ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ  
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

МОСКОВСКИЙ ИНСТИТУТ ЭЛЕКТРОНИКИ И МАТЕМАТИКИ

**ЗАДАНИЕ**

**для самостоятельной работы по индивидуальному плану**

**по курсу**

**«Проектный семинар “Python в науке о данных”»**

Студент группы БИВ222:

Плотников Роман Владимирович ([rvplotnikov@edu.hse.ru](mailto:rvplotnikov@edu.hse.ru) , +79288456340)

Тема работы:

«Программное обеспечение для анализа светимости клеток»

Руководитель:

Поляков Константин Львович

Москва 2023

**ОГЛАВЛЕНИЕ**

[Файлы библиотеки 3](#_Toc136669064)

[Файл Label.py: 3](#_Toc136669065)

[Файл Observe.py: 3](#_Toc136669066)

[Файл GraphWIndow.py: 3](#_Toc136669067)

[Файл PictureViewer.py: 3](#_Toc136669068)

[Файл ProgressBar.py: 4](#_Toc136669069)

[Файл Slider.py: 4](#_Toc136669070)

[Файл Toolbar.py: 4](#_Toc136669071)

[Файл SettingsWindow.py: 5](#_Toc136669072)

[Файл proceccing.py: 5](#_Toc136669073)

[Основной файл app.py 6](#_Toc136669074)

[Импортирует: 6](#_Toc136669075)

[class SaveMenu(QMenu, Observer) 6](#_Toc136669076)

[class MainWindow(QMainWindow, Observable, Observer) 6](#_Toc136669077)

[Запуск: 7](#_Toc136669078)

[Настройки 8](#_Toc136669079)

# Файлы библиотеки

## Файл Label.py:

1. class MyLabel(QWidget, Observer) – класс всех составных надписей в программе. При инициализации задаётся параметр title – текст, который впоследствии будет напечатан жирным шрифтом. Основной текст устанавливается при помощи метода def set\_text(self, \*args), принимающего на вход список строк.

## Файл Observe.py:

1. class Observable – класс для главного окна, реализующий паттерн «наблюдатель». Метод def attach(self, observer) прикрепляет объект класса Observer. Метод notify – обходит все прикреплённые объекты и вызывает у них метод update\_.
2. class Observer – класс всех элементов графического интерфейса, реализующий паттерн «наблюдатель».

Метод def update\_(self, dark\_mode) изменяет стиль объекта на соответствующий параметру bool dark\_mode.

## Файл GraphWIndow.py:

1. class Graph(QLabel) – класс ответственный за отображение графа в программе
2. class GraphWindow(QMainWindow, Observer) – класс реализующий окно графа.

Метод def init\_buttons\_stylesheets(self) устанавливает стили для кнопок

Метод def update\_buttons(self) обновляет стили кнопок

Метод def save\_graph(self) ответственен за сохранение графа в виде графического файла в формате png.

Метод def save\_raw(self) ответственен за сохранение данных, по которым строится граф в виде табличного файла в формате csv.

Метод def eventFilter(self, obj, event) – стандартный метод библиотеки PyQt5, отслеживающий наведение указателя мыши на кнопку и меняющий стиль кнопки.

## Файл PictureViewer.py:

1. class Scene(QGraphicsScene) – класс ответственный за отрисовку областей интереса на изображении.

Метод def mousePressEvent(self, event) – стандартный метод библиотеки PyQt5, отслеживающий нажатие левой кнопки мыши и устанавливающий начало координат для области интереса.

Метод def mouseMoveEvent(self, event) - стандартный метод библиотеки PyQt5, отслеживающий движение мыши с зажатой левой клавишей и непрерывно перерисовывающий область интереса.

Метод def mouseReleaseEvent(self, event) - стандартный метод библиотеки PyQt5, отслеживающий окончание нажатие левой и правой кнопок мыши. В случае с левой кнопкой в сцену и список содержимого сцены (self.item\_list) добавляется область интереса, представляющая из себя овал с порядковым номеров в его центре.

Метод def add\_oval(self, x, y, r), также как и метод def \_\_prepare\_for\_oval(self, event) используется в методах mouseMoveEvent и mouseReleaseEvent для создания области интереса.

Метод def repaint(self) перерисовывает элементы сцены в соответствующей теме.

1. class Viewer(QGraphicsView, Observer) – класс виджета загрузки и отображения изображений.

Метод def wheelEvent(self, event) стандартный метод библиотеки PyQt5, отслеживающий движение колёсика мыши и изменяющий масштаб отображения изображения.

Метод def show\_image(self, value=0, pixmap=None) реализует установку изображения в окно отображения. Если задан параметр pixmap, то отображаться будет переданное изображение, если не задан, то будет отображаться изображение из списка self.image\_paths с индексом value.

Группа методов def dragEnterEvent(self, event), def dragMoveEvent(self, event), def dropEvent(self, event) – стандартные методы библиотеки PyQt5, отслеживающий перенос графический файлов с помощью мыши на область виджета. Метод dropEvent добавляет пути для каждого переданного файла в список self.image\_paths и устанавливает первый файл в окно отображения, также испускает сигнал add\_pictures с длинной списка self.image\_paths в качестве параметра.

Сигнал add\_pictures используется для обозначения события добавления файлов.

## Файл ProgressBar.py:

1. class ProgressBarWindow(QProgressBar, Observer) – класс реализующий строку состояния операции вычисления.

## Файл Slider.py:

1. class Slider(QSlider, Observer) – класс реализующий слайдер для изображений.

## Файл Toolbar.py:

1. class Toolbar(QWidget, Observer) – класс реализующий виджет содержащий все инструменты, доступные в программе

Метод def turn\_shape\_mode(self, mode) реализует переключение в режим рисования областей интереса. Он отправляет соответствующее булево значение в качестве параметра сигнала draw\_mode.

Сигнал draw\_mode обозначает состояние программы.

Метод def update\_buttons(self) обновляет стили кнопок

Метод def eventFilter(self, obj, event) – стандартный метод библиотеки PyQt5, отслеживающий наведение указателя мыши на кнопку и меняющий стиль кнопки

## Файл SettingsWindow.py:

1. class ThemeModeButton(QPushButton) – реализует кнопку изменения стиля интерфейса.
2. class CmapList(QComboBox) – реализует набор палитр для окрашивания изображения.
3. class SettingsWindow(QMainWindow, Observer) – реализует окно настроек.

Метод def change\_mode(self, dark\_mode) реализует смену темы всего интерфейса.

Метод def change\_cmap(self, cmap) меняет палитру для инструмента окрашивания на выбранную

Метод def close\_(self) отменяет изменения и закрывает окно настроек.

Метод def apply (self) применяет выбранные настройки, сохраняет их в конфигурационный файл config.json и закрывает окно настроек.

Метод def init\_buttons\_stylesheets(self) устанавливает стили для кнопок

Метод def update\_buttons(self) обновляет стили кнопок

Метод def eventFilter(self, obj, event) – стандартный метод библиотеки PyQt5, отслеживающий наведение указателя мыши на кнопку и меняющий стиль кнопки.

## Файл proceccing.py:

1. class ROI – класс, реализующий области интереса. При инициализации необходимы координаты центра и радиус окружности.

Метод def measure(self, img: np.ndarray) -> float принимает на вход изображение в виде ndarray, и возвращает среднее число по яркости всех пикселей в области интереса на этом изображении.

Метод def get\_info(self) -> list возвращает информацию об области в виде списка из координат и радиуса окружности.

1. def open\_image(url) -> np.ndarray – функция открывающая изображение и возвращающая его 8-ми битовую серую версию в виде ndarray.
2. def make\_graph(data, f0, name="Example") -> str – функция создающая граф по данным data и f0 и сохраняющая во временном графическом файле формата png с именем name.
3. def colorize(pixmap, cmap) -> QPixmap: - функцию принимающая на вход изображение в виде QPixmap и название цветовой палитры из matplotlib и возвращающая окрашенное изображение в виде QPixmap.

# Основной файл app.py

Импортирует:   
from PyQt5.QtWidgets import QApplication, QMainWindow, QVBoxLayout, \

QHBoxLayout, QWidget, QGraphicsView, QGridLayout, QFileDialog, QMenu, QAction

from PyQt5.QtGui import QIcon, QPixmap

from PyQt5.QtCore import QMargins, pyqtSlot, pyqtSignal

import csv

import os

import json

from library import proceccing

from library.Observe import Observable, Observer

from library.Toolbar import Toolbar

from library.Label import MyLabel

from library.PictureViewer import Viewer

from library.Slider import Slider

from library.GraphWindow import GraphWindow

from library.SettingsWindow import SettingsWindow

from library.ProgressBar import ProgressBarWindow

class SaveMenu(QMenu, Observer) – реализует контекстное меню при нажатии на кнопку файла. Метод def leaveEvent(self, event) – стандартный метод библиотеки PyQt5, отслеживающий прекращение нахождения указателя мыши в области виджета и меняющий стиль кнопки файла.

class MainWindow(QMainWindow, Observable, Observer) – реализует главное окно.

Метод def init\_ui(self) создаёт все элементы графического интерфейса и подключает к соответствующим функциям.

Метод def change\_image(self, value) реализует смену изображений в окне отображения, а также значение имени изображения и порядкового номера кадра.

Метод def set\_slider\_max(self, val) устанавливает максимальное значение объекта класса Slider равным val.

Метод def calculate(self) реализует вычисления, показ прогрессбара и отображения окна графа.

Метод def save\_roi(self) сохраняет все текущие области интереса в файлу формата csv.

Метод def open\_roi(self) открывает файл формата csv с записанными областями интереса и добавляет их в текущую сцену.

Метод def save\_cur\_frame(self) сохраняет отображаемое изображение в графическом файле формата png.

Метод def colorize\_cur\_frame(self) окрашивает отображаемое изображение в установленную палитру.

Метод def closeEvent(self, e) -> None – стандартный метод библиотеки PyQt5, отслеживающий закрытие главного окно. Он удаляет временные файлы(Exapmle) и сохраняет настройки в конфигурационный файл config.json.

@pyqtSlot(int) - стандартный декоратор библиотеки PyQt5, необходимый для создания слота.

def handle\_add\_pics(self, quantity\_of\_pics) – слот, устанавливающий максимальное значение для слайдера равное quantity\_of\_pics, изменяющий имя изображения и разблокирующий панель инструментов.

def turn\_draw\_mode(self, checked) – слот, устанавливающий режим рисования в соответствии с bool checked. В режиме рисования пользователь не может перемещать изображение перетаскиванием с зажатой левой кнопкой мыши, вместо этого рисуется область интереса.

def change\_mode(self, dark\_mode) - слот, устанавливающий стили всех элементов графического интерфейса в соответствии с bool dark\_mode.

## Запуск:

app = QApplication([])

window = MainWindow()

window.show()

app.exec\_()

# Настройки

В папке work/scripts/settings/styles/dark хранятся тёмные стили для всех элементов графического интерфейса

В папке work/scripts/settings/styles/light хранятся светлые стили для всех элементов графического интерфейса

В папке work/scripts/settings хранится конфигурационный файл config.json содержащий: {"dark\_mode": false, "cmap": "cividis"}

В папке work/images/cmaps хранятся изображения цветовых палитр

В папке work/images/theme\_mode\_button\_icons хранятся два изображения, являющимися иконками для кнопки смена темы.

В папке work/images/ icons хранятся два изображения стрелочки выпадающего списка: cmap\_list\_arrow\_light.png для светлой темы и cmap\_list\_arrow\_dark.png для тёмной

В папке work/images/icons/dark\_theme/basic хранятся иконки невыделенных кнопок инструментов в тёмном стиле

В папке work/images/ icons/dark\_theme/chosen хранятся иконки выделенных кнопок инструментов в тёмном стиле

В папке work/images/ icons/light\_theme/basic хранятся иконки невыделенных кнопок инструментов в светлом стиле

В папке work/images/ icons/light\_theme/chosen хранятся иконки выделенных кнопок инструментов в светлом стиле

В папке work/images хранятся два изображения-плейсхолдера: default\_image.png – для пустого окно отображения изображений и cat.jpg для инициализации графа.