8. データ構造

プログラミング・データサイエンス I

2021/6/10

1　今日の目的

今日の目的

- タプル (tuple)
  - 値の組
- 集合
  - 同じ要素は一つだけ
- 辞書
  - キーと値の組

前回は、リストという、データが鎖状に繋がったデータ構造を扱いました。重要な点は、番号でリストの要素を管理していることでした。

Python では、そのほかに、タプル、集合、辞書というデータ構造を使うことができます。使用する括弧が異なることに注意してください。

使用する例題は、前回に配布しています。

2 タプル : Tuples

タプル : Tuples

- 値の組を柔軟に作る

```
1 data = (1, 2)
2 data += (3,) #要素の追加
3 print(data)
4 print(data[0]) #要素の参照
```

- 一旦作成したタプルの要素は変更できないことに注意
- 括弧は省略できる

```
1 data = 1, 2, 3
2 print(data)
3 a, b, c = data
4 print(a, b, c)
```

タプル (tuple) は、他のプログラミング言語ではあまり見かけない特色あるデータ構造です。いくつかの値をまとめて扱いたいときに、特に、後で扱う関数において、複数の値を戻すときに重宝します。

なお、一旦作成したタプルの要素を変更することはできません。`tuples.ipynb` を見ながら進めましょう。

タプルでは、() という括弧を使いますが、この括弧を省略することも可能です。最初の例は、三つの値をまとめて、`data` というタプルに保存する例です。二番目の例は、保存したデータを三つに分けている例です。

2.1 タプルの操作

タプルの操作

- 番号で管理

タプル は、その要素を番号で扱うことが可能です。1 行目で定義した文字列のタプルに対して、2 行目 3 行目では、一つ一つプリントしています。

一方、6 行目では、`i` という変数を 0 から `colors` の要素の数マイナス 1 まで変化させ

つつ、7 行目で番号と対応する要素を印刷しています。

ソースコード 2.1　タプルの操作

```
1 colors=('green', 'red', 'blue', 'yellow', 'orange')
2 for c in colors:
3     print(c)
4
5 n = len(colors)
6 for i in range(n):
7     print(f'{i} : {colors[i]}')
```

2.2　タプルとリスト

タプルとリスト

- タプルをリストの要素にできる
- リストをタプルの要素にできる

タプルに限らず、様々なデータ構造は、相互に入れ子にして使うことができます。ソースコード 2.2 では、results というリストの各要素がタプルになっています。

7 行目の for ループを見てください。results というリストの各要素は二つの要素を持つタプルです。そこで、そのタプルの二つの要素を name と result という変数として受け取っています。8 行目では、それぞれを印刷します。

ソースコード 2.2　タプルがリストの要素になる例

```
1 results = [
2     ('Bob', 80),
3     ('Sue', 90),
4     ('Tim', 70),
5     ('Beth', 90)
6 ]
7 for (name, record) in results:
8     print(f'{name}の成績は{record}点')
```

ソースコード 2.3 では、タプルの中に、文字列とリストを要素とするタプルが入っている例です。色の文字列と、対応する RGB 値です。

ソースコード 2.3　リストがタプルの要素になる例

```
1 colors2 = (
2     ('red', [255, 0, 0]),
3     ('green', [0, 255, 0]),
```

```
4       ('blue', [0, 0, 255]),
5       ('yellow', [255, 255, 0]),
6       ('orange', [255, 165, 0])
7   )
8   for c in colors2:
9       print(f'{c[0]}={c[1]}')
```

3 集合：Sets

集合：Sets

- リストと異なり、同じ要素は一つだけに制限される

集合はリストと似ていますが、大きな違いがあります。同じ要素を複数入れることができません。使用する括弧は{}です。また、リストと異なり、要素に番号が付いていません。

ソースコード 3.1 では、3 つの色の名前が入った colorSet に、2 行目で要素を追加しています。4 行目は、要素を指定して、削除しています。5 行目の for ループですが、どのような順序を各要素が出てくるかを指定することはできない点に注意してください。7 行目の pop() は、要素を一つ取り出し、集合からその要素を削除します。要素を指定することはできません。

ソースコード 3.1　集合の例

```
1   colorSet = {'red', 'green', 'blue'}
2   colorSet.add('yellow')
3   print(colorSet)
4   colorSet.remove('blue')
5   for c in colorSet:
6       print(c)
7   c = colorSet.pop()
8   print(c)
9   print(colorSet)
```

3.1 集合の操作：Operation of sets

集合の操作：Operation of sets

- `set()`：空の集合を作る
- `add(`**要素**`)`：要素を追加
- `remove(`**要素**`)`：要素を削除
- `pop()`：要素を削除
  - 削除する要素を指定できない
- `clear()`：全ての要素を削除

集合の要素に対する基本的な操作をまとめておきます。ソースコード 3.2 を見てください。

ソースコード 3.2　集合の操作

```
1  setB=set()#空の集合を生成
2  print(setB)
3  setB.add('orange')
4  setB.add('yellow')
5  setB.add('blue')
6  print(setB)
7  setB.remove('blue')
8  print(setB)
9  c = setB.pop()
10 print(c)
11 print(setB)
```

3.2 変更できない集合

変更できない集合

- `frozenset()` で生成すると変更できない

要素を変更できない集合を作成することも可能です。ソースコード 3.3 では、リストから変更できない集合を作っています。3 行目で無理に変更しようとすると、エラーが発生します。予めエラーとなることが分かっているので、例外処理のコードを入れています。

ソースコード 3.3　変更できない集合

```
1  setC = frozenset(['apple','orange'])#変更できない集合
```

```
try:
    setC.add('banana')#要素を追加しようとすると、エラーとなる
except Exception as e:
    print(e)
```

3.3 集合の演算

集合の演算

- 集合の和：|または union() メソッド
- 集合の共通部分：&または intersection() メソッド
- 集合の差：-

Python の集合には、もう一つ重要な操作があります。数学で出てくる集合の演算に対応する操作です。

二つの集合の和は、いずれかの集合の要素であるものの集合を作ります。

$$A \cup B = \{x \mid x \in A \vee x \in B\} \tag{3.1}$$

set1 と set2 では、red が共通ですから、和集合は 5 色を要素として持ちます。和集合の演算は union() か|という記号を使います。

二つの集合の共通部分は、両方の集合の要素であるものの集合を作ります。

$$A \cap B = \{x \mid x \in A \wedge x \in B\} \tag{3.2}$$

set1 と set2 では、red が共通ですから、共通部分は red のみを要素として持ちます。共通部分の演算は、intersection() か&という記号を使います。

集合の差は馴染みがないと思います。

$$A \setminus B = \{x \mid x \in A \wedge x \notin B\} \tag{3.3}$$

A の要素であって、かつ B の要素でないものの集合です。Python では-を使って求めます。

ソースコード 3.4　集合の演算

```
set1 = {'red', 'green', 'blue'}
set2 = {'red', 'yellow', 'orange'}
set3 = set1.union(set2)#和集合
print(set3)
set3 = set1 | set2
```

```
6  print(set3)
7  set4 = set1.intersection(set2)#共通部分
8  print(set4)
9  set4 = set1 & set2
10 print(set4)
11 set5 = set4 | {'red'}#同じオブジェクトは一度しか入らない
12 print(set5)
13 set6 = set1 - set2
14 print(set6)
```

4 辞書 : Dictionaries

辞書 : Dictionaries

- 鍵 (key) と値 (value) の対応関係を保持
- 他のプログラミング言語と呼び名が違う

python の辞書は、名前の付け方としてわかりにくいものです。キー (鍵) になるものと、対応する値 (value) を組を保持します。他のプログラミング言語では、連想配列や写像と呼ぶことがあります。使用する括弧は{}ですが、キーと値の間にコロン:を打ちます。リストで要素に番号がキーとしてついていたものが、文字列などをキーとして自由につけられると考えても良いでしょう。

ソースコード 4.1 では、airports という辞書は、アルファベット 3 文字からなる空港のコードをキーに、対応する空港名を値として保持しています (1 行目)。6 行目と 7 行目では、羽田空港と新千歳空港を新たに登録しています。

9 行目の for ループでは、キーを一つ一つ取り出し、10 行目で空港名とコードを印刷しています。

ソースコード 4.1　辞書型の例

```
1  airports = {'HSG' : '佐賀有明空港',
2              'FUK' : '福岡空港',
3              'CTS' : '新千歳空港'
4              }
5
6  airports['HND'] = '羽田空港' #新しい組の追加
7  airports['HSG'] = '九州佐賀国際空港' #キー HSG に対応した値の変更
8
9  for key in airports:
10     print(f'{airports[key]}のコードは{key}')
```

辞書からは、キーの一覧と値の一覧をそれぞれ取り出すことができます。ソースコード 4.2 では、airports という辞書から、そのキーと値を取り出し提案す。

ソースコード 4.2　辞書型の例

```
1 print(airports.keys())
2 print(airports.values())
3 for airportname in airports.values():
4     print(airportname)
```

4.1　辞書の活用例

辞書の活用例

- リストの中にキーワードが出てくる回数

辞書の利用例を見ましょう。ソースコード 4.3 を見てください。リスト flights は、利用した空港のコードのリストです。利用した空港の回数、つまり flights の中の空港コードの出現回数を数えましょう。辞書型オブジェクト flightFreq は、空港コードと、そのコードの出現回数を対応付けています。

for ループで、flights から一つ一つ空港コードを取り出し、そのコードが flightFreq にあれば、出現回数を一つ増やします。一方、flightFreq になければ、新しいエントリーを作っています。

ソースコード 4.3　利用した空港の回数を数える

```
1  #利用した空港のリスト
2  flights=['HSG', 'HND', 'FUK', 'CTS', 'HND', 'HSG', 'FUK', 'NGO', 'FUK',
3           'FUK', 'CTS', 'FUK', 'HSG', 'HND', 'FUK', 'KMQ', 'FUK', 'CTS']
4  flightsFreq={} # 利用した回数の dictionary
5  for f in flights: #flights の各要素
6      if f in flightsFreq.keys(): #空港名が既に登録されている場合
7          flightsFreq[f] += 1
8      else:
9          flightsFreq[f] = 1
10 print(flightsFreq)
```

5　次回

プログラムを効率的に書くには、一度書いたプログラムを再利用できることが重要です。自分用の処理を関数として定義します。次回は、教科書 10 章「ユーザ定義関数」の

内容です。