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

確認 ファイルシステム

- フォルダの階層構造たどる
- テキストファイルの内容を確認
- csvファイルの内容を確認

ファイル操作

ファイルとは

- ファイルは日常的に使用していると思いますが…

wordファイル(拡張子 .docx)とか
PDFファイル(拡張子 .pdf)とか

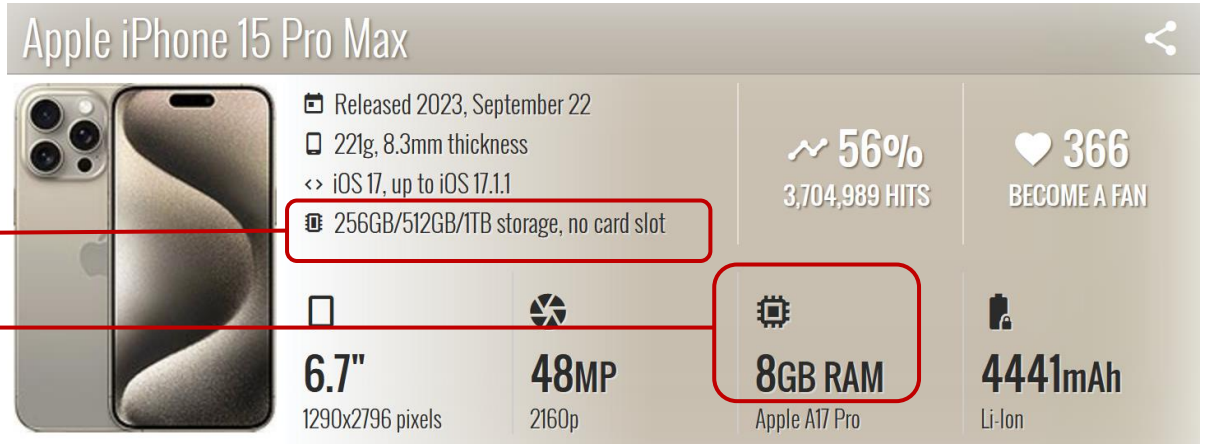
- ファイルとは何か?
  - コンピュータでデータを(主に記録媒体に)記録する際の基本的なまとまり
    - 記録媒体=ハードディスク，USBメモリ，etc.
    - ファイル名で区別される．拡張子はファイルの種類を表す．

メモリとファイルの違い

- PCがデータを保持する媒体として，主にメモリとファイルがある．
- 実行するプログラムや計算対象のデータは，メモリ上に置かれる必要がある．
- プログラムで処理した結果を保存したり，プログラムに大きなデータを与えるためには，ファイルを使用する必要がある．
  - 最近ではクラウド上のストレージもあるが，場所がネットワークの向こうにあるだけで実体はファイルと変わりない．

スマートフォンの場合は?

こちらがファイル保存用
(電源Offでも消えない)
こちらがメモリ



余談: ファイルの形式(内容)とファイル名は独立

- 本当にあった話
  - 「レポートはwordで記述し，PDFに変換して提出してください」と指示
  - wordで書く
    →保存(.docxファイルができる)
    →名前(拡張子)を.pdfに変える
    →提出

⇒ってしても，ファイルの形式(内容の構造)はwordファイルのままで，PDFに変換はされたりはしません．

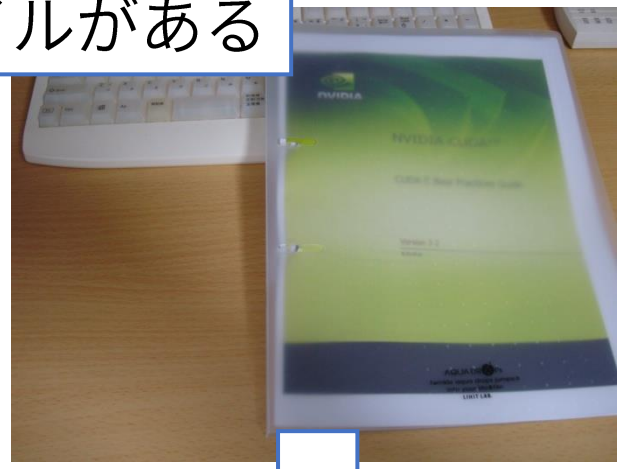
ファイル入出力

- これまでのプログラム
  - 入力: キーボードから打ち込む or プログラムに埋め込む
  - 出力: 画面に表示
- 実用的なプログラム
  - 入力: ファイルから読み込む
  - 出力: ファイルに書き込む

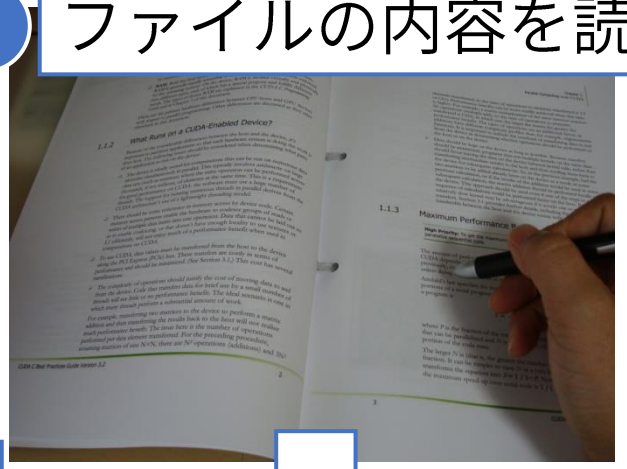
Pythonに用意されている機能を使って
ファイルの読み書きを行う方法を学ぶ

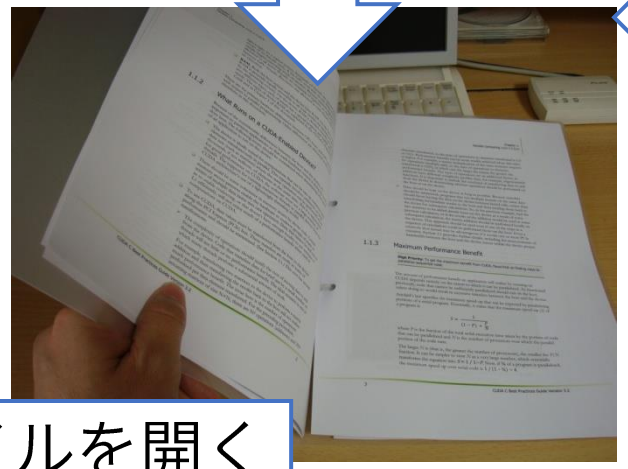
ファイル入出力の手順

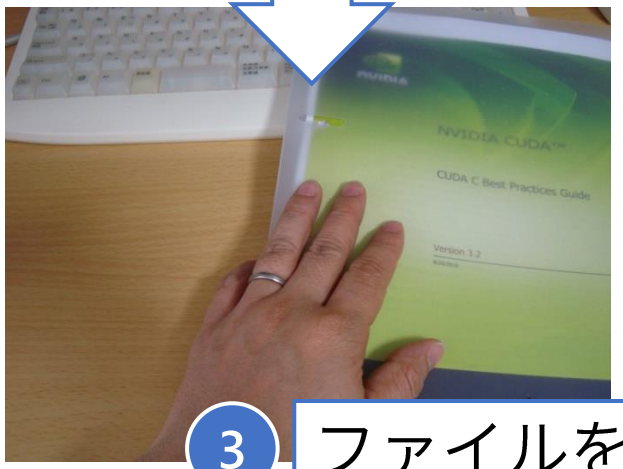
ファイルがある



2 ファイルの内容を読み書き







1 ファイルを開く

3 ファイルを閉じる

Pythonのファイル操作関数

- ファイルを開く: 組み込み関数 open() 1
  - ファイルを開く際に，読み込む(read)ために開くのか，書き込む(write)ために開くのか指定する
  - ファイルを開くと，ファイルオブジェクトが返される．
    - ファイルオブジェクトの実際のデータ型は，読み書きモードなどに応じて変化する．
- ファイルの内容を読み書き 2
  - 読み込み/書き込みとも，いくつかのメソッドが用意されているので，適宜使い分ける．
    - 読み出し `read()`, `readline()`, `readlines()` など
    - 書き込み `write()`, `writelines()` など
- ファイルを閉じる: メソッド `close()` 3

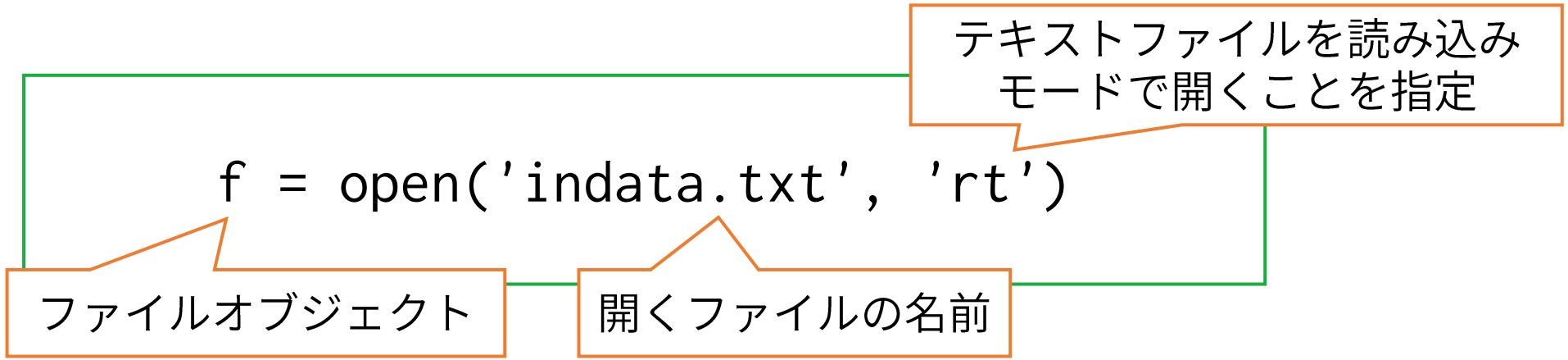
ファイルの種類

- 本実習で扱うのはテキストファイル
  - テキストエディタ(サクラエディタなど)で開いて中身を読むことができるファイル
  - 内部は，文字コードの連なり．
- テキストファイルでないファイル = バイナリファイル
  - 画像・動画・音楽
  - wordファイル・excelファイル・PDFファイル
  - 実行ファイル (Windowsならば.exe)

ファイルを開く・閉じる

ファイルを開く 1

- ファイルを開くには組み込み関数open()を用いる．



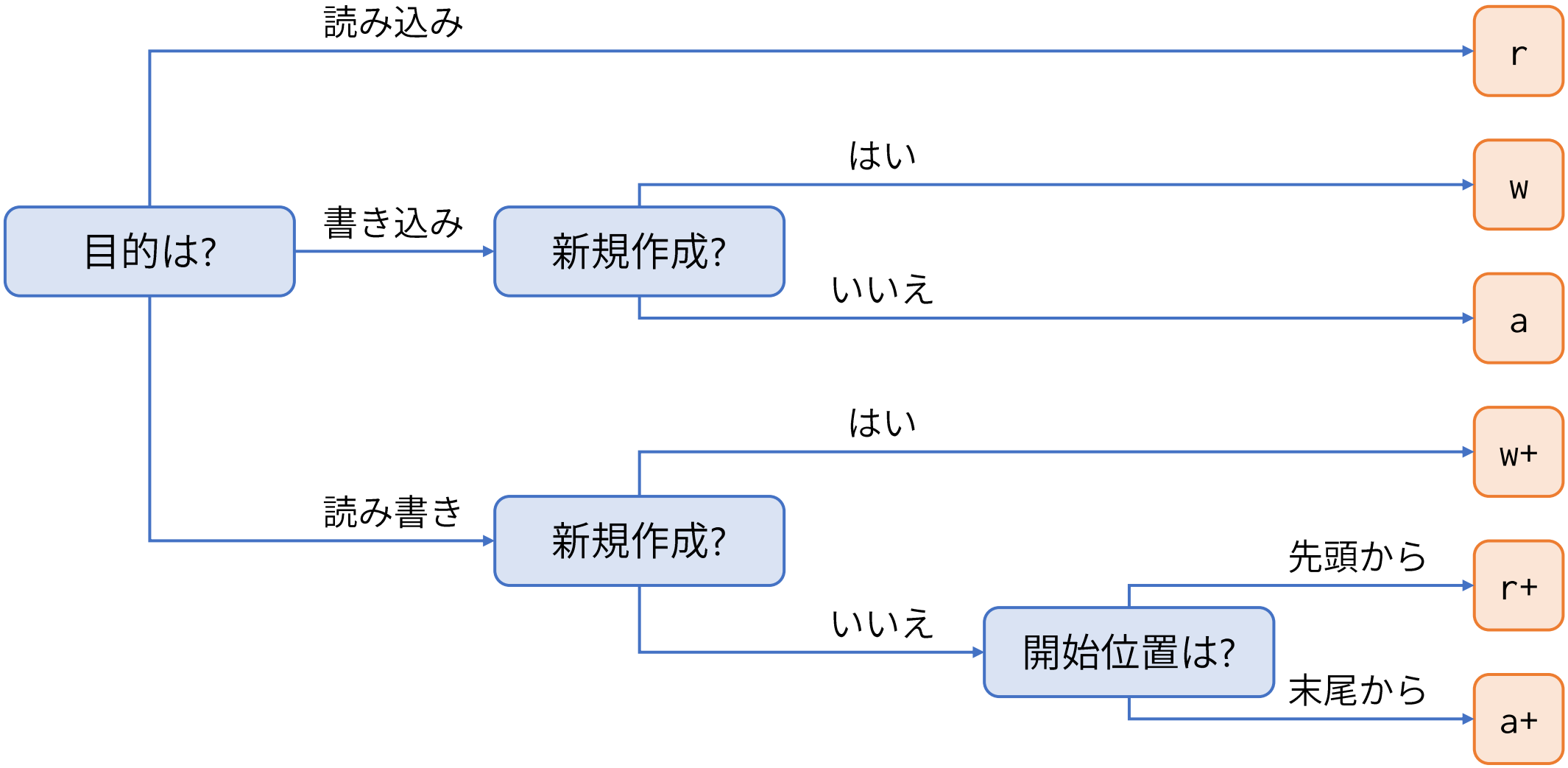
- 開いたファイルを区別する(どのファイルに対して操作を行うか指定する)ために，ファイルオブジェクトが返される．
- ファイルに対する操作は，ファイルオブジェクトのメソッドによって行う．

open()のモード

- open()の第2引数では，ファイルが開かれるモードを，以下を組み合わせた文字列で指定する．

文字	意味
r	読み込み用 [デフォルト]
w	書き込み用 (既に存在しているファイルの内容は消去する)
x	排他的に生成する (既に存在していると失敗する)
a	ファイル末尾に追記する
t	テキストモード [デフォルト]
b	バイナリモード
+	読み書き両用に開く

モード選択チャート


目的は?
読み込み
r
書き込み
新規作成?
はい
w
いいえ
a
読み書き
新規作成?
はい
w+
いいえ
開始位置は?
先頭から
r+
末尾から
a+

ファイルのオープンについて補足

- ファイルの読み込みは，ファイルの先頭から順になされる．一度ファイルをclose()して，再度open()すると，読み込みは先頭からに戻る．
  - なので，一度open()したら，必要なデータを読み書きしている間は開いたままのことが多い．
- 同時に開けるファイルの数には上限がある．変数のように際限なく同時使用することはできない．
- モードは省略するとテキストファイル(t)の読み込み(r)になる．

同じ

```
f = open('indata.txt', 'rt')
```

同じ

```
f = open('indata.txt', 'r')
```

同じ

```
f = open('indata.txt')
```

同じ

```
f = open('indata.txt', 'wt')
```

同じ

```
f = open('indata.txt', 'w')
```

ファイルを閉じる 3

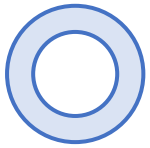
- 使い終わったファイルはメソッド close() で閉じる.

```
f.close();
```

ファイルへの書き込み

ファイルに文字列を書き込む

- 書き込み用のメソッド: write(string)
  - stringの内容をファイルに書き込む
  - 戻り値: 書き込まれた文字数
- テキストファイルの場合，整数型なども全て文字列に変換して書き込む．
  - 整数型のまま引数に与えても出力されずエラーになるので注意．



```
f.write('42¥n')
```

```
f.write(42)
```

- ファイルを書き込み可能でオープンしておく必要がある．

書き込みについての補足

- メソッド write() はあくまでも文字列を「そのまま」書き出すので，改行は自分で付け加える必要がある．
  - 画面に表示する組み込み関数 print() とは異なるので注意．
- ファイルを閉じる処理 close() あるいは途中で flush() しないと，ファイルへの書き込みが正常になされない可能性があるので注意．
  - close()すれば，あらかじめflush()しておく必要はない．

エスケープシーケンス escape sequence

- 特殊な機能や文字を表現するための文字
  - ソースコードに記述される際には複数文字だが，論理的には1文字として扱われる．
- 覚えておくべきエスケープシーケンス
  - ¥n 改行
  - ¥t タブ
  - ¥¥ バックスラッシュ
  - ¥' シングルクォーテーション(')
  - ¥" ダブルクォーテーション(")

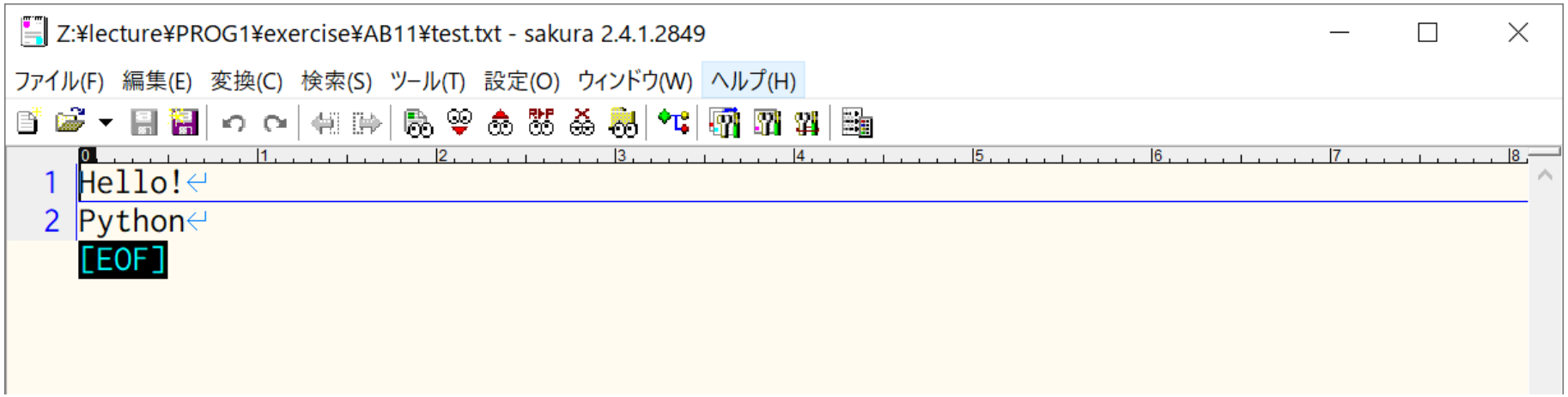
Pythonの場合は，
文字列中に ' を記述したい場合には文字列全体を "..." で，
文字列中に " を記述したい場合には文字列全体を '...' で，
それぞれ囲むことで表現できるため，
他の言語ほど使用頻度は高くない．

- ￥と＼(本当はいずれも半角)はフォントによって見た目が変わるが同じもの
  - 同じ文字コードが割り当てられているため

書き込みの例

```
f = open('test.txt', 'wt')
f.write('Hello!¥n')
f.write('Python¥n')
f.close()
```

出力結果: test.txt の内容



複数の文字列を一括して書き込む

- 文字列のリストを書き込むメソッド: writelines(lines)
  - 文字列を要素とするリスト lines を与えると，要素を順にファイルに書き出す．
  - 特に改行文字を追加してくれるわけではない．
- 実質的に，以下と同じ機能を有する．

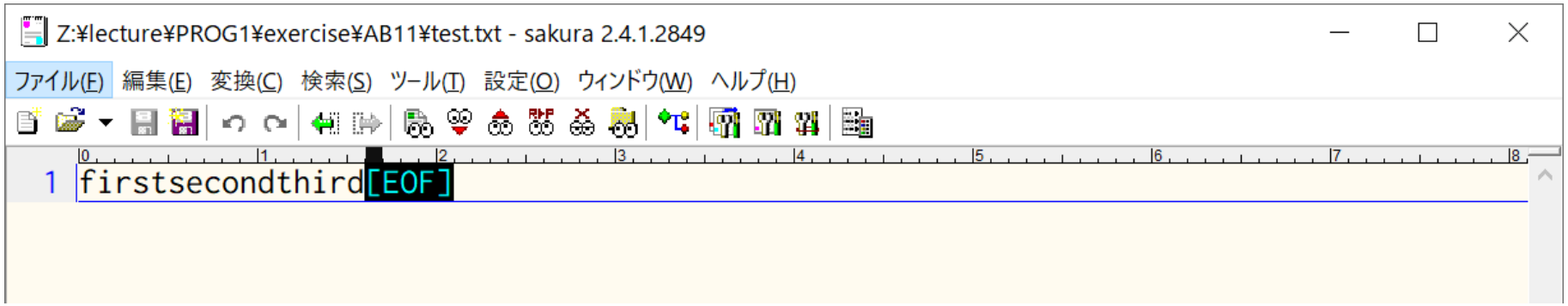
```
f = open('tmp.txt', 'wt')
lines = ['first¥n', 'second¥n', 'third¥n']
for line in lines:
    f.write(line)
f.close()
```

```
f = open('tmp.txt', 'wt')
lines = ['first¥n', 'second¥n', 'third¥n']
f.writelines(lines)
f.close()
```

ありがちなミス

```
f = open('test.txt', 'wt')
lines = ['first', 'second', 'third']
f.writelines(lines)
f.close()
```

出力結果: test.txt の内容



そうだった，改行は自分で付与しないと…

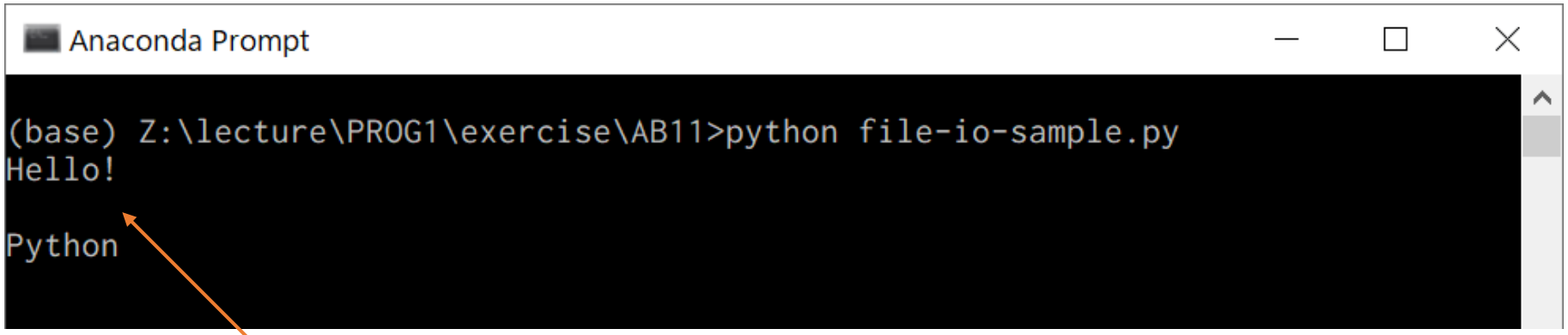
ファイルからの読み込み

ファイルから文字列を読み込む

- 読み込み用のメソッド: `readline()`
  - ファイルから1行読み込み，文字列として返す．
- テキストファイルの場合，文字列が返される．
  - 数値として使用する場合は変換が必要である．
- 文字列の末尾に改行文字が付いたままになる．
  - 改行文字を削除したい場合は文字列型のメソッド `strip()` を用いる．
- ファイルを読み込み可能なようにオープンしておく必要がある．

読み込みの例 (行末の改行の除去なし)

```
f = open('test.txt', 'rt')
line = f.readline()
print(line)
line = f.readline()
print(line)
f.close()
```

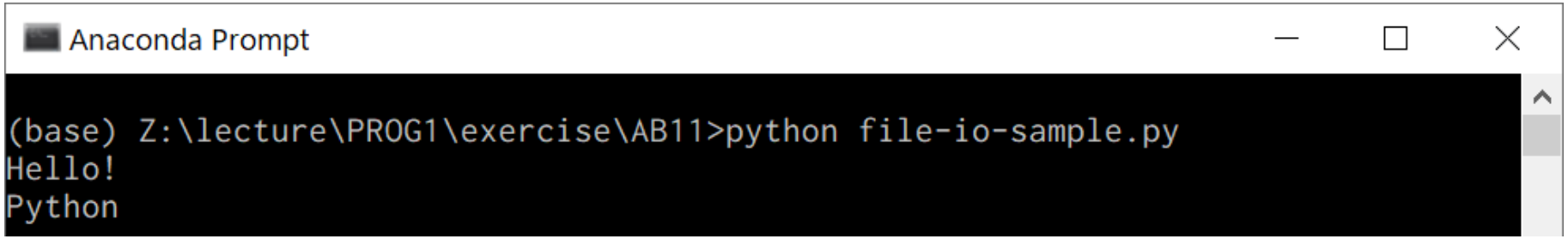


strip() しないと，読み込んだ文字列 line の末尾の改行 + print() の改行で
2回改行して空行が入ってしまう．

読み込みの例 (行末の改行の除去あり)

```
f = open('test.txt', 'rt')
line = f.readline().strip()
print(line)
line = f.readline().strip()
print(line)
f.close()
```

f.readline() までがまず実行されて，文字列になる．
この文字列に対して，メソッド strip() を実行．



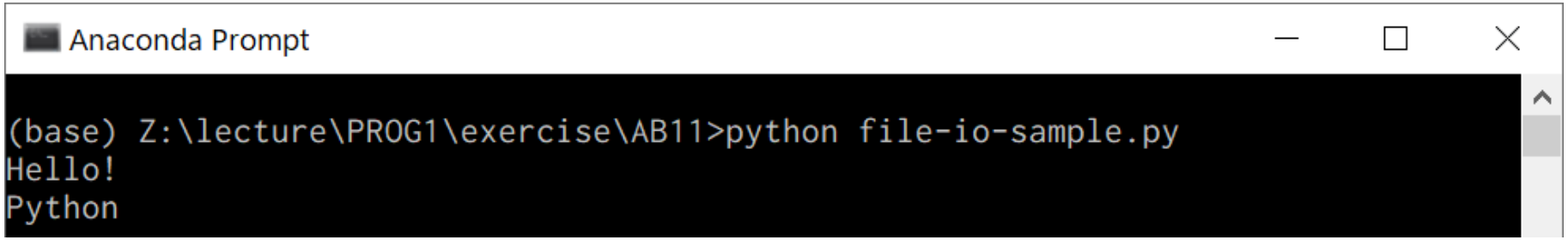
常にstrip()した方がよいわけではない．用途によって使い分ける．

for文を用いた複数行の読み込み

- ファイルオブジェクトをfor文のinの後ろに与えると，1行読み込みが行われ，戻り値が変数にセットされる．

```
f = open('test.txt', 'rt')
for line in f:
    print(line.strip())
f.close()
```

結果は先程と同じ



参考: 複数の文字列を読み込むメソッド

- 複数行を読み込むメソッド: readlines()
  - 引数として読み込む最大サイズが指定できる.
  - 巨大なファイルに対して引数なしで使うと，ファイル全体を一括してメモリに読み込もうとするため注意が必要.

```
f = open('test.txt', 'rt')
lines = f.readlines()
f.close()
for line in lines:
    print(line.strip())
```

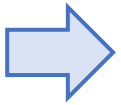
ここまででファイル処理は終了している

with構文

with構文

- with構文を使うと，ファイルを使い終えたときの後処理を自動的に行ってくれる．
  - close()を明示的に実行する必要がない．
  - 途中でエラーが生じた場合も，適切に後処理を行ってくれる．

```
f = open('test.txt', 'rt')
for line in f:
    print(line.strip())
f.close()
```



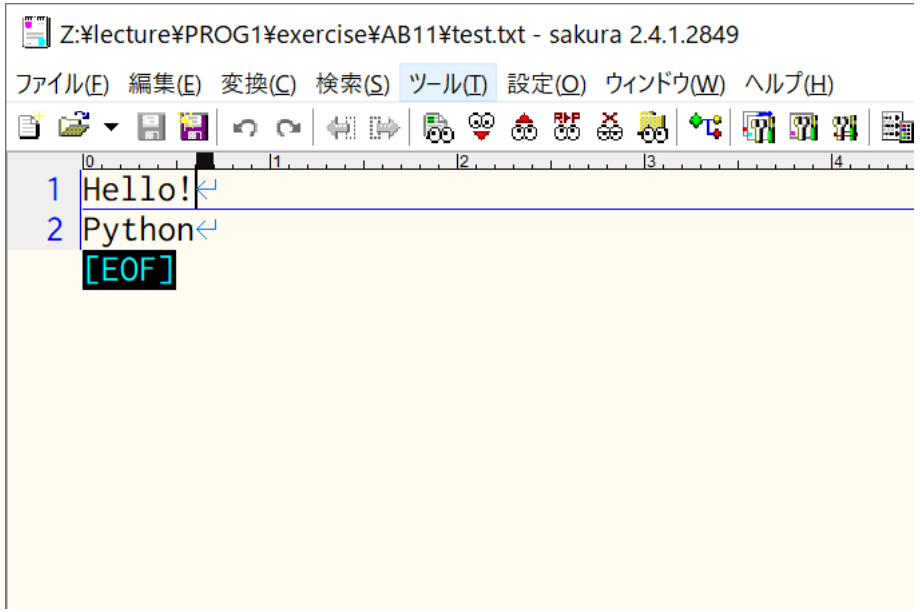
```
with open('test.txt', 'rt') as f:
    for line in f:
        print(line.strip())
```

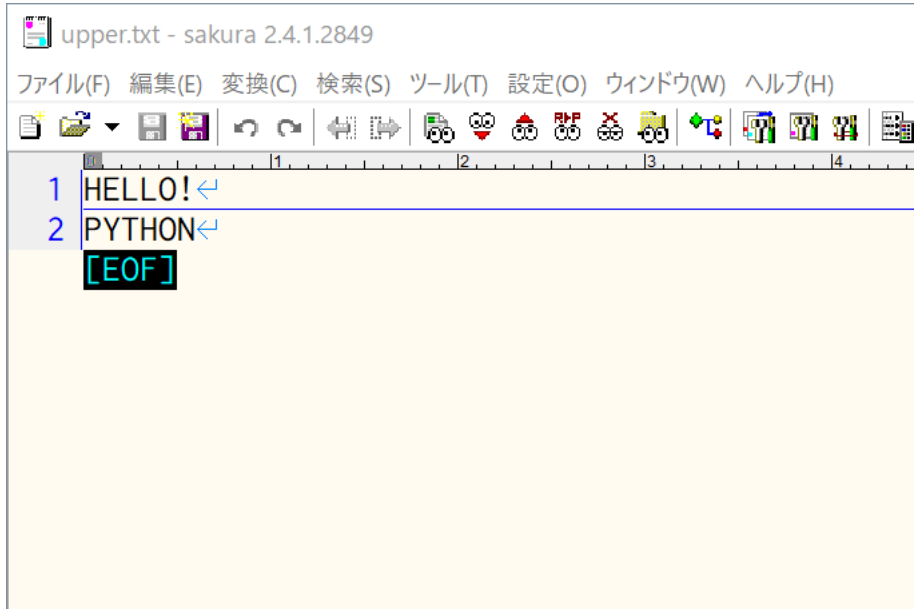
with構文を使ってファイルを開く方が
便利なので積極的に使おう．

with構文で2つのファイルを同時に扱う

- with の後ろにカンマ(,)で区切った open … as を並べる．

```
with open('test.txt', 'rt') as f_in, open('upper.txt', 'wt') as f_out:
    for line in f_in:
        f_out.write(line.upper())
```





文字列の便利なメソッド

文字列の便利なメソッド

- 改行を取り除く: strip()
  - 本当は，先頭と末尾両方の，空白や改行を取り除く
  - https://docs.python.org/ja/3/library/stdtypes.html#str.strip
- 文字列を分割する: split(sep)
  - sepに区切り文字を指定する
    - 文字列でも可
  - https://docs.python.org/ja/3/library/stdtypes.html#str.split
- リストの要素を結合する: join(lists)
  - listsの各要素を，間に文字列を挟んで結合する
  - https://docs.python.org/ja/3/library/stdtypes.html#str.join

メソッド join() の例

- 区切り文字.join(リスト)なのがちょっと奇妙に思えるかも

```
>>> numbers = ['one', 'two', 'three']
>>> '-'.join(numbers)
'one-two-three'
>>> '<>'.join(numbers)
'one<>two<>three'
```

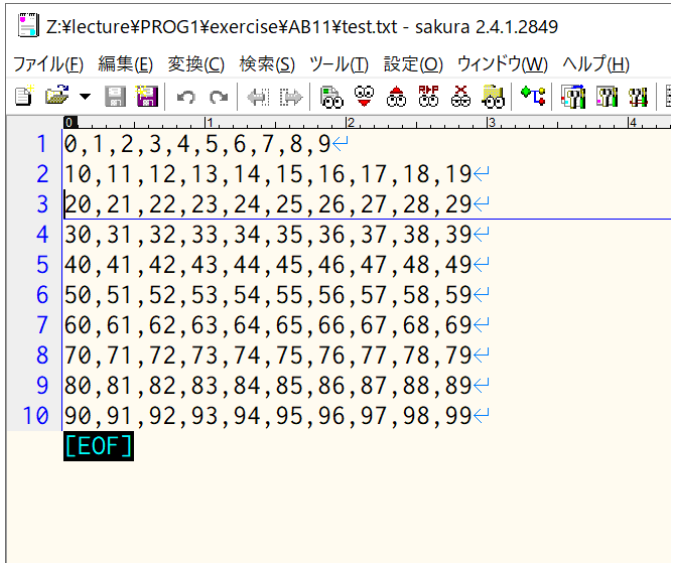
- split()と組み合わせる例: 区切り文字の変更

```
>>> input_str = 'confusion will be my epitaph'
>>> '...'.join(input_str.split(' '))
'confusion...will...be...my...epitaph'
```

メソッドの連続適用

- 例: ファイルから読み込んだ1行の文字列から，末尾の改行を取り除いて(strip())，カンマで区切る(split()).

```
f = open('test.txt', 'rt')
for line in f:
    print(line.strip().split(','))
f.close()
```



```
(base) Z:\lecture\PROG1\exercise\AB11>python file-io-sample.py
['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
['10', '11', '12', '13', '14', '15', '16', '17', '18', '19']
['20', '21', '22', '23', '24', '25', '26', '27', '28', '29']
['30', '31', '32', '33', '34', '35', '36', '37', '38', '39']
['40', '41', '42', '43', '44', '45', '46', '47', '48', '49']
['50', '51', '52', '53', '54', '55', '56', '57', '58', '59']
['60', '61', '62', '63', '64', '65', '66', '67', '68', '69']
['70', '71', '72', '73', '74', '75', '76', '77', '78', '79']
['80', '81', '82', '83', '84', '85', '86', '87', '88', '89']
['90', '91', '92', '93', '94', '95', '96', '97', '98', '99']
```

for文と一緒に使うと便利な機能

繰り返すときにインデックス番号も同時に欲しい

- enumerate() は，引数としてリストを与えると，(インデックス番号, 値) のタプルを順に返す．

```
>>> seasons = ['春', '夏', '秋', '冬']
>>> for i, season in enumerate(seasons):
...     print(f'{i}: {season}')
...
0: 春
1: 夏
2: 秋
3: 冬
```

複数のリストから同時に要素を取り出して繰り返したい

- zip() は，引数として複数のリストを与えると，各リストから1要素ずつ取り出したタプルを順に返す．

```
>>> seasons_en = ['spring', 'summer', 'autumn', 'winter']
>>> seasons_jp = ['春', '夏', '秋', '冬']
>>> for e, j in zip(seasons_en, seasons_jp):
...     print(f'{e}: {j}')
...
spring: 春
summer: 夏
autumn: 秋
winter: 冬
```