



# 深入瞭解Django的MVC架構

1

## 學習目標

### ➤ Django的MVC架構簡介

- MVC架構簡介、MTV架構
- Django網站的構成以及配合
- 在Django MTV架構下的網站開發步驟

### ➤ Model簡介

- 在models.py中建立資料表、admin.py資料表管理介面
- 在Python Shell中操作資料表、資料的查詢與編輯

### ➤ View簡介

### ➤ Template簡介

## MVC架構簡介

➔ MVC是一種軟體工程設計方法

➔ Model資料模組

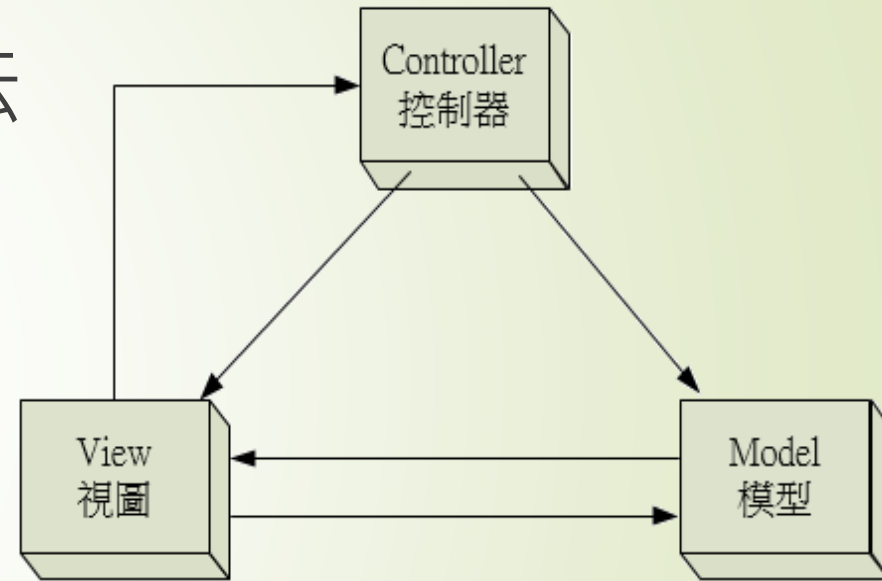
➔ View視圖模組

➔ Controller控制器模組

➔ 這3個模組之間相互配合，  
因應使用者的操作而顯示出使用者想要的結果



使用者



# MVC架構簡介

## ➤ MVC各模組說明

模組種類	說明
<b>Model</b> 資料模組	包含了系統中的資料內容，通常都是以資料庫的型式儲存，如果這些內容有變動的話，會通知View即時更改顯示的內容，一些處理資料的程式邏輯也會放在這邊。
<b>View</b> 視圖模組	建立和使用者的介面，傳遞使用者的請求給Controller，並負責依照Controller的要求呈現出來自於Model中的資料。
<b>Controller</b> 控制模組	透過由View傳來的使用者請求，派發這些請求，並依這些請求進行處理資料內容以及設定要呈現的資料

## Django的MTV架構

### ➤ Model

- 網站的資料儲存model.py

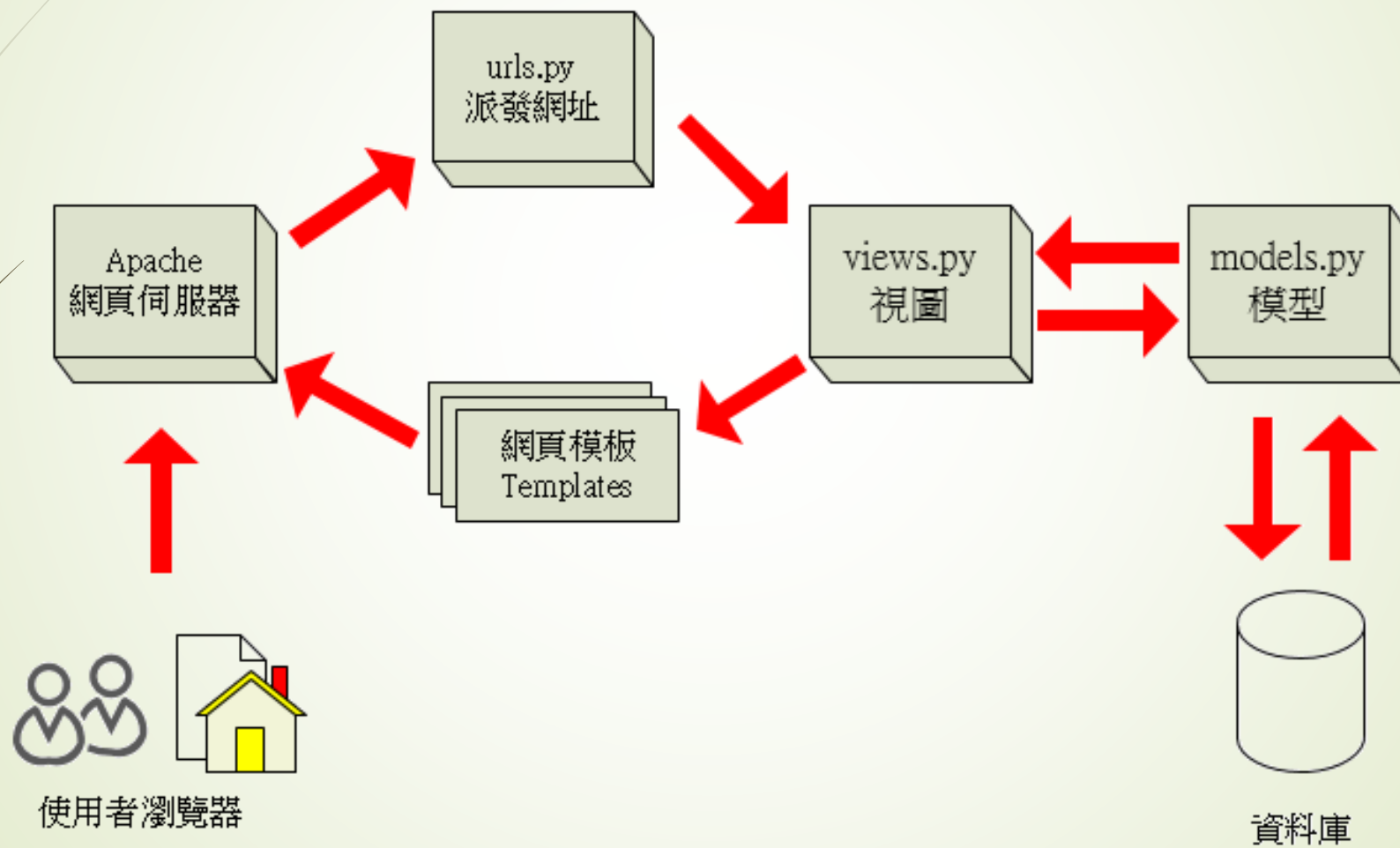
### ➤ View

- 負責控制如何處理資料程式邏輯的views.py

### ➤ Template

- 網站的模板文件群  
( 一般都是放在templates資料夾下的html檔案 )

# Django的MTV架構



# Django MTV架構下的網站開發步驟

## ➤ 需求分析

- 具體列出本次網站專案所要達成的目標 (包括簡單的畫面草圖與功能方塊圖等)

## ➤ 資料庫設計

- 建立資料模組之前，釐清網站中所有會用到的資料內容、格式以及各資料之間的關係。
- 瞭解網站的每一個頁面，並設計網頁模板(.html)檔案。
- 使用virtualenv建立並啟用虛擬環境
- 使用pip install 安裝django

```
mynewsite
├── manage.py
├── myapp
│   ├── admin.py
│   ├── __init__.py
│   ├── migrations
│   │   └── __init__.py
│   ├── models.py
│   ├── tests.py
│   └── views.py
├── mynewsite
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── static
└── templates
```



## Django MTV架構下的網站開發步驟

- 使用 `django-admin startproject XXX (myblog)` 產生專案
- 使用 `python manage.py startapp xxx (mainsite)` 建立 app
- 在 app 內建立 `templates` 資料夾，並把所有屬於該 app 的網頁模板 (`.html`) 檔案都放在此資料夾中
- 在 app 內建立 `static` 資料夾，並把所有屬於該 app 的靜態檔案（影像檔案、`.css` 檔案以及 `.js` 等等）都放在此資料夾中
- 修改 `settings.py`，把相關的資料夾設定都加入，也把產生的 app 名稱加入 `INSTALLED_APPS` 串列中
- 編輯 `models.py`，建立資料庫表格

## Django MTV架構下的網站開發步驟

- 編輯 `views.py`，先import在 `models.py` 中建立的資料模型
- 編輯 `admin.py`，把 `models.py` 中定義的資料模型加入，並使用 `admin.site.register` 註冊新增的類別，讓admin介面可以開始處理資料庫內容
- 編輯 `views.py`，設計處理資料的相關模組，輸入和輸出均透過 `templates` 相關的模組操作取得來自於網頁的輸入資料，以及顯示和 `.html` 檔案中渲染後的網頁內容
- 編輯 `urls.py`，先import在 `views.py` 中定義的模組
- 編輯 `urls.py`，建立網址和 `views.py` 中定義的模組之對應關係

## Django MTV架構下的網站開發步驟

- 執行 `python manage.py makemigrations`
- 執行 `python manage.py migrate`
- 執行 `python manage.py runserver` 測試網站

## Model簡介

- ▶ Model是Django表示資料的模式，以Python的類別為基礎在models.py中設定資料項目與資料格式，基本上**每一個類別就對應到一個資料庫中的資料表**。
- ▶ 定義每一個資料項目的時候，除了資料項目名稱之外，也要定義**此項目的格式以及這張表格和其它表格相互之間的關係（資料關聯）**

## 在models.py中建立資料表

- 以在第2堂課中介紹的簡易部落格網站之models.py為例，程式碼如下：
  - `class Post(models.Model):`
  - `title = models.CharField(max_length=200)`
  - `slug = models.CharField(max_length=200)`
  - `body = models.TextField()`
  - `pub_date = models.DateTimeField(default=timezone.now)`

# models.Model中常用的資料欄位格式

欄位格式	可以使用的參數	說明
<b>BigIntegerField</b>		64位元之 <b>大整數</b>
<b>BooleanField</b>		<b>布林值</b> ，只有True/False兩種
<b>CharField</b>	max_length：指定可接受之字串長度	用來儲存 <b>較短資料的字串</b> ，通常使用於單行的文字資料
<b>DateField</b>	auto_now：每次物件被儲存時即自動加入目前日期 auto_now_add：只有在物件被建立時才加入目前日期	<b>日期</b> 格式，可用於datetime.date
<b>DateTimeField</b>	同上	<b>日期時間</b> 格式，對應到datetime.datetime
<b>DecimalField</b>	max_digits：可接受的 <b>最大位數</b> decimal_places：在所有的位數中， <b>小數佔幾個位數</b>	<b>定點小數</b> 數值資料，適用於Python的Decimal模組之實例

# models.Model中常用的資料欄位格式

<b>EmailField</b>	max_length : 最長字數	可接受電子郵件位址格式之欄位
<b>FloatField</b>		浮點數欄位
<b>IntegerField</b>		整數欄位，是通用性最高的整數格式
<b>PostiveIntegerField</b>		正整數欄位
<b>SlugField</b>	max_length : 最大字元長度	和CharField是一樣的，通常用來當做是網址的一部份
<b>TextField</b>		長文字格式，一般是用在HTML表單中的Textarea輸入項目
<b>URLField</b>	max_length : 最大字元長度	和CharField是一樣的，特別用來記錄完整的URL網址

# models.Model各欄位常用的屬性說明

欄位選項	說明
null	此欄位是否接受儲存空值NULL，預設值是False
blank	此欄位是否接受儲存空白內容，預設值是False
choices	以選項的方式（只有固定內容的資料可以選用） 當做是此欄位的候選值
default	輸入此欄位的預設值
help_text	欄位的求助訊息
primary_key	把此欄位設定為資料表中的主鍵KEY，預設值為False
unique	設定此欄位是否為唯一值，預設值為False



- 首次設定Model的內容要先執行`makemigrations`的指令，以及`migrate`指令，如下所示：
  - `python manage.py makemigrations`
  - `python manage.py migrate`
- `0001_initial.py`
  - `0001_initial.py`檔案記錄第一次Model設定的資料表內容，因為一開始只有一個設定，所以只有0001這個版號

## 建立測試專案

- `django-admin startproject MTV01`
- `cd MTV01`
- `python manage.py startapp mysite`
- 編輯 `settings.py`

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'mysite',  
]
```

以下是一些簡單的設定例子：

19

```
from django.db import models
```

```
class NewTable(models.Model):
    models_f = models.BigIntegerField()
    bool_f = models.BooleanField()
    date_f = models.DateField(auto_now=True)
    char_f = models.CharField(max_length=20, unique=True)
    datetime_f = models.DateTimeField(auto_now_add=True)
    decimal_f = models.DecimalField(max_digits=10,
    decimal_places=2)
    float_f = models.FloatField(null=True)
    int_f = models.IntegerField(default=2010)
    tesxt_f = models.TextField()
```

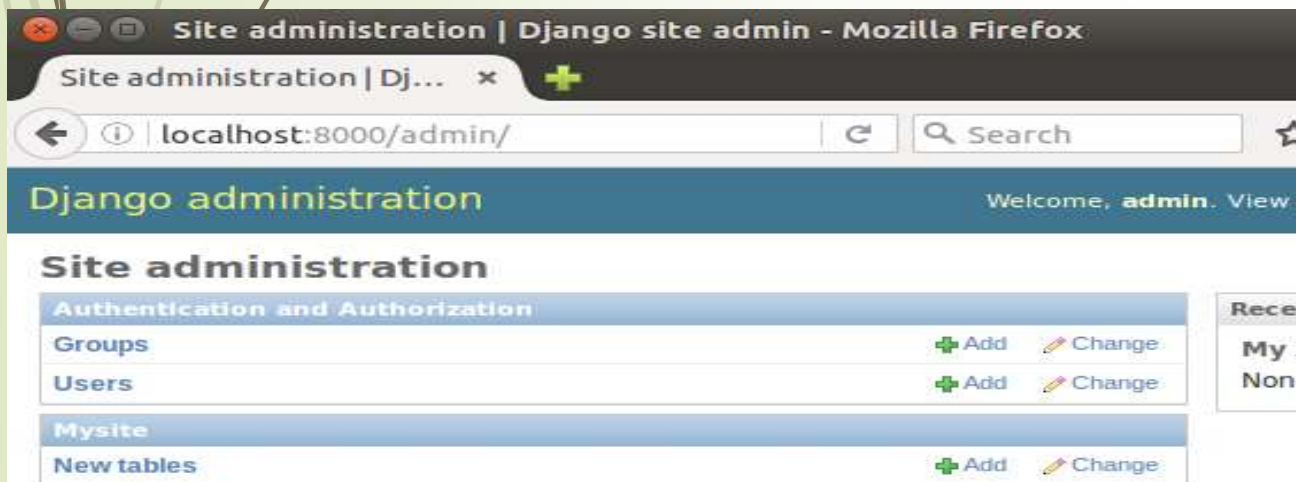
```
BEGIN;
CREATE TABLE "mysite_newtable" ("id" integer NOT NULL PRIMARY KEY
AUTOINCREMENT, "bigint_f" bigint NOT NULL, "bool_f" bool NOT NULL, "date_f"
date NOT NULL, "char_f" varchar(20) NOT NULL UNIQUE, "datetime_f" datetime
NOT NULL, "decimal_f" decimal NOT NULL, "float_f" real NULL, "int_f" integer
NOT NULL, "text_f" text NOT NULL);
```

```
COMMIT;
```

# admin.py中建立資料表管理介面

20

- ▶ 在admin.py中加入這個NewTable，就可以在/admin中管理這張資料表
  - ▶ `from django.contrib import admin`
  - ▶ `from mysite.models import NewTable (Post)`
  - ▶ `admin.site.register(NewTable)`
- ▶ `python manage.py createsuperuser` (記得要migrate)
- ▶ 瀏覽器連線到localhost:8000/admin



```
from django.contrib import admin
# Register your models here.
from .models import Post

class PostAdmin(admin.ModelAdmin):
    list_display = ('title', 'slug', 'pub_date')

admin.site.register(Post, PostAdmin)
```

## 欄位提供候選資料

- 在models.py中加入如下所示的內容：
  - `class Product(models.Model):`
  - `SIZES = (`
  - `('S', 'Small'),`
  - `('M', 'Medium'),`
  - `('L', 'Large'),`
  - `)`
  - `sku = models.CharField(max_length=5)`
  - `name = models.CharField(max_length=20)`
  - `price = models.PositiveIntegerField()`
  - `size = models.CharField(max_length=1, choices=SIZES)`

## 欄位提供候選資料

- 編輯models.py之後，再執行migrate才行（如果中間有修改過，則需要先執行makemigrations
  - 回到admin.py加入這個新的類別並註冊：
  - from django.contrib import admin
  - from mysite.models import NewTable, Product
  - admin.site.register(NewTable)
  - admin.site.register(Product)

The screenshot shows the Django administration interface in a web browser. The browser's address bar displays 'localhost:8000/admin/'. The page title is 'Django administration' and the user is logged in as 'ADMIN'. The interface is divided into several sections:

- Site administration:** A header section with the title 'Site administration'.
- AUTHENTICATION AND AUTHORIZATION:** A section containing two items:
  - Groups:** Includes '+ Add' and 'Change' (pencil icon) links.
  - Users:** Includes '+ Add' and 'Change' (pencil icon) links.
- MYSITE:** A section containing two items:
  - New tables:** Includes '+ Add' and 'Change' (pencil icon) links.
  - Products:** Includes '+ Add' and 'Change' (pencil icon) links.
- Recent actions:** A section titled 'Recent actions' containing a list of actions under the heading 'My actions':
  - Product object (4) - Product
  - Product object (3) - Product
  - Product object (2) - Product
  - Product object (1) - Product
  - NewTable object (2) - New table
  - NewTable object (1) - New table

# 欄位提供候選資料

Django administration

WELCOME, [ADMIN](#) / [VIEW SITE](#) / [CHANGE PASSWORD](#) / [LOG OUT](#)

[Home](#) > [App01](#) > [Products](#) > Add product

Add product

SKU:

Name:

Price:

Size:

- Small
- Medium
- Large

Save and add another

Save and continue editing

SAVE



## 小練習

- 請新增models Product 的欄位 qty 為一正整數 為庫存
- 並新增幾樣物品

Django administration  
WELCOME, ADMIN. [VIEW SITE](#) / [CHANGE PASSWORD](#) / [LOG OUT](#)

Home > App01 > Products > GrayBox

Change product HISTORY

Skus: 0001

Name: GrayBox

Price: 100

Size: Small

Qty: 10

Delete Save and add another Save and continue editing SAVE

## 小練習2

➡ 並修改 Product 與 NewTable 的顯示成為 **Name** 與 **Char\_f**

Select product to change

Action:   0 of 4 selected

- PRODUCT
- Product object (4)
- Product object (3)
- Product object (2)
- Product object (1)

4 products

Select new table to change

Action:   0 of 3 selected

- NEW TABLE
- NewTable object (3)
- NewTable object (2)
- NewTable object (1)

3 new tables

Select product to change

Action:   0 of 4 selected

- PRODUCT
- 呆河馬
- 呆呆獸
- 烈焰馬
- 小火馬

4 products

- 在上一章之中我們使用過下列方式修改admin中post資料表的顯示方式
- 請試著用相同的方法修改Product的管理介面顯示式

```
from django.contrib import admin
from .models import Post

# Register your models here.

class PostAdmin(admin.ModelAdmin):
    list_display = ('title', 'slug', 'pub_date')

admin.site.register(Post, PostAdmin)
```

## 小練習3

- 新增管理介面的欄位
- 品名 價格 庫存

Home > Mysite > Products

Select product to change ADD PRODUCT +

Action:   0 of 4 selected

<input type="checkbox"/>	NAME	PRICE	QTY
<input type="checkbox"/>	呆河馬	250	3
<input type="checkbox"/>	呆呆獸	150	5
<input type="checkbox"/>	烈焰馬	200	2
<input type="checkbox"/>	小火馬	100	6

4 products

## 在Python Shell中操作資料表

- 以ORM的方式來存取資料庫裡的內容
- ORM(Object Relational Mapper)，是一種物件導向程式設計技術的一種，它以物件的方式來看待每一筆資料，可以解決底層資料庫相容性的問題
- 可以在Python的交談式介面中直接操作使用，如下：
  - # `python manage.py shell`
  - Python 3.5.2 (default, Nov 23 2017, 16:37:01)
  - [GCC 5.4.0 20160609] on linux
  - Type "help", "copyright", "credits" or "license" for more information.
  - (InteractiveConsole)

## 在Python Shell中 創建新的資料

```
➤ >>> from mysite.models import Product
```

```
➤ >>> Product.objects.create(sku='0001',name='雷丘',price=100,size='L',qty=2)
```

```
➤ >>> p = Product (sku='0002',name='皮丘',price=30,size='S',qty=30)
```

```
➤ >>> p.save()
```

```
➤ >>> exit()
```

The screenshot shows a web browser window with the address bar displaying `localhost:8000/admin/mysite/product/5/change/`. The page title is "Change product | Djang". The main header is "Django administration" with a welcome message for "ADMIN" and links for "VIEW SITE", "CHANGE PASSWORD", and "LOG OUT". The breadcrumb trail is "Home > Mysite > Products > 雷丘". The "Change product" form includes the following fields:

- Sku:** 0001
- Name:** 雷丘
- Price:** 100
- Size:** Large (dropdown menu)
- Qty:** 2

At the bottom, there are four buttons: "Delete" (red), "Save and add another" (blue), "Save and continue editing" (blue), and "SAVE" (dark blue). A "HISTORY" button is located in the top right corner of the form area.

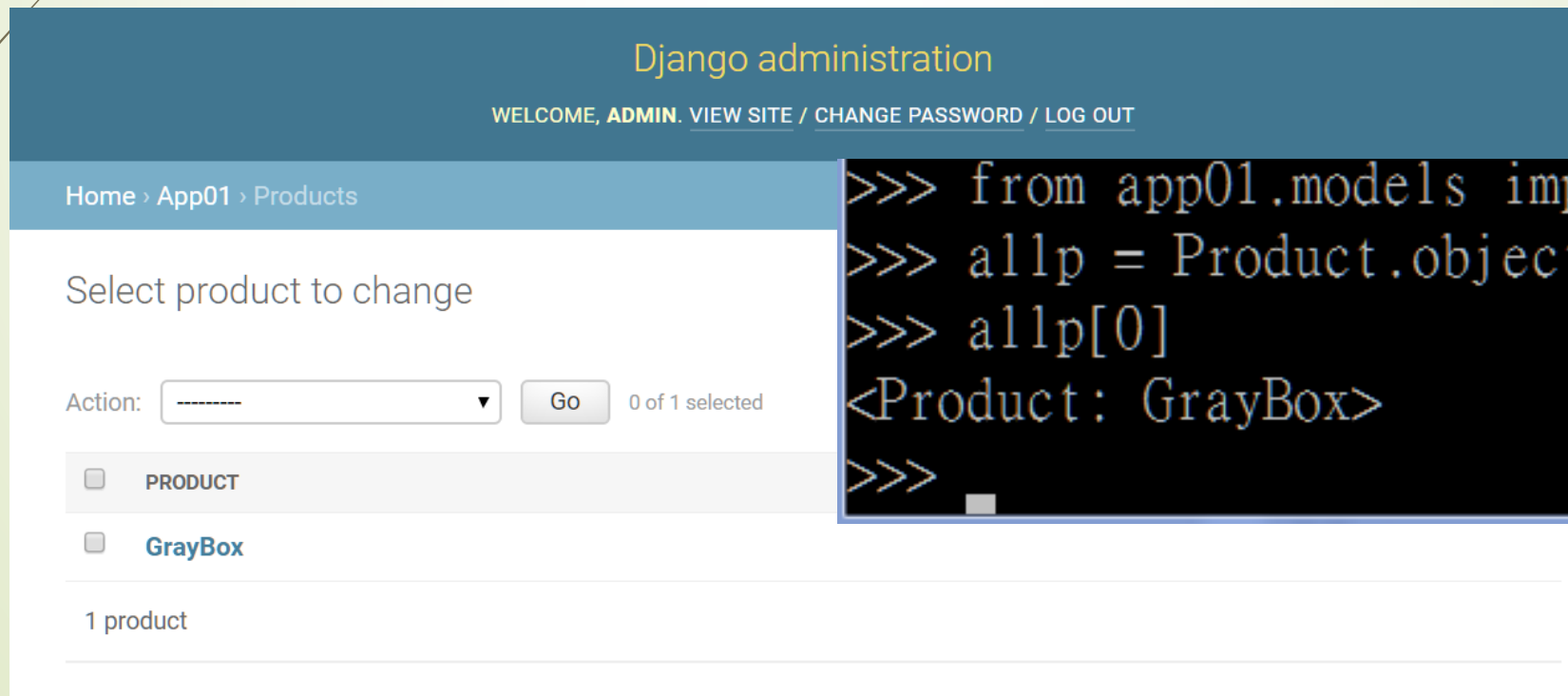
## 在Python Shell中 查詢資料

- 使用`Product.objects.all()`函數以取得所有的資料，而其資料型態稱為`QuerySet`，操作過程如下所示：
  - `# python manage.py shell`
  - `Python 3.5.2 (default, Nov 23 2017, 16:37:01)`
  - `[GCC 5.4.0 20160609] on linux`
  - `Type "help", "copyright", "credits" or "license" for more information.`
  - `(InteractiveConsole)`
  - `>>> from mysite.models import Product`
  - `>>> allp = Product.objects.all()`
  - `>>> allp[0]`
  - `<Product: Product object >`



# 在Python Shell中操作資料表

- `__str__` 函數是當這個類別的實例被列印出來的時候會呼叫的函數，我們直接把它覆寫成顯示其中的 `name` 這個欄位
  - `def __str__(self):`
  - `return self.name`



Django administration

WELCOME, ADMIN. [VIEW SITE](#) / [CHANGE PASSWORD](#) / [LOG OUT](#)

Home > App01 > Products

Select product to change

Action:   0 of 1 selected

<input type="checkbox"/>	PRODUCT
<input type="checkbox"/>	GrayBox

1 product

```
>>> from app01.models import Product
>>> allp = Product.objects.all()
>>> allp[0]
<Product: GrayBox>
>>>
```

# 資料的查詢與編輯

3

8

- 除了之前的 `create()`、`save()` 和 `all()` 這三個函數之外，其它常用的函數以及可以加在函數中的修飾元摘要，如下表：

函數名稱或修飾元	說明
<code>filter()</code>	傳回符合指定條件的 <code>QuerySet</code>
<code>exclude()</code>	傳回不符合指令條件的 <code>QuerySet</code>
<code>order_by()</code>	串接到 <code>QuerySet</code> 之後，針對某一指定的欄位進行 <b>排序</b> 的動作。
<code>all()</code>	傳回 <b>所有的</b> <code>QuerySet</code>
<code>get()</code>	取得 <b>指定符合條件的唯一元素</b> ，如果找不到或是一個以上符合條件，均會產生 <code>exception</code>
<code>first()/last()</code>	取得 <b>第1個</b> 和 <b>最後1個</b> 元素
<code>aggregate()</code>	可以用來計算資料項目的 <b>聚合函數</b>
<code>exists()</code>	用來檢查 <b>是否存在某指令條件之記錄</b> ，通常是附加在 <code>filter()</code> 後面

## 資料的查詢與編輯

<code>update()</code>	用來快速更新某些資料記錄中的欄位內容
<code>delete()</code>	刪除指定的記錄
<code>exact / iexact</code>	精確的比對 / 不區分大小寫的條件設定
<code>contains/icontains</code>	設定條件為含有某一字串即符合，像是SQL語法中的LIKE和ILIKE / 不區分大小寫
<code>in</code>	提供一個串列，只要符合串列中的任一值均可
<code>gt/gte/lt/lte</code>	大於/大於等於/小於/小於等於 (數字，日期 可用)
<code>startswith/istartswith</code>	大小寫，開頭的字串相符 / 大小寫不需一樣，開頭相符
<code>endswith/iendswith</code>	大小寫，結尾的字串相符 / 大小寫不需一樣，結尾相符

## 新增五支手機到 product 如下表

欄位依序為：Name, Price, Qty

```
HTC Magic , 100 , 0
SONY Xperia Z3 , 15000 , 1
Samsung DUOS , 800 , 2
Nokia Xpress 5800 , 500 , 1
Infocus M370 , 1500 , 2
```

```
Product.objects.create(sku='p01',name='HTC Magic',price=100,size='S',qty=0)
Product.objects.create(sku='p02',name='SONY Xperia Z3',price=15000,size='S',qty=1)
Product.objects.create(sku='p03',name='Samsung DUOS',price=800,size='S',qty=2)
Product.objects.create(sku='p04',name='Nokia Xpress 5800',price=500,size='S',qty=1)
Product.objects.create(sku='p05',name='Infocus M370',price=1500,size='S',qty=2)
```

# 獲取物件的方法

Product.objects.all() # 查詢所有

Product.objects.all()[:10] # 切片操作，獲取前10筆，不支援負索引，切片可以節約記憶體

Product.objects.get(name="Pikachu") # 名稱為 WeizhongTu 的一條，多條會報錯

# get是用來獲取一個物件的，如果需要獲取多個滿足條件的QuerySet，就要用到filter

Product.objects.filter(name="abc") # 等於Person.objects.filter(name\_\_exact="abc") 名稱嚴格等於 "abc" 的

Product.objects.filter(name\_\_iexact="abc") # 名稱為 abc 但是不區分大小寫，可以找到 ABC, Abc, aBC，這些都符合條件

Product.objects.filter(name\_\_contains="abc") # 名稱中包含 "abc"的

Product.objects.filter(name\_\_icontains="abc") # 名稱中包含 "abc"，且abc不區分大小寫

Product.objects.filter(name\_\_regex="^abc") # 正規表示式運算式查詢

Product.objects.filter(name\_\_iregex="^abc") # 正規表示式不區分大小寫查詢

# filter是找出滿足條件的，當然也有排除符合某條件的

Product.objects.exclude(name\_\_contains="qq") # 排除包含 qq 的Product物件

Product.objects.filter(name\_\_contains="abc").exclude(qty=2) # 找出名稱含有abc，但是排除庫存僅2支的

# 查詢特定資料

44

- 在上述的表格中，有一些函數如`reverse()`以及`exists()`等是可以串接在另外一些函數後面做進一步過濾資訊的，而修飾元則是放在參數中，把欄位名稱後面加上2個底線之後再串接，可以為條件設定增加更多的彈性
- 原本使用`filter`只能設定等號
  - `Product.objects.filter(price=100)`
- 如果要使用小於2的條件，則要改為如下：(不能用「>」)
  - `greater_than_100 = Product.objects.filter(price__gt=100)`
  - `Product.objects.filter(headline__startswith="pika")`

```
>>> greater_than_100 = Product.objects.filter(price__gt=100)
>>> greater_than_100
<QuerySet []>
>>> greater_than_100 = Product.objects.filter(price__gte=100)
>>> greater_than_100
<QuerySet [<Product: GrayBox>]>
>>>
```

- python manage.py shell
- from mysite.model import Product
- # **QuerySet 是可迭代的**
- allpd = Product.objects.all()
- for p in allpd:
- print(p.name,p.price,p.qty,sep='\t')
- #**QuerySet 查詢結果排序**
- Product.objects.all().order\_by('price')
- Product.objects.all().order\_by('-price')

## #QuerySet支持鍊式查詢

```
Product.objects.filter(name__contains="pikachu").filter(qty=2)
```

```
Product.objects.filter(name__contains="尼多").exclude(price=100)
```

# 找出名稱含有尼多, 但是排除庫存只有3隻的

```
Product.objects.filter(name__contains="尼多").exclude(qty=3)
```

## # QuerySet不支援負索引

```
Product.objects.all()[:10] 切片操作, 前10條
```

```
Product.objects.all()[-10:] 會報錯!!!
```

## # 1. 使用 reverse() 解決

```
Product.objects.all().reverse()[:2] # 最後兩條
```

```
Product.objects.all().reverse()[0] # 最後一條
```

## # 2. 使用 order\_by, 在欄目名 (column name) 前加一個負號

```
Product.objects.order_by('-id')[:20] # id最大的20條
```



# Aggregation 聚合

47

- `from django.db.models import Sum, Avg, Max, Min`
- `Product.objects.all().aggregate(Sum('price'))`
- `Product.objects.all().aggregate(Avg('price'))` # {'price\_\_avg': 37.36}
- `Product.objects.all().aggregate(Max('price'))` # {'price\_\_max': Decimal( '82.22')}
- `Product.objects.all().aggregate(Min('price'))`
  
- `from django.db.models import FloatField` # 最大值減平均值
- `Product.objects.aggregate( price_diff=Max('price', output_field=FloatField()) - Avg('price'))`
- `#{'price_diff': 46.85}`
  
- `from django.db.models import Count`
- `Product.objects.annotate(num_product=Count('price'))`
- `Product.objects.aggregate(Count('price'))`
- `Product.objects.aggregate(Count( 'price' , distinct=True))` # 不計算重覆的

▼ 表 4-5：查詢範例說明

想要達成的目標	查詢寫法和執行結果
取出所有的資料內容	<pre>&gt;&gt;&gt; Product.objects.all() [&lt;Product: HTC Magic&gt;, &lt;Product: SONY Xperia Z3&gt;, &lt;Product: Samsung DUOS&gt;, &lt;Product: Nokia Xpress 5800&gt;, &lt;Product: Infocus M370&gt;]</pre>
找出已經沒有庫存的二手機	<pre>&gt;&gt;&gt; Product.objects.filter(qty=0) [&lt;Product: HTC Magic&gt;]</pre>
找出有庫存的二手機	<pre>&gt;&gt;&gt; Product.objects.exclude(qty=0) [&lt;Product: SONY Xperia Z3&gt;, &lt;Product: Samsung DUOS&gt;, &lt;Product: Nokia Xpress 5800&gt;, &lt;Product: Infocus M370&gt;]</pre>
找出價格低於 500 元的二手機	<pre>&gt;&gt;&gt; Product.objects.filter(price__lte=500) [&lt;Product: HTC Magic&gt;, &lt;Product: Nokia Xpress 5800&gt;]</pre>
算出價格低於 500 元的二手機有幾種	<pre>&gt;&gt;&gt; from django.db.models import Count &gt;&gt;&gt; Product.objects.filter(price__lte=500). aggregate(Count('qty')) {'qty__count': 2}</pre>
算出價格低於 800 元的共有幾支二手機庫存	<pre>&gt;&gt;&gt; from django.db.models import Sum &gt;&gt;&gt; Product.objects.filter(price__lte=800). aggregate(Sum('qty')) {'qty__sum': 3}</pre>

想要達成的目標	查詢寫法和執行結果
找出所有 SONY 的二手機	>>> Product.objects.filter(name__icontains='sony') [<Product: SONY Xperia Z3>]
找出庫存 1 支或 2 支的二手機	>>> Product.objects.filter(qty__in=[1,2]) [<Product: SONY Xperia Z3>, <Product: Samsung DUOS>, <Product: Nokia Xpress 5800>, <Product: Infocus M370>]
檢查庫存中是否有 SONY 的二手機	>>> Product.objects.filter(name__contains='SONY').exists() True

同樣都是查詢資料，使用 filter 會傳回一個串列，而 get 則是只值回一個唯一的值。如果在設定的條件下找不到任何資料，則 filter 會傳回一個空的串列，而 get 則會產生一個 DoesNotExist 的例外，如果設定的條件超過一個元素符合，get 也會產生例外。因此，get 通常都是用在明確知道該資料只有一筆的情況下才會使用，而且使用的時候也要以 try/except 做好例外處理的工作。也因此，大部份的情形下，筆者大都是使用 filter 來做資料的搜尋。

## 刪除與修改資料

#得到滿足條件的結果，然後 **delete** 就可以 (**危險操作**，正式場合操作務必謹慎)  
Product.objects.filter(name\_\_contains="abc").delete() # 刪除名稱中包含 "abc"的

#如下寫法也可，效果也是一樣的，Django實際只執行一條 SQL 語句。  
people = Product.objects.filter(name\_\_contains="abc")  
people.delete()

#批量更新，適用於 .all() .filter() .exclude() 等後面 (**危險操作**，正式場合操作務必謹慎)  
Product.objects.filter(name\_\_contains="abc").update(name='xxx')  
# 名稱中包含 "abc"的都改成 xxx  
Product.objects.all().delete() # 刪除所有 Product 記錄，極危險操作務必謹慎

#單個 object 更新，適合於 .get(), get\_or\_create(), update\_or\_create() 等得到的 obj，和新建類似，對QuerySet的查詢結果切片操作無效果  
twz = Product.objects.get(name= "pikachu")  
twz.name= "Cute Pikachu"  
twz.price=99999  
twz.save() # 保存即更新!!!

# QuerySet重複的問題，使用.distinct()去重複

```
qs1 = Product.objects.filter(name__contains='x')
```

```
qs2 = Product.objects.filter(price__gt=100)
```

```
qs3 = Product.objects.filter(qty__le=5)
```

# 合併到一起，這個時候就有可能出現重複的

```
qs = qs1 | qs2 | qs3 #union
```

# 去重複方法

```
qs = qs.distinct()
```

## 小練習

➡ 請在資料庫中新增以下資料

```
Product.objects.create(sku='a01',name='尼多蘭',price=500,size='S',qty=5)
Product.objects.create(sku='a02',name='尼多娜',price=600,size='S',qty=3)
Product.objects.create(sku='a03',name='尼多后',price=800,size='S',qty=2)
Product.objects.create(sku='a04',name='尼多朗',price=550,size='S',qty=7)
Product.objects.create(sku='a05',name='尼多力諾',price=650,size='S',qty=4)
Product.objects.create(sku='a06',name='尼多王',price=1300,size='S',qty=2)
```

<https://docs.djangoproject.com/en/dev/topics/db/>

<https://docs.djangoproject.com/en/2.2/topics/db/models/>

<https://docs.djangoproject.com/en/2.2/topics/db/queries/>

<https://docs.djangoproject.com/en/2.2/topics/db/aggregation/>

## 小練習

- 新增上述5樣資料後，進行查詢練習
- 請列出所有名稱中是'尼多'開頭的
- 請列出所有名稱中是'尼多'開頭的，且價格是(含)800以下的
- 請列出所有名稱中是'尼多'開頭的，但不要'尼多娜'
- 請列出所有名稱中是'尼多'開頭的，價格總合和平均
- 請列出所有名稱中是'尼多'開頭的，還有名稱中有'龍'字的
- 請列出所有名稱中是'尼多'開頭的，或是價格高於1000

## 補充-備份DB資料

➤ 備份資料

➤ `python manage.py dumpdata [app name] > appname_data.json`

➤ ex:`python manage.py dumpdata mysite > mysite_data.json`

➤ 還原資料(注意json檔的編碼需為utf-8)

➤ `python manage.py loaddata mysite_data.json`



## View簡介

- 是Django最重要的程式邏輯所在的地方
- 網站大部份的程式設計都放在這邊
- 對初學者來說，這裡放了許多我們要操作資料，以及安排哪些資料需要被顯示出來的函數
- 在函數中把這些資料傳遞給網頁伺服器或是先交由Template的渲染器之後再送到網頁伺服器中
- 這些放在views.py中的函數，再由urls.py中的設計做對應和派發

# 建立簡易的HttpResponse網頁

- 直接顯示資料到網頁的步驟是
  - 先到`urls.py`設定一個網址的對應
    - `path('about/', about),`
  - 然後在`views.py`中編寫一個函數，透過HttpResponse傳遞出想要顯示的資料
  - `from django.http import HttpResponse`
    - `def about(request):`
      - `html = "<!DOCTYPE html>`
      - `<html><head><title>About Myself</title></head>`
      - `<body><h2>Andy Wood</h2><hr>`
      - `<p>Hi, I am Andy Wood. Nice to meet you!</p>`
      - `</body>`
      - `</html>`
      - `""`
      - `return HttpResponse(html)`

# Andy Wood

---

Hi, I am Andy Wood. Nice to meet you!

## 在views.py顯示查詢資料列表

- 示範的是如何在views.py中查詢在models.py中定義且已儲存的資料，並顯示在使用者端的網頁上。設定的網址是 [localhost:8000/list](http://localhost:8000/list)，而顯示出來的結果如下圖



中古機列表 - Mozilla Firefox

中古機列表

192.168.161.131:8000/list/

以下是目前本店販售中的二手機列表

品名	售價	庫存量
HTC Magic	100	0
SONY Xperia Z3	15000	1
Samsung DUOS	800	2
Nokia Xpress 5800	500	1
Infocus M379	1500	2

# 在views.py顯示查詢資料列表

- ▶ views.py建立一個函數listing
  - ▶ `from django.http import HttpResponse`
  - ▶ `from mysite.models import Product`
  - ▶ `def about(request):`
    - ▶ `html = ""`
    - ▶ About Myself
    - ▶ `""`
    - ▶ `return HttpResponse(html)`
  - ▶ `def listing(request):`
    - ▶ `html = ""`
    - ▶ `<!DOCTYPE html>`
    - ▶ `<html>`
    - ▶ `<head>`
      - ▶ `<meta charset='utf-8'>`
      - ▶ `<title>中古機列表</title>`
    - ▶ `</head>`
    - ▶ `<body>`
      - ▶ `<h2>以下是目前本店販售中的二手機列表</h2>`

## 在views.py顯示查詢資料列表

- ▶ `<hr>`
- ▶ `<table width=400 border=1 bgcolor='#ccffcc'>`
- ▶ `{`
- ▶ `</table>`
- ▶ `</body>`
- ▶ `</html>`
- ▶ `'''`
- ▶ `products = Product.objects.all()`
- ▶ `tags = '<tr><td>品名</td><td>售價</td><td>庫存量</td></tr>'`
- ▶ `for p in products:`
- ▶ `tags = tags + '<tr><td>{</td>'.format(p.name)`
- ▶ `tags = tags + '<td>{</td>'.format(p.price)`
- ▶ `tags = tags + '<td>{</td></tr>'.format(p.qty)`
- ▶ `return HttpResponse(html.format(tags))`

## 在views.py顯示查詢資料列表

- 在urls.py中，加入對於listing函數的import以及URL的對應：
  - `from django.contrib import admin`
  - `from django.urls import path`
  - `from mysite.views import about, listing`
- `urlpatterns = [`
- `path('admin/', admin.site.urls),`
- `path('about/', about),`
- `path('list/', listing),`
- `]`

## 小練習

- ▶ 請讓排列出來的資料
  - ▶ 依價格由低到高排列
  - ▶ 依價格由高到低排列
  - ▶ 依庫存排列
  - ▶ 不要印出庫存為0的資料
  - ▶ .....



## Tempalte簡介

- 相信對於HTML內容一定很頭大
- 難道要顯示一個網頁非要弄到這麼麻煩嗎？當然不是
- 要建立專業網站，一定要使用進階功能的樣板渲染方法才行。也就是把HTML檔案另外存成樣板檔案
- 然後把想要顯示在網頁的資料另外以變數的方式傳遞給渲染器，讓渲染器根據變數的內容和指定的樣板檔案做整合，
- 再把結果輸出給網頁伺服器，本節即說明如何使用Template來建立專業的網站

## 建立template資料夾與檔案

- ▶ 在使用之前，要先在網頁中建立放置樣板檔案的資料夾，並在[settings.py](#)中設定此資料夾的存取住址

```
▶ TEMPLATES = [  
▶     {  
▶         'BACKEND': 'django.template.backends.django.DjangoTemplates',  
▶         'DIRS': [os.path.join(BASE_DIR, 'templates')],  
▶         'APP_DIRS': True,  
▶         'OPTIONS': {  
▶             'context_processors': [  
▶                 'django.template.context_processors.debug',  
▶                 'django.template.context_processors.request',  
▶                 'django.contrib.auth.context_processors.auth',  
▶                 'django.contrib.messages.context_processors.messages',  
▶             ],  
▶         },  
▶     },  
▶ ]
```

# 建立template資料夾與檔案

- 在templates資料夾中建立一個about.html檔案，如下所示：

- <!-- about.html -->
- <!DOCTYPE html>
- <html>
- <head>
- <meta charset='utf-8'>
- <title>About Myself</title>
- </head>
- <body>
- <h2>Andy Wood</h2>
- <hr>
- <p>
- Hi, I am Andy Wood. Nice to meet you!
- </p>
- <em>今日佳句 : {{ quote }}</em>
- </body>
- </html>

## 傳遞變數到template檔案中

➤ Views.py

➤ def about(request):

➤ quotes = ['今日事，今日畢',

➤ '要怎麼收穫，先那麼栽',

➤ '知識就是力量',

➤ '一個人的個性就是他的命運']

➤ quote = random.choice(quotes)

➤ return render(request, 'about.html', locals())

## 傳遞變數到template檔案中

➤ Views.py

➤ 在listing函數中要顯示全部的資料項目列表

➤ `def listing(request):`

➤ `products = Product.objects.all()`

➤ `return render(request, 'list.html', locals())`

# 在template中處理串列變數

➤ `list.html` · 如下所示：

```
<!-- list.html -->
<!DOCTYPE html>
<html>
<head>
<meta charset='utf-8'>
<title>中古機列表</title>
</head>
<body>
<h2>以下是目前本店販售中的二手機列表</h2>
<hr>
<table width=400 border=1 bgcolor='#ccffcc'>
  <tr><td>品名</td><td>售價</td><td>庫存量</td></tr>
  {% for p in products %}
  <tr>
    <td>{{p.name}}</td>
    <td>{{p.price}}</td>
    <td>{{p.qty}}</td>
  </tr>
  {% endfor %}
</table>
</body>
</html>
```

- Views.py
- from django.http import Http404
- #from django.http import HttpResponseRedirect
  
- def disp\_detail(request, sku):
- try:
- p = Product.objects.get(sku=sku)
- except Product.DoesNotExist:
- raise Http404('找不到指定的品項編號')
- #return HttpResponseRedirect('找不到指定的品項編號')
- #return HttpResponseRedirect('<h1>Page not found</h1>')
  
- return render(request, 'disp.html', locals())

# 傳遞變數到template檔案中

► 同樣的方法亦可馬上套用到disp\_detail函數中。也是先建立一個disp.html，如下所示：

```
► <!-- disp.html -->
► <!DOCTYPE html>
► <html>
► <head>
► <meta charset='utf-8'>
► <title>{{p.name}}</title>
► </head>
► <body>
► <h2>{{p.name}}</h2>
► <hr>
► <table width=400 border=1 bgcolor='#ccffcc'>
► <tr><td>品項編號</td><td>{{p.sku}}</td></tr>
► <tr><td>品項名稱</td><td>{{p.name}}</td></tr>
► <tr><td>二手售價</td><td>{{p.price}}</td></tr>
► <tr><td>庫頁數量</td><td>{{p.qty}}</td></tr>
► </table>
► <a href='/list'>回列表</a>
► </body>
► </html>
```



# 小練習

- 請新增一頁面如下表：
- 並自行修改 `urls.py` 進行頁面的對應
  - (提示) `path('list/<sku>/', disp_detail)`,
  - 或是 `django 1.x` 的語法
  - `django.conf.urls import include, url`
  - `url(r'^list/(.*)$', disp_detail)`,
- 新增 `/list/` 頁面中的品名 可以超連結到上述頁面
  - (提示) `<td><a href='/list/{{p.sku}}'>{{p.name}}</a></td>`

## GrayBox

品項編號	0001
品項名稱	GrayBox
二手售價	100
庫頁數量	10

[回列表](#)

以下是目前本店販售中的二手機列表

品名	售價	庫存量
<a href="#">GrayBox</a>	100	10
<a href="#">abc</a>	200	2

## 回家作業

- 新依照第二章的教學完成一首頁
- 可以有 list 與 about 的連結
- 並可以隨機產生今日佳句

