Parking (a) Lot

The first assignment focuses on processing parking lot records. You will review how to work with files and basic data types in Python, and you get experience with processing data in different formats. Apart from basic operations we will look at how we can transform data between various representations, statistical calculations, and optimization.

Your solution will work with data that you will load from two files – you can find examples in the `samples` folder. The first type of file is called `parking_logs_XX.csv`, where `XX` is the number of the sample, and it holds data in the comma separated values format. In these files, each line represents the parking of a car, while the line looks as follows:

```
TV333XI,6,2,16,46
```

where the first column has the car's license plate number, the next two columns define the arrival time (6:02 in our example), and the last two columns the time when the car left the parking lot (16:46 in our example). Columns are separated by commas and the last line in the file is empty. The records are ordered based on the car's arrival time at the parking lot.

The second file holds the parking fees and is called `prices_XX.txt`, where `XX` is the number of the sample. The file has lines in the following format:

```
30m: 0.5
```

where the value before the colon defines the length of the ticket and the price (in euros). Each file contains six keys, based on which you will calculate the amount of money to be paid for parking for a given time: `30m` – parking for half an hour; `1h` – parking for an hour; `3h` – parking for three hours; `6h` – parking for six hours; `1d` – parking for the entire day; `h+` – charge for each additional hour. The way the fee is calculated is described later on.

You will be implementing functions for loading data from files, calculating the parking fee, determining various statistical metrics, and optimizing the price for maximum operator revenue.

Task 1 – 0.75 points

Implement the function `load_parking_records(file_path)`, which reads parking records stored in CSV format from a file available at the given path (`file_path`), as described above. The method returns the loaded data as a list of tuples, where each tuple contains five values: license plate number, hour and minute of arrival, hour and minute of departure. The license plate number should be represented as a string, and the other values should be integers.

You will receive 0.25 points for the correct format and 0.5 points for correctly loading the values.

Task 2 – 0.75 points

Implement the function `load_prices(file_path)`, which loads the parking fees from a file located at `file_path`. The file will be a `txt` file with the format specified above. The method returns a dictionary with the loaded values. The keys will be short representations of the various time intervals (`30m`, `1h`, `3h`, `6h`, `1d`, `h+`), and the values will be the charges corresponding to these intervals. The keys should be strings and the values should be of type float.

In grading, you will receive 0.25 points for the correct format and 0.5 for the correctly loaded values.

Task 3 – 0.5 points

Implement the function `calculate_parking_time(start_h, start_m, end_h, end_m)`, which calculates the duration of parking in minutes based on the arrival and departure times received as parameters. The method has one return value: an integer representing the parking duration in minutes.

For example, calling `calculate_parking_time(7, 45, 11, 23)` will return a value of 218 (15 + 3 * 60 + 23).

Task 4 – 1 point

Implement the method `get_parking_fee(time_in_minutes, prices)`, which has two parameters:

- `time_in_minutes` – parking duration in minutes (as calculated by `calculate_parking_time()`)
- `prices` – a dictionary with the parking price list (from `load_prices()`).

The method returns the parking fee as a floating-point number, where parking for up to 15 minutes is free (returning `0.0`). If someone parks for less than half an hour, they will pay for a half-hour ticket (`30m`). In all other cases, they first pay the base price for parking and then pay extra for each started hour. For example, if someone parked for a total of 138 minutes (2 hours and 18 minutes), they would first pay an hour ticket (`1h`) and then pay twice for the extra started hour (`h+`). However, they will look for the more advantageous option, i.e. if the price for parking up to three hours (`3h`) is lower than the previously calculated amount, they will buy a ticket for three hours. The same applies to other categories (`3h` vs `6h` and `6h` vs `1d`).

Note: You can assume that two half-hour tickets cost the same as an hour ticket and that the extra charge for a started hour is less than the price of an hour ticket. If a customer starts a new hour, they must pay for the full hour; they cannot buy a half-hour ticket.

Task 5 – 0.5 points

Implement the method `calculate_average_parking_fee(records, prices)`, which calculates and returns the average amount paid for parking throughout the day based on the loaded records and the price list. The method has only one return value of type float. The `records` parameter has the structure according to the return value of `load_parking_records()`, and the `prices` parameter has the structure according to `load_prices()`.

Task 6 – 0.5 points

Implement the method `calculate_average_parking_time(records)`, which calculates and returns the average parking duration based on the loaded records. The method has only one return value of type float. The parameter `records` has a structure according to the return value of `load_parking_records()`.

Task 7 – 0.5 points

Implement the method `calculate_average_stays(records)`, which calculates and returns the average number of visits to the parking lot by a vehicle based on the loaded records. In some cases, you may find that the same car parked multiple times, and this method takes this into account. The method has only one return value of type float. The parameter `records` has a structure according to the return value of `load_parking_records()`.

Task 8 – 1 point

Implement the method `get_most_common_region(records)`, which finds and returns the most common district code based on the loaded records, taking into account multiple visits by the same vehicle. The method returns a single string - the two-letter district code from which cars visit the parking lot the most times (the first two letters of the license plate number). The parameter `records` has a structure according to the return value of `load_parking_records()`.

Note: If multiple districts have the same number of occurrences in the data, the method returns the code of the district that appeared first in the records.

Task 9 – 0.5 points

Implement the method `get_busiest_hour(records)`, which, based on the loaded records, finds and returns the hour when the parking lot is busiest. The `records` parameter has a structure according to the return value of `load_parking_records()`.

The method goes through all the hours when the parking lot is open and determines how many cars used the parking lot in that hour. Count a car if it arrived before or at the given hour and left at or after the given hour. For example, if a car arrived at 8:47 and left at 15:27, it will be counted for hours 8 to 15 (including). Determine the opening hours of the parking lot directly from the records. The method returns a single integer.

Note: If multiple hours have the same number of parked cars, the method returns the earliest of these options.

Task 10 – 2 points

Implement the method `get_max_number_of_cars(records)`, which, based on the loaded records, finds and returns the maximum number of cars that were parked in a minute. The `records` parameter has a structure according to the return value of `load_parking_records()`. The method returns two values - the maximum number of cars at any one time, and a list of the number of parked cars for each minute. Determine the opening hours of the parking lot directly from the records.

When counting cars, follow Python indexing rules: if a car arrived at 8:02 and left at 10:39, it will be counted for each minute from 8:02 (including) to 10:38. At 10:39, the car is no longer counted towards the number of parked cars.

Note: For an efficient solution, you can use the fact that records are sorted according to the arrival of cars in the parking lot. Also, remember to take into account that cars may have already left the parking lot.

Task 11 – 2 points

Implement the method `optimize_hourly_fee(records, prices)`, which based on the loaded `records` and `prices` determines the fee for additional hours started that will maximize the total revenue of the parking operator. The parameter `records` has the structure according to the return value of `load_parking_records()`, and `prices` according to the method `load_prices()`. The method returns one value: the optimal price for starting an additional hour as a float.

The optimal value sought must be higher than the price of a half-hour ticket and lower than the price of an hourly ticket. The price must also be a multiple of 10 cents, i.e. `X.X0 EUR`.

You can test your solution using the test suite available in `tests_1a.py`. Similar tests will be used to evaluate your solution, but there will be more of them.

The approximate length of the solution: *about 180 lines of formatted code without comments*.