Week 6-正規表示式與程式優化

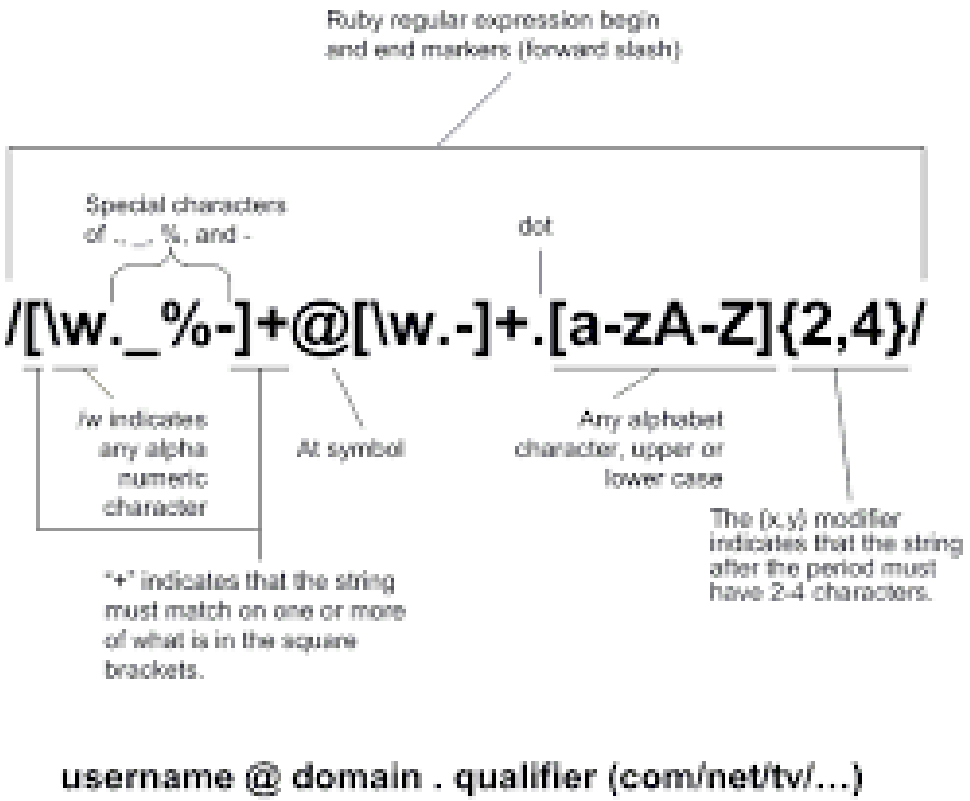
正規表示式

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。



簡單例子(身分證字號)

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，接下來數字1代表男性，2代表女性。之後是八位數字。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][12]\d{8}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。同樣的第二個字元必須為1或2。\d代表了一個字元長度的數字，後面的{8}代表了前面的字元，這裡也就是數字。要連續出現八遍。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。

正規表示法編程步驟

找出字串模式

解析字串模式

(寫出REGEX)

寫出程式碼

(搜尋/取代)

Regular expression cheatsheet

Special characters

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

Quantifiers

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

Special sequences

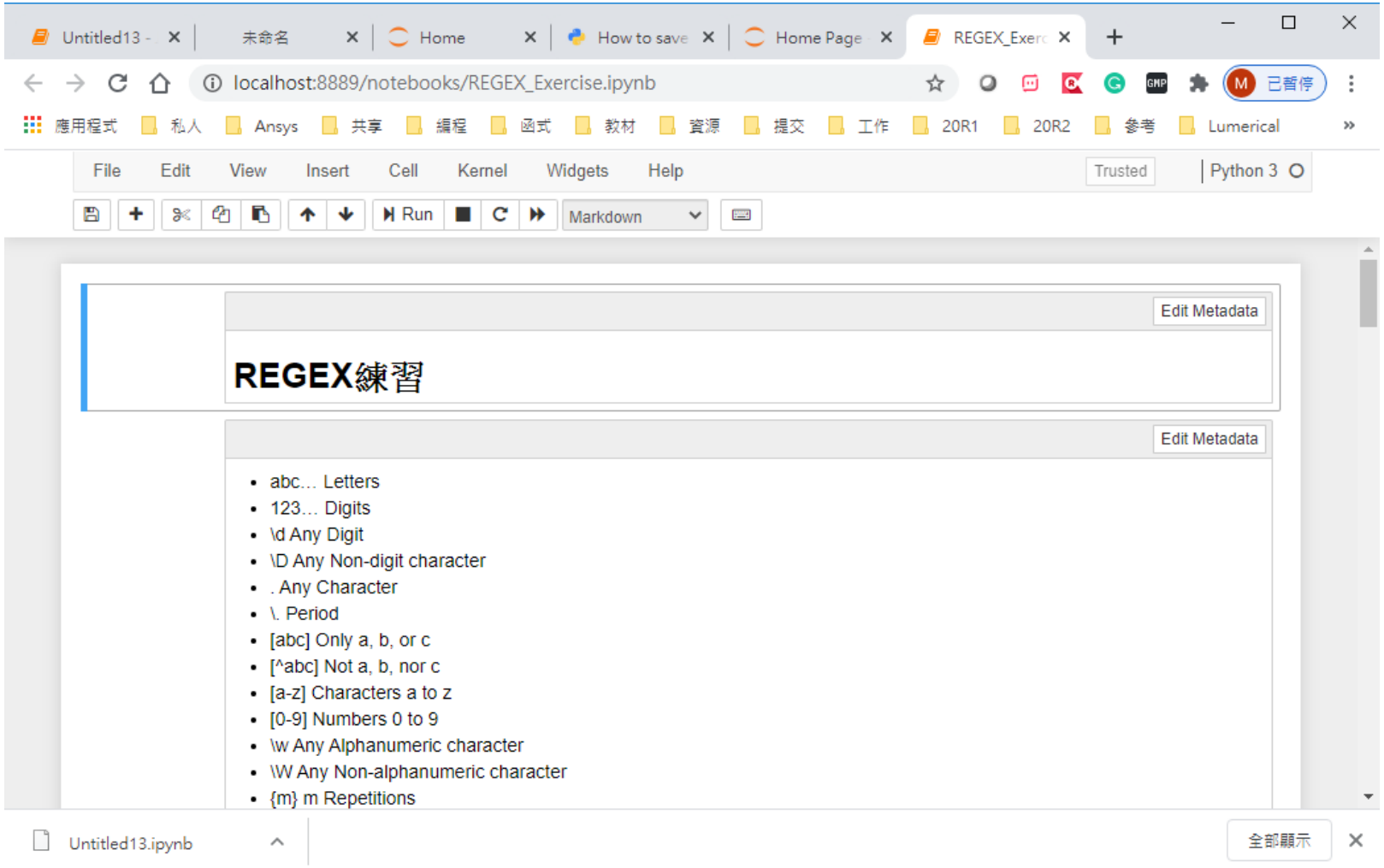
`\A`	start of string
`\b`	matches empty string at word boundary (between `\w` and `\W`)
`\B`	matches empty string not at word boundary
`\d`	digit
`\D`	non-digit
`\s`	whitespace: `[ \t\n\r\f\v]`
`\S`	non-whitespace
`\w`	alphanumeric: `[0-9a-zA-Z_]`
`\W`	non-alphanumeric
`\Z`	end of string
`\g<id>`	matches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

Special sequences

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

Based on tartley's python-regex-cheatsheet.

基本練習：


Untitled13 -
未命名
Home
How to save
Home Page
REGEX_Exerc
localhost:8889/notebooks/REGEX_Exercise.ipynb
已暫停
應用程式
私人
Ansys
共享
編程
函式
教材
資源
提交
工作
20R1
20R2
參考
Lumerical
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Markdown
Edit Metadata
REGEX練習
Edit Metadata
abc… Letters
123… Digits
\d Any Digit
\D Any Non-digit character
. Any Character
\. Period
[abc] Only a, b, or c
[^abc] Not a, b, nor c
[a-z] Characters a to z
[0-9] Numbers 0 to 9
\w Any Alphanumeric character
\W Any Non-alphanumeric character
{m} m Repetitions
Untitled13.ipynb
全部顯示

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。

範例解說：使用REGEX來處理XML及Prof

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

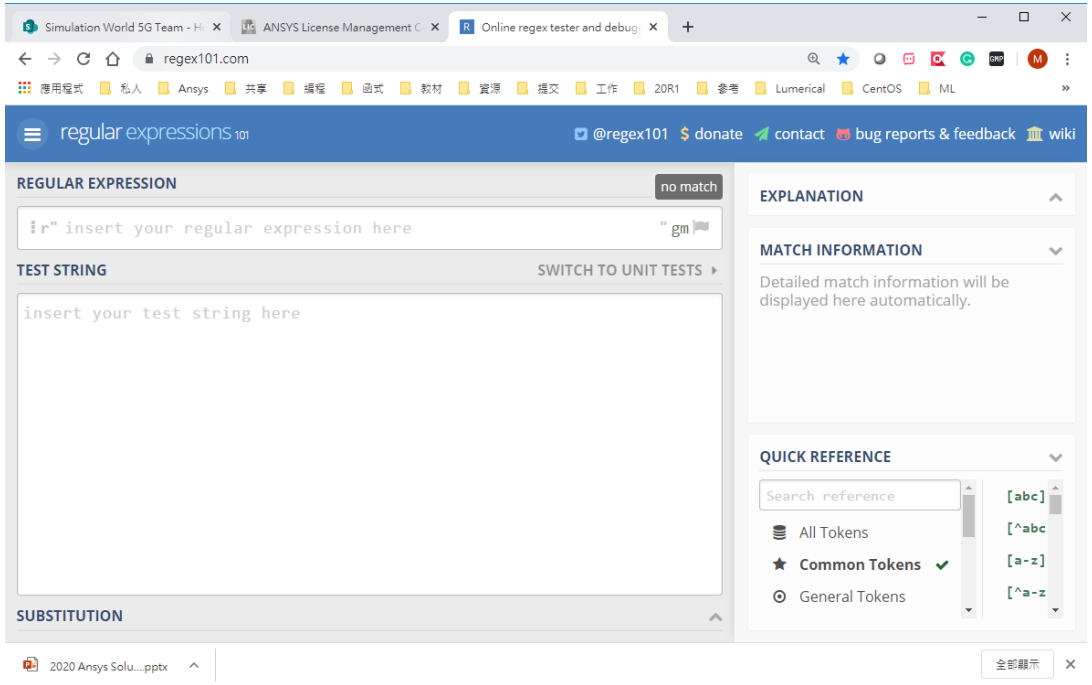
以.prof為例

- 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。

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 ，

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"/>'
>>>
```

調換group

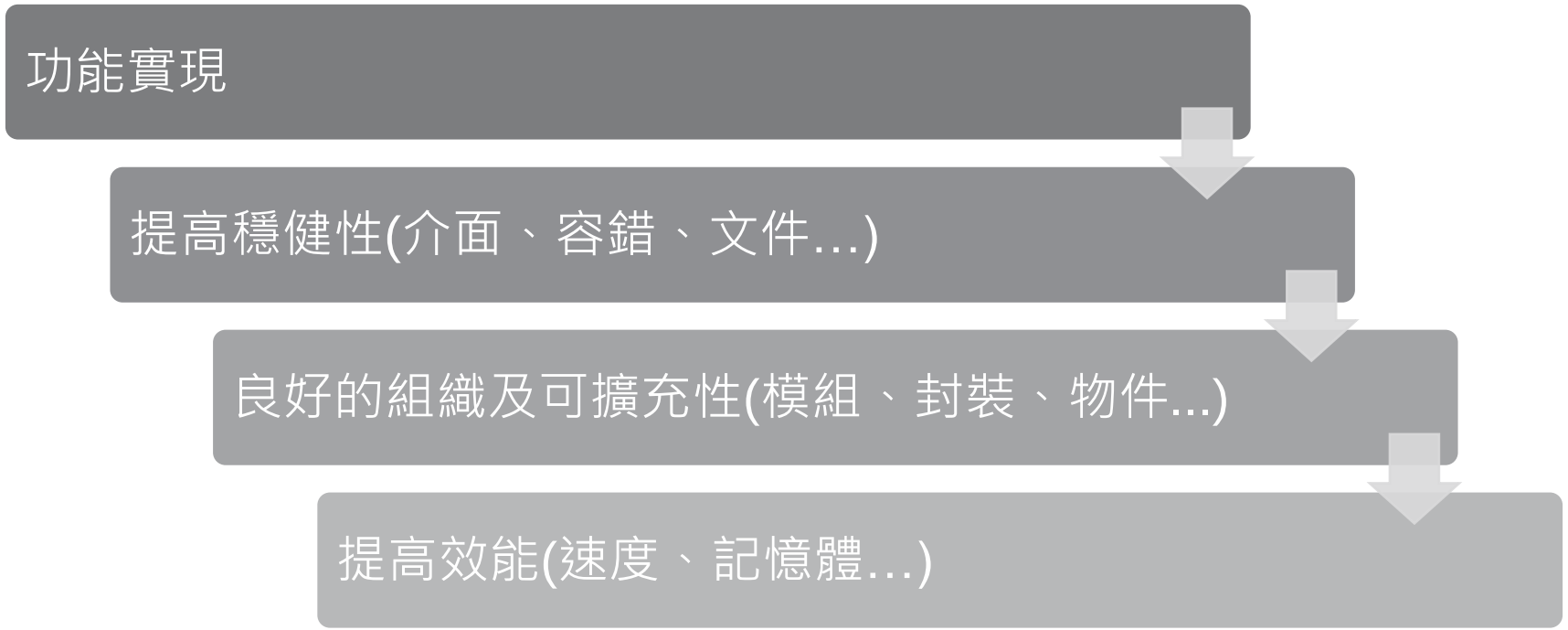
```
In [50]: re.sub('(\w+),(\w+),(\w+)', '\g<2>:\g<3>:\g<1>', 'abc,def,ghi')
Out[50]: 'def:ghi:abc'
```

提升運算速度及減少消耗記憶體的Python編程技巧

Ansys

程式發展進程

- 程式設計講求的不只是功能正確，還希望能用較少的時間及運算資源(核心數、記憶體)來完成工作。
- 程式設計追求持續的優化。



測試程式性能

- 測試執行時間
  - time.time()
  - Profiler in Anaconda

- 測試使用記憶體大小

```
import os
import psutil
process = psutil.Process(os.getpid())
print(process.memory_info().rss)  # in bytes
```

Python程式效能優化技巧

- 對超大型檔案(>GB)採取單行讀取檔案。
- 迴圈優化，適當採用break與continue指令降低迴圈次數。
- 善用map, zip, range, enumerate等函數，減少記憶體使用量。
- 善用內建函式，如max(), min(), sum()…
- 生成器(Generator)的使用。
- 使用numpy來處理電磁場向量的運算。
- 以二進制紀錄資料以加快下次載入速度(pickle)。
- 用del()+gc.collect()回收記憶體
- 編譯Python檔，例如Cython

AEDT程式效能優化技巧

- 針對大量3D建模採用命令列模式處理，不要在繪圖視窗處理。
- 對於複雜電路板或封裝處理，可採用EDB框架處理。
- 當圖表數量很大的時候，匯出資料再以Matplotlib生成圖表。
- 將原始碼編譯為.dll檔
- 以陣列傳遞參數：https://www.youtube.com/watch?v=ef5LvPMu3Ro

生成器(Generator)介紹-(時間/記憶體)

```
import time

t0=time.time()
x=range(1000000000)
t1=time.time()
y=(i for i in x if i%7==0 or i%5==0)
t2=time.time()
z=sum(y)
t3=time.time()
print(f'{t1-t0:.3}')    0.719
print(f'{t2-t1:.3}')    0.87
print(f'{t3-t2:.3}')    97.6
print(f'{t3-t0:.3}')    99.2
print(z)                157142856357142861
```

```
import time

t0=time.time()
x=list(range(1000000000))
t1=time.time()
y=[i for i in x if i%7==0 or i%5==0]
t2=time.time()
z=sum(y)
t3=time.time()
print(f'{t1-t0:.3}')    15.9
print(f'{t2-t1:.3}')    92.4
print(f'{t3-t2:.3}')    8.89
print(f'{t3-t0:.3}')    1.17e+02
print(z)                157142856357142861
```

Ansys

善用内建函式，如max(), min(), sum()…

```python
import time
t0 = time.time()
x = range(10000000)
mysum = 0
for i in x:
    mysum += i
print(f'result:{mysum}')
print(f'time:{time.time()-t0}')
```

```
result:49999995000000
time:0.8880486488342285
```

```python
import time
t0 = time.time()
x = range(10000000)
y = sum(x)
print(f'result:{mysum}')
print(f'time:{time.time()-t0}')
```

```
result:49999995000000
time:0.3400413990020752
```

```python
import time
import math
import random
t0 = time.time()
x = [random.random() for i in range(10000000)]
```

```python
t0 = time.time()
mymax= -math.inf
for i in x:
    if i > mymax:
        mymax = i
print(f'result:{mymax}')
print(f'time:{time.time()-t0}')
```

```
result:9999999
time:0.8810136318206787
```

```python
t0 = time.time()
print(f'result:{max(x)}')
print(f'time:{time.time()-t0}')
```

```
result:9999999
time:0.27100229263305664
```

Ansys

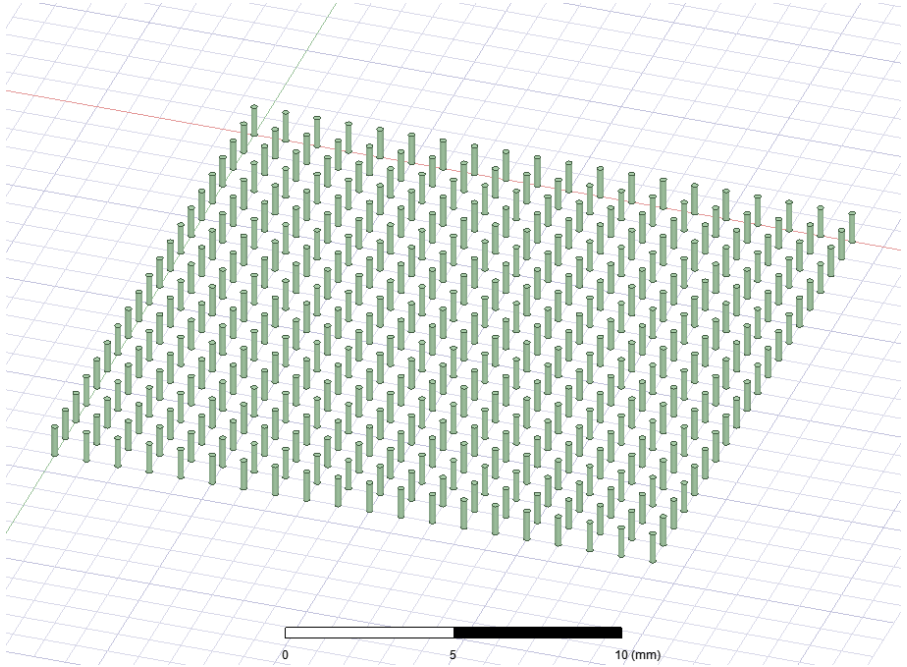
Numpy大幅加快運算速度

```python
import numpy as np
big_array = np.random.rand(100000000)

t0 = time.time()
mysum=0
for i in big_array:
    mysum += i
print(f'time:{time.time()-t0}')

t0 = time.time()
sum(big_array)
print(f'time:{time.time()-t0}')

t0 = time.time()
np.sum(big_array)
print(f'time:{time.time()-t0}')
```

```
time:17.710095405578613
time:8.193042755126953
time:0.13800430297851562
```

批次檔建模

- 在HFSS建立3D模型時，可以明顯感覺得出來當模型數量越多時，執行速度會越來越慢。舉例來說，利用雙迴圈的python代碼產生20x20的圓柱陣列(如圖一)花了約97秒的時間。更複雜的結構十多分鐘跑不掉。
- 有沒有方法可以加速3D模型的產生？其實是有的，透過在命令行執行腳本，由於不需要打開AEDT視窗，省去了模型渲染的時間，時間大幅縮短到1.1秒，快了足足有88倍之多。以下是批次檔的程式碼：




```
SET PATH=%PATH%;C:\Program Files\AnsysEM\AnsysEM20.1\Win64
ansysedt.exe -features=beta -ng -RunScriptAndExit c:\demo\test.py
```


Ansys