2022年11月21日(月) 4限
@研究棟A302

プログラミングA2
第8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

本日のお品書き

- 本日：標準ライブラリ
  - webbrowserモジュールによるブラウザ操作
  - urllib.requestモジュールによるHTTPリクエスト
  - reモジュールによる正規表現

【おまけ】webbrowserモジュール

- **open(URL, オプション)：**
  デフォルトブラウザの開いているウィンドウでURLを開く
- **open_new(URL)：**
  デフォルトブラウザの新しいウィンドウでURLを開く
- **open_new_tab(URL)：**
  デフォルトブラウザの開いているウィンドウの新しいタブでURLを開く

web_browse.py

```
import webbrowser
# import time

url = https://www.teu.ac.jp/
webbrowser.open_new_tab(url)
# time.sleep(1)
```

URLのページが開かないときは,
← sleep(1)を入れてみる
or
以下のコマンドで実行する ↓

実行例

```
PS C:¥Users¥admin¥Desktop¥ProA2> python ./lec08/web_browse.py
```

HTTPリクエスト

urllibパッケージ

URLを扱うモジュール群をまとめたパッケージ

- **requestモジュール：**
  URLを開いて読むためのモジュール

- **errorモジュール：**
  requestが発生させる例外が定義されるモジュール

- **parseモジュール：**
  URL文字列を構成要素に分解するためのモジュール

requestモジュール

基本的な認証，暗号化認証，リダイレクション，Cookie，
その他の介在する複雑なアクセス環境においてURLを
開くための関数とクラスを定義している．

- **urlopen(URL)関数：**
  URLを開き，http.client.HTTPResponseオブジェクトを返す

urlopenの使い方

```
from urllib.request import urlopen

res = urlopen(URL)
html = res.read()
headers = res.getheaders()
print(res.status)
```

http.client.HTTPResponse

https://docs.python.org/ja/3.9/library/http.client.html#http.client.HTTPResponse

サーバからのHTTPレスポンスに関するIterableオブジェクトであり，リクエストヘッダとエンティティボディへアクセスすることができる.

[おもなインスタンスメソッド]

- **read()**：レスポンスの本体全体を返す
- **getheaders()**：
  httpヘッダと値のタプルが並んだリストを返す

[おもなインスタンス変数]

- **status**：サーバから返される状態コード

httpヘッダの確認

url_open.py

```
url = "https://www.teu.ac.jp/"
res = urlopen(url)
pprint(res.getheaders())
```

実行例

```
[('Date', 'Sun, 13 Nov 2022 04:11:03 GMT'),
 ('Content-Type', 'text/html;charset=UTF-8'),
 ('Transfer-Encoding', 'chunked'),
 ('Connection', 'close'),
 ('Strict-Transport-Security', 'max-age=31536000;
includeSubDomains'),
 ('X-UA-Compatible', 'IE=edge'),
 ('X-FRAME-OPTIONS', 'SAMEORIGIN'),
 ('Vary', 'User-Agent,Accept-Encoding'),
 :
]
```

レスポンス本体の確認

url_open.py

```
url = "https://www.teu.ac.jp/"
res = urlopen(url)

html = res.read()
# print(type(html)) # bytes
html = html.decode()
# print(type(html)) # str
print(html)
```

← バイトオブジェクト

← バイト列をデコードした文字列を返す
デフォルトでUTF-8

実行例

```
<!DOCTYPE html>
<html lang="ja">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<meta name="author" content="">
 :
<title>東京工科大学</title>
 :
```

練習問題：read_html.py

HTTPResponseオブジェクトはIterableであるため，
for-in文で1行ずつhtmlソースを読み込むことができる．

for-in文で1行ずつ読み込み，改行文字を削除後に空である行を除いてprintするコードを実装せよ．

また，コンテキストマネージャ(with構文)を用いること．

read_html.py

```
url = "https://www.teu.ac.jp/"

with URLをopenする as res:
    for Iterableからイテレータを抽出
        row = row.decode().rstrip()  ← 改行文字削除後に
        if row == "":                ← 空行だったら
            continue
        print(row)
```

正規表現

正規表現：Regular Expression

文字列の集合に対して，共通するパターンにより一つの文字列で表現する方法

- **例：6文字の名前のビールを探すとき**
  - 正規表現では　**[アーン]{3}ビール**　とあらわす
    - サッポロビール　×
    - アサヒビール　○
    - キリンビール　○

reモジュールの使い方

```
import re

text = "対象文字列"
pattern = re.compile(パターン)
pattern.search(text)
```

← 対象文字列の中から
← 正規表現で表現されたパターンに
← マッチする文字列を検索する

文字クラス

文字クラスとは，マッチングの対象となる文字列中に，
[ ] (ブラケット)で囲んだ文字のいずれかをマッチさせるための表現

-	範囲
^	先頭に置くと　“**以外**にマッチ”という意味
[a-z]	小文字の半角英文字
[A-Z]	大文字の半角英文字
[0-9]	数字
[都道府県]	都か道か府か県のどれかの１文字
[^都道府県]	都でも道でも府でも県でもない１文字
[,.]	カンマかピリオドのどちらかの１文字
[¥t¥s]	タブかスペースのどちらかの１文字

エスケープシーケンス

¥w	単語を構成する文字　英数文字かアンダスコア
¥s	空白文字（※半角）
¥d	数字
¥n	改行
¥r	復帰
¥t	タブ
¥W	単語を構成する文字以外
¥S	空白文字以外
¥D	数字以外
¥¥	バックスラッシュ
¥b	バックスペース
¥"	ダブルクオート
¥'	シングルクオート
¥/	スラッシュ

量／位置を指定する

●量を指定する

.	1つの任意文字(A, B, C, ...) (¥nを除く)
*	0回以上の連続する文字
+	1回以上の連続する文字
?	0または1回だけの文字
{n}	n回の連続する文字
{n,}	n回以上の連続する文字
{n,m}	n回以上，m回以下の連続する文字

●位置を指定する

^	文字列の先頭
$	文字列の末尾

その他

- **パターングループ：「( )」（パーレン）**
  - 複数の文字列をまとめた形でパターンマッチを行う
  - グループ化

- **文字クラス：「[ ]」（ブラケット）**
  - マッチングの対象となる文字列中に，[ ]で囲んだ文字のいずれかをマッチさせるため表現

- **パターン論理和：「|」（パイプ）**
  - 複数の文字列をパイプで分割すると，複数の文字列のどれかにマッチする

パターンの書き方　具体例①

正規表現	説明	一致する文字列
A*	0個以上連続したA	, A, AA, AAA
A+	1個以上連続したA	A, AA, AAA
A.	Aの次に何れかの1文字がある場合 改行文字は除く	AB, A1, A.
AB?C	AとCの間にBがないか，Bがある場合	AC, ABC
A{3}	Aが3個	AAA
A{2,4}	2個以上，4個以内のA	AA, AAA, AAAA
[a-z]+	a～zの何れか，つまり小文字アルファベットが1回以上続く	value, ascii
[A-Z]+	A～Zの何れか，つまり大文字アルファベットが1回以上続く	VALUE, ASCII
¥w+	1個以上の英数文字	abc, a001, 001
¥d+	1個以上の数字	1, 12, 123, 001

パターンの書き方　具体例②

- **文字列からURLの検索**
  - `^https?://[^/][¥w¥d/%#$&?()~_.=+-]+`　← 修正しました
- **文字列から日付の検索**
  - `¥d{4}/¥d{1,2}/¥d{1,2}`
- **html文書から<table>タグを検索**
  - `<table.*?>`
  - 解説：
    - 「<」と「>」はそのままの文字
    - 「.」は文字(どんな文字でも)を表す
    - 「.*」は.が0回以上続くことを表す
    - 「?」は最短マッチ

最長マッチと最短マッチ

- 最長マッチ：なるべく長く対応する
- 最短マッチ：なるべく短く対応する
- 「*」は「0回以上のくり返し(最長一致)」
- 「*?」は「0回以上のくり返し(最短一致)」

- 例：下記のhtml文書からタグだけを抽出したい

```
<html>
  <head>
    <title>サンプルページ</title>
  </head>
  <body>こんにちは　このページはサンプルです．</body>
</html>
```

最長マッチ：<.*>　　最短マッチ：<.*?>

正規表現オブジェクト：Pattern

正規表現オブジェクトとは，re.compile(パターン)により生成されるPatternクラスのインスタンスで，対象文字列からパターンにマッチする文字列の検索や抽出を行う以下のインスタンスメソッドを持つ．

- **search(対象文字列)：**
  - 対象文字列からパターンに**マッチする最初の**場所を探し，対応するMatchオブジェクトを返す
- **match(対象文字列)：**
  - 対象文字列の**先頭で0**文字以上がパターンにマッチする場合，Matchオブジェクトを返す
- **fullmatch(対象文字列)：**
  - 対象文字列**全体**がマッチした場合，Matchオブジェクトを返す
- **findall(対象文字列)：**
  - 対象文字列からパターンにマッチした**すべての部分**のリストを返す
- **sub(置換文字列，対象文字列)：**
  - 対象文字列からマッチした部分を置換した文字列を返す

マッチオブジェクト：Match

前述の，search，match，fullmatchメソッドが返すオブジェクトで，
正規表現パターンにマッチした場合の情報を格納するMatchクラスの
インスタンスであり，以下のインスタンスメソッドを持つ．

- **group(idx)：**
  - idx番目のグループの該当文字列を返す
  - idxが0の場合，マッチした全体の文字列を返す
- **groups()：**
  - マッチしたすべてのグループをタプルにして返す
- **__getitem__(idx)：**
  - group(idx)と同じ
- **start(idx)，end(idx)：**
  - idx番目のマッチグループが，対象文字列において何文字目から何文字目までに相当するのかインデックスを返す

例題：poke_regexp1.py

ポケモンの名前から長音「ー」を検索するコードを実装せよ.

poke_regexp1.py

```
titles = read_names("lec08/data/poke_names.txt")
pattern = re.compile("ー")
print(pattern, type(pattern))
for i, title in enumerate(titles, 1):
    print(title, pattern.search(title))
```

← Patternクラスのインスタンス

実行例

```
re.compile('ー') <class 're.Pattern'>
フシギダネ None
フシギソウ None
フシギバナ None
ヒトカゲ None
リザード <re.Match object; span=(2, 3), match='ー'>
リザードン <re.Match object; span=(2, 3), match='ー'>
```

← Patternクラスのインスタンス

← マッチしない場合はNoneが返される

↑ マッチした場合はMatchクラスのインスタンスが返される

ためしてみよう

poke_regexp1.pyのsearchメソッド部分を
以下のメソッドに変更し，挙動を確認してみよう．

- **match：**
  - 先頭に「ー」があるポケモンはいないので，すべてがNone
- **fullmatch：**
  - 全体が「ー」であるポケモンはいないので，すべてがNone
- **findall：**
  - 長音部分を抽出し，リストとして返される
  - 「ホーホー」や「フリーザー」など，「ー」を2つ持つポケモンの場合，要素数2のリストとなる

練習問題：poke_regexp2.py

「ュウ」または「ュー」で終わる名前を検索し，マッチした場合のみ名前とMatchオブジェクトを表示せよ．

poke_regexp2.py

```
titles = read_names("lec08/data/poke_names.txt")
pattern = re.compile(      正規表現      )
for i, title in enumerate(titles, 1):
    match = pattern.search(title)
      マッチしたら
    titleとmatchをprint
```

実行例

```
ピカチュウ <re.Match object; span=(3, 5), match='ュウ'>
ライチュウ <re.Match object; span=(3, 5), match='ュウ'>
ミニリュウ <re.Match object; span=(3, 5), match='ュウ'>
ハクリュー <re.Match object; span=(3, 5), match='ュー'>
カイリュー <re.Match object; span=(3, 5), match='ュー'>
ミュウ <re.Match object; span=(1, 3), match='ュウ'>
ピチュー <re.Match object; span=(2, 4), match='ュー'>
デンリュウ <re.Match object; span=(3, 5), match='ュウ'>
```

練習問題：poke_regexp2.py

以下の正規表現を記述し，正しくマッチできるか確認せよ．

- **「ン」を2回含む名前**

```
pattern = re.compile(          )
```

```
サンドパン <re.Match object; span=(1, 5), match='ンドパン'>
コンパン <re.Match object; span=(1, 4), match='ンパン'>
ランターン <re.Match object; span=(1, 5), match='ンターン'>
ヤンヤンマ <re.Match object; span=(1, 4), match='ンヤン'>
アンノーン <re.Match object; span=(1, 5), match='ンノーン'>
マンタイン <re.Match object; span=(1, 5), match='ンタイン'>
ドンファン <re.Match object; span=(1, 5), match='ンファン'>
```

- **「ラリルレロ」のいずれかを2個以上含む名前**

```
pattern = re.compile(                                  )
```

```
トランセル <re.Match object; span=(1, 5), match='ランセル'>
オニドリル <re.Match object; span=(3, 5), match='リル'>
ラフレシア <re.Match object; span=(0, 3), match='ラフレ'>
オコリザル <re.Match object; span=(2, 5), match='リザル'>
レアコイル <re.Match object; span=(0, 5), match='レアコイル'>
ビリリダマ <re.Match object; span=(1, 3), match='リリ'>
カラカラ <re.Match object; span=(1, 4), match='ラカラ'>
：
デルビル <re.Match object; span=(1, 4), match='ルビル'>
```

↑
最後に「.*」を付けてみると
matchがどう変わる？

練習問題：poke_regexp2.py

以下の正規表現を記述し，正しくマッチできるか確認せよ．

- **3文字の名前**

```
pattern = re.compile(          )
```

または

```

```

```
ポッポ <re.Match object; span=(0, 3), match='ポッポ'>
ラッタ <re.Match object; span=(0, 3), match='ラッタ'>
アーボ <re.Match object; span=(0, 3), match='アーボ'>
：
ブビィ <re.Match object; span=(0, 3), match='ブビィ'>
ルギア <re.Match object; span=(0, 3), match='ルギア'>
```

- **小さな「ッ」と任意の1文字で終わる名前**

```
pattern = re.compile(          )
```

```
ポッポ <re.Match object; span=(1, 3), match='ッポ'>
ピジョット <re.Match object; span=(3, 5), match='ット'>
コラッタ <re.Match object; span=(2, 4), match='ッタ'>
ラッタ <re.Match object; span=(1, 3), match='ッタ'>
アーボック <re.Match object; span=(3, 5), match='ック'>
：
エレキッド <re.Match object; span=(3, 5), match='ッド'>
```

難しい正規表現の例

- 同じ文字が連続する名前

```
pattern = re.compile("(.)¥¥1")
```

- ¥1：()によりグループ化されたパターンの
  1つ目のグループを表している
- ¥¥：¥は，¥をエスケープしている
- (.)：任意の1文字をグループ化している

```
pattern = re.compile(r"(.)¥1")
```

- r"文字列"：raw文字列は，文字列内の
  エスケープシーケンスを無効化する
  ＝¥なしで，¥を表現している

```
ドククラゲ <re.Match object; span=(1, 3), match='ククラ'>
ビリリダマ <re.Match object; span=(1, 3), match='リリ'>
オオタチ <re.Match object; span=(0, 2), match='オオ'>
ププリン <re.Match object; span=(0, 2), match='ププ'>
モココ <re.Match object; span=(1, 3), match='ココ'>
ポポッコ <re.Match object; span=(0, 2), match='ポポ'>
オドシシ <re.Match object; span=(2, 4), match='シシ'>
```

例題：html_regexp1.py

index.htmlから，文字列をすべて検索し，URL部分と文字列部分のみ表示せよ．

html_regexp1.py

```
html = read_html("lec08/data/index.html")
pattern = re.compile("<a href=¥"(.*?)¥".*?>(.*?)</a>")
match_lst = pattern.findall(html)
for match in match_lst:
    print(match)
```

↑ ↑
2つのグループ

実行例

```
('/english/index.html', 'EN')
('/gaiyou/006493.html', 'お問い合わせ')
('/campus/access/006644.html', '交通案内')
('/010816.html', 'サイトマップ')
 :
```

← マッチしたサブグループを要素としたタプル

のリスト

練習問題：html_regexp1.py

前ページのURLのうち，大学学外へのもののみprintするように変更せよ．

html_regexp1.py

```
html = read_html("lec08/data/index.html")
pattern = re.compile(                    )
match_lst = pattern.findall(html)
for match in match_lst:
    print(match)
```

実行例

```
：
('http://twitter.com/tut_tweet', '<span>twitter</span>')
('http://www.youtube.com/user/TokyoKokaDaigaku',
'<span>YouTube</span>')
('http://line.naver.jp/ti/p/%40koukadai', '<span>LINE</span>')
('https://page.line.me/koukadai', '<span>LINE</span>')
('https://www.instagram.com/tut_koukaton.official/',
'<span>instagram</span>')
```

練習問題：html_regexp2.py

<!--と-->で囲まれたコメントアウト部分を削除するコードを実装せよ.

ただし，コメントアウトは複数行にまたがることが多いことに注意せよ.

html_regexp2.py

```
html = read_html("lec08/data/index.html").
pattern = re.compile(                    )
match_lst = pattern.findall(html)
for match in match_lst:
    print(match)

html = pattern.
```

改行文字の削除

← 正規表現が
← 正しいか
← 確認

削除