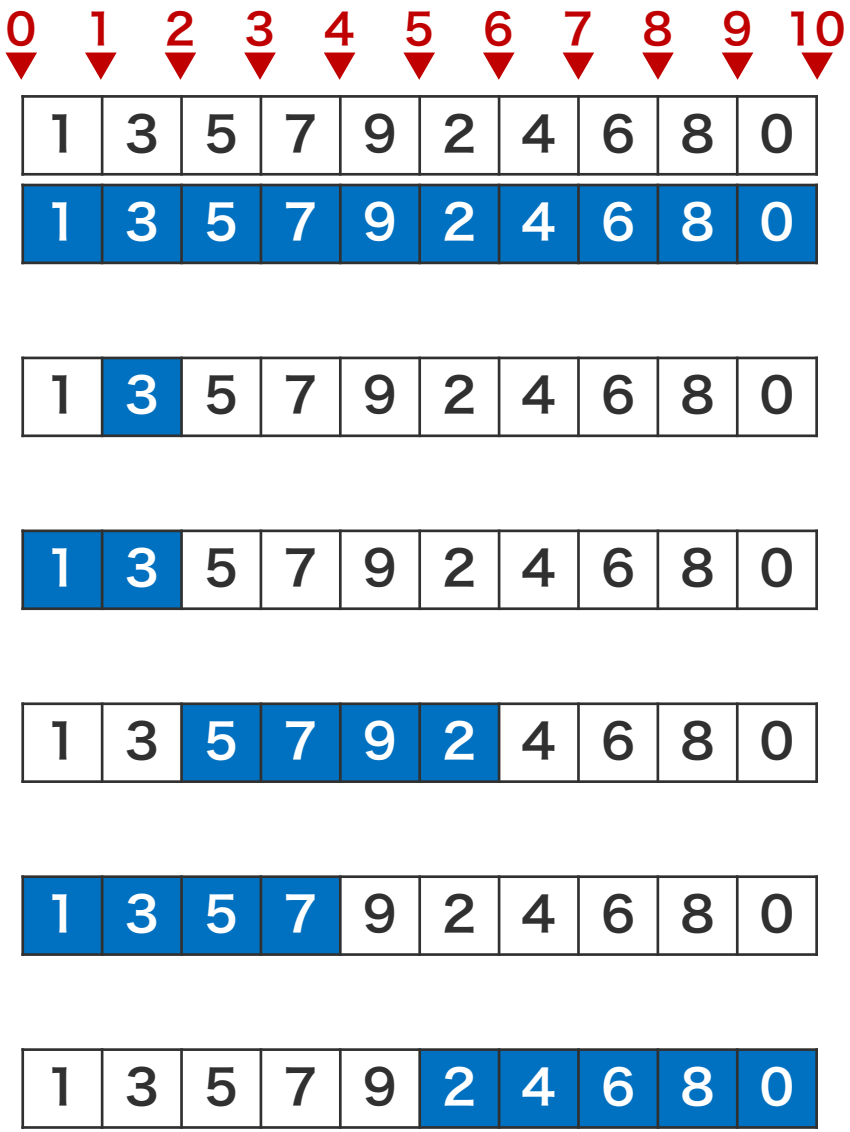

```
a = [1, 3, 5, 7, 9, 2, 4, 6, 8, 0]


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]
```

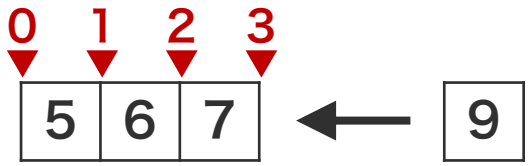
リスト操作

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。

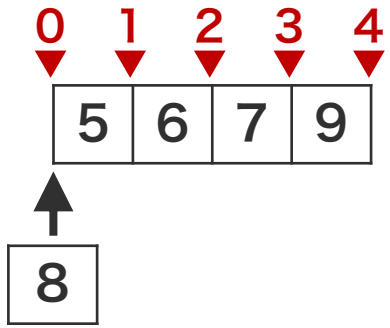
```
a = [5, 6, 7]


a.append(9)
a


a.insert(0, 8)
a


a.insert(2, 1)
a


a.append(4)
a
```

リスト操作



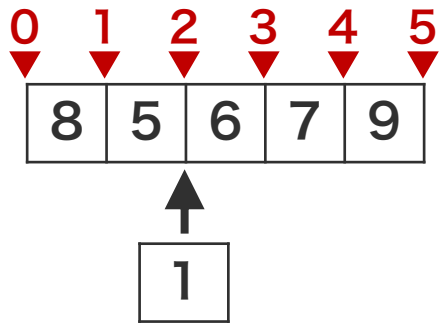

```
a = [5, 6, 7]

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



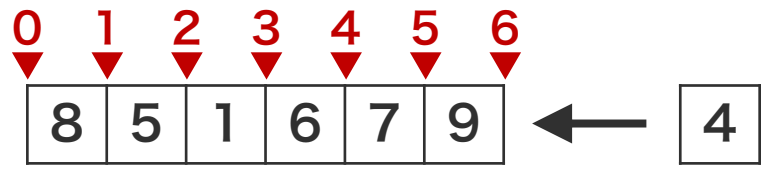

```
a = [5, 6, 7]

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
```

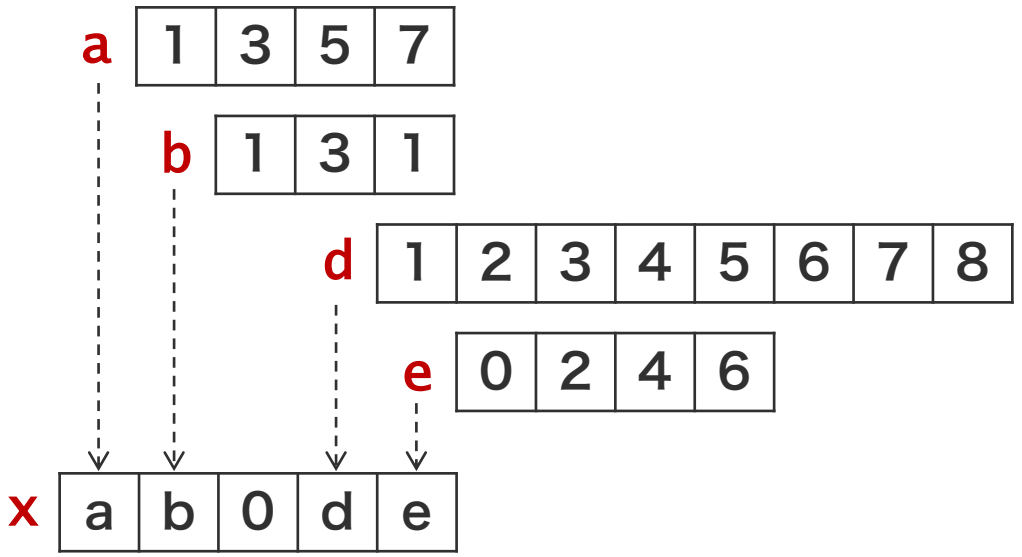
リスト操作




```
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。

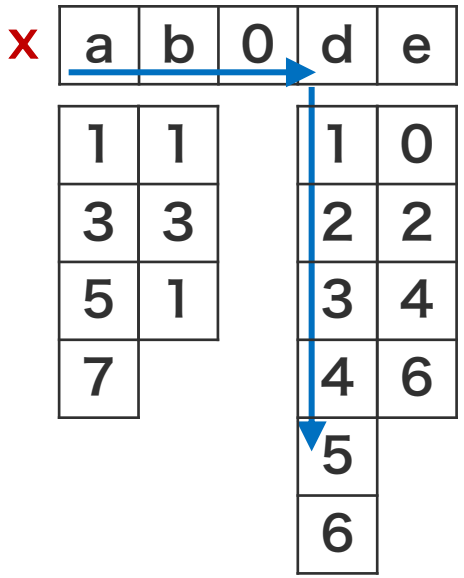



```
a = [1, 3, 5, 7]
b = [1, 3, 1]
d = [1, 2, 3, 4, 5, 6, 7, 8]
e = [0, 2, 4, 6]

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 SO-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。

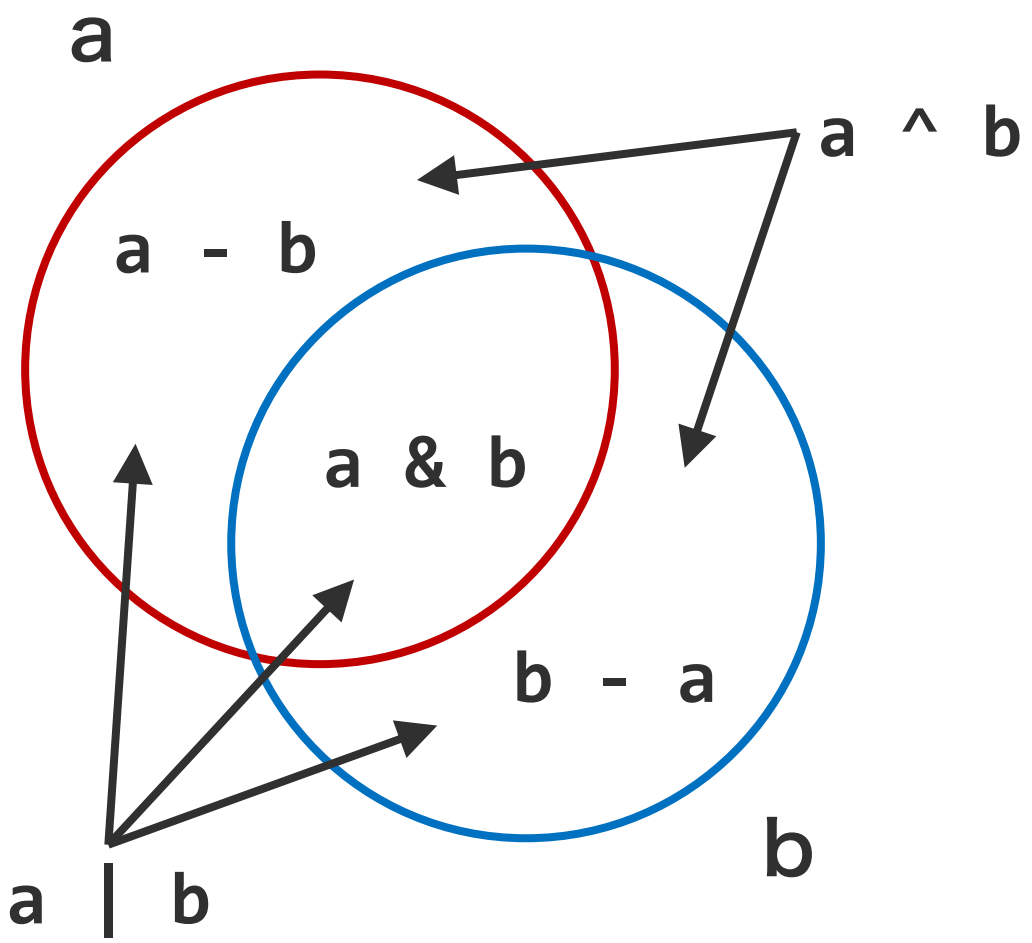
順序集合を取り扱う場合は、collections パッケージ中の OrderedDict 型のオブジェクトを使用する。この資料では順序集合を取り上げない。

```
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

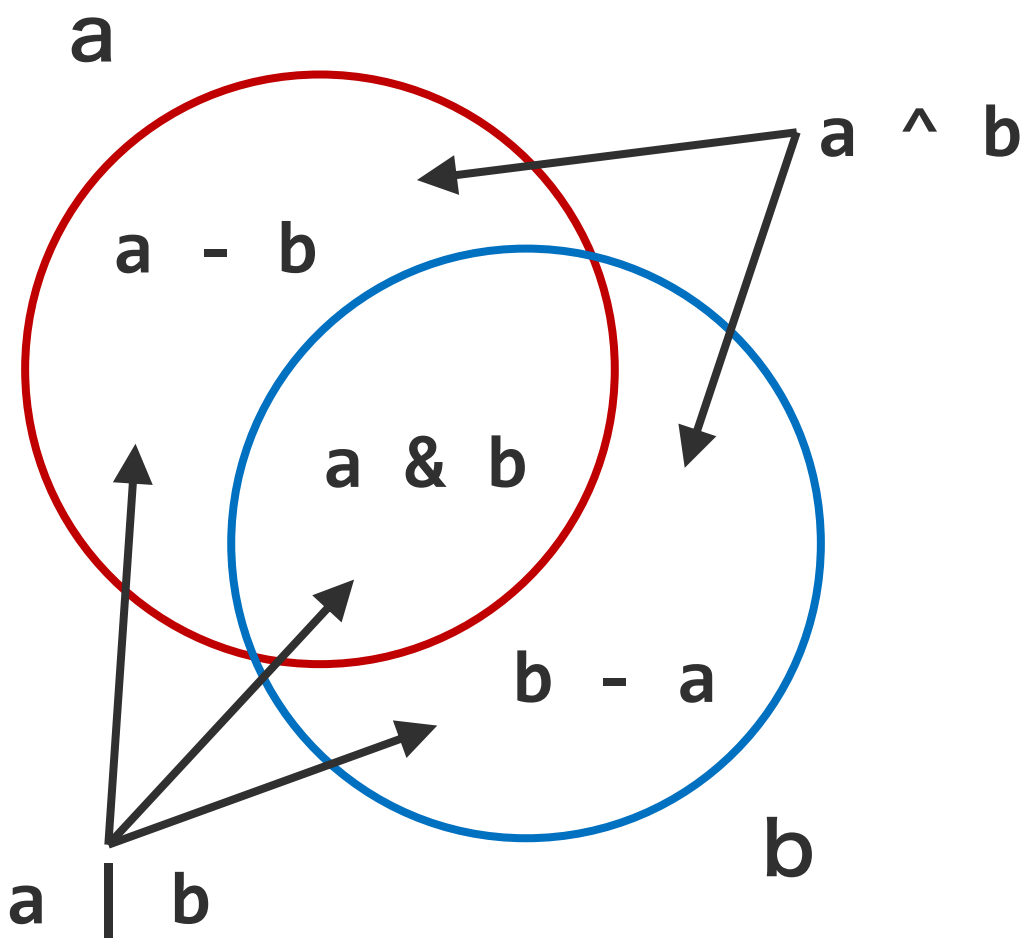
集合は、リストに含まれる重複要素を取り除いたり、２つのリストを比較して共通要素を取り出したりする際に使われる。ただし、リストを集合に変更した時に順位情報が失われるので、集合を再びリストに戻した時は元の順序が失われる。

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



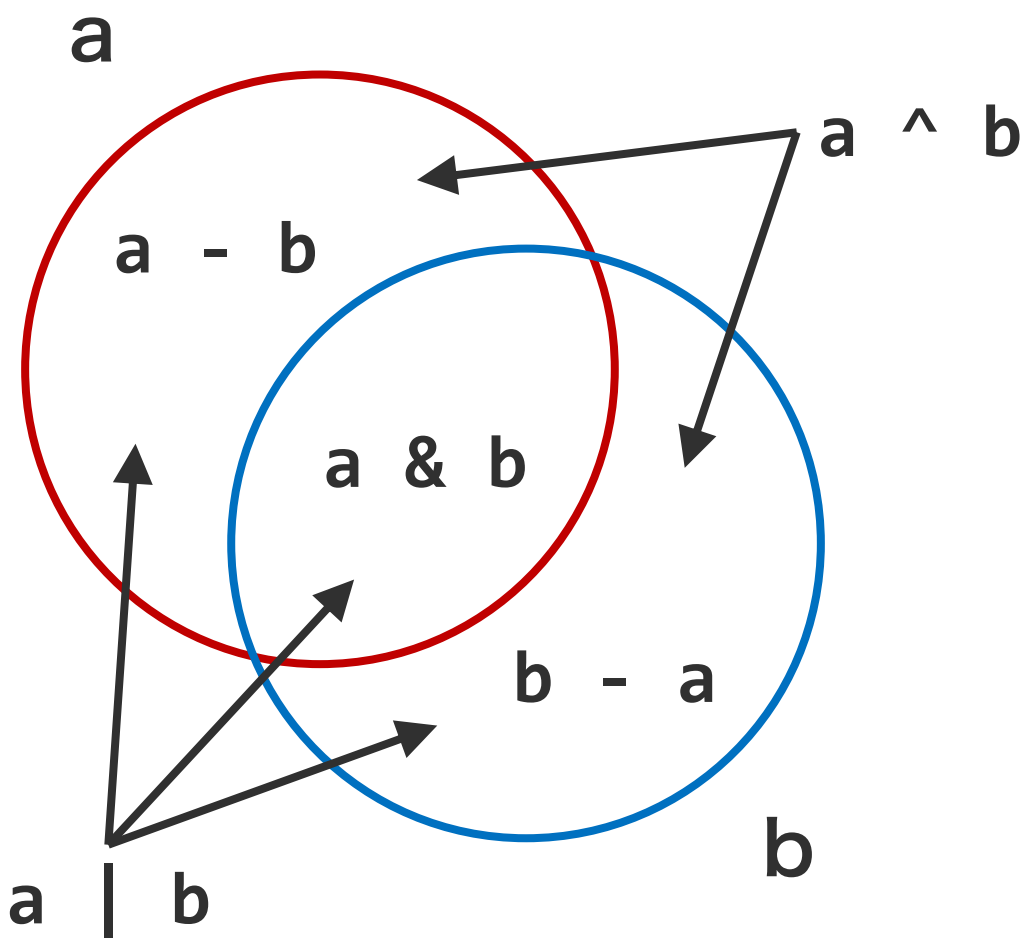
集合演算

```
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

順序なし

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。

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。

農学生命情報科学特論Ⅰ

1

- 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

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					

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

d = b + c
d
```

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

d = b + c
d
```

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

d = b + c
d
```

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。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。
- 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


if takeout:
    s = ???



if eatin:
    s = ???
```

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。

```
takeout = True
eatin = False
coupon = True

s = 1100



if (coupon)  and  (s > 1000) :

    s = (a + b) * 0.95

s
```

```
takeout = True
eatin = False
coupon = True

s = 1100



if (coupon)  or  (s > 1000) :

    s = (a + b) * 0.95

s
```

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。

1000 円以上の消費でかつクーポン使用で 5% OFF

```
takeout = True
eatin = False
coupon = True

s = 1100

if (coupon)  and  (s > 1000) :

    s = (a + b) * 0.95

s
```

論理演算子
判定条件1
判定条件2

1000 円以上の消費またはクーポン使用で 5% OFF

```
takeout = True
eatin = False
coupon = True

s = 1100

if (coupon)  or  (s > 1000) :

    s = (a + b) * 0.95

s
```

論理演算子
判定条件1
判定条件2

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

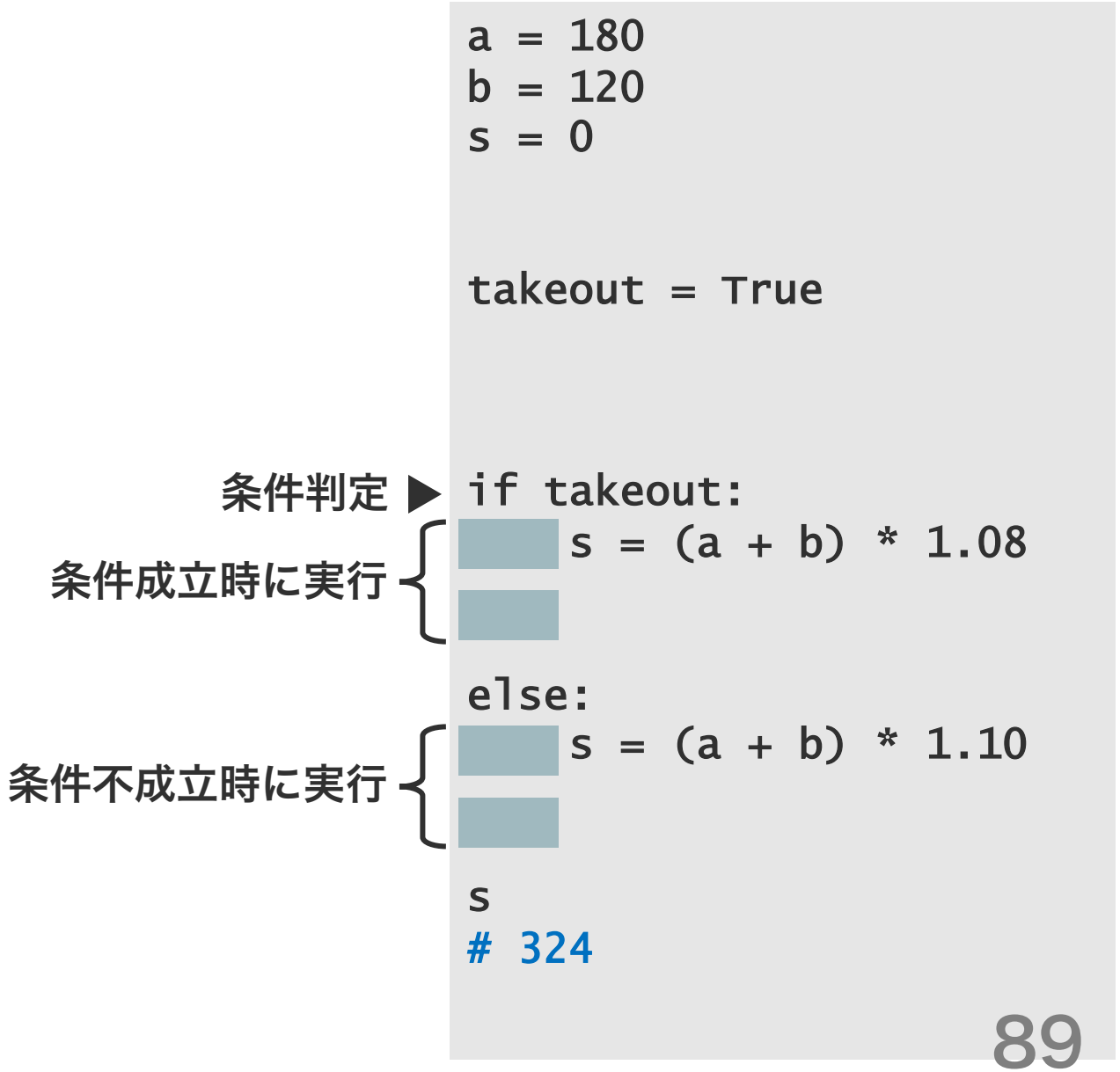
ある条件の真偽判定に基づいて処理を分けたい場合は、if 構文を 2 つ書くことで対応できる。しかし、if 構文を 2 つ続けて書くことで、同じ条件を 2 回判断していることになる。理論的にも、処理的にも煩雑である。このような場合は、if と else を用いて条件構文を作成すると、理論的にも処理的にもシンプルになる。

```
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。



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。なお、商品購入時に、可能であればクーポン券を使用する。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、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
```

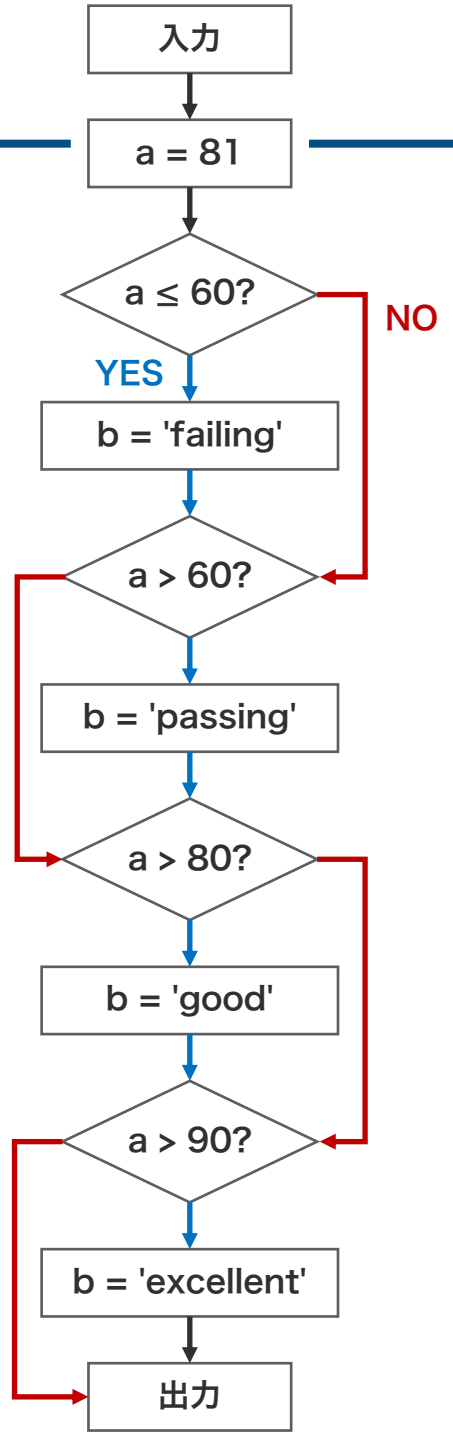
```
a = 81
b = ''

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

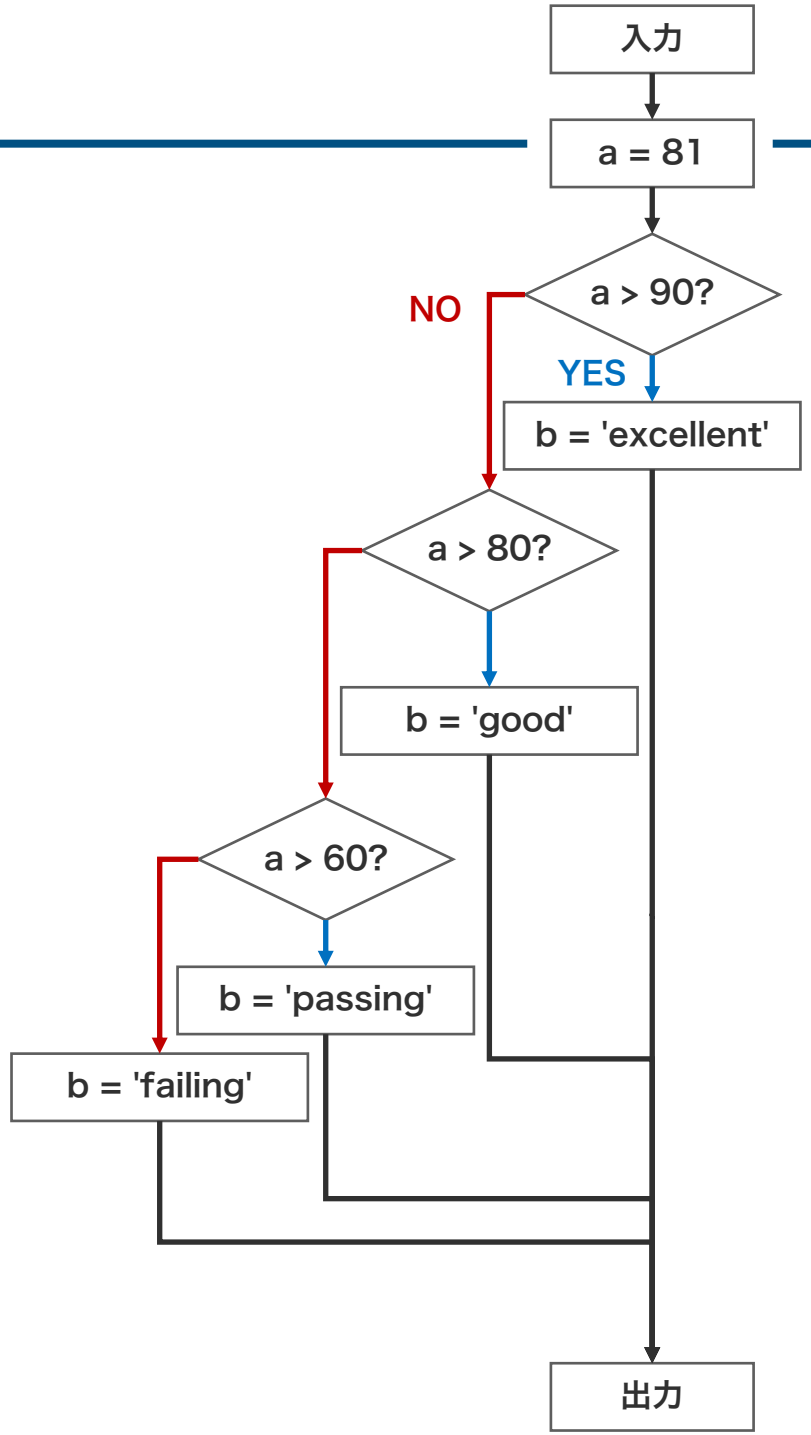
```
a = 81
b = ''

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

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

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 S1-5

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。

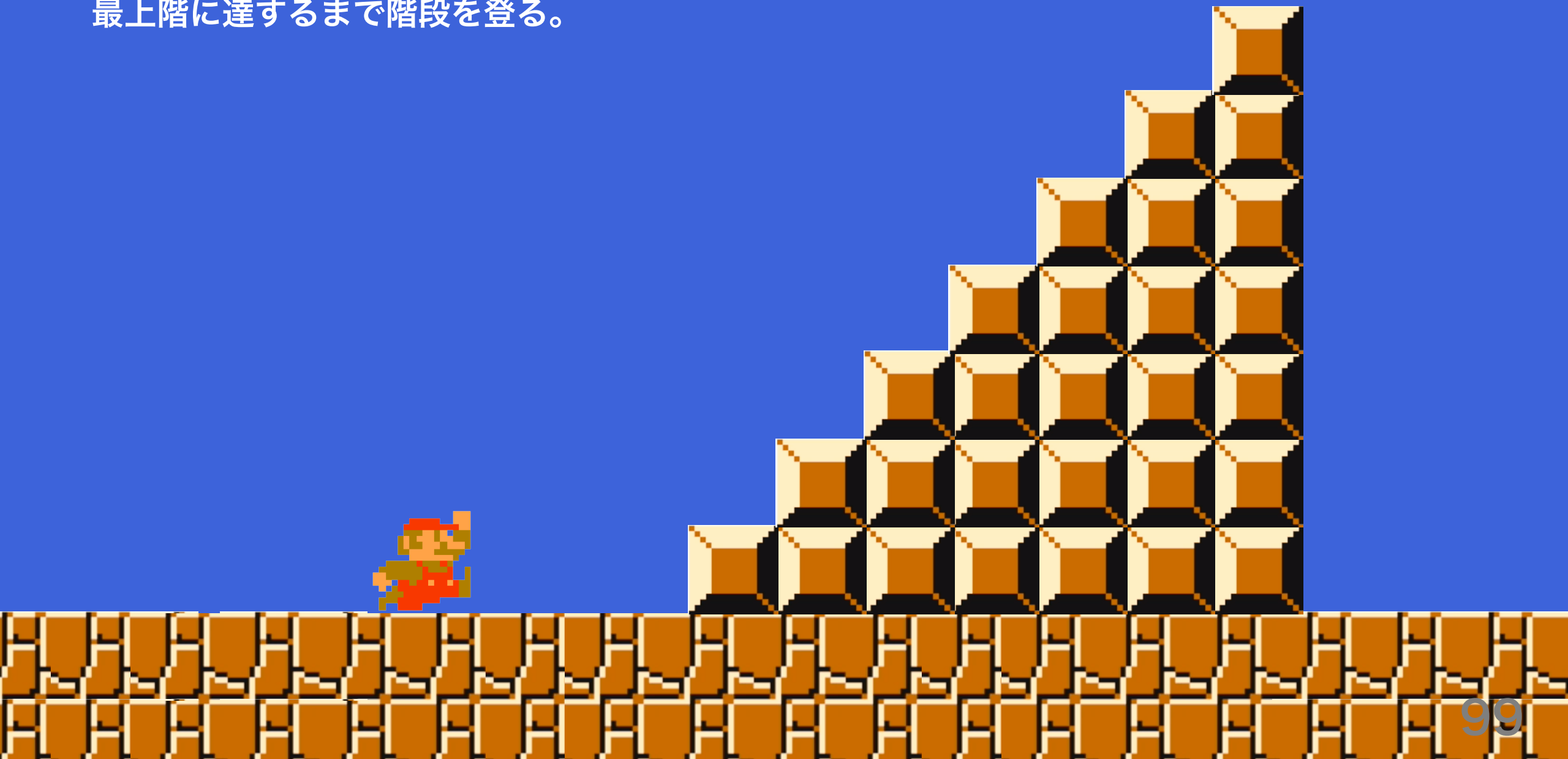
```
a = 11
s = ''
if (3 <= a) and (a <= 5):
    s = 'spring'

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

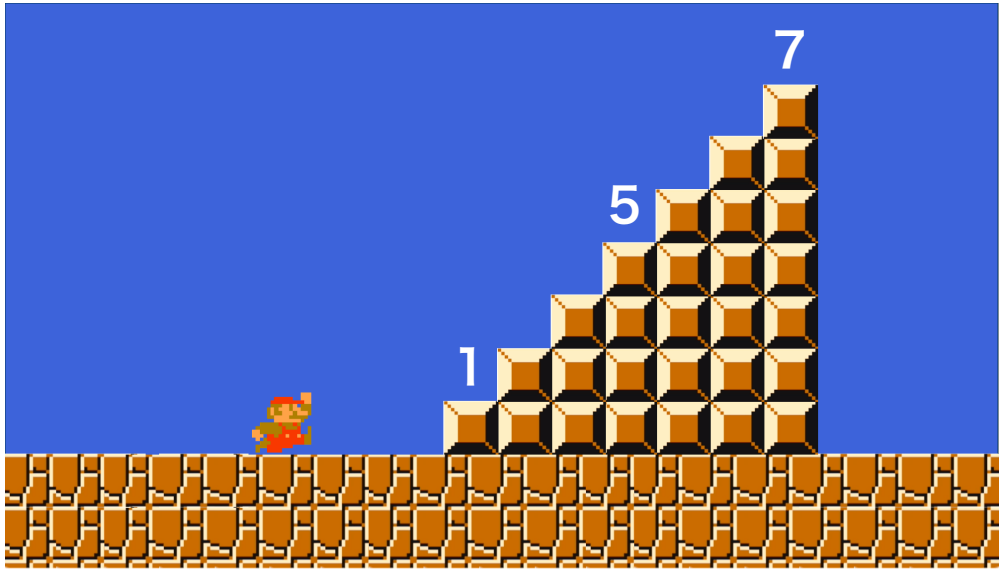
- ○ 予約語
- ○ 条件構文
- ○ 繰り返し構文
- ○ 関数
- ○ パッケージ

最上階に達するまで階段を登る。



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:

    current_stage = current_stage + 1


print(current_stage)
```

while 構文

```
current_stage = 0

while current_stage < 7 :
    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 文を使用する時に、繰り返す回数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

繰り返し回数　繰り返す範囲
      ↓           ↓
for i in range(len_b) :
    b[i] = b[i] * 1000


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

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
```

問題 S2-5

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
```

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
has_coupon = True

s = 0

if has_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
has_coupon = True

s = 0

if has_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
has_coupon = False

s = 0

if has_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
has_coupon = False

s = 0

if has_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
has_coupon = False

s = 0

if has_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
has_coupon = True

s = 0

if has_coupon:
    s = a * 0.95

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

s
```

関数

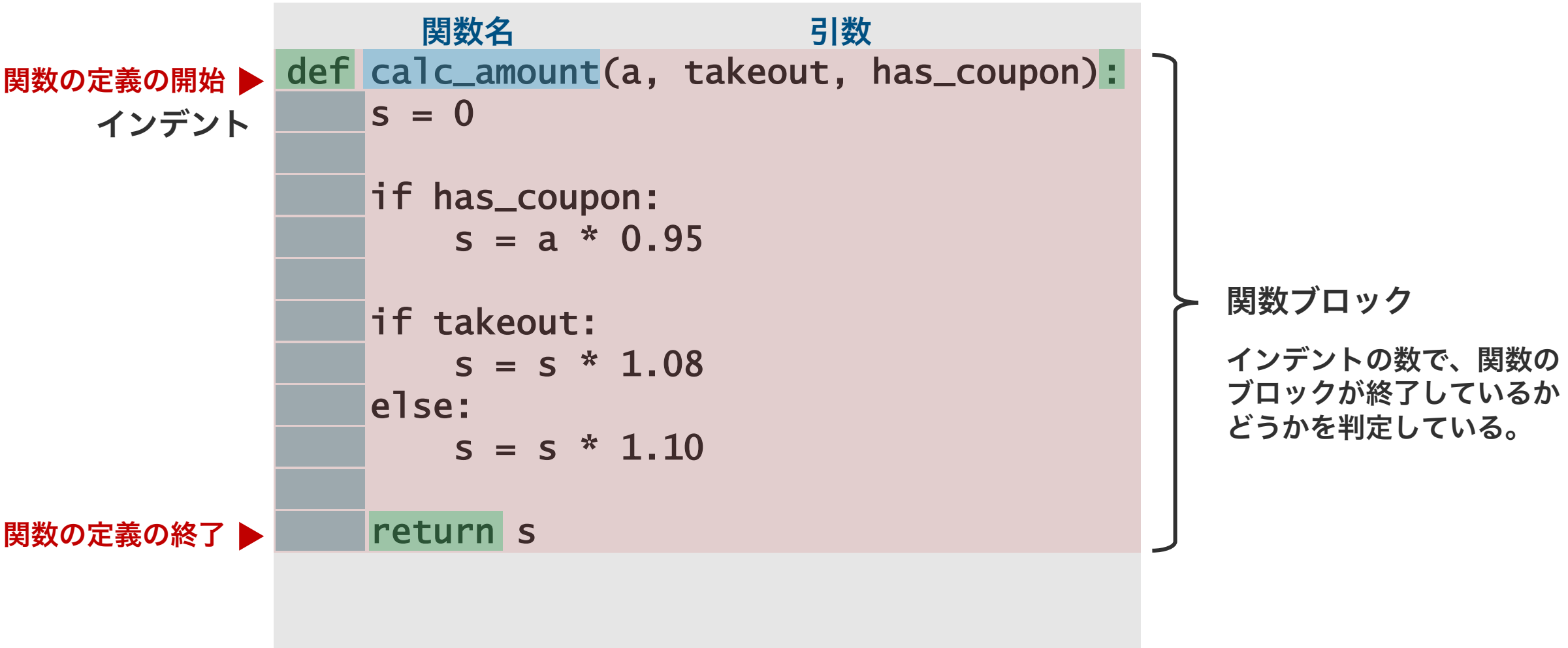
```
a = 120
takeout = True
has_coupon = True

s = 0

if has_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, has_coupon):
    s = 0

    if has_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, has_coupon):
s = 0
if has_coupon:
s = a * 0.95
if takeout:
s = s * 1.08
else:
s = s * 1.10
return s
関数の定義の終了
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, has_coupon):
    s = 0
    if has_coupon:
        s = a * 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。

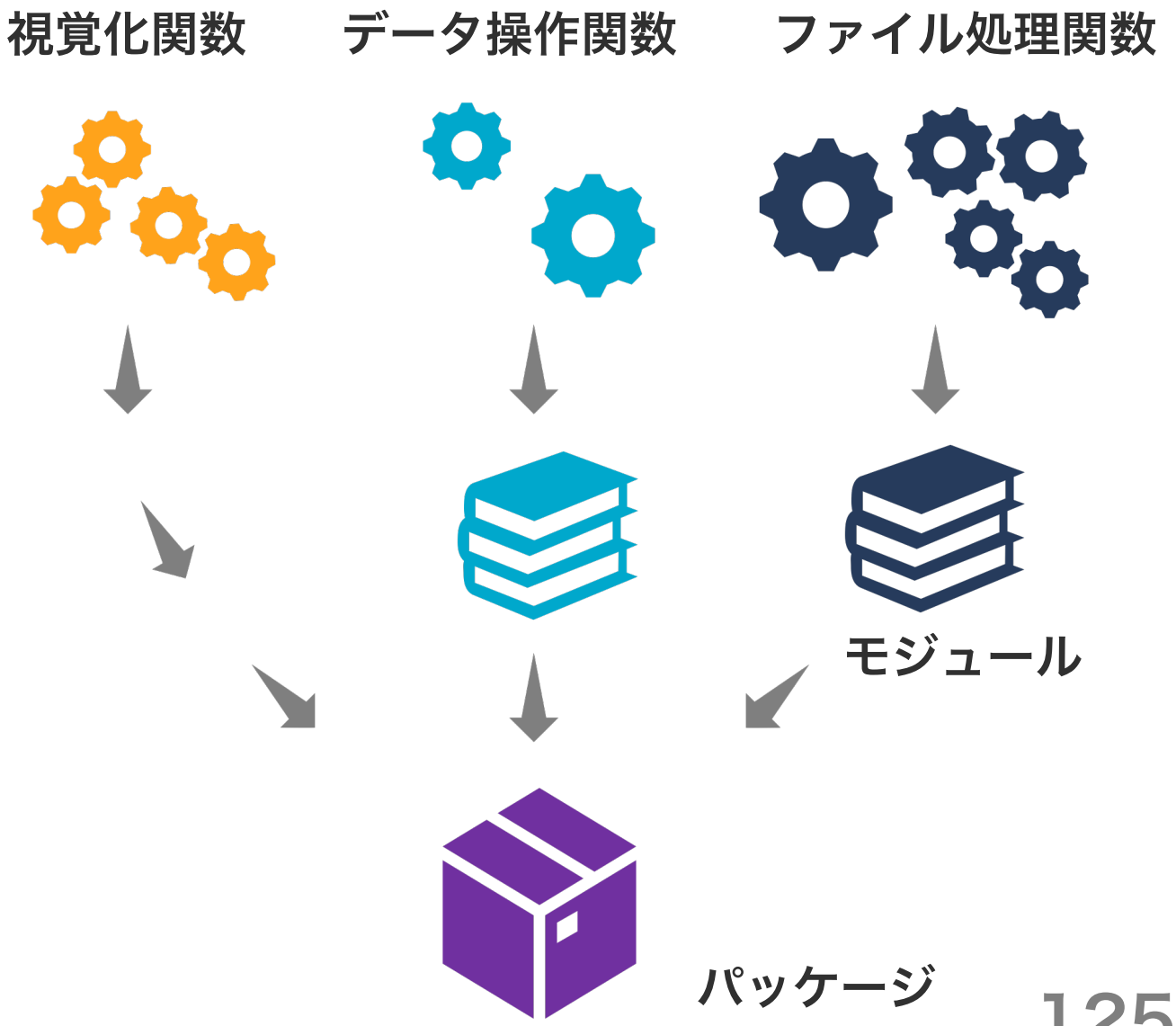
```
a = [1, 7, 4, 8, 3]

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

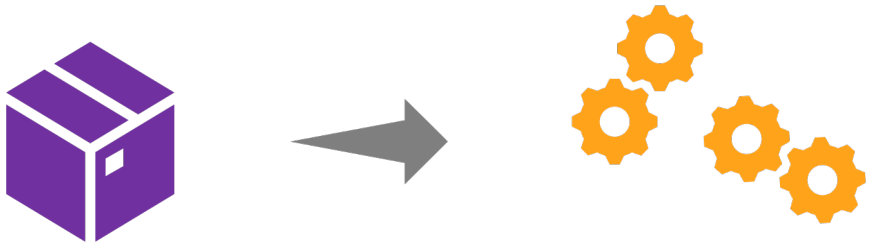
パッケージは、各開発者や開発団体の公式ウェブサイトあるいは GitHub や PyPI などのレポジトリで配布されている。例えば、PyPI には約 17 万のパッケージが登録されている。PyPI への登録は個人でも簡単に行えるので、研究目的の場合は、あまりマイナーなパッケージを使わないようにすること。

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（調達する）。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()
```