1 はじめに

シーケンス型である `リスト` / `タプル` , マッピング型である `辞書` について見ていきます.
また, rangeについてもみていきます.

2 目次

3 シーケンス型

概要

シーケンス型というのは, 複数の要素をまとめてある型のこと全般を指します.
他の言語をやっている人は `配列` といった方がピンとくるかもしれません.
また, これらの各要素は `順番に取り出せる` ということがとても重要です.

C言語しかやってない人はこのシーケンス型をfor文で利用した時にたまげます.
(少なくとも自分はたまげました.)

リスト (list)

`リスト` では可変長の配列を作ることができます.
データ群を `,` で区切り, それらすべてを `[]` でくくるだけです.

```
a = 10
list1 = ["hoge", "fuga", 3, 1+2.0j, a]
list2 = ["a", [2, 3], "b", list1]

print(a)
print(list1)
print(list2)
```

```
10
['hoge', 'fuga', 3, (1+2j), 10]
['a', [2, 3], 'b', ['hoge', 'fuga', 3, (1+2j), 10]]
```

どんなデータであっても入れられ, 型の違いを気にせずなんでも入れられます.
また, 入れ子構造にして, リストの中にリストを入れて, その中にリストを…
という感じに無限にすることも可能です.

リストへ変数を入れる時の挙動

> 以下の内容は読み飛ばしてもOK

また変数をリストへ放り込んだときの挙動は一応変数のときと同じく, その参照先をコピーしています.

リストの最後尾に変数 `a` を入れ, 変数 `a` の内容を変更した時について見ていきます.

```
a = 3
list1 = [1, 2, a]

print("a         ->", id(a), ":", a)
print("list1     ->", id(list1), ":", list1)
print("list1[-1] ->", id(list1[-1]), ":", list1[-1])

a = 10; print("--- aの変更 ---")
```

```
print("a         ->", id(a), ":", a)
print("list1     ->", id(list1), ":", list1)
print("list1[-1] ->", id(list1[-1]), ":", list1[-1])
```

```
a         -> 10910464 : 3
list1     -> 140419809304648 : [1, 2, 3]
list1[-1] -> 10910464 : 3
--- aの変更 ---
a         -> 10910688 : 10
list1     -> 140419809304648 : [1, 2, 3]
list1[-1] -> 10910464 : 3
```

リストの最後尾に変数 `a` を入れ, リストの最後尾の内容を変更した時について見ていきます.

```
a = 3
list1 = [1, 2, a]

print("a         ->", id(a), ":", a)
print("list1     ->", id(list1), ":", list1)
print("list1[-1] ->", id(list1[-1]), ":", list1[-1])

list1[-1] = 0; print("--- list1[-1]の変更 ---")

print("a         ->", id(a), ":", a)
print("list1     ->", id(list1), ":", list1)
print("list1[-1] ->", id(list1[-1]), ":", list1[-1])
```

```
a         -> 10910464 : 3
list1     -> 139724117942344 : [1, 2, 3]
list1[-1] -> 10910464 : 3
```

```
--- list1[-1]の変更 ---
a          -> 10910464 : 3
list1      -> 139724117942344 : [1, 2, 0]
list1[-1] -> 10910368 : 0
```

以上のように, 変数をリストの中に入れると参照先がコピーされますが, どちらかの内容を変更した途端に, それの参照する先が変更となるので, お互いの変更はお互いに干渉しないということになります.

リストにリストを入れた時の挙動

> この内容も読み飛ばしてOK

少し難しいのはリストを他者へコピーした場合です.
こちらも同様に参照先をコピーしているのですが, 各要素の参照先は変わっていてもリストそのものの参照先は変わらないため, コピー元の要素を変更するとコピー先の要素も変化してしまいます.

```
list1 = [1, 2, 3, 4]
list2 = [-1, 0, list1]

print("list1 ->", id(list1), ":", list1)
print("list2 ->", id(list2), ":", list2)
print("list2[-1] ->", id(list2[-1]), ":", list2[-1])

list1[0] = 999; print("list1の内容変更")

print("list1 ->", id(list1), ":", list1)
print("list2 ->", id(list2), ":", list2)
print("list2[-1] ->", id(list2[-1]), ":", list2[-1])
```

```
list1 -> 140582490984520 : [1, 2, 3, 4]
list2 -> 140582490984584 : [-1, 0, [1, 2, 3, 4]]
list2[-1] -> 140582490984520 : [1, 2, 3, 4]
list1の内容変更
```

```
list1 -> 140582490984520 : [999, 2, 3, 4]
list2 -> 140582490984584 : [-1, 0, [999, 2, 3, 4]]
list2[-1] -> 140582490984520 : [999, 2, 3, 4]
```

タプル (tuple)

`タプル` はリストと似たような使い方ができます.
データ群を `,` で区切り, それらすべてを `()` でくくるだけです.

```
tuple1 = (1, 2, "hoge", 1+3j)
tuple2 = ("fuga", tuple1, [3, 4])

print(tuple1)
print(tuple2)
```

```
(1, 2, 'hoge', (1+3j))
('fuga', (1, 2, 'hoge', (1+3j)), [3, 4])
```

リストとの決定的な違いは内容の変更の可否です.
誤解を恐れず, 簡単に説明するなら, タプルでは要素の変更/追加/削除ができません.

range

関数 `range` を利用して生み出される型です.
関数 `range` を使うタイミングになったらまた詳しく説明します.

```
a = range(0, 20, 2)
print(a.__class__.__name__, a)
print(list(a))
```

```
range range(0, 20, 2)
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

文字列

文字列もシーケンス型として扱われます.

```
str = "hogefuga"
print(str[2:6])
```

```
gefu
```

シーケンス型の一部を取り出す (スライス)

先ほどまでいきなり利用してきた `[]` というものは, シーケンス型の要素にアクセスする時に使用するものです.
この機能を `スライス` といいます.
各要素は先頭から順に「0番目, 1番目, 2番目, …」という風に考えてアクセスします.
「要素単体を取り出す操作」/「部分を選択して取る操作」があります

要素単体を取り出す

シーケンス型の後ろに `[]` をつけて, 取り出したい要素の数字を入れます.
もちろん, `要素数-1` を超えるような値を入れるとエラーがでるので注意してください.

```
list1 = [10, 20, "hoge", "fuga"]
print(list1[0], list1[3])
```

```
10 fuga
```

また, 一番後ろから要素を選択するには, `[]` 内に負の数を入れます.
こちらは0から始まらないので注意してください. (0からだと正順の方と被るため)

```
list1 = [10, 20, "hoge", "fuga"]
print(list1[-1], list1[-3])
```

```
fuga 20
```

`len()`という関数を使うと, オブジェクトの要素数を得られます.

```
list1 = [10, 20 , 30, 40]
print( len(list1) )
```

```
4
```

それを利用して, 末尾からの要素選択は以下のようにもかけます.

```
list1 = [10, 20 , 30, 40]
print(list1[ len(list1) - 1])
print(list1[ len(list1) - 2])
```

```
40
30
```

一部分を切り出す

`[start:end:step]`という感じに取り出します.
シーケンス型の`start`から`end-1`までが, `step`おきに取り出されます.
各インデックスは整数でなければなりません.

名称	内容	デフォルト値
start	取り出す最初のindex	0
end	とりだす末端のindex(end自身は除く)	len(array)
step	取り出す向き/ステップ数	1

```
list1 = [10, 20, 30, 40]
print(list1[0:3])
print(list1[0:3:2])
```

```
print(list1[3:0:-1])
print(list1[3:3])

# 一部を省略して書く
print(list1[:])
print(list1[1:])
print(list1[:3])
```

```
[10, 20, 30]
[10, 30]
[40, 30, 20]
[]
[10, 20, 30, 40]
[20, 30, 40]
[10, 20, 30]
```

ちなみにこの場合, 指定したインデックスが要素数を超えても問題ありません.

```
list1 = [10, 20, 30, 40]
print(list1[3:50])
print(list1[30:50])
```

```
[40]
[]
```

シーケンスのネスト

入れ子構造にあるものを `ネスト` というような言い方をします.
シーケンスをネストした時, 値の参照方法は `[][]` のように連鎖して書いていきます.

```
a = [ [1, 2, 3], [4, 5, 6], [7, 8, 9] ]
b = [ [[1, 2], [3, 4]], [[5, 6], [8.9]] ]

print("   a[0]:", a[0])
print("a[0][0]:", a[0][0])
print("a[1][2]:", a[1][2])
print()
print("      b[0]:", b[0])
print("   b[0][0]:", b[0][0])
print("b[0][0][0]:", b[0][0][0])
```

```
   a[0]: [1, 2, 3]
a[0][0]: 1
a[1][2]: 6

      b[0]: [[1, 2], [3, 4]]
   b[0][0]: [1, 2]
b[0][0][0]: 1
```

4 マッピング型

マッピング型はハッシュ可能なオブジェクトをキーとして, 任意のオブジェクトを特定できるもの.
いまのところマッピング型は `dict` しかないので, マッピング型= `dict` という認識で良いと思います.

辞書(dict)の作り方/呼び出し方

`{key1:value1, key2:value2 …}` という感じで作る.

`key` を入れることで `value` が呼び出されるようになっており, `key` はハッシュ可能でなければならないので, 他の `key` と重複してはいけない. (`value` は重複可)

```
dictionary = {10: 30, "hoge":4.4, 3.4:"fuga"}


print(dictionary[10])
print(dictionary["hoge"])
print(dictionary[3.4])
```

```
30
4.4
fuga
```

辞書の順序

Python3.7以降では, 辞書の要素の順番を保持するようになっている(らしい)
(以前までは順番を保持しておらず, 順序が実行のたびに変わっていた.)