T1A3 TERMINAL APPLICATION

Valentinas Kornijenka

FEATURES

Feature 1: Generation of a workout schedule based on days and goal

Feature 2: Ability to customize workouts by adding, swapping and deleting exercisesAreas of growth

Feature 3: Ability to export workout to a text file

FEATURE 1: KEY CODE OVERVIEW

Generation of a workout schedule based on days and goal

(workout_generator.py)

```python
def _get_and_remove_random_exercise(self, exercise_list, excluded_names=set()):
    exercise_list[:] = [e for e in exercise_list if e.name not in excluded_names]
    if not exercise_list:
        return None
    selected_exercise = random.choice(exercise_list)
    exercise_list.remove(selected_exercise)
    return selected_exercise

def generate_workout(self, workout_days, goal_type):
    workout_schedule = []
    used_compound_exercises = set()
    body_part_counts = {part: 0 for part in self.body_parts}
    self.sets_reps_generator = SetsRepsGenerator(goal_type)

    for day in range(1, workout_days + 1):
        daily_plan = self.get_workout_plan_for_day(
            day, workout_days, used_compound_exercises, body_part_counts
        )
        workout_schedule.append(daily_plan)

    return workout_schedule
```

Gets workout from pseudo database and removes it to get rid of repetition

Initializing an empty list for the workout schedule

Creating a set to track used compound exercises to ensure no repetition.

Creating a set to track used compound exercises to ensure no repetition.

Initializing the sets and reps generator with the specified goal type

Iterating over each day to generate a workout plan.

```python
def get_workout_plan_for_day(
    self, day, workout_days, used_compound_exercises, body_part_counts
):  Initializing an empty list for the workout plan for the day..
    workout_plan = []
    compound_exercises_copy = self.compound_exercises.copy()
    accessory_exercises_copy = self.accessory_exercises.copy()

    If there's only one workout day, select up to 4 compound exercises
    if workout_days == 1:
        while len(workout_plan) < 4:
            compound_exercise = self._get_and_remove_random_exercise(
                compound_exercises_copy, used_compound_exercises
            )
            if compound_exercise:
                workout_plan.append(compound_exercise)
                used_compound_exercises.add(compound_exercise.name)
    else:
        compound_exercise = self._get_and_remove_random_exercise(
            compound_exercises_copy, used_compound_exercises
        )
        if compound_exercise:
            workout_plan.append(compound_exercise)
            used_compound_exercises.add(compound_exercise.name)
            body_part_counts[compound_exercise.body_part] += 1

        Keep adding exercises to the workout plan until it has 4 exercises
        while len(workout_plan) < 4:
            underworked_body_parts = [  Find body parts that have been worked the least.
                part
                for part, count in body_part_counts.items()
                if count == min(body_part_counts.values())
            ]
            possible_exercises = [ Filter accessory exercises based on underworked body parts
                ex
                for ex in accessory_exercises_copy
                if ex.body_part in underworked_body_parts
            ] Select a random exercise from the possible exercises
            accessory_exercise = self._get_and_remove_random_exercise(
                possible_exercises
            )

            if accessory_exercise:
                workout_plan.append(accessory_exercise)
                body_part_counts[accessory_exercise.body_part] += 1
```

```python
return {      You, yesterday • added code in workout_generator.py to ret
    "Day": day,   Return plan in dictionary format
    "Exercises": [
        {
            "Name": exercise.name,
            "Body Part": exercise.body_part,
            "Reps": reps,
            "Sets": sets,
        }
        for exercise, (sets, reps) in zip(
            workout_plan,
            [
                self.sets_reps_generator.get_sets_reps(e.type)
                for e in workout_plan
            ],
        )
    ],
}
```

FEATURE 2: KEY CODE OVERVIEW

Ability to customize workouts by adding, swapping and deleting exercises

(workout_displayer.py)

```python
@_handle_exception
def remove_workout(self, workout_schedule):
    day, to_remove = self._get_valid_day_and_exercise(workout_schedule, "remove")
    if day and to_remove is not None:
        del workout_schedule[day - 1]["Exercises"][to_remove]
```

Method to remove function based on user input.

```python
@_handle_exception
def modify_workout_plan(self, workout_schedule):
    options = {
        "1": self.remove_workout,
        "2": self.swap_workout,
        "3": self.add_exercise,
        "4": self.export_workout,
        "5": "exit",
    }

    while True:
        self.display_workout_schedule(workout_schedule)
        self._display_options(options)
        option = input("Select an option: ")
        if option in options:
            if option == "5":
                break
            options[option](workout_schedule)
        else:
            print("Invalid option. Try again.")
```

Method to modify the workout plan.

Dictionary to hold the user option menu

Infinite loop to offer the user option menu

Displaying the current workout schedule

Taking user input for selected option.

Checking for valid inputs

Checking if user wants to exit application

Method to add exercise

Use can choose between body part to work and if they want a compound or accessory movement

Finds exercise based on a filter

If not available prints a message and returns.

```python
@_handle_exception
def add_exercise(self, workout_schedule):
    day = self._get_valid_day(workout_schedule)
    body_part = input("Enter the body part for the exercise: ")
    exercise_type = input("Enter the exercise type (compound/accessory): ").lower()

    available_exercises = [
        e
        for e in workout_database.workouts
        if e.body_part == body_part
        and e.type == exercise_type
        and e.name
        not in [ex["Name"] for ex in workout_schedule[day - 1]["Exercises"]]
    ]
    if not available_exercises:
        print(f"No available exercises for {body_part} as {exercise_type}.")
        return

    new_exercise = available_exercises[0]
    sets_reps_generator = SetsRepsGenerator(
        workout_schedule[day - 1]["Exercises"][0]["Sets"]
    )
    sets, reps = sets_reps_generator.get_sets_reps(exercise_type)

    workout_schedule[day - 1]["Exercises"].append(
        {
            "Name": new_exercise.name,
            "Body Part": new_exercise.body_part,
            "Reps": reps,
            "Sets": sets,
        }
    )

    print(f"Added {new_exercise.name} for Day {day}.")
```

Method to swap exercises.

Method to swap exercises.

User can either choose to have the same body part as they're replacing or one of their choice.

Appends exercise to schedule.

Finds new exercise in pseudo database and if it's new it swaps it.

```python
@_handle_exception
def swap_workout(self, workout_schedul
    day, to_swap = self._get_valid_day_and_exercise(workout_schedule, "swap")
    if day and to_swap is not None:
        swap_with_body_part = input(
            "Enter the body part to swap with (or type 'same' for the same body part):
        )
        if swap_with_body_part == "same":
            swap_with_body_part = workout_schedule[day - 1]["Exercises"][to_swap][
                "Body Part"
            ]
        new_exercise = next(
            e
            for e in workout_database.workouts
            if e.body_part == swap_with_body_part
            and e.name
            not in [ex["Name"] for ex in workout_schedule[day - 1]["Exercises"]]
        )
        workout_schedule[day - 1]["Exercises"][to_swap]["Name"] = new_exercise.name
        workout_schedule[day - 1]["Exercises"][to_swap][
            "Body Part"
        ] = new_exercise.body_part
```

FEATURE 3: KEY CODE OVERVIEW

Ability to export workout to a text file

(workout_displayer.py & workout_exporter.py)

```python
@_handle_exception
def export_workout(self, workout_schedule):
    filename = input("Enter the filename (or press enter for 'workout_plan.txt'): ")
    if not filename:
        filename = "workout_plan.txt"
    export_workout_to_file(workout_schedule, filename)
    print(f"Workout plan exported to {filename}!")
```

Method to export workout.

User can enter file name or choose default name.

Confirmation message.

Exports workout onto a text file.

```python
def export_workout_to_file(workout_schedule, filename="workout_plan.txt"):
    with open(filename, "w") as file:
        for daily_plan in workout_schedule:
            day = daily_plan["Day"]
            file.write(f"Day {day}:\n")
            for exercise in daily_plan["Exercises"]:
                file.write(
                    f"  {exercise['Name']} ({exercise['Body Part']})"
                    f" - {exercise['Reps']} reps, {exercise['Sets']} sets\n"
                )
            file.write("\n")
```

Formats document in an easy to read method.

APP USAGE EXAMPLE

```
Welcome to the Workout Application!
Choose your goal type (muscle gain/strength gain) or type 'exit' to quit: strength gain
Enter the number of workout days (1-5) or type 'exit' to quit: 2
Day 1:
  pull-ups (back) - 4 reps, 5 sets
  dumbbell pullovers (chest) - 6 reps, 3 sets
  hammer curls (arms) - 8 reps, 3 sets
  seated leg curls (legs) - 9 reps, 3 sets

Day 2:
  deadlifts (back) - 5 reps, 5 sets
  face pulls (shoulders) - 6 reps, 3 sets
  skull crushers (arms) - 5 reps, 3 sets
  front raises (shoulders) - 9 reps, 3 sets

Options:
1. Remove workout
2. Swap workout
3. Add exercise
4. Export workout
5. Exit
Select an option:
```

- The user is asked for their goal type and how many days they would like to workout.
- Then, they get a workout plan provided to them.
- They also get a menu to either use the app further or exit.

```
Options:
1. Remove workout
2. Swap workout
3. Add exercise
4. Export workout
5. Exit
Select an option: 1
Enter the day number: 2
1. deadlifts
2. face pulls
3. skull crushers
4. front raises
Choose an exercise number to remove: 1
Day 1:
  pull-ups (back) - 4 reps, 5 sets
  dumbbell pullovers (chest) - 6 reps, 3 sets
  hammer curls (arms) - 8 reps, 3 sets
  seated leg curls (legs) - 9 reps, 3 sets

Day 2:
  face pulls (shoulders) - 6 reps, 3 sets
  skull crushers (arms) - 5 reps, 3 sets
  front raises (shoulders) - 9 reps, 3 sets

Options:
1. Remove workout
2. Swap workout
3. Add exercise
4. Export workout
5. Exit
Select an option:
```

- The user gets several options to manipulate the provided workout plan.
- In this example, the user removed the deadlifts from Day 2.
- The user also has options to swap, add and export workouts.

WHAT I ENJOYED & CHALLENGES

- It was really fun to challenge myself!
- I learnt so much about Python doing this
- Using Trello for project management was really useful and I will definitely use it more in the future.

- I did found the project to be very challenging.
- In retrospective, I should have considered a minimum viable product more as my code ended up being very complex.
- I used OOP which potentially may have made writing code more difficult for my skill level.

THANK YOU