**Software Workshop I**

**Assessment 1**

**Marks available: 50**

**Date of assessment: Mock**

**Set by: Jacqui Chetty**

**Background:**

You are tasked with completing a menu-driven application that focuses on the running of a zoo.

**Instructions:**

1. There is a template project that needs to be downloaded from Canvas. Make use of this to complete your solution. The template project consists of zooTemplate.py and zoo\_schedule.csv.
2. Once you have completed your solution, zip the project and upload it to Canvas.
3. When coding you may add more variables and / or functions, but you **must not remove existing ones**.

**Submission instructions:**

Submit your work by zipping the **project** (for example, Smith123456 – see step 1 above) and upload the .zip file to Canvas.

**Now you are ready to start your assignment solution:**

Please familiarise yourself with the template before starting. The template consists of several function header signatures that provide you with information about the function. The menu is as follows:

A black screen with white text

Description automatically generated

The instructions on how to complete each function begin now.

**The main menu ->main()**

This function consists of the menu and the calls to other functions. Most of main() has been completed for you. Where there are gaps to fill, these will be described as required. Take note of the first statement in main(). The zoo\_info() loads the data from the zoo\_schedule.csv file, as a dictionary, into the information{} found in main(). This is in case the user does not choose option 1 first, and the zoo information is not loaded into a dictionary. The coding of zoo\_info() will be explained below.

The main() also has a menu that runs as part of a while loop. There are several match / case each one calling functions.

The following are instructions to complete for each match / case.

**Menu option 1 -> zoo\_info() [12 marks]**

The goal of the zoo\_info()is to create a dictionary called information {} – seen in main().

Take note from main() - this function call takes place prior to the menu being generated. The first time it is called from main(), True is passed to the zoo\_info(). This is to ensure that the dictionary is created in case a user does not choose option 1 first. If True is passed to zoo\_info(), then the function must not print out a summary of the zoo information. Only the loading of the contents is required. This is why zoo\_info(first=False) forms part of the signature.

**Instructions for zoo\_info()**

1. Read the contents of zoo\_schedule.csv into the function.
2. Create a dictionary of all the zoo information - zoo\_info {} has been defined for you. Each row of data is as follows:

ZK001 – zookeeper id

Hema – name of the zoo keeper

09:00:00 - am feed for an animal

17:00:00 - pm feed for an animal

08:00:00 – start time of the zoo keeper  
The dictionary (one row shown here) looks as follows:  
{'ZK001': {'name': 'Hema', 'enclosure': 'Cats', 'feedAm': '09:00:00', 'feedPm': '17:00:00', 'start\_time': '08:00:00'}  
ZK001 is a key to a nested {} with name, enclosure etc each having a key pair.

1. If this is the first time that the zoo\_info() is loading the dictionary, then a message “Zoo information loaded” should be printed. This only happens **once**, before the menu is presented. Hint: make use of first=True to determine this.

A screen shot of a computer

Description automatically generated

1. If the user has selected option 1 then all the zoo information from the {} should be printed to the terminal the schedule displayed as follows:  
   A screen shot of a computer

   Description automatically generated

**Menu option 2 -> display\_times() [6 marks]**

This function allows the user to verify the feeding times for an animal. The call is made from main as display\_times() and the information {} is passed to the function.

**Instructions for display\_times(zoo\_info)**

1. If the dictionary does not exist, i.e. no zoo information, this must be displayed as a message.
2. Ask the user to enter an animal’s name, for example, cats or monkeys.
3. Allow for upper or lower case characters to be inputted.
4. The function must print out a message as follows:   
   “The cats have feeding times at 09:00:00 and at 17:00:00”.
5. Make sure to print a message if the animal does not exist.

**Menu option 3 -> check in()**

This option starts off in main() – as coded for you. There are 3 functions that must be completed, namely check\_key(), validate\_time() and check\_in().

In main() you will see that there is an input in case 3 asking the user to enter their keeper id. This is passed to check\_key() together with the information {}.

**Instructions for check\_key(keeper\_id, information) [4 marks]**

For check\_key() you need to ensure that the zookeeper exists. You will see that there is a msg = “” in the check\_key() that needs to be returned. If the zookeeper is found, then set msg to “Valid”. Otherwise, it remains null. In main() outcome holds either “Valid” or “”. The rest of the code is there.

**Instructions for check\_in(keeper\_id, information) [10 marks]**

1. You will see that the code to call validate\_time(check\_in\_time) is there for you. Go to the instructions for validate\_time – below - before continuing with this function.
2. You will notice that the “if not outcome: code” has been filled in for you.
3. Determine the difference between the start time of a zookeeper and the actual check in time of the zookeeper. This can be found in the zoo\_schedule.csv file as the last field.
4. You need to ensure that the zookeeper’s start time is converted to a proper datetime object / variable. Hint: make use of datetime.strptime().
5. If the zookeeper checks in after the start time then call a function logging\_late\_check\_in(id, msg). Otherwise, if the zookeeper has checked in on time, then print “checked in on time”. Instructions for logging\_late\_check\_n() found below.

**Instructions for validate\_time(check\_in\_time) [5 marks]**

1. The check\_in\_time is the time entered by the user. It should be in HH:MM:SS format. This function must validate the date format. Hint: use datetime.strptime() to validate if the time was entered correctly. Return either True if the time is in the correct format or False if not.
2. Go back to the instructions for check\_in() instruction 2.

**Instructions for logging\_late\_check\_in(name, msg)**

1. Write the late check in zookeeper to a late\_checkin.csv file.

**Menu option 4 -> report\_feeds\_complete() [13 marks]**

This option records feeds that have been completed.

**Instructions for report\_feeds\_complete (record)**

You will notice in main() that there is a record{} that is passed and returned from / to this function.

1. Ask the user for the enclosure name, such as cats, monkeys, etc.
2. Ask the user which feed – 1. for am or 2. for pm
3. These have been given to you.
4. Using the record{} to capture / record the feed using all\_fed[]. The record {} builds up a record of feeds in such a way that an animal has 2 recorded feeds. So, the cats [0,0] here have not been fed. When the morning feed has taken place the list changes to [1,0]. If both feeds have taken place, then the list is [1,1]. The first 0 indicates the morning feed and the second 0 indicates the evening feed. Depending on whether 1 or 2 has been entered, will determine which 0 changes. So, if a morning feed has taken place for the cats, it could look as follows: record[cats] = [1,0]. Also include a time\_feed and assign either “AM” or “PM” to it.
5. Once the feed has been recorded print a message to say that the “cats are fed for the PM feed”.
6. If both feeds have taken place, for example, cats: [1,1], append the record to all\_fed and write the feed data to a file “enclosure\_fed.csv”.
7. Return the record{} to main() so that the {} remains constant.