The purpose of this project was creating an antivirus that can perform a full scan or a quick scan of a path, create or update an index of a path and enable or disable a regular quick scan of a path that is executed in given time intervals. There are a few assumptions about this antivirus that I have made: each scan has to be performed on a directory, quick scan requires an already existing index, there can be more than one regular quick scan of a certain path and if you want to disable it you have to enter the time_interval set when enabling. I have also implemented two methods of searching for viruses in files, which I will describe

There are two basic classes that this program uses. First one is `Database` which allows looking up virus data used in scanning. The other one is `File`, which contains default file status – "`Unknown`" and also a hashing function that generates a MD5 hash of a file from a given path.

Both `full_scan` and `quick_scan` use `check_file`, which calls `check_file_fh` function. The second one checks the file's contents with two methods.

- The first method uses a database with hashes of known virus files and compares them to the hash of the currently checked file.
- The other method reads the binary code of the currently scanned file and searches for a byte sequence from the database of virus sequences.

Function `check_file` receives information if a virus was found, if so it asks the user and reacts to the user's answer whether to fix a file (when it has a virus byte sequence) or to remove it (when its hash can be found in the database). It also returns information about the file's status and the user's answer to the scanning function it was called by.

Function `full_scan` first checks if an index for a given folder path already exists, if it doesn't it creates a new one. Then it proceeds to iterate the given directory and checks all of the files with the help of `check_file` function described in the previous paragraph. After checking a single file it updates the index. Function `full_scan` also makes sure if a file still exists after calling `check_file`, meaning it checks if the user didn't decide to remove this virus file, if in fact this file was a virus. If it was removed, `full_scan` either removes it from the index dictionary or doesn't add this file to the index dictionary. If this file still exists and isn't in the index dictionary, `full_scan` creates a new item in the index dictionary. Function `full_scan` returns information whether the directory contained any viruses, if it did, full_scan returns paths to infected files. For example:

- `full_scan` didn't find any infected files:

```
No viruses found.
```

- `full_scan` found infected files:

```
Found virus in:
/home/zgodek/project_pipr/antivirus/infectedfile.py
```

Function `quick_scan` differs from `full_scan` mainly by demanding an already existing index of a given path and while scanning it compares the hash of a currently checked file with the hash of the file saved in the index. If they are the same and the file was scanned previously it doesn't search the file for viruses.

Each index is a json file located in a folder given in the configuration file. It is created according to a simple pattern – as a dictionary of file paths, each one being a key to another dictionary with three keys: "`status`", "`hash`" and "`last_scanned`": `status` informs if a file was ever scanned and whether it was clean or was it fixed if it had a virus sequence, `hash` is just a MD5 hash of the file and `last_scanned` is the time when the file was last scanned. Example of an index item:

```
"/dir/with/files/file.py": {
      "status": "Clean",
      "hash_md5": "4de75381143b95eb9d6978c2550f90d1",
      "last_scanned": 1643061296.6349633
}
```

Updating an index is performed by first reading an old index of the given path and then iterating the given path. If the currently checked file is in the index dictionary then it checks if it was modified after being scanned. If so it updates the index by setting the status to "`Unknown`". Update index also adds new files to the index and removes non-existent files from the index.

An autoscan (regular quick scan) uses python-crontab library, which allows to create "jobs" and set time in what time intervals they are performed. In this case "job" is a quick scan of a path, which is performed in time intervals given in minutes. There is also an option to disable an autoscan for a given path, but you also have to specify the previously set time interval.

Functions such as: `full_scan`, `quick_scan`, `create_index`, `update_index`, `enable_autoscan` and `disable_autoscan` can be called by the user through the `main.py` file. Examples what user needs to enter for:

full scan:

```
python3 /path_dir_with_antivirus_files/main.py scan full "/dir/to/scan"
```

creating index:

```
python3 /path_dir_with_antivirus_files/main.py index create "/dir/create/index"
```

setting regular quick scan every 1 minute:

```
python3 /path_dir_with_antivirus_files/main.py autoscan enable "/dir/to/scan" 1
```

To run this program, the user needs to put all of the listed files: `file.py`, `database.py`, `index.py`, `scan.py`, `cron_set_up.py`, `main.py` and `config_database.json` in one folder. Then the user needs to create 3 different folders. One folder for files with MD5 hashes of known viruses, each hash has to be separated with a newline, but they can be spread over many files in this folder. The other folder is for files with virus sequences, each sequence has to be in a separate file. Finally the third folder is for the index files. Those three folders can be located anywhere in the system, but they can't be inside one another. Their paths need to be entered to the `config_database.json` file to their right keys, for example:

```
{"virus_hashes_path": "/folder1/virus_hashes", "virus_sequences_path": "/folder2/virus_sequences", "index_path": "/folder3/index"}
```

There is also one library that needs to be installed beforehand called 'python-crontab'. The rest is in the Python Standard Library.

When I reflect on the result and the progress of writing this program I think I'm least satisfied with the tests, which don't cover enough cases and they could have been written more elegantly. When it comes to the actual program I think I covered all of the functional aspects of the antivirus as listed in the project description with two different methods of identifying the infected files. Even though the code works, it isn't organized well enough, some parts are hard to understand and sometimes I would get lost in it, which is why, in retrospect, I think I should've planned out the structure of the code better. However I also think I've learnt a lot throughout this process – from file management to setting cron jobs and creating temporary files. This project has also taught me the importance of frequent committing and, as I've said before, the importance of planning out code before actually writing it.