

2018年航空宇宙情報システム学第2
第2部「プログラミングと数値計算」

第4回 Python入門 3
～文字列・リスト～

2018年5月29日

8章 文字列

文字「列」

- 文字列の n 番目の文字にアクセスする

```
>>> fruit = 'banana'
```

```
>>> fruit[1]
```

a

- 変数fruitに代入された文字列の「1番目」の文字
- [] (大かっこ)を用いた表現をインデックスと呼ぶ
- Python (や多くのプログラミング言語)では、インデックスは0から始まる
- インデックスの値は整数でなければならない
 - 例えば、fruit[1.5]はNG

len 関数とマイナスインデックス

- len関数を用いて文字列の「長さ」(文字数)を調べられる

```
>>> len(fruit)
```

- 変数fruitに代入されている文字列の最後の文字を調べるにはどうしたら良いか？

- 方法 1 (正攻法): `fruit[len(fruit)-1]`

- 長くて面倒

- 方法 2 (負のインデックス): `fruit[-1]`

- C言語的には気持ち悪いが便利

- `fruit[-2]` で末尾から2番目の文字にアクセス可

for文による文字列の走査

- トラバーサル(走査): 文字列の先頭から最後まで1文字ずつ処理

- whileを用いたトラバーサルの例

```
index = 0
while index < len(fruit):
    letter = fruit[index]
    print(letter)
    index = index + 1
```

- for文を使えばずっと簡単

```
for letter in fruit:
    print(letter)
```

#文字列の先頭から末尾まで
#1文字ずつ letter に代入して処理

for文と文字列連結の例

```
prefixes = 'JKLMNOPQ'
```

```
suffix = 'ack'
```

```
for letter in prefixes: # 1文字ずつletterに代入  
    print(letter + suffix) # 文字列の連結
```

文字列のスライス

- 文字列の「部分列」を「スライス」と呼ぶ
- 演算子 `[n:m]` でアクセスする
 - ただし、`n` と `m` はインデックス
 - `n`番目の文字から`m-1`番目の文字まで取り出される
- 例:

```
>>> s = 'Monty Python '  
>>> print(s[0:5])    # s[:5] でも良い  
Monty  
>>> print(s[6:len(s)]) # s[6:] でも良い  
Python
```

Pythonの文字列は変更不可

```
>>> greeting = 'Hello, world!'
```

- ここで、先頭の1文字だけ変更したいとする

```
>>> greeting[0] = 'J'    #エラー！
```

- Pythonでは1度作った文字列を変更できない

– やや不便にも思えるが、仕様上やむを得ない

- 新しい文字列を作り直すことで対応する

```
>>> new_greeting = 'J' + greeting[1:]
```

```
>>> print(new_greeting)
```


文字列から文字を探す

wordに代入されている**文字列**から、letterに代入されている**文字**を探し、そのインデックスを返す関数

```
def find(word, letter):  
    index = 0  
    while index < len(word):  
        if word[index] == letter:  
            return index  
        index = index + 1  
    return -1    # 見つからなかったら-1を返す
```

(余談) 実用上は、findメソッドを使う(後述)
>>> 'banana'.find('a')

文字の出現回数を数える

- (例) 'banana'の中の 'a' を数える

```
>>> word = 'banana'
>>> count = 0
>>> for letter in word:
...     if letter == 'a':
...         count = count + 1
...
>>> count
3
```

(余談) 実用上は count メソッドを使う(後述)
>>> 'banana'.count('a')

文字列メソッド

- メソッド: 関数に似ているが、呼び出し方が異なる (C++ではメンバー関数と呼ばれる)
 - オブジェクト指向プログラミングの考え方
- 文字列メソッド upper(): 大文字に変換

```
>>> word = 'banana'
>>> word.upper()
'BANANA'
```
- find()メソッド: 文字または部分文字列を検索

```
>>> word.find('na')
2
```
- count()メソッドもある

in 演算子

- in 演算子は、文字列に文字列(または文字)が含まれているかどうかを調べる2値演算子

```
>>> 'a' in 'banana'  
True
```

- 同様の処理をin演算子を使わないで実現するのは結構面倒
- 使用例: 引数で与えられた2つの文字列に共通する文字を表示する関数

```
def in_both(word1, word2):  
    for letter in word1:  
        if letter in word2:  
            print(letter)  
  
>>> in_both('apples', 'oranges')
```

文字列の比較

- **== 演算子**は、両辺の文字列が**完全に一致**するかどうかを判定

```
>>> word = 'banana'
>>> word == 'banana'
True
```

```
>>> word == 'Banana'
False # 大文字小文字は区別される
```

- **不等号記号 < および >**は**アルファベット順での前後関係**を判定

```
>>> 'pinapple' > 'banana'
True
```

– 注意: 任意の大文字は全ての小文字よりも「先」

10章 リスト

リストとは

- リスト(list)とは、「値の列(sequence)」である
 - 文字列は「文字」の列だったが、リストはあらゆる型の値の列
 - リストは、[値1, 値2, ...] の形で表す
- リストの例:
 - >>> cheeses = ['Cheddar' , 'Edam' , 'Gouda']
#文字列のリスト
 - >>> numbers = [17, 123] #整数値のリスト
 - >>> empty = [] #「空」リスト
 - >>> mixed = ['spam' , 2.0, 5, [10, 20]] #型を混ぜてもOK

リストは変更可能

- インデックスの使い方は文字列の場合と同じ

```
>>> print(cheeses[0])
```

```
Cheddar
```

- 負のインデックス も利用可。(例) `cheeses[-1]`

- 文字列と異なり、リストの要素は変更可能

```
>>> numbers = [17, 123]
```

```
>>> numbers[1] = 5          # 要素の値を変更可能
```

- 文字列と同様に、`in` 演算子で要素が含まれているか調べられる

```
>>> 'Edam' in cheeses
```

```
True
```


for 文とリストの走査

- リストの要素を一つ一つ順番に処理する

```
>>> for cheese in cheeses:  
...     print(cheese),  
Cheddar Edam Gouda
```

- 要素のインデックスが必要な場合、range関数と組み合わせる

```
>>> range(10)  
range(0, 10)  
>>> list(range(10))  
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  
>>> for i in range(len(numbers)):  
...     numbers[i] = numbers[i]*2
```

リスト演算子

- 文字列演算子と類似

- **+** 演算子(連結)

```
>>> a = [1, 2, 3]
```

```
>>> b = [4, 5, 6]
```

```
>>> c = a + b    #2つのリストを連結
```

```
>>> print c
```

```
[1, 2, 3, 4, 5, 6]
```

- ***** 演算子(繰り返し)

```
>>> [1, 2, 3] * 3    #リストの中身を繰り返し
```

```
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

ベクトル演算にはならない

リストのスライス(部分列)

- 文字列の場合とほぼ同じだが、**変更可能**

```
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
>>> t[1:3]
['b', 'c']
>>> t[:4]      # t[0:4] と同じ (0を省略できる)
['a', 'b', 'c', 'd']
>>> t[3:]      # t[3:len(t)] と同じ
['d', 'e', 'f']
>>> t[:]       # リスト全体のコピー
['a', 'b', 'c', 'd', 'e', 'f']
>>> t[1:3] = ['x', 'y']  # 要素を変更
>>> t
['a', 'x', 'y', 'd', 'e', 'f']
```

リストのメソッド(1) : append, extend

- appendメソッドは要素を1つ付け足す

```
>>> t = ['a', 'b', 'c']  
>>> t.append('d')  
>>> print(t)  
['a', 'b', 'c', 'd']
```

- extendメソッドはリストの全ての要素を加える

```
>>> t1 = ['a', 'b', 'c']  
>>> t2 = ['d', 'e']  
>>> t1.extend(t2) # t1.append(t2) だとどうなる？  
>>> print(t1)  
['a', 'b', 'c', 'd', 'e'] # t1 = t1 + t2  
でも同じ結果になる
```

リストのメソッド(2) : sort

- 数値のリストは小さい順に並び替え

```
>>> lstnum = [3, 5, 2, 10, 4, 25, 13]
```

```
>>> lstnum.sort()
```

```
>>> lstnum
```

```
[2, 3, 4, 5, 10, 13, 25]
```

- 文字のリストはabc順(大文字優先)

```
>>> lstchar = ['c', 'b', 'B', 'd', 'A', 'a', 'Z']
```

```
>>> lstchar.sort()
```

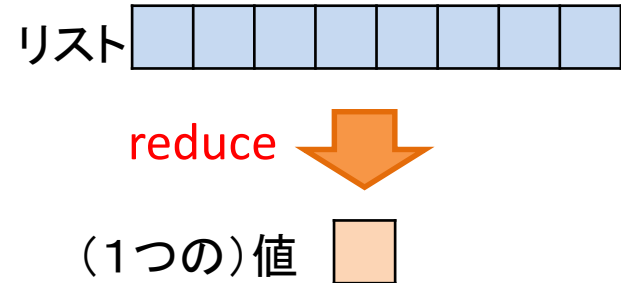
```
>>> lstchar
```

```
['A', 'B', 'Z', 'a', 'b', 'c', 'd']
```

リストの reduce 処理

- リスト t に含まれる数値の総和を求める関数

```
def add_all(t):  
    total = 0 # 初期化  
    for x in t:  
        total += x # total=total+x と同じ  
    return total
```



- 通常は、sum関数を使えば良い

```
>>> t = range(10) # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  
>>> sum(t)  
45
```

- リストを統合して1つの値を求めることを reduce処理 と呼ぶ
 - 他の例: 最大値、最小値、平均値、など
- (参考) reduce 関数の利用

```
>>> from functools import reduce # Python2では組込関数だった  
>>> reduce(lambda a, b:a+b, t) # 無名関数を使用
```

リストの map 処理

- リストの各要素に共通の関数を適用して新たなリストを得ることを map処理と呼ぶ
- 例: リストに含まれる全ての文字列の頭文字を大文字に変換する

```
def capitalize_all(t):
```

```
    res = []
```


```
    for s in t:
```

```
        res.append(s.capitalize())
```

```
    return res
```

```
>>> cities = [ 'tokyo' , ' osaka' , ' nagoya' ]
```

```
>>> capitalize_all(cities)
```

リスト 

map



加工されたリスト



- (参考) 内包表記を使った実装

```
def capitalize_all(t):
```

```
    return [s.capitalize() for s in t]
```

(参考) map関数を使う場合

```
list(map(lambda s: s.capitalize(), cities))
```

リストのmap処理：別の例

- 1から10までの整数の平方根のリストを求める
- 正直なやり方

```
>>> from math import sqrt
```

```
>>> sqroots = []
```

```
>>> for x in range(1,11):
```

```
...     sqroots.append(sqrt(x))
```

- 内包表記を利用する場合

```
>>> sqroots = [sqrt(x) for x in range(1,11)]
```


リストの filter 処理

- リストから**特定の条件を満たす要素を取り出して**新たなリストを作ることを filter 処理と呼ぶ
- 例: 整数値のリストから3で割り切れる要素を取り出す

```
>>> lst = [10, 13, 24, 25, 18]
```

```
>>> ans = []
```

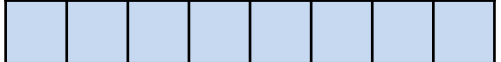
```
>>> for x in lst:
```

```
...     if x % 3 == 0:
```

```
...         ans.append(x)
```

```
>>> ans
```

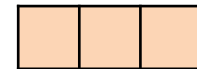
```
[24, 18]
```

リスト 

filter



抽出された
リスト



- (参考) 内包表記を使う方法

```
>>> ans = [x for x in lst if x%3 ==0]
```

- (参考) filter 関数と無名関数を使う方法

```
>>> ans = list(filter(lambda x: x%3==0, lst))
```

リスト要素の削除

- pop メソッドを使う方法（取り出して削除）

```
>>> t = ['a', 'b', 'c']  
>>> x = t.pop(1)    # x に 'b' が取り出される  
>>> print t  
['a', 'c']
```

- del 演算子を使う（インデックスで指定）

```
>>> del t[1]    # 1番目の要素が削除される  
スライスを使って一部分をまとめて削除することもできる  
>>> del t[1:]   # 1番目から最後の要素まで削除
```

- removeメソッドを使う（削除する値を指定）

```
>>> t.remove('c')
```

リストに同じ値が複数含まれている場合はどうなるか？

文字列とリスト

- list関数：文字列から「文字のリスト」を作成

```
>>> s = 'spam'
>>> t = list(s)
>>> print t
['s', 'p', 'a', 'm']
```

- フレーズから単語のリストを作成

```
>>> s = 'pining for the fjords'
>>> t = s.split() # スペースで区切ってリストを作成
>>> print t
['pining', 'for', 'the', 'fjords']
```

文字列とリスト(続き)

- 区切り文字を指定することも可能

```
>>> s = 'spam-spam-spam'
```

```
>>> s.split( '-' )
```

```
['spam', 'spam', 'spam']
```

- 文字列のリストを(区切り文字)で連結するには

```
>>> t = ['pinning', 'for', 'the', 'fjords']
```

```
>>> delimiter = ' ' # 半角スペースを区切り文字に
```

```
>>> delimiter.join(t)
```

```
'pinning for the fjords'
```

オブジェクトと値

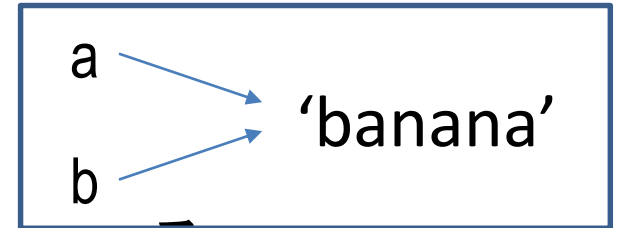
- 同じ文字列を2つの変数に代入

```
>>> a = 'banana'
```

```
>>> b = 'banana'
```

この場合は、同じオブジェクトを指している

同じオブジェクトを参照



- リストの場合はどうか？

```
>>> a = [1,2,3]
```

```
>>> b = [1,2,3]
```

```
>>> a is b
```

```
False
```

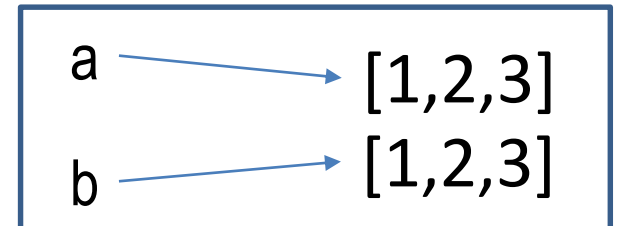
オブジェクトとしては異なる

```
>>> a == b
```

```
True
```

値としては等しい

別のオブジェクトを参照



オブジェクトの別名 (Aliasing)

- リストを他の変数に代入すると..

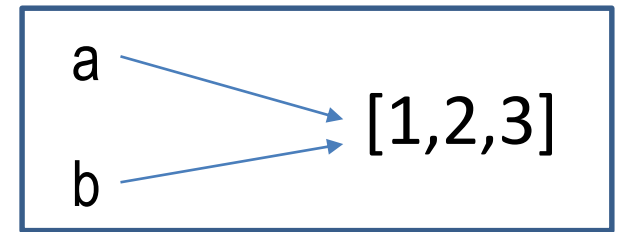
```
>>> a = [1,2,3]
```

```
>>> b = a
```

```
>>> b is a
```

True

同じオブジェクトを参照



– この場合、a と b は同じリストオブジェクトへの参照、あるいは別名となっている

- 同じオブジェクトへの別名なので、一方の要素を変更するともう一方も影響を受ける

```
>>> a[0] = -1 # 先頭を変更
```

```
>>> b
```

```
[-1, 2, 3]
```

この後、
>>> a = [1,2,3]
としたら、b はどうなるか？

(補足)リストのコピー

- リストのコピー(クローン)を作るには？

```
>>> a = [1,2,3]
```

- 方法1: スライスを使う

```
>>> b = a[:]
```

- 方法2 : list 関数を使う

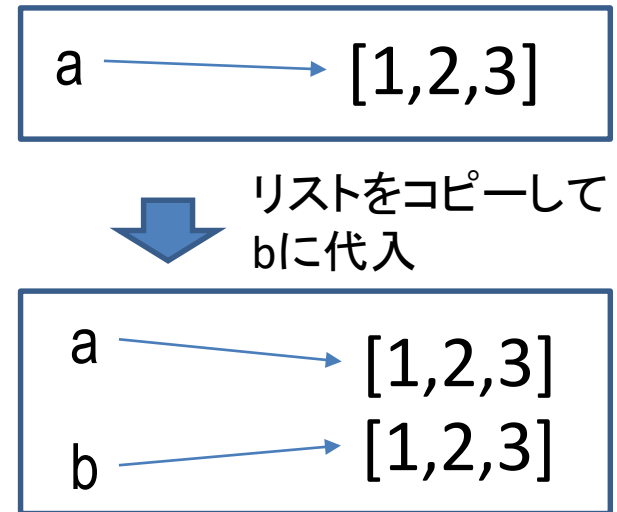
```
>>> b = list(a)
```

- 方法3 : copyメソッドを使う

```
>>> b = a.copy()
```

- ただし、リストの要素がリスト(リストが入れ子)の場合は注意が必要

– 「浅い」コピー v.s. 「深い」コピー



関数の引数にリストを渡す

- 関数の引数にリストを渡すと、**コピーではなくて参照が渡される**

```
>>> def square(lst):  
...     for i in range(len(lst)):  
...         lst[i] = lst[i]*lst[i]    # 値を2乗し  
...     て変更  
...  
>>> l = [1.0, 2.0, 3.0]  
>>> square(l)  
>>> l  
[1.0, 4.0, 9.0] # リスト l の中身が変更されている
```


今日の宿題

減衰振動の運動方程式を表す微分方程式

$$\ddot{x} + 2\gamma\dot{x} + \omega_0^2 x = 0$$

の挙動を、オイラー法による数値計算で調べよ。
ただし、パラメータの γ や ω_0^2 の値、および初期値は以下のようにする。

$$\gamma = 0.3, \quad \omega_0^2 = 1.0,$$

$$x(0) = 1.0, \quad x'(0) = 1.0$$

- プログラムファイルに homework03.py という名前を付けて、ITC-LMSから提出すること
- 締切：6月5日午前8時（次回授業の直前）

(参考)オイラー法による常微分方程式の数値解法

ステップ 1 : 微分方程式を次の形式に変形する

$$\dot{\mathbf{x}} = f(\mathbf{x}) \quad \text{ただし、} \mathbf{x} \equiv [x, \dot{x}]^T$$

ステップ 2 : 十分小さい Δt に対して、以下が成り立つ

$$\mathbf{x}(t + \Delta t) \approx \mathbf{x}(t) + \Delta t \cdot f(\mathbf{x})$$

ステップ 3 : $\mathbf{x}_0 = \mathbf{x}(0)$, $\mathbf{x}_1 = \mathbf{x}(\Delta t)$, \dots , $\mathbf{x}_n = \mathbf{x}(n\Delta t)$

と置けば、下の漸化式(差分方程式)が得られる

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \Delta t \cdot f(\mathbf{x}_n)$$

あとは、 $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \dots$ を機械的に計算していく

詳しくは、「工科のための状微分方程式」6.2節を参照