Take-Home Task 2: Staff Directory (30%) – Individual Task

Requirement

Accompanying these instructions is a copy of a very large dataset containing details of employees in an organisation. The dataset has been provided as an SQLite database copy, `employees.db`. (A dump script, `employees_dump.sql`, has also been provided in case you have trouble opening the database.) When viewed in a database application such as the *DB Browser for SQLite* the database appears as follows:

	emp_no	birth_date	first_name	last_name	gender	hire_date
	Filter	Filter	Filter	Filter	Filter	Filter
1	10001	1953-09-02	Georgi	Facello	M	1986-06-26
2	10002	1964-06-02	Bezalel	Simmel	F	1985-11-21
3	10003	1959-12-03	Parto	Bamford	M	1986-08-28
4	10004	1954-05-01	Chirstian	Koblick	M	1986-12-01
5	10005	1955-01-21	Kyoichi	Maliniak	M	1989-09-12
6	10006	1953-04-20	Anneke	Preusig	F	1989-06-02
7	10007	1957-05-23	Tzvetan	Zielinski	F	1989-02-10
8	10008	1958-02-19	Saniya	Kalloufi	M	1994-09-15
9	10009	1952-04-19	Sumant	Peac	F	1985-02-18
10	10010	1963-06-01	Duangkaew	Piveteau	F	1989-08-24
11	10011	1953-11-07	Mary	Sluis	F	1990-01-22
12	10012	1960-10-04	Patricio	Bridgland	M	1992-12-18
13	10013	1963-06-07	Eberhardt	Terkki	M	1985-10-20
14	10014	1956-02-12	Berni	Genin	M	1987-03-11

|< < 1 - 14 of 300024 > >| Go to: 1

The challenge is to develop an application which allows the company's human resources department to quickly find employees based on their name. Rather than forcing the user to type in the full name, the application should allow the user to enter a short prefix. The application should then respond by listing details of all employees whose first name or last name begin with this prefix.

(To avoid any privacy issues, the dataset is synthetic and has been generated automatically. As a result most prefixes that match any employee records tend to return a large number of similar results. It would appear that this company hires lots of families!)

For instance, the image below shows one possible version of the intended application and the results returned by a particular search. In this case the user has entered the prefix '`Tei`' and the application has returned all employees whose first name begins with this prefix, such as '`Teiji`', and whose last name begins with this prefix, such as '`Teitelbaum`'. (There are actually many more results than are shown below, which can be revealed by scrolling the text widget in the top right.) Only the following details are displayed for each employee: employee number, first name, last name and date of birth.

Step 1 - Define the Front-End/Back-End Interface

Before you can begin work you must consider the interface between the front-end and the back-end code. Clearly this will be a function which accepts the user's 'search term', i.e., the prefix of the employee's name, as a parameter. Less clear is what format the results will be returned in. Here are some questions worth considering before you start the implementation:

- Should the results be returned as a list of matching employee details or as a single character string ready to display in the GUI?
- If a list, what should be the format of each item, a list of fields or a single character string?
  - If the former, should all fields for each matching employee be returned or just those that need to be displayed in the GUI? Notice above that our desired application does not use the employees' genders or hiring dates, so do these need to be returned at all?

- If the latter, should the string contain comma-separated fields or ready-to-display text?

- What should be returned if no matches are found?

Step 2a - Develop the Graphical User Interface

First you should develop a GUI similar to that shown above, with a "stub" representing the incomplete back-end function. The minimal requirements for the GUI are:

- a text Entry widget to allow the user to enter employee name prefixes;
- a Button widget to allow the user to start the look-up process; and
- a Text or Label widget to display the results.

In our example above we have also included a "Staff Directory" image (actually a Label widget), but you should leave such decorative features to the end, if time permits.

Step 2b - Develop the Back-End Search Function

Then you have the task of developing the back-end function which, given the prefix of an employee's name returns details of all employees whose first or last names begin with this prefix. This can be done using the SQLite database.

- Write a Python function which accesses the SQLite database copy of the employee data. In this case you will need to use the usual SQLite methods for opening and closing the database, executing a query on its contents, and fetching the result set. The query in this case can use the "*C* `like 'P%'`" Boolean expression to find rows in which a text-valued column *C* begins with text pattern *P*.

If time permits you should also create some unit tests for your function which help document its intended behaviour and aid future maintenance.

Step 3 - Integration Testing and Fine Tuning

Finally, when both the above steps have been completed, you should integrate the results to produce the final "app". You may need to modify the code you've written in step 2a and 2b to make them fit together properly. This is also the opportunity to make final improvements to the complete IT system before delivering it to the appreciative customer!

Deliverable

You should develop your solution by completing and submitting the provided Python template file **`staff_directory.py`** and **backend_function.py** as follows.

1. Complete the "statement" at the beginning of the Python file to confirm that this is your own individual work by inserting your name and student number in the places indicated. **We will assume that submissions without a completed statement are not your own work and they will not be marked.**
2. Complete your solution by developing Python code at the place indicated. You must complete your solution using **only the modules already imported by the provided template**. You may **not** use or import any other modules to complete this program. In particular, you may **not** import any image files into your solution.
3. Submit **both Python files** containing your solution for marking.

Apart from working correctly your program code must be well-presented and easy to understand, thanks to (sparse) commenting that explains the *purpose* of significant code segments and *helpful* choices of variable and function names. **Professional presentation** of your code will be taken into account when marking this task.

If you are unable to solve the whole problem, submit whatever parts you can get working. You will receive **partial marks for incomplete solutions**.

How to submit your solution

Your solution file should be submitted by the deadline to the Baidu drive under the "Assessment" folder.