ファイル入出力

名古屋大学 情報基盤センター
情報基盤ネットワーク研究部門
基盤ネットワーク研究グループ
嶋田 創

今回の講義のゴール

ファイル入出力はいろいろ発展内容があるが、とりあえず、以下のことができることを今回の講義のゴールとする

- ファイル読み込みは、文字列の1行単位での読み込みができればOK
- ファイル書き込みは、文字列の新規書き出しと追記ができればOK
- 読み書き対象のファイルは実行時の引数で指定できればOK

ファイルの操作

- 一般的なプログラミング言語では、「ファイルデスクリプタ」という変数みたいな物と個々のファイルを関連付けて操作
  - ファイルデスクリプタに対して処理を定義するので、複数ファイル処理時は関連付けを変えて繰り返すだけでOK
- ファイルの操作開始前に実ファイルとファイルデスクリプタを関連付ける(ファイルのオープン)
  - 「読み込み用にオープン」「新規書き込み(上書き)用にオープン」「追記書き込み用にオープン」など、オープン時に操作を決める
- ファイルの操作終了後に実ファイルとファイルデスクリプタの関連を切る(ファイルのクローズ)
- ループ処理内で同じファイルデスクリプタに別ファイルを割り当ててオープン/クローズして操作するという使い方も

ファイルのオープン

- *ファイルデスクリプタ* = open(*ファイル名*[, *読み書きモード*])
  例: log_file = open('auth.log', 'r')
- 読み書きモード('rb'のように組み合わせることも)
  - r : 読み込み(省略時にはこれ)
  - w: 書き込み(上書き)
  - a: 追記書き込み
  - r+: 読み書き両方
  - b: バイナリモード(データを1byteの値として読み出す)
- 全角文字を読み込む時は必要に応じて文字エンコード指定
  - *ファイルデスクリプタ* = open(*ファイル名*, *読み書きモード*, encoding='*文字エンコード*')
    - 何も指定しないとユニコード(utf-8)
    - WindowsのシフトJIS(sjis)と古いUNIXのEUC(euc)は稀に使う

ファイル内容の読み込み(1/2)

- 1行ずつ読み込み:*ファイルデスクリプタ*.readline()
  - ループの中で「1行読み込み→行の内容を処理」の構造のプログラムはよく使われる
- 全部読み込み: *ファイルデスクリプタ*.read()
  - 改行も含めた文字列型の文字として読み込まれる
  - 使うのはおすすめしない(一気に大サイズのデータが来てもてあます)
  - 同様に各行で分割してリストに入れるreadlines()もあるが、使うのはおすすめしない
- ファイル内容の読み込み時に、ファイルディスクリプタには「どこまで読み込んだか」情報がある点に注意
  - *ファイルディスクリプタ*.seek(数字)で情報を変更できる
    - 最初に戻りたければ*ファイルディスクリプタ*.seek(0)
    - 単位はbyteなので注意(日本語などマルチバイト文字を途中で切ってしまうとエラーが出る)

ファイル内容の読み込み(2/2)

- 「ループの中で1行ずつ読み込みをしつつ処理」を行う時のための専用構文がある
  - for文でリストの代わりにファイルディスクリプタを指定

    「for *読み込んだ1行を入れる変数* in *ファイルデスクリプタ*」
    - ループの中で*読み込んだ1行を入れる変数*に対して処理を実施
  - 例: ファイルの内容をそのまま表示するプログラム

```
for line_data in fp:
    line_data = line_data.strip()
    print(line_data)
```

  - 読み込み時に行の末尾の改行が含まれたままになるので、そのままprint関数に渡すと改行が2連続することに注意
    - 上の例ではstrip()メソッドで改行を含む行頭/行末の空白を消してある

ファイルのクローズ

- ファイルのクローズ: *ファイルデスクリプタ*.close()
- プログラム終了時には自動的にファイルはクローズされる
  - Ctrl + Cによる強制終了なども含めて
- ループ処理とかで同じファイルデスクリプタを使いまわす時には、次のループでオープンする前にちゃんとクローズする
  - 例: ループでdata1.txt, ..., data10.txtを読み出す

```
for num in range(1, 11):
    filename = "data" + str(num) + ".txt"
    fp = open(filename, 'r')
    for line_data in fp:
        ...(1行ごとの処理)....
    fp.close()
```

ファイルへの書き込み

- 書き込み: *ファイルデスクリプタ*.write(*書き込み文字列*)
  - print関数とは違い、末尾に改行文字が入らない点に注意(必要ならば改行を入れたい部分に「¥n」を追加)

    例1: fp.write('改行は明示的に追加しないとだめ¥n')

    例2: fp.write(str(*write_data*) + '¥n')
    - 改行がないと、テキストエディタ等で書き出されたファイルを見た時に延々と続く1行が表示されることになる
    - 厳密には、MacやWindowsだと改行文字が異なるが(それぞれ¥rと¥r¥n)、Pythonが出力時に自動変換してくれる
- 書き込みフラッシュ: *ファイルデスクリプタ*.flush()
  - コンピュータへの負荷を減らすため、一般的に、出力処理はあるタイミングでまとめて行われる(人間にはその遅延はまず分からないが)

    →即座の出力が必要ならばflush処理を入れる
  - ファイル書き込み処理は基本的に「書き込みの成否を確認しない」

    →どうしても成否を確認するならばflush()の処理を入れて待つ

実行時の引数でのファイル指定(1/2)

- Pythonスクリプト実行時のように「program.py in_file out_file」の形式で実行したい →引数リストsys.argvを使う
  - 後ろのデータは「プログラムの引数」と呼ばれる(後で出る関数の引数と同じ文脈)
  - 「import sys」とsysモジュールのインポートを忘れずに
- 以下をスクリプトに入れてプログラムの引数を表示してみる

```
import sys
print(sys.argv)
```

  - sys.argv[0]にはプログラム名が入る点に注意
  - 例: 「program.py in_file.txt out_file.txt」を実行した時、sys.argv[0]にはprogram.py、sys.argv[1]にはin_file.txt、sys.argv[2]にはout_file.txtが入る
  - 「何個引数があるか」を知りたい場合はlen(sys.argv)-1を使う

実行時の引数でのファイル指定(2/2)

- あとはsys.argv[1]などを使ってファイルをオープンすればOK
  - 例: program.py in_file.txt out_file.txt

    ```
    import sys
    fp_in = open(sys.argv[1], 'r')
    fp_out = open(sys.argv[2], 'w')
    ....
    ```
  - 必要に応じて、「ファイルのオープンに失敗した場合」の処理も入れる
- 複数のファイルを順次処理したい場合はsys.argv[1:]で引数部だけのリストを作った方が便利
  - あとは、そのリストをfor文に渡して1ファイルずつオープン/処理/クローズ(スライド6のコード例も参考に)
  - もちろん、リストargvの長さを求めた上でsys.argv[1]から順にsys.argv[*index*]の形で配列を読み出す形でもOK

発展: バイナリファイルの読み書き

- 画像データなどは、データを文字として解釈するのではなく、データをそのままの2進数の数字として解釈する必要がある
  - 通常は1バイト(8ビット)単位で解釈

→バイナリファイルとして読み書き

- ファイルのオープン時に'rb'や'wb'の形でバイナリ形式で読み書きを指定
- nバイトのデータの読み込み: *ファイルデスクリプタ*.read(*n*)
  - データは文字列として読み込まれる(文字にならない値は'¥x8F'などの16進数表記)なので、必要に応じて変換
- データの書き込み: 文字列の書き込みに同じ
- 本格的にバイナリファイルを触るのならばstructモジュールを使うのが良い