P07 – Dictionaries

1. Vocabulary Trainer (optional)

Write a program that helps you to memorize English words.

The program has two tasks:

- Load the translations from a file into a Python dictionary. To start you can use the provided file “dictionary.txt”, which you can extend by any words you want to memorize.

```
# Python dictionary that contains the translations loaded from the file
{'key': ['Schluessel', 'Taste'], 'expression': ['Ausdruck']}
```

- While the user did not know all translations correctly, randomly choose an English word in the dictionary and ask the user for its translation. If the translation is correct you can remove the entry from the dictionary.

Hint: For one English word you can have multiple German translations. Note how this is reflected in the data structure above.

2a. Center of Gravity (selective: AV/VS)

Write a program that computes the center of gravity (CG) for your DA 40 aircraft and decides if the plane is allowed to take-off or not.

Background: Remember your program that calculates the take-off mass (TOM) of your plane (Lab P02, exercise 1.1 TOM calculations.) You can re-use the code here. In order to take-off not only TOM needs to be below the maximum structural take-off mass, but also the CG needs to be in a certain range. More precisely the CG needs to be between 2.40m and 2.53m depending on the TOM of the plane.

Physics Recap:

- The center of gravity can be computed as

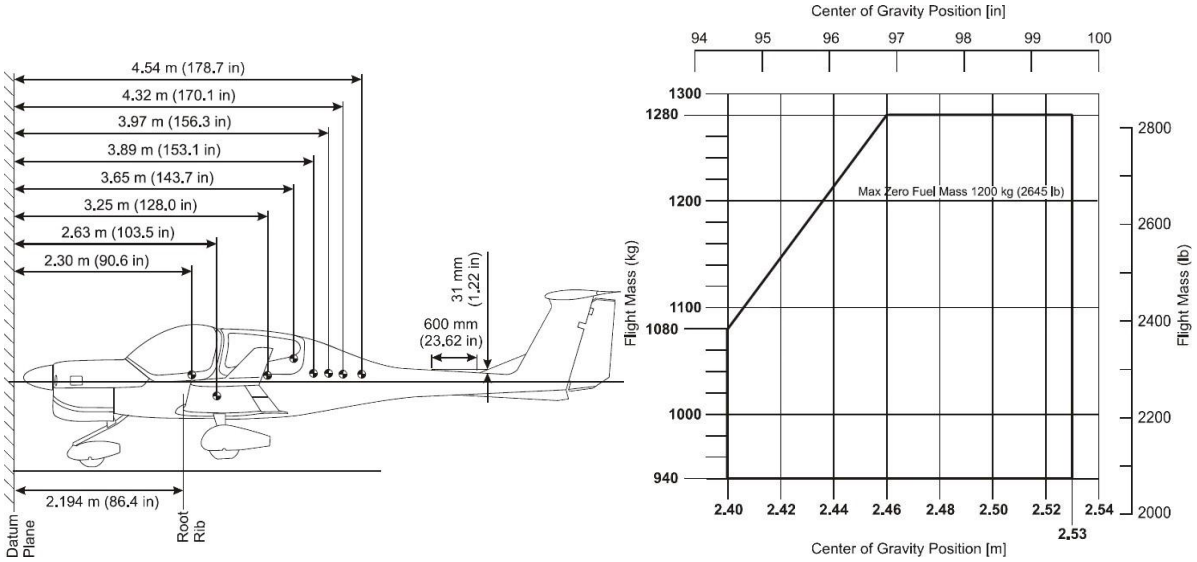
$$centerof\,gravity = \frac{totalmoment}{totalmass}$$

- The total moment can be computed by multiplying the masses with their distance to the planes front and adding them up.

```
total_moment = empty_moment +
               front_seat_mass * front_seat_distance +
                rear_seat_mass * rear_seat_distance +
                baggage_compartment_mass * std_baggage_compartment_distance + …
```

The distances to the plane's front are provided in the file "plane_data". It contains a dictionary-like structure, called JSON[1].
A file containing a JSON structure can easily be loaded into a dictionary.

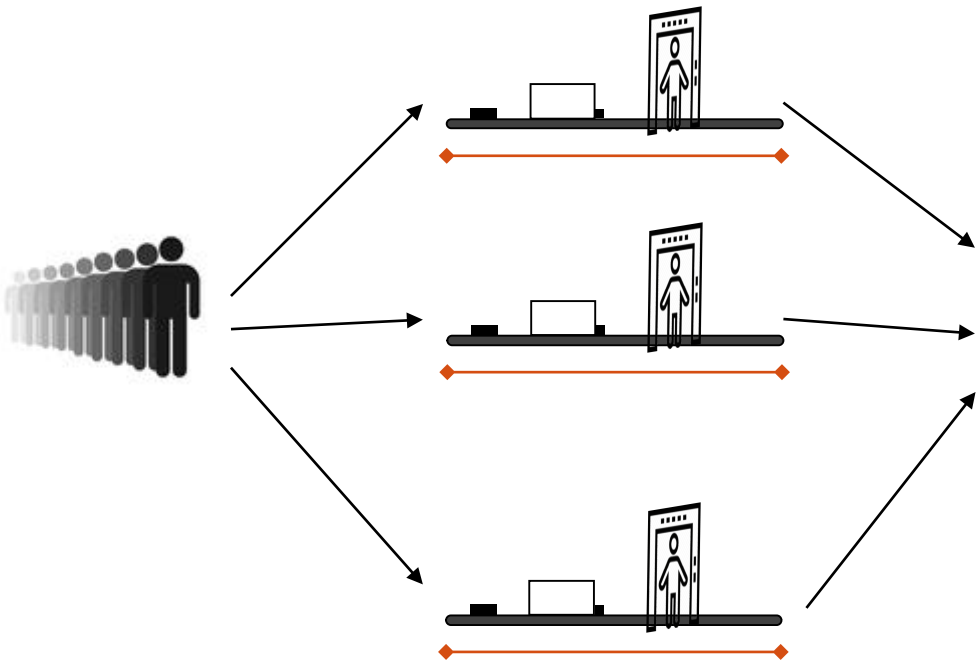
```
plane_data = open('plane_data')
plane_spec = json.load(plane_data)
```



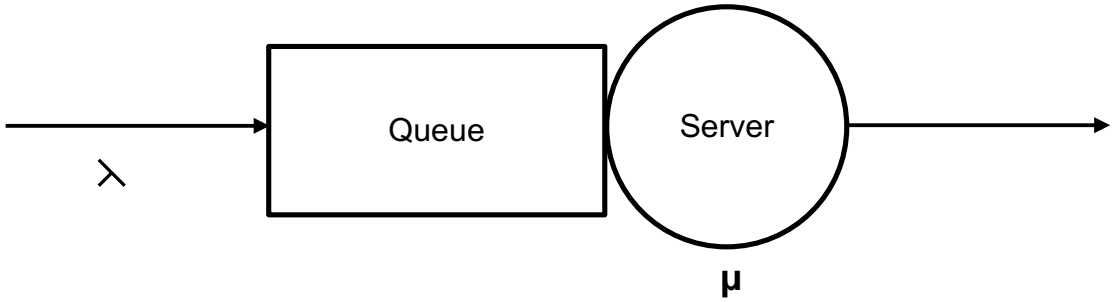
[1] Read more about the JSON format: http://en.wikipedia.org/wiki/JSON

2b. Airport security checkpoint (selective: WI)

In this exercise we will design the security area, where the passenger has to walk through an x-ray and his baggage scanned.



Queuing theory is a mathematical model, which can be used to predict queue lengths and waiting times.



where:

λ: *the average arrival rate*
μ: *the average service rate*

The utilization of the system with one server is:

$$\rho = \frac{\lambda}{\mu}$$

The following formulas are **only valid** for a queueing system with **one** server:

$$Average\ number\ of\ customers\ in\ the\ system: E[N] = \frac{\lambda}{\mu - \lambda}$$

$$Average\ number\ of\ customers\ in\ the\ queue: E[Nq] = \frac{\lambda^2}{\mu(\mu - \lambda)}$$

$$Average\ waiting\ time\ in\ the\ sysstem: E[W] = \frac{1}{\mu - \lambda}$$

$$Average\ waiting\ time\ in\ the\ queue: E[Wq] = \frac{\lambda}{\mu(\mu - \lambda)}$$

Your task

Write a program that reads a text file (**`settings.txt`**) with the information you need to calculate the average waiting time in the system and in the queue and the average number of passengers in the system and in the queue. Write an own function for every calculation. Then use all your functions and output the results on the console and in a new text file.

For simplicity reasons you will only do the calculations for a system with **one** security checkpoint (server).

Hints:

- To solve this problem, you first need to calculate the mean arrival rate [passenger / min]).
- Use the same units everywhere

3. Phone Book

Write a program that manages telephone numbers. It should be possible to

- add a new contact (name and number)
- remove a contact
- lookup a phone number for a given name
- save the contacts into a file
- load the contacts from a file

3.1 Basic Phone Book

You can use the template provided below. Implement the missing function bodies without altering the interface they comprise in conjunction with the given functions calls.

```
def print_numbers(numbers):
#Todo
def add_number(numbers, name, number):
    #Todo
def lookup_number(numbers, name):
    #Todo
def remove_number(numbers, name):
    #Todo
def load_numbers(numbers, filename):
    #Todo
def save_numbers(numbers, filename):
    #Todo

def print_menu():
    print '1. Print Phone Numbers'
    print '2. Add a Phone Number'
    print '3. Remove a Phone Number'
    print '4. Lookup a Phone Number'
    print '5. Load numbers'
    print '6. Save numbers'
    print '7. Quit'
    print

phone_book = {}
menu_choice = 0
print_menu()
while True:
    menu_choice = int(input("Type in a number (1-7): "))
    if menu_choice == 1:
        print_numbers(phone_book)
    elif menu_choice == 2:
        print "Add Name and Number"
        name = raw_input("Name: ")
        phone = raw_input("Number: ")
        add_number(phone_book, name, phone)
    elif menu_choice == 3:
        print "Remove Name and Number"
        name = raw_input("Name: ")
        remove_number(phone_book, name)
    elif menu_choice == 4:
        print "Lookup Number"
        name = raw_input("Name: ")
        print lookup_number(phone_book, name)
    elif menu_choice == 5:
        filename = raw_input("Filename to load: ")
        load_numbers(phone_book, filename)
    elif menu_choice == 6:
        filename = raw_input("Filename to save: ")
        save_numbers(phone_book, filename)
    elif menu_choice == 7:
```

```
        break
    else:
        print_menu()
```

3.2 Phone Book Pro (optional)

Implement at least two of the following extensions:

- Make it possible to save multiple numbers for the same person
- Make it possible to find people with typos
- Make it possible to find all that start with a letter
- Make it possible to save more data that belongs to that person, birthday, address, …
- Make it possible to save the data into an excel sheet

➔ *In any case, discuss all of your results and findings with your advisor before the end of the second week's lab session.*