

Methodological Recommendations

Memory Card Application. Part 3

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 lesson goal is to apply knowledge of object-oriented programming, the ability to create functions, and the ability to handle events to the creation of the Memory Card application.

In the first half of the lesson, students will move from storing question data in isolated variables to storing the data in instances of a class, collected into a list. In the second half of the lesson, students will program the transition from one question in the list to another. Thus, the basic functionality of the Memory Card application should be finished by the end of this lesson.









LINKS AND ACCESSORIES:

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

EDUCATIONAL OUTCOMES FOR THIS LESSON

<i>After the lesson, the students will:</i>	<i>The result is achieved when the students:</i>
<ul style="list-style-type: none">• import the necessary modules and their components;• handle a widget button click with a special handler function;• use object-oriented programming to optimize the storage of question data;• use the list data structure to work with a set of questions ;• create new functions and reworks old functions to fit the new data storage system.	<ul style="list-style-type: none">• have participated in the discussion and asked clarifying questions;• have incorporated the Question class into the project;• have worked with a list of instances of the Question class ;• have programmed functionalities that display a question, check its answer and display the next question;• have a basic solution to the Memory Card project by the end of the lesson;• have answered the teacher's questions at the reinforcement stage.

RECOMMENDED LESSON STRUCTURE

Time	Stage	Goals for this stage
3 min 	Storyline. Discussion: “Memory Card”	<ul style="list-style-type: none"> ❑ Remind students of their story-related goal: to create an application for memorizing information for the “Citizen of the World” cultural center. ❑ Remind students of the tasks they’ve already completed and announce the plan for the day.
10 min 	Qualification Testing	<ul style="list-style-type: none"> ❑ Organize a review of the topics: <ul style="list-style-type: none"> ❑ lists and methods for working with them ❑ class creation
10 min 	Brainstorming: “Question Storage System”	<ul style="list-style-type: none"> ❑ Set a goal: to move from working with one question to working with multiple questions. ❑ Realize the necessity of using OOP and data structures. ❑ Describe filling out the Question class. ❑ Assuming that the program works with one question, discuss the transition from variables to class fields.
10 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: “A System for Working with Questions”	<ul style="list-style-type: none"> ❑ Set a goal: to move from one question to multiple questions. ❑ Realize the necessity of storing a set of questions in a list. It will be possible to iterate over the questions in order. ❑ Realize the necessity of writing a function to switch between questions, as well as a new handler function for button clicks. ❑ Describe the anticipated functionality for the program.
30 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 	Lesson Wrap-Up. Reflection	<ul style="list-style-type: none"> ❑ Complete a technical interview on the topics of the brainstorming stage. ❑ Suggest adding some “documentation” on the platform.

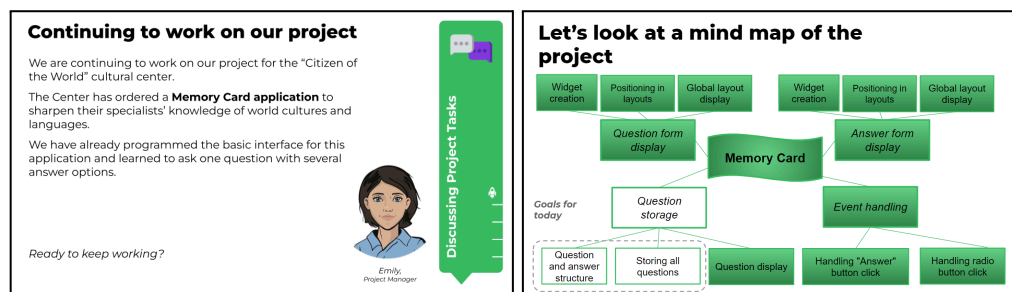
Storyline. Discussion: “Memory Card”

(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 programmed the display of one question and the checking of its answer. Today, we will realize two important goals from the “Question Storage” block: we will code a convenient structure for one question and program the way our application switches between questions.”

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 if they work well in today’s lesson, they’ll end the day with a basic version of Memory Card.

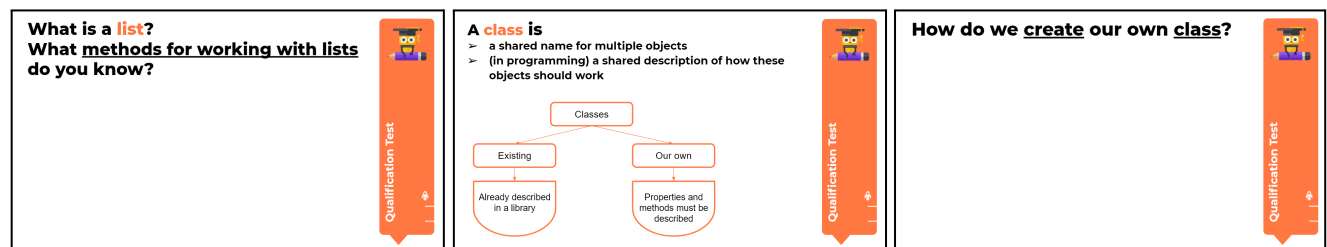


State the goal of today’s work day and announce its content.

Qualification Testing

(10 min.)

This time, the qualification test will cover the topics “Lists” and “Class creation.” These topics must be reviewed before we can effectively meet the goals of the day.



Brainstorming: “Question Storage System”

(10 min.)

Show on the mind map that, in the first half of the work day, you will be developing a system for storing questions. To move from one question to multiple questions, it’s vital to choose a convenient structure for storing data.

Discuss the nature of data structures. Various technical solutions are possible here but, for the current project, students are asked to program a Question class whose fields will be the question itself, the correct answer, and three incorrect answers.

1. Begin by enumerating the actions the program performs on a question. Many of these have already been programmed, so the developers should confidently name: adding questions to the program, displaying a question with answer options, checking the user's answer, and displaying the correct answer.

Next, list the data necessary to perform these actions. Pose the question: which data structure is optimal for the integrated and convenient storage and use of this data?


A question and the information about it

Question-related actions:

- ☐ storing a question
- ☐ displaying a question in an application window
- ☐ reading and checking the user's answer
- ☐ displaying the correct answer

Data we need about the question:

- ☐ the text of the question
- ☐ the correct answer option
- ☐ three incorrect answer options



A question and the information about it

Data we need about the question:

- ☐ the text of the question
- ☐ the correct answer option
- ☐ three incorrect answer options

To ask multiple questions, we need a structure that can store a lot of data. **What structure should we use?**

2. Suggest creating a Question class. Discuss:

- What properties should every object of type Question have?
- When we create an instance of the class, how do we give it these properties? What method do we have to create?
- Give examples of creating an instance of the Question class using the given constructor.

3. List the steps necessary to incorporate the Question class into the program.

- A new question should be added, not by introducing new line variables, but by creating an instance of the class.
- Therefore, the widgets will display data from class fields, not from variables.
- The ask() function, which asks the questions, should also work with the properties of an object of type Question, not with variables.


Implementing the Question class

To implement Question, we must make some changes to our program.

What	Before	After
Adding a new question	Data were stored in variables	Data are the properties of an instance of the class
Displaying a question	Widgets displayed data from variables	Widgets display fields of the class instance
Asking a question (calling ask())	The entry parameters were the constants with the question data	The entry parameter is an instance of the Question class

Expected result:


- ☐ "Inside" the application, the data storage system will change. Instead of storing question data in variables, we'll have the Question class and its instances.
- ☐ The application will not change externally and its functionality will not increase, but it will work differently.



Let's put it all together:

```
class Question():
    # Class description
    # Application interface
    # Functions that display the question

def ask(q: Question):
    # Changed function body with the properties of instance q
    # Creating a window, launching the application
    # Creating instance q of Question
    # Calling ask with argument q
```



Summarize everything you've said and move on to introducing the Question class in VSC. Note that, externally, the program should not change.

Visual Studio Code: “VSC. PyQt. Memory Card”

(15 min.)

Organize the students' completion of the task that introduces the Question class. To continue working, open the same task as last time.

You'll find a sample solution for the first half of the lesson at the end of this methodological plan.

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 the [suggested physical activities](#).

"Brainstorm": “A System for Working with Questions”

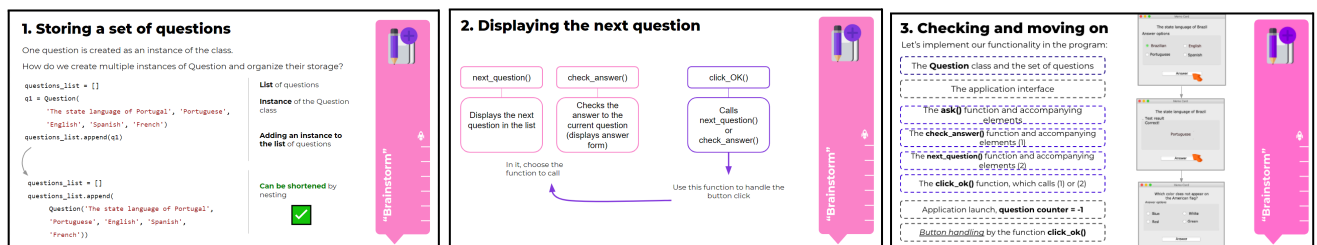
(15 min.)

Show the latest goal on the mind map: programming the ability to work with several objects of the Question class. On a basic level, this goal requires the ability to iterate over the questions in order. The display of a random question from the list will be discussed during the next workday.

1. Begin by listing the tasks required to move from one question to multiple questions. It is necessary to:

- choose a structure for storing the set of questions.
- describe the mechanics of question display (including during the transition to the next question). The question display process must be encapsulated in a function.
- describe the process of choosing the next action: either displaying a new question or checking the answer to the current question (in previous versions of the project, we had the test() handler function).

2. Move on to a discussion of the tasks.



- Discuss which data structure is the most convenient for storing question objects and iterating over them in order. Come to the necessity of using a list.
- Before discussing the function for displaying the next question, imagine it has already been written and think about how it will fit into the project as a whole. Memory Card has two key functions: next_question(), which asks a question, and

check_answer(), which checks the answer. It's convenient to call the appropriate function (based on the button label) through the intermediary function click_ok().

- Once the position of next_question() in the program has been clarified, move on to describing its functionality. Draw students' attention to the necessity of implementing a question counter. On behalf of developer Cole, suggest making it a property of the application window. This is logical since the question is displayed in the window's widgets.

Summarize what's been said and demonstrate how the new functionality will supplement the code that has already been written. Describe the anticipated solution and move on to programming.

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

(30 min.)

Have the students complete the task on programming the data structures and new functions. To continue working, open the same task as last time.

You'll find a sample solution for the second half of the lesson at the end of this methodological plan.

Wrapping up the work day. Reflection

(5 min.)

Use the presentation to wrap up the work day. Conduct a technical interview with questions about the materials of the brainstorming stage. The reflection stage is especially important when working on a large project.

Suggest an additional task: to look through the code one more time and add 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)
from random import shuffle

class Question():
    ''' contains the question, one correct answer and three incorrect answers'''
    def __init__(self, question, right_answer, wrong1, wrong2, wrong3):
        # all the lines must be given when creating the object, and will be recorded as properties
        self.question = question
        self.right_answer = right_answer
        self.wrong1 = wrong1
        self.wrong2 = wrong2
        self.wrong3 = wrong3

app = QApplication([])

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

RadioGroupBox = QGroupBox("Answer options") # on-screen group for radio buttons with answers

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

RadioGroup = QButtonGroup() # this groups the radio buttons so we can 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() # the vertical ones will be inside the horizontal one
layout_ans3 = QVBoxLayout()
layout_ans2.addWidget(rbtn_1) # two answers in the first column
layout_ans2.addWidget(rbtn_2)
layout_ans3.addWidget(rbtn_3) # two answers in the second column
```

```

layout_ans3.addWidget(rbtn_4)

layout_ans1.addLayout(layout_ans2)
layout_ans1.addLayout(layout_ans3) # put the columns in the same line

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

AnsGroupBox = QGroupBox("Test result")
lb_Result = QLabel('Were you correct or not?') # the word "correct" or "incorrect" will be written here
lb_Correct = QLabel('The answer will be here!') # the text of the correct 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() # 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 because the question panel must be visible first

layout_line3.addStretch(1)
layout_line3.addWidget(btn_OK, stretch=2) # the button must 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 contents

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

```



```
def show_question():
    ''' Show the answer panel. '''
    RadioGroupBox.show()
    AnsGroupBox.hide()
    btn_OK.setText('Answer')
    RadioGroup.setExclusive(False) # remove the limits so we can reset the radio buttons
    rbtn_1.setChecked(False)
    rbtn_2.setChecked(False)
    rbtn_3.setChecked(False)
    rbtn_4.setChecked(False)
    RadioGroup.setExclusive(True) # reset the limits so that only one radio button can be selected at a time

answers = [rbtn_1, rbtn_2, rbtn_3, rbtn_4]

def ask(q: Question):
    ''' This function writes the value of the question and answers in the corresponding widgets. The answer
    options are distributed randomly. '''
    shuffle(answers) # shuffle the list of buttons; now a random button is first in the list
    answers[0].setText(q.right_answer) # fill the first element of the list with the correct answer and the
    other elements with incorrect answers
    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) # answer
    show_question() # show question panel

def show_correct(res):
    ''' Show the result - put the text that was passed to this function into the "result" label and show the
    relevant panel. '''
    lb_Result.setText(res)
    show_result()

def check_answer():
    ''' If one of the answer options is selected, check it and show the answer panel. '''
    if answers[0].isChecked():
        # a correct answer!
        show_correct('Correct!')
    else:
        if answers[1].isChecked() or answers[2].isChecked() or answers[3].isChecked():
            # an incorrect answer!
            show_correct('Incorrect!')
```

```

window = QWidget()
window.setLayout(layout_card)
window.setWindowTitle('Memo Card')
q = Question('Select the most appropriate English name for the programming concept to store some data',
'variable', 'variation', 'variant', 'changing')
ask(q)
btn_OK.clicked.connect(check_answer) # remove the test, we need to check the answer here
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

class Question():
    def __init__(self, question, right_answer, wrong1, wrong2, wrong3):
        # all the lines must be given when creating the object, and will be recorded as properties
        self.question = question
        self.right_answer = right_answer
        self.wrong1 = wrong1
        self.wrong2 = wrong2
        self.wrong3 = wrong3

questions_list = []
questions_list.append(Question('The state 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('A traditional residence of the Yakut people', 'Urasa', 'Yurt', 'Igloo',
'Hut'))

app = QApplication([])

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

RadioGroupBox = QGroupBox("Answer options") # on-screen group for radio buttons with answers

rbtn_1 = QRadioButton('Option 1')

```

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

RadioGroup = QButtonGroup() # this groups the radio buttons so we can 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() # the vertical ones will be inside the horizontal one
layout_ans3 = QVBoxLayout()
layout_ans2.addWidget(rbtn_1) # two answers in the first column
layout_ans2.addWidget(rbtn_2)
layout_ans3.addWidget(rbtn_3) # two answers in the second column
layout_ans3.addWidget(rbtn_4)

layout_ans1.addLayout(layout_ans2)
layout_ans1.addLayout(layout_ans3) # put the columns in the same line

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

AnsGroupBox = QGroupBox("Test result")
lb_Result = QLabel('Are you correct or not?') # "correct" or "incorrect" will be written here
lb_Correct = QLabel('The answer will be here!') # the correct answer text will be written 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() # 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 because the question panel should be visible first

layout_line3.addStretch(1)
layout_line3.addWidget(btn_OK, stretch=2) # the button must 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 contents

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

def show_question():
    ''' Show the question panel. '''
    RadioGroupBox.show()
    AnsGroupBox.hide()
    btn_OK.setText('Answer')
    # clear selected radio button
    RadioGroup.setExclusive(False) # remove the limits so we can reset the radio buttons
    rbtn_1.setChecked(False)
    rbtn_2.setChecked(False)
    rbtn_3.setChecked(False)
    rbtn_4.setChecked(False)
    RadioGroup.setExclusive(True) # reset the limits so that only one radio button can be selected at a time

answers = [rbtn_1, rbtn_2, rbtn_3, rbtn_4]

def ask(q: Question):
    ''' This function writes the value of the question and answers in the corresponding widgets. The answer
    options are distributed randomly. '''
    shuffle(answers) # shuffle the list of buttons; now a random button is first in the list
    answers[0].setText(q.right_answer) # fill the first element of the list with the correct answer and the
    other elements with incorrect answers
    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) # answer
    show_question() # show the question panel
    
```

```
def show_correct(res):
    ''' Show the result - put the text that was passed to this function into the "result" label and show the
    relevant panel. '''
    lb_Result.setText(res)
    show_result()

def check_answer():
    ''' If one of the answer options is selected, check it and show the answer panel. '''
    if answers[0].isChecked():
        # a correct answer!
        show_correct('Correct!')
    else:
        if answers[1].isChecked() or answers[2].isChecked() or answers[3].isChecked():
            # an incorrect answer!
            show_correct('Incorrect!')

def next_question():
    ''' Asks the next question in the list. '''
    # this function needs a variable that gives the number of the current question
    # this variable can be made global, or it can be the property of a "global object" (app or window)
    # we will create the property window.cur_question (below)
    window.cur_question = window.cur_question + 1 # move on to the next question
    if window.cur_question >= len(questions_list):
        window.cur_question = 0 # if the list of questions has ended, start over
    q = questions_list[window.cur_question] # take a question
    ask(q) # ask it

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

window = QWidget()
window.setLayout(layout_card)
window.setWindowTitle('Memo Card')
# Make the current question from the list a property of the "window" object. That way, we can easily change
its functions:
window.cur_question = -1 # ideally, variables like this one should be properties
                        # we'd have to write a class whose instances have these properties,
                        # but Python allows us to create a property for a single instance

btn_OK.clicked.connect(click_OK) # when a button is clicked, we choose what exactly happens
```

```
# Everything is set up. Now we ask the question and show the window:  
next_question()  
window.resize(400, 300)  
window.show()  
app.exec()
```