Napraviti projekat u kom se nalazi taster i labela. Kada se program pokrene tajmer meri vreme 1 s i kada istekne uvećava vrednost upisanu u labeli. Pritiskom na taster pokreće se pauza od 5 s, što izaziva zamrzavanje programa.

```
from PySide6.QtWidgets import QVBoxLayout, QLabel, QPushButton, QWidget,
QMainWindow, QApplication
from PySide6.QtCore import QTimer

import sys
import time

class MainWindow(QMainWindow):

    def __init__(self):
        super(MainWindow, self).__init__()
        self.counter = 0
        layout = QVBoxLayout()

        self.l = QLabel("Start")
        b = QPushButton("DANGER!")
        b.pressed.connect(self.oh_no)

        layout.addWidget(self.l)
        layout.addWidget(b)

        w = QWidget()
        w.setLayout(layout)
        self.setCentralWidget(w)
        self.show()

        self.timer = QTimer()
        self.timer.setInterval(1000)
        self.timer.timeout.connect(self.recurring_timer)
        self.timer.start()

    def oh_no(self):
        time.sleep(5)

    def recurring_timer(self):
        self.counter +=1
        self.l.setText("Counter: %d" % self.counter)

app = QApplication(sys.argv)
window = MainWindow()
app.exec()
```

Izmeniti metodu oh_no kako bi se osvežio Eventloop.

Ovo nije uobičajen način rada!

```
    def oh_no(self):
        print('Oh no enters')
        for n in range(5):
            QApplication.processEvents()
            time.sleep(1)
        print('Oh no is over!')
```

Tredovi (programske niti) koriste istu memorijske lokacije što im daje veću brzinu i bržu međusobnu komunikaciju. Ovakav način rada može izazvati probleme sa pristupom memoriji (više niti pristupa jednoj lokaciji). Ovo je sređeno korišćenjem GIL-a (Global Interpreter lock) koji dozvoljava samo jednom trenutku u datom trenutku da se izvršava. To znači da u Pajtonu ne postoji realna multitreding aplikacija već se samo na jednom jezgru izvršava u jednom trenutku jedna nit. Ako se želi koristiti više jezgara onda se moraju koristiti procesi. Proces ima nezavisnu memoriju, kompleksniji su, zahtevaju više memorije itd.

Primer sa korišćenjem programskih niti.

```
from PySide6.QtWidgets import QVBoxLayout, QLabel, QPushButton, QWidget,
QMainWindow, QApplication
from PySide6.QtCore import QRunnable, Slot, QThreadPool

import sys
import time

class Worker(QRunnable):

    #@Slot()  #moze a i ne mora
    def run(self):
        print("Thread start")
        time.sleep(5)
        print("Thread complete")

class MainWindow(QMainWindow):

    def __init__(self, *args, **kwargs):
        super(MainWindow, self).__init__(*args, **kwargs)

        self.threadpool = QThreadPool()
        print("Multithreading with maximum {}
threads".format(self.threadpool.maxThreadCount()))

        self.counter = 0
        layout = QVBoxLayout()
        self.l = QLabel("Start")
        b = QPushButton("DANGER!")
        b.pressed.connect(self.oh_no)
        layout.addWidget(self.l)
        layout.addWidget(b)
        w = QWidget()
        w.setLayout(layout)
        self.setCentralWidget(w)
        self.show()

    def oh_no(self):
        worker = Worker()
        self.threadpool.start(worker)

app = QApplication(sys.argv)
window = MainWindow()
app.exec()
```

Moguće je proslediti funkciju koja nešto radi i nju pozivati iz posebne niti.

```
from PySide6.QtWidgets import QVBoxLayout, QLabel, QPushButton, QWidget,
QMainWindow, QApplication
from PySide6.QtCore import QRunnable, Slot, QThreadPool
import sys

class Worker(QRunnable):
    def __init__(self, fn, *args, **kwargs):
        super(Worker, self).__init__()
        self.fn = fn
        self.args = args
        self.kwargs = kwargs

    @Slot()  # QtCore.Slot
    def run(self):
        self.fn(*self.args, **self.kwargs)
        #time.sleep(5)

class MainWindow(QMainWindow):

    def __init__(self, *args, **kwargs):
        super(MainWindow, self).__init__(*args, **kwargs)

        self.threadpool = QThreadPool()
        print("Multithreading with maximum {}
threads".format(self.threadpool.maxThreadCount()))

        self.counter = 0
        layout = QVBoxLayout()
        self.l = QLabel("Start")
        b = QPushButton("DANGER!")
        b.pressed.connect(self.oh_no)
        layout.addWidget(self.l)
        layout.addWidget(b)
        w = QWidget()
        w.setLayout(layout)
        self.setCentralWidget(w)

        self.show()

    def execute_this_fn(self,bla=None):
        print("Hello!")
        self.counter+=1
        self.l.setText("Thread {}".format(self.counter))
        if bla != None:
            print(bla)

    def oh_no(self):
        # Pass the function to execute
        worker = Worker(self.execute_this_fn,'test')
        # Execute
        self.threadpool.start(worker)

app = QApplication(sys.argv)
window = MainWindow()
app.exec()
```

Moguće je i emitovati podatke iz niti ka glavnom programu.

```
import time
from PySide6.QtWidgets import QVBoxLayout, QLabel, QPushButton, QWidget,
QMainWindow, QApplication
from PySide6.QtCore import  QObject, QRunnable, Slot, QThreadPool, Signal
import sys
import random

class Worker(QRunnable,QObject):
    started = Signal(float)
    def __init__(self, fn, *args, **kwargs):
        QObject.__init__(self)
        QRunnable.__init__(self)
        self.fn = fn
        self.args = args
        self.kwargs = kwargs

        self.started.connect(fn)

    @Slot()  # QtCore.Slot
    def run(self):
        data = random.uniform(0,100)
        self.started.emit(data)
        time.sleep(5)

class MainWindow(QMainWindow):

    def __init__(self, *args, **kwargs):
        super(MainWindow, self).__init__(*args, **kwargs)

        self.threadpool = QThreadPool()
        print("Multithreading with maximum {}
threads".format(self.threadpool.maxThreadCount()))

        layout = QVBoxLayout()
        self.l = QLabel("Start")
        b = QPushButton("DANGER!")
        b.pressed.connect(self.oh_no)
        layout.addWidget(self.l)
        layout.addWidget(b)
        w = QWidget()
        w.setLayout(layout)
        self.setCentralWidget(w)

        self.show()

    def execute_this_fn(self,bla=None):
        print("Hello!")
        self.l.setText("Thread {}".format(bla))

    def oh_no(self):
        # Pass the function to execute
        worker = Worker(self.execute_this_fn)
        # Execute
        self.threadpool.start(worker)

app = QApplication(sys.argv)
window = MainWindow()
app.exec()
```

Rad sa serial portom. Napraviti prozor sa jednim plainTextEdit-om i dva tastera (Open) i (Close), za otvaranje i zatvaranje komunikacije.

```
from PySide6 import QtCore
from PySide6.QtWidgets import QApplication, QMainWindow
from PySide6 import QtSerialPort
from ui import Ui_MainWindow

class MainWindow(QMainWindow):
    def __init__(self):
        super(MainWindow, self).__init__()
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)

        self.ui.pushButton_2.setDisabled(True)
        self.ui.pushButton_2.clicked.connect(self.close_com)
        self.ui.pushButton.clicked.connect(self.start_com)

        self.serial = QtSerialPort.QSerialPort(
                    '/dev/ttyACM1',
                    baudRate=QtSerialPort.QSerialPort.Baud9600,
                    readyRead=self.receive
                )

    def receive(self):
        while self.serial.canReadLine():
            text = self.serial.readLine().data().decode()
            text = text.rstrip('\r\n')
            self.ui.plainTextEdit.appendPlainText(text)

    def start_com(self):
        if not self.serial.isOpen():
            self.ui.pushButton.setDisabled(True)
            self.ui.pushButton_2.setDisabled(False)
            self.serial.open(QtCore.QIODevice.ReadWrite)

    def close_com(self):
        self.serial.close()
        self.ui.pushButton_2.setDisabled(True)
        self.ui.pushButton.setDisabled(False)

app = QApplication()
window = MainWindow()
window.show()
app.exec()
```