What are we watching?

In the second assignment, you will help a group of friends choose which screening they will go to in the cinema. You will get experience with the basics of object-oriented modelling, creating classes and working with objects. You will use the following models:

- `Movie` - represents a movie;
- `Auditorium` - represents a movie theater;
- `Screening` - represents the screening of a movie in a specific auditorium at a specific time;
- `Cinema` - represents a cinema with auditoriums and screenings offered;
- `Person` - a person who wants to go to the cinema and wants to see films that belong to his/her interests;
- `FriendGroup` - a group of friends who want to go to the cinema together.

In addition to these classes, the solution contains the `constants.py` file with a list of valid movie genres that will exist in the solution. If you want to define additional helper variables to use as constants, you can do so in this file.

The classes in the solution will be interconnected, so we recommend reading the assignment all the way through first so that it is clear how the components are supposed to work together. The order of the class descriptions represents the recommended order of implementation.

`Movie` – 2.5 points

The `Movie` class represents a movie that is currently showing in theaters. Each movie is described by several values, which are set in the predefined constructor:
- `title` (`string`) - the title of the movie;
- `length` (`int`) - length of the movie (in minutes);
- `genre` (`string`) - the genre of the movie, which must be from the `constants.GENRES` list;
- `age_limit` (`int`) - age limit, i.e. the minimum age that cinema-goers must be to watch the movie;
- `release_date` (`string`) - the release date in `YYYY/MM/DD` format, for example 2023/03/20 for March 20, 2023.

You need to add the following functionality to the class:
- in the constructor, validate the parameter values before setting the values of the member variables as follows:
    - `title` must be of type `string`, otherwise generate a `TypeError` with the error message *Movie title must be string*;
    - `length` must be of type `integer` (otherwise generate a `TypeError` with the error message `Movie length must be integer`) and at least 1 (otherwise generate a `ValueError` with the error message `Movie length must be at least 1`);
    - `genre` must be from the `constants.GENRES` list, otherwise generate a `ValueError` with *Unknown genre "XX"* where you add the value of the genre parameter instead of XX (don't forget the quotation marks);

- `age_limit` must be of type `integer` (otherwise generate a `TypeError` with *Age limit must be integer*) and must be at least 1 (otherwise generate a `ValueError` with *Age limit must be at least 1*);
- `release_date` is validated in the separate `validate_date` method, but it must be called in the constructor itself.

- implement the `validate_date(date)` method, which is used to check the value of the `release_date` parameter, which must meet the following conditions:
  - it must be of type `string`, otherwise generate a `TypeError` with the error message *Release date must be string*;
  - must contain two `/` characters, otherwise generate a `ValueError` with the message *Release date must meet format YYYY/MM/DD*;
  - the first four characters before the first slash represent the year, the two characters between the two slashes represent the month, and the last two characters represent the release day - if you cannot convert any of these substrings to integer, generate a `ValueError` with the message *Could not load date from string: "XX"*, where instead of XX you print the original value of the parameter (don't forget the quotation marks);
  - if the number representing the month is not in the interval <1, 12>, generate a `ValueError` with the message *Invalid month XX*, where you add the loaded number instead of XX;
  - if the day number is not valid for the given month, generate a `ValueError` with the message *Invalid day for XX: YY*, where instead of XX you type the name of the month (i.e. January, February, March, ...) and instead of YY you type the day number you loaded from the string - for simplicity do not deal with leap years, for February numbers from 1 to 29 (inclusive) will be accepted.

- implement the `get_time_passed(date)` method, which will return the number of days elapsed from the movie's premiere date to the date it receives as a parameter (this has the same format as `release_date` when the constructor is called, and in the case of an invalid value, the same values will be generated as when checking the premiere date). The return value of the method is a single `integer` - the number of days elapsed, for the premiere date it will be 0.

Note: You do not need to check the input parameters for other classes, the tests will always use valid values.

`Auditorium` – 1 point

The `Auditorium` class represents a theater with two internal variables:
- `capacity` - the capacity (number of seats) of the auditorium, which is set in the constructor based on the parameter;
- `screenings` - a list of screenings in the auditorium, set as an empty list in the constructor.

You need to add the following methods to the class:
- `is_available(new_screening)` - the method checks if the auditorium is available at the given time; the input parameter `new_screening` is an object of type `Screening`. The method returns `True` if the hall is free at the given time, and `False` on the contrary. When determining occupancy, consider all screenings already added in a given hall, and check the start and end time of the screenings - no screening can start at a time when another screening is already in progress.

- `add_screening(new_screening)` - method will add a new screening to the list if it can take place at the specified time. The time will be part of the `new_screening` object. The method returns `True` if the screening has been added, `False` otherwise.

`Cinema` – 1.5 points

The `Cinema` class is used to represent a cinema, which is defined by two internal variables:
- `auditoriums` - a list of auditoriums, which is given to the constructor as a parameter;
- `screenings` - a list of screenings, initialized as an empty list in the constructor.

You need to implement the following methods in the class:
- `add_movie(movie, screening_times)` - the method will add several screenings of `movie` at the times given by the `screening_times` list. Each time is given as a tuple with two values representing hours and minutes (for example `(20, 30)` for 20:30). If an available auditorium is found for a movie screening at a given time, add the screening (a new object of type `Screening`) to the list `screenings`, otherwise generate a `RuntimeError` with the error message *Could not add movie XX at HH:MM*, where instead of XX you type the name of the movie, and HH:MM represents the time at which you did not find an available theater (write the values with leading zero if necessary - so for example 09:30, 09:05, 20:25, 15:30). The method has no return value.
- `get_movies_shown()` – the method returns a list of movies that the theater is showing. Thus, the return value of the method is a list of unique movies (objects of type `Movie`). The order of these movies does not matter.
- `get_screenings_for_movie(movie)` - the method returns a list of screenings of the movie it receives as a parameter movie (object of type `Movie`). The return value is a list of objects of type `Screening`, the order does not matter.

`Screening` – 1.5 points

Another class `Screening` is used to represent a specific film at a specific time in a specific hall. The following values are set in the constructor:
- `movie` - an object of type `Movie` - the movie that is being screened;
- `auditorium` - object of type `Auditorium` - the theater in which the screening will take place;
- `time` - a `tuple` with two integers - the time of the screening, the first value indicates hours and the second value minutes;
- `tickets_sold` - `integer` - number of tickets sold, initialized to 0 in the constructor.

In the class, complete the implementation of the methods:
- `sell_tickets(count)` - represents the sale of several tickets, their number is given by the `count` parameter. If the required number of tickets is still available (the capacity of the theater is not exceeded), the tickets will be sold, so you update the `tickets_sold` value. The method returns `True` if the tickets have been sold, otherwise it returns `False`.
- `get_occupancy()` - returns the occupancy of the screening as the ratio of tickets sold to the capacity of the theater. The return value is thus a single value of type float between 0 and 1.
- `get_end_time()` - returns the end time of the screening based on the start and length of the movie. The return value is a `tuple` with two integers indicating hours and minutes. Make

sure the minutes are from the interval <0, 59>, you don't need to normalize the hours (you'll never have a screening that ends after midnight).

`Person` – 0.5 points

The penultimate class you need to implement is the `Person` class, which represents a person who wants to visit the cinema. Each visitor is described by values:

- `interests` - a `list` of `strings` - a list of genres that the person is interested in (they would prefer to watch a movie belonging to their interests);
- `age` - `integer` - the age of the visitor, they can only come to a movie that has an age limit lower or equal to their age;
- `bedtime` - `integer` - represents the time when the person wants to be at home, or the screening must end before this time - specified only as an hour;
- `tolerance` - `float` - a value between 0 and 1, representing that at what level of crowdedness of the auditorium is the person still willing to buy a ticket. For example, if a person tolerates a crowdedness of up to 80% percent, but 90% of the tickets have already been sold, they will be reluctant to choose the given screening.

You need to implement methods in the class:

- `is_interested(movie)` - the method returns `True` or `False` depending on whether the genre of the movie it receives as a parameter (an object of type `Movie`) is one of the person's interests.
- `is_allowed(movie)` - the method returns `True` or `False` depending on whether the person is old enough to visit the movie given as a parameter (object of type `Movie`).
- `can_attend(screening)` - the method returns `True` or `False` depending on whether the person is already old enough to visit the movie that is screened (object of type `Screening`) and the screening ends before their bedtime (check only the hours). For example, if the screening ends at 23:00 or later and they have a bedtime of 23, the method returns `False`.
- `will_attend(screening)` - the method returns `True` or `False` depending on whether the screening (an object of type `Screening`) is still tolerable by the person (i.e., has not reached the crowdedness tolerance value).

`FriendGroup` – 2 points

The last component of our solution is the `FriendGroup` class, which represents a group of friends who want to go to the cinema together. The class is defined by a list of people (objects of type `Person`) who belong to it (member variable members, set directly in the constructor). We use the class to select the screenings that our group will attend using the following methods:

- `order_movies(cinema)` - the method receives as a parameter an object of type `Cinema`, i.e. the cinema the group will go to. Then from this cinema you need to get the list of movies that are currently showing (you don't deal with specific screenings yet). From this list, you select those movies that all members of the group can see - the consideration is whether they are old enough with respect to the age limit of the movie. You rank the potential films according to how many in the group are interested in the genre of the film. The return value will be a list of movies sorted by this indicator, so it will be a list of tuples, where each tuple contains first the movie and then the number of people in the group who are interested in the genre. The list will be sorted in descending order, i.e. the first position will contain the pair of values with the movie that most people in the group want to see and will not contain movies that someone in the group cannot see because of their age.

- `choose_screening(cinema)` - the method receives as a parameter an object of type `Cinema`, i.e. the cinema the group will go to. Then, from this cinema, a specific screening of any movie is selected so that as many people as possible are satisfied with the selection. The screenings are sorted and selected according to the following criteria:

    1. the screening that can be attended by as many people as possible given the age limit and the end time of the screening (if the screening ends after a person's bedtime, they would prefer not to come) is selected;
    2. if several screenings have the same first number of people, the screening that would be attended by the most people given the crowdedness of the auditorium is selected - here only take into account the number of people who would realistically want to come given the first point (so if someone cannot come to the screening because it is too late for them, don't count them even if the crowdedness of the hall would not bother them);
    3. if you still have the same number of people willing to come to multiple screenings, take the screening with a film that is more recent, i.e. fewer days have passed since its premiere.

    The method returns two values - the first value is an object of type `Screening`, i.e. the screening that was selected as the most convenient for the group. The second value is a list of all the screenings that came into consideration, where each element will be a tuple with the values: the screening (an object of type `Screening`), and the value of the three indicators listed above (all integer values). The list will be sorted in descending order with respect to the three indicators with the preferences listed (ascending order for the third parameter).

- `buy_tickets(sreening)` – the method represents the purchase of tickets for people from the group that will come to the screening (`screening` parameter) - their bedtime is after the screening is over. People's interests and tolerances are not taken into account here, as a proper friend, everyone will come even if they are not interested in the genre of the movie, or if the hall is too crowded. In the method, call the appropriate method from the `Screening` class to update the number of tickets sold. The method has no return value.

You can add additional internal variables to the classes, but you must leave the ones given by the assignment (otherwise the tests will not pass and you will get 0 points for that class).

You can get a maximum of 9 points for implementing classes. **Since the assignment was published late, if you get at least 3 points out of 9, one point will be added automatically.**