7. リスト：Lists

プログラミング・データサイエンス I

2021/6/3

1　今日の目的

今日の目的

- データ構造
  - データの塊に構造を導入
- リスト：一列にデータが並んだ構造

今日はリスト (lists) を扱います。これまで、幾つかの例を見てきました。

プログラミング言語が扱うデータには、変数に一つだけの値が入ったものばかりではありません。データの集まりに、なんらかの構造のある「データ構造 (data structure)」というものもあります。その構造を使って、内部にあるデータを効率的に扱うことができます。

例えば、1000 個の整数があるとして、その和を求めることにしましょう。ここまで見た多くのプログラムのように、1000 個全てに名前をつける、つまり `a0`、`a1` というふうに `a999` まで、1000 個の名前を付けるのはどうでしょう。これだけで 1000 行になります。

それに、`a0` とか `a1` とかいう変数名は、プログラムを書いている人にとっては、`a` の後ろに番号という意味があります。しかし、プログラミング言語から見ると、相互に関連のない独立な変数として扱われます。

`a` という名前がついたデータの塊の何番目という表現ができればよいですね。それがリストです。

本日のサンプルプログラムを取得してください。

`https://github.com/first-programming-saga/dataStructure`

2　リストの基本: Fundamentals of lists

リストの基本: Fundamentals of lists

	0	1	2	3	4	5	6	7	8
d	60	78	95	78	85	98	100	60	70

- 一次元的なデータの列

$$D = [d_0, d_1, d_2, d_3, d_4, d_5, d_6, d_7, d_8]$$

- 一つの名前を付けて、順番で管理する
- 0 番から始まることに注意する

リストとは、データを一次元的、つまり鎖のように並べて、番号を付けて管理することができるデータ構造です。例を見てください。ここで D という名前のリストには、9 個のデータが入っています。先頭が 0 番で、最後が 8 番です。この例では、リストの中には整数が入っていますが、整数に限らず様々なオブジェクトを入れることができます。

2.1　リストの基本

リストの基本

- 複数の値を一次元的にまとめたもの
  - 各要素には番号（インデクス）が付く
  - 0 番から始まる
- 要素は同じ型が基本だが
  - python では、別の型を混ぜることも可能
- 高次元のリストも使える

リストは、オブジェクトを一次元的、つまり鎖のように並べたものです。先頭が 0 番で、以降、番号で要素を指定することができます。要素の数と、終端の要素の番号が一つズレることに注意してください。python では、番号の範囲を指定して部分的なリストを取り出すことも容易です。

後で見るように、リストの終端に要素を追加できるだけでなく、指定した場所に要素を挿入することもできます。もちろん、指定した場所の要素を置き換えることも可能です。

つまり、リストの操作はかなり柔軟に行うことができます。

リストの要素は、同じ型のオブジェクトとするのが基本です。しかし、Python では、異なる型のオブジェクトを混ぜたリストを作ることができます。

リストの要素は、どのようなオブジェクトでも構いません。つまり、リストの要素にリスト入った高次元のリストを作ることができます。例えば、二次元構造をもったようなリストです。さらに、次週に説明する他のデータ構造が要素として入ることもあります。

3 リストの生成: Creating lists

リストの生成: Creating lists

- 要素を指定した生成
- 全て同じ要素が入ったリストの生成
- 他のものから生成
- 空のリスト

初めにリストの作り方を説明します。`listSamples.ipynb` を開いてください。

最初のセルと二番目のセルでは、要素をカンマで区切って列挙することでリストを定義しています (ソースコード 3.1 と 3.2)。両端の括弧 `[`と`]` がリストであることを示しています。他の括弧を使うと、次回に説明するように、別のデータ構造になります。注意してください。

使用している環境では、セルの最後に、変数名だけを書くと、その内容を印刷することができます。

ソースコード 3.1　要素を指定したリストの生成

```
1 numbers = [10, 20, 0, −10, 5, 7, −7]
2 numbers
```

ソースコード 3.2　要素を指定したリストの生成

```
1 colors = ['red', 'blue', 'green']
2 colors
```

配列を作る際に、同じ値を書きこんでおきたい場合があります。3 番目と 4 番目のセルが、そのような例です (ソースコード 3.3 と 3.4)。すぐには、使い道がわからないかもしれませんが、リストの初期値を指定する必要がある場合に使うことができます。必要になったときに、思い出しましょう。

ソースコード 3.3　同じ要素を繰り返し指定したリストの生成

```
1 zeros = [0]*5
2 zeros
```

ソースコード 3.4　同じ要素を繰り返し指定したリストの生成

```
1 xyz = ['x', 'y', 'z']*3
2 xyz
```

`list()` という関数は、何も指定しない場合には、要素がゼロ個のリストを作ります。Iterable、つまり順序のあるものを引数とすると、その要素を持つリストを作ります。

ソースコード 3.5 では、`range(0,10,2)`、つまり 0 から 10 未満の 2 つおきの数、0 から 8 までの偶数が入ります。`range()` は、順序のある数値の列を作りますから、その生成物は Iterable です。

ソースコード 3.6 では、`saga` という文字列がばらばらになって、4 つの文字からなるリストができます。文字列は、文字の位置で順序を指定できるため、Iterable です。

ソースコード 3.5　Iterable なものからリストの生成

```
1 evens = list(range(0, 10, 2))
2 evens
```

ソースコード 3.6　Iterable なものからリストの生成

```
1 chars = list('saga')
2 chars
```

3.1　リスト内包表記：List Comprehensions

リスト内包表記：List Comprehensions

- ある条件を満たすモノからリストを作る
- `for` と `if` を使って表記

リストを生成する際に、`for` を使って、ある規則に従った値を格納することができます。リスト内包表記と言います。

ソースコード 3.7 を見てください。`negatives` というリストは、ソースコード 3.1 で定義した `numbers` から負の値だけを抜き出したリストです。一方、`positives` は、`numbers` から正の値だけを抜き出したリストです。この例のように、既存のリストから短い条件文を書くことで、その一部を抜き出したリストを作ることができます。

ソースコード 3.7　リスト内包表記の例

```
1 negatives = [x for x in numbers if x < 0]
2 positives = list(x for x in numbers if x > 0)
3 print(negatives)
4 print(positives)
```

4　リストの要素の参照：Accessing elements of lists

リストの要素の参照：Accessing elements of lists

- 文字列と同様に番号で要素を指定
  - 先頭から 0、1、2、
  - 終端から-1、-2、-3

リストができたら、使ってみましょう。リストの要素には、先頭を 0 として、番号が付いています。python の面白いところは、終端からもマイナス方向に番号がついていることです。

ソースコード 4.1 では、ソースコード 3.6 で作ったリスト chars の要素を印刷しています。

ソースコード 4.2 では、ソースコード 3.1 で作ったリスト numbers の先頭の数字を書き換えています。

ソースコード 4.1　リストの要素を読む

```
1 print(chars[0])
2 print(chars[1])
3 print(chars[−1])
4 print(chars[−2])
```

ソースコード 4.2　リストの要素を書き換える

```
1 numbers[0] = 15
2 numbers
```

4.1 for を使って要素を辿る

for を使って要素を辿る

- for を使って、リストの要素を順に辿る
  - 要素を取り出す
  - インデクスを要素を指定する

リストを使う目的の最も重要なものは、リストの要素に同じ操作を行うことです。このような処理には、for ループを使います。要素を順に取り出す方法と、インデクスを使って要素を指定する方法があります。

ソースコード 4.3 を見てください。for x in numbers とすると、numbers という配列の要素を一つ一つ取り出し、それを一時的に x という変数に代入し、処理を行うことができます。各要素が、リストのどの位置にあるかが重要でない場合には、これが簡単な方法です。この例では、numbers の要素を取り出して、変数 s に加算しています。つまり、リスト numbers の要素の和が、最終的に変数 s に入ります。

ソースコード 4.3　for を使ったリスト操作

```
1 s = 0
2 for x in numbers:
3     s += x
4     print(f's に{x}を加算,␣s={s}')
5 print(s)
```

次に、ソースコード 4.4 を見てください。要素の位置に意味がある場合には、リストの長さを計り、インデクスを使ってリストの要素を利用します。リストの長さを調べるには、len() 関数を使います。3 行目で始まる for ループでは、インデクス i を 0 から len(sqrs) の一つ手前まで変化させて処理します。この例では、numbers の要素を sqrs に一旦コピーし、各要素を二乗して元の位置に書きこんでいます (5 行目)。

ソースコード 4.4　for を使ったリスト操作

```
1 sqrs = list(numbers)
2 #リストのインデクスを用いて値を参照
3 for i in range(len(sqrs)):
4     x = sqrs[i]
5     sqrs[i] = x * x
6 print(sqrs)
```

課題 1　listSamples.ipynb の最後の課題です。リストに正と負の整数が保存されているとする。その絶対値の和を計算しなさい。絶対値は標準のモジュールにある abs() を使う。

5　リストの操作：Modifying Lists

リストの操作：Modifying Lists

- 要素の追加：append()
  - +=も使える
- 要素の挿入：insert()
- 要素の取り出し：pop()
- 要素の削除：remove()、del()
- 一部分の切り出し:インデクスの範囲を指定

次は、modifyList.ipynb を開きましょう。

データの追加には、append() を使う方法と、+=を使う方法があります。指定した位置の前に挿入するには、insert() を使用します。

pop() は指定した位置の要素を取り出し、削除します。通常の [] を使った要素指定と違い、その要素を削除することに注意してください。位置を指定しないと、最後尾が対象となります。

remove() には、削除候補の要素を指定します。先頭から探し、最初に見つけた要素を削除します。del() は、指定した位置にある要素を削除します。

これらの操作の例をソースコード 5.1 に例を示します。del() の使い方だけが違うことに注意してください。

ソースコード 5.1　リスト要素の操作

```
1 data.insert(0, '-1') # 0番の前に要素を挿入
2 data.pop(0) # 先頭の要素を取り出して削除
3 data.remove('9') # 指定した要素を削除
4 del data[0] # 先頭要素を削除
5 data
```

文字列と同様に、インデクスの範囲を指定して、一部分からなるリストを切り出すこともできます。ソースコード 5.2 に示します。

ソースコード 5.2　部分リストの取り出し

```
1 data3 = data[:6] # 5番までの取り出し
2 data4 = data[3:] # 3番以降の取り出し
3 data5 = data[2:6] # 2ばんから 5番までの取り出し
```

課題 2　以下の操作でできる data6 の内容を予想するとともに、動作させることで確認しなさい。

```
1 data6 = list()
2 for k in range(0, 5):
3     data6.append(str(k))
4 for k in range(5, 10):
5     data6 += str(k)
6 data6 += ['a', 'b', 'c']
7 data6
```

5.1　リストの判定

リストの判定

- あるモノがリストに含まれているか

```
1 listA = ['red', 'green', 'blue']
2 'red' in listA
```

目指す要素がリストに含まれているかを調べるのはとても簡単です。'red' in listA は、listA に'red'が含まれていれば True に、含まれていなければ False になります。

5.2　リストとその名前

リストとその名前

- リストの変数名は、リストのデータが保存されている領域を指示している
  - –「参照」(reference) と言う
- 代入は、リストに新たな参照を付けることに注意
  - – 二つは同じもの

リストにつけた名前について説明します。少し難しい内容ですが、理解をしてください。

ソースコード 5.3 を見てください。整数を保持する変数 a を考えます。b = a は、a の値を b に代入しています。a の値を変更しても、b には影響はありません。

ソースコード 5.3　通常の変数の名前

```
1  a = 1
2  b = a
3  a += 1
4  print(a)
5  print(b)
```

しかし、リストについた名前は違います。リストについている名前は、リストを保存している場所を指しています。「参照 (reference)」と言います。従って、代入も「参照」の代入であって、リストの内容の代入ではありません。

ソースコード 5.4 を見てください。listA と listB は、内容は同じですが、独立したリストです。listC は listA と同じ参照ですから、まったく同じものに別名をつけただけです。

ソースコード 5.4　リストの名前は参照

```
1  listA = ['red', 'green', 'blue']
2  listB = ['red', 'green', 'blue']
3  #listA に別名 listC をつける
4  listC = listA
5  print(listA == listB)
6  print(listC == listB)
7  print(listA is listB)
8  print(listA is listC)
9  listC.append('alpha')
10 print(listA)
```

リストの比較の演算には、二種類あります。==は、内容が同じことを調べます。三つのリスト listA、listB、listC は同じ内容です。is は同じオブジェクトであることを調べます。listA と ListC は同じオブジェクトですが、listA と listB は異なるオブジェクトです。

listA と ListC は同じオブジェクトですから、listA を変更すると listC も変わります。

5.3 リストの複写

リストの複写

- `list.copy()` で複写
  - 内容が同じだが、別のオブジェクトを生成
  - `listD = listA.copy()`
- `list()` の引数として指定も可能
  - `listD = list(listA)`

リストの中身を複写するには、以前に出てきた空のリストを生成する `list()`、またはリストオブジェクトに付随するメソッド `copy()` を使います。

ソースコード 5.5　リストの複写

```
1 # listD と listE は listA と同じ内容をもった、別のリスト
2 listD = listA.copy()
3 listE = list(listA)
```

5.4 リスト要素の並べ替え

リスト要素の並べ替え

- 整列：`sort()`
- 逆順に整列：`reverse()`
- でたらめな順 :`random.shuffle()`

リストの要素を並べ替える必要があることがあります。大きい順、小さい順などです。一般に「整列」と言います。

ソースコード 5.6 に例を示します。リストには、`sort()` と `reverse()` というメソッドが付いています。`sort()` は小さい順に、`reverse()` は大きい順に元のリストを変更します。ここの例では文字列が対象です。文字列の場合には、辞書順になります。

ソースコード 5.6　リスト要素の並べ替え

```
1 data = ['Kim', 'Bob', 'Mary', 'Tom', 'Sam', 'Beth', 'Ann']
2 data.sort()
3 print(data)
4 data.reverse()
5 print(data)
```

```
6 import random
7 random.shuffle(data)
8 print(data)
```

元のリストを変更したくない場合には、sorted() という関数を使います。ソースコード 5.7 では、元のリスト data は変更せず、整列した結果を data2 に保存しています。

ソースコード 5.7　リスト要素の並べ替え

```
1 data2 = sorted(data)
2 print(data2)
3 print(data)
```

5.5　文字列とリスト

文字列とリスト

- 文から単語リストを作る
- 単語リストから文を作る

「テキストから単語を切り出す」、あるいは「単語リストからテキストを作る」ためにプログラムを作る必要がありません。split() は指定した文字でテキストから単語を切り出します。逆に join() は、リスト中の単語を連結します。ソースコード 5.8 では、text を空白で区切り、リストにします。また、リストをカンマで連結し text2 を生成しています。

ソースコード 5.8　文字列とリスト

```
1 text = 'may␣god␣bless␣you'
2 wordList = text.split('␣')
3 print(wordList)
4
5 joiner = ',␣'
6 text2 = joiner.join(data)
7 print(text2)
```

5.6　多次元リスト

多次元リスト

- 要素がリストであるリスト

リストの要素としてリストを使うと 2 次元のリストになります。ソースコード 5.9 では、colors[0] は [‘red’,255,0,0] というリストになります。colors[0][1:] は、[255,0,0] です。

ソースコード 5.9　多次元リスト

```
colors = [
    ['red', 255, 0, 0],
    ['green', 0, 255, 0],
    ['blue', 0, 0, 255],
    ['yellow', 255, 255, 0]
            ]
for c in colors:
    print(c)
print(colors[0])
print('red の要素')
redComponent = colors[0][1:]
print(redComponent)
```

6　次回

次回は、他のデータ構造について説明します。

- 7 章「タプル」
- 8 章「セット（集合）」
- 9 章「辞書」