# プログラミング実習I クラス5 (井村担当)

知能・機械工学課程 井村 誠孝

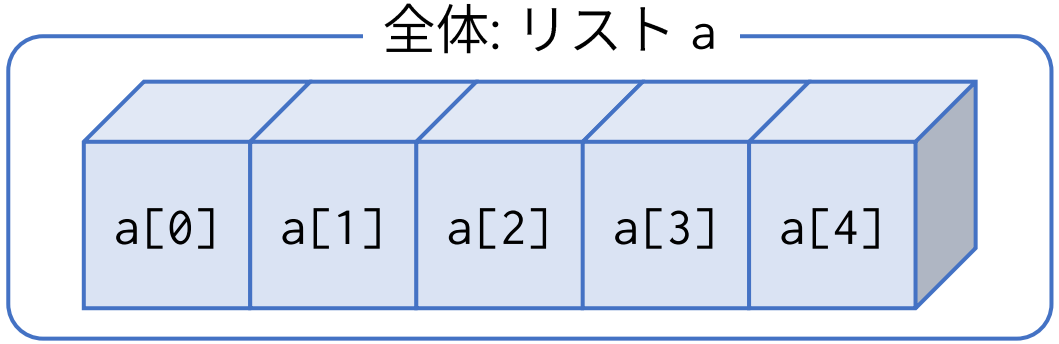
m.imura@kwansei.ac.jp

# データの入れ物

- 複数のデータを一括して管理するために，データの入れ物(データ構造)があると便利である．
- Pythonでは，基本的なデータ構造としてリスト型が多用される．
- 4章を2回に分け，まずリスト型について学習する．

# リスト型

- リストとは
  - 複数のデータに，0からはじまる番号を付けて，まとめて管理するデータ構造




2.2節 pp. 49-50で少し紹介されていました．

- リストの特徴
  - 全体に一つの名前が付いている．
  - 番号で要素を区別する．
  - リスト内の要素のデータ型は全て同じでも，異なっていてもよい．

# リスト型データの作成

- 角括弧([ ])の中にカンマ(,)で区切って要素を並べる.

```
list_int = [0, 1, 2, 3]
```

- 複数の型が混在したリスト

```
list_mix = [2, 1.732, 'test']
```

- 要素が1個のリスト

```
list_one = [100]
```

- 要素の無いリスト(空リスト)

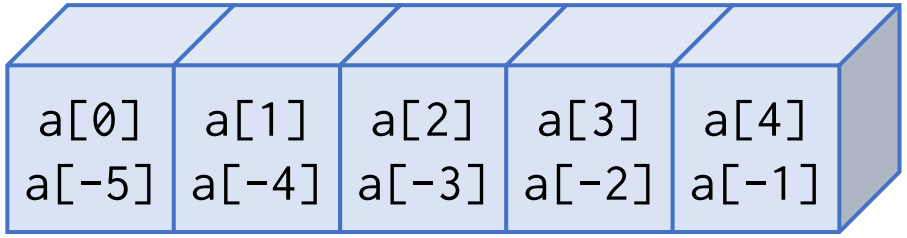
```
list_empty = []
```

空のリストを作ってどうする?
→後からデータを追加して使う

# リストの要素へのアクセス

- リスト名の後ろに角括弧[ ]を付け，その中に番号を指定する．
- 番号のことを添字(そえじ)と呼ぶ．
  - リストの長さ(項目数)がNの場合，添字は0からN-1まで．
  - マイナスの添字を使うと，後ろからアクセスできる．
    - a[-1]がリストの最後の要素
      - 前からは0，後ろからは-1からはじまるのが非対称的に思えるが，末尾を-0とすると，先頭の要素を表す0と区別できなくなる．

N以上の添字を指定するとどうなるかやってみよう




N=5の場合

  - 添字に変数を使うこともちろんできる．
    - 例: `a[i]`, `a[i * 2 + 1]`

# リストの長さ(要素数)

- リストの長さ(要素数)→組み込み関数 len()

```
list_int = [0, 1, 2, 3]
print(len(list_int))
```

4が出力される

```
list_mix = [2, 1.732, 'test']
print(len(list_mix))
```

3が出力される

組み込み関数 len() はリスト型や文字列型などの
要素が並んでいる様々なデータ型を引数に取ることができる.

# 要素の変更と追加

- 添字を指定することで要素を個別に変更できる．

```
list_int = [5, 1, 2, 3]
list_int[0] = -1
list_int[-1] = 10
```

```
[-1, 1, 2, 3]
[-1, 1, 2, 10]
```

- リストの末尾に要素を追加する→リスト型のメソッドappend()

```
list_int.append(4)
```

```
[-1, 1, 2, 10, 4]
```

引数を2つ取るメソッド
引数の順序が重要，順番で意味が違う

- リストの途中に要素を追加する→リスト型のメソッドinsert()

```
list_int.insert(1, 5)
```

位置　データ

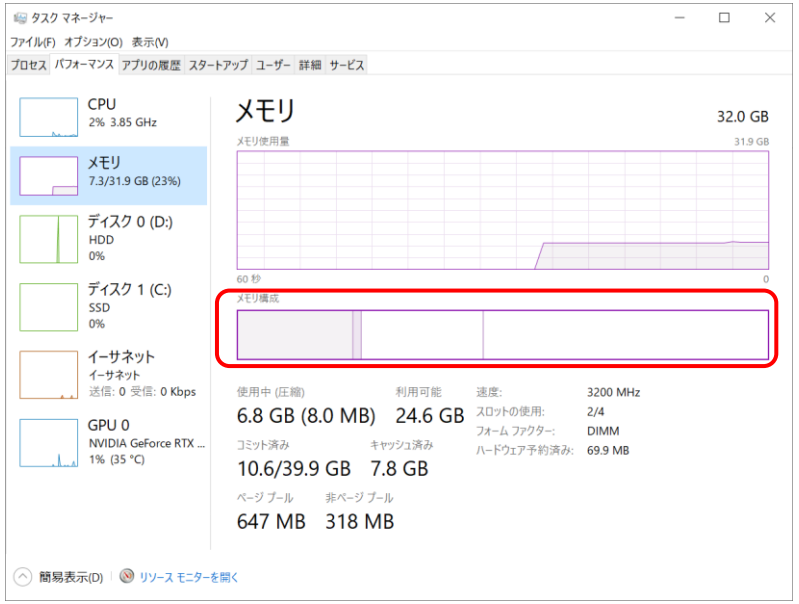
```
[-1, 5, 1, 2, 10, 4]
```

# 同じ追加でも速度が違う

- 末尾に追加するappend()は高速
- 任意の位置に挿入するinsert()は低速

どれくらい違うかというと，リストの長さや挿入位置等の条件によるが，何万倍も違う場合がある．

- データをコンピュータのメモリ上に保持している方法に理由がある．




メモリは一列にずらっと並んでいて，各記憶領域に番地が付いている

append()は後ろに足すだけ

insert()は挿入位置から後ろの要素の移動が発生

# 要素の削除

- 添字で要素を指定して削除→リスト型のメソッドpop()

```
list_int = [0, 5, 1, 2, 3, 4]
print(list_int.pop(1))
```

[0, 1, 2, 3, 4]

5が出力される

popというのはそもそも取り出すという意味
(例: ポップアップ)

- データを指定して削除→リスト型のメソッドremove()

```
list_mix = [2, 1.732, 'test']
list_mix.remove('test')
```

[2, 1.732]

  - 同じデータが複数ある場合は，添字の最も小さい要素が削除される．

```
list_test = [1, 2, 1, 3, 1]
list_test.remove(1)
```

[2, 1, 3, 1]

# リストの連結と拡張

- リストを連結して新しいリストを作成→加算演算子 +

```
list_int = [0, 1, 2, 3, 4]
list_mix = [2, 1.732]
list_cat = list_int + list_mix
```

[2, 1.732]

[0, 1, 2, 3, 4]

[0, 1, 2, 3, 4, 2, 1.732]

それぞれのリストは
元のまま変化しない

リスト型に他の演算子は使えるのか?
→ リスト * 整数 でリストが繰り返される

- メソッド extend()
  - 引数として与えられたリストの中身が追加される

```
list_int.extend(list_mix)
```

[2, 1.732]

[0, 1, 2, 3, 4, 2, 1.732]

# スライスによる柔軟な範囲指定

- スライス: 添字として [開始位置:終了位置] という表記を使用すると，リストの一部を指定できる．

```
list_f = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
print(list_f[0:3])
print(list_f[2:5])
```

添字[0]の要素から，添字[3]の要素の前まで → [0, 1, 1]

添字[2]の要素から，添字[5]の要素の前まで → [1, 2, 3]

切り出したリストの長さは
インデックスの差に等しい

- 開始位置，終了位置を省略すると，それぞれ先頭から，末尾まで，の意味になる．

```
list_f = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
print(list_f[9:])
print(list_f[:4])
```

添字[9]の要素から，末尾まで → [34, 55]

先頭から，添字[4]の要素の前まで → [0, 1, 1, 2]

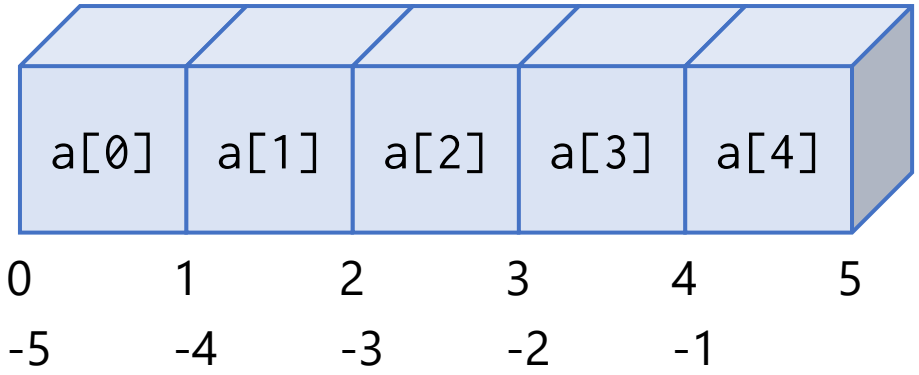
# スライスによる柔軟な範囲指定

- 負の位置も使用可能

```
list_f = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
print(list_f[3:-3])
print(list_f[-3:])
```

```
[2, 3, 5, 8, 13]
[21, 34, 55]
```

- スライスで位置を指定するインデックスは，要素と要素の間を指していて，そこでリストを切断すると考えるとわかりやすい(かも).



# 教科書に書いてないスライスの技

- 代入も可能

```
list_f = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
list_f[1:3] = [10, 20]
print(list_f)
```

```
[0, 10, 20, 2, 3, 5, 8, 13, 21, 34, 55]
```

- 間隔の指定も可能

```
list_f = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
print(list_f[1:10:2])
```

```
[1, 2, 5, 13, 34]
```

書式:
開始位置:終了位置:間隔

# リストの並べ替え

- リストを何かの基準で並べ替える(ソートする)→リスト型のメソッドsort()
- リストの並びを逆にする→リスト型のメソッドreverse()

```
list_test = [4, 9, 3, -1, 0]
list_test.sort()
list_test.reverse()
```

```
[-1, 0, 3, 4, 9]
```
昇順

```
[9, 4, 3, 0, -1]
```
降順

- sort()に引数reverse=Trueを与えることで，逆順のソートが可能
- メソッドreverse()は単に逆にするだけでソートはしないので注意

```
list_test = [4, 9, 3, -1, 0]
list_test.reverse()
```

```
[0, -1, 3, 9, 4]
```
降順にはならない

# ソートの基準

- ソートの基準は，比較演算子 < の定義に従う．
  - 要素が数値の場合: 標準では昇順(小さい方から大きい方へ)
    - 整数(int)と小数(float)が混在していても問題ない．
  - 要素が文字列の場合: 組み込み関数ord()の返値に基く昇順
    - ord()が返す値は，Unicodeのコードポイント
      - コンピュータはある文字をコード(数値)に対応させて扱う．
      - Unicodeは，現在標準的に使用されている文字コード体系．
    - アルファベットは大文字→小文字になるようにコードが割り振られている．
  - 比較演算子が適用できない場合は，エラーになる．
    - 例: リスト内に数値と文字列が混合していると，比較演算子 < による比較が行えないため，以下のようなエラーが発生する．

| A | Ω | 語 | 😂 |
|---|---|---|---|
| 000041 | 0003A9 | 008A9E | 01F60A |

```
TypeError: '<' not supported between instances of 'str' and 'int'
```

# 対象を直接変更するか，新しい対象を生成するか

- リスト型のメソッドsort(): 対象の要素を直接ソートする．

```
list_test = [4, 9, 3, -1, 0]
list_test.sort()
```

```
[-1, 0, 3, 4, 9]
```

- 組み込み関数sorted(): 引数としてリストを与えると，要素をソートした新たなリストを返す．

```
list_test = [4, 9, 3, -1, 0]
list_new = sorted(list_test)
```

```
[-1, 0, 3, 4, 9]
```

```
[4, 9, 3, -1, 0]
```

わからないことは，インタラクティブシェルで試して確認する

メソッドや関数の結果がどのように現れるかを確認して使う