D04 正規表達式

重要知識點

- 延續上一章節學習正規表達式
- 如何使用Python操作正規表達式

Python re 模組來進行正規表達式配對

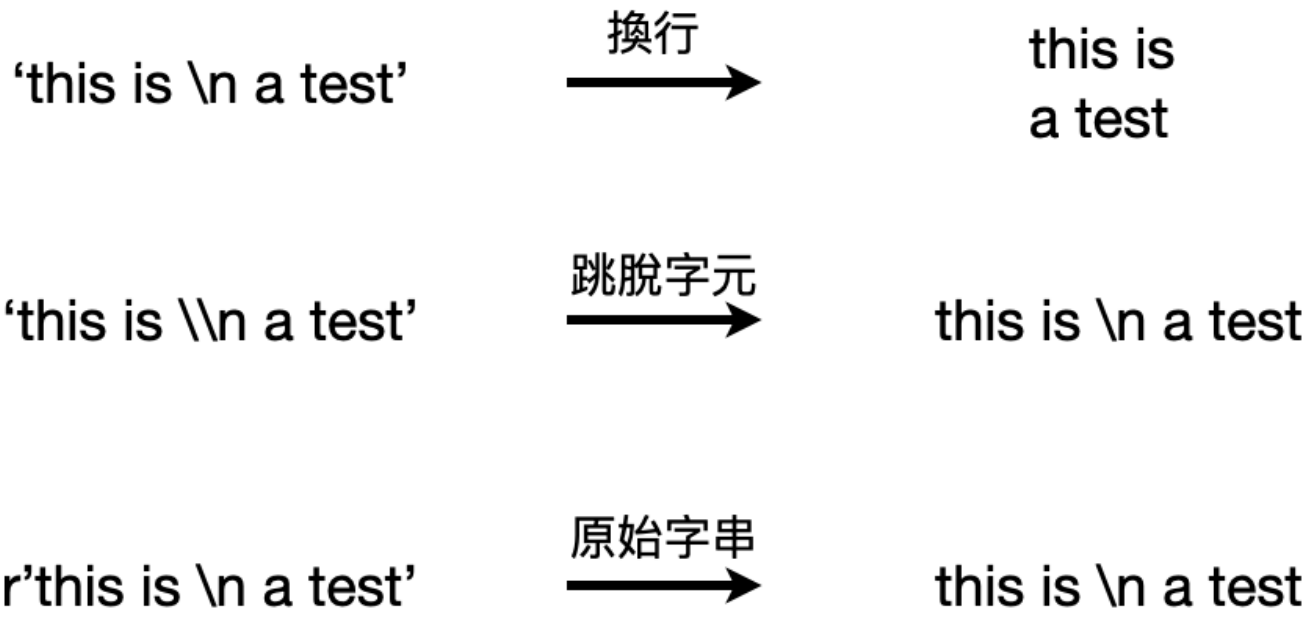
在先前課程中所學的 Regular Expression(正規表達式)，是可以匹配文字片段的一種模式。
要使用 Regex 來處理字串，我們必須要使用支援 Regex 的工具。
而在 Python 中的 re 模組就是專們用來支援處理 Regex 的模組工具。

Python 字串前綴

在正規表達式中，我們常會使用到反斜線(\)來表達一些特殊配對，像是

- \w：配對任何數字字母底線
- \b：字詞字元邊界
- \s：任何空白字元

在 Python 中，反斜線(\) 會被當作是特殊符號得跳脫字元，像是 "\n" 代表換行，加上跳脫字元後 "\\n" 即代表一般 "\n" 符號。
為了避免字串中出現過多的反斜線，進而導致難以維護正規表達式的配對，因此我們可以使用原始字串前綴`r"str"`。 因此`r"\n" == "\\n"`。
因此為了避免出現特殊符號問題，習慣上我們會將正規表達式的模式對象加上字串前綴。
(ex: `r"\W\.\D"`)



建立模式對象(Pattern Object)

re.compile(pattern, flags=0)：

將正規表達式轉為 pattern object 的模式對象。以此方法將其保存下來供後續之用。(但其實這樣的做法是非必要的，詳情請參考延伸閱讀一)

```
###使用前章節的電子郵件配對為例###

#導入re模組
import re

#欲配對文本
txt = "SaveTheWorld@hotmail.com \n foobar@gmail.com"

#建立模式對象
pattern_obj = re.compile(pattern=r"(.*)@(?!gmail)\w+\.com")
#進行配對(請注意這裡是使用pattern.search配對)
x1 = pattern_obj.search(txt) #先別擔心re.research的作用(後續會說明)
print(x1.group())
```

模式對象配對 → SaveTheWorld@hotmail.com

```
#導入re模組
import re
#欲配對文本
txt = "SaveTheWorld@hotmail.com \n foobar@gmail.com"

#建立模式對象
pattern_obj = re.compile(pattern=r"(.*)@(?!gmail)\w+\.com")
#進行配對(請注意這裡是使用pattern.search配對)
x1 = pattern_obj.search(txt) #先別擔心re.research的作用(後續會說明)
print(x1.group())
```

```python
#不使用模式對象
#我們也可以不建立模式對象,直接使用正規表達式配對

#遇配對文本
txt = "SaveTheWorld@hotmail.com \n foobar@gmail.com"
pattern = r"(.*)@(?!gmail)\w+\.com" #這裡使用原始字串作為配對
#進行配對(請注意這裡是使用re.search配對)
x2 = re.search(pattern, txt)
print(x2.group())
```


```python
#配對文本
txt = "SaveTheWorld@hotmail.com \n foobar@gmail.com"
pattern = r"(.*)@(?!gmail)\w+\.com" #這裡使用原始字串作為配對
#進行配對(請注意這裡是使用re.search配對)
x2 = re.search(pattern, txt)
print(x2.group())
```

re.search(pattern, string, flags=0

掃描字符串，查詢匹配正規表達式模式的位置，返回 MatchObject 的物件實例。若沒有可匹配的位置，則返回 None。若有多個可配對位置，只有第一個配對成功的會返回。

```python
###使用前章節的電子郵件配對為例###

#欲配對文本
txt = "SaveTheWorld@hotmail.com \n foobar@gmail.com \n zzzGroup@yahoo.com"
pattern = r".*@(?!gmail)\w+\.com" #這裡使用原始字串作為配對

#進行配對
match = re.search(pattern, txt)
```

```
###使用前章節的電子郵件配對為例###

#欲配對文本
txt = "SaveTheWorld@hotmail.com \n foobar@gmail.com \n zzzGroup@yahoo.com"
pattern = r".*@(?!gmail)\w+\.com" #這裡使用原始字串作為配對

#進行配對
match = re.search(pattern, txt)
print(type(match)) #顯示為re.Match 物件實例
print(match)

print('\n----分隔線----')
```

```
#若無可滿足配對, re.search會返回None
txt = "foobar@gmail.com" #這裡只保留不滿足配對的email
pattern = r".*@(?!gmail)\w+\.com"
match = re.search(pattern, txt)
print(match)
```

配對 → None

```
#若無可滿足配對, re.search會返回None
txt = "foobar@gmail.com" #這裡只保留不滿足配對的email
pattern = r".*@(?!gmail)\w+\.com"
match = re.search(pattern, txt)
print(match)
```

從字串開始的位置進行配對，只會配對字串開頭，若配對成功則返回 Match 的物件實例。若失敗則返回 None。

```
###使用前章節的電子郵件配對為例###

#欲配對文本
txt = "SaveTheWorld@hotmail.com \n foobar@gmail.com \n zzzGroup@yahoo.com"
pattern = r".*@(?!gmail)\w+\.com" #這裡使用原始字串作為配對

#進行配對
match = re.match(pattern, txt)
```

配對 → SaveTheWorld@hotmail.com

```
###使用前章節的電子郵件配對為例###

#欲配對文本
txt = "SaveTheWorld@hotmail.com \n foobar@gmail.com \n zzzGroup@yahoo.com"
pattern = r".*@(?!gmail)\w+\.com" #這裡使用原始字串作為配對

#進行配對
match = re.match(pattern, txt)
print(type(match)) #顯示為re.Match 物件實例
print(match)

print('\n----分隔線----')
```

```
#若開頭無法配對成功，即返回None
txt = "foobar@gmail.com \n SaveTheWorld@hotmail.com \n zzzGroup@yahoo.com"
pattern = r".*@(?!gmail)\w+\.com"
match = re.match(pattern, txt)
print(match)
```

配對 → None

```
#若開頭無法配對成功，即返回None
txt = "foobar@gmail.com \n SaveTheWorld@hotmail.com \n zzzGroup@yahoo.com"
pattern = r".*@(?!gmail)\w+\.com"
match = re.match(pattern, txt)
print(match)
```

掃描字符串，找到正規表達式所配對的所有子串，並組成一個列表(list)返回。若沒有配對成功，則返回空列表(list)。

```
###使用前章節的電子郵件配對為例###

#欲配對文本
txt = "SaveTheWorld@hotmail.com \n foobar@gmail.com \n zzzGroup@yahoo.com"
pattern = r".*@(?!gmail)\w+\.com" #這裡使用原始字串作為配對

#進行配對
match = re.findall(pattern, txt)
```

配對 → [SaveTheWorld@hotmail.com，zzzGroup@yahoo.com]

```
###使用前章節的電子郵件配對為例###
#欲配對文本
txt = "SaveTheWorld@hotmail.com \n foobar@gmail.com \n zzzGroup@yahoo.com"
pattern = r".*@(?!gmail)\w+\.com" #這裡使用原始字串作為配對

#進行配對
match = re.findall(pattern, txt)
print(type(match)) #list 物件實力
print(match)
```

```
###使用前章節的電子郵件配對為例###

#欲配對文本
txt = "SaveTheWorld@hotmail.com \n foobar@gmail.com \n zzzGroup@yahoo.com"
pattern = r"(.*)@(?!gmail)(\w+)\.com" #將.* 改為 group的形式(.*), 且 \w+ 改為 (\w+)

#進行配對
match = re.findall(pattern, txt)
```

配對 → [('SaveTheWorld', 'hotmail'), (' zzzGroup', 'yahoo')]

```
###使用前章節的電子郵件配對為例###

#欲配對文本
txt = "SaveTheWorld@hotmail.com \n foobar@gmail.com \n zzzGroup@yahoo.com"
pattern = r"(.*)@(?!gmail)(\w+)\.com" #將.* 改為 group的形式(.*), 且 \w+ 改為

#進行配對
match = re.findall(pattern, txt)
print(type(match))
print(match)

#可以發現返回的list變成符合配對的email裡面的group字串
```

和 findall 類似，在字符串中找尋正規表達式可以匹配的所有子字串，並返回一個迭代器(iterator)。

```
###使用前章節的電子郵件配對為例###

#欲配對文本
txt = "SaveTheWorld@hotmail.com \n foobar@gmail.com \n zzzGroup@yahoo.com"
pattern = r".*@(?!gmail)\w+\.com" #這裡使用原始字串作為配對

#進行配對
match = re.finditer(pattern, txt)
print(type(match)) #iterator 物件實例
```

打印 → <class 'callable_iterator'>

```
###使用前章節的電子郵件配對為例###

#欲配對文本
txt = "SaveTheWorld@hotmail.com \n foobar@gmail.com \n zzzGroup@yahoo.com"
pattern = r".*@(?!gmail)\w+\.com" #這裡使用原始字串作為配對

#進行配對
match = re.finditer(pattern, txt)
print(type(match)) #iterator 物件實例
print('\n----分隔線----')
```

```
for ma in match:
    print(ma)
    print(f'Match start: {ma.start()}; Match end: {ma.end()}') #使用.start(), .end()返回配對的起點與終點
    print(f'Match text: {ma.group()}') #使用.group() or .group(0)返回配對的字串
```

打印

```
(0, 24)
SaveTheWorld@hotmail.com
(45, 64)
zzzGroup@yahoo.com
```

```
for ma in match:
    print(ma)
    print(f'Match start: {ma.start()}; Match end: {ma.end()}') #使用.start()
    print(f'Match text: {ma.group()}') #使用.group() or .group(0)返回配對的字串
    print('\n----分隔線----')
```

re.sub(pattern, repl, string, count=0, flags=0)

在字符串中找到正規表達式匹配的子字串，使用另外一個字串 repl 替換匹配的字串。若沒有可匹配的字串，即返回未被修改的原始字串。
count 變數可以用來指定要替代的次數，如果 count 是 0(預設值)，所有成功配對的都修改。

```
###使用前章節的電子郵件配對為例###

#欲配對文本
txt = "SaveTheWorld@hotmail.com \n foobar@gmail.com \n zzzGroup@yahoo.com"
pattern = r".*@(?!gmail)\w+\.com" #這裡使用原始字串作為配對

#進行配對
match = re.sub(pattern, 'REPLACE', txt, count=0)
match #配對到的email都修改為REPLACE
```

打印

```
'REPLACE \n foobar@gmail.com \nREPLACE'
```

```
###使用前章節的電子郵件配對為例###

#欲配對文本
txt = "SaveTheWorld@hotmail.com \n foobar@gmail.com \n zzzGroup@yahoo.com"
pattern = r".*@(?!gmail)\w+\.com" #這裡使用原始字串作為配對

#進行配對
match = re.sub(pattern, 'REPLACE', txt, count=0)
match #配對到的email都修改為REPLACE
```

```
###使用前章節的電子郵件配對為例###

#欲配對文本
txt = "SaveTheWorld@hotmail.com \n foobar@gmail.com \n zzzGroup@yahoo.com"
pattern = r".*@(?!gmail)\w+\.com" #這裡使用原始字串作為配對

#進行配對
match = re.sub(pattern, 'REPLACE', txt, count=1) #將count設為1
match #只有一個配對到的修改為REPLACE
```

打印 → 'REPLACE \n foobar@gmail.com \n zzzGroup@yahoo.com'


```
###使用前章節的電子郵件配對為例###

#欲配對文本
txt = "SaveTheWorld@hotmail.com \n foobar@gmail.com \n zzzGroup@yahoo.com"
pattern = r".*@(?!gmail)\w+\.com" #這裡使用原始字串作為配對

#進行配對
match = re.sub(pattern, 'REPLACE', txt, count=1) #將count設為1
match #只有一個配對到的修改為REPLACE
```

功能與`re.sub()`基本上相同，但在返回值時會同時返回新的字符串與替換次數。

```
###使用前章節的電子郵件配對為例###

#欲配對文本
txt = "SaveTheWorld@hotmail.com \n foobar@gmail.com \n zzzGroup@yahoo.com"
pattern = r".*@(?!gmail)\w+\.com" #這裡使用原始字串作為配對

#進行配對
match = re.subn(pattern, 'REPLACE', txt, count=0)
match #可以發現一共配對替換2次
```

打印 → ('REPLACE \n foobar@gmail.com \nREPLACE', 2)

```
###使用前章節的電子郵件配對為例###

#欲配對文本
txt = "SaveTheWorld@hotmail.com \n foobar@gmail.com \n zzzGroup@yahoo.com"
pattern = r".*@(?!gmail)\w+\.com" #這裡使用原始字串作為配對

#進行配對
match = re.subn(pattern, 'REPLACE', txt, count=0)
match #可以發現一共配對替換2次
```

```
###使用前章節的電子郵件配對為例###

#欲配對文本
txt = "SaveTheWorld@hotmail.com \n foobar@gmail.com \n zzzGroup@yahoo.com"
pattern = r".*@(?!gmail)\w+\.com" #這裡使用原始字串作為配對

#進行配對
match = re.subn(pattern, 'REPLACE', txt, count=1) #將count設為1
match #只配對替換1次
```

打印 → ('REPLACE \n foobar@gmail.com \n zzzGroup@yahoo.com', 1)


```
###使用前章節的電子郵件配對為例###

#欲配對文本
txt = "SaveTheWorld@hotmail.com \n foobar@gmail.com \n zzzGroup@yahoo.com"
pattern = r".*@(?!gmail)\w+\.com" #這裡使用原始字串作為配對

#進行配對
match = re.subn(pattern, 'REPLACE', txt, count=1) #將count設為1
match #只配對替換1次
```


re.split(pattern, string, maxsplit=0, flags=0)

利用正規表達式將成功配對的字串部分分割為一個列表，並返回分割後的列表。
其中 maxsplit 是用來指定最多切割多少份，若是 0(預設值)，則所有配對成功的都會進行切割。

```
###使用前章節的電子郵件配對為例###

#欲配對文本
txt = "SaveTheWorld@hotmail.com \n foobar@gmail.com \n zzzGroup@yahoo.com"
pattern = r"\n" #這裡改為配對換行符號

#進行配對
match = re.split(pattern, txt)
match
```

打印 → ['SaveTheWorld@hotmail.com ', ' foobar@gmail.com ', ' zzzGroup@yahoo.com']

```
###使用前章節的電子郵件配對為例###

#欲配對文本
txt = "SaveTheWorld@hotmail.com \n foobar@gmail.com \n zzzGroup@yahoo.com"
pattern = r"\n" #這裡改為配對換行符號

#進行配對
match = re.split(pattern, txt)
match
```

```
###使用前章節的電子郵件配對為例###

#欲配對文本
txt = "SaveTheWorld@hotmail.com \n foobar@gmail.com \n zzzGroup@yahoo.com"
pattern = r"\n" #這裡改為配對換行符號

#進行配對
match = re.split(pattern, txt, maxsplit=1) #設定最多只配對分割一組
match
```

打印 → ['SaveTheWorld@hotmail.com ', ' foobar@gmail.com \n zzzGroup@yahoo.com']

```
###使用前章節的電子郵件配對為例###

#欲配對文本
txt = "SaveTheWorld@hotmail.com \n foobar@gmail.com \n zzzGroup@yahoo.com"
pattern = r"\n" #這裡改為配對換行符號

#進行配對
match = re.split(pattern, txt, maxsplit=1) #設定最多只配對分割一組
match
```

flags 參數

此參數可以控制匹配模式，大部分的匹配模式都可以直接使用正規表達式的規則寫出，但此參數提供我們更方便的方法來控制匹配模式

例如：

- **re.I (re.IGNORECASE)：忽略大小寫模式**
- **re.M (re.MULTILINE)：多行模式**
- **re.S (re.DOTALL)：讓`.`可以匹配所有的字元 (原本`.`無法匹配換行字元)**

可以使用`|`來結合多種模式

flags 參數：re.I (re.IGNORECASE)

#可以發現無法配對大寫的L

```
#欲配對文本
txt = "Leo123 \nkevin456 \n"
pattern = r"[a-z]+" #配對所有小寫a-z字符

#進行配對_1
match = re.findall(pattern, txt) #使用預設的一般配對模式
```

打印 → ['eo', 'kevin']

```
###re.IGNORECASE###

#欲配對文本
txt = "Leo123 \nkevin456 \n"
pattern = r"[a-z]+" #配對所有小寫a-z字符

#進行配對_1
match = re.findall(pattern, txt) #使用預設的一般配對模式
print(type(match))
print(match)
```

#可以發現再加上 re.I 後, 可以忽略大小寫的配對

```
#進行配對_2
match2 = re.findall(pattern, txt, flags=re.I)
```

打印 → ['Leo', 'kevin']

```
#進行配對_2
match2 = re.findall(pattern, txt, flags=re.I)
print(type(match))
print(match2)
```

flags 參數：re.M (re.MULTILINE)

#可以發現只配對到 Leo (因為在一般配對模式下，文本被視為一個含有 \n 的長字串)

```
3  #欲配對文本
4  txt = "Leo123 \nkevin456 \n"
5  pattern = r"^[a-zA-Z]+" #配對所有開頭是a-z或是A-Z的字元
6
7  #進行配對_1
8  match = re.findall(pattern, txt) #使用預設的一般配對模式
```

打印 → ['Leo']

```
1  #欲配對文本
2  txt = "Leo123 \nkevin456 \n"
3  pattern = r"^[a-zA-Z]+" #配對所有開頭是a-z或是A-Z的字元
4
5  #進行配對_1
6  match = re.findall(pattern, txt) #使用預設的一般配對模式
7  print(type(match))
8  print(match)
```

#可以發現加上 re.M 後，可以配對到 Leo，Kevin (因為在 \n 換行符號後會視為新的字串來配對)

```
15 #進行配對_2
16 match2 = re.findall(pattern, txt, flags=re.M) #使用多行配對模式
```

打印 → ['Leo', 'kevin']

```
1  #進行配對_2
2  match2 = re.findall(pattern, txt, flags=re.M) #使用多行配對模式
3  print(type(match))
4  print(match2)
```

flags 參數：re.S (re.DOTALL)

#配對的內容不包含 \n 字串

```
#欲配對文本
txt = "Leo123 \nkevin456 \n"
pattern = r".+" #配對所有開頭是a-z或是A-Z的字元

#進行配對_1
match = re.findall(pattern, txt) #使用預設的一般配對模式
```

打印 → ['Leo123 ', 'kevin456 ']

```
#欲配對文本
txt = "Leo123 \nkevin456 \n"
pattern = r".+" #配對所有開頭是a-z或是A-Z的字元

#進行配對_1
match = re.findall(pattern, txt) #使用預設的一般配對模式
print(type(match))
print(match)
```

#這樣配對也包含了 \n 換行符號

```
#進行配對_2
match2 = re.findall(pattern, txt, flags=re.S) #使用DOTALL配對模式
```

打印 → ['Leo123 \nkevin456 \n']

```
#進行配對_2
match2 = re.findall(pattern, txt, flags=re.S) #使用DOTALL配對模式
print(type(match))
print(match2)
```

flags 參數：結合多種配對模式 (re.I|re.M)

#一般模式下，找不到可配對字串

```
#欲配對文本
txt = "Leo123 \nkevin456 \n"
pattern = r"^[a-z]+" #配對所有開頭是a-z

#進行配對_1
match = re.findall(pattern, txt) #使用預設的一般配對模式
```

打印 → []

```
#欲配對文本
txt = "Leo123 \nkevin456 \n"
pattern = r"^[a-z]+" #配對所有開頭是a-z

#進行配對_1
match = re.findall(pattern, txt) #使用預設的一般配對模式
print(type(match))
print(match)
```

#可以發現加上 re.I|re.M 後，可以配對到 Leo，kevin

```
#進行配對_2
match2 = re.findall(pattern, txt, flags=re.I|re.M) #使用多行配對模式
```

打印 → ['Leo', 'kevin']

```
#進行配對_2
match2 = re.findall(pattern, txt, flags=re.I|re.M) #使用多行配對模式
print(type(match))
print(match2)
```

語法對照表

函式	說明	參數
re.search(pattern, str, flags=0)	掃描整個字符串並返回第一個成功的配對	pattern: 正規表達式 str: 匹配的字串 flags: 標誌位
re.match(pattern, str, flags=0)	從字串開始的位置進行配對，只會配對字串開頭	pattern: 正規表達式 str: 匹配的字串 flags: 標誌位
re.findall(pattern, str, flags=0)	找到正規表達式所配對的**所有**字串	pattern: 正規表達式 str: 匹配的字串 flags: 標誌位
re.finditer(pattern, str, flags=0)	在字符串中找尋正規表達式可以匹配的所有子字串，並返回一個迭代器	pattern: 正規表達式 str: 匹配的字串 flags: 標誌位
re.sub(pattern, repl, str, count=0, flags=0)	在字符串中匹配的子字串，使用另外一個字串repl替換匹配的字串	pattern: 正規表達式 repl: 替換字串 str: 匹配的字串 count: 替換最大次數
re.subn(pattern, str, maxsplit=0, flags=0)	分割匹配字串並返回列表	pattern: 正規表達式 str: 匹配的字串 maxsplit: 分割次數 flags: 標誌位

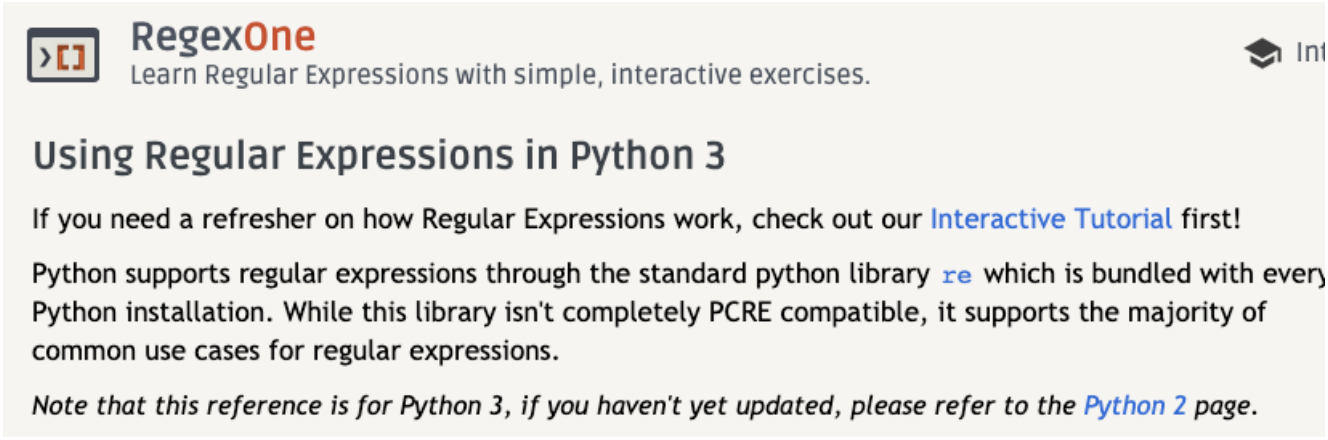
知識點回顧

在這章節我們學習到了

- 如何使用 Python 操作正規表達式配對文字
- Python 的 re 模組使用方式

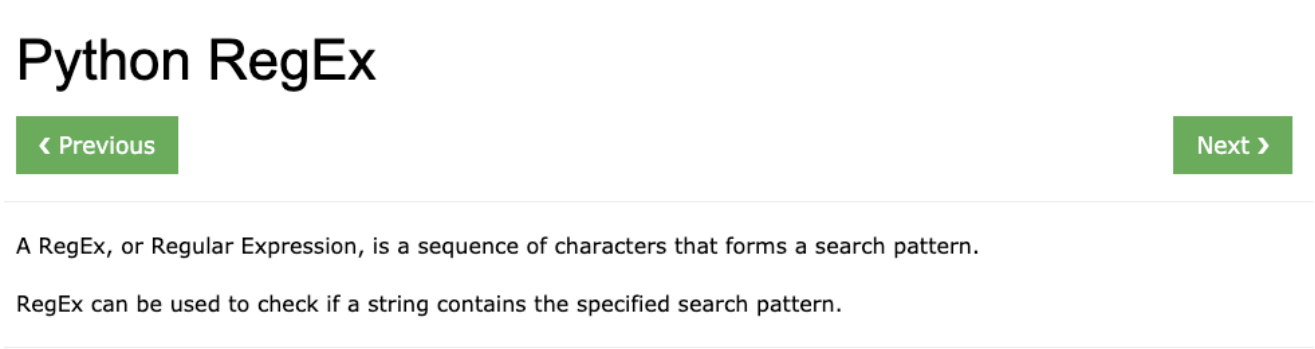
延伸閱讀

網站：[RegexOne]
本網站提供其他詳細的 Python re 模組介紹，可供讀者參考閱讀



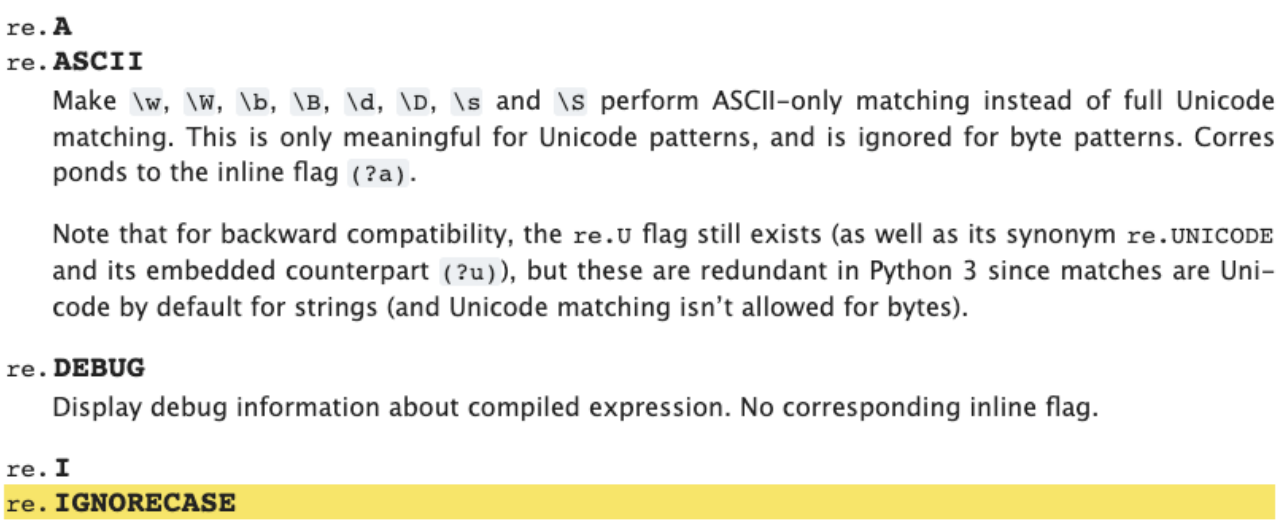
網站：[W3School]

本網站提供基本的 Python re 模組講解與範例，可供讀者查閱



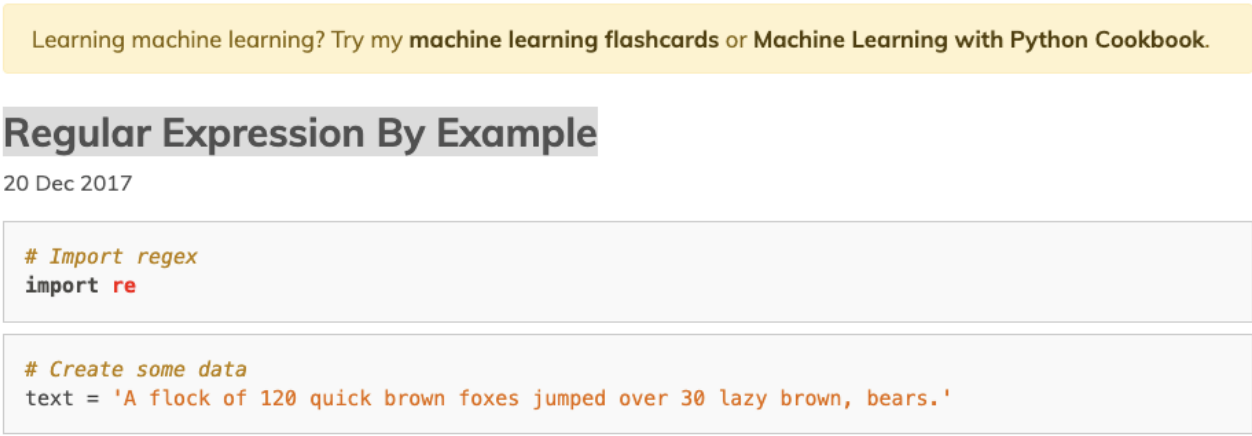
網站：[Python re 模組官網]

這個連結提供更多的可選用 flags 參數，讓有興趣的讀者有更多的參考選擇



網站：[Regular Expression By Example]

這個連結提供一些 Python regular expression 的參考練習，有興趣的讀者可以實際練習操作



Learning machine learning? Try my **machine learning flashcards** or **Machine Learning with Python Cookbook**.

Regular Expression By Example

20 Dec 2017

```
# Import regex
import re
```

```
# Create some data
text = 'A flock of 120 quick brown foxes jumped over 30 lazy brown, bears.'
```

下一步：閱讀範例與完成作業