だれでもPython チャレンジ

Python Challenge for Everyone

おとなも子どももパイソンで遊ぼう!

あらためてPython: まとめと復習

子どもPythonチャレンジシリーズでの対応回

A)	変数と計算	5.1回
B)	条件分岐(ぶんき)	5.2回
C)	繰り返しループ	5.3回
D)	関数	5.4回
E)	文字列、リストなど (Part 1)	25.1回
F)	文字列、リストなど (Part 2)	25.2回

まとめと復習 A

1. 変数と代入と計算
2. 割り算の商と余り など
3. 演算と 短縮(たんしゅく)表現
4. フォーマット文字列 (f-文字列)
5. ...

変数

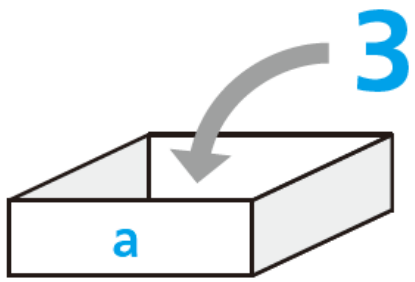
「**変数**」とは、値を入れておく入れ物

```
a = 3
```

←「変数aに3を**代入**する」

変数名

「代入」の一般的なイメージ



a という名前のついた箱に 3 を入れる

問題 1a

対話モードで、次の計算を実行して結果を確認しましょう。

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

問題 1a（解答）

対話モードで、次の計算を実行して結果を確認しましょう。

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

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

print文

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

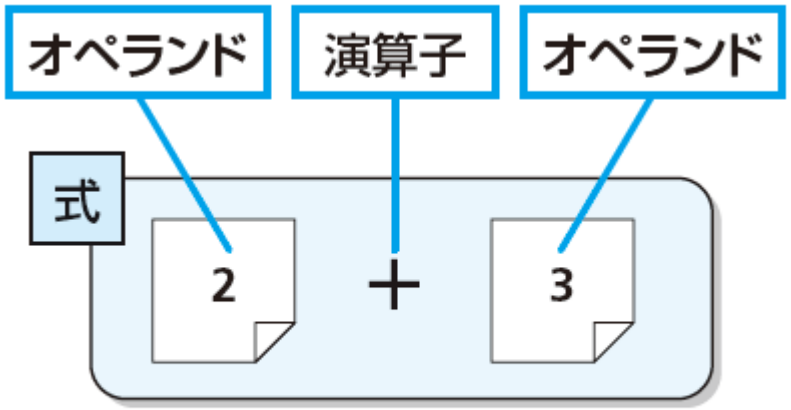
print('出力する内容')
として、文字列を出力できる。

文字列の前後をシングルクォート（’）、
または、ダブルクォート（"）で囲む

数などを出力することもできる
print(65)

算術演算

- 覚えるべき用語



式の値は5

文（代入文）

```
a = 2 + 3
```

リテラル
（プログラムコード中に記述される数値）

演算子の種類

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

変数を含む算術演算

式に変数名が含まれる場合は、
変数に代入されている値が使用される

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

変数aに10を代入します

式a + 3の値を出力します

```
b = a + 3
```

(変数aの値)+3 が変数bに代入される

```
a = a + 3
```

(変数aの値)+3 が変数aに代入される
つまりaの値が3増える

問題 1b

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

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

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

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

問題 1b（解答）

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

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

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

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

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

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

割り算での商と余り など

◆ 7を3で割る　→　「商(しょう) が 2 で 余りが1」
「7 ÷ 3 = 2 あまり 1」

- 商を求めるとき: 7 // 3　→ 2
- 余りを求めるとき: 7 % 3　→ 1

- xが偶数なら →2で割って余りが0 (割り切れる)
  つまり x % 2 は0
- xが奇数なら →2で割って余りが1 → つまり x % 2 は1

◆ 2を3回かけた数 (2*2*2)　→ 2 ** 3 と表す

問題 2

次の値を求める式を書いて、値を求めてください。

（1） 100を9で割った商 と 余り

（2） 1000を7で割った商 と 余り

（3） 3の5乗

（4） 1から10について、3で割った商と余りはどうなる?

算術演算の短縮表現

```
a = a + 3
```

短縮

```
a += 3
```

加算代入

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

問題 3

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

（1） a = a + 5

（2） b = b – 6

（3） c = c * a

（4） d = d / 3

（5） e = e % 2

問題 3 （解答）

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

（1） a = a + 5

a += 5

（2） b = b – 6

b -= 6

（3） c = c * a

c *= a

（4） d = d / 3

d /= 3

（5） e = e % 2

e %= 2

問題 4

次のプログラムコードを実行した後の変数aの値を答えてください。

（1）
```
a = 3
a *= 3
```

（2）
```
a, b = 3, 2
a *= b
```

（3）
```
a = 7
a  //=  3
```

（4）
```
a = 7
a  %=  4
```

変数の値の埋め込み

数値を文字列に変換してから連結

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

フォーマット文字列の使用して簡潔に記述できる

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

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

フォーマット文字列 (f-文字列)

フォーマット文字列 (f-文字列)

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

{変数名} 部分に**式**を入れることもできる

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

問題 5

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

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

問題 5 （解答）

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

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

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

※ フォーマット文字列を使う場合

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

まとめと復習 B

1. インデントとブロック
2. if文による条件分岐/ else / elif など
3. 条件式と関係演算子
4. 論理演算子による組み合わせ
5. …

input関数を使ったキーボードからの入力の受け取り

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

キーボードからの入力を変数nameで受け取ります（注❸-10）

受け取った文字列を使ってメッセージを出力します

実行結果

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

プログラムからの出力です

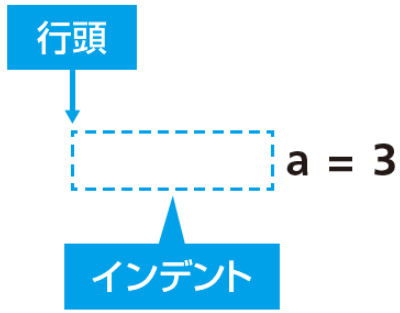
キーボードで入力した文字列です

入力された文字列を使ったメッセージが出力されました

インデントとブロック

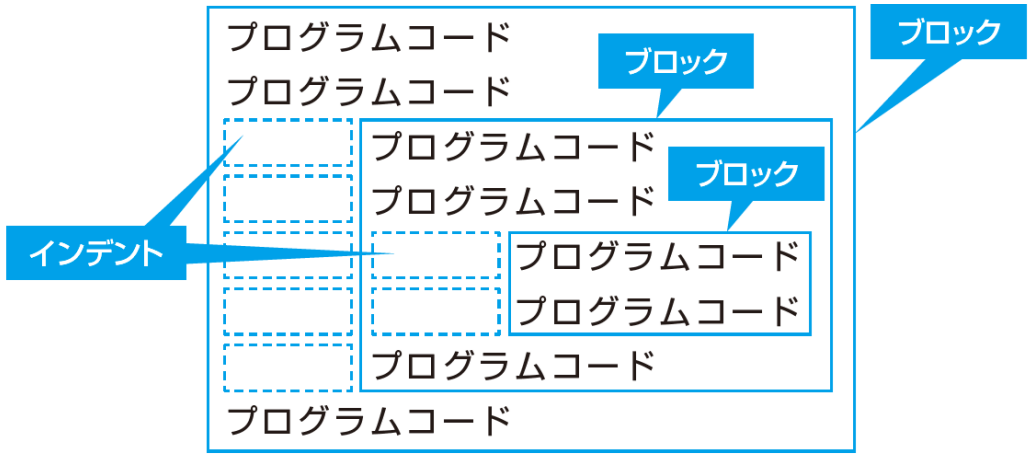
・インデント

行頭から最初の文字までの空白
空白文字4つ分を1つの単位とする



・ブロック

インデントによってプログラムコードを1つのまとまりにしたもの



if文による条件分岐

「もしも○○ならば××を実行する」

構文

コロンを付ける

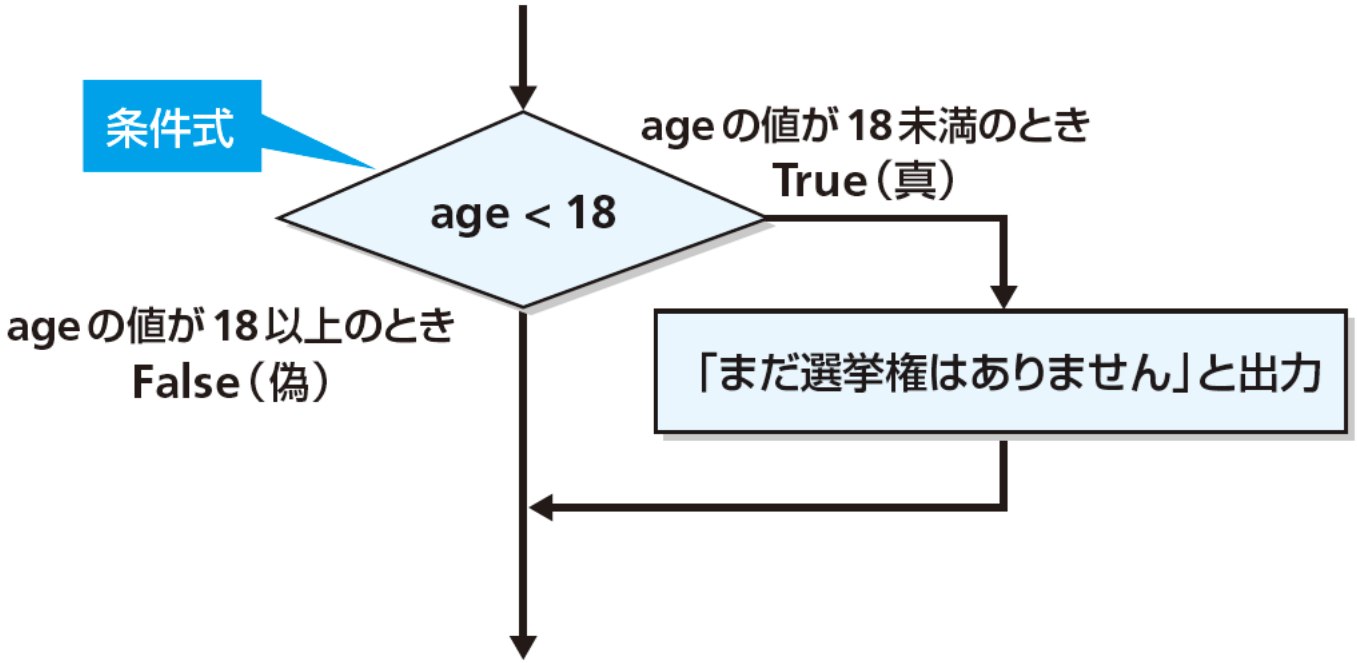
```
if 条件式:
    処理内容
```

インデント

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

if文による条件分岐

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



if文による条件分岐

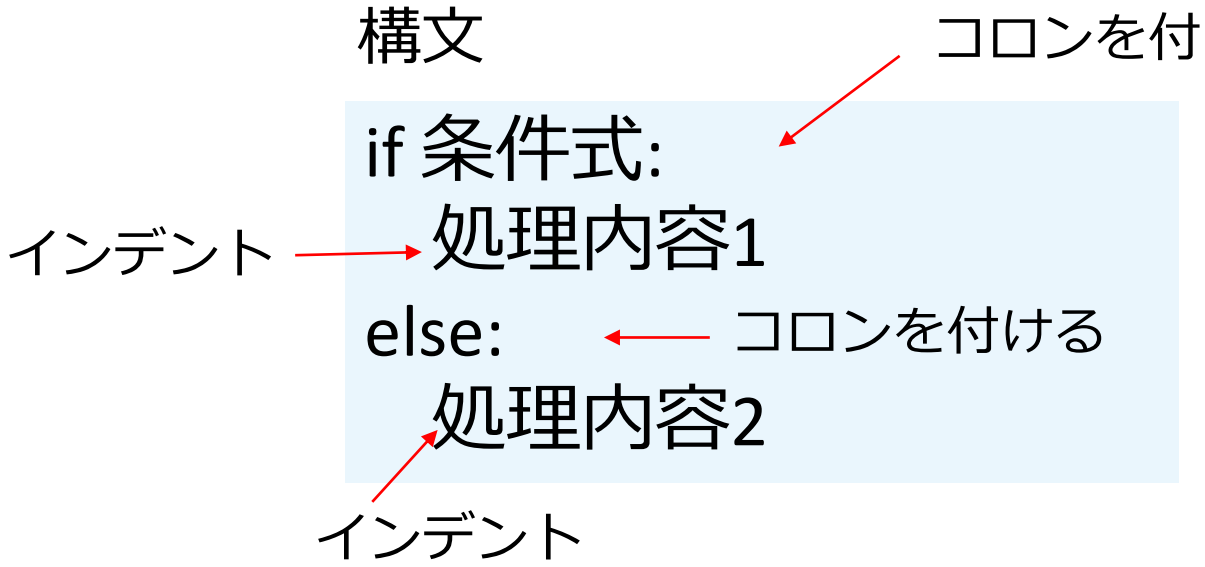
```
age = int(input('年齢を教えてください: '))
if age < 18:
    print('まだ選挙権はありません')
    print('18歳になったら投票に行きましょう')

print('処理を終わります')
```

キーボードから入力された数字を
int型にして、変数ageに代入します

if～else文による条件分岐

「もしも○○ならば××を実行し、そうでなければ△△を実行する」

構文



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

if～else文による条件分岐

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

age < 18がTrueのときに実行されます

age < 18がFalseのときに実行されます

if～elif～else文による条件分岐

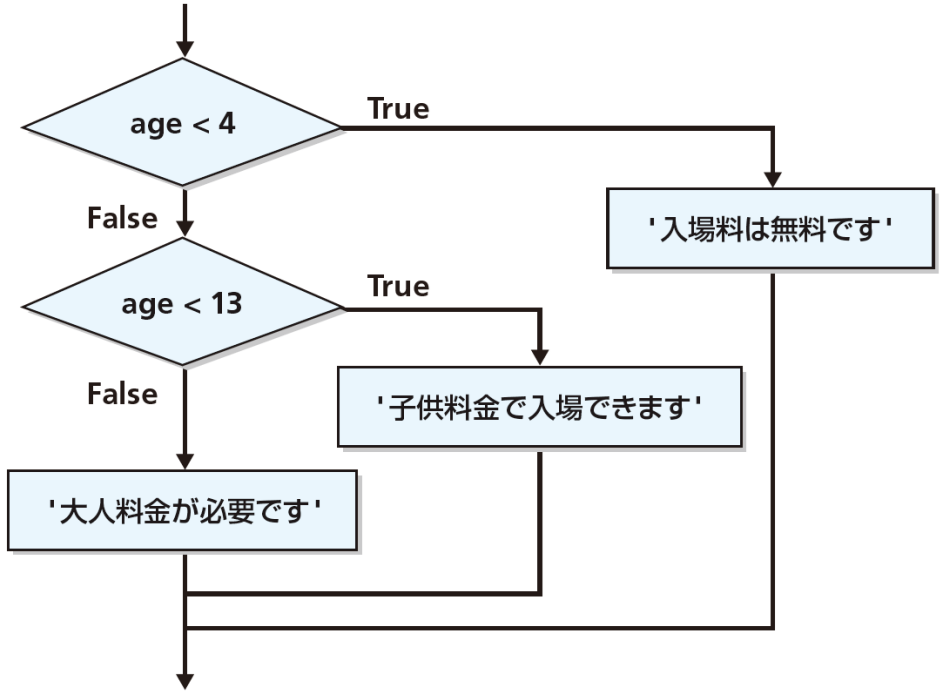
構文

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

条件式1 が **True** のとき **処理内容1** が実行される。
そうでないとき、**条件式2** が **True** のとき **処理内容2** が実行される。
それ以外の時には、**処理内容3**が実行される。

if～elif～else文による条件分岐

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



条件式と関係演算子

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

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

論理演算子による条件の組み合わせ

「変数aが10で、かつ変数bが5である」

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

「変数aが10である、または変数bが5である」

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

論理演算子

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

論理演算子による条件の組み合わせ

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

演算子の優先度とカッコ

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

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

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

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

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

if文と真偽値

```
if a == True:
    処理内容
```

aの値をTrueと比較しています

※ a が True のときだけ処理内容が実行される

```
if a:
    処理内容
```

条件式の代わりに変数aの値を用います

```
if not a:
    処理内容
```

※ a が False のときだけ処理内容が実行される

コメント文

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

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

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

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

※ プログラムコードの一部を一時的に無効にする用途でも使用できる

問題 1

次の条件を、関係演算子を使って記述してください。

問題例　aはbより大きい

解答例　a > b

（1） aはbと等しい
（2） aはbと等しくない
（3） bはcより小さい
（4） aはb以下である
（5） cはb以上である

問題 1（解答）

次の条件を、関係演算子を使って記述してください。

問題例　aはbより大きい

解答例　a > b

（1）　aはbと等しい　　a == b

（2）　aはbと等しくない　　a != b

（3）　bはcより小さい　　b < c

（4）　aはb以下である　　a <= b

（5）　cはb以上である　　c >= b

問題 2

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

```
a = 2021
```

空欄

問題 2（解答）

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

```
a = 2021
```

空欄

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

問題 3

次の条件を、論理演算子と関係演算子を使って記述してください。

（1） aは5または8と等しい

（2） aとcは両方ともb以下

（3） aは1より大きくて10より小さいが、5ではない

（4） aはbまたはcと等しいが、aとdは等しくない

条件式と関係演算子

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

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

論理演算子

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

問題 3（解答）

次の条件を、論理演算子と関係演算子を使って記述してください。

（1） aは5または8と等しい

a == 5 or a == 8

（2） aとcは両方ともb以下

a <= b and c <= b

（3） aは1より大きくて10より小さいが、5ではない

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

（4） aはbまたはcと等しいが、aとdは等しくない

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

まとめと復習 C

1. forループ と rangeオブジェクト
2. whileループ
3. 流れの変更
4. ループのネスト
5. ...

for文による処理の繰り返し

構文

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

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

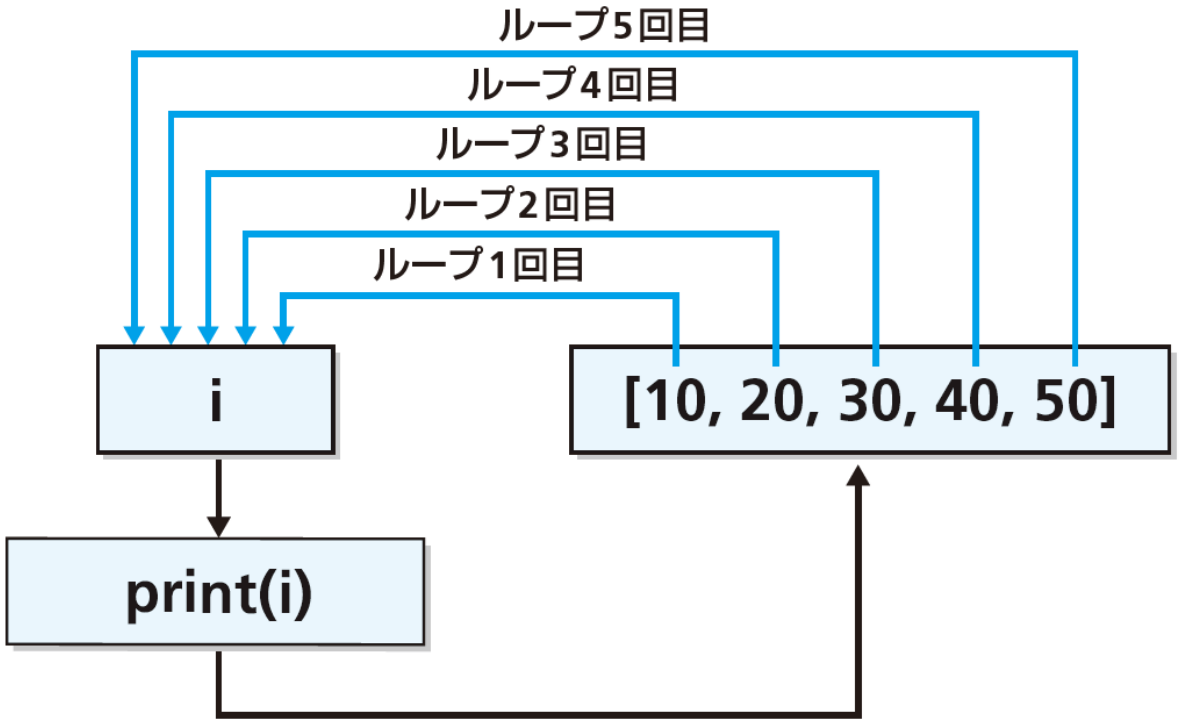
```
for i in [10, 20, 30, 40, 50]:
    print(i)
```

実行結果

```
10
20
30
40
50
```

for 文の処理の流れ

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



rangeオブジェクト

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

変数iに0から9の値が順番に代入されます

実行結果

```
0
1
2
(略)
9
```

0から9の値が1ずつ順番に出力されます

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

range オブジェクト

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

0から9までの数字が並びます

3から9までの数字が並びます

29を超えない範囲で1から10ずつ値が増えます

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

100から始まり5ずつ増える値が、200に達するまで順番に代入されます

実行結果

```
100
105
110
(略)
200
```

while文による処理の繰り返し

構文

```
while 条件式:
    処理内容
```

※ 条件式の値が True の間、処理内容を繰り返す

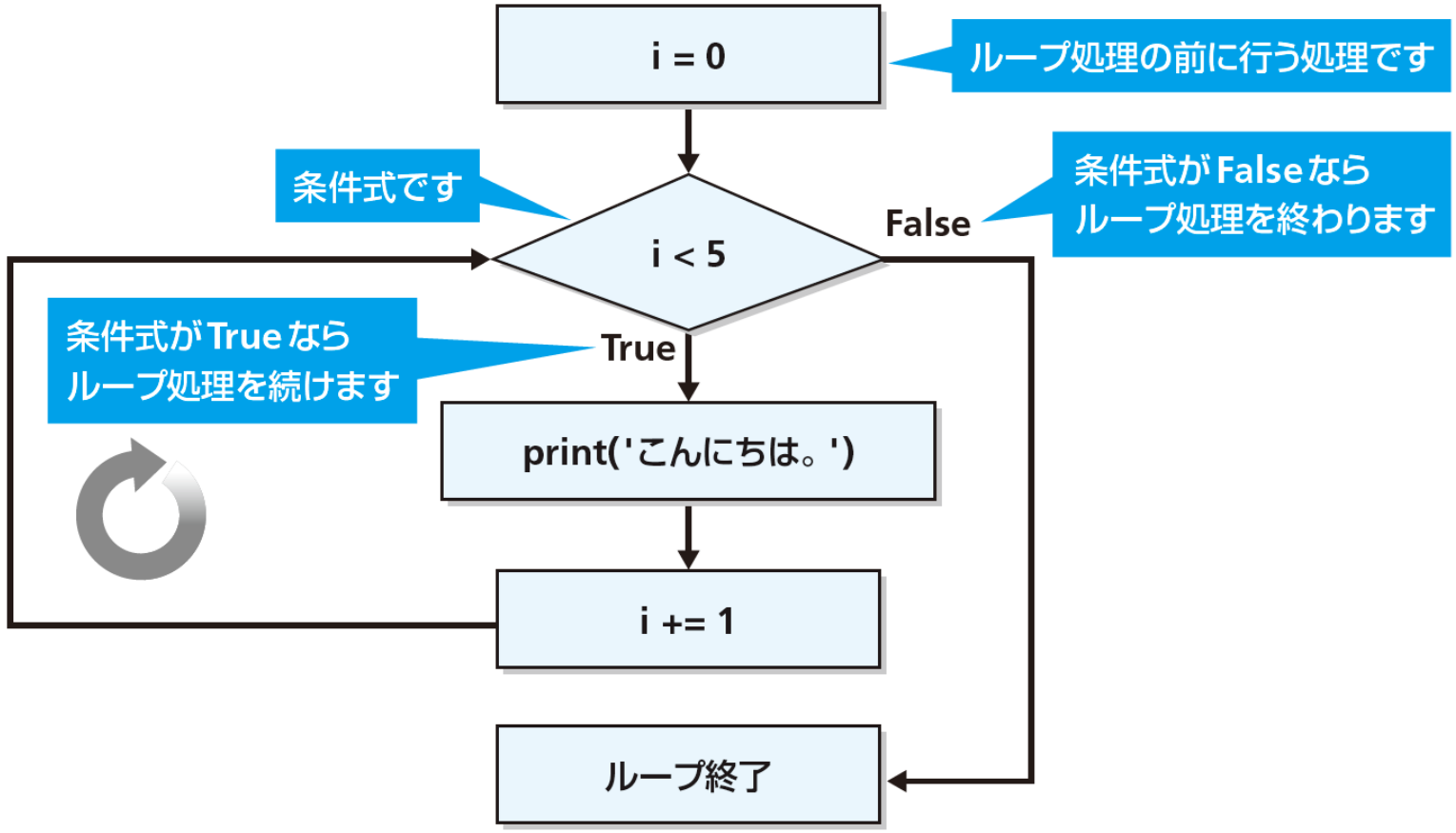
```
i = 0
while i < 5:
    print('こんにちは。')
    i += 1
```

実行結果

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

処理の流れ

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



while文の例

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

変数iの値が0より大きければ次のブロックの処理を繰り返します

変数iの値を出力します

変数iの値を5だけ減らします

実行結果

```
20
15
10
5
```

20から5ずつ小さくなる値が出力されています

ループ処理の流れの変更

break ループの処理を中断する

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

print(i, total)
```

ループ処理の流れの変更

continue ループ内の処理をスキップする

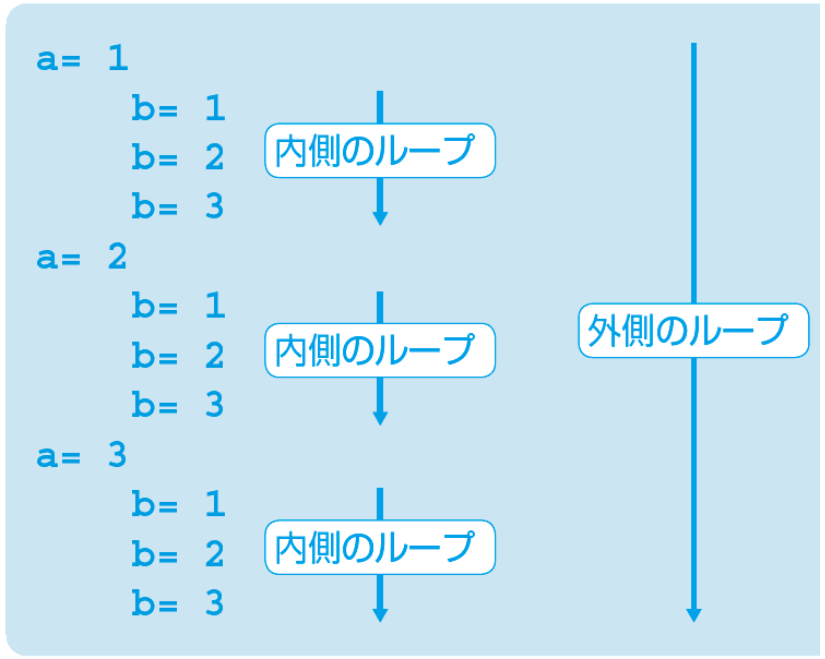
```
total = 0
for i in range(100):
    if i % 3 == 0:
        continue
    print(i)
    total += i

print('合計は', total)
```

ループ処理のネスト

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

forループの中にfor文があります

実行結果



問題 3

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

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

問題 3（解答）

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

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

while 文

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

for 文

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

問題 4

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

for 文

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

問題 4（解答）

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

for 文

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

問題 5

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

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

問題 5（解答）

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

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

まとめと復習 D

1. ユーザー定義関数
    a. 引数
    b. 戻り値
2. 数値の型
3. モジュールの利用 (import)
4. ...

ユーザー定義関数

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

← say_hello 関数の定義

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

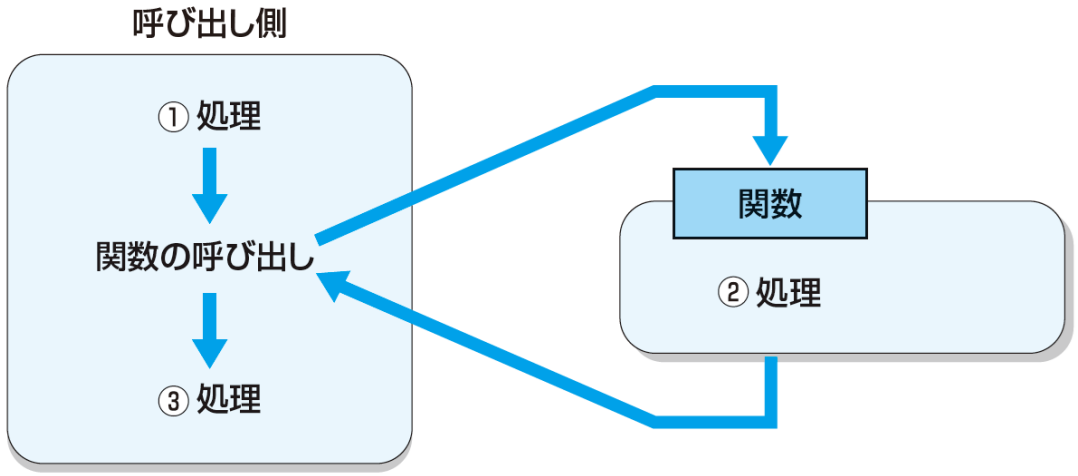
← say_hello 関数の呼び出し

← say_hello 関数の呼び出し

実行結果

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

関数呼び出しの処理の流れ




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

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

function_a()
function_b()
```

関数function_aの定義です

関数function_bの定義です

関数function_aを呼び出します

関数function_bを呼び出します

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

関数の定義位置

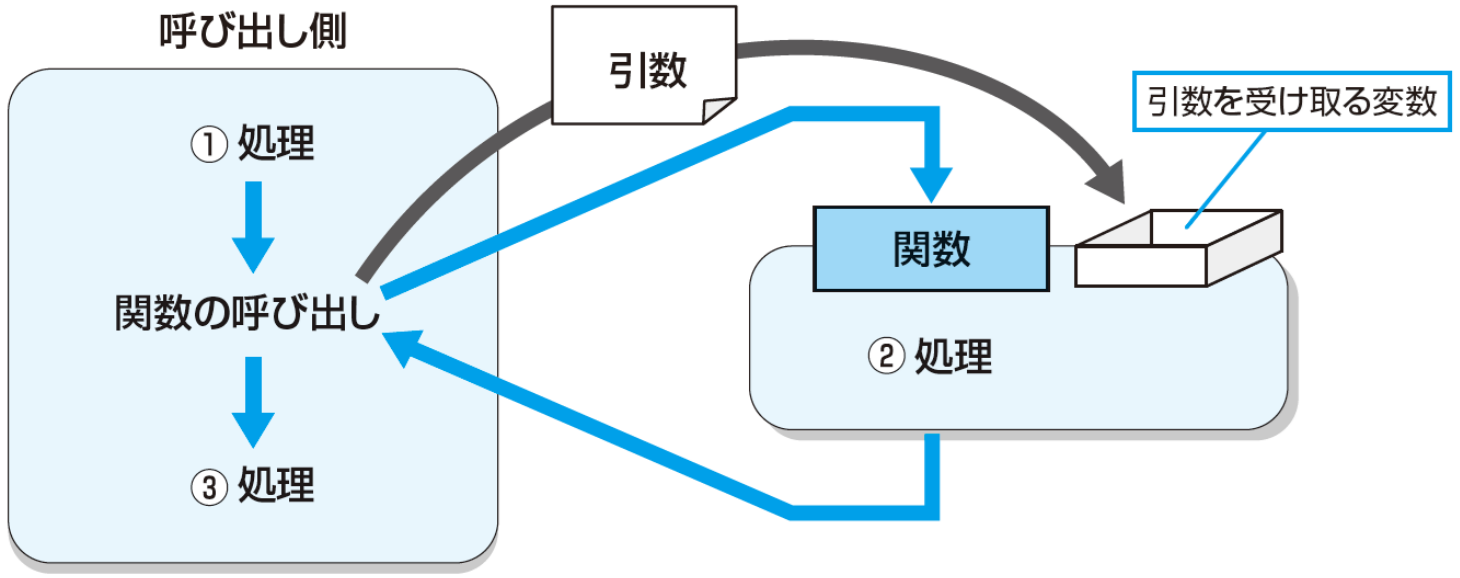
- 関数の定義は、関数の呼び出しの前に行われている必要がある

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

function_a()
function_b()

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

この時点ではfunction_bが定義されていないためエラーになります

引数 (ひきすう)

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



引数のある関数

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

countdown(3)
countdown(10)
```

関数の後ろのカッコ（）の中の変数で値を受け取る

受け取った値を使用する

値3を引数にして関数を呼び出す

値10を
〃

実行結果

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

引数が2つある関数

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

countdown(7, 3)
```

引数をカンマ区切りで並べる
（引数列とよぶ）

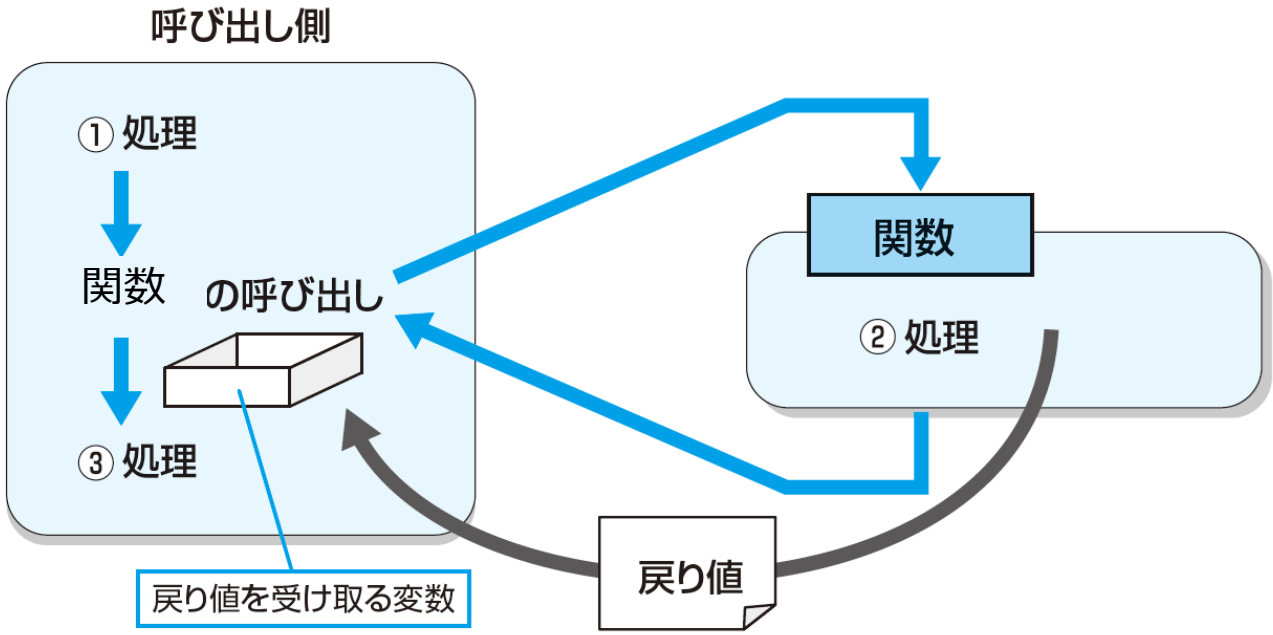
2つの値を引数にして関数を呼び出す

変数名を指定した呼び出し
（キーワード引数）

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

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

戻り値

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



戻り値のある関数

- 値を戻すには、関数の中に

```
return 値
```

と記述する

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

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

真偽値を戻り値とする関数

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

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

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

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

複数の値を戻す

- 複数の値を戻したい場合は、タプルを使用する

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

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

数値の型

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

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

0という表記はint型になります

0.0という表記はfloat型になります

演算と数値の型

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

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

int型どうしを加算、減算、乗算した結果はint型です

int型どうしで除算した結果はfloat型になります

割り切れる場合でもfloat型になります

数値の型変換

- int 型 → float 型

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

← 変数aは float 型になる

- float 型 → int 型

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

← 変数aは int 型になる
小数点以下は切り捨て

モジュールの使用

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

```
import モジュール名
```

- **random**モジュールをインポートすると、randint()などの関数を使用できるようになる

```
import random
```

randomモジュールの利用

- random モジュールに含まれる関数

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

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

← 実行のたびに異なる値が出力される

randomモジュールの利用

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

すでにrandomモジュールをインポートした後であれば記述は不要です

3つの要素を持つリストです

リストjankenに含まれる要素からランダムに1つ選びます

毎回結果が異なります

モジュールに別名をつけて使う

```
import モジュール名 as 別名
```

```
>>> import turtle as t
>>> t.left(90)
```

← mathモジュールの別名をmとする

← 別名を使って記述

モジュール名が長いときに便利

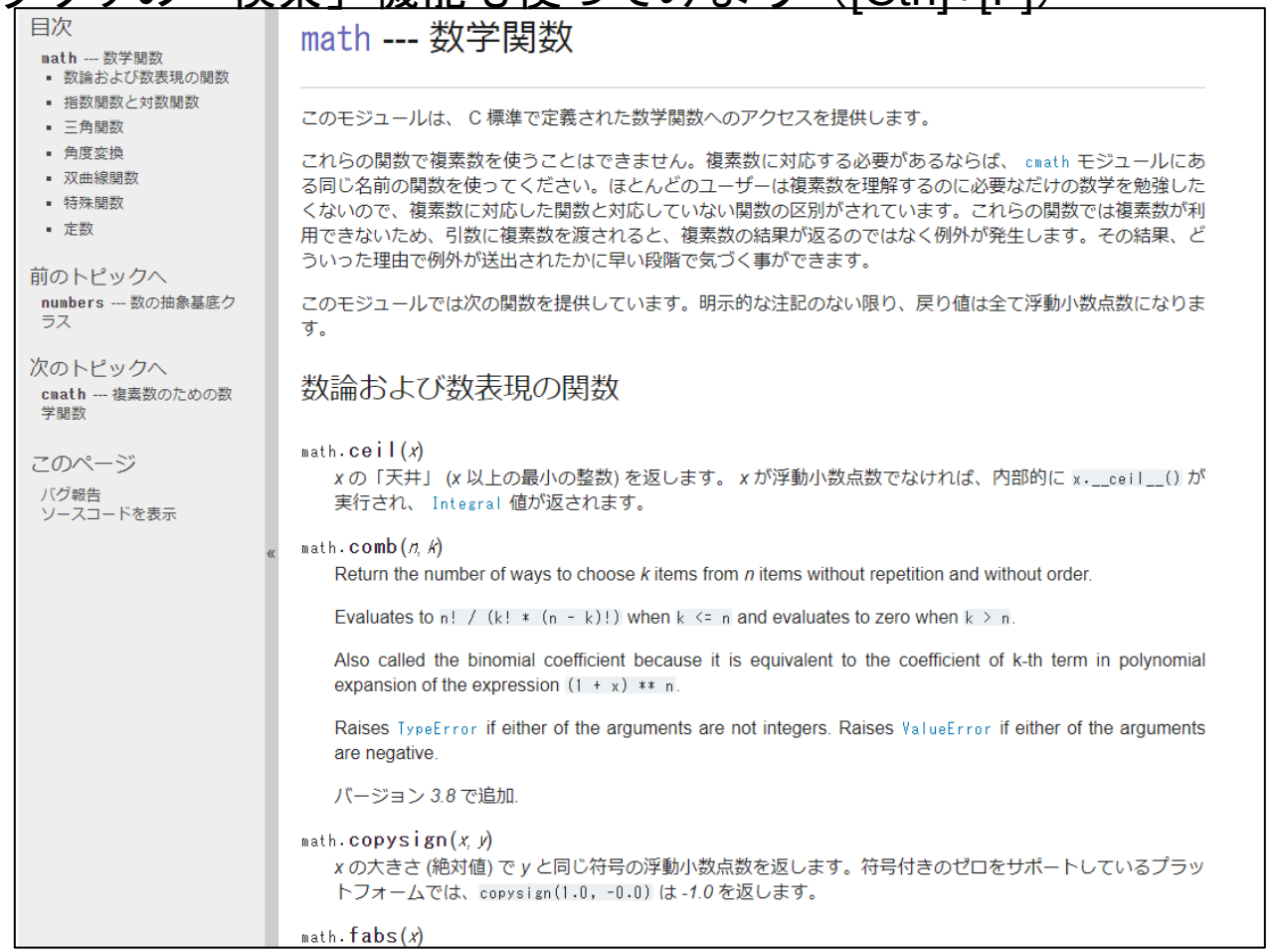
ドキュメントを読む

- Pythonの標準ライブラリのドキュメント

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

モジュールに含まれる関数を調べる

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



目次

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

前のトピックへ
numbers --- 数の抽象基底クラス

次のトピックへ
cmath --- 複素数のための数学関数

このページ
バグ報告
ソースコードを表示

math --- 数学関数

このモジュールは、 C 標準で定義された数学関数へのアクセスを提供します。

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

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

数論および数表現の関数

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

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

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

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

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

バージョン *3.8* で追加.

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

math.fabs(*x*)

まとめと復習 E (Part 1)

1. 文字列
2. リスト

---

3. リスト内包
4. その他のデータ型: 辞書と集合
5. ...

文字列

文字列の扱い

- +演算子による文字列の連結

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

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

- *演算子による連結の繰り返し

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

'ABCABCABC'になる

数値→文字列の変換

数値を文字列のようには扱えない

✕
```
a = 500 + '円'
```

str(数値)で文字列に変換する

○
```
a = str(500) + '円'
```

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

yearの値（int型）を文字列に変換してから連結しています

変数の値の埋め込み

数値を文字列に変換してから連結

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

フォーマット文字列の使用して簡潔に記述できる

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

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

フォーマット文字列の活用

フォーマット文字列 ／ f文字列

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

{変数名} 部分に**式**を入れることもできる

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

f文字列における書式

f文字列の{}の中での書式を指定できる

{変数名:書式} のように記述する）

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

```
>>> print(f'1/3={ x : .3f }' )          # x = 1/3
1/3=0.333
>>> print(f'{ y : , }円')                # y = 10 ** 6
1,000,000円
>>> print(f'答えは{ z : >8}です')          # z = 123
答えは      123です
```

文字列→数値の変換

文字列を数値のようには扱えない

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

← 文字列

← bの値は'500500'になる

int(文字列)で整数に変換する

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

← bの値は1000になる

※ 小数点を含む数値に変換するときは float(文字列)

len関数による文字列の長さの取得

len(文字列) で文字列の長さ（文字数）を取得できる

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

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

strクラスのメソッド

strクラスには文字列を操作するメソッドが定義されている

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

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

strクラスのformatメソッド

文字列に含まれる{} への変数の値の埋め込み

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

文字列の中の {} に数字が埋め込まれました

{}の中でのインデックスの指定

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

変数monthの値が埋め込まれます

変数yearの値が埋め込まれます

変数dayの値が埋め込まれます

in 演算子

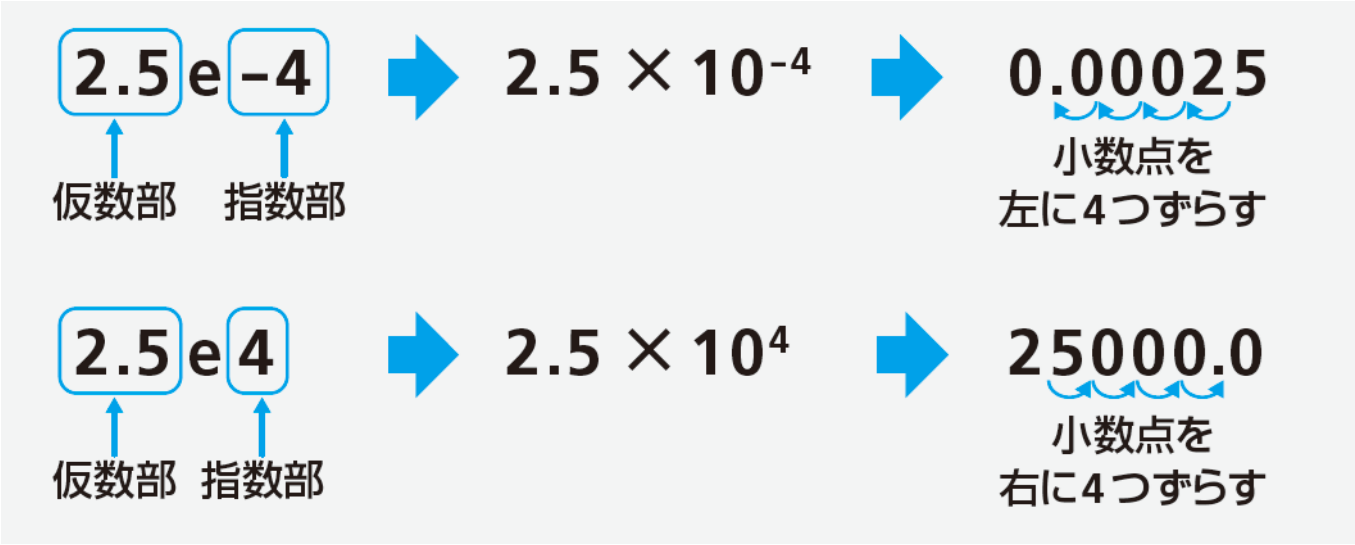
```
文字列1 in 文字列2
```

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

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

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

数値の指数表現




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

リストとタプル

リスト

リストを使って、複数の値をまとめて管理できる

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

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

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

※ インデックスは0から始まる

マイナスのインデックス

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

インデックスに**マイナスの値**も使える

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

← 末尾の要素

← 後ろから2番目の要素

← 後ろから5番目の要素

リスト内の値の変更

インデックスを指定して値を変更できる

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

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

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

← リストの全要素を出力

← 先頭の要素を99に変更

← 文字列にもできる

リストの要素数の確認

len関数でリストの要素数を取得できる

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

in 演算子

データ1 **in** データ2

データ1 が リスト等のデータ**2** に含まれると **True**、
そうでないなら **False** になる

```
>>> 20 in [10, 20, 30, 40, 50]
True
>>> 80 in [10, 20, 30, 40, 50]
False
```

```
>>> s1 = 40
>>> s2 = [10, 20, 30, 40, 50]
>>> s1 in s2
True
```

listクラスの主なメソッド

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

← a はListクラスのインスタンス

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

メソッドを使ったリストの操作

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

メソッド以外のリストの操作

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

まとめと復習 E (Part 2)

1. 文字列
2. リスト

---

3. リスト内包
4. その他のデータ型: 辞書と集合
5. ...

内包表記

for文を用いてリストの要素を作成することができる

構文

```
[式 for 変数 in 反復可能なオブジェクト]
```

例

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

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

if 構文を含む内包表記

構文

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

例

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

← aで始まる単語だけが格納されています

問題 1

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

問題１（解答）

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

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

スライス式

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

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

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

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

リストを含むリスト

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

先頭のリストの先頭の要素を参照します

先頭のリストの2番目の要素を参照します

2番目のリストの先頭の要素を参照します

2番目のリストの2番目の要素を参照します

2番目のリストの3番目の要素を参照します

タプル

- リスト同様に複数の要素を格納する

リスト

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

タプル

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

タプルは要素の値を変更できない

( )の省略とアンパック代入

タプルの作成時に ( )を省略できる

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

複数の変数へ1つずつ値を代入できる
（**アンパック代入とよぶ**）

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

複数の変数への一括代入

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

三項演算子

```
値1 if 条件式 else 値2
```

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

例

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

c の値は a になる（もし a > b なら）。そうでなければ b

※ 英語の文章のようにして左から順番に読んでいくとわかりやすい

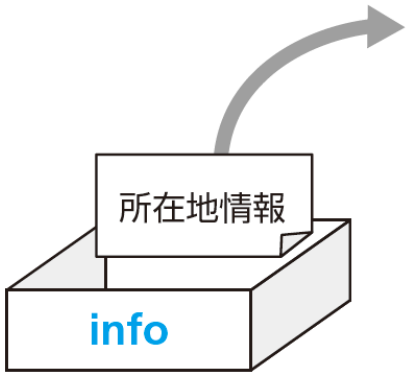
辞書と集合（セット）

辞書

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

構文　`{キー1:値1, キー2:値2, キー3:値3, … }`

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



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

辞書の要素の取得

keysメソッドによる
すべてのキーの取得

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

valuesメソッドによる
すべての値の取得

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

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

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

辞書の操作

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

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

辞書の要素の並べ替え

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

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

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

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

セット

- **値**を格納
- 要素の重複を許さない。順序が無い。

構文 `{値1, 値2, …}`

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

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

```
>>> a = {'A', 'B', 'B', 'C', 'C', 'C'}
>>> a
{'A', 'B', 'C'}          ←それぞれの値が1つずつ含まれる
>>> len(a)
3                        ←len()で要素数を調べられる
```

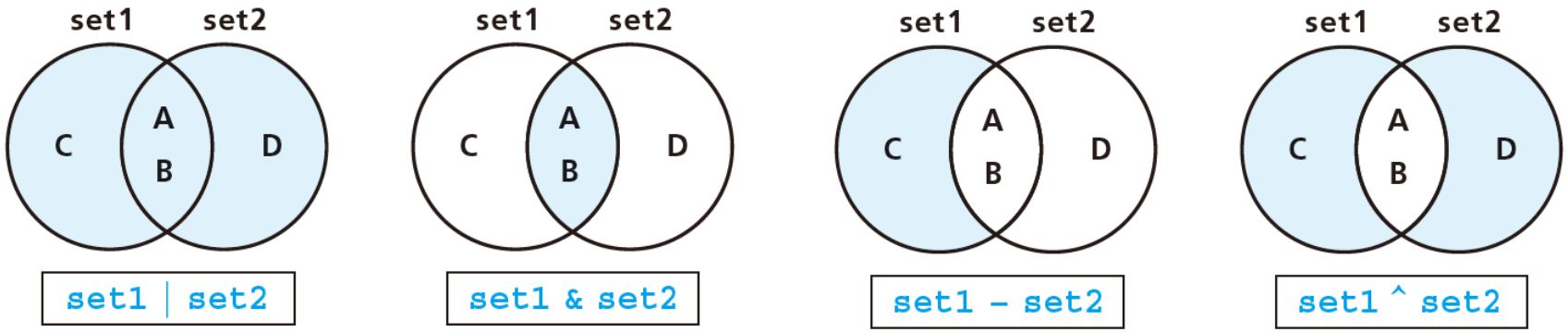
セットの操作

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

>>> 'C’ in set1
True
```

← in演算子も使える

セットどうしの演算（集合演算）

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



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

セットどうしの演算（集合演算）

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

謝辞: 教材のソース

本教材は、筑波大学 システム情報系 三谷純先生著:
『Python ゼロからはじめるプログラミング』
の副教材として作られ公開された資料に基づいています。

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

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