PORTFOLIO DEVELOPMENT PHASE - HABI TRACK

DLBDSOOFPP01 – Object oriented and functional programming with Python

Rafael Mutis-Maschmann, June 2024
Student number: 92204582

SOLUTION AND MAIN FEATURES

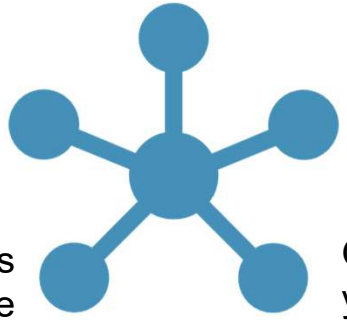
The solution Habi Track is an easy-to-use habit tracker which supports to establish new, desirable habits.

How does ist work? The success factor for implementing new habits depends on the routine. Only if the habit is performed regularly it becomes at long term a routine. Habi Track supports this process by defining, reminding and completing your dedicated habits.

The main features are:

Select a habit from a pre-defined list or create new ones

Select the frequency how often the habit shall be checked



Automatic creation of a habit task including due date

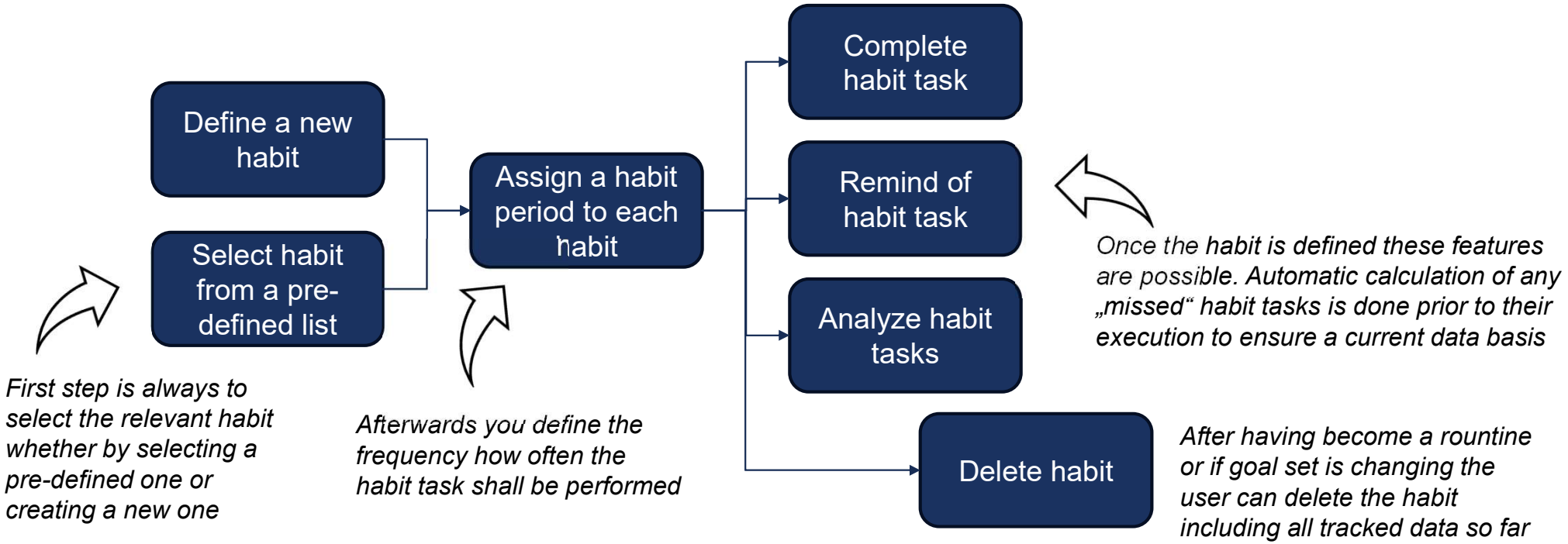
Complete the habit tasks before the due date

Get a reminder of your current tasks

Analyse how good you stick to your habits

PROCESS OF HABIT TRACKING

Following graphic shows the general steps for using Habi Track:



MODULES AND FUNCTIONS

The solution is was created in PyCharm Community Edition using Python 3.12 to create following main modules:

main.py
- Main module to be started by user
- Contains command line interface for interaction with user

habit.py
Contains class definition for habit and habit task including function for storage

analysis.py
Contains functions for data analysis

db.py
- Module for all database (DB) interactions
- Contains scripts for DB creation and data selection, update or insert

test_project.py
Module for unit tests

MODULES AND FUNCTIONS

main.py

- Main module which is executed by user
- It contains all relevant code concerning direct interaction with the user (see main options on right-handed side)
- Based on the answer selected the user will be navigated to a follow-up menu or the respective other modules will be used

- Interfaces to other modules:
  - Habit module for creating instances of habits and task & for using functions for storage
  - Db module for calling functions for database operations of already stored habits and tasks or for analysis purposes
- Usage of following Python packages:
  - Questionary: Easy to use for navigation of the user
  - Datetime: Using the exact time for task start and due date
  - Pandas: For further data analysis

```
def cli():
    choice = questionary.select(
        message: "Please select an option:",
        choices=["Select pre-defined habit",
                 "Create new habit",
                 "Complete habit task",
                 "Remind of current habits",
                 "Analyze habits",
                 "Delete habit",
                 "Exit"]
    ).ask()
```

MODULES AND FUNCTIONS

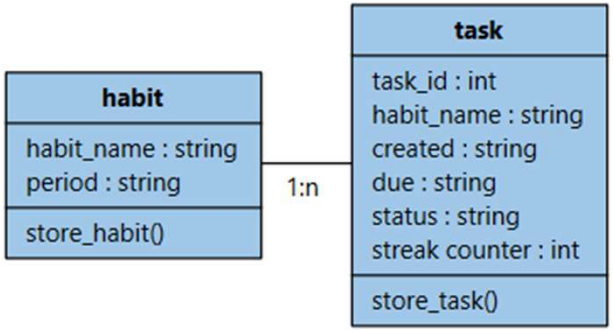
habit.py

The habit.py module contains a class definition for habits and their derived tasks. With creation of a habit the respective task will be added as well containg detailed information like due date, status and streak counter. Once the task is completed or missed a succesor task will be created. In consequence a habit can be linked with n tasks after some time. Besides the instantiation of habits and their tasks the classes also contain a function for calling an insert-function in the db-module.

- Interface to other modules:
  - Db module for storing instances of class habit and task
- Usage of following Python packages:
  - Datetime: Using the exact time for task start and due date

```python
class Habit:
    def __init__(self, name: str, period: str):
        self.name = name
        self.period = period

    2 usages
    def habit_store(self):
        insert_habit(self.name, self.period)
```



MODULES AND FUNCTIONS

analysis.py

Contains functions for data analysis e.g. calculation for run streak or percentage of completed habit tasks.

- Interface to other modules:
  - Db module to retrieve habit and task information from the db
- No usage of further packages

```python
def percent_calc(db):
    task_list = return_all_tasks(db)
    habit_list = return_habit(db)
    habit_list["Percent of completed habits"] = ""
    for x in range(0, len(habit_list)):
        task_cur_hab = task_list.loc[task_list["habit_name"] == habit_list.iloc[x, 0]]
        completed = task_cur_hab[task_cur_hab["status"] == "completed"].count()
        missed = task_cur_hab[task_cur_hab["status"] == "missed"].count()
        if completed.iloc[0] == 0:
            habit_list.iloc[x, 2] = "nan"
        else:
            habit_list.iloc[x, 2] = completed.iloc[0] * 100 / (completed.iloc[0] + missed.iloc[0])
    return habit_list
```

MODULES AND FUNCTIONS

db.py

The db-module contains all functions concerning database operations:

- Creation of the tables habit and habit_task
- Insert of new entries for habit and habit_task
- Update current entries, especially for habit_task
- Return all habits or habit tasks with a certain status

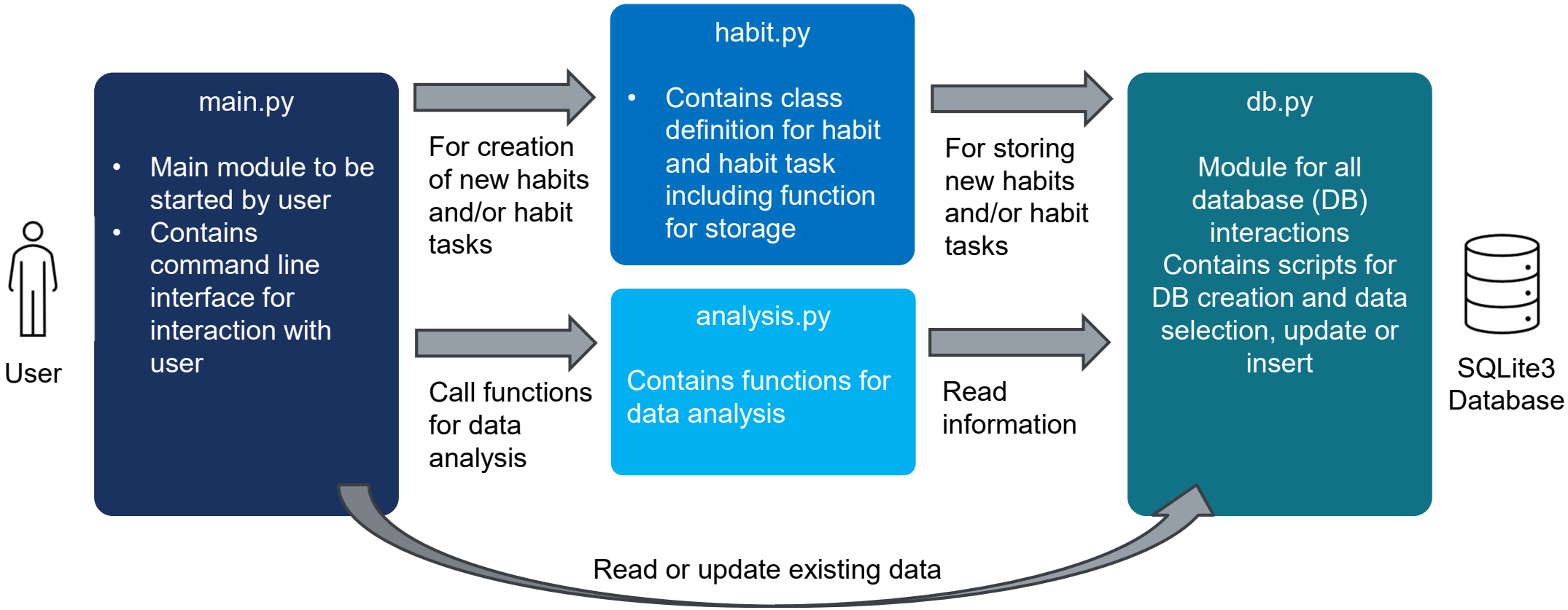
This module has no interfaces to the other modules
Usage of following packages:

- sqlite3: Easy to use database soultion for storing, retrieving and updating information
- Pandas: Used for returning multiple rows from the database via the function read_sql_query(). Handling of the results is more pleasant compared to cursor()-function

```python
# Function to insert a new habit task in the table habit_task
2 usages
def insert_task(habit_name, created, due, status, streak_counter):
    db = get_db()
    cur = db.cursor()
    cur.execute( __sql: "INSERT INTO habit_task "
                 "(habit_name, created, due, status, streak_counter)"
                 "VALUES (?, ?, ?, ?, ?)",
                  __parameters: (habit_name, created, due, status, streak_counter))
    db.commit()
    db.close()
```

MODULES AND FUNCTIONS

The following graphic shows the interaction between the modules:



MODULES AND FUNCTIONS

test_project.py

This module contains relevant code for unit tests of the complete application using pytest. The unit tests involve test data of two different habits for over a month to check that calculcations and analysis is done accordingly. For test purposes a dedicated test database is created only for the test run and just deleted afterwards.

- This module has interfaces to all other modules in order to re-use the exisiting code of the main modules
- Usage of following packages:
  - os: to delete the test database after the unit tests
  - freezegun: to simulate differenz points of time
  - datetime: used for checking if the calculated due times are correct

```python
@freeze_time("2024-04-25")
def test_success_seven():
    new_due_date, new_counter = success()
    assert new_due_date == datetime.now() + timedelta(days=2)
    assert new_counter == 2
```

FIRST VERSION AVAILABLE AT
HTTPS://GITHUB.COM/RMUTIS/OOFPP_HABIT_PROJECT

THANK YOU VERY MUCH

RAFAEL.MUTIS-MASCHMANN@STUDY-IU.ORG