ISYS90088 Introduction to Application Development

Semester 2, 2018

School of Computing and Information Systems
The University of Melbourne

Assignment 2

Due date: Monday 15th October 2018, 11.30pm

This assignment is worth 20 marks, and will count as **20% of your final mark** in this subject. The assignment specification is on the ISYS90088 GROK environment, under the heading 'Assignment 2' (`https://groklearning.com/learn/unimelb-isys90088-2018-s2/ass2/0/`).

There are FIVE (5) questions in this assignment. The fifth question will require you to call the functions you wrote in the first four questions, but we will provide sample versions so that you can complete the fifth question even if you don't complete all the others.

This is an ***individual project***. While you may discuss the problems with your classmates, you must not show written solutions to another student or use written solutions from another student.

You are reminded that your submission for this project is to be your own individual work. For most people, collaboration will form a natural part of the undertaking of this project. However, it is still an individual task, and so reuse of code or excessive influence in algorithm choice and development will be considered misconduct. We will check submissions for originality and will invoke the University's Academic Misconduct policy (`http://academichonesty.unimelb.edu.au/policy.html`) where inappropriate collusion or plagiarism appear to have taken place. Your code will be passed through our plagiarism software.

Late submissions: A 10% penalty will be applied for each 'late' day and no late submissions will be accepted after 5 days from deadline. If you submit after the deadline, your submission will be treated as late and will be penalised.

Marking rubric: A sample marking rubric is uploaded on the LMS for your information.

> **Note:** Three types of automated test cases will be run against your submission on GROK: (i) example test cases from the examples given to you in the specification – you will see a tick mark if you pass them; (ii) hidden test cases—you won't see the test cases but you will receive a tick if your code has passed each of them; and (iii) assessment test cases, which you will not see, but the markers will see and use to assess your project. **Be careful!** GROK **will allow** you to submit code that does not pass tests.

Read the specification carefully and follow the instructions for each question.

Only assessment test cases will be used to calculate your mark, as outlined in the marking rubric. Make sure to use a good programming style that includes relevant comments and formatting of the code. Five (5) marks out of the 20 will be allocated to style, formatting and approach to solving the questions.

Good Luck!

Dr Antonette Mendoza, semester 2 2018

1 Background

Things to look out for in solving the questions are:

- don't be afraid to create extra variables, e.g. to break up the code into conceptual sub-parts, improve readability, or avoid redundancy in your code
- we also encourage you to write helper functions to simplify your code – you can write as many functions as you like, as long as one of them is the function you are asked to write
- commenting of code is one thing that you will be marked on; get some practice writing comments in your code, focusing on:
  1. describing key variables when they are first defined, but not things like index variables in `for` loops; and
  2. describing what 'chunks' of code do, i.e. not every line, but chunks of code that perform a particular operation, such as
     ```
     #find the maximum value in the list
     ```
     or
     ```
     #count the number of vowels.
     ```
- **be aware of tradeoffs** with extra variables, functions, and comments – whilst we encourage these, if we can't see the important parts of your code easily, it is a style issue and may be penalized.
- **start early** and **seek help** from the LMS discussion board or your tutor if you have trouble understanding what the question asks you to do, or if your output does not match expectations.

2 Welcome to the Census

The Census, a nationwide survey conducted by the Australian Bureau of Statistics (ABS), collects accurate data on the key characteristics of people in Australia and the dwellings in which they live. In 2016, the Census counted close to 10 million dwellings and approximately 24 million people, the largest number counted to date. The Census data provides a snapshot of Australia, showing how our nation has changed over time, and helps to estimate Australia's population. The data is used to distribute government funds and plan services for your community – housing, transport, education, industry, hospitals and the environment.

Congratulations! You have been appointed by the ABS as a programmer-analyst to make sense of the large volumes of data they collected in 2016 and conduct some data analysis to understand the population. As part of your new job, your manager has asked you to write four (4) functions that perform specific tasks, plus a 'main' function that uses all the other functions.

3 Census data format

The census data is given to you in one or more *comma-separated values* (CSV) files.

CSV is a simple file format which is widely used for storing *tabular data* (data that consists of columns and rows). In CSV, columns are separated by commas, and rows are separated by newlines (so every line of the text file corresponds to a row of the data).

Usually, the first row of the file is a header row which gives names for the columns.

> **Note:** If CSV uses commas to separate columns, what do we do when our data itself contains commas? The solution is to put quotation marks around the cell containing a comma. Python's `csv` library does this automatically.

The census data contains the following columns:

ID A unique ID assigned to each individual.

age The age of the individual.

salary The individual's annual salary.

suburb The suburb where the individual lives.

language The language spoken in the individual's home.

Here is a sample of the CSV data provided to you by the ABS:

```
ID,age,salary,suburb,language
P0,22,41838.0,St. Kilda,Chinese
P1,68,23242.0,Flemington,English
```

4 Getting the data into Python

In order to clean up and analyse the data, we need a way to take data from a CSV file and put it into a Python data structure. Fortunately, Python has a built-in `csv` library which can do most of the work for us.

You won't have to use the `csv` library directly, though. We will provide you with a helper function called `read_data` which uses the `csv` library to read the data and turn it into a *dictionary of dictionaries*. For example, suppose the data we saw in the previous slide:

```
ID,age,salary,suburb,language
P0,22,41838.0,St. Kilda,Chinese
P1,68,23242.0,Flemington,English
```

was stored in a file called `sample_data.csv`. To work with this data in Python, we would call

```
read_data("sample_data.csv")
```

which would return the following Python dictionary:

```
{
    "P0": {
        "age": "22",
        "salary": "41838.0",
        "suburb": "St. Kilda",
        "language": "Chinese"
    },
    "P1": {
        "age": "68",
        "salary": "23242.0",
        "suburb": "Flemington",
        "language": "English"
    }
}
```


> **Note:** Notice that all of the values in the nested dictionaries are strings, even the numeric values. If you want to use the values in numerical calculations, you will have to typecast them yourself.

Nested dictionaries can be confusing. Here are some simple examples of how to access data in a nested dictionary:

```
# save the data in a variable
data = {
    "P0": {
        "age": "22",
        "salary": "41838.0",
        "suburb": "St. Kilda",
        "language": "Chinese"
    },
    "P1": {
        "age": "68",
        "salary": "23242.0",
```

```
        "suburb": "Flemington",
        "language": "English"
    }
}

# how old is the first individual?
print(data["P0"]["age"])
# what language does the second individual speak at home?
print(data["P1"]["language"])
# what is the difference in salary between the two individuals?
print(float(data["P0"]["salary"])-float(data["P1"]["salary"]))
```

5 Workspace files

You'll notice that in this assignment there are multiple files in the Grok workspace (the area to the right of this slide, where you write your code). A quick explanation of these files is in order:

`program.py` The file where you will write your code. We have included a little bit of code in `program.py` to get you started.

`header.py` A file containing some useful functions and constants. We have already imported the relevant functions and constants for you in each question.

Various CSV files You will see some files in the workspace called `noisy_sample.csv`, `noisy_data.csv`, `cleaned_sample.csv`, or `cleaned_data.csv`. (The exact files vary from question to question.) These are census CSV files provided to you by the ABS. You can use them to test your functions as you work through the questions. The `sample` files are quite small (only a few lines), while the `data` files are relatively long (1000 lines).

6 Question 1: Signal To Noise (3 marks)

You have been provided with large CSV files containing census data about Melbourne. Unfortunately, the data is 'noisy': some people have made mistakes on the census form, or intentionally entered incorrect data. Your first task as a programmer-analyst is to clean up the noisy data for later analysis.

There are a few particular errors in this data:

- People have accidentally included too many zeroes in their salary, or entered salaries which are too large (or even negative!) as a joke. All salaries should be greater than or equal to $0, and less than or equal to `MAX_SALARY`, a variable which is given to you.
- People have mischievously entered a suburb which doesn't exist. The valid suburbs are listed in a variable called `VALID_SUBURBS`, which is given to you.
- Some people have entered their ages as words instead of digits, e.g. inputting 'twenty' instead of '20' – others have entered negative ages.

Write a function `clean_data(data)` which takes one argument, a dictionary of data in the format returned by `read_data`. This data has been read directly from a CSV file, and is noisy! Your function should construct and return a new data dictionary which is identical to the input dictionary, except that invalid data values have been replaced with `None`. **You should not modify the argument dictionary, `data`.**

For example, let's look at the data contained in `noisy_sample.csv`:

```
>>> data_noisy = read_data('noisy_sample.csv')
>>> for key, value in sorted(data_noisy.items()):
...     print(key)
...     print(value)
P1
{'age': 'eighty two', 'language': 'English', 'suburb': 'Toorak', 'salary':
```

```
'60196.0'}
P2
{'age': '49', 'language': 'Chinese', 'suburb': 'St. Kilda', 'salary':
'-16945514.0'}
P3
{'age': '54', 'language': 'Italian', 'suburb': 'Neverland', 'salary':
'49775.0'}
```

Clearly some of the values are invalid! Let's call `clean_data` on the data, and look at the result:

```
>>> data_cleaned = clean_data(data_noisy)
>>> for key, value in sorted(data_cleaned.items()):
...     print(key)
...     print(value)
P1
{'age': None, 'language': 'English', 'suburb': 'Toorak', 'salary': '60196.0'}
P2
{'age': '49', 'language': 'Chinese', 'suburb': 'St. Kilda', 'salary': None}
P3
{'age': '54', 'language': 'Italian', 'suburb': None, 'salary': '49775.0'}
```

Notice the `None` values in the nested dictionaries of the cleaned data.

You can assume the following:

- the input data dictionary does not contain `None` values;
- all salaries (both valid and invalid) will be strings that can be cast to `float`;
- all valid ages will be strings that can be cast to non-negative `int`s.

> **Testing your function**: We have included some code at the bottom of your file to make it easier for you to test your code. Don't worry if you don't understand all the details. When you open the Grok terminal, the code we've added will read the data from the file specified in `test_file` into a dictionary called `data_noisy`.
> To manually test your code, change the value of `test_file` to the file you want to use to test your code, then execute the following in the Grok terminal:
> `>>> data_cleaned = clean_data(data_noisy)`

> **Note:** For the remainder of the assignment, we will provide you with nice clean data for analysis, so you can work on later questions without completing this one. (In the real world, this probably wouldn't happen!) Remember that the clean data will contain some `None` values.

> **Be careful!** When GROK says your submission passed our tests, this only means **the submission was accepted for marking**. In this assignment we allow you to submit code that does not pass tests. Such code will not receive correctness marks. Therefore, pay close attention to the advisory messages that GROK gives after saying your submission passed our tests.

7 Question 2: Better Than Average (4 marks)

Write a function called `average_salary(data, lower_age, upper_age)` which takes a dictionary containing census data and an age bracket (specified by two `float` parameters, `lower_age` and `upper_age`), and calculates the average salary (as a `float`) for individuals in that age bracket.

You may assume the census data in `data` is 'clean', that is all invalid values have been replaced by `None`. If a nested dictionary contains a `None` value for the `salary` key or `age` key, you should ignore it in your calculation. (If the dictionary has `None` for a different key, e.g. `suburb`, you should still include it in the calculation.)

You may assume that both `lower_age` and `upper_age` are positive. Note that `lower_age` and `upper_age` are *inclusive* bounds, meaning an individual is in the age bracket if their age is greater than or equal to `lower_age`, and less than or equal to `upper_age`.

If `lower_age > upper_age`, your function should return `0.0`. If there are no individuals in the age bracket, again you should return `0.0`.

Here are some examples of what your function should return for different datasets and age brackets:

```
>>> data_cleaned = read_data("cleaned_sample.csv")
>>> average_salary(data_cleaned, 40, 50)
0.0
>>> average_salary(data_cleaned, 40, 54)
49775.0
>>> data_cleaned = read_data("cleaned_data.csv")
>>> average_salary(data_cleaned, 30, 50)
77963.76392572944
>>> average_salary(data_cleaned, 50, 70)
122097.66666666667
```

> **Be careful!** When GROK says your submission passed our tests, this only means **the submission was accepted for marking**. In this assignment we allow you to submit code that does not pass tests. Such code will not receive correctness marks. Therefore, pay close attention to the advisory messages that GROK gives after saying your submission passed our tests.

8 Question 3: Money, Money, Money (5 marks)

Your employers are interested in the distribution of wealth in the population. One way to analyse this is to divide the range of possible salaries into a number of equal-sized 'bins', where a bin is just a subset of the overall range, then count the number of salaries falling into each bin (if you've ever worked with histograms before, this should be very familiar).

For example, we could divide the total salary range ($0–$200,000) into ten bins: $0–$20,000, $20,001–$40,000, $40,001–$60,000, and so on, up to $180,001–$200,000. The distribution of salaries would then be summarised by 10 integers corresponding to the ten bins. In general, we experiment with the number of bins to find the number that gives the most informative distribution.

Write a function called `wealth_distribution(data, n_bins, max_salary)`, which calculates the distribution of salaries greater than or equal to 0 and less than or equal to `max_salary`, by dividing that range into `n_bins` bins and counting the number of salaries that fall into each bin. **The bin width should be an integer.** Your function should return a `list` of `int`s, with each integer representing the number of salaries falling in the corresponding bin.

If a nested dictionary in `data` contains a `None` value for the `salary` key, you should ignore it in your calculation. (If the dictionary has `None` for a different key, you should still include it in the calculation.)

You may assume that both `n_bins` and `max_salary` are positive integers. Notice that the first bin will always start at 0, and that including `max_salary` in the last bin may make the last bin slightly 'wider' than the others. For example, if `max_salary == 100` and `n_bins == 5`, the bins would be 0–19, 20–39, 40–59, 60–79, and 80–100.

Here is an example of how your function should behave:

```
>>> data_cleaned = read_data("cleaned_data.csv")
>>> wealth_distribution(data_cleaned, 10, MAX_SALARY)
[309, 106, 160, 160, 129, 40, 3, 1, 0, 0]
>>> wealth_distribution(data_cleaned, 20, MAX_SALARY)
[294, 15, 42, 64, 67, 93, 74, 86, 77, 52, 28, 12, 3, 0, 1, 0, 0, 0, 0, 0]
>>> wealth_distribution(data_cleaned, 10, max_salary=105259)
[294, 19, 50, 69, 74, 91, 87, 85, 74, 41]
>>> wealth_distribution(data_cleaned, 20, max_salary=105259)
[293, 1, 7, 12, 19, 31, 32, 37, 31, 43, 47, 44, 43, 43, 48, 38, 42, 32, 21, 20]
```

> **Be careful!** When GROK says your submission passed our tests, this only means **the submission was accepted for marking**. In this assignment we allow you to submit code that does not pass tests. Such code will not receive correctness marks. Therefore, pay close attention to the advisory messages that GROK gives after saying your submission passed our tests.

9 Question 4: Location, Location, Location (5 marks)

One way in which suburbs vary is in the age profile of their residents. Some suburbs will be more popular with students and young people, others will be more attractive to families with young children, others will be more appropriate for elderly residents, and so on.

Write a function called `location_age_counts(data, lower_age, upper_age)` which returns a dictionary of the number of individuals in the given age bracket for each suburb. That is, each key in the dictionary should be a suburb name, and the value for that suburb should be an `int` corresponding to the number of individuals in the suburb who fall in the age bracket specified by `lower_age` and `upper_age` (as in question 2, the bracket *includes* `lower_age` and `upper_age`). Your dictionary should have a key for all suburbs in `VALID_SUBURBS`, even the ones that have no residents in the age bracket.

In this question, you should ignore any nested dictionary with a `None` value for the `age` key or the `suburb` key. `None` values for other keys are acceptable. As with question 2, you may assume that `lower_age` and `upper_age` are positive. If `lower_age > upper_age`, your function should return a dictionary with a value of 0 for every suburb.

Here are some examples of how your function should behave:

```
>>> data_cleaned = read_data("cleaned_data.csv")
>>> location_age_counts(data_cleaned, 0, 17)
{'Dandenong': 29, 'Docklands': 19, 'Frankston': 35, 'Collingwood': 17,
'Brunswick': 16, 'Parkville': 27, 'Fitzroy': 23, 'Footscray': 31, 'Kensington':
27, 'Richmond': 22, 'St. Kilda': 16, 'Toorak': 8, 'Caulfield': 22,
'Flemington': 18, 'Southbank': 26, 'Hawthorn': 23}
>>> location_age_counts(data_cleaned, 18, 34)
{'Dandenong': 17, 'Docklands': 8, 'Frankston': 15, 'Collingwood': 10,
'Brunswick': 11, 'Parkville': 8, 'Fitzroy': 13, 'Footscray': 14, 'Kensington':
12, 'Richmond': 10, 'St. Kilda': 3, 'Toorak': 8, 'Caulfield': 8, 'Flemington':
8, 'Southbank': 12, 'Hawthorn': 8}
>>> location_age_counts(data_cleaned, 35, 44)
{'Dandenong': 13, 'Docklands': 17, 'Frankston': 20, 'Collingwood': 12,
'Brunswick': 11, 'Parkville': 19, 'Fitzroy': 18, 'Footscray': 19, 'Kensington':
13, 'Richmond': 13, 'St. Kilda': 14, 'Toorak': 11, 'Caulfield': 23,
'Flemington': 21, 'Southbank': 18, 'Hawthorn': 14}
>>> location_age_counts(data_cleaned, 99, 100)
{'Dandenong': 0, 'Docklands': 0, 'Frankston': 0, 'Collingwood': 0, 'Brunswick':
0, 'Parkville': 0, 'Fitzroy': 0, 'Footscray': 0, 'Kensington': 0, 'Richmond':
0, 'St. Kilda': 0, 'Toorak': 0, 'Caulfield': 0, 'Flemington': 0, 'Southbank':
0, 'Hawthorn': 0}
```


> **Be careful!** When GROK says your submission passed our tests, this only means **the submission was accepted for marking**. In this assignment we allow you to submit code that does not pass tests. Such code will not receive correctness marks. Therefore, pay close attention to the advisory messages that GROK gives after saying your submission passed our tests.

10 Question 5: Generation Y (3 marks)

A prestigious Victorian university has asked the ABS to produce a report on the financial status and living situation of 18–34 year olds in Melbourne. They have asked you to help them generate some of the data for this report.

Write a function called `main(datafile)` which takes a filename as an argument, which reads the census data contained in that file, cleans the data, and uses the data to print out some facts about 18–34 year olds. You should assume that the data in `datafile` is noisy. Your function should calculate and print out the following facts:

1. The average salary for 18–34 year olds.
2. A list of the suburbs in which 18–34 year olds live.

> **Note:** You will probably find it useful to call `read_data`, `clean_data`, etc. in your `main` function. To help, we have provided implementations of all the functions from preceding questions. These are imported at the top of `program.py`. **You do not need to copy code from previous questions.**

The average salary should be printed to **two decimal places**. Use *string formatting* (either the `format` method or old-style `%` formatting) to do this. You do not need to round the `float` values themselves.

Suburbs should be listed in **alphabetical order**, and only listed if they have **at least one 18–34 year old resident**. Next to the suburb name, in brackets, print the number of 18–34 year olds that live in the suburb.

Here is an example of what your function should print. Make sure your function matches the format exactly.

```
>>> main("noisy_data.csv")
Average salary for 18-34 year olds: $50139.66
18-34 year olds live in the following suburbs:
Brunswick (11)
Caulfield (8)
Collingwood (10)
Dandenong (17)
Docklands (8)
Fitzroy (13)
Flemington (8)
Footscray (14)
Frankston (15)
Hawthorn (8)
Kensington (12)
Parkville (8)
Richmond (10)
Southbank (12)
St. Kilda (3)
Toorak (8)
>>> main("noisy_sample.csv")
Average salary for 18-34 year olds: $0.00
18-34 year olds live in the following suburbs:
```


> **Be careful!** When GROK says your submission passed our tests, this only means **the submission was accepted for marking**. In this assignment we allow you to submit code that does not pass tests. Such code will not receive correctness marks. Therefore, pay close attention to the advisory messages that GROK gives after saying your submission passed our tests.

End of assignment.