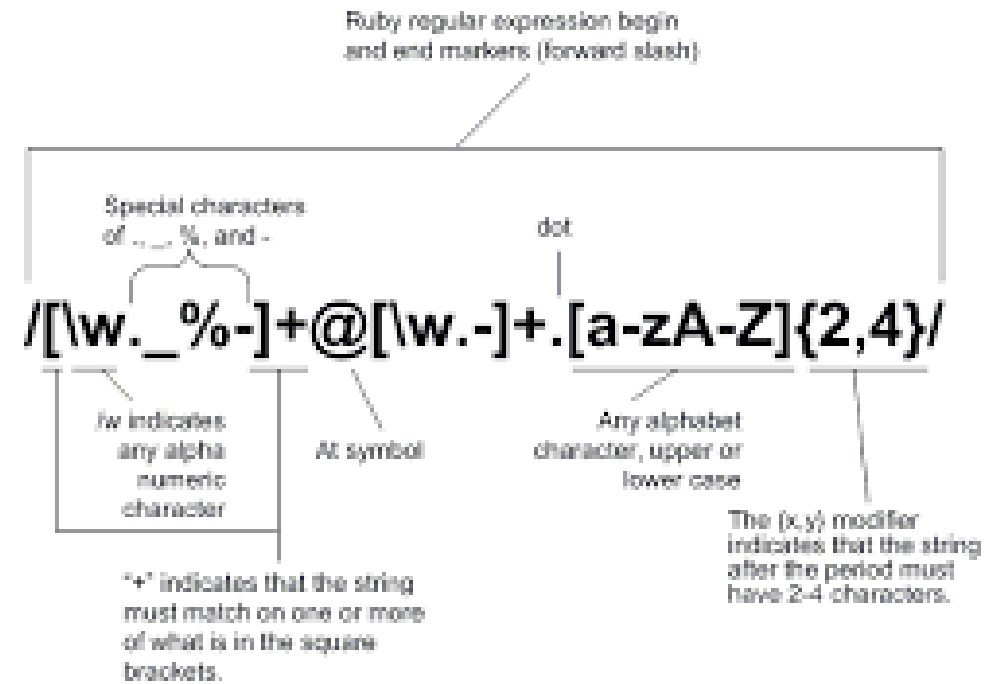


# Week 6-正規表示式

# 正規表示式

# 正規表示式 (Regular Expression, Regex)

正規表示式是用來找出符合某一種模式的字串，對初次接觸的人來說，看到宛如天書的模式字串，覺得困惑是很正常的事。這些看似令人不解的字串其實並不是如此的高深莫測。一旦掌握了模式字串的意義便可以輕鬆的處理各式的文本檔案，原本處理起來相當棘手的字串搜尋，取代的工作也迎刃而解。正規表示式是進階高階自動化程式設計的鑰匙。你一定要了解並熟練的掌握這項強大的工具。



username @ domain . qualifier (com/net/tv/...)

## / 簡單例子

首先解釋何謂模式字串，舉例來說，我們每個人都有一組身分證號碼。第一位是大寫英文數字，接下來是九位數字。這就是身分證號碼的模式字串。在正規表示式便可以用[A-Z]\d{9}來表示。中括號表示了一個字元的長度，字元必須為大寫字母A-Z其中一個。\\d代表了一個字元長度的數字，後面的{9}代表了前面的字元，這裡也就是數字。要連續出現九遍。我們用這個模式字串對整個文檔做搜尋便可以將所有的身分證字號給找出來。

## Regular expression cheatsheet

### Special characters

|                      |   |
|----------------------|---|
| <code>\</code>       | escape special characters                                   |
| <code>.</code>       | matches any character                                       |
| <code>^</code>       | matches beginning of string                                 |
| <code>\$</code>      | matches end of string                                       |
| <code>[5b-d]</code>  | matches any chars '5', 'b', 'c' or 'd'                      |
| <code>[^a-c6]</code> | matches any char except 'a', 'b', 'c' or '6'                |
| <code>R S</code>     | matches either regex <code>R</code> or regex <code>S</code> |
| <code>()</code>      | creates a capture group and indicates precedence            |

### Special sequences

|                           |  |
|---------------------------|--|
| <code>\A</code>           | start of string  |
| <code>\b</code>           | matches empty string at word boundary (between <code>\w</code> and <code>\W</code> ) |
| <code>\B</code>           | matches empty string not at word boundary  |
| <code>\d</code>           | digit  |
| <code>\D</code>           | non-digit  |
| <code>\s</code>           | whitespace: <code>[\t\n\r\f\v]</code>  |
| <code>\S</code>           | non-whitespace   |
| <code>\w</code>           | alphanumeric: <code>[0-9a-zA-Z_]</code>  |
| <code>\W</code>           | non-alphanumeric   |
| <code>\Z</code>           | end of string  |
| <code>\g&lt;id&gt;</code> | matches a previously defined group   |

### Quantifiers


|                      |  |
|----------------------|--|
| <code>*</code>       | 0 or more (append <code>?</code> for non-greedy)   |
| <code>+</code>       | 1 or more (append <code>?</code> for non-greedy)   |
| <code>?</code>       | 0 or 1 (append <code>?</code> for non-greedy)  |
| <code>{m}</code>     | exactly <code>m</code> occurrences   |
| <code>{m, n}</code>  | from <code>m</code> to <code>n</code> . <code>m</code> defaults to 0, <code>n</code> to infinity |
| <code>{m, n}?</code> | from <code>m</code> to <code>n</code> , as few as possible                                       |

### Special sequences

|                             |   |
|-----------------------------|---|
| <code>(?iLmsux)</code>      | matches empty string, sets re.X flags           |
| <code>(?:...)</code>        | non-capturing version of regular parentheses    |
| <code>(?P...)</code>        | matches whatever matched previously named group |
| <code>(?P=)</code>          | digit   |
| <code>(?#...)</code>        | a comment; ignored                              |
| <code>(?=...)</code>        | lookahead assertion: matches without consuming  |
| <code>(?!...)</code>        | negative lookahead assertion                    |
| <code>(?&lt;=...)</code>    | lookbehind assertion: matches if preceded       |
| <code>(?&lt;!=...)</code>   | negative lookbehind assertion                   |
| <code>(? (id)yes no)</code> | match 'yes' if group 'id' matched, else 'no'    |

Based on [tartley's python-regex-cheatsheet](#).

# 基本練習：<https://regexone.com/>

 **RegexOne**  
Learn Regular Expressions with simple, interactive exercises.

Interactive Tutorial

References & More

## Lesson 1: An Introduction, and the ABCs

**Regular expressions** are extremely useful in extracting information from text such as code, log files, spreadsheets, or even documents. And while there is a lot of theory behind formal languages, the following lessons and examples will explore the more practical uses of regular expressions so that you can use them as quickly as possible.

The first thing to recognize when using regular expressions is that **everything is essentially a character**, and we are writing patterns to match a specific sequence of characters (also known as a string). Most patterns use normal ASCII, which includes letters, digits, punctuation and other symbols on your keyboard like `%$@!.`, but unicode characters can also be used to match any type of international text.

Below are a couple lines of text, notice how the text changes to highlight the matching characters on each line as you type in the input field below. To continue to the next lesson, you will need to use the new syntax and concept introduced in each lesson to write a pattern that matches all the lines provided.

Go ahead and try writing a pattern that matches all three rows, **it may be as simple as the common letters on each line.**

Exercise 1: Matching Characters

| Task  | Text    |
|-------|---------|
| Match | abcdefg |
| Match | abcde   |
| Match | abc     |

Type your pattern

Continue >

Solve the above task to continue on to the next problem, or read the [Solution](#).

### Lesson Notes

|                        |                                |
|------------------------|--------------------------------|
| <code>abc...</code>    | Letters                        |
| <code>123...</code>    | Digits                         |
| <code>\d</code>        | Any Digit                      |
| <code>\D</code>        | Any Non-digit character        |
| <code>.</code>         | Any Character                  |
| <code>\.</code>        | Period                         |
| <code>[abc]</code>     | Only a, b, or c                |
| <code>[^abc]</code>    | Not a, b, nor c                |
| <code>[a-z]</code>     | Characters a to z              |
| <code>[0-9]</code>     | Numbers 0 to 9                 |
| <code>\w</code>        | Any Alphanumeric character     |
| <code>\W</code>        | Any Non-alphanumeric character |
| <code>{m}</code>       | m Repetitions                  |
| <code>{m,n}</code>     | m to n Repetitions             |
| <code>*</code>         | Zero or more repetitions       |
| <code>+</code>         | One or more repetitions        |
| <code>?</code>         | Optional character             |
| <code>\s</code>        | Any Whitespace                 |
| <code>\S</code>        | Any Non-whitespace character   |
| <code>^...\$</code>    | Starts and ends                |
| <code>(...)</code>     | Capture Group                  |
| <code>(a(bc))</code>   | Capture Sub-group              |
| <code>(.*)</code>      | Capture all                    |
| <code>(abc def)</code> | Matches abc or def             |

# / REGEX使用場合

上面的例子只是一個簡單的說明，正規表示式還可以處理不定長度，夾雜著字母，數字甚至標點符號的模式。要補充的是針對一個模式，正規表示式的模式字串有多種不同的寫法。不同的寫法可能都可以在特定文檔類型當中找到同樣的結果，但有可能在另一類文檔找到的結果卻不盡然相同。

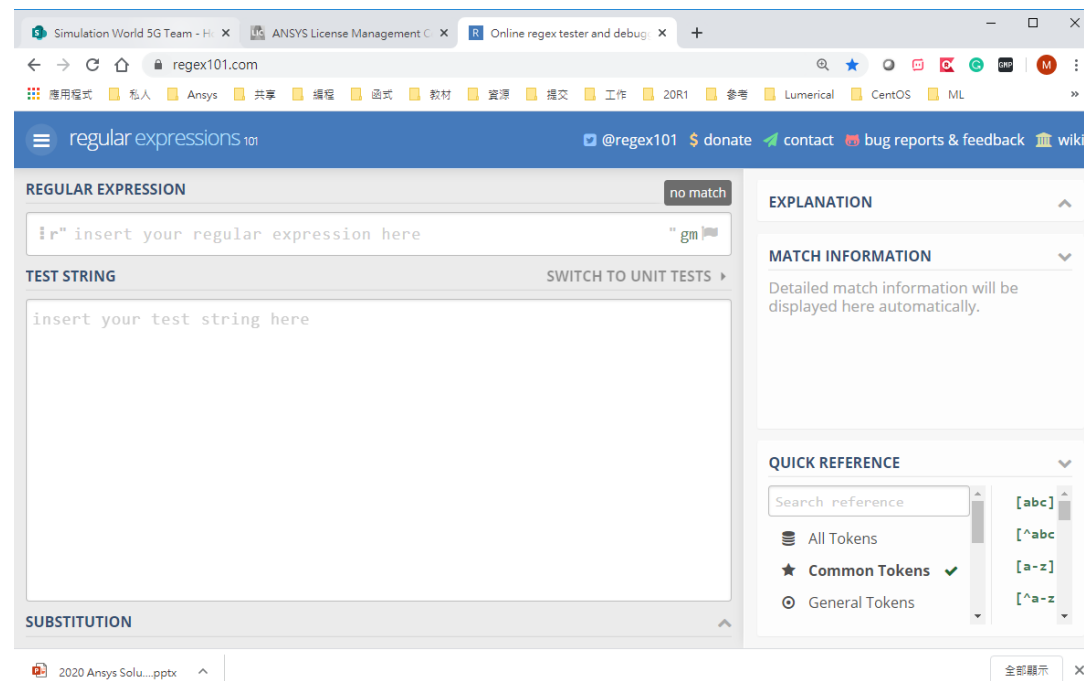
雖然這裡一直強調正規表示式的方便，但是如果可以用簡單的方式就可以找出字串，就不要使用正規表示式。比方說一個CSV檔用split就可以把數值資料分離出來。用正規表示式雖然也可以做到，但可耗費數十倍的時間。

# 測試模式字串

實務上要一次寫出正確的正規表示式來精確的吻合所要的結果，不多不少，並不是一件容易的事。通常要經過多次的嘗試才能找到最適合的模式字串。這裡推薦一個網站

[regex101](https://regex101.com)，

將要搜尋的文本複製到文本框當中，在正規表示式的欄位輸入模式字串，文檔當中所有匹配到的字串便立即以顏色標示出來，這讓我們很容易的判斷模式字串是否有過度匹配或有匹配不足的狀況。透過不斷調整直到符合所需。最後便可以將找到的模式字串寫到程式當中。





# / Python Regex

三個最常用的Python 正規表示式函數，需先import re

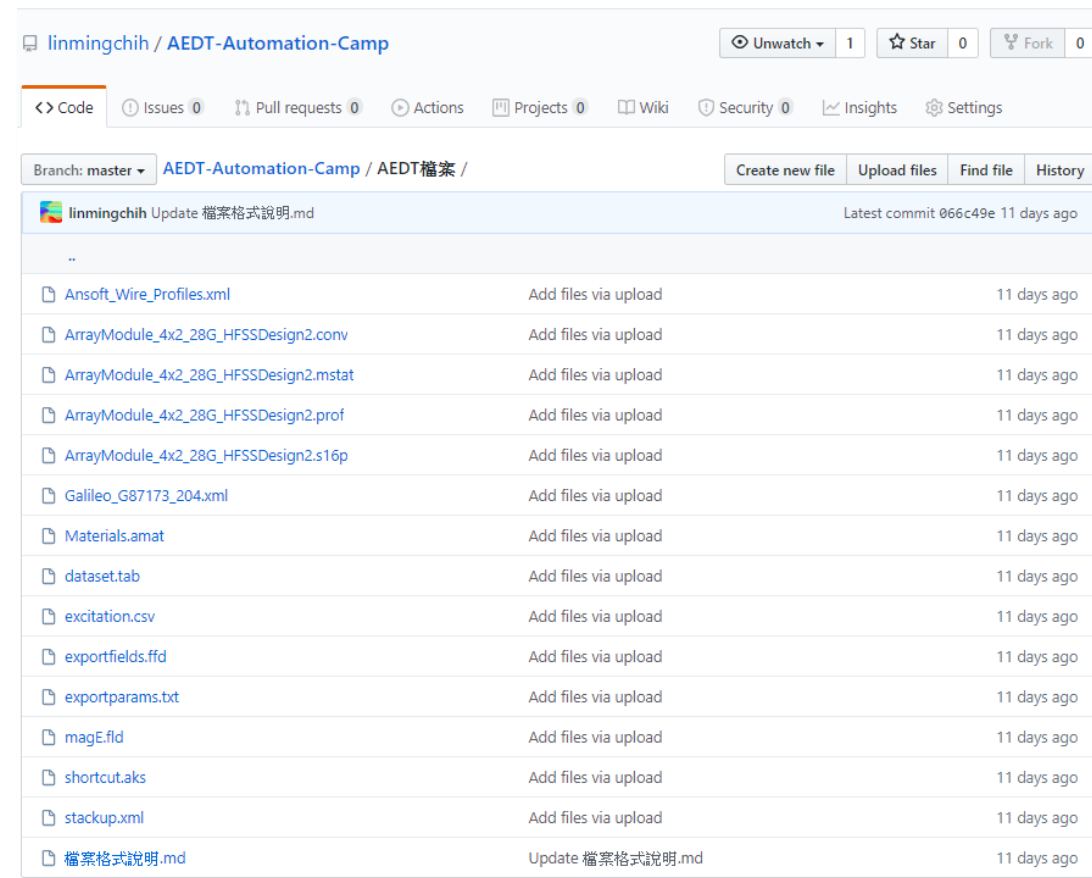
- re.search()
- re.findall()
- re.sub()

```
>>> x='<Layer Color="#44614c" Material="AIR" Name="UNNAMED_000" Thickness="0" Type="dielectric"/>'
>>> import re
>>> m = re.search('Material=\"(\w+)\"', x)
>>> m.group(1)
'AIR'
>>> m = re.findall('(\w+)=\"(\w+)\"', x)
>>> m
[('Material', 'AIR'), ('Name', 'UNNAMED_000'), ('Thickness', '0'), ('Type', 'dielectric')]
>>> y = re.sub('Material=\"(\w+)\"', 'Material="Copper"', x)
>>> y
'<Layer Color="#44614c" Material="Copper" Name="UNNAMED_000" Thickness="0" Type="dielectric"/>'
>>> |
```

# 範例解說：使用REGEX 來處理XML及Prof

# / AEDT可匯出之資料檔

- 檔案範例連結
- 格式
  - Touchstone1.0 .sNp
  - Touchstone 2.0 .ts
  - SPICE .cir, .sp
  - Result .csv
  - Far Field .ffd
  - Near Field .nfd
  - Excitation .csv
  - Profile .prof
  - Mesh .mstat
  - Dataset .tab
  - Stackup .xml
- Material .amat
- Option .xml
- Welement .sp
- Variables .autovar
- ...



# 處理XML與Prof

在AEDT當中有許多的文檔紀錄著模擬相關的資訊及模擬結果，接下來我們用這些實際的檔案來練習如何使用正規表示式來找出特定的訊息。

以stackup.xml為例

- 找出所有的材料名
- 找出所有的層數名稱及對應的材料及厚度

以.prof為例

- 找出每一次adaptive mesh網格(tetrahedra)的數量
- 找出每一次Solver所花的時間及耗用的記憶體。

# 專案問題解析