IA160-3-FY Computer Programming

Project 1 Assignment

Personnel Database

1902160

Lecturer: Mothersole, Ian P

14 January 2020

Contents

Introduction

The task was to create a python script capable of creating and managing a database of employee records that a human resources department could use. This was to have the database as a python list, with each record being in the form of a python tuple. Personnel records had to consist of, a unique ID number, surname, employee first name(s), job title, full time in the form of "Yes" or "No", hourly rate (in £) and hours per week.
There had to be a repeating menu that showed the user their options, which were to include; addition of new records, to display the full names of all records, to search for an employee by name, to delete an employee record, to show just full-time or part-time employees, quit script, and if the advanced tasks were completed; to view a list of all records sorted alphabetically by either surname or job title, and to view all employees above a specified weekly cost.

Solution Explanation

Functions [3]

The top_table() function takes an integer argument referred to as size, size can be either '2' or '7', because these were the only two configurations needed for the script. When '2' is entered as an argument, it uses string formatting to print a two column table with one row comprised of "Surname" and "First Name" headers. When '7' is entered as an argument, it prints a seven column table with one row comprised of headers corresponding to all of the items in each record. full_name() prints just the Surname and First name records, and full_record() prints the whole record, and both use string formatting to print the record in a way that is compatible with top_table(2) and top_table(7) respectively. By printing the records in a table represented by formatted strings, it was possible to have the fields at the top and all records in line with the respective fields, meaning that the headers did not need to be repeated for every record printed.

Persistence

The database has persistence/retains records even when the script is not running, in order to be more practical as a database. This is achieved by having records stored in a separate python file that can be overwritten by the main python file via the write_database() function. The write_database() function uses the open()/write() methods[1] to overwrite the database with a copy of itself that has had a record appended or removed. The database is formatted as a string in the right syntax, so that python can interpret it as a script while imported. The script will also create a new database.py file if one is not present, this means the database can be reset easily by deleting the database.py file if necessary.

Menu

Following the importation/creation of database.py and the definition of the functions, most of the remaining script is contained within a while loop. This while True: loop is infinite, and will continue until break is used. The repeating menu is the first block of code within this loop, because it is the first thing the user needs to see at runtime, and after each task has been completed. The selection from the menu is saved to an input variable called sel, this input is handled by having a list of valid selections in the form of a single string, if there is only one

character and that string is amongst the valid selections, it can break from the while True loop, otherwise the user is prompted again.
Most options check that there are records to print and display a message otherwise. The delete record option requires a unique ID number to be input to ensure that the correct record is deleted.

Adding Records

By selecting ‘a’, the user is given a sequence of prompts for data about the employee. Following the successful entry of the record, it is appended to the record as a tuple along with a unique ID number which comes from the unique variable in database.py, the unique variable is then increased by one and the whole database is overwritten with the new values with write_database(). ID numbers being assigned to employees in ascending order of entry avoids duplication of ID numbers and constant changes to ID numbers potentially caused by the deletion of employee records.

All records in alphabetical order

To organise the records by surname and job title the sorted() method was used[2]. The sorted method used on a tuple will, by default, sort the tuples by index [0] of each record, as this would mean that the records are sorted by ID Number, which they are already, this is useless without setting the “key =” argument to a lambda function of the index to sort by, which is [1] and [3] for surname and job title respectively.

All records above a specified weekly cost

This option simply multiples the hours per week by the hourly rate for each record and prints the record in a table if it is above the user specified integer. Also some variables were added so that the script can display a message if the user specified integer is higher than the highest weekly cost, to explain why there are no results, and to print a message if the specified weekly cost is less than the lowest weekly cost to explain that the results include all employees.

Testing

The testing was carried out efficiently during development by initially having a list of tuples containing records, saved in the database_main.py file, so that records did not need to be entered every time the script was executed in order to test the other options, such as deletion and searching. Later, this was replaced with database.py .
Code was also tested by typing lines of code directly into the shell, to see how it worked in isolation, or independently of the script. This helped when unsure of the syntax, and to rapidly test many variations of code, without needing to go through the menu interface of the script for every iteration. This helped understand the open() method:

```
>>> test = (open("E:\Documents\Test.py", "r+"))
>>> test.write("Hello, World!")
13
>>> test.read()
''
```

```
>>> test = (open("E:\Documents\Test.py", "r"))
>>> test.read()
'Hello, World!'
>>>
```

Most of the tests, were by running the script and trying different values. When the script is executed, it prints a title that that shows that the script is a personnel database, a counter that shows how many records are in the database. And a menu, this appears as follows:

```
Human Resources department
Personnel Database
Employee Records: 10

a. Add a new employee record
b. Display all employees
c. Search for employee record (view details)
d. Delete an employee record
e. Show all full-time employees
f. Show all part-time employees
g. View all records (A-Z)
h. View employees above a specified weekly cost
x. Exit

Choose an option (a to h) or x to Exit: a
```

The output 'Employee Records: 10' was correct at runtime. The menu has displayed as intended, and the user has selected option 'a', which yields a succession of inputs used to add a new employee record to the database.

```
Please enter the surname of the employee (Max 20 char): jenkins
Please enter the first name of the employee (Max 20 char): mark
Please enter the job title of the employee: caretaker
Is the employee full time? Please enter 'Y' or 'N': y
What is the employee's Hourly rate (in £)?: 8
How many hours per week does the employee work?: 30

Human Resources department
Personnel Database
Employee Records: 11

a. Add a new employee record
b. Display all employees
c. Search for employee record (view details)
d. Delete an employee record
e. Show all full-time employees
f. Show all part-time employees
g. View all records (A-Z)
h. View employees above a specified weekly cost
x. Exit

Choose an option (a to h) or x to Exit: b
```

The 'Employee Records: ' counter has behaved predictably by increasing by one after entering a new record. The menu has repeated as it should, and the user has selected option b from the menu, which should display the first/last names of all employees:

```
|      Surname       |     First Name     |
|--------------------|--------------------|
|      Whitman       |        Jane        |
|       Smith        |       Danny        |
|      Johnson       |        John        |
|       Baker        |        John        |
|       Thorne       |        Max         |
|       Smith        |       Steven       |
|      McGreeve      |       Larry        |
```

```
|      O'Tally       |       Clair        |
|      Garrett       |       Denise       |
|       Edison       |        Ash         |
|      Jenkins       |        Mark        |

Human Resources department
Personnel Database
Employee Records: 11

[menu]

Choose an option (a to h) or x to Exit: c
```

The table has shown the user the first/last names of all employee records, including the record that was just added ‘’Mark Jenkins’. The counter has once again met expectations by remaining at 11 and not resetting or increasing, as no records have been added or removed since ‘Mark Jenkins’. The user has selected option c from the menu and will be prompted to enter a name to search for.

```
Please enter the first name or surname of the employee you wish to search
for: mark
```

ID Number	Surname	First Name	Job Title	Full time	Hourly Rate	Hours Per Week
0015	Jenkins	Mark	Caretaker	Yes	£8	30

```
[Title and counter]
[Menu]
```

The record of the searched name has been displayed in a table as intended, another test was conducted to see what the result would be if the name matched multiple records.

```
Choose an option (a to h) or x to Exit: c
Please enter the first name or surname of the employee you wish to search
for: smith
```

ID Number	Surname	First Name	Job Title	Full time	Hourly Rate	Hours Per Week
0003	Smith	Danny	Worker	Yes	£9	40
0009	Smith	Steven	Head of dept.	Yes	£20	30
0016	Smith	John	Worker	Yes	£9	30

```
Choose an option (a to h) or x to Exit: d
```

As ‘smith’ matched multiple records in the database, the table displayed all three names in the database that matched ‘smith’. The user selects option ‘d’ from the menu and will be prompted to enter the ID number of the employee they wish to delete.

```
Please enter the unique ID number of the employee you wish to delete from
database: 15

|      Surname       |     First Name     |
|--------------------|--------------------|
|      Jenkins       |        Mark        |
Are you sure you wish to delete? (Y/N): y

Human Resources department
Personnel Database
Employee Records: 10

[Menu]

Choose an option (a to h) or x to Exit: e
```

The full name of the employee is displayed to avoid error. A confirmation message is displayed, to which the user inputs ‘y’ to signify that they wish to delete ‘Mark Jenkins’ from the records.

This has been successful because the 'Employee Records' counter has decreased to '10'. The user selects option 'e' from the menu and will be shown a table of Full Time Employees:

```
Full-time employees:


|      Surname        |     First Name      |
|---------------------|---------------------|
|      Whitman        |         Jane        |
|       Smith         |        Danny        |
|      Johnson        |         John        |
|       Baker         |         John        |
|       Smith         |        Steven       |
|      O'Tally        |        Clair        |
|       Edison        |         Ash         |


[Title and Counter]
[Menu]

Choose an option (a to h) or x to Exit: f
```

All full-time employees have been displayed, notice how 'Mark Jenkins' a former full-time employee is not included in this list, as his record was deleted previously. The user selects 'f', and will be shown a list of part-time employees.

```
Part-time employees:


|      Surname        |     First Name      |
|---------------------|---------------------|
|       Thorne        |         Max         |
|      McGreeve       |        Larry        |
|      Garrett        |        Denise       |

[Title and Counter]
[menu]

Choose an option (a to h) or x to Exit: g
```

The table has displayed as intended and the user has selected 'g', this will show all records sorted alphabetically:

```
Records sorted Alphabetically by [S]urname or [J]ob title?: s
```

ID Number	Surname	First Name	Job Title	Full time	Hourly Rate	Hours Per Week
0007	Baker	John	Worker	Yes	£9	40
0014	Edison	Ash	Worker	Yes	£9	40
0012	Garrett	Denise	Marketing	No	£13	20
0005	Johnson	John	Worker	Yes	£9	40
0010	McGreeve	Larry	Accountant	No	£12	15
0011	O'Tally	Clair	Receptionist	Yes	£8	40
0003	Smith	Danny	Worker	Yes	£9	40
0009	Smith	Steven	Head of dept.	Yes	£20	30
0008	Thorne	Max	Security	No	£8.9	14
0002	Whitman	Jane	Worker	Yes	£9	40

```
[Title and Counter]
[Menu]

Choose an option (a to h) or x to Exit: g

Records sorted Alphabetically by [S]urname or [J]ob title?: j
```

ID Number	Surname	First Name	Job Title	Full time	Hourly Rate	Hours Per Week
0010	McGreeve	Larry	Accountant	No	£12	15
0009	Smith	Steven	Head of dept.	Yes	£20	30
0012	Garrett	Denise	Marketing	No	£13	20
0011	O'Tally	Clair	Receptionist	Yes	£8	40
0008	Thorne	Max	Security	No	£8.9	14
0002	Whitman	Jane	Worker	Yes	£9	40
0003	Smith	Danny	Worker	Yes	£9	40
0005	Johnson	John	Worker	Yes	£9	40

| 0007 | Baker | John | Worker | Yes | £9 | 40 |
| 0014 | Edison | Ash | Worker | Yes | £9 | 40 |

```
[Title and Counter]
[Menu]

Choose an option (a to h) or x to Exit: h
```

The user has been prompted to decide how they want the records to be sorted and has selected 's' to have them sorted alphabetically by surname, the table shows displays as expected. The user then selects 'g' to sort the records, and is prompted again, but this time selects 'j' to sort the records alphabetically by job title. This has also displayed as expected. The user then selects option 'h' and will be prompted for a weekly cost to check against employees.

```
Please enter the total weekly cost you want to check against employees: 400
```

ID Number	Surname	First Name	Job Title	Full time	Hourly Rate	Hours Per Week
0009	Smith	Steven	Head of dept.	Yes	£20	30

```
[title and counter]
[Menu]

Choose an option (a to h) or x to Exit: h
Please enter the total weekly cost you want to check against employees: 10
```

ID Number	Surname	First Name	Job Title	Full time	Hourly Rate	Hours Per Week
0002	Whitman	Jane	Worker	Yes	£9	40
0003	Smith	Danny	Worker	Yes	£9	40
0005	Johnson	John	Worker	Yes	£9	40
0007	Baker	John	Worker	Yes	£9	40
0008	Thorne	Max	Security	No	£8.9	14
0009	Smith	Steven	Head of dept.	Yes	£20	30
0010	McGreeve	Larry	Accountant	No	£12	15
0011	O'Tally	Clair	Receptionist	Yes	£8	40
0012	Garrett	Denise	Marketing	No	£13	20
0014	Edison	Ash	Worker	Yes	£9	40

```
All employees earn more than £10 per week

[Title and Counter]
[Menu]

Choose an option (a to h) or x to Exit: h
Please enter the total weekly cost you want to check against employees: 700
```

ID Number	Surname	First Name	Job Title	Full time	Hourly Rate	Hours Per Week

```
No employees Found, the highest cost per week is £600

[Title and Counter]
[Menu]

Choose an option (a to h) or x to Exit: x
Goodbye
>>>
```

The user has input 400 as the weekly cost and has been shown all records above this cost. The script also displays messages if the weekly cost entered is higher than the highest cost employee or lower than the lowest cost employee, which are tested by inputting 700 and 10 respectively. The user then selects 'x' from the menu, which acts as expected by printing a 'goodbye' message, and exiting the script.

The input handling for employee records was later tested:

```
What is the employee's Hourly rate (in £)?: Ketchup
Must be a number.
What is the employee's Hourly rate (in £)?: -6
No negative numbers, please.
What is the employee's Hourly rate (in £)?:
```

Conclusion

This solution to a personnel database is comprehensive and designed to be of practical use. By having the database saved in a separate python file to the main script, it can be updated and stored in between sessions. Much care and attention has also gone into; Layout: By having the records printed in a tabular format they are easy to read and compare, Input Handling: There are many input handling measures in place that should mean that if the user inputs something that is in the wrong format or incompatible with the script, the script should catch it and prompt the user to try again rather than crashing. The script is well tested and should not contain any bugs, and well commented so as to be easy to follow.
Next time it may be interesting to experiment more with functions and possibly have one function with multiple arguments that dictate the output rather than multiple functions that do similar things. A downside of the full_record() function is that the table it prints is quite wide, and because the text wraps, it can become messy if the shell window is not of a large enough size, so it would be worth considering ways of counteracting this.

References

[1] Python.org, '7.2.1. Methods of File Objects', [Online]. Available:https://docs.python.org/3.7/tutorial/inputoutput.html?highlight=write#methods-of-file-objects . [Accessed: 20-December 2019].

[2] Python.org, 'Sorting HOW TO', [Online]. Available:https://docs.python.org/3/howto/sorting.html#key-functions . [Accessed: 20-December 2019].

[3] Python.org, '4.6. Defining Functions**', [Online]. Available:https://docs.python.org/3/tutorial/controlflow.html#defining-functions** . [Accessed: 20-December 2019].

Appendix

database_main.py:

```
try:
    import database #Database list of employee records
except: # If database does not exist or has been deleted, a new empty file will be created
    print("database.py not found")
    db = open("database.py", "w")
    db.write("#This is a list of tuples containing the records\n" + "records = []\n" +
                    "\n#This is a number that increases with every entry but does not decrease with
deletions, it is used to assign a unique number to employees\n" +
                    "unique = 1")
    db.close()
    print("database.py created")
    import database


def write_database(): # Writes the databases current state to the database file
```

```
            db = open("database.py", "w")
            db.write("#This is a list of tuples containing the records\n" + "records =
{}\n".format(database.records) +
                     "\n#This is a number that increases with every entry but does not decrease with
deletions, it is used to assign a unique number to employees\n" +
                     "unique = {}".format(database.unique))
            db.close()

def full_name(record): # Prints the first and second names in a two column table
    print(("|{:" "^20}"*2 + "|").format(record[1], record[2]))
def full_record(record):# Prints whole record
    print(("|{:" "^20}"*7 + "|").format(str(record[0]).zfill(4), record[1], record[2], record[3],
record[4], "£"+str(record[5]), str(record[6])))

# This function displayes the table in a suitable format, either two columns for first and surname, or
seven for a whole record
def top_table(size):
    S = size
    if S == 2:
        print("\n" + ("|{:" "^20}"*S + "|").format("Surname", "First Name"))
    elif S == 7:
        print("\n" + ("|{:" "^20}"*S + "|").format("ID Number", "Surname", "First Name", "Job Title",
"Full time", "Hourly Rate", "Hours Per Week"))
    print(("|" + ("-"*20))*S + "|")

while True:
    #A menu of options
    print("\nHuman Resources department\nPersonnel Database")
    print("Employee Records: {}".format(len(database.records))) # Counts number of records in database
by checking how many tuples are in the "records" list.
    print("""
a. Add a new employee record
b. Display all employees
c. Search for employee record (view details)
d. Delete an employee record
e. Show all full-time employees
f. Show all part-time employees
g. View all records (A-Z)
h. View employees above a specified weekly cost
x. Exit
""")
    options = "abcdefghx" # Possible menu selections for input handling

    while True:
        sel = input("Choose an option (a to h) or x to Exit: ").lower() # Takes user menu selection,
stores as a lower case string for easy checking
        if options.count(sel) == 1 and len(sel) == 1: # the .count() checks that the selection is a
valid character, and the len() checks that there is only one character
            break

    if sel == "a": # a sequence of inputs for employee record data
        while True:
            sname = input("Please enter the surname of the employee (Max 20 char): ").capitalize()
            if len(sname) > 0:
                break
        while True:
            fname = input("Please enter the first name of the employee (Max 20 char): ").capitalize()
            if len(fname) > 0:
                break
        while True:
            jtitle = input("Please enter the job title of the employee: ").capitalize()
            if len(jtitle) > 0:
                break
        while True:
            ft = input("Is the employee full-time? Please enter 'Y' or 'N': ").lower()
            if ft == "y":
```

```
                ft = "Yes"
                break
            elif ft == "n":
                ft = "No"
                break
        while True:
            try:
                rate = int(input("What is the employee's Hourly rate (in £)?: "))
            except:
                print("Must be a number.")
            if rate >= 0:
                break
            else:
                print("No negative numbers, please.")
        while True:
            try:
                hpw = int(input("How many hours per week does the employee work?: "))
            except:
                print("Must be a number.")
            if hpw >= 0:
                break
            else:
                print("Please only use numbers for this entry")
        database.records.append((database.unique, sname, fname, jtitle, ft, rate, hpw))
        database.unique += 1
        write_database()

    elif sel == "b": # Display all employees
        if len(database.records) > 0: # Checks that there are records to print
            top_table(2)
            for record in database.records:
                full_name(record)
        else:
            print("There are no records to print")

    elif sel == "c": # Search for records by name
        match = input("Please enter the first name or surname of the employee you wish to search for:
").capitalize()
        top_table(7)
        found = False
        for record in database.records:
            if record[1] == match or record[2] == match: # prints record if input is equal to first
name or surname
                full_record(record)
                found = True
        if found == False: # Prints a message if there are no matches found
            print(("\n{:" "^140}").format('N0 MATCHES FOR "' + match + '" FOUND!'))

    elif sel == "d": # Delete an employee record
        rm = int(input("Please enter the unique ID number of the employee you wish to delete from
database: "))
        top_table(2)
        match = False
        for record in database.records:
            if record[0] == rm:
                print(("|{:" "^20}"*2 + "|").format(record[1], record[2]))
                delete = input("Are you sure you wish to delete? (Y/N): ")
                match = True
                if delete == "y":
                    database.records.remove(record)
                    write_database()
        if match == False:
            print(("\n{:" "^60}").format('N0 MATCHES FOR "' + str(rm) + '" FOUND!'))


    elif sel == "e": # Show all full-time employees
```

```
        if len(database.records) > 0: # Checks that there are records to print
            print("Full-time employees: \n")
            top_table(2)
            for record in database.records:
                if record[4] == "Yes":
                    full_name(record)
        else:
            print("There are no records to print")

    elif sel == "f": # Show all part-time employees
        if len(database.records) > 0: # Checks that there are records to print
            print("Part-time employees: \n")
            top_table(2)
            for record in database.records:
                if record[4] == "No":
                    full_name(record)
        else:
            print("There are no records to print")

    elif sel == "g": # Show all records in alphabetical order of surname or job title
        if len(database.records) > 0: # Checks that there are records to print
            while True:
                srt = input("Records sorted Alphabetically by [S]urname or [J]ob title?: ").lower()
                if srt == "s":
                    top_table(7)
                    for record in sorted(database.records, key = lambda record: record[1]):
                         full_record(record)
                    break
                elif srt == "j":
                    top_table(7)
                    for record in sorted(database.records, key = lambda record: record[3]):
                         full_record(record)
                    break
                else:
                    print("Invalid entry!")
        else:
            print("There are no records to print")

    elif sel == "h": # Show everyone who costs above a certain amount per week
        if len(database.records) > 0: # Checks that there are records to print
            while True:
                try:
                    twc = int(input("Please enter the total weekly cost you want to check against
employees: "))
                except:
                    print("Must be a number.")
                if twc >= 0:
                    break
                else:
                    print("No negative numbers, please.")

            highest = 0 # These two variables will hold the highest and lowest weekly cost employees
            lowest = 0
            print("\nEmployees who earn more than £{} per week:\n".format(twc))
            top_table(7)
            for record in database.records:
                total = record[6] * record[5] #Hours per week multiplied by hourly rate
                if lowest == 0:
                    lowest = total
                if total > highest:
                    highest = total
                if total > twc:
                    full_record(record)
                if total < lowest:
                    lowest = total
            if twc > highest:
```

```
                print("\nNo employees Found, the highest cost per week is £{}".format(str(highest)))
            elif twc < lowest:
                print("\nAll employees earn more than £{} per week".format(str(twc)))
        else:
            print("There are no records to print")

    elif sel == "x": # Exit
        break
print("Goodbye")
```

database.py (default):

```
#This is a list of tuples containing the records
records = [(2, 'Whitman', 'Jane', 'Worker', 'Yes', 9, 40), (3, 'Smith', 'Danny', 'Worker', 'Yes', 9,
40), (5, 'Johnson', 'John', 'Worker', 'Yes', 9, 40), (7, 'Baker', 'John', 'Worker', 'Yes', 9, 40),
(8, 'Thorne', 'Max', 'Security', 'No', 8.9, 14), (9, 'Smith', 'Steven', 'Head of dept.', 'Yes', 20,
30), (10, 'McGreeve', 'Larry', 'Accountant', 'No', 12, 15), (11, "O'Tally", 'Clair', 'Receptionist',
'Yes', 8, 40), (12, 'Garrett', 'Denise', 'Marketing', 'No', 13, 20), (14, 'Edison', 'Ash', 'Worker',
'Yes', 9, 40)]

#This is a number that increases with every entry but does not decrease with deletions, it is used to
assign a unique number to employees
unique = 17
```

Back to top