Methodological guidelines.

Smart Notes app. P. 2

🚀**STORYLINE:**

The Scientific Institute of Theoretical Physics has addressed to ProTeam specialists. A group of theoretical physicists requires an app for managing scientific notes. The scientists should be able to create, delete, edit, tag, and search for scientific notes.

During the working day, developers develop the app functionality, including the search for notes by tags.

⚠ **SUMMARY:**

The **lesson goal** is to apply knowledge and skills in storing data in files and developing applications using PyQT to develop a Smart Notes app.

In the lesson, students continue to develop the Smart Notes in the VS Code programming environment. In the first half of the lesson, students will program creating, deleting, and saving notes. In the second half, they will add capabilities to set a tag to a note delete it and search for notes by the specified tag.

💾 **LINKS AND ACCESSORIES:**

❏ Presentation,
❏ Lesson tasks: Smart Notes app (Visual Studio Code).

Note. During the next three lessons, the students will work on the same exercise.
Technical comment. To avoid incorrect displaying of project files on different operating systems, the files are set to utf-8 encoding. For correct processing of data by your computer, use the extra parameter encoding='utf-8'.

🎯 LEARNING RESULTS

After the lesson, the students will:	*The result is achieved when students:*
• understand how temporary program data structures and json file are connected; • know and apply commands for working with json files; • know the attributes for saving data to a json file (e.g. sort_keys); • display data from a json file in relevant widgets; • retrieve data from a json file using keys.	• can explain in their own words that it is better to solve simple tasks using ordinary text files, and json files are better suited for data with complex structures; • know all the commands for opening a json file, and for reading and writing data; • have programmed the Smart Notes functionality and set up a search by tags; • have answered the teacher's questions at the review stage.

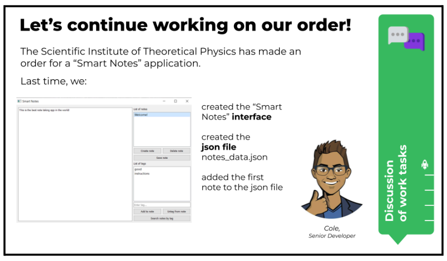
RECOMMENDED LESSON STRUCTURE

Time	Stage	Stage aims
5 min	**Storyline. Discussion: Smart Notes**	❏ Remind the content of the previous working day. ❏ Motivate the students to develop a large and interesting new app. ❏ Lead the students to an understanding that they need to review working with json files and methods of working with widgets.
10 min	**Qualification**	❏ Using the presentation, review of the following: ❏ json files; ❏ working with the app widgets.
10 min	**Brainstorming: Editing of Smart Notes**	❏ Discuss the structure of handler functions for creating, saving, and deleting notes. ❏ List the required tools.
20 min	**Visual Studio Code: VSC. The Smart Notes App**	❏ Organize interface development using VSC in the task “VSC. Smart Notes app.”
5 min	**Break**	❏ Do a warm-up or change students’ activity.
15 min	**Brainstorming: Working with tags of the Smart Notes app**	❏ Discuss the structure of handler functions for attaching a tag to a note, removing a tag, and searching for notes by tag.
20 min	**Visual Studio Code: VSC. The Smart Notes App**	❏ Have the students program storing of notes in the task “VSC. Smart Notes app.”
5 min	**Wrapping up the lesson. Reflection**	❏ Review the material learned. ❏ Propose an additional task and theoretical documentation.

Storyline. Discussion: Smart Notes app

(5 min)

Welcome the developers. Remind them that the interface of Smart Notes was created last time. The developers also created a json file for storing data, wrote the first note into it, and displayed information about it in the interface.



Let's continue working on our order!

The Scientific Institute of Theoretical Physics has made an order for a "Smart Notes" application.

Last time, we:

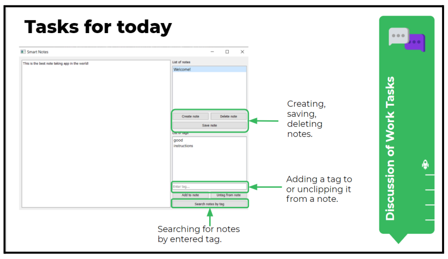
created the "Smart Notes" **interface**

created the **json file** notes_data.json

added the first note to the json file

Cole, *Senior Developer*

Discussion of work tasks



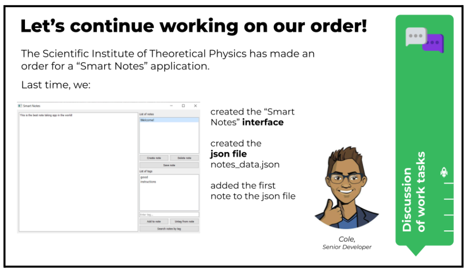
Tasks for today

Creating, saving, deleting notes.

Adding a tag to or unclipping it from a note.

Searching for notes by entered tag.

Today, the developers are to program the process of working with a set of notes and their tags. For that purpose, they need to create six functions: three — for creating, deleting, and saving notes, and the next three — for adding and deleting tags and searching by tag.



Let's continue working on our order!

The Scientific Institute of Theoretical Physics has made an order for a "Smart Notes" application.

Last time, we:

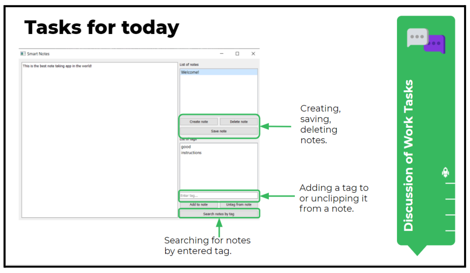
created the "Smart Notes" **interface**

created the **json file** notes_data.json

added the first note to the json file

Cole, *Senior Developer*

Discussion of work tasks



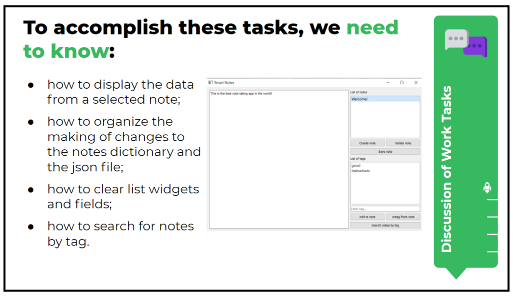
Tasks for today

Creating, saving, deleting notes.

Adding a tag to or unclipping it from a note.

Searching for notes by entered tag.

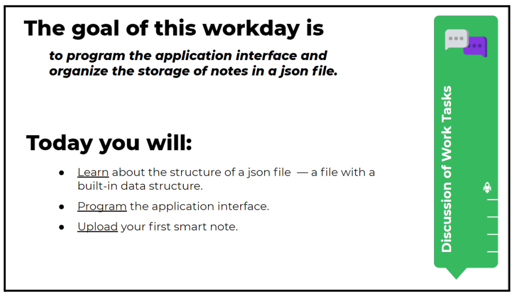
"To cope with the tasks set, you need to confidently work with files and correctly display data in widgets. For example, the function of creating a note should add a new note to the file and make changes to the interface."



To accomplish these tasks, we need to know:

- how to display the data from a selected note;
- how to organize the making of changes to the notes dictionary and the json file;
- how to clear list widgets and fields;
- how to search for notes by tag.

Discussion of Work Tasks



The goal of this workday is

to program the application interface and organize the storage of notes in a json file.

Today you will:

- Learn about the structure of a json file — a file with a built-in data structure.
- Program the application interface.
- Upload your first smart note.

Discussion of Work Tasks

After you have set the tasks, move on to review.

Qualification

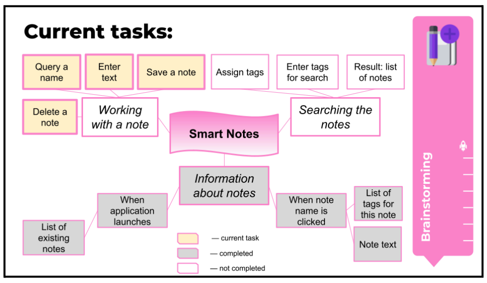
(10 min)

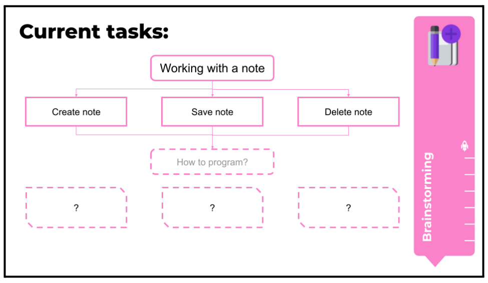
Qualify the developers' skills.

Brainstorming: Editing Smart Notes

(10 min)

Return to the discussion of Smart Notes. Use the mind map of the project to show the tasks for the first half of the lesson. Proceed to the discussion: “How to program the process of working with notes?”

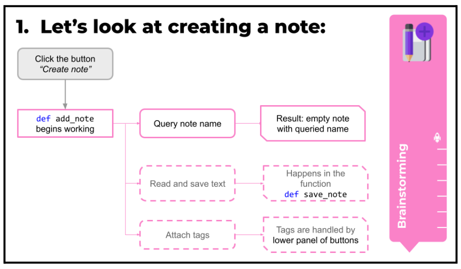


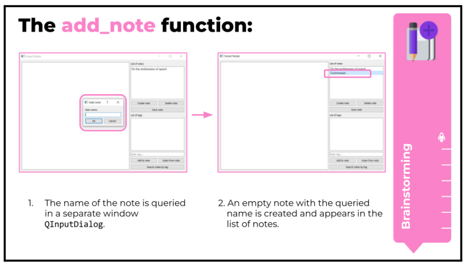


“What shall we program to be able to create our own notes, save them, and delete them in the app? (Students’ answers). *Yes, these actions should occur when we press respective buttons. We need to write a handler function for each of these buttons.”*

Move on to the overview of functions for working with notes. Show that when you click on “Create Note”, a blank note should be created. Other interface elements are in charge of working with text and tags.

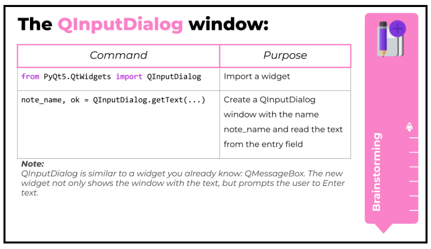
The name of the new note should be displayed in the general list_notes widget; therefore, you need to request a name from the user. Not to overload the app with another QLineEdit widget, suggest entering the name into QInputDialog (a separate pop-up window with an input field). It is similar to the QMessageBox widget you studied earlier, which showed the message in a separate window.


1. Let’s look at creating a note:
Click the button “Create note”
def add_note begins working
Query note name
Result: empty note with queried name
Read and save text
Happens in the function def save_note
Attach tags
Tags are handled by lower panel of buttons
Brainstorming



1. The name of the note is queried in a separate window `QInputDialog`.
2. An empty note with the queried name is created and appears in the list of notes.

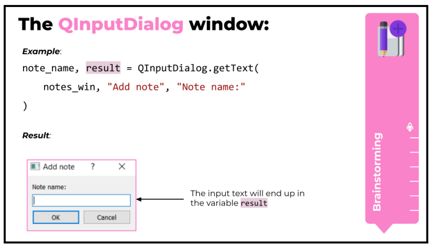
Demonstrate the getText() method to create a widget window and read a line of text. Show that the window title and text comment are set as parameters.



The QInputDialog window:

Command	*Purpose*
`from PyQt5.QtWidgets import QInputDialog`	Import a widget
`note_name, ok = QInputDialog.getText(...)`	Create a QInputDialog window with the name note_name and read the text from the entry field

Note:
QInputDialog is similar to a widget you already know: QMessageBox. The new widget not only shows the window with the text, but prompts the user to Enter text.



The QInputDialog window:

Example:

```
note_name, result = QInputDialog.getText(
    notes_win, "Add note", "Note name:"
)
```

Result:

Sum up what was said. Show the intended view of the add_note function. Please note that the presentation has a hidden slide with the full text of the add_note. Use it when working with weaker students.

Technical note. Recommend the developers to output intermediate results of the program operation to the console. For example, when adding a new note, you can not only show it in the widgets but also print the complete notes dictionary (it should appear there as well). Comparing console outputs and front-end views will help you avoid errors.

Similarly, review the del_note functions to delete the note and save_note to save the note.

Technical note. Please note that the save and delete functions make changes to the json file as well. Accordingly, after making changes, the notes dictionary must be rewritten to the file.

After the discussion, state the programming tasks for the first half of the working day and start programming.

Visual Studio Code: VSC. Smart Notes App

(20 min)

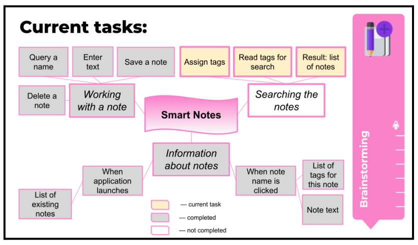
Arrange the work in the VS Code. Invite the developers to open the development environment and complete task 3 of the exercise titled VSC. Smart Notes App. When programming, it is allowed to consult with a neighbor.

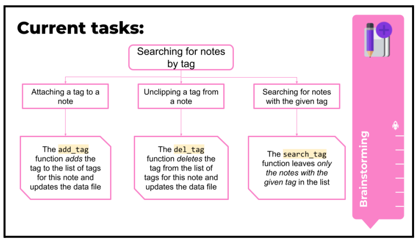
The task is to create three functions for editing notes. You will find the answer to the exercise at the end of the methodological guidelines.

Topic: Working with tags in Smart Notes

(15 min)

Once again, organize an analysis of the new topic using the presentation. State the tasks for the second half of the lesson. Show that the only thing that remains to complete the app is to program the tag search.




Current tasks:
Searching for notes by tag
Attaching a tag to a note
Unclipping a tag from a note
Searching for notes with the given tag
The add_tag function adds the tag to the list of tags for this note and updates the data file
The del_tag function deletes the tag from the list of tags for this note and updates the data file
The search_tag function leaves only the notes with the given tag in the list
Brainstorming

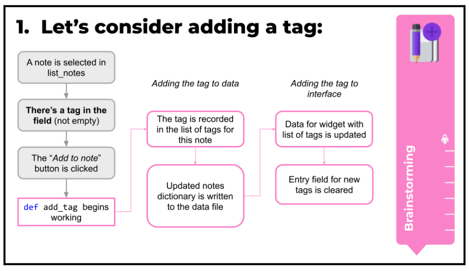
Proceed to discuss the functions. The add_tag and del_tag handler functions are similar to the note functions.

Pay the most attention to examining the search for notes by tag (search_tag). Break the search programming task into two sub-tasks: first, finding notes and displaying the result; second, resetting search results.

“How can the app interface prompt the user to reset the search results? Maybe make another button? (Students’ answers). *It is actually easier to use the same button. It is enough to change the caption on it from “Search by tag” to “Reset results”.*

What do we need to do so that when we click on “Reset results” a new search by tag does not occur (the button itself remains the same)? (Students’ answers). *Right, we need to program branching in the search_tag function. If the button caption is “Search by tag” and a tag is entered, then perform a search. If the caption is “Reset results”, then clear the tag input field and show the entire list of notes again. In all other cases, do nothing.”*

Use the presentation to show how the search_tag function works.



1. Let’s consider adding a tag:

A note is selected in list_notes → **There’s a tag in the field** (not empty) → The “*Add to note*” button is clicked → `def add_tag` begins working

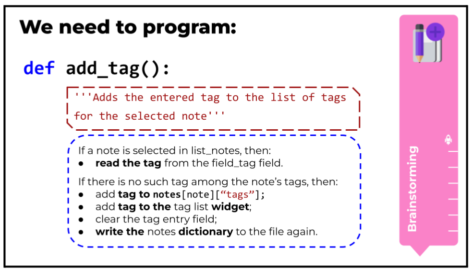
Adding the tag to data

The tag is recorded in the list of tags for this note → Updated notes dictionary is written to the data file

Adding the tag to interface

Data for widget with list of tags is updated → Entry field for new tags is cleared

Brainstorming



We need to program:

```
def add_tag():
    '''Adds the entered tag to the list of tags
    for the selected note'''
```

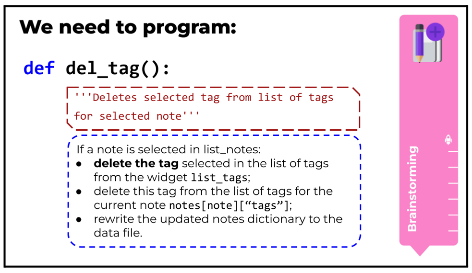
If a note is selected in list_notes, then:

- **read the tag** from the field_tag field.

If there is no such tag among the note’s tags, then:

- add **tag to notes**`[note][“tags”]`;
- add **tag to the** tag list **widget**;
- clear the tag entry field;
- **write the** notes **dictionary** to the file again.

Brainstorming



We need to program:

```
def del_tag():
    '''Deletes selected tag from list of tags
    for selected note'''
```

If a note is selected in list_notes:

- **delete the tag** selected in the list of tags from the widget `list_tags`;
- delete this tag from the list of tags for the current note `notes[note][“tags”]`;
- rewrite the updated notes dictionary to the data file.

Brainstorming

If the group is rather weak, use hidden slides with the finished code. Finalize the stage by stating programming tasks for the second half of the working day.

Visual Studio Code: VSC. Smart Notes App

(20 min)

Arrange the work in the VS Code. Invite the students to open the development environment and complete task 4 of the exercise “VSC. Smart Notes App.” When programming, it is allowed to consult with a neighbor.

The exercise involves creating of three functions to implement the search for notes with the entered tag.

Wrapping up the working day

(5 min)

Invite the developers to test the resulting app. Do not require deep testing (it will be discussed in detail later in the course), it is enough to answer the following questions:

1. Are all interface elements active?
2. Are notes created correctly? And are they deleted correctly?
3. Is it possible to tag a note? And to set multiple tags?
4. Is it possible to search for notes by the entered tag? Can you reset the search results?

Exercises answers

Exercise “VSC. Smart Notes App”

Task 3. Editing Smart Notes — Task 4. Search by a tag

```
from PyQt5.QtCore import Qt
from PyQt5.QtWidgets import QApplication, QWidget, QPushButton, QLabel, QListWidget, QLineEdit, QTextEdit,
QInputDialog, QHBoxLayout, QVBoxLayout, QFormLayout

import json

app = QApplication([])

#App interface
#app window parameters
notes_win = QWidget()
notes_win.setWindowTitle('Smart Notes')
notes_win.resize(900, 600)

#app window widgets
list_notes = QListWidget()
list_notes_label = QLabel('List of notes')

button_note_create = QPushButton('Create note') #a window with field "Enter note name" appears
button_note_del = QPushButton('Delete note')
button_note_save = QPushButton('Save note')

field_tag = QLineEdit('')
field_tag.setPlaceholderText('Enter tag...')
field_text = QTextEdit()
button_tag_add = QPushButton('Add to note')
button_tag_del = QPushButton('Unpin from note')
button_tag_search = QPushButton('Search notes by tag')
list_tags = QListWidget()
list_tags_label = QLabel('List of tags')

#arrangement of widgets by layouts
layout_notes = QHBoxLayout()
col_1 = QVBoxLayout()
col_1.addWidget(field_text)

col_2 = QVBoxLayout()
col_2.addWidget(list_notes_label)
col_2.addWidget(list_notes)
row_1 = QHBoxLayout()
row_1.addWidget(button_note_create)
row_1.addWidget(button_note_del)
row_2 = QHBoxLayout()
row_2.addWidget(button_note_save)
col_2.addLayout(row_1)
```

```
col_2.addLayout(row_2)

col_2.addWidget(list_tags_label)
col_2.addWidget(list_tags)
col_2.addWidget(field_tag)
row_3 = QHBoxLayout()
row_3.addWidget(button_tag_add)
row_3.addWidget(button_tag_del)
row_4 = QHBoxLayout()
row_4.addWidget(button_tag_search)

col_2.addLayout(row_3)
col_2.addLayout(row_4)

layout_notes.addLayout(col_1, stretch = 2)
layout_notes.addLayout(col_2, stretch = 1)
notes_win.setLayout(layout_notes)

#App functionality

#Working with note text
def add_note():
    note_name, ok = QInputDialog.getText(notes_win, "Add note", "Note name: ")
    if ok and note_name != "":
        notes[note_name] = {"text" : "", "tags" : []}
        list_notes.addItem(note_name)
        list_tags.addItems(notes[note_name]["tags"])
        print(notes)

def show_note():
    #receiving text from the note with highlighted title and displaying it in the edit field
    key = list_notes.selectedItems()[0].text()
    print(key)
    field_text.setText(notes[key]["text"])
    list_tags.clear()
    list_tags.addItems(notes[key]["tags"])

def save_note():
    if list_notes.selectedItems():
        key = list_notes.selectedItems()[0].text()
        notes[key]["text"] = field_text.toPlainText()
        with open("notes_data.json", "w") as file:
            json.dump(notes, file, sort_keys=True, ensure_ascii=False)
        print(notes)
    else:
        print("Note to save is not selected!")
```

```python
def del_note():
    if list_notes.selectedItems():
        key = list_notes.selectedItems()[0].text()
        del notes[key]
        list_notes.clear()
        list_tags.clear()
        field_text.clear()
        list_notes.addItems(notes)
        with open("notes_data.json", "w") as file:
            json.dump(notes, file, sort_keys=True, ensure_ascii=False)
        print(notes)
    else:
        print("Note to delete is not selected!")

#Working with note tags
def add_tag():
    if list_notes.selectedItems():
        key = list_notes.selectedItems()[0].text()
        tag = field_tag.text()
        if not tag in notes[key]["tags"]:
            notes[key]["tags"].append(tag)
            list_tags.addItem(tag)
            field_tag.clear()
        with open("notes_data.json", "w") as file:
            json.dump(notes, file, sort_keys=True, ensure_ascii=False)
        print(notes)
    else:
        print("Note to add a tag is not selected!")

def del_tag():
    if list_tags.selectedItems():
        key = list_notes.selectedItems()[0].text()
        tag = list_tags.selectedItems()[0].text()
        notes[key]["tags"].remove(tag)
        list_tags.clear()
        list_tags.addItems(notes[key]["tags"])
        with open("notes_data.json", "w") as file:
            json.dump(notes, file, sort_keys=True, ensure_ascii=False)
    else:
        print("Tag to delete is not selected!")

def search_tag():
    print(button_tag_search.text())
    tag = field_tag.text()
    if button_tag_search.text() == "Search notes by tag" and tag:
        print(tag)
        notes_filtered = {} #notes with the highlighted tag will be here
        for note in notes:
            if tag in notes[note]["tags"]:
                notes_filtered[note]=notes[note]
        button_tag_search.setText("Reset search")
        list_notes.clear()
        list_tags.clear()
        list_notes.addItems(notes_filtered)
        print(button_tag_search.text())
    elif button_tag_search.text() == "Reset search":
        field_tag.clear()
        list_notes.clear()
        list_tags.clear()
        list_notes.addItems(notes)
        button_tag_search.setText("Search notes by tag")
        print(button_tag_search.text())
    else:
        pass

#App startup
#attaching event handling
button_note_create.clicked.connect(add_note)
list_notes.itemClicked.connect(show_note)
button_note_save.clicked.connect(save_note)
button_note_del.clicked.connect(del_note)
button_tag_add.clicked.connect(add_tag)
button_tag_del.clicked.connect(del_tag)
button_tag_search.clicked.connect(search_tag)

#app startup
notes_win.show()

with open("notes_data.json", "r") as file:
    notes = json.load(file)
list_notes.addItems(notes)

app.exec_()
```