

2022年09月26日(月) 4限
@研究棟A302

プログラミングA2 第1回

担当：伏見卓恭

連絡先：fushimity@edu.teu.ac.jp

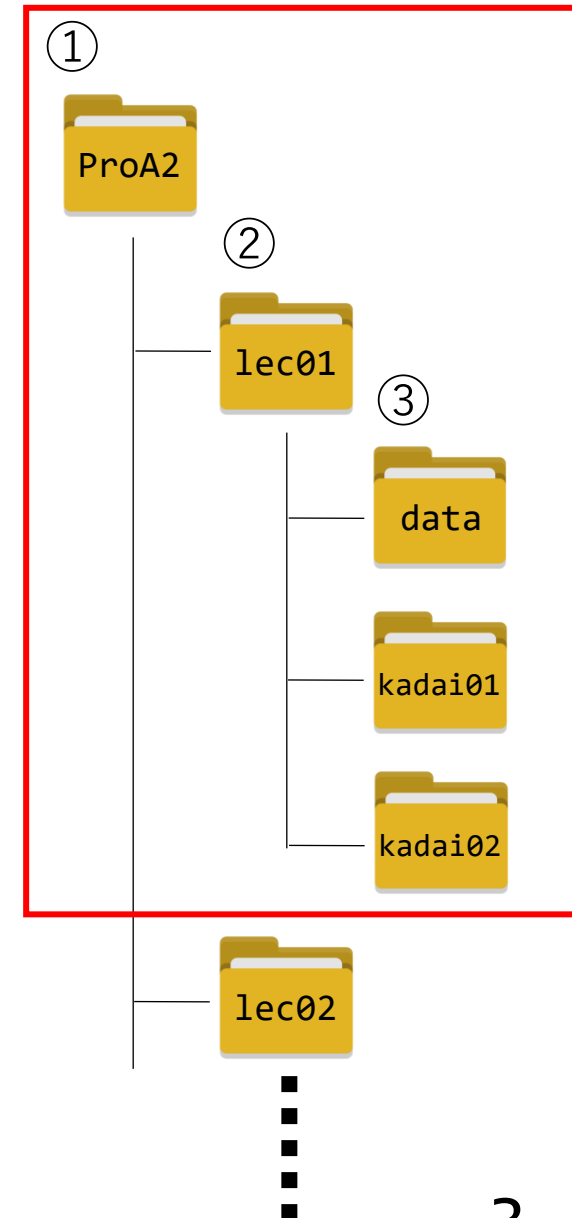
居室：研A1201

プログラミングA2の流れ

- 第 1回：＜復習編＞関数，ファイル入出力，コンテナデータ型
- 第 2回：＜復習編＞クラスとオブジェクト
- 第 3回：＜文法編＞関数の高度な利用法 1
- 第 4回：＜文法編＞関数の高度な利用法 2
- 第 5回：＜文法編＞オブジェクト指向プログラミング
- 第 6回：＜応用編＞データ構造とアルゴリズム 1
- 第 7回：＜応用編＞データ構造とアルゴリズム 2
- 第 8回：＜実践編＞HTTPクライアント
- 第 9回：＜実践編＞スクレイピング
- 第10回：＜実践編＞データベース
- 第11回：＜実践編＞並行処理
- 第12回：＜総合編＞総合演習(複合問題)
- 第13回：＜総合編＞まとめ
- 第14回：＜総合編＞Python力チェック ← 確認テストのこと

準備

1. デスクトップなどの任意の場所にプログラミングA2用のフォルダを作成する
2. 1.の下に第1回授業用のフォルダを作成する
3. 2.の下にデータファイル用のフォルダ、課題01、課題02用のフォルダを作成する



本日のお品書き

● ファイル入出力の復習

- open/close
- with構文

● コンテナデータ型の復習

- コンテナオブジェクトの性質
- 内包表記
- リスト/辞書固有のメソッド
- ソート

● 関数の復習

- 定義方法
- 使用方法

ファイル入出力

ファイル入力

通常、プログラムとデータは独立して別々のファイルに記述する。
外部ファイルに書かれたデータを読み込み、処理する。

基本的な手順

- ① 対象ファイルのファイルパスを定義する
- ② `open()`関数で`r`readモードで開き、ファイルオブジェクトを得る
- ③ ファイルの中身を読み込み、変数に格納する
- ④ `close()`メソッドでファイルオブジェクトが指す対象ファイルを閉じる

① ファイルパスの定義 `file_path = "ファイルのある場所/ファイル名"`

② 開く `rfo = open(file_path, "r")`

※readモードのときは `r` を省略可

④ 閉じる `rfo.close()`

③ 読み込み用の関数

● **read(x) : x文字だけ読み込む**

※引数xが負の値 or Noneなら、
全体読み込み

```
文字列 = ファイルオブジェクト.read()
```

```
txt = rfo.read()
```

● **readline() : 1行だけ読み込む**

※改行文字も読み込む

```
文字列 = ファイルオブジェクト.readline()
```

```
row = rfo.readline()
```

● **readlines() : 全体を行ごとに読み込む**

※改行文字も読み込む

```
行ごとのリスト = ファイルオブジェクト.readlines()
```

```
rows = rfo.readlines()
```

読み込んだデータの処理

● `rstrip()`：行末改行文字の削除

※`rstrip()`自体は、右側(r)から
引数で指定した文字を削除する

```
文字列 = 文字列.rstrip()
```

```
row = row.rstrip()
```

● `split(d)`：文字列をデリミタdでの分割

```
リスト = 文字列.split(デリミタ)
```

```
cols = row.split(" ")
```

← 文字列`row`に含まれる半角スペース" "で
分割した結果をリスト`cols`に代入している

例題：file_open.py

file_open.py

```
rfo = open("lec01/data/poke_names.txt", "r")
while True:
    row = rfo.readline().rstrip()
    if row:
        print(row)
    else:
        break
rfo.close()
```

← 1行ずつ読み込み, 改行文字を削除

← 空行でないなら

windowsな方は,

エラー「UnicodeDecodeError: 'cp932' codec can't decode」が出る.

対応策：以下を追記

```
rfo = open("lec01/data/poke_names.txt", "r", encoding="utf8")
```

イテレータとして扱う場合

ファイルオブジェクトは、行単位で要素を取り出せるイテレータである

file_open_iter.py

```
rfo = open("lec01/data/poke_names.txt", "r")
```

```
for row in rfo:
```

← ファイルオブジェクトから1行ずつ取り出し

```
    row = row.rstrip()
```

```
    if row:
```

```
        print(row)
```

```
    else:
```

```
        break
```

```
rfo.close()
```

ファイル出力

処理結果を外部ファイルに書き込むために、ファイルを開く

基本的な手順

- ① 対象ファイルのファイルパスを定義する
- ② `open()`関数で`w`riteモードで開き、ファイルオブジェクトを得る
- ③ ファイルに書き込む
- ④ `close()`メソッドでファイルオブジェクトが指す対象ファイルを閉じる

① ファイルパスの定義 `file_path = "ファイルのある場所/ファイル名"`

② 開く `wfo = open(file_path, "w")`

※writeモードのときは `w` を省略できない
※追記する場合は、 `a` を指定する

③ 書き込む `wfo.write(書き込む文字列)`

④ 閉じる `wfo.close()`

with構文

with構文を使用することで、コンテキストマネージャーの中でエラーが発生した場合でも、正常に終了処理をすることができる

※コンテキストの例：ファイルopen～close／データベース接続～切断など

with_open.py

コンテキストマネージャー：open関数

```
with open("lec01/data/poke_names.txt", "r") as rfo:
    for row in rfo:
        print(row.rstrip())
```

← withブロックから出る際に自動でclose()されているためclose()を省略できる

比較：file_open_exc.py

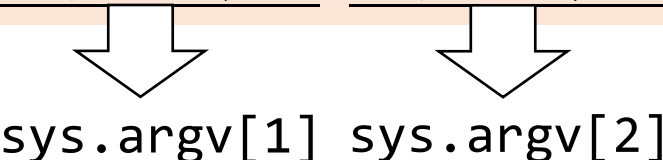
```
rfo = open("lec01/data/poke_names.txt", "r")
try:
    for row in rfo:
        print(row.rstrip())
finally:
    rfo.close()
```

コマンドライン引数 `sys.argv`

プログラム実行時に、プログラム名の後にパラメータを指定することで、プログラムに引数として値を渡すことができる

文法

```
python プログラム名.py パラメータ 1 パラメータ 2 . . .
```



`sys.argv[1]` `sys.argv[2]`

cmdline_argv.py

```
import sys

txt = sys.argv[1]
num = int(sys.argv[2])
for _ in range(num):
    print(txt)
```

実行例

```
$ python lec01/cmdline_argv.py hoge 3
hoge
hoge
hoge
```

※コマンドライン引数は文字列として扱われる
→ 数値として扱いたい場合は、
変換する必要がある

例： `n = int(sys.argv[2])`

例： `x = float(sys.argv[2])`

練習問題：read_names.py

データファイルlec01/data/poke_names.txtを1行ずつ読み込み，行番号がmの倍数の時だけ標準出力するコードを実装せよ

[要件]

1. with構文を用いること
2. 読み込みファイル名とmをコマンドライン引数とすること

実行例

```
$ python lec01/read_names.py lec01/data/poke_names.txt 5
5      リザード      ほのお
10     キャタピー    むし
15     スピアー      むし どく
:
245    スイクン      みず
250    ホウオウ      ほのお ひこう
```

解答例：read_names.py

read_names.py

```
import sys
```

```
file_path = [ ][1]
```

```
m = int([ ][2])
```

```
[ ](file_path, "r") as rfo:  
    for i, row in enumerate(rfo, 1):  
        row = row.rstrip()  
        if [ ]:  
            print(row)
```

```
[ ]
```

コンテナデータ型

コンテナ

複数のオブジェクトを格納できるオブジェクトのことで、`in`演算子を利用できる

●標準的なコンテナ

- 文字列 (`str`) : シーケンス, `immutable`, `iterable`
- リスト (`list`) : シーケンス, `mutable`, `iterable`
- タプル (`tuple`) : シーケンス, `immutable`, `iterable`
- 辞書 (`dict`) : ~~シーケンス~~, `mutable`, `iterable`
- 集合 (`set`) : ~~シーケンス~~, `mutable`, `iterable`

●似た概念

- コレクション : `iterable`かつ`sized`なコンテナ
- シーケンス : 要素を順序立てて処理できる
 - リスト, タプル, 文字列, `range`オブジェクトなど
 - シーケンス[整数]で要素にアクセスできる

オブジェクトの性質

- **iterable** : `for`で繰り返し可能なオブジェクト
 - 例 : コンテナやファイルオブジェクト
- **sized** : `len()`で要素数を返すオブジェクト
- **mutable** : 作成後に値を変更できるオブジェクト
 - 例 : リスト, 集合, 辞書など
 - 文字列, タプルはmutableでない (=immutable)
 - ※再代入は, 別オブジェクトへの変数名の付け替えなので可能
 - ※immutableなものはhashable
 - ※hashableなものだけが, 辞書のキー, 集合の要素とすることが可能
- **callable** : `()`を付して呼び出せるオブジェクト
 - 例 : クラス, 関数, ラムダ式
 - ジェネレータはcallableでない

内包表記

● メリット

- 短く簡潔に書ける／処理速度が速い

● 書き方

- リスト：`[式 for 変数 in iterable if 条件式]`

↑ 要素となる

- タプル：ない

※以下はジェネレータとなる

`(式 for 変数 in iterable if 条件式)`

- 集 合：`{式 for 変数 in iterable if 条件式}`

↑ 要素となる

- 辞 書：`{式:式 for 変数 in iterable if 条件式}`

↑ キー:値となる

※式が三項選択式(`if-else`)であるのと、条件式は別物

`[三項選択式 for 変数 in iterable]`

例題：read_stats.py

251匹のポケモンの6種類の種族値が書かれたデータファイルlec01/data/base_stats.txtを1行ずつ読み込み、intに変換した値を2次元リストstatsに格納する例

```
read_stats.py
```

```
from pprint import pprint
```

```
stats = []
```

← stats自体は空の1次元リストとして初期化

```
with open("lec01/data/base_stats.txt", "r") as rfo:
```

```
    for row in rfo:
```

```
        row = row.rstrip()
```

```
        stats.append([int(col) for col in row.split(" ")])
```

↑ appendする要素をリストにすることで2次元リストを実現

```
pprint(stats)
```

練習問題：read_stats_dct.py

lec01/data/base_stats.txtを，辞書の一覧として読み込め．

辞書のキーは左から順に，hp，attack，defense，sp_atk，sp_def，speedとする．

つまり，1匹の種族値を1つの辞書とし，その辞書を251個並べたリストstatsを構築する．

実行例

```
$ python lec01/read_stats_dct.py
[{'attack': 49, 'defense': 49, ... 'speed': 45},
 {'attack': 62, 'defense': 63, ... 'speed': 60},
 {'attack': 82, 'defense': 83, ... 'speed': 80},
 :
 {'attack': 100, 'defense': 100, ... 'speed': 100}]
```

解答例：read_stats_dct.py

read_stats_dct.py

```
from pprint import pprint
```

```
stats = []
```

← stats自体は空の1次元リストとして初期化

```
key_lst = ["hp", "attack", "defense",  
           "sp_atk", "sp_def", "speed"]
```

```
with open("lec01/data/base_stats.txt", "r") as rfo:
```

```
    for row in rfo:
```

```
        cols = row.rstrip().split(" ")
```

```
        stats.append(
```

辞書の内包表記

)

```
pprint(stats, width=100)
```

リスト固有のメソッド

● `sort(key=None, reverse=False)` : リストを破壊的にソートする

- `key` : 並び替える際の基準を指定する
- `reverse` : 降順ソートしたい場合は`True`にする
- 戻り値はない

【リスト固有のメソッド】

※要素の順序を保持するシーケンスであるタプルや文字列は`immutable`であるため、破壊的なメソッドは定義されていない

※要素に順序がない辞書や集合には、そもそもソートという概念がない

リスト.`sort()`

プログラム例

```
lst = [5, 9, 2, 4, 1]
lst.sort()
print(lst)
```

```
print(lst.sort())
```

← [1, 2, 4, 5, 9]

← None

※戻り値がない関数やメソッドを`print`すると`None`になる

ラムダ式によるソート基準の指定

ラムダ式の詳細は第3回で説明する

key = lambda リストの要素 : 要素に対する処理

●2次元リストstatsに対して

- 1列目(0始まり)の値で降順ソートする例

sort_stats.py

```
stats.sort(key=lambda s:s[1], reverse=True)
```

↑
sort対象の要素
今回はstatsの要素(=リスト)

↑
sort基準
今回はstatsの要素(=リスト)の1列目

- 合計値で降順ソートする例

sort_stats.py

```
stats.sort(key=lambda s:sum(s), reverse=True)
```

↑
sort基準
今回はstatsの要素(=リスト)の和

練習問題：sort_stats_dct.py

read_stats_dct.pyで読み込んだ辞書のリストstatsに対して、辞書のキーattackで降順ソートせよ

sort_stats_dct.py

読み込み + 辞書のリスト構築部分省略

```
stats.sort(key=lambda s: , reverse=True)
for i, stat in enumerate(stats, 1):
    print(stat)
    if i == 5: break
```

← 上位5件のみ表示している

実行例

```
$ python lec01/sort_stats_dct.py
{'hp': 91, 'attack': 134, 'defense': 95, 'sp_atk': 100, 'sp_def': 100, 'speed': 80}
{'hp': 100, 'attack': 134, 'defense': 110, 'sp_atk': 95, 'sp_def': 100, 'speed': 61}
{'hp': 90, 'attack': 130, 'defense': 80, 'sp_atk': 65, 'sp_def': 85, 'speed': 55}
{'hp': 55, 'attack': 130, 'defense': 115, 'sp_atk': 50, 'sp_def': 50, 'speed': 75}
{'hp': 105, 'attack': 130, 'defense': 120, 'sp_atk': 45, 'sp_def': 45, 'speed': 40}
```

辞書固有のメソッド

- `keys()` : ビュー = 辞書.`keys()`
辞書からキーだけを抽出したビューを返す
- `values()` : ビュー = 辞書.`values()`
辞書から値だけを抽出したビューを返す
- `items()` : ビュー = 辞書.`items()`
辞書からタプル(キー, 値)のビューを返す

※ビュー：リストではないがiterableなので、`for _ in ビュー`として使える

プログラム例

```
dct = {"name": "fsm", "age": 45, "level": 99}
```

```
print(dct.keys())  
print(dct.values())  
print(dct.items())
```

```
← dict_keys(['name', 'age', 'level'])
```

```
← dict_values(['fsm', 45, 99])
```

```
← dict_items([('name', 'fsm'), ('age', 45), ('level', 99)])
```

KeyErrorの回避方法

存在しないキーへのアクセス時に発生するKeyErrorを回避するために条件文や例外処理をするよりget()メソッドやsetdefault()メソッドを使用することで簡潔に記述できる

- **get(key, value)** : 辞書.get(キー, デフォルト値)
指定したキーに対応する値を取得する際、
キーが存在しない場合はデフォルト値を返す
- **setdefault(key, value)** : 辞書.setdefault(キー, デフォルト値)
指定したキーが存在しない場合はデフォルト値を設定し、
存在する場合は何もしない
- **defaultdict(type)** : 辞書 = collections.defaultdict(データ型)
指定したデータ型をデフォルト値とした辞書を作成する

プログラム例

```
from collections import defaultdict
# dct = dict()
dct = defaultdict(int)
dct[24] += 1
print(dct)
```

← この場合、存在しないキー24にアクセスした時点でKeyError発生

← defaultdict(<class 'int'>, {24: 1})

辞書のソート

※辞書の要素をソートすることを考える
辞書を要素としたリストのソートではない

辞書はシーケンスではないため、辞書のままソートすることはできない
＝破壊的なメソッドは存在しないが、

- ① `items()`メソッドによりタプル(キー, 値)からなるビューを構築し、
- ② `sorted()`関数によりソート後のリストを得ることができる。

● `sorted(シーケンス, key=None, reverse=False)` : ソート後のリストを返す

プログラム例

```
stat = {'hp': 45, 'attack': 49, ..., 'sp_def': 65, 'speed': 45}
print(sorted(stat.items(), reverse=True))
```

▶ 辞書の**キー文字列**で降順ソートされる

```
[('speed', 45), ('sp_def', 65), ('sp_atk', 65), ('hp', 45), ('defense', 49), ('attack', 49)]
```

プログラム例

```
print(sorted(stat.items(), key=lambda t:t[1], reverse=True))
```

▶ 辞書の**値**で降順ソートされる

↑
タプル(キー, 値)

↑
タプルの1番目(=値)

```
[('sp_atk', 65), ('sp_def', 65), ('attack', 49), ('defense', 49), ('hp', 45), ('speed', 45)]
```

関数

関数

関数は、呼び出し元から受け取った値（パラメータ、引数）を処理し、処理した結果を呼び出し元に戻す・返すオブジェクト

●関数定義

```
def 関数名(パラメータ):  
    処理1  
    処理2  
    return 結果
```

●関数使用

戻り値 = 関数名(引数)

引数として指定した値がパラメータ変数に代入されて、関数の中で処理が行われるため

基本的には引数の数とパラメータの数は一致する。

●ローカル変数

- 関数内で定義された変数
- 関数外からアクセス不可

●グローバル変数

- 関数外で定義された変数
- 関数内で値を変更するには `global` 宣言が必要

●引数(ひきすう)

引数の詳細は
第3回で説明する

- 位置引数
- デフォルト引数
- キーワード引数
- 可変長引数

※引数がない関数もある

dunder属性：__name__

※ダンダー（dunder）とは、double underscoreのことで、pythonでは、左右にダンダーがある属性やメソッドは特殊な役割を果たす

'__main__' はトップレベルのコードが実行される名前空間を表す。モジュール（***.pyファイル）が、標準入力から読み込まれたとき、スクリプトとして実行されたとき、インタラクティブプロンプトのとき、__name__ には '__main__' が設定される。

- 他ファイルからimportされたときではなく、コマンドラインから実行されたときのみに動くようにする

```
if __name__ == "__main__":
```

練習問題：stats_average.py

以下の要件を満たすコードを実装せよ.

[要件]

1. read_stats(ファイルパス)関数

- 引数で指定したファイルを読み込む
- 読み込んだ値を2次元リストに格納する
- 2次元リストをreturnする

2. calc_average(1次元リスト)関数

- 引数で指定したリストの平均 sum/len をreturnする

3. トップレベル

- read_stats()関数にlec01/data/base_stats.txtを引数として渡し、2次元リストを戻り値として得る
- calc_average()関数に列ごとのリストを引数として渡し、平均値を戻り値として得る
- 列番号と平均値を標準出力する

解答例：stats_average.py

stats_average.py

```
def read_stats():  
    stats = []  
    with open(file_path, "r") as rfo:  
        for row in rfo:  
            row = row.rstrip()  
            stats.append([int(col) for col in row.split(" ")])  

```

```
def calc_average(lst):  
    return 
```

```
if __name__ == "__main__":  
    stats = read_stats()  
    for col_idx in range(6):  
        avg = calc_average( 内包表記で列ごとのリストを作成)  
        print(col_idx+1, f"{avg:.2f}")
```

実行例

```
$ python lec01/stats_average.py  
1 66.75  
2 70.64  
3 68.61  
4 65.81  
5 68.30  
6 65.86
```