Python圖形化介面

圖形化介面

圖形化介面

- Python對於圖形化介面:
    - 優勢: 在簡單的介面實作上, Python可以快速簡單的繪製出介面
    - 劣勢: 在複雜的介面實作上, Python因為缺乏好用的工具(拖拉即可畫出介面), 所以並不適合
- 所以, 我們在使用上, 會拿來繪製簡單的圖形化介面

tkinter

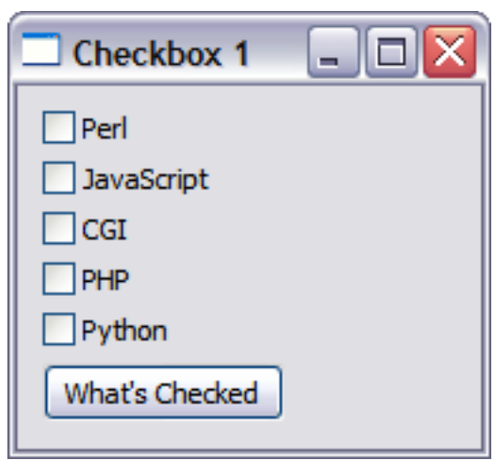
- 我們使用Python內建的tkinter函式庫來繪製圖形化介面
- 元件使用原生作業系統的長相
- 跨平台
- 使用方法較為簡單
- 可參考: https://www.tutorialspoint.com/python/python_gui_programming.htm

常用的元件

- Label: 不可變的文字標籤
- Button: 按鈕
- Radio Button: 單選
- Check Box: 複選
- Entry: 單行輸入
- Text: 多行輸入

radio button

check box



GUI結構

- 通常我們會在最底層加入一層畫布(Frame), 在接下來再在畫布上放上各種元件

- (重要) 這是有上下層關係的一個結構, 這些元件都屬於底層的Frame

Layout

- 我們有了元件以後, 還要以排版來約束他們的位置

- 我們大部分使用grid layout來約束位置

Grid Layout

- 透過行與列來排列元素
- 就如同Excel表格的感覺, 同樣可以使用(columnspan=x)來合併x個儲存格

Demo

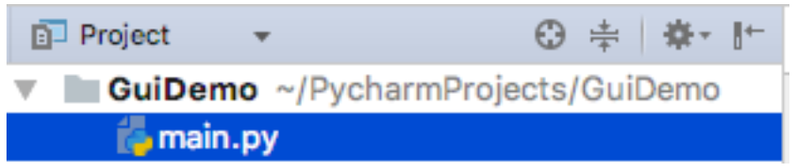
猜英文單字遊戲

Source: https://drive.google.com/open?id=0B1uHb3qi7BSOdWRmVGZhODBCZW8

完整版

- 上面會出現要你猜的單字
- 你必須在Guess Word裡輸入你的猜測
- “猜”會看看你的猜測是否正確
- 太難可以選擇”換一個”來換一個單字
- https://drive.google.com/open?id=0B1uHb3qi7BSOdWRmVGZhODBCZW8

Step1. 建構GUI



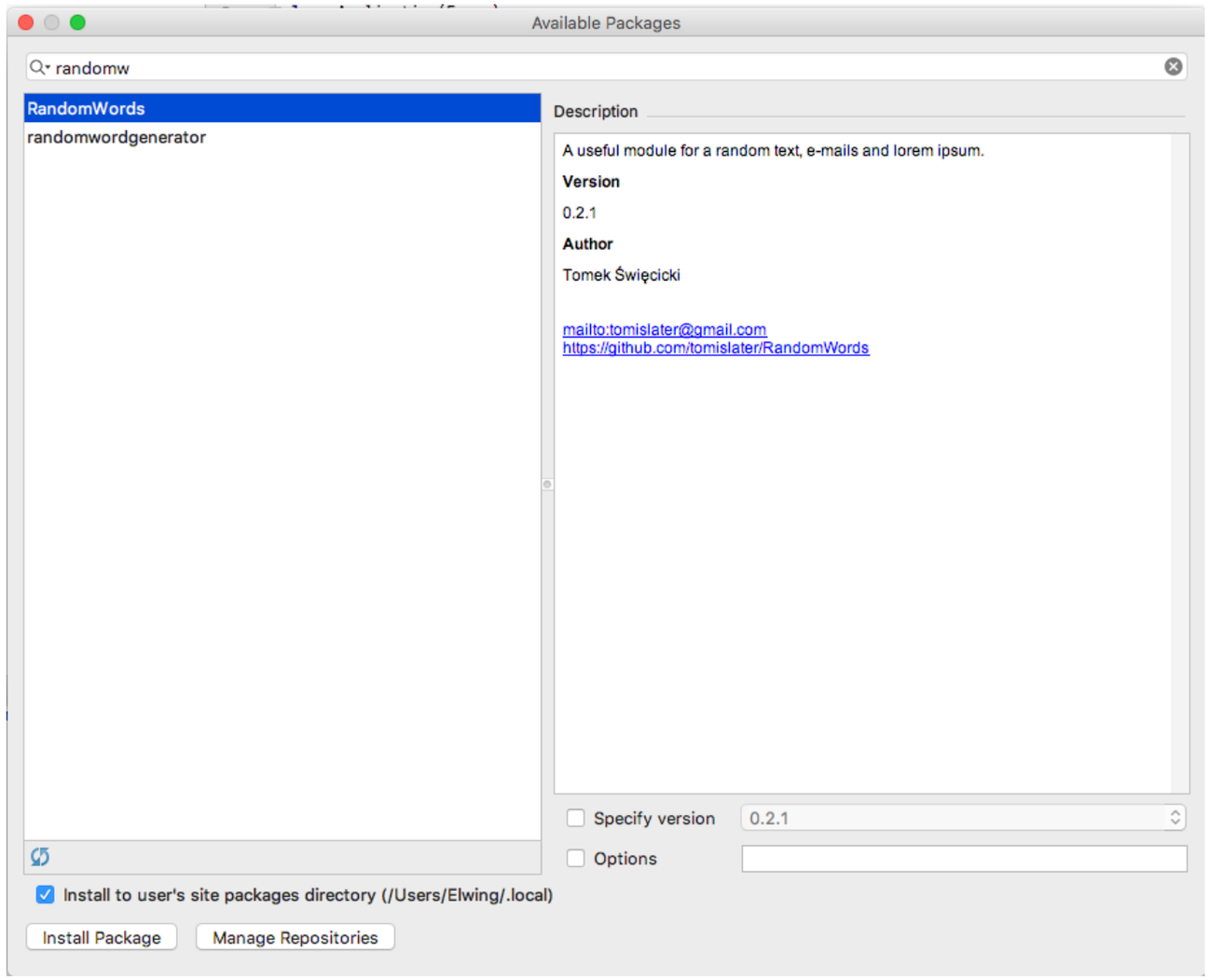
- 先將我們需要的整個介面畫好

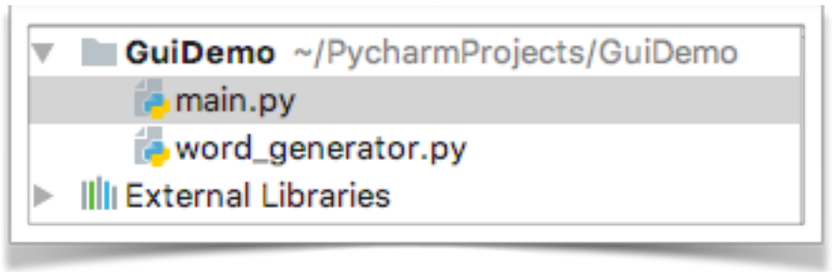
- 最後一定要呼叫mainloop()才會跳出視窗

```python
from tkinter import *

# 我們自製的Frame(繼承自Frame)
class GuessFrame(Frame):

    def createWidgets(self):
        # 在這自製的Frame上面放上Label(上方的單字) 合併兩格儲存格(columnspan=2)
        self.word_label = Label(self, font=("Times New Roman", 30, "bold"), text = "T_S_")
        self.word_label.grid(row = 0, column = 0, columnspan = 2)
        # 左方的GuessWord Label
        self.guess_label = Label(self, text="Guess Word:")
        self.guess_label.grid(row=1, column = 0)
        # 單行輸入
        self.guess_text = Entry(self)
        self.guess_text.grid(row=1, column = 1)
        # 猜的Button, sticky='WE'的時候, 他會將整個button擴充成跟你的儲存格一樣大
        self.guess_button = Button(self, text="猜")
        self.guess_button.grid(row=2, column = 1, sticky='WE')
        # 換一個的Button
        self.change_button = Button(self, text = "換一個")
        self.change_button.grid(row=3, column = 1, sticky = 'WE')
        # 猜對以後會累積猜對次數Label
        self.result = Label(self, fg='red')
        self.result.grid(row = 4, column=0, columnspan=2, sticky='WE')


    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.pack(expand="YES")
        self.createWidgets()

root = Tk()
app = GuessFrame(master=root)
app.mainloop()
```

Step2. 加入遊戲機

- 首先必須先安裝一個外掛函式庫(RandomWords), 來幫我們產生出random的英文單字





- 實作我們的遊戲機, 增加 word_generator.py

```python
from random_words import RandomWords
import random


class WordGenarator():
    def __init__(self):
        self.rw = RandomWords()
        self.generate_random_word()
        self.right_count = 0

    # 這函式會負責幫我們產生出隨機的英文單字
def generate_random_word(self):
    self.word = self.rw.random_word()
    # 如果產生出的字少於三個字, 重新產生一下
    while len(self.word) <= 3:
        self.word = self.rw.random_word()

    print(self.word)

    self.show_word = self.word
    underline = 0
    # 將產生出的字的隨機兩個位置使用_下底線來取代
    while underline < 2:
        random_index = random.randint(0, len(self.word) - 1)
        if self.show_word[random_index] != '_':
            s = list(self.show_word)
            s[random_index] = '_'
            self.show_word = "".join(s)
            underline = underline + 1
```

Step3. 修改GUI

- 將我們的遊戲機加入回main.py

```
from tkinter import *
from word_generator import WordGenarator

class GuessFrame(Frame):

    def createWidgets(self):
        # 上方的label, 裡面的text設定為generator產生出的要猜的英文
        self.word_label = Label(self, font=("Times New Roman", 30, "bold"), text = self.generator.show_word)
        self.word_label.grid(row = 0, column = 0, columnspan = 2)
        # Guess Word Label
        self.guess_label = Label(self, text="Guess Word:")
        self.guess_label.grid(row=1, column = 0)
        # 單行輸入, 輸入你的猜測
        self.guess_text = Entry(self)
        self.guess_text.grid(row=1, column = 1)
        # 點選以後會走到guess_word函式驗證猜測
        self.guess_button = Button(self, text="猜", command=self.guess_word)
        self.guess_button.grid(row=2, column = 1, sticky='WE')
        # 點選以後會走到change_word來換一個英文
        self.change_button = Button(self, text = "換一個", command=self.change_word)
        self.change_button.grid(row=3, column = 1, sticky = 'WE')
        # 勝利的次數記錄
        self.result = Label(self, fg='red')
        self.result.grid(row = 4, column=0, columnspan=2, sticky='WE')
```

```
25
26      # 重新產生一個英文
27      def change_word(self):
28          self.generator.generate_random_word()
29          self.word_label['text'] = self.generator.show_word
30
31      # 驗證猜測
32      def guess_word(self):
33          if self.guess_text.get() == self.generator.word:
34              print(self.guess_text['text'])
35              self.generator.right_count = self.generator.right_count + 1
36              self.change_word()
37              self.result['text'] = '猜對' + str(self.generator.right_count) + '次囉'
38
39      def __init__(self, master=None):
40          Frame.__init__(self, master)
41          # 把遊戲機初始化
42          self.generator = WordGenarator()
43          self.pack(expand="YES")
44          self.createWidgets()
45
46  root = Tk()
47  app = GuessFrame(master=root)
48  # 記得呼叫mainloop跳出視窗
49  app.mainloop()
```

資料庫概論

資料庫概論

資料庫簡介

- 資料庫其實就跟你平常使用的EXCEL是相同的東西, 就是一種表格

- 為了資料庫的維護性, 我們必須對其做一些特別的處理(正規化)

資料庫正規化

- 前人在實踐資料庫的維護性的時候, 歸納出一些通則, 當你的資料庫設計遵守這些通則的時候, 維護性至少就是沒問題的

- 一個資料庫通常會遵守前三個正規化守則

- 必須先符合1NF(第一正規化), 再符合2NF(第二正規化), 最後再符合3NF(第三正規化)

Demo

經由例子來體會正規化吧

第一正規化(1NF)

- 第一正規化: 唯一性, 也就是消除重複性
- 顯示在資料表的設計:
    - 每個欄位只能有一個值, 而每一個列必須有其獨特性

一般人的設計(1)

- 一般人會將多次的紀錄記錄在同一筆欄位, 或者是開創多個欄位紀錄

- 缺點: 由於欄位裡的值不定, 每次的查詢要先確認裡面到底有幾個值

交易

顧客	日期	數量
Pete	Monday	19.00 -28.20
Pete	Wednesday	-84.00
Sarah	Friday	100.00 150.00 -40.00

個人資料

人	喜歡的顏色	不喜歡的食物 (1)	不喜歡的食物 (2)	不喜歡的食物 (3)
Jim	Green	Liver	Goat's cheese	
Alice	Fuchsia	Broccoli		
Norman	Blue	Pheasant	Liver	Peas
Emily	Yellow			

1NF以後(1)

- 讓每一個欄位裡面的值都是唯一值, 但你可能會發現一點問題了

交易

顧客	日期	數量
Pete	Monday	19.00
Pete	Monday	−28.20
Pete	Wednesday	−84.00
Sarah	Friday	100.00
Sarah	Friday	150.00
Sarah	Friday	−40.00

發現問題

- 如果同一個顧客在同一天買了兩次同樣數量, 你會發現每一筆資料開始不遵守唯一性了

交易

顧客	日期	數量
Pete	Monday	19.00
Pete	Monday	19.00

1NF(2)

- 那我們怎麼讓兩筆資料是唯一性呢？
    - 最直覺的想法, 為每一筆資料加上一個不一樣的值, 我們稱這值為主鍵(Primary Key), 拿來分辨每一筆資料
    - 一個實務上的做法, 是為資料加上一個每次增加1的整數值

交易

交易 ID	顧客	日期	數量
1	Pete	Monday	19.00
2	Pete	Monday	19.00

Primary Key

第二正規化

- 第二正規化: 追求欄位的相關性, 事實上也有部分是為了唯一性

- 顯示在資料表的設計:

    - 每個欄位必須對於主鍵有完全的相關性

一般人的設計

- 主鍵由元件ID和供應商ID共同構成(唯一性)
- 設計的問題:
  - 供應商名稱和供應商住址和主鍵不是”完全相關”, 而只和供應商ID相關而已, 會造成改天要改住址或名稱的時候必須改動無數個欄位或造成欄位值的不一致

元件來源

元件 ID (主鍵)	價格	供應商ID (主鍵)	供應商名稱	供應商住址
65	59.99	1	Stylized Parts	VA
73	20.00	1	Stylized Parts	VA
65	69.99	2	ACME Industries	CA

2NF以後

- 分成兩個表來定義

元件來源

元件 ID (主鍵主鍵)	價格	供應商 ID（主鍵、非關鍵詞）
65	59.99	1
73	20.00	1
65	69.99	2

供應商

供應商 ID (主鍵)	名稱	住址
1	Stylized Parts	VA
2	ACME Industries	CA

第三正規化

- 第三正規化: 消除重複性, 不必要出現的欄位就不該出現
- 顯示在資料庫的設計:
    - 每一個欄位不應該對除了主鍵的欄位有依賴性

一般人的設計

- 每一個欄位都跟訂單編號相關, 所以符合2NF
- 設計的問題:
  - 小計其實就是單價*數量, 今天如果你要修改單價或數量欄位的時候, 小計就必須跟著變動

訂單 (Order)

訂單編號(Order Number) (主鍵)	客戶名稱 (Customer Name)	單價 (Unit Price)	數量 (Quantity)	小計 (Total)
1000	David	$35.00	3	$105.00
1001	Jim	$25.00	2	$50.00
1002	Bob	$25.00	3	$75.00

3NF以後

- 將小計拿掉以後, 每個欄位都跟其他非主鍵欄位沒有相依性

訂單 (Order)

訂單編號(Order Number) (主鍵)	客戶名稱 (Customer Name)	單價 (Unit Price)	數量 (Quantity)
1000	David	$35.00	3
1001	Jim	$25.00	2
1002	Bob	$25.00	3

總結

- 三個正規化的形式其實都在強調資料的唯一性以及降低欄位的相依性, 來避免重複定義或者是之後修改一個欄位就造成資料的不一致現象

SQL概論

SQL概論

SQL v.s. NoSQL

- 經過第一章介紹, 我們可以開始介紹SQL和NoSQL的不同了
    - SQL: 結構化表格(每一筆資料都擁有一樣多的欄位)
        - 可以使用標準化的SQL語法來查詢
            - ex: SQLite, MySQL, PostgreSQL
    - NoSQL: 半結構化表格(每一筆資料擁有不一樣多的欄位)
        - 沒有標準化的語法
        - ex: MangoDB

常用SQL語法

- https://www.w3schools.com/sql/default.asp
- SELECT: 查詢
- UPDATE: 改動某欄位
- DELETE: 刪除
- JOIN: 連結兩個表格

SQLite v.s. MySQL

- 我們選用SQLite來作為這堂課的實作, 不過MySQL的原理是完全一樣的
- SQLite:
    - 資料庫就是直接一個你可以看到的檔案, 搬移方便
    - 適合輕量級APP
    - Python內建支持
- MySQL:
    - 較為大型的解決方案, 有比較成熟的發展史, 穩定且快速
    - 適合中或重量級的APP
    - Python無內建支持, 需額外安裝driver

ORM

- 這堂課我們不教你基礎的SQL語法, 而是透過ORM來操作
- Object Relational Mapping(物件關聯對映):
    - 將每一筆資料轉成物件導向語言的一個物件來操作
    - 優點: 我們不用顧慮每一個資料庫一些細微差異的語法以及優化, ORM函式庫會幫我們解決這方面的問題

資料表關聯性

資料表關聯性

主鍵 & 外鍵

- 主鍵: 如上所述, 是為了讓一筆資料(一列)保持其”唯一性”
- 外鍵: 三個正規化後, 我們擁有了多個表格, 但他們其實是有關聯的, 這個關聯我們由外鍵來定義

- 通常我們會由一個表格的”外鍵”, 去聯繫另外一個表格的”主鍵”, 並且約束外鍵一定是另外一個表格內的主鍵值

資料表關聯性

- 一對一, 一對多: 由正規化後衍伸出來的關係, 並不需要特別處理

- 多對多: 正規化後並不存在這關係, 這關係是橫跨多表格後才存在

Demo

資料表關聯性

資料

- 我們要把這個表格依據之前教的正規化正規化後建立資料表關係

學生-課程ID(主鍵)	學生名字	班級名稱	老師名稱	聯絡電話		修習課程	
1	李白	大數據班	李清照	0913431232	0912346123	Python基礎	Python爬蟲
2	李白	JAVA班	辛棄疾	0913431232	0912346123	JAVA基礎	JDBC
3	李商隱	JAVA班	辛棄疾	0983412345		JDBC	Servlet
4	王維	大數據班	李清照	0973412342		Python爬蟲	Python NLP

正規化

- 大方向檢視
- 主鍵是學生 - 課程關係, 所以表格應該簡單就是關係的對應, 學生的基本資料和課程的基本資料可以拿出來獨立成一個表格

- Step1. 先把相關聯的東西拉出來,於是我們有了這麼樣的學生資料表, 班級資料表, 以及對應資料表

學生ID	學生名字	聯絡電話	
1	李白	0913431232	0912346123
2	李商隱	0983412345	
3	王維	0973412342	

班級ID	班級名稱	老師名稱
1	大數據班	李清照
2	JAVA班	辛棄疾

學生-課程ID(主鍵)	學生id	班級id	修習課程	
1	1	1	Python基礎	Python爬蟲
2	1	2	JAVA基礎	JDBC
3	2	2	JDBC	Servlet
4	3	1	Python爬蟲	Python NLP

- Step2. 來處理對多關係

電話-學生ID	電話	學生ID
1	0913431232	1
2	0912346123	1
3	0983412345	2
4	0973412342	3

學生ID	學生名字
1	李白
2	李商隱
3	王維

班級ID	班級名稱	老師名稱
1	大數據班	李清照
2	JAVA班	辛棄疾

課程ID	課程名稱	開課班級
1	Python基礎	1
2	Python爬蟲	1
3	Python NLP	1
4	JAVA基礎	2
5	JDBC	2
6	Servlet	2

學生id	修習課程
1	1
1	2
1	4
1	5
2	5
2	6
3	2
3	3

- Step3. 來看看每個紅線(FK-PK)關係是哪一種關係

電話-學生ID	電話	學生ID
1	0913431232	1
2	0912346123	1
3	0983412345	2
4	0973412342	3

多

學生ID	學生名字
1	李白
2	李商隱
3	王維

班級ID	班級名稱	老師名稱
1	大數據班	李清照
2	JAVA班	辛棄疾

多

課程ID	課程名稱	開課班級
1	Python基礎	1
2	Python爬蟲	1
3	Python NLP	1
4	JAVA基礎	2
5	JDBC	2
6	Servlet	2

多 多

學生id	修習課程
1	1
1	2
1	4
1	5
2	5
2	6
3	2
3	3

結論

- 經由上面三個個步驟結束後, 其實我們已經建立了一個符合3NF, 並且將資料表的關係確實地寫出來了, 你會發現我們只擁有一對多和一對一關係

- 那什麼是所謂的多對多關係呢?

    - 學生和修習課程的關係就是多對多(橫跨學生和課程表格), 但我們不會允許這麼一條線存在(無法處理), 所以存在的其實是兩個一對多關係

Python資料庫

Python資料庫

準備

- Python內建SQLite資料庫以及SQLite驅動

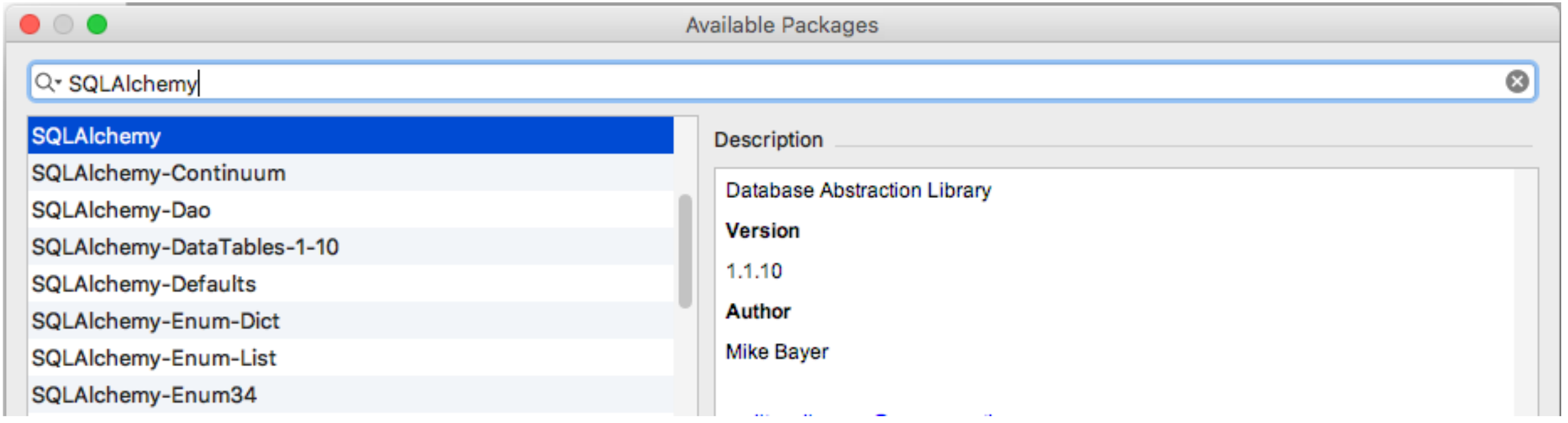
- 我們使用第三方的ORM函式庫SQLAlchemy來操作

Demo

使用Python來實作吧

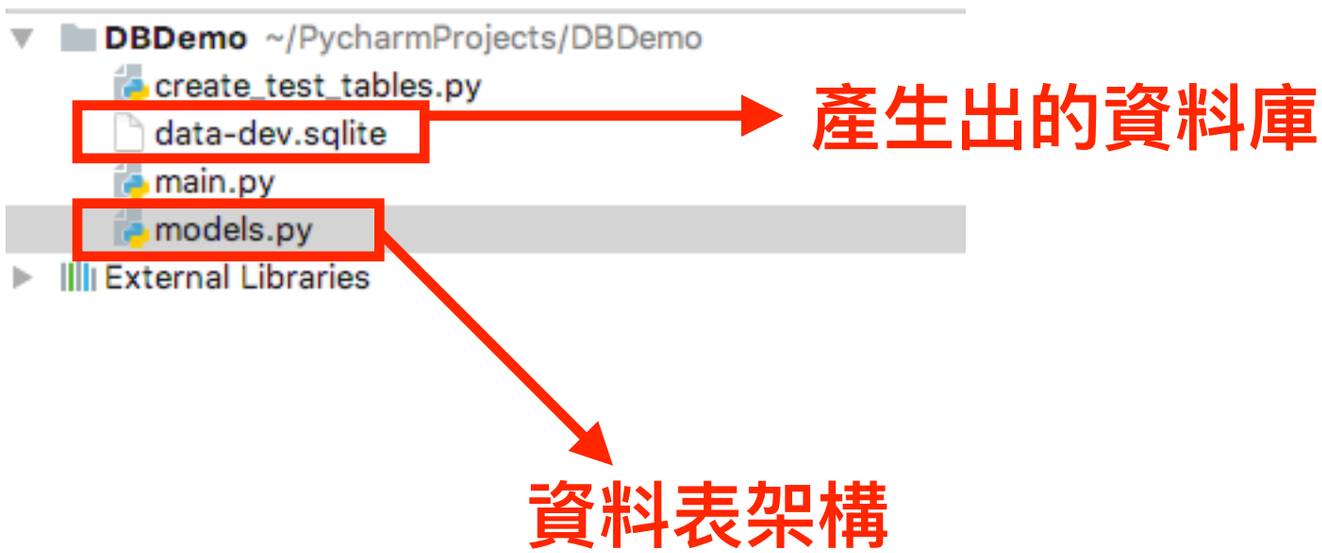
安裝SQLAlchemy

- (Pycharm) 設定 -> Project Interpreter -> 安裝
- (或) 透過PIP安裝



資料夾架構

- 源代碼: https://drive.google.com/open?id=0B1uHb3qi7BSOekIxSy1RRnc0OTQ



models.py

```python
from sqlalchemy import Column, Integer, String
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import ForeignKey
from sqlalchemy.orm import relationship, backref
from sqlalchemy import Table

# 準備個資料表的父類別
Base = declarative_base()

# 學生id-修習課程這個表，其實並不是我們care的主要項目，所以不需要專門寫一個class
# 寫成两个表之间的一個關聯Table就可
student_course_table = Table('student_course_association', Base.metadata,
    Column('student_id', Integer, ForeignKey('students.id')),
    Column('course_id', Integer, ForeignKey('courses.id'))
)

# 準備電話-學生ID 表
class Mobile(Base):
    # 資料表的名稱(必須)
    __tablename__ = 'mobiles'

    # 自動增長的Primary Key(Python會將看到的第一個非FK的PK設定為Auto Increment)
    id = Column(Integer, primary_key = True)
    number = Column(String(16))
    student_id = Column(Integer, ForeignKey('students.id'))

    # 覆寫__repr__來讓我們可以使用print印出比較有意義的資訊
    def __repr__(self):
        return '<Mobile %r>' %self.number

class Student(Base):
    __tablename__ = 'students'
    id = Column(Integer, primary_key = True)
    name = Column(String(32))

    #backref會自動幫你在關聯的class (CourseInfo) 創建students變數
    courses = relationship("CourseInfo", secondary=student_course_table, backref="students")
    mobiles = relationship("Mobile", backref="student")

    def __repr__(self):
        return '<Student %r>' %self.name
```

```python
class ClassInfo(Base):
    __tablename__ = 'classes'
    id = Column(Integer, primary_key = True)
    class_name = Column(String(32))
    teacher_name = Column(String(32))
    courses = relationship("CourseInfo", backref="class_info")

    def __repr__(self):
        return '<Class %r>' %self.teacher_name

class CourseInfo(Base):
    __tablename__ = 'courses'
    id = Column(Integer, primary_key = True)
    course_name = Column(String(32))
    class_id = Column(Integer, ForeignKey('classes.id'))

    def __repr__(self):
        return '<Course %r>' %self.course_name
```

create_tables.pt

```python
from models import *
from sqlalchemy.orm import sessionmaker
import os
from sqlalchemy import create_engine

# 創建engine(你可以想像成driver)
basedir = os.path.abspath(os.path.dirname(__file__))
engine = create_engine('sqlite:///' + os.path.join(basedir, 'data-dev.sqlite'), echo = True)

Base.metadata.drop_all(engine)
# 根據定義創建表, 這裡的Base是定義在models.py的Base
Base.metadata.create_all(engine)

Session = sessionmaker(bind=engine)
session = Session()

class_1 = ClassInfo(class_name = '大數據班', teacher_name = '李清照')
class_2 = ClassInfo(class_name = 'JAVA班', teacher_name = '辛棄疾')

session.add(class_1)
session.add(class_2)

# class_info由backref創建, 直接帶入你創建好的ClassInfo物件, sqlalchemy就會幫你把class id這外鍵填入正確的值
course_1 = CourseInfo(course_name = 'Python基礎', class_info = class_1)
course_2 = CourseInfo(course_name = 'Python爬蟲', class_info = class_1)
course_3 = CourseInfo(course_name = 'Python NLP', class_info = class_1)
course_4 = CourseInfo(course_name = 'JAVA基礎', class_info = class_2)
course_5 = CourseInfo(course_name = 'JDBC', class_info = class_2)
course_6 = CourseInfo(course_name = 'Servlet', class_info = class_2)

session.add(course_1)
session.add(course_2)
session.add(course_3)
session.add(course_4)
session.add(course_5)
session.add(course_6)
```

```
# courses由relationship創建, 直接帶入你創建好的多個Course物件, sqlalchemy就會幫你handle好多對多關係
student_1 = Student(name = '李白', courses = [course_1, course_2, course_4, course_5])
student_2 = Student(name = '李商隱', courses = [course_5, course_6])
student_3 = Student(name = '王維', courses = [course_2, course_3])

session.add(student_1)
session.add(student_2)
session.add(student_3)

# student是由backref定義在mobile的
mobile_1 = Mobile(number = '0913431232', student = student_1)
mobile_2 = Mobile(number = '0912346123', student = student_1)
mobile_3 = Mobile(number = '0983412345', student = student_2)
mobile_4 = Mobile(number = '0973412342', student = student_3)

session.add(mobile_1)
session.add(mobile_2)
session.add(mobile_3)
session.add(mobile_4)

#最後要commit才能完成這次操作
session.commit()
```

main.py

```python
from models import *
from sqlalchemy.orm import sessionmaker
import os
from sqlalchemy import create_engine

# 創建engine(你可以想像成driver)
basedir = os.path.abspath(os.path.dirname(__file__))
engine = create_engine('sqlite:///' + os.path.join(basedir, 'data-dev.sqlite'), echo = True)

Session = sessionmaker(bind=engine)
session = Session()

print('------查詢課程------')
student = session.query(Student).filter_by(name='李白').first()
for single_course in student.courses:
    print(single_course)

print('------查詢學生------')
course = session.query(CourseInfo).filter_by(course_name='JDBC').first()
for single_student in course.students:
    print(single_student, single_student.mobiles)
```

usages of variable 'basedir'

結果

```
------查詢課程------
2017-06-20 09:57:59,381 INFO sqlalchemy.engine.base.Engine SELECT CAST('test plain returns' AS VARCHAR(60)) AS anon_1
2017-06-20 09:57:59,381 INFO sqlalchemy.engine.base.Engine ()
2017-06-20 09:57:59,382 INFO sqlalchemy.engine.base.Engine SELECT CAST('test unicode returns' AS VARCHAR(60)) AS anon_1
2017-06-20 09:57:59,382 INFO sqlalchemy.engine.base.Engine ()
2017-06-20 09:57:59,382 INFO sqlalchemy.engine.base.Engine BEGIN (implicit)
2017-06-20 09:57:59,383 INFO sqlalchemy.engine.base.Engine SELECT students.id AS students_id, students.name AS students_name
FROM students
WHERE students.name = ?
 LIMIT ? OFFSET ?
2017-06-20 09:57:59,383 INFO sqlalchemy.engine.base.Engine ('李白', 1, 0)
2017-06-20 09:57:59,384 INFO sqlalchemy.engine.base.Engine SELECT courses.id AS courses_id, courses.course_name AS courses_course_name, courses.class_id AS courses_class_id
FROM courses, student_course_association
WHERE ? = student_course_association.student_id AND courses.id = student_course_association.course_id
2017-06-20 09:57:59,385 INFO sqlalchemy.engine.base.Engine (1,)
<Course 'Python基礎'>
<Course 'Python爬蟲'>
<Course 'JAVA基礎'>
<Course 'JDBC'>
------查詢學生------
2017-06-20 09:57:59,385 INFO sqlalchemy.engine.base.Engine SELECT courses.id AS courses_id, courses.course_name AS courses_course_name, courses.class_id AS courses_class_id
FROM courses
WHERE courses.course_name = ?
 LIMIT ? OFFSET ?
2017-06-20 09:57:59,385 INFO sqlalchemy.engine.base.Engine ('JDBC', 1, 0)
2017-06-20 09:57:59,386 INFO sqlalchemy.engine.base.Engine SELECT students.id AS students_id, students.name AS students_name
FROM students, student_course_association
WHERE ? = student_course_association.course_id AND students.id = student_course_association.student_id
2017-06-20 09:57:59,386 INFO sqlalchemy.engine.base.Engine (5,)
2017-06-20 09:57:59,387 INFO sqlalchemy.engine.base.Engine SELECT mobiles.id AS mobiles_id, mobiles.number AS mobiles_number, mobiles.student_id AS mobiles_student_id
FROM mobiles
WHERE ? = mobiles.student_id
2017-06-20 09:57:59,387 INFO sqlalchemy.engine.base.Engine (2,)
<Student '李商隱'> [<Mobile '0983412345'>]
2017-06-20 09:57:59,388 INFO sqlalchemy.engine.base.Engine SELECT mobiles.id AS mobiles_id, mobiles.number AS mobiles_number, mobiles.student_id AS mobiles_student_id
FROM mobiles
WHERE ? = mobiles.student_id
2017-06-20 09:57:59,388 INFO sqlalchemy.engine.base.Engine (1,)
<Student '李白'> [<Mobile '0913431232'>, <Mobile '0912346123'>]
```