Python

- 【Python】安裝psycopg2錯誤
- 【Python】刪除 window 下執行時產生的暫存檔
- 【Python】登入網路設備擷取資訊
- 【Python】程式練習-ZeroJudge網站
- 【Python】解析Python模組(Module)和套件(Package)的概念(轉)
- 【Python】psycopg2 防止SQL injection(轉)
- 【Python】Pyenv 版本管理工具
- 【Python】python讀取json
- 【Python】selenium自動播放flash
- 【Python】selenium性能優化
- 【Python】【PrettyTable】 資料格式化排版工具
- 【Python】【tabulate】 資料格式化排版工具
- 【Python】【環境建置】venv 虛擬環境建置
- 【 Python】常用自訂函數
- 【Python】import 用法
- 【Python】【__init__.py】,【 __all__】說明

【Python】安裝psycopg2錯誤

```
[root@proxy ~]# yum -y install  gcc gcc-c++ make
[root@proxy ~]# pip3 install psycopg2
Error: pg_config executable not found.

#找到pg_config 真實路徑(有可能自訂安裝路徑不同)
[root@proxy ~]export PATH=/usr/pgsql-12/bin/:$PATH

[root@proxy ~]# yum install -y postgresql-devel
./psycopg/psycopg.h:35:20: fatal error: Python.h: No such file or directory
 #include <Python.h>
                   ^
compilation terminated.

[root@proxy ~]# yum install -y python36-devel

[root@proxy ~]# pip3 install psycopg2
```

【Python】刪除 window 下執行時產生的暫存檔

```
for /f "delims=" %F in ('Dir /B /S .\*.py ^|findstr /IE "\\migrations\\[^\\]*.py"^|findstr /IEV "\\__init__.py" ') Do @echo del "%F"
```

【Python】登入網路設備擷取資訊

- https://www.cnblogs.com/guxh/p/12375801.html
- https://www.cnblogs.com/guxh/p/9831226.html

netmiko

- https://www.csdn.net/tags/MtTaEgxsNDk2NDIzLWJsb2cO0O0O.html
- https://zhuanlan.zhihu.com/p/367962211
- 最强Netmiko攻略疑□□症篇 - 知乎 (zhihu.com)

【Python】程式練習-ZeroJudge網站

ZeroJudge網站

http://zerojudge.tw/

適合所有中學生及初學者的 Online Judge 系統

AC (Accept): 即表示通過
NA (Not Accept): 在多測資點的題目中若未通過所有測資點則出現 NA
WA (Wrong Answer): 表示答案錯誤，並在訊息中指出錯誤行數及正確答案
TLE (Time Limit Exceed): 表示執行超過時間限制
MLE (Memory Limit Exceed): 表示程序執行超過記憶體限制
OLE (Output Limit Exceed): 表示程序輸出檔超過限制
RE (Runtime Error): 表示執行時錯誤，通常為記憶體配置錯誤 如：使用了超過陣列大小的位置
RF (Restricted Function): 表示使用了被禁止使用的函式，並在錯誤訊息中指明使用了什麼不合法的函式。
CE (Compile Error): 表示編譯錯誤，並在訊息中列出完整錯誤訊息，以利判斷。 關於編譯器
SE (System Error): 包含 Compile, Runtime 等未定義錯誤均屬於 System Error

【Python】解析Python模組(Module)和套件(Package)的概念(轉)

當我們在開發大型應用程式時，如果沒有適當的組織程式碼，除了會降低開發的效率外，也不易於維護，所以模組(Module)化就顯得相當的重要，讓程式碼能夠透過引用的方式來重複使用，提升重用性(Reusable)。

但是隨著專案模組(Module)的增加，將難以管理及問題的追蹤，這時候就能將模組(Module)打包成套件(Package)，利用其階層式的結構來彈性規劃模組(Module)。

本篇文章就帶大家瞭解Python模組(Module)及套件(Package)的重要觀念，包含：

什麼是模組(Module)
模組引用方式(Import)
什麼是套件(Package)
dir()函式(dir function)

將模組當作腳本來執行(Executing a Module as a Script)
一、什麼是模組(Module)
模組(Module)就是一個檔案，包含了相關性較高的程式碼。隨著應用程式的開發規模越來越大，我們不可能把所有的程式碼都寫在同一份Python檔案中，一定會將關聯性較高的程式碼抽出來放在不同的檔案中來形成模組(Module)，主程式再透過引用的方式來使用。所以模組(Module)可以提高程式碼的重用性(Reusable)且易於維護。

假設我們現在要開發一個部落格，主程式為 app.py ，在還沒有模組化時，程式碼可能長得像這樣：

```
#取得作者
def get_author():
    return "Mike"
#取得電子郵件
def get_email():
    return "learncodewithmike@gmail.com"
#新增文章
def add_post(title):
    pass
#刪除文章
def delete_post(title):
    pass
add_post()
author = get_author()
email = get_email()
```

以此範例來說，取得作者及電子郵件可以獨立出來建立一個關於模組(about.py)，而新增及刪除文章則可以獨立出來為文章模組(post.py)，專門處理文章相關的動作，如下範例：
about.py

```
#取得作者
def get_author():
return "Mike"
#取得電子郵件
def get_email():
return "learncodewithmike@gmail.com"
post.py

#新增文章
def add_post(title):
pass
#刪除文章
def delete_post(title):
pass
```

post.py

```
#新增文章
def add_post(title):
    pass
```

```
#刪除文章
def delete_post(title):
    pass
```

當然，模組(Module)除了可以包含函式(Function)外，也可以為類別(Class)，我們以 post.py 為例：

```
class Post:
    # 建構式
    def __init__(self):
        self.titles = []
    # 新增文章
    def add_post(self, title):
        self.titles.append(title)
    # 刪除文章
    def delete_post(self, title):
        self.titles.remove(title)
```

所以現在我們專案中有一個主程式 app.py 及兩個模組(Module)，分別為 about.py 和 post.py。

二、模組引用方式(Import)
我們將程式碼進行模組化後，主程式 app.py 要如何使用呢?首先，可以使用 from-import 語法，如下範例：

```
# 引用模組中的特定物件
from post import Post
from about import get_author, get_email
p = Post()
p.add_post("Python Programming")
author = get_author()
email = get_email()
print(p.titles)  #執行結果：['Python Programming']
print(author)  #執行結果：Mike
print(email)  #執行結果：learncodewithmike@gmail.com
```

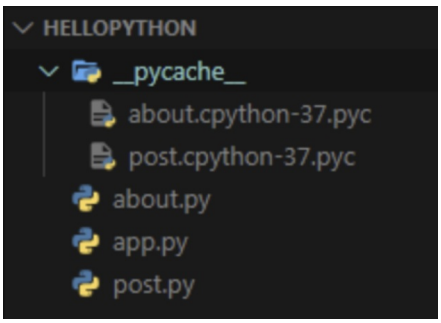
from 之後加上模組(Module)的檔名，注意沒有 .py 副檔名，接著 import 引用所需的物件。

當然，在 import 之後可以使用 * 來引用模組中的所有物件，但是這樣的寫法，可能會在引用的過程中，發生同名方法覆寫(Method Overriding)的風險，所以建議引用所需要的物件即可。另一種語法則是透過 import 語法，如下範例：

```
# 引用整個模組
import post
import about
p = post.Post()
p.add_post("Python Programming")
author = about.get_author()
email = about.get_email()
print(p.titles) #執行結果：['Python Programming']
print(author) #執行結果：Mike
print(email) #執行結果：learncodewithmike@gmail.com
```

import 之後加上模組(Module)的檔名，和上一個語法不一樣的地方是，此語法雖然引用整個模組(Module)，但是在主程式中必須透過模組(Module)的名稱來存取其中的成員。

在主程式 app.py 中引用模組(Module)，並且執行後，會發現多了一個 pycache 資料夾，如下圖：



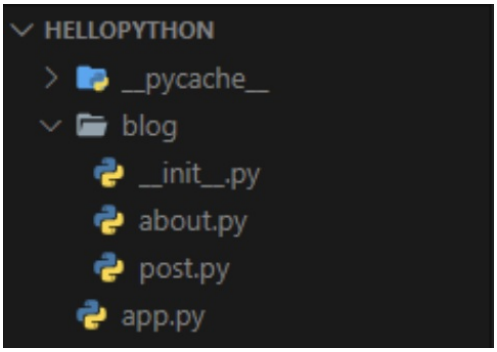
這個資料夾中，可以看到包含了引用模組的已編譯檔案，當下一次執行主程式 app.py 時，Python編譯器看到已編譯的模組檔案，會直接載入該模組(Module)，而省略編譯的動作，藉此來加速載入模組(Module)的速度。

當然Python編譯器在每一次執行時，會檢查來源模組及已編譯檔案的時間，當來源模組的時間較新，則代表該模組(Module)有經過修改，則Python編譯器會再編譯一次，更新已編譯檔案。

各位有沒有覺得奇怪，那為什麼沒有 app.py 的已編譯檔案，因為在此範例中，我們將 app.py 當作程式的進入點，所以每一次執行 python app.py 指令時，Python編譯器都要進行編譯，所以沒有將 app.py 進行快取的動作。
三、什麼是套件(Package)
就是一個容器(資料夾)，包含了一個或多個的模組(Module)，並且擁有__init__.py檔案，其中可以撰寫套件(Package)初始化的程式碼。

我們將程式碼模組化後，專案中的模組(Module)就會越來越多，這時候就可以再將相似的模組(Module)組織為套件(Package)。那要如何建立套件(Package)呢?

舉例來說，現在我們要將專案中的 post.py 及 about.py 模組(Module)打包為部落格套件(Package)，首先，建立blog資料夾，接著在資料夾中新增__init__.py檔案，最後將 post.py 及 about.py 模組(Module)移至blog資料夾中，如下範例：



而在主程式 app.py 中引用套件(Package)的方式和模組(Module)大同小異，我們先來看 from-import語法，如下範例：

```
# 從套件中引用模組
from blog import post
from blog import about
p = post.Post()
p.add_post("Python Programming")
author = about.get_author()
email = about.get_email()
```

另一個引用套件(Package)的 import 語法如下範例：

```
# 從套件中引用模組
import blog.post
import blog.about
p = blog.post.Post()
p.add_post("Python Programming")
author = blog.about.get_author()
email = blog.about.get_email()
```

四、dir()函式(dir function)
Python提供了一個內建函式dir()，用來顯示物件(Object)的屬性(Attribute)及方法(Method)，我們利用此函式(Function)來看一下模組(Module)所擁有的屬性(Attribute)及方法(Method)，如下範例：

```
#從blog套件引用about模組
from blog import about
print(dir(about))
```

從執行結果可以看到模組(Module)中有自建的get_author及get_email方法(Method)，其餘的則是Python自動幫我們產生的，我們來看幾個常用的屬性(Attribute)，如下範例：

```
# 從blog套件引用about模組
from blog import about
print(about.__name__) # 模組名稱
print(about.__package__) # 套件名稱
print(about.__file__) # 模組的檔名及路徑
```

五、將模組當作腳本來執行(Executing a Module as a Script)

我們來看一個範例，在about模組(Module)中加上以下程式碼，並且執行該模組(Module)，如下範例：

```
#取得作者
def get_author():
return "Mike"
#取得電子郵件
def get_email():
return "learncodewithmike@gmail.com"
print("about module name: ", __name__)
# 執行結果：about module name: __main__
```

而這時候換成執行 app.py ，__name__屬性(Attribute)則為blog.about，我們就可以利用這個特性，撰寫腳本來彈性的控制當執行模組(Module)的檔案時，要進行哪些行為，而這些行為是在被其他模組(Module)引用時，不會被執行的，如下範例：

```
#取得作者
def get_author():
return "Mike"
#取得電子郵件
def get_email():
return "learncodewithmike@gmail.com"
if __name__ == "__main__":
print("about module initialized.")
get_author()
```

範例中將about模組(Module)加上了判斷式，當執行about模組(Module)時，__name__屬性(Attribute)為__main__，所以會執行我們設定的任務，而這些任務是在執行主程式 app.py 時，不會被執行的，因為__name__屬性(Attribute)為blog.about。

六、小結
以上就是Python模組(Module)及套件(Package)的重要觀念，除了能夠提高程式碼的重用性(Reusable)外，也有利於未來的單元測試及維護。

【Python】psycopg2 防止SQL injection(轉)

使用Python防止SQL注入攻_似繁星跌入梦的博客-CSDN博客_python防止sql注入

```
from psycopg2 import sql

def count_rows(table_name: str, limit: int) -> int:
    with connection.cursor() as cursor:
        stmt = sql.SQL("""
            SELECT
                COUNT(*)
            FROM (
                SELECT
                    1
                FROM
                    {table_name}
                LIMIT
                    {limit}
            ) AS limit_query
        """).format(
            table_name = sql.Identifier(table_name),
            limit = sql.Literal(limit),
        )
        cursor.execute(stmt)
        result = cursor.fetchone()

    rowcount, = result
    return rowcount
```

【Python】Pyenv 版本管理工具

來源：[Python 教學] 如何切換 Python 版本，讓 Pyenv 幫你輕鬆管理版本 | Max行銷誌 (maxlist.xyz)

安裝

```
# 安裝相關套件
$ brew update
$ brew install openssl readline sqlite3 xz zlib
$ brew install pyenv
```

```
# 加入啟動環境變數
$ echo 'export PYENV_ROOT="$HOME/.pyenv"' >> ~/.zshrc
$ echo 'command -v pyenv >/dev/null || export PATH="$PYENV_ROOT/bin:$PATH"' >> ~/.zshrc
$ echo 'eval "$(pyenv init -)"' >> ~/.zshrc
```

常用指令

```
# 查看□前版本
pyenv version

# 查看所有版本
pyenv versions

# 查看所有可安装的版本
pyenv install --list

# 安装指定版本
pyenv install 3.6.5
# 安装新版本后rehash一下
pyenv rehash

# 删除指定版本
pyenv uninstall 3.5.2

# 指定全局版本
pyenv global 3.6.5

# 指定多个全局版本, 3版本优先
pyenv global 3.6.5 2.7.14

# 实际上□你切□版本后, 相□的pip和包□□都是会自动切□□去的

# 切回系統預設版本
pyenv global system
```

使用`pyenv-virtualenv`建立虛擬環境

如果你希望使用虛擬環境，可以安裝並使用`pyenv-virtualenv`：

1. **安裝`pyenv-virtualenv`**：

   ```
   brew install pyenv-virtualenv
   ```

   將以下內容新增至您的`~/.bashrc`或`~/.zshrc`文件：

   ```
   eval "$(pyenv virtualenv-init -)"
   ```

   然後重新載入shell設定檔：

   ```
   source ~/.bashrc  # 如果使用 bash
   source ~/.zshrc   # 如果使用 zsh
   ```

2. **創建Python 2虛擬環境：**

   ```
   pyenv virtualenv 2.7.18 my-virtual-env
   ```

3. **啟動虛擬環境：**

   ```
   pyenv activate my-virtual-env
   ```

4. **失效虛擬環境：**

   ```
   pyenv deactivate
   ```

透過上述步驟，您可以在 macOS 上使用`pyenv`安裝和管理 Python 2 版本，並在不同的專案之間輕鬆切換。

【Python】python讀取json

在 Python 中，JSON 是一種常用的資料格式，用來儲存和交換資料。Python 提供內建的 `json` 模組來讀取與處理 JSON 資料，支援從檔案或字串中解析 JSON，以及將資料轉換成 JSON 格式輸出。

以下是 Python 中讀取 JSON 的相關用法：

---

1. Python 的 `json` 模組介紹

Python 的 `json` 模組提供以下幾個主要功能：

- `json.load()`：從檔案中讀取 JSON 資料。
- `json.loads()`：從字串中解析 JSON 資料。
- `json.dump()`：將 Python 資料寫入檔案，並格式化為 JSON。
- `json.dumps()`：將 Python 資料轉換為 JSON 格式的字串。

---

2. 讀取 JSON 的方式

2.1 從檔案讀取 JSON

假設有一個 `data.json` 檔案，內容如下：

```
{
  "name": "Alice",
  "age": 25,
  "skills": ["Python", "JavaScript"]
}
```

用法：

```
import json

# 打開 JSON 檔案並讀取
with open('data.json', 'r', encoding='utf-8') as file:
  data = json.load(file)

# 輸出讀取的資料
print(data)          # {'name': 'Alice', 'age': 25, 'skills': ['Python', 'JavaScript']}
print(data['name'])  # Alice
```

2.2 從字串解析 JSON

如果 JSON 資料是以字串的形式存在，可以使用 `json.loads()` 來解析：

```
import json

# JSON 字串
json_string = '{"name": "Bob", "age": 30, "skills": ["Java", "C++"]}'

# 解析 JSON 字串
data = json.loads(json_string)

# 輸出讀取的資料
print(data)          # {'name': 'Bob', 'age': 30, 'skills': ['Java', 'C++']}
print(data['skills']) # ['Java', 'C++']
```

---

3. JSON 資料寫入檔案

3.1 寫入 JSON 到檔案

使用 `json.dump()` 將 Python 資料寫入 JSON 檔案：

```
import json

# Python 資料
data = {
    "name": "Charlie",
    "age": 35,
    "skills": ["Go", "Rust"]
}

# 寫入 JSON 檔案
with open('output.json', 'w', encoding='utf-8') as file:
    json.dump(data, file, ensure_ascii=False, indent=4)

print("JSON 資料已成功寫入檔案")
```

- **`ensure_ascii=False`**：避免非 ASCII 字元（例如中文）被編碼為 Unicode（如 `\u4e2d\u6587`）。
- **`indent=4`**：設定縮排，讓輸出的 JSON 更易於閱讀。

輸出的檔案內容：

```
{
    "name": "Charlie",
    "age": 35,
    "skills": [
        "Go",
        "Rust"
    ]
}
```

3.2 將 Python 資料轉換為 JSON 字串

使用 `json.dumps()` 將資料轉換為 JSON 格式字串：

```
import json

# Python 資料
data = {
    "name": "Daisy",
    "age": 28,
    "skills": ["HTML", "CSS"]
}

# 轉換為 JSON 字串
json_string = json.dumps(data, ensure_ascii=False, indent=2)
print(json_string)
```

輸出：

```
{
  "name": "Daisy",
  "age": 28,
  "skills": [
    "HTML",
    "CSS"
  ]
}
```

---

4. 常見的 JSON 資料類型對應

Python 與 JSON 的資料類型有一對一的對應關係：

JSON 類型	Python 類型
Object	Dictionary (`dict`)
Array	List (`list`)
String	String (`str`)
Number (int/float)	Integer (`int`)/Float (`float`)
Boolean	Boolean (`bool`)
null	None

範例：

```
json_string = '''
{
   "name": "Eve",
   "age": 40,
   "is_active": true,
   "children": null
}
'''

data = json.loads(json_string)
print(type(data))          # <class 'dict'>
print(data['is_active'])   # True (Python 的布林值)
print(data['children'])    # None (Python 的 None)
```

---

5. 捕捉 JSON 解析錯誤

處理 JSON 時，可能會遇到格式不正確的情況，可以透過 `try-except` **捕捉例外：**

```
import json

json_string = '{"name": "Frank", "age": 45,'  # 少了結尾的 }

try:
   data = json.loads(json_string)
except json.JSONDecodeError as e:
   print(f"JSON 解析錯誤: {e}")
```

輸出：

```
JSON 解析錯誤: Expecting property name enclosed in double quotes: line 1 column 30 (char 29)
```

---

6. 自訂 JSON 編碼與解碼

6.1 處理非內建類型

如果你需要將自定義類型的資料轉換為 JSON，可以透過繼承 `json.JSONEncoder` **來實現：**

範例：

```
import json

class Person:
```

```
    def __init__(self, name, age):
        self.name = name
        self.age = age

class PersonEncoder(json.JSONEncoder):
    def default(self, obj):
        if isinstance(obj, Person):
            return {"name": obj.name, "age": obj.age}
        return super().default(obj)

person = Person("Grace", 50)
json_string = json.dumps(person, cls=PersonEncoder, indent=4)
print(json_string)
```

輸出：

```
{
    "name": "Grace",
    "age": 50
}
```

6.2 自訂解碼器

你也可以自訂解碼器將 JSON 資料轉換為自定義的 Python 類別：

```
import json

class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

def decode_person(dct):
    if "name" in dct and "age" in dct:
        return Person(dct['name'], dct['age'])
    return dct

json_string = '{"name": "Henry", "age": 60}'
person = json.loads(json_string, object_hook=decode_person)
print(type(person))  # <class '__main__.Person'>
print(person.name)   # Henry
```

結論

- **使用** `json.load()` **和** `json.loads()` **來解析 JSON 檔案或字串。**
- **使用** `json.dump()` **和** `json.dumps()` **將 Python 資料輸出為 JSON 格式。**
- **可以透過** `ensure_ascii` **和** `indent` **參數來控制 JSON 的輸出格式。**
- **若需處理自定義類型，可使用自訂的編碼器和解碼器。**
- **遇到格式錯誤時，使用** `try-except` **捕捉** `json.JSONDecodeError`**。**

這些功能幾乎涵蓋了日常開發中處理 JSON 資料的需求，讓你能更高效地處理結構化數據！

---

json to distionary

```
import json

# json 的資料形式字串
x =  '{ "name":"jim", "age":25, "city":"Taiwan"}'

# 轉換json
person = json.loads(x)
```

```
print(type(person)) #<class 'dict'>
print(person){'name': 'jim', 'age': 25, 'city': 'Taiwan'}
print(person['age']) #25
```

distionary to json

```
import json

person = {'name': 'jim', 'age': 25, 'city': 'Taiwan'}

data = json.dumps(person)

print(type(data)) #<class 'str'>
print(data) #{"name": "jim", "age": 25, "city": "Taiwan"}
```

【Python】selenium自動播放flash

python selenium firefox 控制devtools 一些□索_wujiuqier的博客-CSDN博客

軟體載點

- https://ftp.mozilla.org/pub/mozilla.org/firefox/releases/
- https://github.com/mozilla/geckodriver/releases

後來測試可用的組合為

python 3.7.0
firefox 57.0.4
geckodriver 0.19.1
selenium 3.141.0

版本55或以上的Firefox不支持Flash自动播放。
建□使用Firefox 52.9.0 延长支持版。
此版本需要使用Gecko Driver 0.18.0才能正常使用。但是不支持 set_window_size（会□□ selenium.common.exceptions.WebDriverException: Message: setWindowRect），所以必□手动□整视口尺寸。 □□使用 find_element_by_id(element_id).screenshot(output_file_name)的方法□截长□，或者也可以用 find_elements_by_tag_name('body')[0].screenshot(output_file_name)的方法□截整页长□。
自动播放Flash的设置（参考https://blog.csdn.net/STL_CC/article/details/104968669）：

此版本□□有移除GCLI□发者工具□，可以通□快捷□ `Shift + F2` 呼出。
□整视口尺寸的GCLI命令是 `resize to 4320 7680` 。第一个参□是□，第二个参□是高。
截□的命令是

既然不能控制视口，那就直接控制外□ `div` 的样式，然后□行元素□点截□。直接用 `execute_script` 即可。
□上，截取小花仙人物面板形象的操作有：

```
from selenium import webdriver
from time import sleep
fp = webdriver.FirefoxProfile()
fp.set_preference("plugin.state.flash", 2)
fp.set_preference("security.insecure_field_warning.contextual.enabled", False)
driver = webdriver.Firefox(firefox_profile = fp)
driver.get("http://hua.61.com/play.shtml")
driver.execute_script("document.getElementById('flashContent').innerHTML=\"<embed src='http://hua.61.com/(
# 打开玩家面板
driver.execute_script("document.getElementById('flashContent').style.width='4320px';document.getElementBy1
sleep(5)
driver.execute_script("document.getElementsByTagName('embed')[0].Zoom(500)")
driver.execute_script('document.getElementsByTagName("embed")[0].Zoom(20)')
sleep(5)
# Flash缩放后的视野位置微调，还需要完善
driver.execute_script('document.getElementsByTagName("embed")[0].Pan(-4500,-2000,0)')
sleep(5)
driver.find_element_by_id('flashContent').screenshot('output.png')
driver.execute_script("document.getElementsByTagName('embed')[0].Zoom(500)")
driver.execute_script("document.getElementById('flashContent').style.height='100%';document.getElementByIc

```

```
from time import sleep
from selenium import webdriver
import win32api
import win32gui
import win32con
import os
import requests
import sys
import time
```

```
from PIL import ImageGrab
import win32com.client

login_info = [
    {
        "file_name": "db1.jpeg",
        "url": "https://1.2.3.4/em/console/database/instance/waitDetails?
event=doLoad&target=mall&type=oracle_database&waitClass=Overview&datasource=SQL",
        "username": "aa",
        "title_id": None,
        "password": "aaaaaaaa",
        "hwnd": None,
        "driver": None},
    {
        "file_name": "db2.jpeg",
        "url": "https://1.2.3.4/em/faces/sdk/nonFacesWrapper?
target=reportdb&_em.coBM=%2Fconsole%2Fdatabase%2Finstance%2FwaitDetails%3Fevent%26target%3Dreportdb%26type%3Doracle

        "username": "bb",
        "password": "bbbbbbbb",
        "title_id": None,
        "j_username": "cc",
        "j_password": "cccccccc",
        "hwnd": None,
        "driver": None}
]


def api_upload(filename):
    url = "http://192.168.1.1/api/v1/db/uploadIMG.php"
    local_folder_path = getRootPath()  # 取得目前執行檔目錄
    image = local_folder_path + "\\" + filename
    files = {"file": (filename, open(image, "rb"), "rb")}
    response = requests.request("POST", url, data=None, files=files)
    # print(response.text)


def getRootPath():
    return os.path.dirname(os.path.abspath(__file__))


def main():
    for config in login_info:
        try:
            profile = webdriver.FirefoxProfile()
            profile.set_preference('plugin.state.flash', 2)
            profile.set_preference('security.tls.version.min', 1)
            profile.set_preference('security.insecure_field_warning.contextual.enabled', False)
            if config["driver"] == None:
                driver = webdriver.Firefox(executable_path=r'C:\screenshot\geckodriver.exe', firefox_profile=profile)
                config["driver"] = driver
                driver.get(config["url"])

                if config["file_name"] == 'mallDB.jpeg':
                    username_input = driver.find_element_by_id("M__Id")
                    username_input.send_keys(config["username"])
                    password_input = driver.find_element_by_id("M__Ida")
                    password_input.send_keys(config["password"])
                    driver.find_element_by_xpath('//table/tbody/tr[4]/td[3]/a/img').click()
                    sleep(7)
                    config["title_id"] = win32gui.GetForegroundWindow()
                elif config["file_name"] == 'reportDB.jpeg' or config["file_name"] == 'wmsDB.jpeg' or config["file_name"] == 'scmDB.jpeg':
                    j_username_input = driver.find_element_by_id("j_username::content")
                    j_username_input.send_keys(config["j_username"])
                    j_password_input = driver.find_element_by_id("j_password::content")
                    j_password_input.send_keys(config["j_password"])
                    login_btn = driver.find_element_by_id("login")
                    login_btn.click()
                    sleep(5)
```

```
                username_input2 = driver.find_element_by_id("emT:r1:0:r1:0:username::content")
                username_input2.send_keys(config["username"])
                password_input2 = driver.find_element_by_id("emT:r1:0:r1:0:password::content")
                password_input2.send_keys(config["password"])
                login_btn_emT = driver.find_element_by_id("emT:logon")
                login_btn_emT.click()
                sleep(5)
                config["title_id"] = win32gui.GetForegroundWindow()

            title_id = win32gui.GetWindowText(config["title_id"])
            hwnd = win32gui.FindWindow(None, title_id)
            shell = win32com.client.Dispatch("WScript.Shell")
            shell.SendKeys('%')
            win32gui.ShowWindow(hwnd, win32con.SW_MAXIMIZE)
        config["driver"].save_screenshot(config["file_name"])
        api_upload((config["file_name"]))
        sleep(3)

    except Exception as e:
        print("exception:", e)


if __name__ == "__main__":
    while True:
        main()
```

【Python】selenium性能優化

selenium性能優化

```
chrome_options = Options()
chrome_options.add_argument("--window-size=1920,1080")
#禁用插件
chrome_options.add_argument("--disable-extensions")

#無圖形化
chrome_options.add_argument("--headless")
chrome_options.add_argument("--disable-gpu")
chrome_options.add_argument("--disable-software-rasterizer")

#禁用圖片
chrome_options.add_argument("blink-settings=imagesEnabled=false")

chrome_options.add_argument('--no-sandbox')
chrome_options.add_argument('--ignore-certificate-errors')
#允許不安全內容(ssl問題)
chrome_options.add_argument('--allow-running-insecure-content')
driver = webdriver.Chrome(options=chrome_options)
```

【Python】【PrettyTable】 資料格式化排版工具

安裝PrettyTable

```
pip install PrettyTable
```

範例程式

```
from prettytable import PrettyTable
  x = PrettyTable()

  x.field_names = ["First name", "Last name", "Salary", "City", "DOB"]
  x.add_row(["Shubham", "Chauhan", 60000, "Lucknow", "22 Feb 1999"])
  x.add_row(["Saksham", "Chauhan", 50000, "Hardoi", "21 Aug 2000"])
  x.add_row(["Preeti", "Singh", 40000, "Unnao", "10 Jan 1995"])
  x.add_row(["Ayushi", "Chauhan", 65000, "Haridwar", "30 Jan 2002"])
  x.add_row(["Abhishek", "Rai", 70000, "Greater Noida", "16 Jan 1999"])
  x.add_row(["Dinesh", "Pratap", 80000, "Delhi", "3 Aug 1998"])
  x.add_row(["Chandra", "Kant", 85000, "Ghaziabad", "18 Sept 1997"])
  table = x.get_string()
  print(table)
```

```
+------------+-----------+--------+---------------+--------------+
| First name | Last name | Salary |      City     |     DOB      |
+------------+-----------+--------+---------------+--------------+
|  Shubham   |  Chauhan  | 60000  |    Lucknow    | 22 Feb 1999  |
|  Saksham   |  Chauhan  | 50000  |     Hardoi    | 21 Aug 2000  |
|   Preeti   |   Singh   | 40000  |     Unnao     | 10 Jan 1995  |
|   Ayushi   |  Chauhan  | 65000  |    Haridwar   | 30 Jan 2002  |
|  Abhishek  |    Rai    | 70000  | Greater Noida | 16 Jan 1999  |
|   Dinesh   |   Pratap  | 80000  |     Delhi     |  3 Aug 1998  |
|  Chandra   |    Kant   | 85000  |   Ghaziabad   | 18 Sept 1997 |
+------------+-----------+--------+---------------+--------------+
```

【Python】【tabulate】資料格式化排版工具

安裝

```
pip3 install tabulate
```

範例

```
from tabulate import tabulate

data = [
    ["Alice", 25, "Engineer"],
    ["Bob", 30, "Developer"],
    ["Charlie", 35, "Manager"],
]

headers = ["Name", "Age", "Job"]

table = tabulate(data, headers=headers, tablefmt="psql")
print(table)
```

```
+---------+-----+------------+
| Name    | Age | Job        |
|---------+-----+------------|
| Alice   | 25  | Engineer   |
| Bob     | 30  | Developer  |
| Charlie | 35  | Manager    |
+---------+-----+------------+
```

你可以根據需要選擇不同的表格格式，例如 `"plain"`、`"simple"`、`"grid"` 等。`tabulate` 模組還支援對齊選項、數值格式化等功能，

`tabulate` 模組的 `tablefmt` 參數用於指定要使用的表格格式。下面列出了一些常用的 `tablefmt` 參數值：

1. `"plain"`: 這是默認的表格格式，使用簡單的 ASCII 字符。
2. `"simple"`: 使用更精簡的 ASCII 字符繪製表格。
3. `"grid"`: 使用 ASCII 字符繪製網格狀的表格。
4. `"fancy_grid"`: 使用更複雜的 ASCII 字符繪製網格狀的表格。
5. `"pipe"`: 使用□□字符 `|` 繪製表格。
6. `"orgtbl"`: 使用 `+` 和 `-` 字符繪製表格。
7. `"jira"`: 生成符合 Jira Wiki 格式的表格。
8. `"presto"`: 生成符合 Presto 命令行客戶端輸出的表格格式。
9. `"pretty"`: 使用 Unicode 字符繪製表格，具有較好的可讀性。

這只是 `tablefmt` 參數的一些常見值，`tabulate` 還支援其他一些特殊格式，如 HTML、LaTeX 等。你可以參考 `tabulate` 模組的官方文件以獲取完整的 `tablefmt` 參數列表和相應的表格格式示例。

【Python】【環境建置】venv 虛擬環境建置

```
python3 -m venv .venv
source .venv/bin/activate
python3 -m pip install -r requirements.txt
python3 todo.py
```

python3 -m venv .venv

虛擬環境是一個獨立的 Python 環境，可以讓你在同一台機器上使用不同版本的 Python 和套件，而不會相互干擾。這對於開發和測試項目時非常有用，因為每個項目都可以有自己的獨立環境。

在命令 `python3 -m venv .venv` 中，`python3` 是用於執行 Python 3 解釋器的命令，`-m venv` 是告訴 Python 解釋器使用 venv 模組來建立虛擬環境，`.venv` 是虛擬環境的目錄名稱。

執行這個命令後，會在當前目錄下建立一個名為 `.venv` 的目錄，這個目錄就是虛擬環境的根目錄。在這個目錄中，會包含一個名為 `bin`（或 `Scripts` 在 Windows 上）的子目錄，其中包含了虛擬環境使用的 Python 解釋器和相關的工具。

要啟用虛擬環境，可以使用以下命令：

- 在 macOS/Linux 上：`source .venv/bin/activate`
- 在 Windows 上：`.venv\Scripts\activate.bat`

啟用虛擬環境後，你可以在該環境中安裝和使用 Python 套件，而這些套件不會影響到全域 Python 環境。當你在虛擬環境中完成工作後，可以使用 `deactivate` 命令來停用虛擬環境。

啟用 Python 虛擬環境 ource .venv/bin/activate

`source .venv/bin/activate` 是用於啟用 Python 虛擬環境的命令，該虛擬環境是使用 `python3 -m venv` 建立的。

當你執行這個命令時，它會讀取虛擬環境目錄（`.venv`）中的 `activate` 腳本並執行它。這個腳本會設置一些環境變數和修改你的 shell 提示符，以將你的 Python 環境切換到虛擬環境。

具體而言，`source .venv/bin/activate` 的作用如下：

1. 啟動虛擬環境：這個命令會將你的 shell 環境切換到虛擬環境中，使你在該環境中運行的 Python 解釋器和安裝的套件與全域環境隔離開來。
2. 設置環境變數：這個腳本會設置幾個環境變數，如 `PATH`、`PYTHONPATH` 等，以便在虛擬環境中優先使用該環境中的 Python 解釋器和套件。
3. 修改提示符：這個腳本還可以修改你的 shell 提示符，通常會在提示符前加上虛擬環境的名稱，以提醒你正在使用虛擬環境。

當虛擬環境被啟用後，你可以在該環境中使用 `pip` 命令安裝套件，執行 Python 腳本等。所有的操作都會在虛擬環境中進行，不會影響到全域環境。

如果你想停用虛擬環境，可以執行 `deactivate` 命令，它會恢復你的 shell 環境到原始狀態，不再使用虛擬環境。

【 Python】常用自訂函數

特數字元轉譯

```
def escape_string(input_string):
    special_chars = ['<', '>', '&', '*', '_', '~', '`', '|', '#', '!']
    escaped_string = ""

    for char in input_string:
        if char in special_chars:
            escaped_string += '\\' + char
        else:
            escaped_string += char

    return escaped_string
```

【Python】import 用法

在 Python 中，`import` 是用來引入其他模組、套件或特定功能的關鍵字，讓你可以重複利用現有的程式碼，避免重複撰寫功能。以下是 `import` 的詳細說明及常見用法：

---

1. `import` 的基本概念

`import` 用於引入一個 Python 模組或套件，讓你可以使用其中的函數、類別或變數。

模組與套件

- **模組**：是一個 Python 檔案（以 `.py` 結尾），其中包含定義的函數、類別或變數。
  - 範例：`math` 模組、你自己撰寫的 `mymodule.py`。
- **套件**：是一個包含多個模組的目錄，且該目錄下有 `__init__.py` 文件。
  - 範例：`os` 套件、`requests` 套件。

---

2. `import` 的使用方式

2.1 匯入整個模組

- 語法：

```
import module_name
```

- 範例：

```
import math

print(math.sqrt(16))  # 使用 math 模組中的 sqrt 函數
```

- 特點：需要使用 `模組名稱.功能` 的方式來存取模組內容。

2.2 匯入模組的特定部分

- 語法：

```
from module_name import specific_function_or_variable
```

- 範例：

```
from math import sqrt

print(sqrt(16))  # 直接使用 sqrt 函數，無需加上模組名稱
```

- 特點：只匯入需要的部分，節省記憶體，但可能引發命名衝突。

2.3 匯入模組並重新命名

- 語法：

```
import module_name as alias
```

- 範例：

```
import numpy as np

arr = np.array([1, 2, 3])
print(arr)
```

- 特點：透過別名縮短模組名稱，讓程式碼更簡潔。

2.4 匯入模組所有內容

- 語法：

```
from module_name import *
```

- 範例：

```
from math import *

print(sqrt(16)) # 可以直接使用 math 中的所有功能
```

- 特點：
  - 匯入所有內容，但不推薦，因為可能導致命名衝突。
  - 建議明確列出需要匯入的內容（用 `__all__` 控制）。

2.5 匯入套件中的子模組

- 語法：

```
from package_name import submodule_name
```

- 範例：

```
from os import path

print(path.exists("example.txt")) # 使用 os.path 模組中的 exists 函數
```

---

3. Python 搜尋模組的順序

當執行 `import` 時，Python 按照以下順序尋找模組：

1. **內建模組**：Python 標準庫中的模組，例如 `math`、`os`。
2. **當前目錄**：程式執行時所在的目錄。
3. **PYTHONPATH**：環境變數中指定的路徑。
4. **全域安裝的目錄**：例如 `site-packages`。

如果找不到模組，會拋出 `ModuleNotFoundError`。

---

4. 常見的 import 模式比較

匯入方式	使用方式	優點	缺點
`import module_name`	`module_name.function()`	清楚來源，避免命名衝突	使用時需要加上模組名稱
`from module_name import func`	`func()`	使用簡單，僅匯入需要的內容	可能導致命名衝突
`import module_name as alias`	`alias.function()`	模組名稱簡潔，程式碼更易閱讀	增加了別名學習的成本
`from module_name import *`	`function()`	簡單直接，適用於了解所有內容的情況	容易命名衝突，降低可讀性

---

5. 自訂模組的匯入

5.1 自訂模組

假設你有一個名為 `mymodule.py` 的檔案，內容如下：

```
# mymodule.py
def greet(name):
    return f"Hello, {name}!"
```

你可以在同目錄下使用：

```
import mymodule

print(mymodule.greet("Alice"))  # 輸出: Hello, Alice!
```

5.2 結構化套件

假設你有以下檔案結構：

```
myproject/
├── main.py
└── mypackage/
    ├── __init__.py
    ├── module1.py
    └── module2.py
```

`mypackage/__init__.py` **的內容：**

```
from .module1 import func1
from .module2 import func2

__all__ = ["func1", "func2"]
```

使用：

```
from mypackage import func1

func1()
```

---

6. 結論

- `import` **是 Python 中的核心功能，用於模組的重複利用。**
- **掌握不同的匯入方式，可以讓你的程式碼更高效、更清晰。**
- **盡量避免使用** `from module import *`**，選擇明確的匯入方式，提升程式碼可讀性與維護性。**

【Python】【__init__.py】,【__all__】說明

Python 中的 `__init__.py` 與 `__all__` 用途

在 Python 中，`__init__.py` 是專門用來初始化套件的重要檔案，而 `__all__` 是用來控制模組或套件的對外公開接口。以下將針對這兩者的用途及使用方式進行完整說明。

`__init__.py` 的用途

1. 標記目錄為 Python 套件

在 Python 3.3 之前，`__init__.py` 是標示一個目錄為套件的必要檔案。沒有這個檔案，目錄無法被視為套件。雖然 Python 3.3 之後允許隱式套件（目錄中沒有 `__init__.py` 也能被視為套件），但仍建議在目錄中保留 `__init__.py`，以明確表示該目錄是套件。

---

2. 套件初始化

`__init__.py` 的內容會在匯入套件時執行，因此你可以在這裡添加初始化代碼，例如：

- 設定全域變數或環境配置。
- 匯入套件內的模組或子模組，簡化使用者匯入的流程。
- 處理套件的必要初始化邏輯。

範例：

```
# 檔案結構
# mypackage/
# ├── __init__.py
# ├── module1.py
# ├── module2.py

# __init__.py
print("Initializing mypackage...")
from .module1 import func1
from .module2 import func2

# 匯入時會執行初始化代碼
import mypackage
# 執行結果: "Initializing mypackage..."
```

---

3. 控制對外公開接口

`__init__.py` 可以用來控制使用者可以從套件匯入的功能，例如：

- 只公開指定的模組或函式。
- 隱藏內部的模組，避免使用者直接操作。

範例：

```
# __init__.py
from .module1 import func1
from .module2 import func2

__all__ = ['func1', 'func2']
```

使用者只需匯入套件即可：

```
from mypackage import *

func1()  # 正常執行
func2()  # 正常執行
```

`__all__` 的用途

`__all__` 是 Python 中的特殊變數，用於定義模組或套件的對外公開接口，特別是在使用 `from module import *` 時，`__all__` 可以控制哪些成員可以被匯入。

1. 限制公開的內容

`__all__` 是一個列表，用來列出所有允許匯入的成員。如果未定義 `__all__`，則預設會匯入所有不以下劃線 `_` 開頭的公開成員。

範例：

```
# example.py
def func1():
    return "This is func1"

def func2():
    return "This is func2"

def _private_func():
    return "This is a private function"

__all__ = ['func1']  # 只允許 func1 被匯入
```

在其他程式中使用：

```
from example import *

print(func1())  # 正常執行
print(func2())  # NameError: name 'func2' is not defined
```

---

2. 提升可讀性

使用 `__all__` 可以明確標示模組或套件的公共接口，讓開發者清楚哪些功能是穩定且推薦使用的。

---

3. 結合套件使用

當一個套件有多個模組時，可以在 `__init__.py` 中使用 `__all__` 控制整個套件的公共接口。

範例：

```
# 檔案結構
# mypackage/
# ├── __init__.py
# ├── module1.py
# ├── module2.py

# module1.py
def func1():
    return "Function 1"

# module2.py
def func2():
    return "Function 2"

# __init__.py
```

```
from .module1 import func1
from .module2 import func2

__all__ = ['func1']  # 只公開 func1
```

在使用者端：

```
from mypackage import *

print(func1())  # 正常執行
print(func2())  # NameError: name 'func2' is not defined
```

總結

1. **`__init__.py` 的用途：**
   - 標記目錄為 Python 套件。
   - 執行套件初始化邏輯，例如匯入模組或設定全域變數。
   - 控制套件的對外接口，組織套件結構，簡化使用者的匯入流程。
2. **`__all__` 的用途：**
   - 控制 `from module import *` 時匯入的內容，避免暴露內部實現細節。
   - 提升程式碼的可讀性，讓開發者清楚哪些功能是公開的。
   - 結合 `__init__.py` 使用，控制整個套件的公共接口。

最佳實踐建議

- 總是為套件添加 `__init__.py`，即使在 Python 3.3 之後不強制要求。
- 明確定義 `__all__`，只公開穩定且推薦使用的功能，隱藏內部實現細節。
- 避免使用 `from module import *`，除非必要，以提升程式碼的可讀性和穩定性。

這樣可以讓你的模組和套件更加結構化、易於維護，並避免不必要的命名衝突。