

## Methodological guidelines.

# The Memory Card app. Part 4

### STORYLINE:

ProTeam developers are finalizing an order from the “Citizen of the World” Cultural Center. To prevent the Center specialists from losing qualifications and navigate well in traditions of different nationalities and their languages, the Center ordered the Memory Card app. The app is supposed to ask user questions and give answer options stored in the program’s memory.

### SUMMARY:

The goal of the lesson is to finalize the Memory Card app by configuring the display of a random question from the list and collection of response statistics.

In the first half of the lesson, students program the display of a random question from the list and collection of response statistics. In the second half of the lesson, students fill the app with data, test, and arrange the result of their work as a collage.

**Note.** Depending on the group level, this lesson can be used not only to refine basic features but also to improve the app and to shape the result presentation skill.

### LINKS AND ACCESSORIES:

- [Presentation](#) for the lesson;
- Exercises: [Memory Card](#) (Visual Studio Code).

### LEARNING RESULTS

---

*After the lesson, the students will:*








- use object-oriented programming to optimize data storage;
- use the list data structure to work with a set of questions;
- use the random module to display a random question from the list;
- present the final product revealing its capabilities.

---

*The result is achieved when the students:*

- have participated in the discussion and asked clarifying questions;
  - configured the display of a random question;
  - have programmed statistics collection;
  - by the end of the lesson, prepared a complete solution for the Memory Card project;
  - have presented the app.
-

## RECOMMENDED LESSON STRUCTURE

Time	Stage	The tasks of the stage
5 min 	<b>Storyline.</b> <b>Discussion:</b> <b>"Memory card"</b>	<ul style="list-style-type: none"> <li>❑ Remind the storyline task: complete an app to memorize information for the "Citizen of the World" Cultural Center.</li> <li>❑ Discuss the details remaining to be implemented in the app and presentation of the work result.</li> </ul>
10 min 	<b>Brainstorming:</b> <b>"Finalizing the app"</b>	<ul style="list-style-type: none"> <li>❑ Set a task: display a random question from the list.</li> <li>❑ Arrive at the idea that it is required to finalize the next_question() function and formulate a solution.</li> <li>❑ Set a task: configure the collection of response statistics.</li> <li>❑ Arrive at the idea that it is required to introduce several app properties and display their value in the console.</li> </ul>
20 min 	<b>Visual Studio Code:</b> <b>"VSC. PyQt. Memory Card"</b>	<ul style="list-style-type: none"> <li>❑ Organize the finalization of the "VSC. PyQt. Memory Card" app in the Visual Studio Code environment.</li> </ul>
5 min 	<b>Break</b>	<ul style="list-style-type: none"> <li>❑ Help regain concentration while playing.</li> </ul>
10 min 	<b>Brainstorming:</b> <b>"Finalizing Work"</b>	<ul style="list-style-type: none"> <li>❑ Set a task: test and present the product developed to a ProTeam representative.</li> <li>❑ Describe the app testing process (data entry and performance verification).</li> <li>❑ Describe the process of arranging the work result into a collage (remote presentation of work via the Laboratory).</li> </ul>
20 min 	<b>Laboratory:</b> <b>"Testing and Presentation"</b>	<ul style="list-style-type: none"> <li>❑ Organize work on the product presentation and publication of the result in the Laboratory.</li> </ul>
15 min 	<b>Conclusion of the lesson.</b> <b>Reflection</b>	<ul style="list-style-type: none"> <li>❑ Invite developers to present their product according to a given scenario.</li> <li>❑ Suggest shooting a video presentation outside the working hours.</li> </ul>

## Storyline. Discussion: “Memory Card”

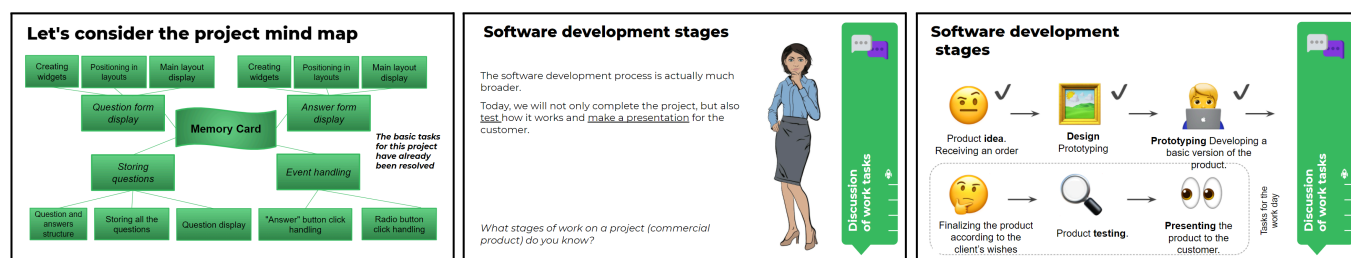
(5 min)

Open the presentation. The developers do not need computers yet.

*“Hello, colleagues! Today, we are finalizing a big order from the “Citizen of the World” Cultural Center. We have already programmed the basic functional features of the app. Today, we will add a couple of interesting mechanics and present the result to the customer.”*

Use the mind map to show that the basic work on the project is complete. Emphasize that developing a commercial product involves not only programming but also testing and presenting the project.

Show the approximate stages of work on the project. Tell the students that, depending on the project complexity and division of duties between developers, stages may get bigger or smaller, so testing and presentation may be different in other projects.



Formulate the goal of the working day and announce that the kids are going to complete work on the Memory Card app, test it, and make a report presentation.

## Brainstorming: “Finalizing the app”

(10 min)

Formulate two wishes for the app finalization that came from the “Citizen of the World” Center. It is required to change the order of displaying questions to random and supplement the app with a system for collecting statistics and calculating the user rating.

**Discuss** changing the order of questions.

- Remind that previously the questions were asked one by one. The number of the current question in the list was stored in the `cur_question` parameter, which was defined as a property of the app window. Now, we don't need this parameter
- How do we set a random order of questions? Can we stop using the app window property (with global visibility) and introduce a regular variable (locally in the function)? In which function should the question number change?
- How do we set the `cur_question` variable to the number of a random question? What is the maximum and minimum value it can take?

Show how the changes you have told about should be embedded in the general program code.

The image consists of three panels illustrating code changes for statistics collection and rating calculation in the Memory Card app.

- Panel 1: Statistics collection and rating calculation**
  - Task:** Add statistics and rating output to the console. Statistics means the current number of questions asked and correct answers given. Calculate the rating using the formula:  $\text{Rating} = \frac{\text{Number of correct answers}}{\text{Number of questions asked}} \times 100$ .
  - Code:** Shows the formula and a console output: "Statistics: Total questions: 1, Correct answers: 1, Total questions: 1, Correct answers: 1, Rating: 100.0%".
  - UI:** A screenshot of the Memory Card app window showing a question: "Which color does not appear on the American flag?" with options "Correct" and "Green". A "Next question" button is at the bottom.
- Panel 2: Statistics collection and rating calculation**
  - Task:** Add statistics and rating output to the console. Statistics means the current number of questions asked and correct answers given. Calculate the rating using the formula:  $\text{Rating} = \frac{\text{Number of correct answers}}{\text{Number of questions asked}} \times 100$ .
  - Code:** Introduces two accumulators: one for all the questions (total) and one for correct answers (score). To access them from different functions, let's make them window properties. Let's reset the accumulators when starting the program: `def __init__(self): self.window.total = 0; self.window.score = 0; self.next_question()`.
  - UI:** A screenshot of the Memory Card app window showing a question: "Which color does not appear on the American flag?" with options "Correct" and "Green". A "Next question" button is at the bottom.
- Panel 3: Changes in the program:**
  - Code:**

```
class Question():
    ...
    def check_answer():
        ...
    def next_question():
        ...
```
  - UI:** A screenshot of the Memory Card app window showing a question: "The state language of Brazil" with options "Brazilian", "English", "Portuguese", and "Spanish". An "Answer" button is at the bottom.

**Discuss** the answer statistics collection and user rating calculation.

- Demonstrate the expected functionality. Do we need to modify the application window interface or is it enough to output data to the console? What parameters do we need to monitor to print statistics and rating?
- How to determine (count) the current number of questions asked and correct answers given? Will these values change in one function only? How do we provide access to these parameters from different functions?
- Suggest defining these counters as properties of the window object (app window), the same way you used to set the number of the current question. In which functions do we need to change these values? When should they be printed? Where will the rating be calculated?

Show how the changes you have told about should be embedded in the general program code.

Wrap up and proceed to finalize the project in VS Code.

## Visual Studio Code: "VSC. PyQt. Memory Card"

(20 min)

Organize the students to complete an exercise on introducing additional mechanics to the Memory Card app. To continue work, you need to open the same exercise as last time.

The reference solution for the first half of the lesson can be found at the end of methodological guidelines.

**Procedural comment.** Pay special attention to slower students. In the case children do not have time to finish the app, you can assign testing and presentation of the project for homework as an extra optional exercise.

## Break

(5 min)

Switch the developers off their computers. The purpose of the break is to shift attention and warm up. Organize one of the [suggested physical activities](#).

## Brainstorming: “Finalizing Work”

(10 min)

Remind developers that the project does not end when the program ends. Announce two important stages: testing and presentation.

1. Begin with **testing** the product.

- Tell the students that the number and depth of testing stages depend on the product scale and resources available to the company. In this case, the testing mechanism has already been determined by the senior developer Cole; he made it up as a table and submitted it for review.
- Testing will comprise three stages. First, the developer fills the app with an extended set of data on the customer’s topic. Then he starts the app and works as a user with it. Is there a random order of questions? Do the answer options shuffle? Are statistics and ratings displayed? Do they correlate with the user’s actual results?
- If self-testing is successful, the project can be sent to another specialist for review.

**Procedural comment.** In the case of online lessons, the third stage of testing is not carried out.

### Product testing

There are different approaches to testing. For our Memory Card project, testing will be as follows:

Stage Name	Essence
Preliminary stage	<b>At least 10 questions</b> on the subject given by the customer are added to the program.
Self-testing	The developer starts the app and interacts with it as a user: <ul style="list-style-type: none"> <li>• <b>External attributes test</b> (randomizing questions and answer options, displaying the correct answer)</li> <li>• <b>Functionality test</b> (correct statistics calculation)</li> </ul>
Third-party testing	The project is <b>given to another specialist</b> for testing. They check how the program works and provide feedback.

### Presenting the product

In addition to implementing a project, a good developer must also be able to present the result to the customer.

**A report on the work done** will be sent to the “Citizen of the World” Center **via the Laboratory**.

We will need to:


- ☐ Arrange the work result **visually**
- ☐ **Demonstrate important mechanics**
- ☐ **Demonstrate test results**

For a short and effective presentation you can use:

- graphics editors
- presentation apps
- video editing
- and any other tools!

### Presenting the product

The program interface and how it works can be presented in a collage.



Explain the purpose of the app.

Demonstrate the interface elements.

Show off the key mechanics.

Report the test results.

2. Proceed to the discussion of the **result presentation**.

The work done can be presented in different formats. Since the Cultural Center worked with ProTeam in the remote mode, the result presentation will be virtual as well.

Invite the developers to create their report presentation in any service they know. If a developer does not know how to make presentations, offer a simple and free collage service.

The result must be downloaded and published in the Laboratory.

## Laboratory: “Testing and Presentation”

(20 min)

Organize app testing and presentation of the work done.

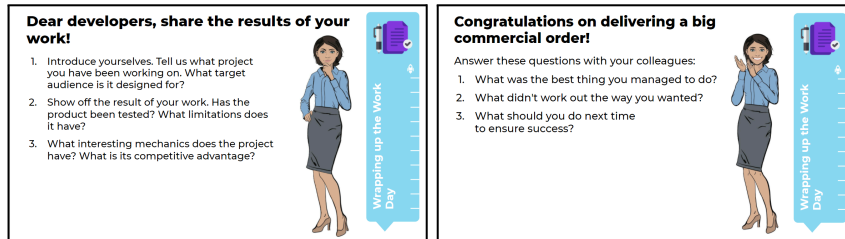
The result of this stage must be published in the Laboratory.

## End of the working day. Reflection

(15 min)

Invite the developers to share their work, show the test results (especially with unusual sets of questions), and the product presentation collage.

Request the developers to provide feedback after such a large project.



Assign an extra task to the interested developers: to shoot a video of their product presentation at home.

## Exercises answers

### Exercise “VSC. PyQt. Memory Card”.

#### 1.1. Program text:

```
from PyQt5.QtCore import Qt
from PyQt5.QtWidgets import (
    QApplication, QWidget,
    QHBoxLayout, QVBoxLayout,
    QGroupBox, QButtonGroup, QRadioButton,
    QPushButton, QLabel)
from random import randint, shuffle

class Question():
    '''contains a question, a correct answer and three incorrect ones'''
    def __init__(self, question, right_answer, wrong1, wrong2, wrong3):
        self.question = question
        self.right_answer = right_answer
        self.wrong1 = wrong1
        self.wrong2 = wrong2
        self.wrong3 = wrong3

questions_list = []
questions_list.append(
    Question('Official language of Brazil', 'Portuguese', 'English', 'Spanish', 'Brazilian'))
questions_list.append(
    Question('Which color does not appear on the American flag?', 'Green', 'Red', 'White', 'Blue'))
questions_list.append(
    Question('Yakut national house', 'Urasa', 'Yurta', 'Igloo', 'Khata'))

app = QApplication([])

btn_OK = QPushButton('Reply') # the reply button
lb_Question = QLabel('The most difficult question in the world!') # question text

RadioGroupBox = QGroupBox("Answer options") # on-screen group of radio buttons

rbtn_1 = QRadioButton('Option 1')
rbtn_2 = QRadioButton('Option 2')
rbtn_3 = QRadioButton('Option 3')
rbtn_4 = QRadioButton('Option 4')

RadioGroup = QButtonGroup() # this is to group radio buttons to control their behavior
RadioGroup.addButton(rbtn_1)
RadioGroup.addButton(rbtn_2)
RadioGroup.addButton(rbtn_3)
RadioGroup.addButton(rbtn_4)
```

```

layout_ans1 = QHBoxLayout()
layout_ans2 = QVBoxLayout() # vertical guides inside the horizontal one
layout_ans3 = QVBoxLayout()
layout_ans2.addWidget(rbtn_1) # two answer options in the first column
layout_ans2.addWidget(rbtn_2)
layout_ans3.addWidget(rbtn_3) # two answer options in the second column
layout_ans3.addWidget(rbtn_4)

layout_ans1.addLayout(layout_ans2)
layout_ans1.addLayout(layout_ans3) # columns placed in one line

RadioGroupBox.setLayout(layout_ans1) # the "panel" with answer options is ready


AnsGroupBox = QGroupBox("Test results")
lb_Result = QLabel('are you right or not?') # here it will be written if you are "right" or "wrong"
lb_Correct = QLabel('answer will be here!') # here will be the text of the correct answer


layout_res = QVBoxLayout()
layout_res.addWidget(lb_Result, alignment=(Qt.AlignLeft | Qt.AlignTop))
layout_res.addWidget(lb_Correct, alignment=Qt.AlignHCenter, stretch=2)
AnsGroupBox.setLayout(layout_res)

layout_line1 = QHBoxLayout() # question
layout_line2 = QHBoxLayout() # answer options or test result
layout_line3 = QHBoxLayout() # "Answer" button


layout_line1.addWidget(lb_Question, alignment=(Qt.AlignHCenter | Qt.AlignVCenter))
layout_line2.addWidget(RadioGroupBox)
layout_line2.addWidget(AnsGroupBox)
AnsGroupBox.hide() # hide the answer panel, the question panel should be visible first
layout_line3.addStretch(1)
layout_line3.addWidget(btn_OK, stretch=2) # the button should be large


layout_line3.addStretch(1)


layout_card = QVBoxLayout()


layout_card.addLayout(layout_line1, stretch=2)
layout_card.addLayout(layout_line2, stretch=8)
layout_card.addStretch(1)
layout_card.addLayout(layout_line3, stretch=1)
layout_card.addStretch(1)
layout_card.setSpacing(5) # spaces between the content elements
def show_result():
    ''' show answer panel '''
    RadioGroupBox.hide()

```



```

AnsGroupBox.show()
btn_OK.setText('Next')

```

```

def show_question():
    ''' show question panel '''
    RadioGroupBox.show()
    AnsGroupBox.hide()
    btn_OK.setText('Reply')
    RadioGroup.setExclusive(False) # removed the restrictions so as to reset the radio button choice
    rbtn_1.setChecked(False)
    rbtn_2.setChecked(False)
    rbtn_3.setChecked(False)
    rbtn_4.setChecked(False)
    RadioGroup.setExclusive(True) # returned the restrictions, now only one radio button can be selected

answers = [rbtn_1, rbtn_2, rbtn_3, rbtn_4]

def ask(q: Question):
    ''' the function writes the values of the question and answers to the corresponding widgets,
    at the same time the answer options are distributed randomly'''
    shuffle(answers) # shuffled the list of buttons, now some random button is first in the list
    answers[0].setText(q.right_answer) # fill the first element of the list with the right answer, the rest
with wrong ones
    answers[1].setText(q.wrong1)
    answers[2].setText(q.wrong2)
    answers[3].setText(q.wrong3)
    lb_Question.setText(q.question) # question
    lb_Correct.setText(q.right_answer) # reply
    show_question() # show question panel

def show_correct(res):
    ''' show the result - set the text passed to the "result" inscription and show the panel we need '''
    lb_Result.setText(res)
    show_result()

def check_answer():
    ''' if any answer option is chosen, we need to check and show the answer panel'''
    if answers[0].isChecked():
        # right answer!
        show_correct('Right!')
        window.score += 1
        print('Statistics\n-Total questions: ', window.total, '\n-Right answers: ', window.score)
        print('Rating: ', (window.score/window.total*100), '%')
    else:
        if answers[1].isChecked() or answers[2].isChecked() or answers[3].isChecked():

```

```
# wrong answer!
show_correct('Wrong answer!')
print('Rating: ', (window.score/window.total*100), '%')
```

```
def next_question():
    ''' asks a random question from the list '''
    window.total += 1
    print('Statistics\n-Total questions: ', window.total, '\n-Right answers: ', window.score)
    cur_question = randint(0, len(questions_list) - 1) # we don't need the old value,
                                                        # this means that you can use a local variable!
    # randomly picked a question within the list
    # if you enter about a hundred words, they will rarely be repeated
    q = questions_list[cur_question] # picked a question
    ask(q) # asks
```

```
def click_OK():
    ''' determines whether to show another question or check the answer to this one '''
    if btn_OK.text() == 'Answer':
        check_answer() # answer check
    else:
        next_question() # next question
```

```
window = QWidget()
window.setLayout(layout_card)
window.setWindowTitle('Memo Card')
```

```
btn_OK.clicked.connect(click_OK) # by clicking on the button, we choose what exactly happens
```

```
window.score = 0
window.total = 0
next_question()
window.resize(400, 300)
window.show()
app.exec()
```