Python for bioinformatics

どんぐり研究所　孫 建強

コンテンツ

Introduction

1. プログラミング言語概要
2. Python 環境構築

Basic

3. データ型
4. 基本文法

Packages

5. テキスト処理
6. 数値計算 NumPy
7. データ処理 Pandas
8. データ可視化 matplotlib
9. バイオインフォマティクス

Advanced

10. オブジェクト指向

データ型

- 変数
- リスト
- ディクショナリ
- セット
- タプル

データ型

- 変数
- リスト
- ディクショナリ
- セット
- タプル

変数

変数は、値を保存するための箱のようなものである。変数に値を保持させるためには、その変数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。

変数

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 01-1

3 min

ペプチド PGWR の分子量を計算せよ。ただし、各アミノ酸の分子量は以下の表に示したものを使うこと。

アミノ酸	分子量
PRO / P	115.13
GLY / G	75.07
TRP / W	204.23
ARG / R	174.20

```
aa_p = 115.13

aa_g = 75.07

aa_w = 204.23

aa_r = 174.20


peptide = ? ? ?
```

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 01-2

5 min

3 つのペプチド PGWR、LGRQ、LGLP の分子量の平均を計算せよ。ただし、各アミノ酸の分子量は以下の表に示したものを使うこと。

アミノ酸	分子量
PRO / P	115.13
GLY / G	75.07
TRP / W	204.23
ARG / R	174.20
LEU / L	131.17
GLN / Q	146.15

```
p = 115.13
g = 75.07
w = 204.23
r = 174.20
l = 131.17
q = 146.15


pgwr = ? ? ?

lgrq = ? ? ?

lglp = ? ? ?


peptide_ave = ? ? ?
```

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)
}
```

変数

変数に数値のほか文字や文字列を代入することもできる。

文字を変数に代入するとき、その文字が、変数の名前ではなく、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必要がある。

```
s = 'a'


t = 'apple'


u = 't'
u
# 't'


v = t
v
# 'apple'
```

文字としてのｔ。

変数としてのｔ。

データ型

- ○ 変数
- ○ 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

5 min

ペプチド PGWR の分子量を計算せよ。ただし、各アミノ酸の分子量は以下の表に示したものを使うこと。

アミノ酸	分子量
PRO / P	115.13
GLY / G	75.07
TRP / W	204.23
ARG / R	174.20

```
pgwr = [115.13, 75.07, 204.23, 174.20]
```

問題 O2-2

5 min

3 つのペプチド PGWR、LGRQ、LGLP の分子量の平均を計算せよ。ただし、各アミノ酸の分子量は以下の表に示したものを使うこと。

アミノ酸	分子量
PRO / P	115.13
GLY / G	75.07
TRP / W	204.23
ARG / R	174.20
LEU / L	131.17
GLN / Q	146.15

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。

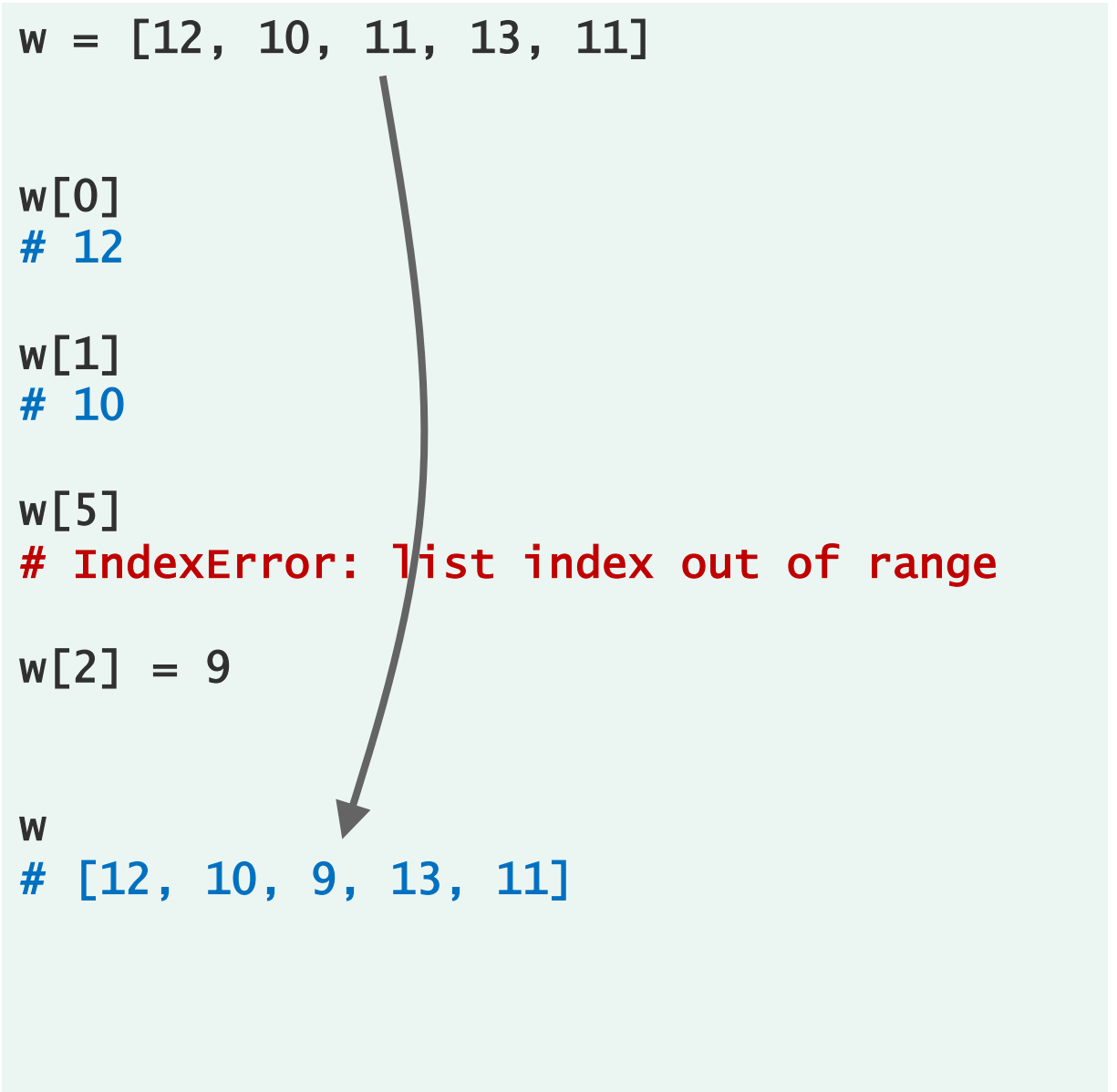
リスト

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。

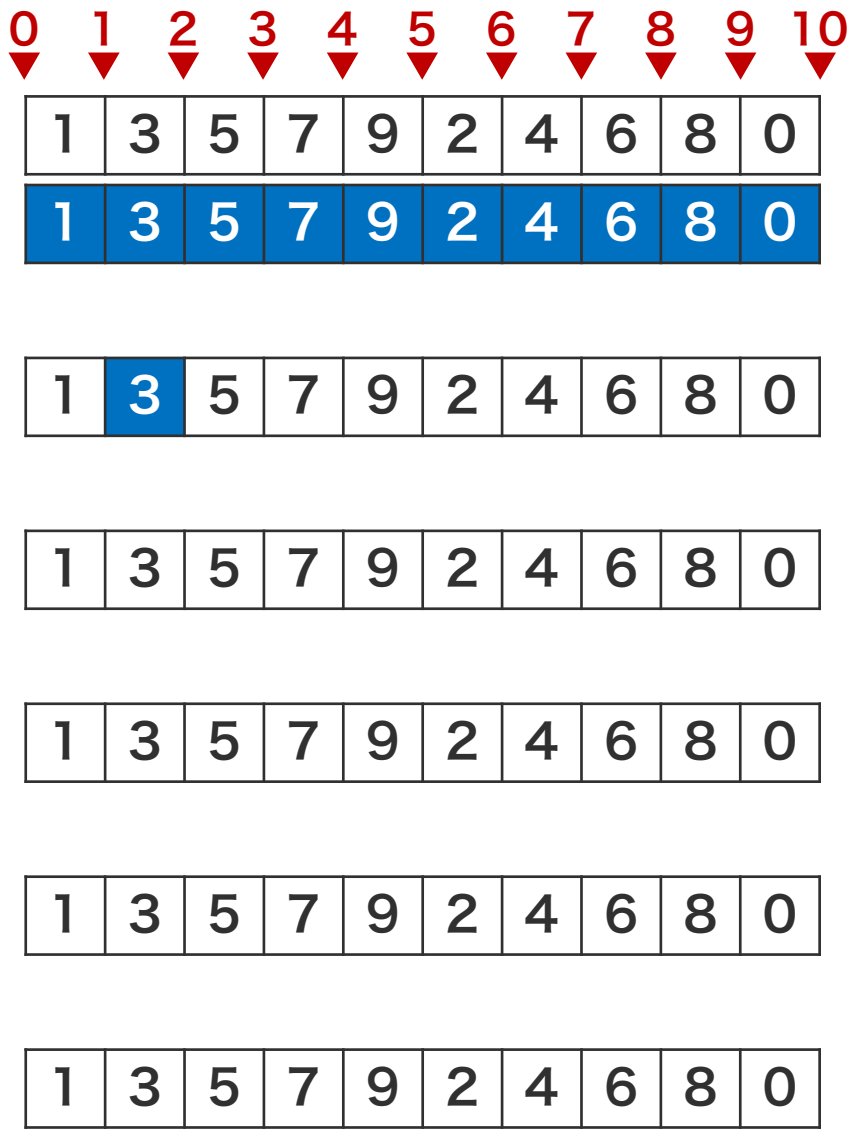


リスト

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。

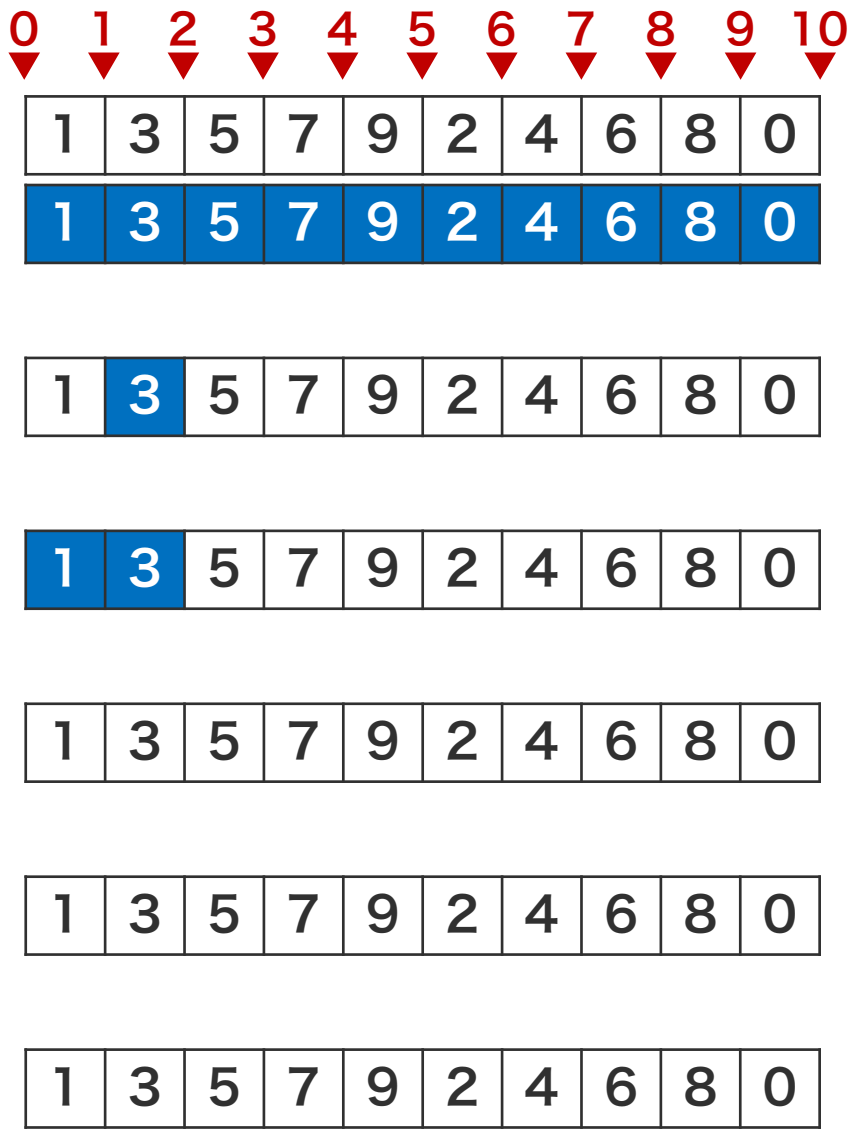
```
a = [1, 3, 5, 7, 9, 2, 4, 6, 8, 0]



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



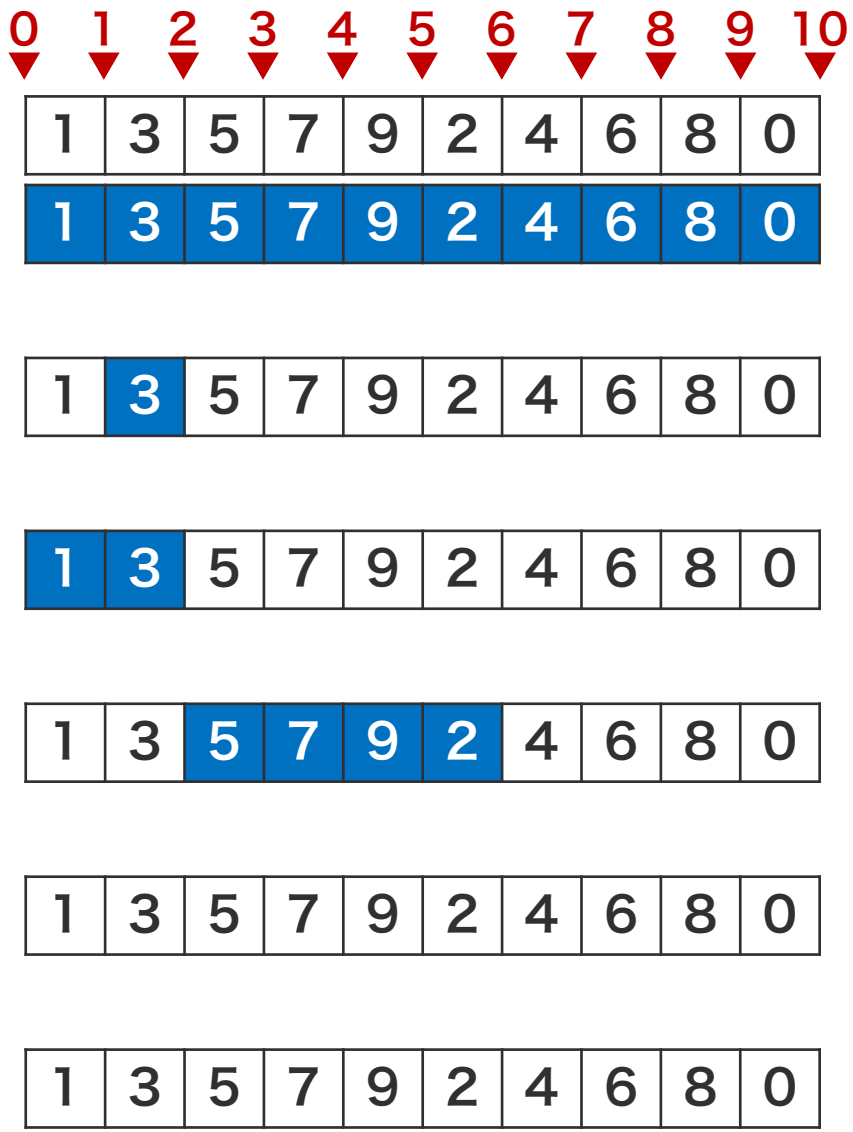

```
a = [1, 3, 5, 7, 9, 2, 4, 6, 8, 0]


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



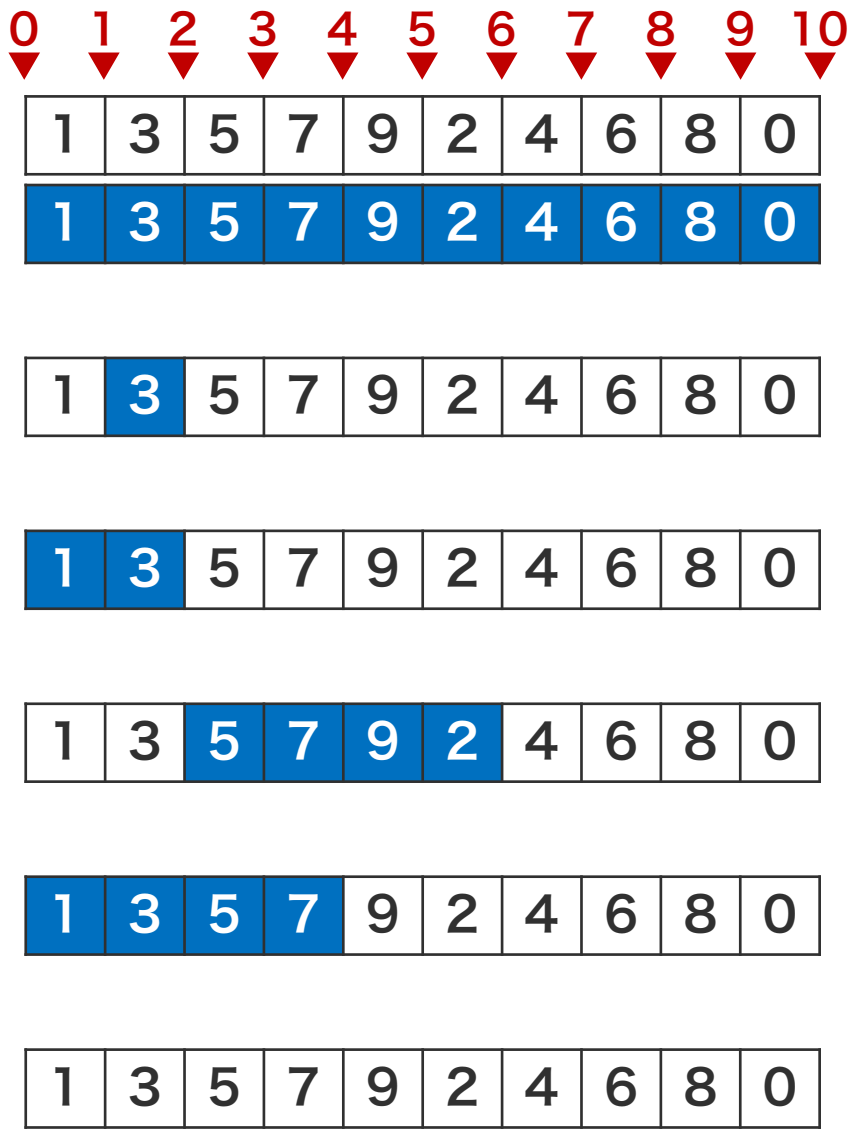

```
a = [1, 3, 5, 7, 9, 2, 4, 6, 8, 0]


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



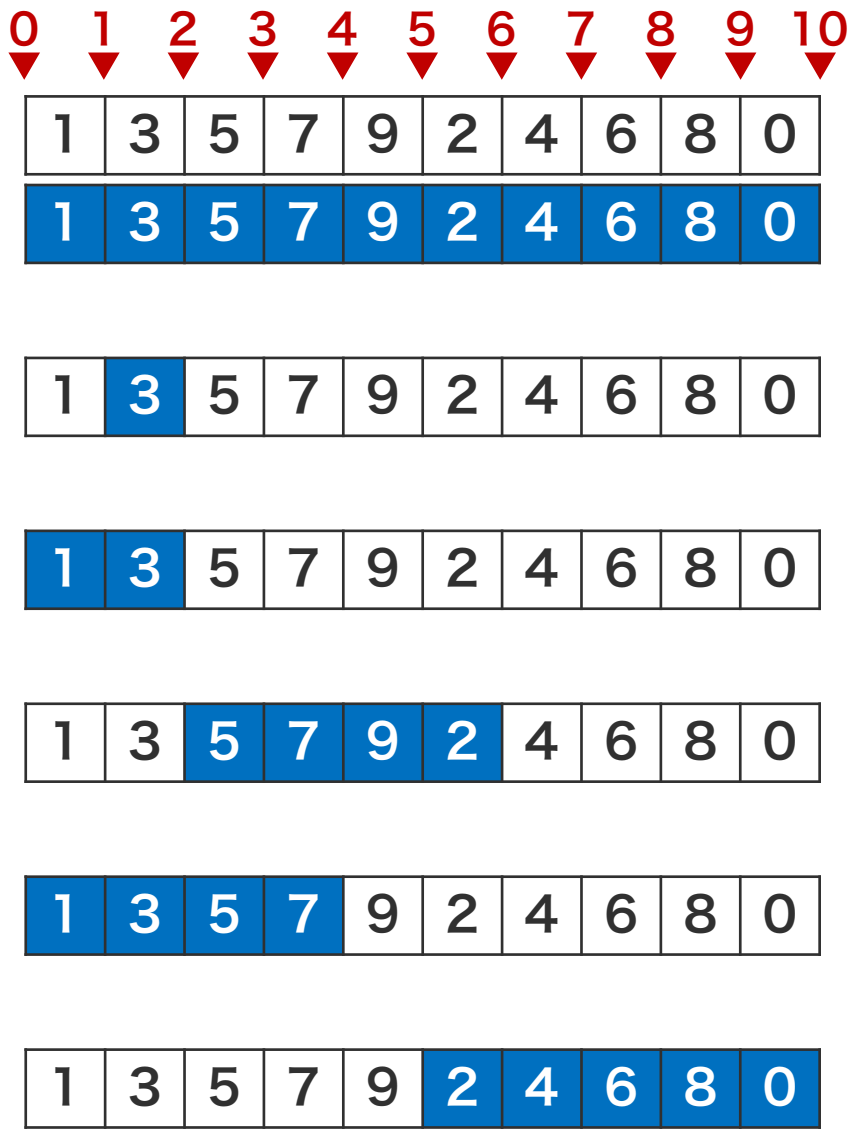

```
a = [1, 3, 5, 7, 9, 2, 4, 6, 8, 0]


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]
```

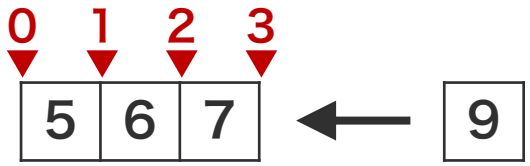
リスト操作

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。

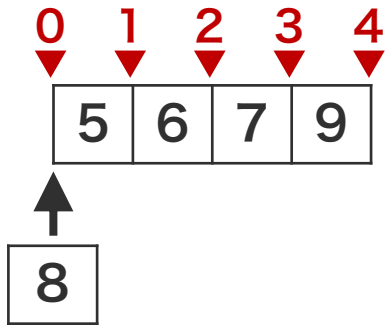
```
a = [5, 6, 7]


a.append(9)
a


a.insert(0, 8)
a


a.insert(2, 1)
a


a.append(4)
a
```

リスト操作



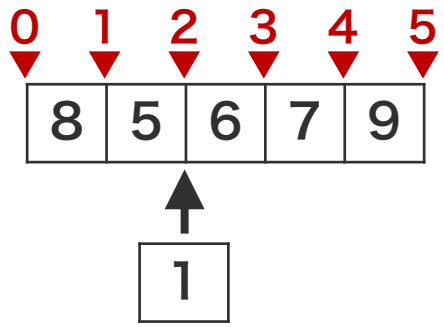

```
a = [5, 6, 7]

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



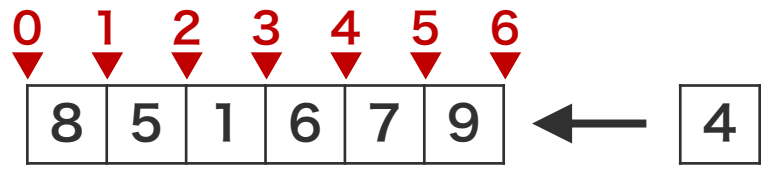

```
a = [5, 6, 7]

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
```

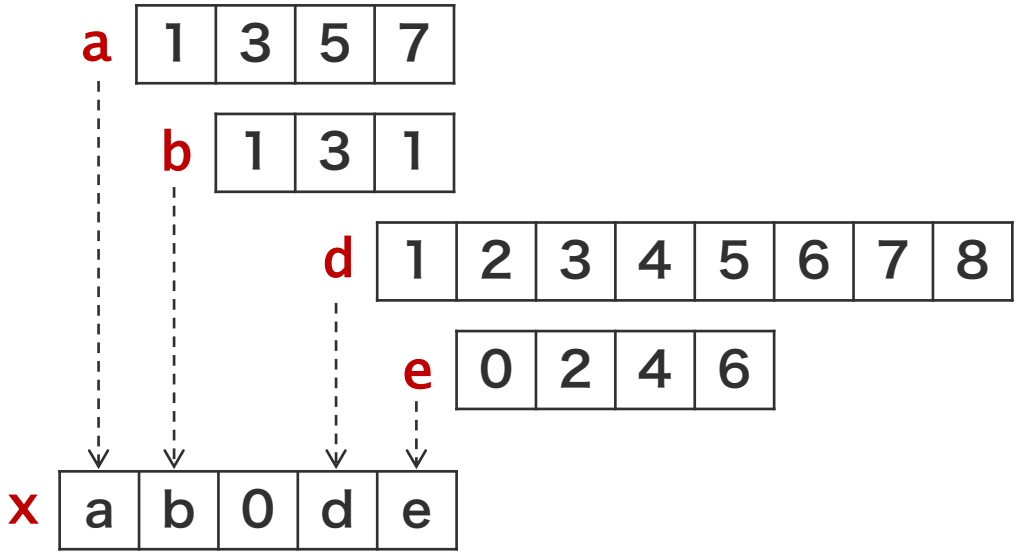
リスト操作




```
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。

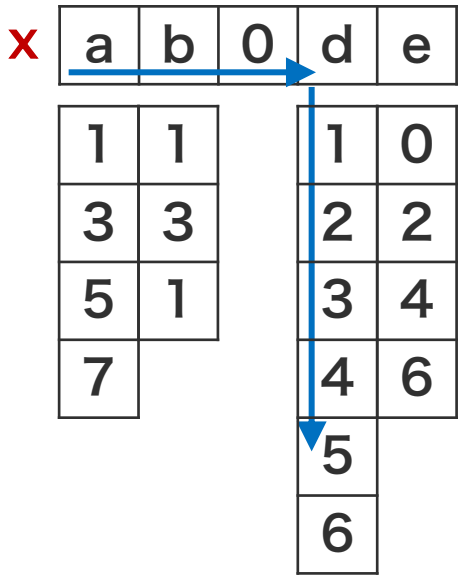



```
a = [1, 3, 5, 7]
b = [1, 3, 1]
d = [1, 2, 3, 4, 5, 6, 7, 8]
e = [0, 2, 4, 6]

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][4]
# 5
```



データ型

- ○ 変数
- ○ 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。

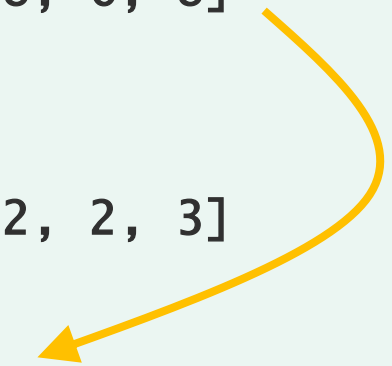
```
weights = {
    'buna': 2.0,
    'kashi': 1.3,
    'nara': 2.8
}


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

5 min

ペプチド PGWR の分子量を計算せよ。ただし、各アミノ酸の分子量は以下の表に示したものを使うこと。

アミノ酸	分子量
PRO / P	115.13
GLY / G	75.07
TRP / W	204.23
ARG / R	174.20

```
aa2mw = {
  'P': 115.13,
  'G':  75.07,
  'W': 204.23,
  'R': 174.20
}

pgwr = ? ? ?
```

問題 O3-2

5 min

3 つのペプチド PGWR、LGRQ、LGLP の分子量の平均を計算せよ。ただし、各アミノ酸の分子量は以下の表に示したものを使うこと。

アミノ酸	分子量
PRO / P	115.13
GLY / G	75.07
TRP / W	204.23
ARG / R	174.20
LEU / L	131.17
GLN / Q	146.15

問題 O3-3

5 min

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
```

データ型

- ◯ 変数
- ◯ 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。

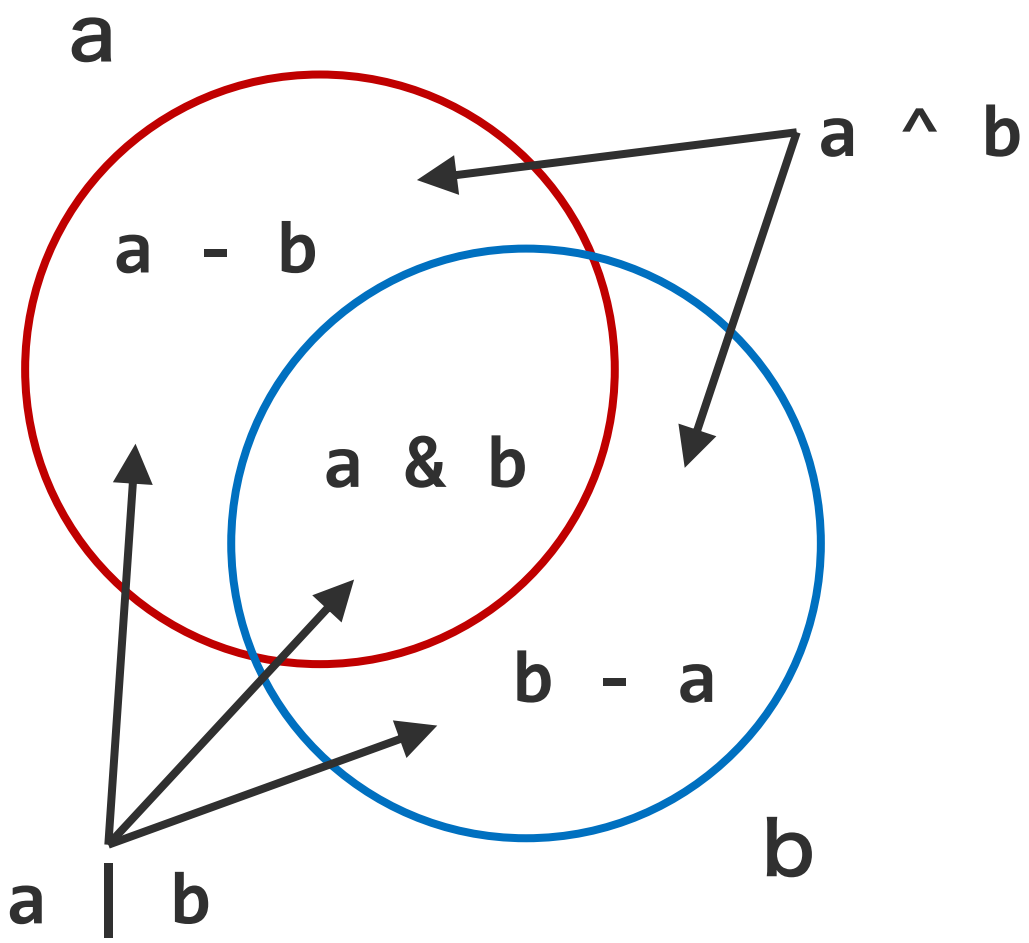
順序集合を取り扱う場合は、collections パッケージ中の OrderedDict 型のオブジェクトを使用する。この資料では順序集合を取り上げない。

```
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

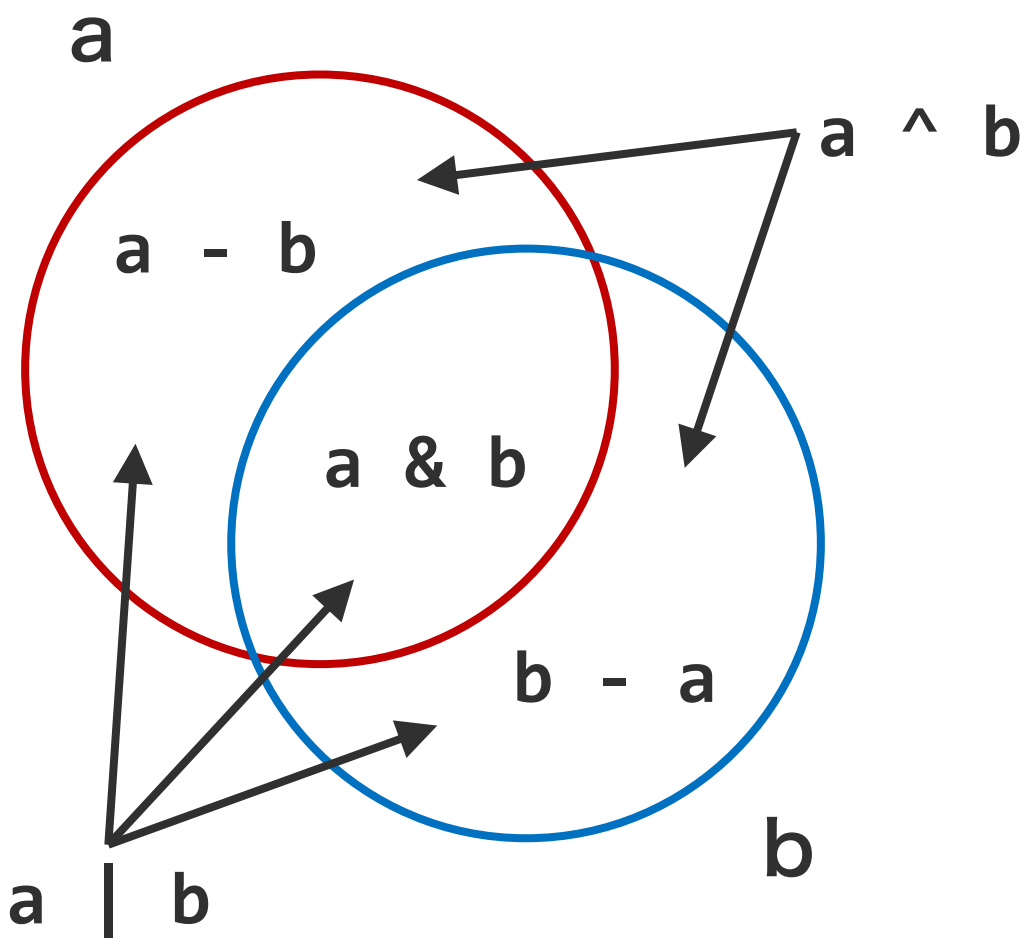
集合は、リストに含まれる重複要素を取り除いたり、２つのリストを比較して共通要素を取り出したりする際に使われる。ただし、リストを集合に変更した時に順位情報が失われるので、集合を再びリストに戻した時は元の順序が失われる。

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



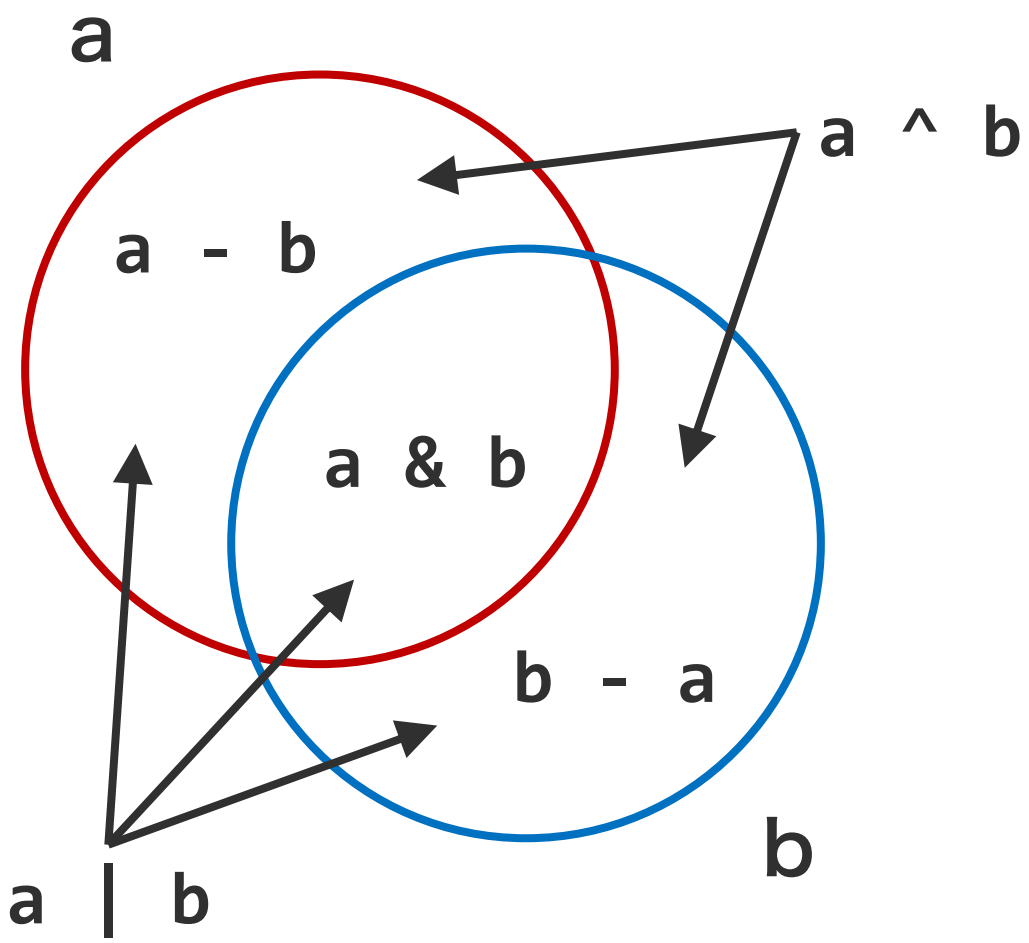
集合演算

```
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}

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


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


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


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


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



データ型

- ◯ 変数
- ◯ 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
# (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。