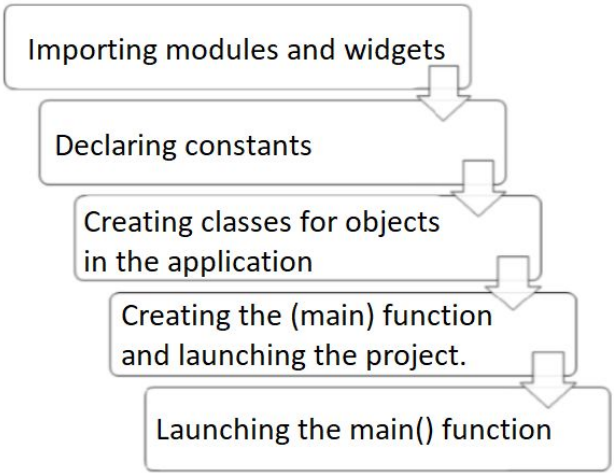
Theory

Applying OOP to the development of a desktop application

A top-level project development plan might look like this:



1. **Importing modules and widgets**. The code for importing the necessary libraries, modules, widgets, etc. is always written at the very top of the program.

```
#importing modules and widgets
from PyQt5.QtCore import Qt
from PyQt5.QtWidgets import QApplication, QWidget, QHBoxLayout,
QPushButton, QLabel, QLineEdit
```

2. **Declaring constants.** Constants contain the data that will be used by the program. They do not change. For example, the size of the future window, the text on the buttons, etc. It's easiest to create this data at the beginning of the program, all in one place, and then use it as variables. That way, it will always be easy to find and correct this data. For example:

```
#importing modules and widgets
```

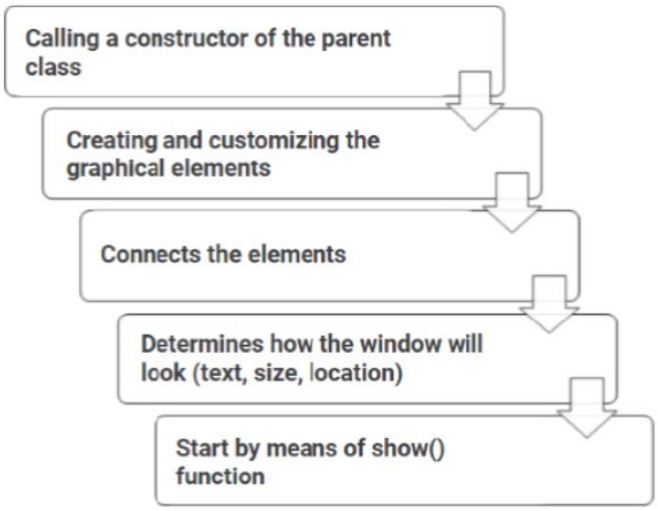
```
from PyQt5.QtCore import Qt
from PyQt5.QtWidgets import QApplication, QWidget, QHBoxLayout,
QPushButton, QLabel, QLineEdit

#declaring constants
win_width, win_height = 800, 300
win_x, win_y = 200, 200
txt_title = "Sending text"
txt_send = "Send"
txt_line = "Entry field"
```

3. **Creating classes for objects in the application.** The objects depend on the goal we are trying to accomplish. We might create the class “Person” if we needed to store data about people, or “Menu”, but there are also classes that will be realized in any windowed application - window classes. Because a window contains many widgets and must display data correctly, it is convenient to create a separate class for each window. Besides, the PyQT5 library makes it possible to inherit this class from the built-in QWidget, which means the program will automatically identify the new class as a widget.

```
#importing modules and widgets
...
#declaring constants
...
class MainWindow(QWidget):
...
```

When creating a window class, it’s important to plan out the structure of the class itself. Here’s how it’s done: the algorithm for creating a window is divided into several steps, each of which is described in a separate method of the class. These methods are called sequentially in the class constructor.

Sample algorithm:



It's important to realize that, if you are inheriting your window class from QWidget and creating a constructor in it, you are reassigning a constructor of the QWidget class and, when an instance of our class is created, it will no longer create a widget. To prevent this, we must call a constructor with specific parameters and, inside it, manually call a constructor of the parent class.

Sample program:

```
#importing modules and widgets
…
#declaring constants
…
class MainWindow(QWidget):
def __init__(self, parent=None, flags=Qt.WindowFlags()):
        #calling a constructor of the parent class
        super().__init__(parent=parent, flags=flags)

        # creating and customizing the graphical elements:
        self.initUI()

        #connects the elements
        self.connects()

      #determines how the window will look (text, size, location)
```

```
        self.set_appear()

        # start:
        self.show()

    def initUI(self):
        ''' creates graphical elements '''
        self.btn_send = QPushButton(txt_send, self)
        self.line = QLineEdit(txt_line)
        self.lable_finish = QLabel()

        self.layout_line = QHBoxLayout()
        self.layout_line.addWidget(self.line, alignment = Qt.AlignLeft)
        self.layout_line.addWidget(self.btn_send, alignment =
Qt.AlignLeft)
        self.layout_line.addWidget(self.lable_finish, alignment =
Qt.AlignCenter)
        self.setLayout(self.layout_line)

    def next_click(self):
        self.lable_finish.setText(self.line.text())

    def connects(self):
        self.btn_send.clicked.connect(self.next_click)

''' determines how the window will look (text, size, location) '''
    def set_appear(self):
        self.setWindowTitle(txt_title)
        self.resize(win_width, win_height)
        self.move(win_x, win_y)
```

4. **Creating the main function and launching the project.** Once the main window class is realized, all that's left is to create an application object and an object for the window class we just developed, so that the program can begin working. This is usually done with a main() function, i.e. the function's main (launch) function.

```
#importing modules and widgets
...
```

```
#declaring constants
...
#creating classes for objects in the application
class MainWindow(QWidget):
def __init__(self, parent=None, flags=Qt.WindowFlags()):
        #calling a constructor of the parent class
        super().__init__(parent=parent, flags=flags)

      # creating and customizing the graphical elements:
        self.initUI()

        #connects the elements
        self.connects()

        #determines how the window will look (text, size, location)
        self.set_appear()

        # start:
        self.show()
    ...
#creating a main function for the project
def main():
    app = QApplication([])
    mw = MainWindow()
    app.exec_()
#launching the project
main()
```