**Title:** Python Programming assignment

**Name of the author:** Ivanov Stepan

**The date when the report was finished:** 31.10.2021

Python- and R-programming (GIK29B), Fall 2021

DALARNA UNIVERSITY

# Table of contents

# 1. Introduction

In this assignment I have created a program, which manages Notes.

Project was assembled with Microsoft Visual Studio and tested in Microsoft Visual Studio and IDLE (Integrated Development and Learning Environment). Didactic materials were mainly Tony Gaddis's "Starting Out with Python, fourth edition" (mostly chapter 10) and various open internet resources like "Github".

The most difficult part was importing classes from the working directory properly and dividing code on smaller parts to be more clear and object-oriented.

Based on the task at hand, I want to disclose the assumptions I have made during this problem-solving:

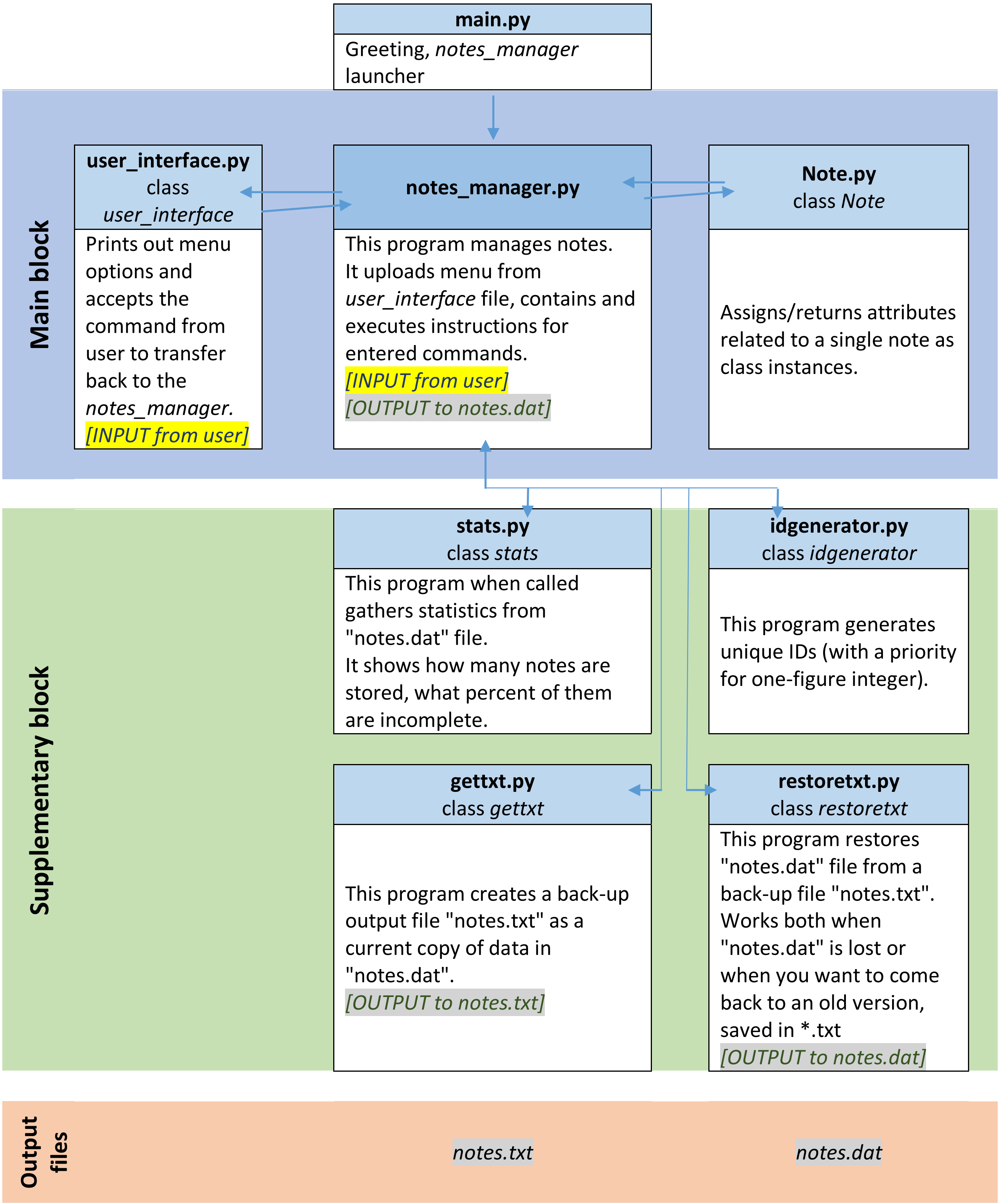
- The only editable fields of a note are its title and text. "ID" is unique for all the notes in database and is set automatically, "Date of creation" is a permanent auto-generated constant. "Date of completion", Boolean "Completed" and "Comment" are filled automatically when the note is actually completed.
- Regarding the note's Boolean feature 'completed' mentioned in the assignment – it is "True" when all the fields (Title, Text) are filled and "False" when text is missing.
- Text file "notes.txt" is used as a recovery file for a main "notes.dat" database.

# 2. Project description

In this section, I provide a description of my solution with classes defined and used.

## 2.1. Program structure in the form of a block diagram

The solution has the following structure (see picture 1).



**main.py**
Greeting, *notes_manager* launcher

**user_interface.py**
class *user_interface*
Prints out menu options and accepts the command from user to transfer back to the *notes_manager*.
*[INPUT from user]*

**notes_manager.py**
This program manages notes. It uploads menu from *user_interface* file, contains and executes instructions for entered commands.
*[INPUT from user]*
*[OUTPUT to notes.dat]*

**Note.py**
class *Note*
Assigns/returns attributes related to a single note as class instances.

**stats.py**
class *stats*
This program when called gathers statistics from "notes.dat" file.
It shows how many notes are stored, what percent of them are incomplete.

**idgenerator.py**
class *idgenerator*
This program generates unique IDs (with a priority for one-figure integer).

**gettxt.py**
class *gettxt*
This program creates a back-up output file "notes.txt" as a current copy of data in "notes.dat".
*[OUTPUT to notes.txt]*

**restoretxt.py**
class *restoretxt*
This program restores "notes.dat" file from a back-up file "notes.txt". Works both when "notes.dat" is lost or when you want to come back to an old version, saved in *.txt
*[OUTPUT to notes.dat]*

*Picture 1. Structure of the solution.*

The solution consists of 8 python files, 6 of which are presented as classes.

The main part of it includes *Notes manager*, *user interface* class and *Notes* class. They are designed to gather Notes as instances of class *Note* using *user interface*, which accepts input from the user and then proceed it with the Notes manager to either add, change or delete notes.

The supplementary part provides additional features, like gathering statistics of Notes stored (*stats* class), *ID generator* class and two other classes: for back up data to *.txt file and recovering from the same *txt file.

The main output file is a byte stream converted “notes.dat”, and “notes.txt” file is used as back-up.

### 2.2. Purposes and responsibilities of classes.

Now for each class created I will explain the purpose and responsibilities of it and collaborations with other classes.

| file / class / function | description | connections |
|---|---|---|
| **1. File main.py:**<br>1.1. import notes_manager | shows a welcome line<br>launches Notes manager | *notes_manager* |
| **2. File notes_manager.py:** | Sets global constants for menu choices, working file, receives menu from user_interface class, then processes input commands according to instructions. | All files/classes below are called from this file:<br>*Note, Idgenerator, user_interface, stats, gettxt, restoretxt* |
| 2.1. function main() | calls for local load_notes() function to set variable 'mynotes'.<br>Evaluates user's 'choice' variable, returned from *get_menu_choice()* function of user_interface class.<br>Depending on 'choice' value, the function executes one of the functions 2.2.-2.10. | *user_interface.get_menu_choice()* |
| 2.2. function load_notes() | local function for de-serializing input file using 'pickle' module. Returns data as dictionary '*notes_dict*'. | |
| 2.3. function save_notes(mynotes) | local function for serializing output file using 'pickle' module. Saved data to the database. | saves data to *'notes.dat'* |
| 2.4. function look_up(mynotes) | Accepts ID to look in 'mynotes' using default get() function. Shows all the information about a note, listed in Note class.<br>If ID is not found then prints out a corresponding message. | Picks data from *Note.__str__(self)* |

| file / class / function | description | connections |
|---|---|---|
| 2.5. function add(mynotes) | Adds a new note, accepting its *title* and/or *text* as variables to parse furhter to the *Note* class. Note's unique ID is automatically generated using *idgenerator* class. Date of creation is automatically generated using *datetime*module. Boolean variable 'completed' is set to 1 only if Note's text value is filled. If the note is completed, the function counts how many days it took to finish this note and assigns this value to the variable *message*. | Calls for main() function of *idgenerator* class.<br>Creates a note object in class *Note(id, date, title, text, completed, datecompleted, message)*. A new note will be saved in 'my notes' with ID.<br>Back in 2.1.<br>function *save_notes(mynotes)* is called right after. |
| 2.6. function change(mynotes) | Changes note (you may change either or both text and title of a note) searching for notes in 'my notes' by inputed ID. Depending on new note's attributes, it uprates corresponding values in the database.<br>Deleting values from a completed note will make it uncompleted. | Updates the note object in class Note(id, date, title, text, completed, datecompleted, message). A new attributes of the note will be saved in 'my notes'.<br>Back in 2.1. function save_notes(mynotes) is called right after. |
| 2.7. function delete(mynotes) | Deletes a note in 'mynotes' by its ID (if found). | A corresponding value in *'mynotes'* will be deleted.<br>Back in 2.1. function *save_notes(mynotes)* is called right after. |
| 2.8. stats(mynotes) | Gathers statistics from 'notes.dat' file.<br>Prints out two lines: how many notes are stored, what percent of them are incomplete. | mynotes' |
| 2.9. gettxt(mynotes) | Creates a back-up output file '*notes.txt*' as a copy of data in '*notes.dat*'. | output: to *'notes.txt'* from *'notes.dat'* |
| 2.10. restoretxt() | Using pickle module de-serialize data from *'notes.text'* and serialize back to '*notes.dat'*.<br>Works both when '*notes.dat*' is lost or when you want to come back to an old version, saved in *.txt.<br>Informs user if '*notes.text'* is empty or does not exist. | Recovers data to *'notes.dat'* from *'notes.text'*.<br>Back in 2.1. functions *load_notes()* and *save_notes(mynotes) are*called right after. |
| **3. File user_interface.py:** | Shows menu options, receives user's input on main menu. | |
| class user_interface | (includes one function) | |

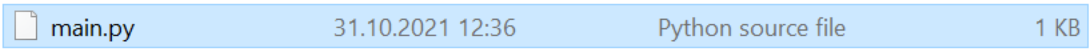
| file / class / function | description | connections |
|---|---|---|
| 3.1. get_menu_choice() | prints menu options, asks to input an integer value representing chosen menu option. Repeats until int value is received. | returns 'choice' variable back to *notes_manager* |
| **4. File Note.py:** | | |
| class Note | Includes 16 functions. | |
| 4.1. functions: __init__ (self, id, date, title, text, completed, datecompleted, message) | The _ _init_ _ method initializes the attributes of a single note. | |
| 4.2. functions set_id, set_date, set_title, set_text, set_completed, set_datecompleted, set_message | set note's attributes | |
| 4.3. functions get_id, get_date, get_title, get_text, get_completed, get_datecompleted, get_message | returns note's attributes | |
| 4.4. __str__(self) | represents class object. | |
| **5. File Idgenerator.py:** | Generates unique ID for a new note. | |
| class idgenerator | (includes one function) | |
| 5.1. function main() | First it generates a random number in range(0,9) to make ID search easy. It unpickles 'the notes.dat' to check if this value is already in it. If so, then it generates a number in a two-digit range. Modules 'random' and 'pickle' are used here. | inspects '*notes.dat*',<br>returns ID to the *notes_manager* |
| **6. File stats.py:** | By call shows some statistics of notes. | |
| class stats | (includes one function) | |
| 6.1. function stats(mynotes) | Shows a total number of keys in the dictionary 'mynotes' variable from *notes_manager.main()*, which is qual to the amount of notes saved.<br>Counts not completed notes in 'mynotes' using *get_completed()* function from class *'Note'.*<br>Shows percent of non-completed notes. | *notes_manager.main()*<br>*Note.get_completed()* |
| **7. File gettxt.py:** | Creates a back-up output file *'notes.txt'* as a copy of data in *'notes.dat'.* | Creates/updates '*notes.txt*' as a copy of current database. |
| class gettxt | (includes one function) | |
| 7.1. function gettxt(mynotes) | Using 'pickle' module it reads *'notes.dat'* file and copies its data to '*notes.txt*' | |
| **8. restoretxt.py** | This program restores '*notes.dat*' file from a back-up file '*notes.txt*'. | |
| class restoretxt | (includes one function) | |

| file / class / function | description | connections |
|---|---|---|
| 8.1. function restoretxt() + nested function load_notes() | Using 'pickle' module it reads '*notes.txt*' with nested function *load_notes()* and then re-writes '*notes.dat*' file. If '*notes.txt'* is empty does or not exist - the user will be notified. | |

## 3. User guide

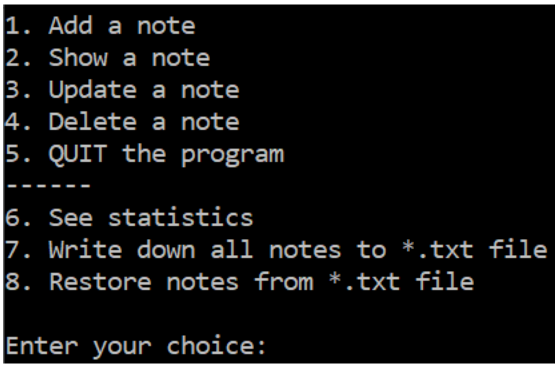
This program can save and edit your notes.

Launch “main.py” to initiate the program.



main.py 31.10.2021 12:36 Python source file 1 KB

Read down the appeared menu and enter in the command shell below your menu option number (1-8).



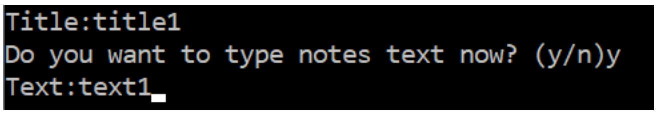
```
1. Add a note
2. Show a note
3. Update a note
4. Delete a note
5. QUIT the program
------
6. See statistics
7. Write down all notes to *.txt file
8. Restore notes from *.txt file

Enter your choice:
```

**Menu option #1** (Add a note):

Entering “1” you go into the section for adding notes.

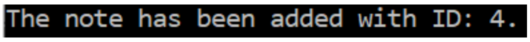
First, enter a title for you note, then you can proceed to the text of the note.

You can leave the text field empty and come back to it later using menu option #3 (Update a note).



```
Title:title1
Do you want to type notes text now? (y/n)y
Text:text1
```

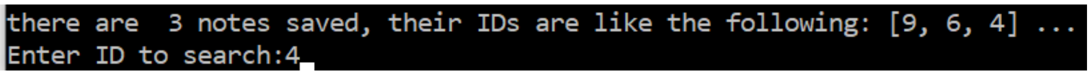
When you added a note, you will see that it is assigned an individual identification number (ID).



```
The note has been added with ID: 4.
```

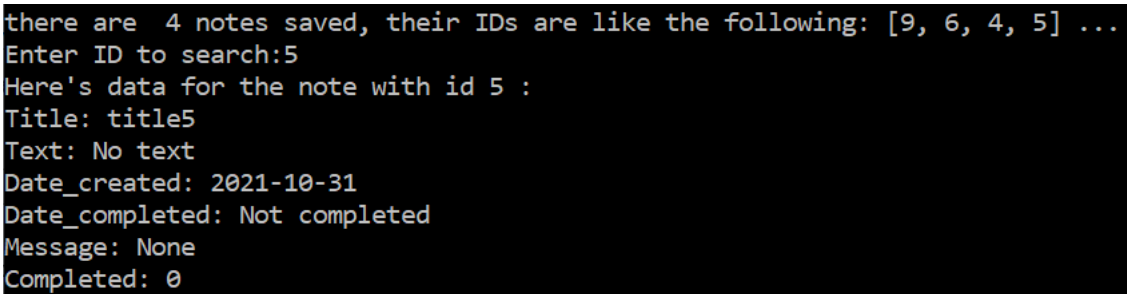
**Menu option #2** (Show a note):

To view the existing notes you can use the second menu option. As a hint, you will be given the first 10 IDs and a total amount of notes. Enter ID number of a note you wish to view.



```
there are  3 notes saved, their IDs are like the following: [9, 6, 4] ...
Enter ID to search:4_
```

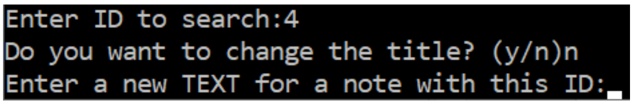
If ID you entered does exist in a base, you will be provided an information about it. You can see note's title, text, when it was created, completed and how long it took to be completed. If the note is not competed, its "Completed" status will be 0, "Date_completed" will be "Not completed" and "Message" will be "None".



```
there are  4 notes saved, their IDs are like the following: [9, 6, 4, 5] ...
Enter ID to search:5
Here's data for the note with id 5 :
Title: title5
Text: No text
Date_created: 2021-10-31
Date_completed: Not completed
Message: None
Completed: 0
```

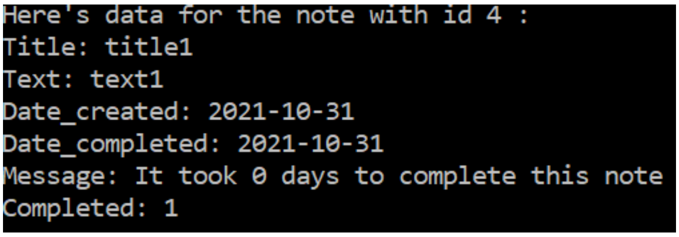
**Menu option #3** (Update a note):

This menu option will allow you to complete the note or to rewrite its title and/or text. Type ID and follow the instructions in command shell.



```
Enter ID to search:4
Do you want to change the title? (y/n)n
Enter a new TEXT for a note with this ID:_
```

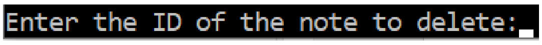
After you update note's title and/or text it will be saved under a same ID. If the note is completed, in the menu option #2 it will look like completed:



```
Here's data for the note with id 4 :
Title: title1
Text: text1
Date_created: 2021-10-31
Date_completed: 2021-10-31
Message: It took 0 days to complete this note
Completed: 1
```

**Menu option #4** (Delete a note):

Similarly, in this menu option type note's ID to delete the note.



```
Enter the ID of the note to delete:_
```

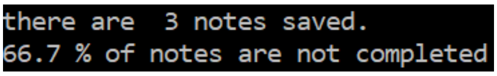
If ID you entered does exist in the base you will be notified that it was deleted.

**Menu option #5** (QUIT the program):

Entering "5" you terminate the program. All your changes so far are saved in your working directory ("notes.dat" file).

**Menu option #6** (See statistics):

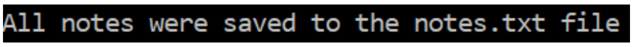
This menu option shows you how many notes are currently in the database and what percent of them are incomplete.



```
there are  3 notes saved.
66.7 % of notes are not completed
```

**Menu option #7** (Write down all notes to *.txt file):

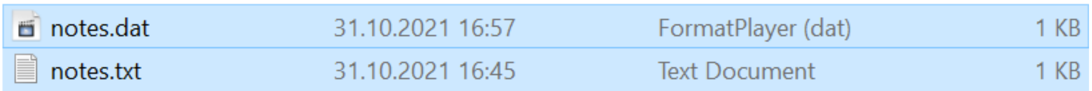
This menu option allows you to create a *.txt file to back up the existing database “notes.dat”.



```
All notes were saved to the notes.txt file
```

**Menu option #8** (Restore notes from *.txt file):

This menu option allows you to return to the database, previously saved in the “notes.txt” file. This option may be needed if you want to revert changes or if “notes.dat” file was lost.



| | | | |
|---|---|---|---|
| notes.dat | 31.10.2021 16:57 | FormatPlayer (dat) | 1 KB |
| notes.txt | 31.10.2021 16:45 | Text Document | 1 KB |