AGH UNIVERSITY OF SCIENCE AND TECHNOLOGY

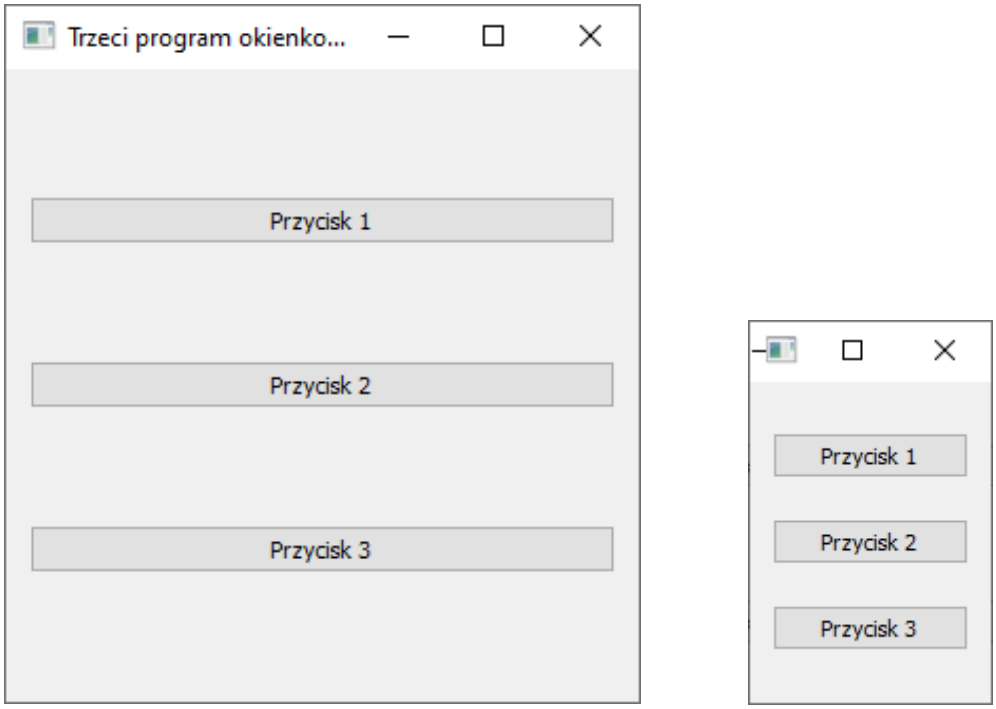
Python GUI 3
Layout'y i wykresy wbudowane

Dotychczas układ przycisków i innych elementów w GUI tworzyliśmy podając dla każdego elementu pozycję względem lewego górnego rogu okna. Rozwiązanie takie jednak jest bardzo czasochłonne, drobna zmiana wymaga potem sporych modyfikacji położenia kolejnych elementów, a także stworzony w ten sposób układ nie dostosowuje się do rozmiaru okna. Na szczęście biblioteka PyQt5 zawiera elementy takie jak **QVBoxLayout, QHBoxLayout, QGridLayout**, które pozwalają na tworzenie wzorców rozmieszczenia elementów, które dodatkowo automatycznie skalują się wraz z rozmiarem okna.

1. QVBoxLayout – pionowy layout

Pierwszym przydatnym układem jest układ pionowy, w którym kolejne dodawane elementy będą znajdować się jeden pod drugim. Poniżej przedstawiono przykładowy program korzystający z tego układu. Warto zauważyć, że kolejność elementów w tym układzie zależna jest od kolejności, w jakiej elementy dodawane są do tego układu. Nie można także zapomnieć o linijce ustawiającej układ dla całego okna.

```
import sys
from PyQt5.QtWidgets import *

class App(QWidget):
    def __init__(self):
        super().__init__()
        self.title = 'Trzeci program okienkowy'
        self.left = 100
        self.top = 100
        self.width = 300
        self.height = 300

        self.setWindowTitle(self.title)
        self.setGeometry(self.left, self.top, self.width, self.height)

        layout1 = QVBoxLayout()
        button1 = QPushButton('Przycisk 1', self)
        button2 = QPushButton('Przycisk 2', self)
        button3 = QPushButton('Przycisk 3', self)

        layout1.addWidget(button1)
        layout1.addWidget(button2)
        layout1.addWidget(button3)

        self.setLayout(layout1)
        self.show()


app = QApplication(sys.argv)
ex = App()
app.exec_()
```

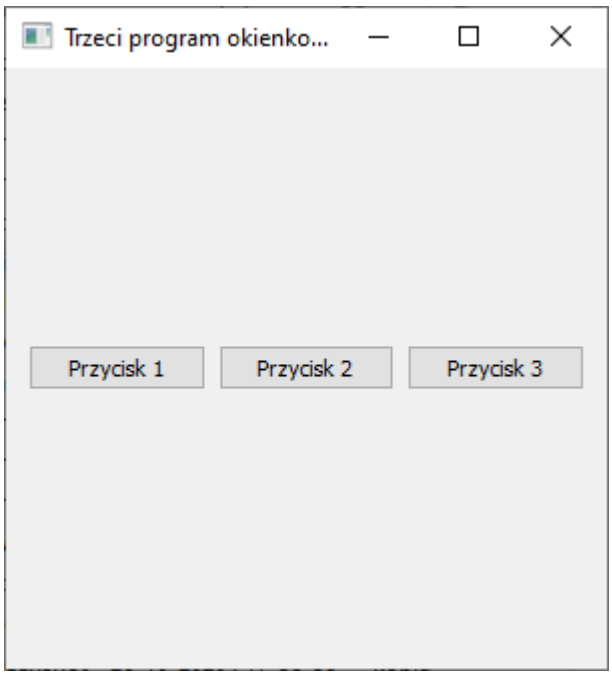


Wartą uwagi cechą tego układu jest to, jak zachowuje się on w momencie zmiany rozmiaru okna, obrazują to dwa powyższe zrzuty ekranu.

2. QHBoxLayout – poziomy layout.

Kolejnym przydatnym układem jest układ poziomy – aby zobaczyć jak działa wystarczy zmienić tylko w naszym programie demonstracyjnym jedną linijkę:

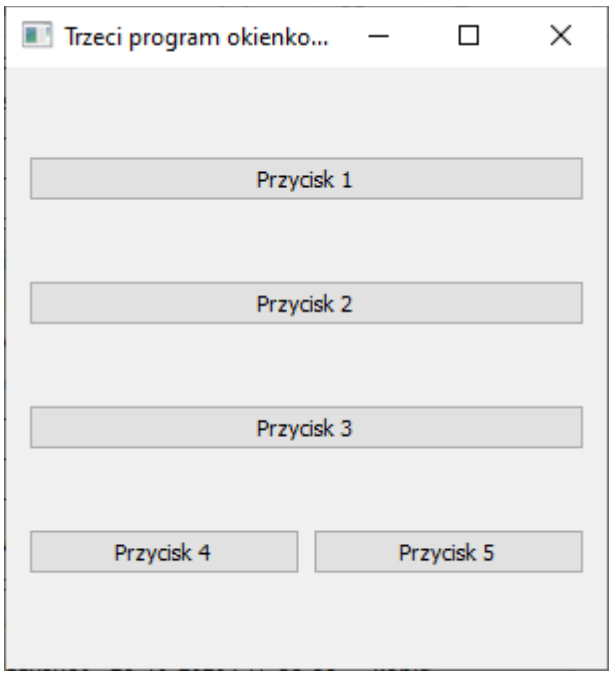
```
layout1 = QHBoxLayout()
```



3. Zagnieżdżanie układów.

Nic nie stoi na przeszkodzie, aby jeden układ (*layout*) był dodany jako element innego, jak w przykładowym kodzie poniżej:

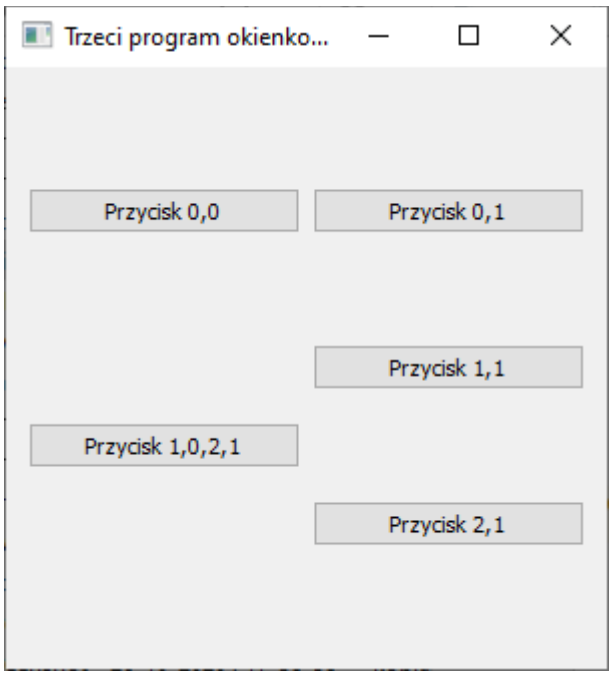
```
button1 = QPushButton('Przycisk 1', self)
button2 = QPushButton('Przycisk 2', self)
button3 = QPushButton('Przycisk 3', self)
button4 = QPushButton('Przycisk 4', self)
button5 = QPushButton('Przycisk 5', self)

layout1 = QVBoxLayout()
layout2 = QHBoxLayout()

layout2.addWidget(button4)
layout2.addWidget(button5)

layout1.addWidget(button1)
layout1.addWidget(button2)
layout1.addWidget(button3)
layout1.addLayout(layout2)
```



4. QGridLayout

Zamiast zagnieżdżania podstawowych układów możemy skorzystać także z umieszczania elementów na siatce.

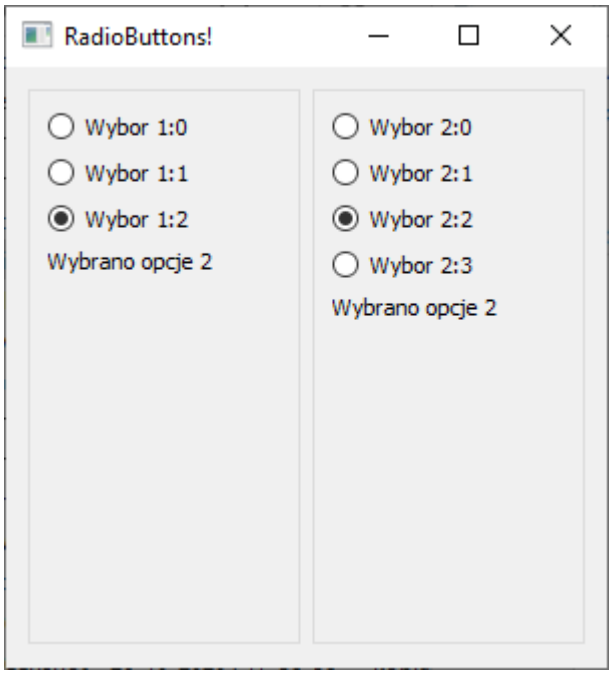
```
layout1 = QGridLayout()

layout1.addWidget(button1, 0, 0)
layout1.addWidget(button2, 1, 0, 2, 1)
layout1.addWidget(button3, 0, 1)
layout1.addWidget(button4, 1, 1)
layout1.addWidget(button5, 2, 1)
```



Pierwsze dwa parametry liczbowe oznaczają współrzędne na siatce, gdzie 0,0 oznacza lewy górny róg. Dwa kolejne (opcjonalne) parametry oznaczają kolejno ile pojedynczych komórek ma zostać scalonych w pionie i poziomie, czyli użyty w przykładzie zapis 1,0,2,1 oznacza, że komórka o współrzędnych 1,0 ma zostać scalona z jedną komórką poniżej (w sumie 2 komórki w pionie) ale nie ma być scalana w poziomie.

5. QRadioButton

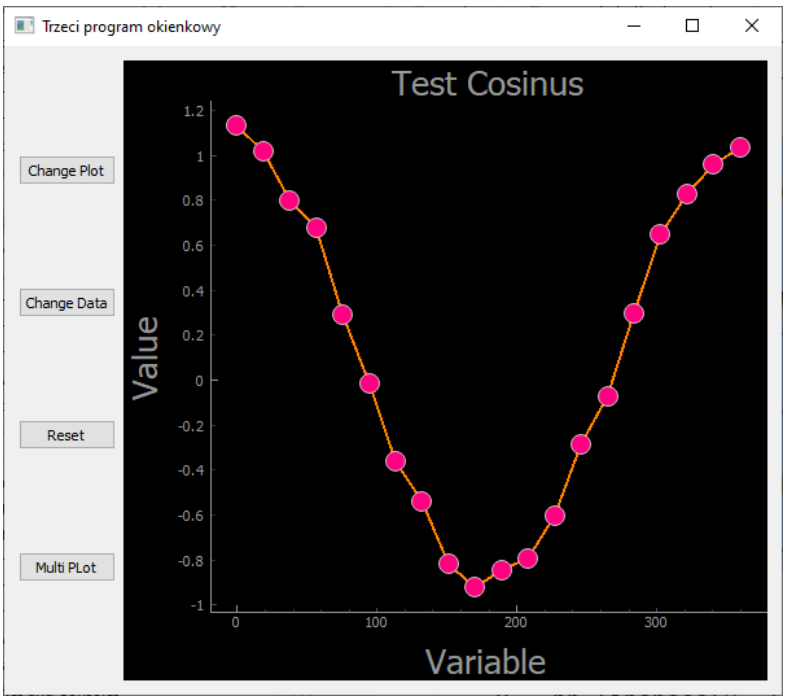
Kolejną możliwością przyjmowania wyboru użytkownika w PyQt5 są pola do zaznaczenia wyboru RadioButton.



Ponieważ przykład ten jest nieco bardziej skomplikowany proszę zapoznać się z dostępnym na UPELU przykładowym programem dołączonym jako materiały do niniejszych zajęć.

6. Wykresy w GUI - PyQtGraph

Aby móc w łatwy i natywny sposób dodawać wykresy do naszego środowiska graficznego możemy posłużyć się biblioteką pyqtgraph (który musimy podobnie jak pyqt5 doinstalować) do której dokumentacja dostępna jest na stronie http://www.pyqtgraph.org/ - są dostępne przykłady użycia do każdego typu wykresów.



Rysowanie podstawowego wykresu można wykonać umieszczając w klasie poniższy fragment kodu:

```
layout = QHBoxLayout()
self.graph = pg.PlotWidget()
layout.addWidget(self.graph)
x = np.linspace(0, 360, 400)
y = np.sin(x / 360 * np.pi * 2)
self.plot = self.graph.plot(x, y)
self.setLayout(layout)
```

W wykresie możemy dodać też różne istotne elementy i modyfikować jego kolory i styl:

```
pen = pg.mkPen(color=(255,128,0), width=2)
self.plot = self.graph.plot(x, y, pen=pen, symbol='o', symbolSize=15,
symbolBrush=(255,0,128))
self.graph.setLabel('left', text='<font size=15>Value</font>')
self.graph.setLabel('bottom', text='<font size=15>Variable</font>')
self.graph.setTitle('<font size=20>Test Cosinus</font>')
```

Rysować można także wiele przebiegów na jednym wykresie:

```
x = np.linspace(0, 360, 400)
y = np.sin(x / 360 * np.pi * 2)
self.graph.clear()
self.plot = self.graph.plot(x, y)
x = np.linspace(0, 360, 20)
y = np.cos(x / 360 * np.pi * 2)
pen = pg.mkPen(width=-1)
self.plot = self.graph.plot(x, y, pen=pen, symbol='o', symbolSize=15,
symbolBrush=(255, 0, 128))
```

Aby usunąć wykres i narysować nowy:

```
x = np.linspace(0, 360, 400)
y = np.sin(x / 360 * np.pi * 2)
self.graph.clear()
self.plot = self.graph.plot(x, y)
```

Lub zmodyfikować dane istniejącego wykresu:

```
x = np.linspace(0, 360, 20)
y = np.cos(x / 360 * np.pi * 2) + 0.15 * np.random.rand(len(x))
self.plot.setData(x, y)
```