@IT
atmarkIT

解決！ Python
文字列チェック編

かわさきしんじ，Deep Insider 編集部［著］

※本 eBook の制作の都合上、Python コード中のシングルクオートやダブルクオートなどの記号類が、コードの実行確認に使用した Python 処理系ではシングルクオートやダブルクオートなどとして解釈されない文字となっています。コードをコピー&ペーストして使う際にはご注意ください。

[解決！Python]文字列が大文字だけで構成されているか、小文字だけで構成されているかを区別するには（isupper メソッド、islower メソッド）

文字列の isupper メソッド／ islower メソッドを使って、文字列が大文字だけで構成されているか、あるいは小文字だけで構成されているかを調べる方法と考慮点を紹介する。

かわさきしんじ，Deep Insider 編集部（2023 年 11 月 07 日）

```
# 文字列が大文字のみで構成されているかどうかを調べる
s = 'Python'
result = s.isupper()
print(result)  # False

s = 'PYTHON'  # s = s.upper()
result = s.isupper()
print(result)  # True

s = ' +P=Y=T=H=O=N+ '  # 大文字小文字の区別がない文字は判定の対象とはならない
result = s.isupper()
print(result)  # True

s = '012= =+'  # 大文字小文字の区別がない文字だけで構成されている場合
result = s.isupper()
print(result)  # False

# 文字列が小文字のみで構成されているかどうかを調べる
s = 'Python'
result = s.islower()
print(result)  # False

s = 'python'  # s = s.lower()
result = s.islower()
print(result)  # True

s = ' +p=y=t=h=o=n+ '  # 大文字小文字の区別がない文字は判定の対象とはならない
result = s.islower()
print(result)  # True

s = '012= =+'  # 大文字小文字の区別がない文字だけで構成されている場合
result = s.islower()
print(result)  # False
```

メソッド	説明
isupperメソッド	文字列が大文字（と大文字小文字の区別がない文字）のみで構成されているとTrueを、そうでなければFalseを返す。大文字小文字の区別がない文字のみで構成されている場合にはFalseを返す
islowerメソッド	文字列が小文字（と大文字小文字の区別がない文字）のみで構成されているとTrueを、そうでなければFalseを返す。大文字小文字の区別がない文字のみで構成されている場合にはFalseを返す

文字列の isupper メソッド／ islower メソッド

→目次に戻る

isupper メソッド：文字列が大文字のみで構成されているかどうかを調べる

文字列が大文字のみで構成されているかどうかを調べるには、文字列の isupper メソッドを使用する。以下に例を示す。

```
s = 'Python'
result = s.isupper()
print(result)  # False

s = 'PYTHON'  # s = s.upper()
result = s.isupper()
print(result)  # True
```

1 つ目の例では文字列 s は大文字と小文字で構成されている。そのため、isupper メソッドは False を返す。2 つ目の例では文字列 s は大文字だけで構成されているので、isupper メソッドは True を返している。

文字列に大文字小文字の区別がない文字が含まれていた場合の扱いには注意すること。isupper メソッド（と islower メソッド）はそれらについては大文字小文字の判定には含まない。以下に例を示す。

```
s = ' +P=Y=T=H=O=N+ '  # 大文字小文字の区別がない文字は判定の対象とはならない
result = s.isupper()
print(result)  # True
```

この例では、文字列 s には大文字に加えて半角の空白文字など大文字小文字の区別がない文字が含まれている。この場合、大文字小文字の区別がある文字だけが判定の対象となり、結果として isupper メソッドは True を返している。

では、大文字小文字の区別がない文字だけで文字列が構成されている場合はどうかというと、そのときには以下に示すように False が返される。

```
s = '012= =+'  # 大文字小文字の区別がない文字だけで構成されている場合
result = s.isupper()
print(result)  # False
```

islower メソッド：文字列が小文字のみで構成されているかどうかを調べる

同様に、文字列が小文字のみで構成されているかどうかを調べるには、文字列の islower メソッドを使用する。以下に例を示す。

```
s = 'Python'
result = s.islower()
print(result)  # False

s = 'python'  # s = s.lower()
result = s.islower()
print(result)  # True
```

1 つ目の例では、文字列 s には大文字と小文字が混在しているので判定結果は False となる。2 つ目の例では文字列 s には小文字のみが含まれているので結果は True となる。

大文字小文字の区別がない文字が含まれているときの扱いは isupper メソッドと同様だ。区別のない文字は判定の対象とはならず、区別のある文字だけを見て結果が決まる。

```
s = ' +p=y=t=h=o=n+ '  # 大文字小文字の区別がない文字は判定の対象とはならない
result = s.islower()
print(result)  # True
```

大文字小文字の区別がない文字だけで文字列が構成されているときも isupper メソッドと同様、判定結果は False となる。

```
s = '012= =+'  # 大文字小文字の区別がない文字だけで構成されている場合
result = s.islower()
print(result)  # False
```

［解決！Python］文字列が英数字（文字および数字）のみで構成されているかどうかを判定するには（isalnum ／ isascii メソッド、正規表現）

isalnum メソッドと isascii メソッドと組み合わせて、あるいは正規表現を使って文字列が半角英数字のみで構成されているかどうかを調べる方法を紹介する。

かわさきしんじ，Deep Insider 編集部（2021 年 02 月 26 日）

```
# 文字列が半角の英数字のみで構成されているかどうかを判定
def isalnum_ascii(s):  # 文字列のメソッドを使用
    return True if s.isalnum() and s.isascii() else False

import re

def isalnum_ascii_re(s):  # 正規表現を使用
    return True if re.fullmatch(r'[¥d¥w]+', s, re.ASCII) else False

s1 = '123abc'
s2 = '123abc '
s3 = '123漢字ひらカタ'

r1 = isalnum_ascii(s1)
r2 = isalnum_ascii(s2)
r3 = isalnum_ascii(s3)
print(f'{s1} isalnum_ascii: {r1}')  # 123abc isalnum_ascii: True
print(f'{s2} isalnum_ascii: {r2}')  # 123abc isalnum_ascii: False
print(f'{s3} isalnum_ascii: {r3}')  # 123漢字ひらカタ isalnum_ascii: False

r1 = isalnum_ascii_re(s1)
r2 = isalnum_ascii_re(s2)
r3 = isalnum_ascii_re(s3)
print(f'{s1} isalnum_ascii_re: {r1}')  # 123abc isalnum_ascii_re: True
print(f'{s2} isalnum_ascii_re: {r2}')  # 123abc isalnum_ascii_re: False
print(f'{s3} isalnum_ascii_re: {r3}')  # 123漢字ひらカタ isalnum_ascii_re: False

# 文字列が英数字（文字および数字）のみで構成されているかどうかを判定
s1 = '123abc'
s2 = '123abc '
s3 = '123漢字ひらカタ'

r1 = s1.isalnum()
r2 = s2.isalnum()
r3 = s3.isalnum()
print(f'{s1} isalnum: {r1}')  # 123abc isalnum: True
print(f'{s2} isalnum: {r2}')  # 123abc isalnum: True
print(f'{s3} isalnum: {r3}')  # 123漢字ひらカタ isalnum: True
```


→目次に戻る

文字列が英数字のみで構成されているかどうかを isalnum メソッドや正規表現を使って判定する

Python の文字列には isalnum メソッドがある。このメソッドは「文字列中の全ての文字が英数字で、かつ 1 文字以上あるなら True を、そうでなければ False」を返す。実際には、このメソッドは文字列中の全ての文字が、isalpha ／ isdecimal ／ isdigit ／ isnumeric のいずれかのメソッドで True を返すのであれば True を、そうでなければ False を返す。このことから、このメソッドで「文字列が半角英数字のみで構成されているかどうか」を判定する際には注意が必要となる。

以下に例を示す。

```
s1 = '123abc'
s2 = '１２３abc'
s3 = '123漢字ひらカタ'

r1 = s1.isalnum()
r2 = s2.isalnum()
r3 = s3.isalnum()
print(f'{s1} isalnum: {r1}')  # 123abc isalnum: True
print(f'{s2} isalnum: {r2}')  # １２３abc isalnum: True
print(f'{s3} isalnum: {r3}')  # 123漢字ひらカタ isalnum: True
```

変数 s1、s2、s3 にはそれぞれ半角英数字のみ、全角英数字のみ、半角数字と漢字／ひらがな／カタカナで構成される文字列が代入されている。しかし、これらに対して isalnum メソッドを呼び出すと、その結果は全て True となる。

既に述べた通り、isalnum メソッドは文字列が isalpha ／ isdecimal ／ isdigit ／ isnumeric メソッドで True と判定される文字だけで構成されていると True を返す。そして、これらのメソッドは全角英数字や漢字、ひらがな、カタカナについても True を返すようになっている。以下は半角英数字、全角英数字、漢字、ひらがな、カタカナについて isalpha メソッドと isdecimal メソッドを呼び出した結果だ。

```
s1 = 'ABC'
s2 = 'ABC '
s3 = ' 漢字ひらがなカタカナ '
s4 = '123'
s5 = '123'

r1 = s1.isalpha()
r2 = s2.isalpha()
r3 = s3.isalpha()
print(f'{s1} isalpha: {r1}')  # ABC isalpha: True
print(f'{s2} isalpha: {r2}')  # ABC isalpha: True
print(f'{s3} isalpha: {r3}')  # 漢字ひらがなカタカナ isalpha: True

r4 = s4.isdecimal()
r5 = s5.isdecimal()
print(f'{s4} isdecimal: {r4}')  # 123 isdecimal: True
print(f'{s5} isdecimal: {r5}')  # 123 isdecimal: True
```

全角のアルファベットや漢字、ひらがな、カタカナについて isalpha メソッドが True を返していることと、全角数字について isdecimal メソッドが True を返していることに注目されたい。こうしたことから、isalnum メソッドだけでは、文字列が半角英数字のみで構成されているかどうかは分からない（もちろん、文字と数字でのみ文字列が構成されているかどうかを調べるのであれば isalnum メソッドをそのまま使えばよい）。

文字列が半角英数字のみで構成されているかどうかを判定するには、isalnum メソッドと isascii メソッドを組み合わせるとよい。isascii メソッドは文字列を構成する全ての文字が ASCII の範囲に含まれるものであれば True を、そうでなければ False を返送する。isalnum メソッドの結果が True かつ isascii メソッドの結果が True であれば、その文字列は半角英数字で構成されていると考えられるだろう。

以下では、そうした判定を行う関数を定義している。

```
def isalnum_ascii(s):  # 文字列のメソッドを使用
    return True if s.isalnum() and s.isascii() else False

s1 = '123abc'
s2 = '123abc '
s3 = '123漢字ひらカタ '

r1 = isalnum_ascii(s1)
r2 = isalnum_ascii(s2)
r3 = isalnum_ascii(s3)
print(f'{s1} isalnum_ascii: {r1}')  # 123abc isalnum_ascii: True
print(f'{s2} isalnum_ascii: {r2}')  # 123abc isalnum_ascii: False
print(f'{s3} isalnum_ascii: {r3}')  # 123漢字ひらカタ isalnum_ascii: False
```

同様な判定は正規表現を使って次のようにも行える。

```
import re

def isalnum_ascii_re(s):  # 正規表現を使用
    return True if re.fullmatch(r'[¥d¥w]+', s, re.ASCII) else False

s1 = '123abc'
s2 = '１２３abc'
s3 = '123漢字ひらカタ'

r1 = isalnum_ascii_re(s1)
r2 = isalnum_ascii_re(s2)
r3 = isalnum_ascii_re(s3)
print(f'{s1} isalnum_ascii_re: {r1}')  # 123abc isalnum_ascii_re: True
print(f'{s2} isalnum_ascii_re: {r2}')  # １２３abc isalnum_ascii_re: False
print(f'{s3} isalnum_ascii_re: {r3}')  # 123漢字ひらカタ isalnum_ascii_re: False
```

こちらのコードでは、正規表現の文字クラス「¥d」と「¥w」を使用している。前者は Unicode の General_Category プロパティの値が「Nd」にあるものにマッチし、後者は単語を構成する文字にマッチする。re.fullmatch 関数にパターンとして「[¥d¥w]+」を渡すことで、文字列がそうした文字のみで構成されているかどうかを判定し、戻り値があれば（全体が英数字のみで構成されていれば）True を、そうでなければ False を返すようにしている。ただし、「¥d」「¥w」は ASCII の範囲外にある文字についてもマッチするので（全角数字など）、ここでは第 3 引数に re.ASCII フラグを指定している。

［解決！Python］文字列が数字だけで構成されているかどうかを判定するには（isdecimal ／ isdigit ／ isnumeric ／ isascii メソッド）

isdecimal メソッドを使って、文字列が数字だけで構成されているかどうかを判定する方法を紹介。isdigit ／ isnumeric メソッドとの違いも取り上げる。

かわさきしんじ，Deep Insider 編集部（2021 年 02 月 12 日）

```
# 文字列が半角 10 進数字のみで構成されているかどうかを判定する関数
def isascnum(s):
    return True if s.isdecimal() and s.isascii() else False

s1, s2, s3 = '123', '１２３', '123foo'

r1 = isascnum(s1)
r2 = isascnum(s2)
r3 = isascnum(s3)
print(f'{s1} isascnum: {r1}')  # 123 isascnum: True
print(f'{s2} isascnum: {r2}')  # １２３ isascnum: False
print(f'{s3} isascnum: {r3}')  # 123foo isascnum: False

# 文字列が 10 進数字のみで構成されているかどうかを判定
s1, s2, s3 = '123', '１２３', '123foo'

r1 = s1.isdecimal()
r2 = s2.isdecimal()  # 全角数字は True となる
r3 = s3.isdecimal()
print(f'{s1} isdecimal: {r1}')  # 123 isdecimal: True
print(f'{s2} isdecimal: {r2}')  # １２３ isdecimal: True
print(f'{s3} isdecimal: {r3}')  # 123foo isdecimal: False

s1, s2 = '-1', '1.23'
r1 = s1.isdecimal()  # 符号が入っていると False
r2 = s2.isdecimal()  # 小数点が入っていると False
print(f'{s1} isdecimal: {r1}')  # -1 isdecimal: False
print(f'{s2} isdecimal: {r2}')  # 1.23 isdecimal: False

# isdecimal ／ isdigit ／ isnumeric メソッドの違い
s = '１２３'  # 10 進数字は全て True
r1 = s.isdecimal()
r2 = s.isdigit()
r3 = s.isnumeric()
print(f'{s} isdecimal: {r1}')  # １２３ isdecimal: True
print(f'{s} isdigit: {r2}')  # １２３ isdigit: True
print(f'{s} isnumeric: {r3}')  # １２３ isnumeric: True

s = '¥u00b9¥u00b2¥u00b3'  # 上付き数字の 1、2、3 は isdigit ／ isnumeric では True
```


→目次に戻る

```
r1 = s.isdecimal()
r2 = s.isdigit()
r3 = s.isnumeric()
print(f'{s} isdecimal: {r1}')  # ¹²³ isdecimal: False
print(f'{s} isdigit: {r2}')  # ¹²³ isdigit: True
print(f'{s} isnumeric: {r3}')  # ¹²³ isnumeric: True

s = '二億四千万'  # 漢数字などは isnumeric でのみ True
r1 = s.isdecimal()
r2 = s.isdigit()
r3 = s.isnumeric()
print(f'{s} isdecimal: {r1}')  # 二億四千万 isdecimal: False
print(f'{s} isdigit: {r2}')  # 二億四千万 isdigit: False
print(f'{s} isnumeric: {r3}')  # 二億四千万 isnumeric: True
```

文字列が 10 進数字のみで構成されているかどうかを判定

文字列の isdecimal メソッドを使うと、その文字列が 10 進数字のみで構成されているかどうかを判定できる。

以下に例を示す。

```
s1, s2, s3 = '123', '１２３', '123foo'

r1 = s1.isdecimal()
r2 = s2.isdecimal()  # 全角数字は True となる
r3 = s3.isdecimal()
print(f'{s1} isdecimal: {r1}')  # 123 isdecimal: True
print(f'{s2} isdecimal: {r2}')  # １２３ isdecimal: True
print(f'{s3} isdecimal: {r3}')  # 123foo isdecimal: False
```

ただし、上記例を見ると分かるが、全角数字のみで構成される文字列 '１２３' についても isdecimal メソッドは True を返す。このメソッドは、Unicode の General_Category プロパティの値が 'Nd' であるものを 10 進数字として判定するようになっているからだ（Unicode Consortium が提供している UnicodeData.txt ファイル）の 3 番目のフィールドを参照）。これらの数字は int 関数で整数値に変換可能でもある。

このため、文字列が半角数字だけを含んでいるかどうかを知りたいときには、isascii メソッドを組み合わせる必要がある。

```
s = '123'
if s.isdecimal() and s.isascii():
    print(f'{s} includes only ASCII digits')
else:
    print(f'{s} includes decimal characters')
# 出力結果：
# 123 include decimal characters
```

これらを組み合わせて以下のような関数を定義してもよいだろう。

```
def isascnum(s):
    return True if s.isdecimal() and s.isascii() else False

s1, s2, s3 = '123', '123', '123foo'

r1 = isascnum(s1)
r2 = isascnum(s2)
r3 = isascnum(s3)
print(f'{s1} isascnum: {r1}')  # 123 isascnum: True
print(f'{s2} isascnum: {r2}')  # 123 isascnum: False
print(f'{s3} isascnum: {r3}')  # 123foo isascnum: False
```

注意点としては、isdecimal メソッドは符号や小数点を含んだ文字列に対しては False を返すことが挙げられる。

```
s1 = '-12'
s2 = '1.23'
r1, r2 = s1.isdecimal(), s2.isdecimal()
print(f'{s1} isdecimal: {r1}')  # -12 isdecimal: False
print(f'{s2} isdecimal: {r2}')  # 1.23 isdecimal: False
```

符号や小数点を含んだ文字列を数値に変換できるかどうかを判定する方法については「文字列が数値へ変換可能かどうかを判定するには（int ／ float 関数の例外、re.fullmatch 関数）」を参照されたい。

isdecimal ／ isdigit ／ isnumeric メソッド

通常は上で述べた isdecimal メソッドを使えば、おおよそ 10 進数字として扱われるものを判定できる。しかし、Python の文字列には isdigit メソッドと isnumeric メソッドの 2 つのメソッドがある。isdecimal メソッドを含むこれら 3 つのメソッドの違いを以下にまとめる。

メソッド	説明
isdecimalメソッド	UnicodeのGeneral_Categoryプロパティの値がNdのものをTrueとする（UnicodeのNumeric_Typeプロパティの値がDecimalのものをTrueとする）
isdigitメソッド	UnicodeのNumeric_Typeプロパティの値がDigit／Decimalのものをtrueとする
isnumeircメソッド	UnicodeのNumeric_Typeプロパティの値がDigit／Decimal／Numericのものをtrueとする

isdecimal ／ isdigit ／ isnumeric メソッドの違い

なお、Unicode の General_Category=Nd である文字と Numeric_Type=Decimal である文字は同一といってよさそうであることから、上の isdecimal メソッドの説明にはその旨を記載してある（「UNICODE CHARACTER DATABASE」の「UCD File Format Invariants」項が始まる直前の段落には「the set of characters having General_Category=Nd will always be the same as the set of characters having NumericType=de」とある）。

つまり、これら 3 つのメソッドでは True と判定する文字種が isdecimal → isdigit → isnumeirc の順に「Numeric_Type=Decimal のみ→ Numeric_Type=Decimal ＋ Digit → Numeric_Type ＝ Decimal ＋ Digit ＋ Numeric」のように増えていくということだ。各カテゴリに含まれる具体的な文字種については以下のリンクを参照されたい。

- Numeric_Type=Decimal
- Numeric_Type=Digit
- Numeric_Type=Numeric

手短にまとめると、Numeric_Type ＝ Decimal（General_Category ＝ Nd）には上で見たような 10 進数字が含まれる。Numeric_Type ＝ Digit には丸付き数字や上付き数字、下付き数字など（の一部）が含まれる。Numeric_Type ＝ Numeric には「零」「一」「壱」「千」「億」やローマ数字など数の概念を表す文字が含まれる。

以下に例を示す。

```
s = '123'
r1 = s.isdecimal()
r2 = s.isdigit()
r3 = s.isnumeric()
print(f'{s} isdecimal: {r1}')  # 123 isdecimal: True
print(f'{s} isdigit: {r2}')  # 123 isdigit: True
print(f'{s} isnumeric: {r3}')  # 123 isnumeric: True

s = '¥u00b9¥u00b2¥u00b3'  # 上付き数字の1、2、3
r1 = s.isdecimal()
r2 = s.isdigit()
r3 = s.isnumeric()
print(f'{s} isdecimal: {r1}')  # ¹²³ isdecimal: False
print(f'{s} isdigit: {r2}')  # ¹²³ isdigit: True
print(f'{s} isnumeric: {r3}')  # ¹²³ isnumeric: True

s = '二億四千万'
r1 = s.isdecimal()
r2 = s.isdigit()
r3 = s.isnumeric()
print(f'{s} isdecimal: {r1}')  # 二億四千万 isdecimal: False
print(f'{s} isdigit: {r2}')  # 二億四千万 isdigit: False
print(f'{s} isnumeric: {r3}')  # 二億四千万 isnumeric: True
```

どんな数字を判定したいかに応じて、これらのメソッドを使い分けられるが、多くの場合は isdecimal メソッドを使えば十分だろう。正規表現を使って、文字列が数字だけで構成されているかどうかを判定することも可能だ。これについては別稿で紹介する。

［解決！Python］文字列が数値へ変換可能かどうかを判定するには（int ／ float 関数の例外、re.fullmatch 関数）

文字列の値を数値に変換する前に、それが変換できるかを調べる必要がある。例外を使ってこれを調べる方法と正規表現を使う方法の 2 つを紹介する。

かわさきしんじ，Deep Insider 編集部（2021 年 02 月 09 日）

```
# 文字列が数値を表し、int ／ float 関数による変換が可能かどうかを判定
def isint(s):  # 整数値を表しているかどうかを判定
    try:
        int(s, 10)  # 文字列を実際に int 関数で変換してみる
    except ValueError:
        return False
    else:
        return True

def isfloat(s):  # 浮動小数点数値を表しているかどうかを判定
    try:
        float(s)  # 文字列を実際に float 関数で変換してみる
    except ValueError:
        return False
    else:
        return True

s1 = '123'
s2 = '123b'
r1 = isint(s1)
r2 = isint(s2)
print(f'{s1}: {r1}, {s2}: {r2}')  # 123: True, 123b: False

s1 = '1.23e-1'
s2 = '1.23e'
r1 = isfloat(s1)
r2 = isfloat(s2)
print(f'{s1}: {r1}, {s2}: {r2}')  # 1.23e-1: True, 1.23e: False

# 正規表現を使う
import re

def isint2(s):  # 正規表現を使って判定を行う
    p = r'[-+]?¥d+'
    return True if re.fullmatch(p, s) else False

def isfloat2(s):  # 正規表現を使って判定を行う
    p = r'[-+]?(¥d+¥.?¥d*|¥.¥d+)([eE][-+]?¥d+)?'
    return True if re.fullmatch(p, s) else False
```


→目次に戻る

```
s1 = '123'
s2 = '123b'
r1 = isint2(s1)
r2 = isint2(s2)
print(f'{s1}: {r1}, {s2}: {r2}')  # 123: True, 123b: False

s1 = '1.23e-1'
s2 = '1.23e'
r1 = isfloat2(s1)
r2 = isfloat2(s2)
print(f'{s1}: {r1}, {s2}: {r2}')  # 1.23e-1: True, 1.23e: False
```

文字列が数値へ変換が可能かどうかを判定する

input 関数で入力された文字列を数値に変換するなど、文字列を数値に変換しなければならないことはよくある。ただし、その際には文字列が数値に変換可能かどうかを前もって調べる必要もある。

文字列を数値に変換可能かどうかを調べるには、実際に int 関数や float 関数にその文字列を渡してみるのが簡単だ。例外が発生すれば変換できず、そうでなければ変換できる。このことを利用して、次のように isint 関数や isfloat 関数を定義できるだろう。

```
def isint(s):  # 整数値を表しているかどうかを判定
    try:
        int(s, 10)  # 文字列を実際に int 関数で変換してみる
    except ValueError:
        return False  # 例外が発生=変換できないので False を返す
    else:
        return True  # 変換できたので True を返す

def isfloat(s):  # 浮動小数点数値を表しているかどうかを判定
    try:
        float(s)  # 文字列を実際に float 関数で変換してみる
    except ValueError:
        return False  # 例外が発生=変換できないので False を返す
    else:
        return True  # 変換できたので True を返す
```

使用例を以下に示す。

```
s1 = '123'
s2 = '123b'
r1 = isint(s1)
r2 = isint(s2)
print(f'{s1}: {r1}, {s2}: {r2}')  # 123: True, 123b: False

s1 = '1.23e-1'
s2 = '1.23e'
r1 = isfloat(s1)
r2 = isfloat(s2)
print(f'{s1}: {r1}, {s2}: {r2}')  # 1.23e-1: True, 1.23e: False
```

Python では全角の 10 進数字などでも数値への変換が可能なことは覚えておこう。以下に例を示す。ただし、符号（＋／－）や小数点、指数表記の e ／ E などは半角でなければならない。

```
s = '１２３'  # 全角の 10 進数字でも変換可能
result = isint(s)
print(result)  # True

s = '＋123' # 符号が全角だと変換できない
result = isint(s)
print(result)  # False
```

半角の数字だけであるかまでを考慮に入れたいのであれば、以下の正規表現を使う方法を使う必要がある。

正規表現を使う

正規表現を使って、次のような関数を定義してもよい。

```
import re

def isint2(s):  # 正規表現を使って判定を行う
    p = r'[-+]?¥d+'
    return True if re.fullmatch(p, s) else False

def isfloat2(s):  # 正規表現を使って判定を行う
    p = r'[-+]?(¥d+¥.?¥d*|¥.¥d+)([eE][-+]?¥d+)?'
    return True if re.fullmatch(p, s) else False
```

isint2 関数は「0 個か 1 個の符号」に続けて「1 個以上の 10 進数字」が並ぶパターンが文字列全体にマッチするかどうかを調べている。

isfloat2 関数は「0 個か 1 個の符号」の後に、「1 個以上の 10 進数字」「0 個か 1 個の小数点（.）」「0 個以上の 10 進数字」という文字の並びあるいは「小数点（.）」「1 個以上の 10 進数字」が続き、最後に「指数表記を表す e か E」「0 個が 1 個の符号」「1 個以上の 10 進数字」というパターンが、文字列全体にマッチするかどうかを調べている。

Python では「1.23」「-1.」「1.e-2」など、さまざまな形で浮動小数点数値を表せる。例えば、「-1.」は「1 個の符号の後に、1 個の 10 進数字、1 個の小数点、0 個の 10 進数字」であり上記パターンにマッチする。また、「.1e1」は「0 個の符号の後に、小数点、1 個の 10 進数字、指数表記を示す e、0 個の符号、1 個の 10 進数字」であり、こちらも上記のパターンにマッチする。

これらのパターンをなるべく網羅したつもりだが、漏れているものや実際には浮動小数点数値に変換できないものがあるかもしれないので、上記のコードをそのまま使う場合にはテストを欠かさずに行ってほしい。

以下に使用例を示す。

```
s1 = '-123'
s2 = '123b'
r1 = isint2(s1)
r2 = isint2(s2)
print(f'{s1}: {r1}, {s2}: {r2}')  # -123: True, 123b: False

s_list = ['1.23', '-1.', '1.e-2', '1.23e']
result = {s: isfloat(s) for s in s_list}  # isfloat 関数で判定
result2 = {s: isfloat2(s) for s in s_list}  # isfloat2 関数で判定
print(result)   # {'1.23': True, '-1.': True, '1.e-2': True, '1.23e': False}
print(result2)  # {'1.23': True, '-1.': True, '1.e-2': True, '1.23e': False}
```

上記コードで使用している正規表現中の文字クラス「¥d」は、Unicode で 10 進数字としての属性が与えられているもの（General_Category プロパティの値が Nd のもの）を表す。よって、正規表現を使っていないコードと同様に全角の 10 進数字なども変換可能と判定する。半角の 10 進数値のみを考慮の対象としたいのであれば、fullmatch 関数の第 3 引数で re.ASCII フラグを指定すること。

以下に例を示す。

```
def isint3(s, flags=0):
    p = r'[-+]?¥d+'
    return True if re.fullmatch(p, s, flags) else False

s = '123'
r1 = isint3(s)
r2 = isint3(s, re.ASCII)
print(f'without re.ASCII: {r1}, with re.ASCII: {r2}')
# 出力結果：
# without re.ASCII: True, with re.ASCII: False
```

ここではフラグを受け取るパラメーターを持つ isint3 関数を定義して、受け取ったフラグの値を re.fullmatch 関数の第 3 引数に渡すようにした。re.ASCII 値を与えることで動作が変わっていることを確認してほしい。

［解決！Python］文字列が英字（文字）のみで構成されているかどうかを判定するには（isalpha ／ isascii メソッド、re.match ／ re.fullmatch 関数）

文字列の isalpha メソッドを使うとそれが英字のみで構成されているかどうかを調べられる。その使い方と注意点、正規表現を使って同様な処理を行う方法を紹介する。

かわさきしんじ，Deep Insider 編集部（2021 年 02 月 02 日）

```
s = 'Deep'
result = s.isalpha()  # 英字（文字）のみで構成されているかどうかを判定
print(f'"{s}" is alpha:', result)  # "Deep" is alpha: True

s = 'Deep Insider'
result = s.isalpha()  # 空白文字は英字（文字）には含まれない
print(f'"{s}" is alpha:', result)  # "Deep Insider" is alpha: False

s = 'AI初学者のためのサイト'
result = s.isalpha()  # ひらがな／カタカナ／漢字も文字として判定される
print(f'"{s}" is alpha:', result)  # "AI初学者のためのサイト" is alpha: True

# isalphaメソッドとisasciiメソッドを組み合わせる
string_list = ['Deep Insider', 'DeepInsider', 'AI初学者のためのサイト']
for item in string_list:
    if item.isalpha():
        if item.isascii():
            print(f'"{item}" includes only alphabets')
        else:
            print(f'"{item}" includes letter characters')
    else:
        print(f'"{item}" includes non-letter characters')
# 出力結果:
# "Deep Insider" includes non-letter characters
# "DeepInsider" includes only alphabets
# "AI初学者のためのサイト" includes letter characters

# 正規表現を使用する
import re

string_list = ['Deep Insider', 'DeepInsider', 'AI初学者のためのサイト']
for item in string_list:
    if re.fullmatch('[a-zA-Z]+', item):  # re.match('^[a-zA-Z]+$', 'Deep')
        print(f'"{item}" includes only alphabets')
    else:
        print(f'"{item}" includes non-alphabetic characters')
# 出力結果:
# "Deep Insider" includes non-alphabetic characters
# "DeepInsider" includes only alphabets
# "AI初学者のためのサイト" includes non-alphabetic characters
```

文字列が英字のみで構成されているかを判定する

Python の文字列にはさまざまな文字種判定メソッドが用意されている。その中で、文字列が英字のみで構成されているかどうかを判定するには isalpha メソッドが使える。これは文字列に含まれている全ての文字が英字（文字）ならば True を返し、そうでなければ False を返す。

以下に簡単な例を示す。

```
s = 'Deep'
result = s.isalpha()  # 英字（文字）のみで構成されているかどうかを判定
print(f'"{s}" is alpha:', result)  # "Deep" is alpha: True
```

この例では英語の大文字と小文字を含んだ 'Deep' という文字列で isalpha メソッドを呼び出している。その結果はもちろん True となる。

これに対して、半角空白文字を含んだ 'Deep Insider' で isalpha メソッドを呼び出すと、以下に示すようにその結果は False となる。

```
s = 'Deep Insider'
result = s.isalpha()  # 空白文字は英字（文字）には含まれない
print(f'"{s}" is alpha:', result)  # "Deep Insider" is alpha: False
```

isalpha メソッドは文字列に含まれるユニコード文字の中で、ユニコード文字データベースの General_Category プロパティの値が「Letter」に分類されるものを英字（alphabetic character）として判定する。具体的には、このプロパティの値が「Lu（Uppercase_Letter）」「Ll（Lowercase_Letter）」「Lt（Titlecase_Letter）」「Lm（Modifier_Letter）」「Lo（Other_Letter）」となっているものが該当する。空白文字の General_Category プロパティの値は「Zs（Space_Separator）」であるため、上のような結果となる。

実は、日本語のひらがなやカタカナ、漢字の多くでは、General_Category プロパティの値は今述べた Letter に分類されるものとなっている。そのため、以下のように英字ではない、日本語の文字についても isalpha メソッドは True を返すことには注意が必要だ。

```
s = 'AI初学者のためのサイト'
result = s.isalpha()  # ひらがな／カタカナ／漢字も文字として判定される
print(f'"{s}" is alpha:', result)  # "AI初学者のためのサイト" is alpha: True
```

何らかの理由で、文字列全体が英大文字／小文字のみで構成されているかどうかを判定したい場合には、isalpha メソッドと isascii メソッドを組み合わせる方法が考えられる。

```
string_list = ['Deep Insider', 'DeepInsider', 'AI初学者のためのサイト']
for item in string_list:
    if item.isalpha():
        if item.isascii():
            print(f'"{item}" includes only alphabets')
        else:
            print(f'"{item}" includes letter characters')
    else:
        print(f'"{item}" includes non-letter characters')
# 出力結果:
# "Deep Insider" includes non-letter characters
# "DeepInsider" includes only alphabets
# "AI初学者のためのサイト" includes letter characters
```

あるいは、例えば、正規表現を扱うreモジュールのmatch関数やfullmatch関数を使う方法もある。fullmatch関数は引数に指定したパターンが、マッチ対象の文字列にマッチしたときにはマッチした部分を表すre.Matchオブジェクトを、マッチしなかったときには戻り値を返さない（Noneを返す）。このことを利用して、以下のようなコードを書けるだろう。

```
string_list = ['Deep Insider', 'DeepInsider', 'AI初学者のためのサイト']
for item in string_list:
    if re.fullmatch('[a-zA-Z]+', item):  # re.match('^[a-zA-Z]+$', 'Deep')
        print(f'"{item}" includes only alphabets')
    else:
        print(f'"{item}" includes non-alphabetic characters')
# 出力結果:
# "Deep Insider" includes non-alphabetic characters
# "DeepInsider" includes only alphabets
# "AI初学者のためのサイト" includes non-alphabetic characters
```

ここでは、正規表現として「[a-zA-Z]+」を使用しているが、これは「1文字以上の連続する英小文字と大文字」を意味する。「単語を構成する文字」を表す文字クラス「¥w」を使ってもよさそうだが、これだと数字やアンダースコアも含まれるし、isalphaメソッドと同様に日本語の文字までマッチしてしまうので、その際にはmatch関数やfullmatch関数の第3引数にre.ASCIIフラグを指定して、ASCII範囲の文字のみをマッチの対象とするなどした方がよいかもしれない。

```
print('without re.ASCII flag')
s = 'AI初学者のためのサイト'
if re.match(r'^¥w+$', item):
    print(f'"{item}" includes only alphabets')
else:
    print(f'"{item}" includes non-alphabetic characters')

print('---')

print('with re.ASCII flag')
s = 'AI初学者のためのサイト'
if re.match(r'^¥w+$', item, re.ASCII):
    print(f'"{item}" includes only alphabets')
else:
    print(f'"{item}" includes non-alphabetic characters')
# 出力結果:
# without re.ASCII flag
# "AI初学者のためのサイト" includes only alphabets
# with re.ASCII flag
# "AI初学者のためのサイト" includes non-alphabetic characters
```

[] の中にはマッチさせたい任意の文字を含めることができるので、isalpha メソッドよりも柔軟に文字種の判定を行えることは覚えておこう。

編集：@IT 編集部
発行：アイティメディア株式会社