

Methodological guidelines.

Interface design

STORYLINE:

The creators of the Crazy People YouTube channel and their subscribers really liked the program from ProTeam. Now they want to run another competition with prizes. The finalists of the competition will have to answer questions about the Crazy People channel, so the YouTubers made a new request for an application with questions that displays messages about the prizes users win.

To complete the request, the developers will need to study widget positioning using layouts from the PyQt5 library, and they will need to study inheritance in order to be able to work with new widgets and layouts in a meaningful way.

SUMMARY:

Lesson goal: to study the process of designing an application interface using layouts and apply it to solve the storyline tasks.

The students will learn that the PyQt library has a hierarchical multi-module structure. To work meaningfully with its elements, the lesson begins with an introduction to inheritance and programming parent and inheritor classes. Then the students move on to implementing the storyline task using PyQt.

LINKS AND DETAILS:

- [presentation](#) for the lesson;
- tasks: [inheritance](#), [PyQt](#) (Visual Studio Code).









EDUCATIONAL OUTCOME OF THE LESSON

After the lesson, the students:

The result is achieved when the students:

- | | |
|---|---|
| <ul style="list-style-type: none">• can explain in their own words what inheritance, superclasses, and inheritor classes are;• create their own superclass and inheritor class;• apply the appropriate superclass and inheritor methods;• position application widgets using layouts;• know and use widgets: QRadioButton, QMessageBox, etc.;• process button clicking using PyQt. | <ul style="list-style-type: none">• have participated in the discussion and asked clarifying questions;• confidently named appropriate widgets;• created their own superclasses and inheritors;• created an application in PyQt in the Visual Studio Code environment;• have answered the teacher's questions at the consolidation stage. |
|---|---|
-

RECOMMENDED LESSON STRUCTURE

Time	Stage	Stage aims
5 min 	Storyline. Discussion: Competition	<ul style="list-style-type: none"> ❑ Set the storyline task: complete the request to create an application that asks a question and displays a message about the prize the user wins. ❑ Formulate the task: the students have to study new widgets and complex layouts.
10 min 	Qualifications	<ul style="list-style-type: none"> ❑ Arrange a review session using the presentation of the following topics: <ul style="list-style-type: none"> ❑ objects and their properties and methods; ❑ creating classes and class constructors; ❑ the PyQt library, basic widgets; ❑ positioning widgets using layouts.
15 min 	Brainstorming: Inheritance	<ul style="list-style-type: none"> ❑ Come to the understanding that we need to learn PyQt class hierarchy and study inheritance. ❑ Explain the students what inheritance is and tell them about its benefits. ❑ Demonstrate how to create a superclass and an inheritor class.
15 min 	Visual Studio Code: Inheritance	<ul style="list-style-type: none"> ❑ Have the class complete the 'Inheritance' task
5 min 	Break	<ul style="list-style-type: none"> ❑ Help restore focus.
15 min 	Brainstorming: Design interface	<ul style="list-style-type: none"> ❑ Remind them about the storyline task: to program an application with a test. ❑ Name the necessary widgets and assign them the necessary modules (and classes). ❑ Demonstrate placing objects along lines (the QVBoxLayout and QHBoxLayout classes). ❑ Demonstrate how to snap lines together.
20 min 	Platform: Crazy People: competition	<ul style="list-style-type: none"> ❑ Have the class complete the 'Crazy People: competition' task.
5 min 	Concluding the lesson. Reflection.	<ul style="list-style-type: none"> ❑ Review the concepts of superclass and inheritor class. Name the new QRadioButton and QMessageBox widgets and describe how they work. ❑ Discuss the performance of each developer during the work day: which tasks were solved successfully and which were not? Why? ❑ Offer additional tasks and supporting documentation on the platform.

Storyline. Discussion: Competition

(5 min.)

Open the presentation. The developers don't need their computers yet.


"Hello, colleagues! Glad to see you with ProTeam as part of the development team. The YouTubers from the Crazy People channel liked your prize drawing app and they have made another request."

Explain that they need to write a competition app that asks a multiple choice question about the channel's history. Depending on the answer, a notification window will need to appear showing the prize the user wins.

List the tools needed to complete the request. To do this, we will need to learn two new widgets (radio buttons and notification windows) and learn how to position widgets using multiple layouts (not just in a row or in a column).

Let's look at the task

- Product type:** windowed application.
- Functionality:**
 - display a test question with answer options;
 - display a prize depending on the answer selected.



What tools do we need to complete such a request?

Let's look at the task

The "Competition" application

```

graph TD
    A[The "Competition" application] --> B[Application functionality]
    A --> C[Application interface]
  
```

- Nothing complicated.
- Creating a question window with a **choice of possible answers**.
- Arranging the widgets in the question window.
- Processing mouse events with interface elements.
- Displaying a result window** depending on the answer.

We need to learn new widgets and better understand layouts.

Formulate the goal of the work day and what it entails.

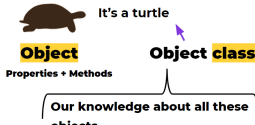
Qualifications

(10 min.)

Today, our qualifications will be on classes and frameworks for developing windowed applications.

What is an **object?**
Property? Method?
What is a **class?**
Do you know ready-made object classes?

A **property**
is a variable inside an object.
A **method**
is a function inside an object.



Button click processing

Possible solution. Write two processor functions.

- the correct radio button responds to the processor function that creates a window with a message about winning;
- the other radio buttons respond to the function that creates the window with the correct answer.

```

def show_win():
    # ...
  
```

btn_answer1.clicked.connect(show_win)

THE FUNCTION IS DESCRIBED SEPARATELY

NAME OF THE PROCESSOR FUNCTION WHICH THE BUTTON RESPONDS TO

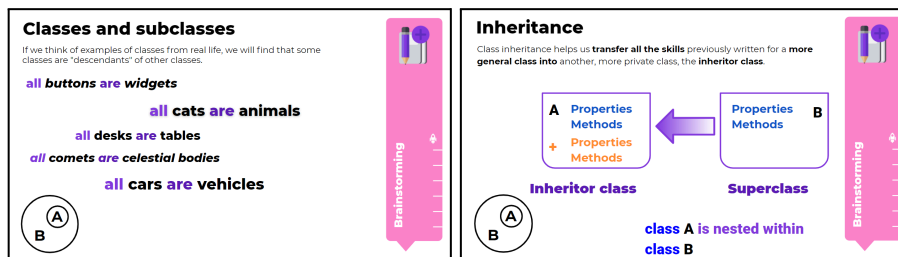
Brainstorming: Inheritance

(15 min.)

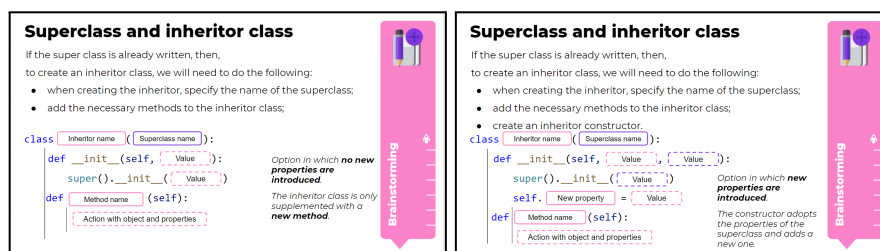
Explain the students why they need to study inheritance within the framework of the object-oriented approach. On behalf of the developer Cole, tell them that the PyQt library has a hierarchical structure, and many elements are "ancestors" and "descendants" of each other. In

order to use PyQt tools in a deliberate way and understand why different classes of widgets can have the same property fields, you need to study inheritance.

Start your conversation about inheritance with a few real-life examples. Work with the developers to come up with examples of object classes that include properties and actions from one class within another. In one of the examples, note that we rarely name the "general" type of the object if we know how to specify it (we say "cat" instead of "whiskered fluffy pet"). Note that since professional programmers are constantly working with objects of different types, they came up with the idea to transfer all the skills of one class to another.



Introduce the terms superclass and inheritor class. Demonstrate how to create an inheritor class: first, a simplified one, but with an additional method, then with new properties and methods. Draw the developers' attention to the super method, which allows you to access the fields of the superclass.



Discuss the Application class, which was already covered in the last lesson.

- Ask the developers to consider Application as a superclass and create a MobileApplication inheritor class. What properties and methods can it be supplemented with?
- Let's say MobileApplication is expanded with the system_type property (the type of phone operating system) and the setup_application method (displays "Installation of <system type> application completed"). What should be in the constructor of the MobileApplication class? How do we assign values to properties inherited from the Application superclass?
- What will the program display if the programmer decides to output information about an instance of the MobileApplication class using a method that displays the fields of the Application class? Why wasn't the system_type field of the inheritor class displayed?

Note that these are tutorial examples, but they will help them understand the possible relationships between these classes. Then ask comprehension questions and move on to the VSC task.

VSC: Inheritance

(15 min.)

Have the students complete the tasks on creating an inheritor class. Their completed tasks must be checked against the screenshot in the conditions and shown to the mentor.

You will find the keys to the tasks at the end of the methodological guidelines.

Break

(5 min.)

Get the ыeraway from their computers. The purpose of the break is to shift their attention and do something active. Have them do one of these [suggested physical activities](#).

Brainstorming: Interface Design

(15 min.)

Remind them about the task for the workday and list the development tools for windowed applications that they need to learn. Then walk them through the steps on how to create a windowed application for the Crazy People competition.

The "Competition" application

Step 1. Creating the necessary widgets.

Object	Designation
Application	QApplication
Application window	QWidget
Label	QLabel
Guide line (vertical, horizontal)	QVBoxLayout, QHBoxLayout
Message window	QMessageBox()
Radio button	QRadioButton

How do we import the necessary library modules?

Importing PyQt modules

QtCore

The **Qt** module: constants and flags which determine how the window looks

It needs to be imported for the alignment parameter.

QtWidgets

The **QApplication, QWidget, QLabel, and QVBoxLayout** modules: widgets and how they are arranged

It needs to be imported for the corresponding widgets.

```


from PyQt5.QtCore import Qt
from PyQt5.QtWidgets import (
    QApplication, QWidget, QLabel, QVBoxLayout, QHBoxLayout, QMessageBox
)

```

1. Start by **listing all the required widgets** and importing libraries. Remind the developers about the important sections in PyQt: QtCore and QtWidgets. Then remind them how to create an empty window, make it visible, and disable automatic shutdown of the application.
2. Demonstrate the **new QRadioButton widget**, remind them about the process of creating a layout with one guide line, and demonstrate an interface layout with a question and answer options in a column. How do we change the layout of the widgets so that the application corresponds to what the customer wants?
3. We can position widgets using **layouts** in different ways. Suggest one option—snapping widgets to horizontal guide lines and adding the horizontal guide lines to the main vertical one.
4. Demonstrate another new widget, **QMessageBox**, for displaying notification windows. Note that this window also needs to be stopped from closing automatically.
5. Discuss application **event processing**. You can implement the display of different messages, depending on the selected radio button, in different ways. Suggest the option with two processor functions. The radio button with the correct answer should be

processed by the `show_win()` function, and the others should be processed by the `show_lose()` function.

Positioning widgets using layouts



To position the widgets the way the customer wants, you need to do the following:


- ❑ Create three horizontal guide lines.
- ❑ Add the necessary widgets to each line. We will get three layouts with widgets.
- ❑ Add these layouts to the main vertical line.
- ❑ Bind the main layout (vertical line layout) to the application window.

Creating a notification window

A window with a notification label and a ready-made "OK" button:

Method	Purpose
<code>victory_win = QMessageBox()</code>	A constructor that creates a notification window.
<code>victory_win.setText('Correct!')</code>	A method that displays the specified text in the window.
<code>victory_win.exec_()</code>	Leaves the window open.

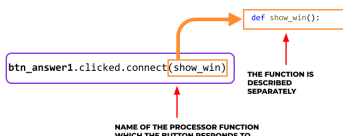
The button is created automatically. Clicking on "OK" closes the window.



Button click handling

Possible solution. Write two processor functions.

- ❑ the correct radio button responds to the processor function that creates a window with a message about winning.
- ❑ the other radio buttons respond to the function that creates the window with the correct answer.



NAME OF THE PROCESSOR FUNCTION WHICH THE BUTTON RESPONDS TO

THE FUNCTION IS DESCRIBED SEPARATELY

Sum up the discussion, answer any questions, and proceed to the task.

VSC: Crazy People: competition

(20 min.)

Have the students complete the task on creating an application that asks a multiple choice question and displays a response to the answer.

You will find the keys to the tasks at the end of the methodological guidelines.

Wrapping up the work day. Reflection

(5 min.)

Sum up the work day using the presentation. Ask the students the technical questions about the brainstorming material.

Tell them about additional activities to improve their performance and the theory documentation.

Task keys

Task VSC. Inheritance

1.1. Program text:

```
class Widget():
    #properties (fields)
    def __init__(self, title_text, x_num, y_num):
        self.title = title_text
        self.x = x_num
        self.y = y_num

    #methods
    def print_info(self):
        print('Label:', self.title)
        print('Location:', self.x, self.y)

class Button(Widget):
    def __init__(self, title_text, x_num, y_num, is_clicked_bool):
        super().__init__(title_text, x_num, y_num)
        self.is_clicked = is_clicked_bool
    def click(self):
        self.is_clicked = True
        print('You are signed up')

lotery_button = Button('Participate', 100, 100, False)
lotery_button.print_info()
answer = input('Want to sign up? (yes/no): ')
if answer == 'yes':
    lotery_button.click()
else:
    print('That's a shame!')
```

Task VSC. Crazy People: competition

1.1. Program text:

```
from PyQt5.QtCore import Qt
from PyQt5.QtWidgets import QApplication, QWidget, QPushButton, QHBoxLayout, QVBoxLayout, QLabel,
QMessageBox, QRadioButton

def show_win():
    victory_win = QMessageBox()
    victory_win.setText('Correct!\nYou win a gyro scooter')
    victory_win.exec_()
```

```
def show_lose():
    victory_win = QMessageBox()
    victory_win.setText('No, it was in 2015\nYou win a company poster')
    victory_win.exec_()

app = QApplication([])
main_win = QWidget()
main_win.setWindowTitle('Competition from Crazy People')
main_win.resize(400, 200)

question = QLabel('What year did the channel receive the "gold play button" from YouTube?')
btn_answer1 = QRadioButton('2005')
btn_answer2 = QRadioButton('2010')
btn_answer3 = QRadioButton('2015')
btn_answer4 = QRadioButton('2020')

layout_main = QVBoxLayout()
layoutH1 = QHBoxLayout()
layoutH2 = QHBoxLayout()
layoutH3 = QHBoxLayout()
layoutH1.addWidget(question, alignment = Qt.AlignCenter)
layoutH2.addWidget(btn_answer1, alignment = Qt.AlignCenter)
layoutH2.addWidget(btn_answer2, alignment = Qt.AlignCenter)
layoutH3.addWidget(btn_answer3, alignment = Qt.AlignCenter)
layoutH3.addWidget(btn_answer4, alignment = Qt.AlignCenter)

layout_main.addLayout(layoutH1)
layout_main.addLayout(layoutH2)
layout_main.addLayout(layoutH3)
main_win.setLayout(layout_main)

btn_answer3.clicked.connect(show_win)
btn_answer1.clicked.connect(show_lose)
btn_answer2.clicked.connect(show_lose)
btn_answer4.clicked.connect(show_lose)

main_win.show()
app.exec_()
```