

Methodological Recommendations.

Memory Card Application. Part 2

STORYLINE:

The ProTeam developers are still working on that big job for the “Citizen of the World” cultural center. The Center has ordered a Memory Card application to sharpen their specialists’ knowledge of world cultures and languages. The application must ask the user a multiple-choice question with several answer options, all of which are kept in the program’s memory.

SUMMARY:

The goal of this lesson is to apply the ability to handle events using PyQt to the creation of the Memory Card application.

In the first half of the lesson, students work on handling clicks to the “Answer” button and program the switch between the question and result forms. In the second half of the lesson, students program a function that displays the question and answer options in the widget.

Note: Over the course of four lessons, students will continue working on the same task: “VSC. Memory Card Application.”








LINKS AND ACCESSORIES:

- lesson [presentation](#)
- tasks: [Memory Card](#) (Visual Studio Code).

LESSON OUTCOMES

<i>After the lesson, the students will:</i>	<i>The result is achieved when the students:</i>
<ul style="list-style-type: none">• connect the relevant modules and their components• handle a click to the widget button with special handler function• handle a button click with different handler functions (analyzing the button label)• can hide and display groups of widgets depending on the signals received	<ul style="list-style-type: none">• have participated in the discussion and asked clarifying questions• have programmed their own handler functions• have correctly handled signals from various widgets• have answered the teacher’s questions at the reinforcement stage

RECOMMENDED LESSON STRUCTURE

Time	Stage	Tasks for this stage
3 min 	Storyline. Discussion: “Memory Card”	<ul style="list-style-type: none"> ❑ Review the storyline goal: to create a memorizing application for the “Citizen of the World” cultural center. ❑ Look back over completed tasks and announce the plan for the day.
10 min 	Qualification	<ul style="list-style-type: none"> ❑ Organize topic review: <ul style="list-style-type: none"> ❑ widget groups ❑ parameters for determining widget placement
10 min 	Brainstorming: “Event handling”	<ul style="list-style-type: none"> ❑ Name the events in the application that are responsible for switching between the question and result forms: <ul style="list-style-type: none"> ❑ a click of the “Answer” button ❑ a click of the “Next question” button ❑ Come to realize the necessity of programming event handling using special handler functions ❑ Describe how these functions work, paying attention to the radio button reset.
20 min	Visual Studio Code: “VSC. PyQt. Memory Card”	<ul style="list-style-type: none"> ❑ Organize completion of the task “VSC. PyQt. Memory Card” in the Visual Studio Code environment.
5 min 	Break	<ul style="list-style-type: none"> ❑ Help students restore their focus in a gamelike format.
15 min 	Brainstorming: “Displaying the question”	<ul style="list-style-type: none"> ❑ Discuss the process of displaying a specific question and answer options. ❑ Describe the functionality of the question-asking function (that displays parameters on widgets). ❑ Describe the functionality of the function for checking the selected answer after a click of the “Answer” button.
20 min 	Visual Studio Code: “VSC. PyQt. Memory Card”	<ul style="list-style-type: none"> ❑ Organize completion of the task “VSC. PyQt. Memory Card” in the Visual Studio Code environment.
10 min 	Lesson wrap-up. Reflection.	<ul style="list-style-type: none"> ❑ Complete a technical interview on the topics: event handling and properties of radio buttons. ❑ Suggest “documentation” on the platform.

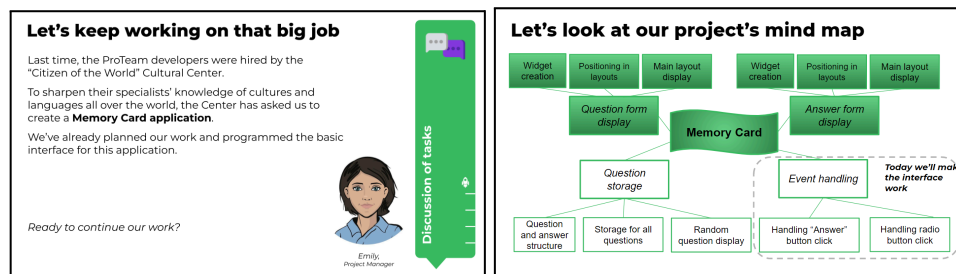
Storyline. Discussion: “Memory Card”

(3 min.)

Open the presentation. The developers will not need their computers yet.

“Hello, colleagues! Today we will continue working on a large job given to us by the “Citizen of the World” Cultural Center. Last time, we divided the project into tasks and sub-tasks, and developed an application interface with forms for the question and correct answer.”

Show the status of the tasks on the project’s mental map. Tasks the developers have already finished are colored in. Tell the students that today, they will be making elements of the interface active, i.e. doing event handling.



State the goal of today’s workday and announce its content.

Qualification

(10 min.)

This time, the qualification will be based on the materials from the previous workday.

What is event handling?
How do we handle a mouse event?

Proficiency Testing

To create a group of widgets and assign widget positions inside the group:

- Create a QGroupBox group (from QWidgets).
- Position the relevant widgets in the layouts separately.
- Set the main widget layout in the group.

If the widgets we need have already been created, then:

Code	Description
<code>RadioGroupBox = QGroupBox('Options')</code>	A constructor for creating the group.
<code>rbtn_1 = QRadioButton('Enets')</code>	Create radio button.
<code>layout_quest = QBoxLayout()</code>	Create layout line.
<code>layout_quest.addWidget(rbtn_1)</code>	Add radio button to it.
<code>RadioGroupBox.setLayout(layout_quest)</code>	Create main layout for the group.

Proficiency Testing

```
RadioGroupBox = QGroupBox("Answer options")
rbtn_1 = QRadioButton("Enets")
rbtn_2 = QRadioButton("Smurfs")
rbtn_3 = QRadioButton("Chulym")
rbtn_4 = QRadioButton("Aleuts")
layout_ans1 = QBoxLayout()
layout_ans2 = QBoxLayout()
layout_ans3 = QBoxLayout()
layout_ans4 = QBoxLayout()
layout_ans1.addWidget(rbtn_1)
layout_ans2.addWidget(rbtn_2)
layout_ans3.addWidget(rbtn_3)
layout_ans4.addWidget(rbtn_4)
layout_ans1.addWidget(layout_ans2)
layout_ans1.addWidget(layout_ans3)
layout_ans1.addWidget(layout_ans4)
RadioGroupBox.setLayout(layout_ans1)
```

Proficiency Testing

Brainstorming: “Event handling”

(10 min.)

Announce that you are about to examine tasks having to do with handling various clicks to the “Answer” button. Remind students that a click of the widget can be handled with either a pre-made handler function or a specially written one.


We will not analyze the correctness of the user’s answer yet.

1. Begin with the main goal — *switching from the question form to the answer form*. Ask the developers to list the interface-changing actions that must be included in the handler function. Then, ask them to name the commands that must be used in the `show_result()` function.

1. Displaying the answer to a question

Clicking "Answer" in the question form:

- hide question form,
- show answer form,
- Change label from "Answer" to "Next question"



How do we program this?

1. Displaying the answer to a question

```
def show_result():
    hide question form,
    show answer form,
    Change label from "Answer" to "Next question"

    # ...

    btn_OK.clicked.connect(show_result)
```

The handler function that displays the answer form.

Calling the function when the "Answer" button is clicked.

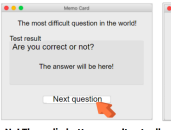
2. In the same manner, ask your students to name the necessary actions and corresponding commands for *switching from the answer form to the question form*. If the developers name the same actions, ask a leading question: "Are these actions enough?" Notice that we haven't thought about the radio buttons - switching back to the old form will not reset them.

Demonstrate the new commands necessary for resetting radio buttons. Show that they must be included, not just in the `show_question()` function, but also in the interface. The radio button group `QButtonGroup` is not visually set off like `QGroupBox`, but it allows for the manipulation of all the elements in the group.

2. Displaying the question and answer options

Answer form, click "Next question":

- hide answer form,
- show question form,
- change the label "Next question" to "Answer"



Not! The radio buttons won't actually reset!

2. Displaying the question and answer options

```
RadioGroup = QButtonGroup()
RadioGroup.addButton(btn_1)
RadioGroup.addButton(btn_2)
RadioGroup.addButton(btn_3)
RadioGroup.addButton(btn_4)

# ...

RadioGroup.setExclusive(False)
btn_1.setChecked(False)
btn_2.setChecked(False)
btn_3.setChecked(False)
btn_4.setChecked(False)
RadioGroup.setExclusive(True)
```

We are **unifying** all the radio buttons in a special group. Now only one of them can be selected at a time.

Let's remove the limits for the choice reset.

Reset the choice for all radio buttons.

Bring back the limits.

2. Displaying the question and answer options

```
def show_question():
    hide answer form,
    show question form,
    change the label "Next question" to "Answer",
    reset all radio buttons.

    # ...
```

But the button click is already being handled by `show_result()`! What do we do? How do we choose the correct handler function after a click?

After describing the function, bring up the division of responsibility between `show_question()` and `show_result()`.

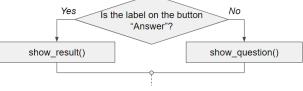
"We've written two functions, `show_question()` and `show_result()`, to handle a click of the same button. How do we determine when to call the first function and when to call the second? (Students' answers).

That's right, we can use the text on the button as our guide. How do we program that choice? Which function will end up handling the button click? (Students' answers).

Choosing the appropriate handler function can be conveniently described in a separate function, `start_test()`. It will check the button label and call the relevant handler function.

3. Function that chooses the handler function

```
def start_test():
    if the label on the button says "Answer", call function show_result().
    Otherwise, call show_question().
```



Let's bring it all together:

```
#unifying the radio buttons into a special group
def show_question():
    # Function body
def show_result():
    # Function body
def start_test():
    # Function body

    btn_OK.clicked.connect(start_test)
```

Notice that these functions could change in the future, since we're not working with a data structure yet, just making elements of the interface active.

Then, ask questions to check students' understanding of the topic and move on to the task in VSC.

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

(20 min.)

Organize completion of the task on programming the handler functions. To continue working, open the same task as last time.

You will find an extended sample solution for the first half of the lesson at the end of this methodological guide.

Break

(5 min.)

Give your workers a break from their computers. The goal of this break is to switch their attention to something else and let them stretch a little. Organize one of these [suggested physical activities](#).

Brainstorming: "Displaying the question"

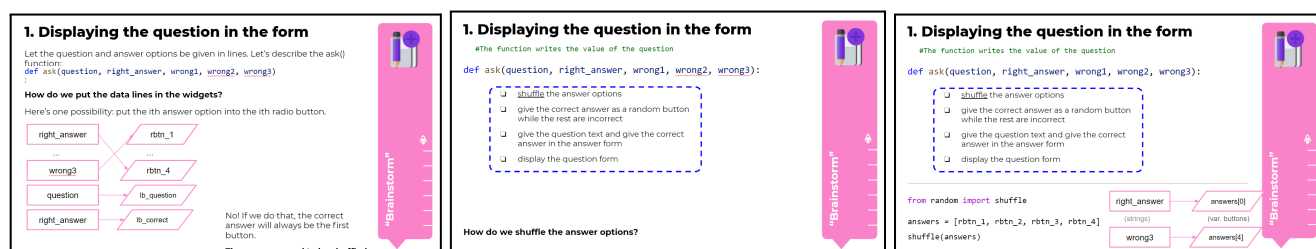
(15 min.)

Move on to the tasks related to displaying a specific question and checking the correctness of a given answer. Mention that, before this, you had simply put text on widgets while creating them. Now, you have to write a function that asks a specific question. After this step, you will be able to move on to asking questions from the set.

To achieve this goal, you must program three functions: **ask** (the question data is passed on as parameters and randomly displayed in the widgets), **check_answer** (called when the "Answer" button is clicked and determines the result; the old start_test function is deleted) and **show_correct** (called from check_answer when the application must display a correct answer).

1. Begin with the ask() function. Demonstrate its purpose: displaying the question that is passed to it. Make it clear that students cannot simply distribute the answer options to the radio buttons, because then the correct answer would always be the first one.

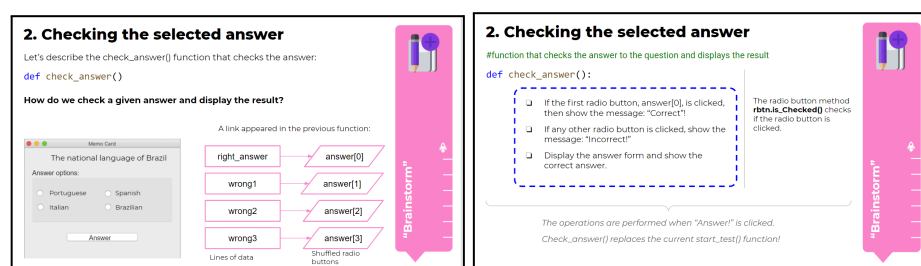
To randomly assign an answer option to each radio button, let's shuffle the radio buttons themselves. We'll assign the first of the shuffled buttons to the correct answer and the others to incorrect answers.



2. Move on to the `check_answer()` function. Draw students' attention to the fact that it will handle a click of the "Answer" button and, at the same time, check the selected answer, so the old `start_test()` must be hidden.

Since, during the previous step, we made the correct answer correspond to the first of the shuffled radio buttons, checking the correct answer means checking whether the first radio button has been clicked.

Draw the developers' attention to the fact that the displayed result forms ("Correct" or "Incorrect" and correct answer) can be programmed right here, but it's more convenient to put them in a separate function: `show_correct()`.



Summarize what's been said. Name the purpose of each function again and show how they are used when the program is launched. Describe the anticipated result of the aforementioned work and move on to the task.

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

(20 min.)

Organize completion of the task on programming the question-asking functions. To continue working, open the same task as last time.

You will find an extended sample solution for the first half of the lesson at the end of this methodological guide.

Workday wrap-up. Reflection

(10 min.)

Use the presentation to wrap up the workday. Conduct a technical interview with questions on the materials of the brainstorming stage. When working on a large project, the reflection stage is especially important.

Suggest an additional task: look through the code again and add comments. Soon the program will become even bigger, and the code will be unreadable without comments.

Answers to the tasks

Task “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)

app = QApplication([])

btn_OK = QPushButton('Answer')
lb_Question = QLabel('The most difficult question in the world!')

RadioGroupBox = QGroupBox("Answer options")

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

RadioGroup = QButtonGroup()
RadioGroup.addButton(rbtn_1)
RadioGroup.addButton(rbtn_2)
RadioGroup.addButton(rbtn_3)
RadioGroup.addButton(rbtn_4)

layout_ans1 = QHBoxLayout()
layout_ans2 = QVBoxLayout()
layout_ans3 = QVBoxLayout()
layout_ans2.addWidget(rbtn_1)
layout_ans2.addWidget(rbtn_2)
layout_ans3.addWidget(rbtn_3)
layout_ans3.addWidget(rbtn_4)

layout_ans1.addLayout(layout_ans2)
layout_ans1.addLayout(layout_ans3)

RadioGroupBox.setLayout(layout_ans1)

AnsGroupBox = QGroupBox("Test result")
lb_Result = QLabel('are you correct or not?')
lb_Correct = QLabel('the answer will be here!')
```

```

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()
layout_line2 = QHBoxLayout()
layout_line3 = QHBoxLayout()

layout_line1.addWidget(lb_Question, alignment=(Qt.AlignHCenter | Qt.AlignVCenter))
layout_line2.addWidget(RadioGroupBox)
layout_line2.addWidget(AnsGroupBox)
AnsGroupBox.hide()

layout_line3.addStretch(1)
layout_line3.addWidget(btn_OK, stretch=2) # this button should be big
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) # spacing the content

# -----
# The widgets and mockups have been created. Next, the functions:
# -----

def show_result():
    ''' show answer panel '''
    RadioGroupBox.hide()
    AnsGroupBox.show()
    btn_OK.setText('Next question')

def show_question():
    ''' show question panel '''
    RadioGroupBox.show()
    AnsGroupBox.hide()
    btn_OK.setText('Answer')
    RadioGroup.setExclusive(False) # remove limits in order to reset radio button selection
    rbtn_1.setChecked(False)

```



```

rbtn_2.setChecked(False)
rbtn_3.setChecked(False)
rbtn_4.setChecked(False)
RadioGroup.setExclusive(True) # bring back the limits so only one radio button can be selected

def test():
    ''' a temporary function that makes it possible to press a button to call up alternating
    show_result() or show_question() '''
    if 'Answer' == btn_OK.text():
        show_result()
    else:
        show_question()

window = QWidget()
window.setLayout(layout_card)
window.setWindowTitle('Memo Card')
btn_OK.clicked.connect(test) # check that the answer panel appears when the button is pressed
window.show()
app.exec()

```

1.2. Program text:

```

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

app = QApplication([])
btn_OK = QPushButton('Answer')
lb_Question = QLabel('The most difficult question in the world!')

RadioGroupBox = QGroupBox("Answer options")
rbtn_1 = QRadioButton('Option 1')
rbtn_2 = QRadioButton('Option 2')
rbtn_3 = QRadioButton('Option 3')
rbtn_4 = QRadioButton('Option 4')

RadioGroup = QButtonGroup()
RadioGroup.addButton(rbtn_1)
RadioGroup.addButton(rbtn_2)
RadioGroup.addButton(rbtn_3)
RadioGroup.addButton(rbtn_4)
layout_ans1 = QHBoxLayout()
layout_ans2 = QVBoxLayout()
layout_ans3 = QVBoxLayout()

```

```
layout_ans2.addWidget(rbtn_1)
layout_ans2.addWidget(rbtn_2)
layout_ans3.addWidget(rbtn_3)
layout_ans3.addWidget(rbtn_4)

layout_ans1.addLayout(layout_ans2)
layout_ans1.addLayout(layout_ans3)

RadioGroupBox.setLayout(layout_ans1)

AnsGroupBox = QGroupBox("Test result")
lb_Result = QLabel('are you correct or not?')
lb_Correct = QLabel('the answer will be here!')

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()
layout_line2 = QHBoxLayout()
layout_line3 = QHBoxLayout()

layout_line1.addWidget(lb_Question, alignment=(Qt.AlignHCenter | Qt.AlignVCenter))
layout_line2.addWidget(RadioGroupBox)
layout_line2.addWidget(AnsGroupBox)
AnsGroupBox.hide()

layout_line3.addStretch(1)
layout_line3.addWidget(btn_OK, stretch=2)
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)

def show_result():
    ''' show answer panel '''
    RadioGroupBox.hide()
```

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

def show_question():
    ''' show question panel '''
    RadioGroupBox.show()
    AnsGroupBox.hide()
    btn_OK.setText('Answer')
    RadioGroup.setExclusive(False)
    rbtn_1.setChecked(False)
    rbtn_2.setChecked(False)
    rbtn_3.setChecked(False)
    rbtn_4.setChecked(False)
    RadioGroup.setExclusive(True)

answers = [rbtn_1, rbtn_2, rbtn_3, rbtn_4]

def ask(question, right_answer, wrong1, wrong2, wrong3):
    ''' the function writes the value of the question and answers into the corresponding widgets while
    distributing the answer options randomly'''
    shuffle(answers)
    answers[0].setText(right_answer)
    answers[1].setText(wrong1)
    answers[2].setText(wrong2)
    answers[3].setText(wrong3)
    lb_Question.setText(question)
    lb_Correct.setText(right_answer)
    show_question()

def show_correct(res):
    ''' show result - put the written text into "result" and show the corresponding panel '''
    lb_Result.setText(res)
    show_result()

def check_answer():
    ''' if an answer option was selected, check and show answer panel '''
    if answers[0].isChecked():
        show_correct('Correct!')
    else:
        if answers[1].isChecked() or answers[2].isChecked() or answers[3].isChecked():
            show_correct('Incorrect!')

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

```
ask('The national language of Brazil', 'Portuguese', 'Brazilian', 'Spanish', 'Italian')  
btn_OK.clicked.connect(check_answer)
```

```
window.show()  
app.exec()
```