ダウンロード特典

さまざまなPythonの関数と記法

[ダウンロード特典]さまざまなPythonの関数と記法

本書の第4章ではPythonがもつ関数をいくつか紹介しましたが、それらはPythonに用意されている関数のうちのほんの一部であり、またその詳しい仕様やオプション引数などを紹介することはしませんでした。

そのためこの付録では、アルゴリズム実技検定で使う機会がありそうな関数や記法を第4章よりも数多く紹介し、その仕様について解説します。

ただし、検定中に欲しい機能をもつ関数を素早く探したい場合は、書籍よりもWeb検索のほうがはるかに便利でしょう。本付録はPCのない場所で学習を行うときに用いたり、ざっと眺めることで「Pythonにはこんなことができる関数が標準で用意されている」ということを把握したりするために活用していただければと思います。

この付録では、以下の関数を紹介します。

- **データ型を変換する関数**
- **数値に対する関数**
- **iterable（配列や文字列など）に対する関数**
- **配列に対する関数**
- **文字列に対する関数**

データ型を変換する関数

int：整数型への変換

文字列を整数に変換します。デフォルトでは文字列を十進数として解釈しますが、baseを指定することでほかの基数で解釈することもできます。例えば、十六進数なら16、二進数なら2が基数です。

```
int("12345")          # 12345
int("ffff", base=16)  # 65535
int("1101", base=2)   # 13
```

また、小数から整数への変換（丸め）をすることもできます。intを用いた場合、小数は0に近いほうに丸められます。

```
int(3.8)   # 3
int(-2.5)  # -2
```

float：小数型への変換

文字列を小数に変換します。通常の小数点を用いた表記だけでなく、"3e5"などの指数表記も解釈することができます。3e5は3×10^5を表します。

```
float("3.5")    # 3.5
float("-5")     # -5.0
float("3e5")    # 300000.0
```

また、整数を小数に変換することもできます。絶対値の大きな整数を変換すると誤差が発生するため注意してください。

```
float(-10)                          # -10.0
int(float(12345678901234567890))  # 12345678901234567168
```

str：文字列型への変換

さまざまなオブジェクトを文字列に変換することができます。どのような文字列になるかは、元のオブジェクト型のほうで定義されています。

```
str(5)                      # "5"
str(3.5)                    # "3.5"
str(10000000000000000.0)    # "1e+16"
str([1,2,3])                # "[1, 2, 3]"
```

ord，chr：文字と文字コード（整数）の変換

ordは1文字だけからなる文字列をその文字の文字コードに変換します。とくに半角英数字または半角記号に対しては、その文字のASCIIコードである0～127の整数を返します。

標準的な環境では、半角英数字のASCIIコードは以下のようになっています。

"0"～"9"	48～57、数字順
"A"～"Z"	65～90、アルファベット順
"a"～"z"	97～122、アルファベット順

chrは逆に、文字コードを対応する1文字からなる文字列に変換します。

アルゴリズム実技検定やプログラミングコンテストでは、この関数は文字同士のアルファベット順での「差」を扱う場合によく用いられます。例えば「"c"と"x"がアルファベット順でいくつ離れているか知りたい」という場合や、「"c"からアルファベット順で10個進んだアルファベットを知りたい」という場合には、以下のように書くことができます。

```
ord("x") - ord("c")     # 21
chr(ord("c") + 10)      # "m"
```

bool：TrueまたはFalseへの変換

オブジェクトをその「真偽」に応じてTrueまたはFalseの値に変換します。

PythonではTrueとFalseだけでなく、全てのオブジェクトに対して「真か偽かの判定」をすることができます。Python自身がもっているデータ型において「偽である」と判定されるものは、例えば以下のようなものです。

- **「偽である」ことを表現する値False**
- **「何もない」ことを表現する値None**
- **整数の0や小数の0.0など、数値型におけるゼロ**
- **空配列や空文字列など、データの集まりを表す型において空であるもの**

逆に、ゼロでない数値、空でない配列や文字列などは「真である」と判定されます。

bool関数は「真である」オブジェクトをTrueに、「偽である」オブジェクトをFalseに変換します。第4章では説明を簡潔にするため触れませんでしたが、TrueやFalse以外をif文やwhile文の条件式などに指定した場合も、同様のルールで真偽の判定が行われます。

```
bool(5)        # True
bool(0)        # False
bool([False])  # True
bool([])       # False
```

数値に対する関数

abs：絶対値

実数xに対して、abs(x)はxが正または0のときはx、負のときは-xを返します。整数と小数の両方に適用可能です。

```
abs(10)    # 10
abs(-10)   # 10
abs(-0.5)  # 0.5
```

round：四捨五入での丸め

小数を四捨五入によって整数に丸めます。ただし小数部がちょうど0.5だった場合は、近い整数のうち偶数である方に丸められます。

```
round(0.8)    # 1
round(3.4)    # 3
round(1.5)    # 2
round(0.5)    # 0
```

math.floor, math.ceil：切り上げ、切り捨て

小数を整数に切り上げ、または切り捨てます。mathモジュールのインポートが必要です。

```
import math
math.floor(3.5)  # 3
math.ceil(3.5)   # 4
```

小数を経由すると誤差が発生するため、整数同士の割り算の切り上げや切り捨てを正確に行いたいときには使わないことを推奨します。整数同士の場合は以下の方法を使ってください。

```
a, b = 7, 2
a // b             # 3   ※切り捨て
(a-1) // b + 1     # 4   ※切り上げ
```

pow：べき乗

数値のべき乗を計算します。pow(x, y)と引数を2つ指定した場合、通常のべき乗 x^y を計算します。y として小数を指定することも可能で、例えばy=0.5とすると正の平方根を計算します。

```
pow(3, 5)    # 243
pow(3.0, 5)  # 243.0
pow(3, 0.5)  # 1.7320508075688772
```

第6章で説明したように、pow(x, y, m)と引数を3つ指定することで x^y をmで割った余りを計算することも可能です。

```
pow(3, 5, 10)   # 43
```

さらにyとして負の値を指定すると、第6章のコラムで解説した「モジュラ逆数」の計算になります。計算のためにはモジュラ逆数が適切に定義できる、つまりxとmが互いに素である必要があります。

```
pow(2, -1, 10**9+7)    # 500000004
```

divmod：割り算との商と余り

割り算の商と余りからなるタプルを返します。

```
divmod(10, 3)    # (3, 1)
```

int.bit_length：二進数でのビット長

整数を二進数で表すために必要なビットの数を返します。

```
int.bit_length(3)    # 2    ※二進数で 11
int.bit_length(4)    # 3    ※二進数で 100
```

iterable(配列や文字列など)に対する関数

iterableとは、配列、文字列、タプル、範囲オブジェクト、辞書などが共通でもっている性質のようなものです。要素が順番に並んでいて、for文で繰り返せるものがiterableだと理解しておくと良いです。

このiterable全般に対して適用可能な関数がいくつかあります。とりあえず「文字列や配列に対して共通で使える関数」と覚えておくと良いでしょう。

`len`: 要素の個数

要素の個数を返します。配列や文字列だけでなく辞書などにも適用できます。

```
len([3, 2, 5])  # 3
len("abcd")     # 4
len([])         # 0
```

`max, min`: 最大値、最小値

それぞれ、最大である要素と最小である要素を返します。要素同士の大小比較が可能である必要があります。

主に数値を要素とする配列に対して使いますが、文字列に対して使うこともできます。その場合はそれぞれの文字が文字コードの値で比較されるため、例えばアルファベットの小文字だけからなる文字列に対して使うと、maxではアルファベット順で最も後ろにある文字、minでは最も前にある文字を返します。

```
max([3, 2, 5])      # 5
min([3, 2, 5])      # 2
max("programming")  # "r"
```

`sum`: 総和

数値を要素とする配列に対して、その総和を返します。

```
sum([3, 2, 5])      # 10
```

all, any：各要素の真偽に基づく判定

それぞれの要素の真偽によって、以下のようにTrueまたはFalseを返します。ほとんどの場合では配列またはタプルに適用し、文字列などに適用することはほぼありません。

all	配列の全ての要素が真であれば(より正確には、偽の要素が存在しなければ)Trueを返し、そうでなければFalseを返す。 空配列に対して all([]) はTrueであることに注意。
any	配列に少なくとも1つ真の要素があればTrueを返し、そうでなければFalseを返す。

map：要素それぞれに関数を適用した結果を返す

mapは第1引数に関数を、第2引数に配列などを取ります。例えば関数をfとすると、map(f, [a, b, c]) は [f(a), f(b), f(c)] に相当するものを返します。

厳密にはmapが返すものは「イテレータ」と呼ばれるオブジェクトであり、配列として使うためにはlist関数で変換する必要があります。

関数は自分で定義したものでも、Pythonそのものの機能としてもつものでも構いません。

```
list(map(int, ["1", "2", "10"]))    # [1, 2, 10]
```

```
def square(x):
    return x*x

list(map(square, [1, 2, 3]))        # [1, 4, 9]
```

zip：配列たちから1つずつ要素を取って組み合わせる

zipは引数として配列などを複数とります。配列の長さだけ「各配列から要素を1つずつ取り、タプルとして組み合わせる」ことを繰り返し、それらを集めたイテレータを返します。list関数で配列にすることができます。

```
a = [1, 2, 3]
b = [11, 12, 13]
list(zip(a, b))   # [(1, 11), (2, 12), (3, 13)]
```

配列に対する関数

sort, sorted: ソート

配列を小さい方から順に並べ替えます。キーワード引数 reversed=True を指定すると大きい方から順に並び替えます。

配列そのものを変更するかどうかによって使う関数が異なります。配列の変数名を arr として、arr そのものを変更したいときには arr.sort() と書きます。

```
arr = [3, 2, 5]

arr.sort()
arr            # [2, 3, 5]

arr.sort(reverse=True)
arr            # [5, 3, 2]
```

対して、arr そのものは変更せずにソートした結果の配列を生成したい場合は sorted(arr) と書きます。

```
arr = [3, 2, 5]

sorted(arr)                 # [2, 3, 5]
sorted(arr, reverse=True)   # [5, 3, 2]

arr                         # [3, 2, 5]     ※変更されない
```

sorted は文字列に対しても適用可能ですが、1文字ずつ配列に分割された後にソートされ、配列として結果が返ってくるので注意してください。文字列に戻したい場合はjoinを使います。

```
sorted("dabc")           # ["a", "b", "c", "d"]
''.join(sorted("dabc"))  # "abcd"
```

reverse, reversed: 反転

配列を反転させます。

配列そのものを変更するかどうかによって使う関数が異なります。配列の変数名を arr として、arr

そのものを変更したいときには arr.reverse() と書きます。

```
arr = [3, 2, 5]
arr.reverse()
arr           # [5, 2, 3]
```

対して、arr そのものは変更せずにソートした結果の配列を生成したい場合はreversedを使います。reversedは配列を逆順に扱うイテレータを返すため、listと組み合わせることで配列を反転させることができます。ただし、後に説明するスライス記法を用いたほうが簡単です。

```
arr = [3, 2, 5]
list(reversed(arr))  # [5, 2, 3]
arr[::-1]            # [5, 2, 3]
```

in演算子：要素が含まれているか判定

x in arr は、配列arrに要素xが含まれているかどうかを判定します。結果を逆にしたい場合はnot inを使います。

```
3 in [0, 1, 2, 3]  # True
5 in [0, 1, 2, 3]  # False

3 not in [0, 1, 2, 3]  # False
```

+演算子：配列の結合

2つの配列を結合した結果を返します。

```
[0, 1] + [2, 3]   # [0, 1, 2, 3]
```

*演算子：配列の繰り返し

arr * nは、配列arrをn回繰り返した結果を返します。

```
[0, 1] * 3   # [0, 1, 0, 1, 0, 1]
```

注意点として、もし配列の要素が配列などのミュータブルなものであった場合、繰り返したそれぞれの

要素は同一のデータを参照します。とくに二重配列の初期化などに用いるとバグの原因になりやすいため、要素が配列などである場合には使わないようにしましょう。下記のコードでは2つの要素として同一の配列が参照されているため、1箇所の書き換えで2箇所の値が変わっているように見えてしまっています。

```
a = [[0, 0]] * 2
a[0][0] = 1
a                   # [[1, 0], [1, 0]]
```

index：配列の中の要素を探す

arr.index(x) は、配列arrの中でxが最初に登場するインデックスを返します。ただし、xが存在しない場合には実行時エラーとなるので注意が必要です。

```
[3, 6, 2, 1].index(6)    # 1
[3, 6, 2, 1].index(7)    # 実行時エラーが発生
```

count：配列に値が登場する個数を数える

arr.count(x) は、配列arrの中にxが登場する個数を返します。

```
[1, 3, 2, 2, 1].count(1)    # 2
```

append：配列に要素を追加する

arr.append(x) は、配列arrの末尾要素としてxを追加します。

```
arr = [0, 1, 2]
arr.append(5)
arr             # [0, 1, 2, 5]
```

clear：配列の全ての要素を取り除く

arr.clear() は、配列arrの全ての要素を取り除きます。

```
arr = [0, 1, 2]
arr.clear()
arr             # []
```

insert：配列に要素を挿入

arr.insert(i, x) は、配列arrのインデックスiの位置に要素xを挿入します。

```
arr = [0, 1, 2]
arr.insert(1, 5)
arr             # [0, 5, 1, 2]
```

pop：配列から要素を削除

arr.pop() は、配列arrの末尾要素を取り除きます。

```
arr = [0, 1, 2, 3]
arr.pop()
arr             # [0, 1, 2]
```

末尾以外の要素を取り除きたい場合は、その要素のインデックスを指定します。

```
arr = [0, 1, 2, 3]
arr.pop(1)
arr             # [0, 2, 3]
```

文字列に対する関数

ここでは、全ての文字列が半角英数字または半角記号であることを前提として解説します。アルゴリズム実技検定において、これら以外の文字を使うことはほぼないでしょう。

文字列はイミュータブルであるため、文字列自身を変更する関数は存在しません。

in演算子：部分文字列として含まれているか判定

t in s は、文字列sの部分文字列としてtが含まれていればTrue、含まれていなければFalseを返します。

```
"abcde".startswith("bcd")    # True
"abcde".startswith("bce")    # False
```

count：部分文字列の登場回数

s.count(t) は、文字列sの部分文字列として**重複せずに**文字列tを最大いくつ取れるかを返します。一般に「部分文字列として登場する回数」というと、重複は考慮せずに部分文字列として取り出せる場所の数を表しますが、そうではないことに注意してください。

例えば "aaaaa" に部分文字列として "aa" が登場する回数は、一般的には4回として扱われますが、"aaaaa".count("aa") は2を返します。

tを1文字の文字列とすることで、単にsに登場する特定文字の個数を数えることができます。

```
"aaaaa".count("aa")    # 2
"aaaaa".count("a")     # 5
```

startswith, endswith：接頭辞または接尾辞の判定

s.startswith(t) は、文字列sが文字列tから始まっていればTrue、そうでなければFalseを返します。

```
"abcde".startswith("abc")    # True
"abcde".startswith("bc")     # False
```

s.endswith(t) は、文字列sが文字列tで終わっていればTrue、そうでなければFalseを返します。

```
"abcde".startswith("cde")    # True
"abcde".startswith("cd")     # False
```

find, rfind：部分文字列の登場位置

s.find(t) は、文字列sの部分文字列として文字列tが最初に登場するインデックスを返します。もし登場しなければ-1を返します。

tを1文字の文字列とすることで、単にsに最初に登場する特定文字のインデックスを知ることができます。

```
"abcdebc".find("bc")    # 1
"abcdebc".find("f")     # -1
```

s.rfind(t) は逆に、文字列sの部分文字列として文字列tが最後に登場するインデックスを返します。もし登場しなければ-1を返します。

```
"abcdebc".rfind("bc")   # 5
"abcdebc".rfind("f")    # -1
```

join：配列などを文字列として結合

s.join(arr) は、配列arrの各要素を文字列に変換し、文字列sで挟んで結合します。sを空文字列にするとそのまま結合します。

```
"-".join([1, 2, 3])    # "1-2-3"
```

split：文字列を指定文字で分割

s.split(t) は、文字列tを区切りとして文字列sを分割した配列を返します。

```
"1-2-3".split("-")    # ["1", "2", "3"]
```

isalpha, isdigit：文字種類の判定

s.isalpha() は、sが1文字以上の英字のみで構成されていればTrue、そうでなければFalseを返します。

```
"abcde".isalpha()    # True
"abcd5".isalpha()    # False
```

s.isdigit() は、sが1文字以上の数字のみで構成されていればTrue、そうでなければFalseを返します。

```
"12345".isdigit()    # True
"1234e".isdigit()    # False
```

isupper, islower：大文字、小文字の判定

s.isupper() は、sに1文字以上の英大文字が含まれ、かつ英小文字が含まれていなければTrue、そうでなければFalseを返します。数字や記号は結果に影響しません。

```
"ABC_123".isupper()    # True
"ABc_123".isupper()    # False
```

s.islower() は、sに1文字以上の英小文字が含まれ、かつ英大文字が含まれていなければTrue、そうでなければFalseを返します。数字や記号は結果に影響しません。

```
"abc_123".islower()    # True
"Abc_123".islower()    # False
```

lower, upper：大文字、小文字に変換

s.upper() は、sに含まれる小文字を全て大文字に変換した文字列を返します。

```
"abc_123".upper()    # "ABC_123"
```

s.lower() は、sに含まれる大文字を全て小文字に変換した文字列を返します。

```
"ABC_123".lower()    # "abc_123"
```

capitalize：先頭だけを大文字に

s.capitalize() は、文字列sの先頭を大文字に、それ以外の文字を小文字にした文字列を返します。数字や記号はそのまま残されます。

```
"aa-BB-00".capitalize()    # "Aa-bb-00"
```

strip, lstrip, rstrip：前後の空白文字の除去

s.strip() は、文字列sの先頭と末尾に存在する空白文字を全て除去した文字列を返します。空白文字とは半角スペース、タブ文字、改行文字などです。先頭だけ除去する場合はlstrip, 末尾だけ除去する場合はrstripを用います。

```
"   abc     ".capitalize()    # "abc"
```