# 文字処理

## マッチング

　pythonが最も得意とするのが、文字処理です。

【例題】

　テキストファイルinput5.txtのうち、先頭に # がつく行だけを抽出してoutput.txtに書き出しなさい。

import re

with open("input5.txt", "r", encodeing = "utf-8") as f:

with open("output.txt", "w") as f2:

for line in f: # 1 行ずつ line に入る

line = line.rstrip("\n") # 行末の改行コードを削除する

m = re.search("^#", line)

if m:

print(line, file = f2)

　文字処理のときに最もよく使う関数はre.serach()です。この関数を使うために、プログラム先頭で

import re

を記述します。

m = re.search("^#", line)

という行があります。変数 m は別の名前でも構いません。これは、lineという文字列の中に ^# にマッチする部分があるか否かをサーチします。^ は行頭を表します。従って、行頭に # がある場合、サーチ成功となります。サーチ成功のときmにはオブジェクトが入り、不成功のとき m には None が入ります。

if m:

と書くと、サーチに成功したときのみ、ifが成立します[[1]](#footnote-1)。output.txtの中は先頭に # を含む行のみとなります。

　この ^# などの表現を「正規表現」と言います。正規表現の主なものを以下に示します。

. 任意の１文字

a? 0 個あるいは 1 個の a

a\* 0 個以上の a

a+ 1 個以上の a

.\* 0 個以上の任意の文字 ex. "" "a" "c" "ab" "0x" など無数

.+ 1 個以上の任意の文字 ex. 上の例から "" を除く

^ 行の先頭

$ 行末

\s 空白文字 (スペース, タブ, 改行)

\S 空白以外の文字

[abx] a か b か x のいずれか

[0-9\-] 0～9 の文字あるいは - 記号

- 記号は 0-9 のように範囲を表す。- 記号そのものは \- で表す

【例題】

　http://denki.nara-edu.ac.jp/~yabu/soft/python.html にアクセスし、ファイルとしてセーブして下さい。ファイル名はpython.htmlにして下さい。

　python.htmlの中を見ると、見出しは

<h3><a name="x">◆◆ 見出し ◆◆</a></h3>

となっています。見出しの行のみを抽出し、midasi.txtに入れなさい。

import re

with open("python.html", "r", encodeing = "utf-8") as f:

with open("midasi.txt", "w") as f2:

for line in f:

line = line.rstrip("\n")

m = re.search("<h3><a name=\".+\">◆◆ .+ ◆◆</a></h3>", line)

if m:

print(line, file = f2)

　pythonでは文字列は " あるいは ' で囲みます。" で囲んだ文字列の中に " が入っている場合は、\" と書きます。あるいは ' で囲んで以下のように書いても構いません。

m = re.search('<h3><a name=".+">◆◆ .+ ◆◆</a></h3>', line)

　サーチする文字列として name = "x" の x には1桁か2桁の数字が入っています。これを .+ で表しました。. は任意の1文字で + は一個以上を表すので、.+ は任意の文字が1文字以上あることを意味します。見出しの部分も .+ でマッチングさせています。

　この例は正規表現の使い方を学習するために冗長な書き方をしました。以下のように書いてもマッチします。

m = re.search("◆◆", line)

【例題】

　見出しの部分のみを抽出しmidasi2.txtに書き込みなさい。

m = re.search("<h3><a name=\".+\">◆◆ (.+) ◆◆</a></h3>", line)

if m:

print(m.group(1), file = f2)

( ) で囲むと、その中の文字列がm.group(1) の中に入ります。見出しの部分がm.group(1) に入っているので、それをプリントアウトしています。

　( ) は複数あっても構いません。左から順番に m.group(1), m.group(2) と対応します。

　この例は冗長なので、以下のように書いても同じ結果となります。

m = re.search("◆◆ (.+) ◆◆", line)

【課題5‑1】

　python.txtを読み込み、先頭に #\_ が付く行のみを取りだして、midasi3.txtに書き込むプログラムを作りなさい。python.txtはpython.htmlの元となったファイルです。このファイルをpythonのプログラムを使うことで、python.htmlを自動生成しています。

　自動生成するプログラムmakehtml.pyも参考のために、ファイルのフォルダに入れておきます。

## split関数

　文字列を分解するのがsplit関数です。以下のように使います。

moji = " abc def gh iii "

list = moji.split() # () の中を空欄にしたときは区切り文字（スペース、

# タブ）で分解する

# リストの内容の確認　3通りの方法がある

print(list)

for element in list:

print(element)

for i in range(len(list)):

print(list[i])

　スペースで区切ったときの4個の文字列が、list[0] ～ list[3] に入りました。コンマで区切った文字列の場合は、以下のように書きます。

moji = "abc,def,12,gh,"

list = moji.split(',')

print(list)

　split(',') はコンマで区切ることを示します。Excelなどの表計算ソフトのデータをテキストファイルに変換するとき、CSV (Comma-Separated Values) 形式でセーブします。このCSV形式はセルのデータの区切りとしてコンマを用います。ExcelでCSV形式でセーブするときは注意点があります。「CSV UTF-8」と「CSV」の2通りがあります。CSV UTF-8を選択すると、ファイルの先頭に3バイトのBOM (Byte Order Mark) がつきます。このファイルを読み込むときは

f = open("fname.csv", "r", encoding="utf-8")

のようにencoding="utf-8" が必要です。これを怠るとエラーが発生します。

　splitに対して、区切り文字を与える場合、注意点があります。最初の例に対して、以下のように書くと、どうなるでしょうか？　試してみて下さい。

moji = " abc def gh iii "

list = moji.split(' ')

　区切り文字として空白を指定しているので、同じ結果が得られそうですが、そうではありません。'' という長さのない文字列が多数得られます。その理由は半角空白を , に置き換えると理解できます。

moji = ",,abc,def,,,gh,,iii,,"

　CSVファイルに上記の文字列が入っていると考えると、splitの結果に納得できます。

## ネットワークのアクセス

　webサイトへ接続して、サイトの内容を取得して画面に表示するコードは以下のように書きます。

import requests

url = "http://denki.nara-edu.ac.jp/~yabu/"

result = requests.get(url)

result.encoding = result.apparent\_encoding

html = result.text

list = html.split("\n")

for line in list:

print(line)

　このプログラムはrequestsというモジュールを使っています。モジュールをインストールするには、以下のようにします。

(1) pythonをインストールするとpip.exeというファイルもインストールされる。そのフォルダへパスを通す。具体的な手順は以下の通り。

(1a) 左下のウィンドウアイコン → 歯車マーク → path を検索

(1b) ユーザー環境変数 PATH に pip.exe が存在するフォルダを加える。

筆者の場合

C:\Users\ユーザー名\AppData\Local\Programs\Python\Python39\Scripts

(2) requests をインストールする。具体的な手順は以下の通り。

(2a) コマンドプロンプトを起動する。

(2b) pip install request と打つ

　このサンプルプログラムでは、htmlファイルの内容を表示するだけですが、必要な情報を抽出してファイルに記録するなど、様々な応用が考えられます。

1. if m != None: と書いても構いません。 [↑](#footnote-ref-1)