

# Занятие 25

PyQt6

# Какая из этих строчек неправильная?

1. `xyz = 1,000,000`
2. `x y z = 1000 2000 3000`
3. `x,y,z = 1000, 2000, 3000`
4. `x_y_z = 1_000_000`

# Что напечатает?

```
lst = [1, 0, 2, 0, 'hi', '', []]
```

```
result = list(filter(bool, lst))
```

```
print(result)
```

# Параметры и аргументы функций

```
def fun1(parameter):  
    return parameter * 2
```

```
def fun2(parameter):  
    return argument * 2
```

```
argument = 100  
print(fun1(argument), fun2(argument))
```

```
x = 200  
print(fun1(x), fun2(x))
```

```
# Что будет напечатано?
```

## Задача 24-2

Создайте запрос, который находит авторов, у которых только минимальное количество книг на складе (в таблице book).  
Используйте для этого View.

# Задача 24-3

Создайте функцию, которая принимает на входе строку из открывающих и закрывающих круглых скобок.

Функция возвращает False, если число скобок "(" не равно числу скобок ")".

Функция возвращает True, если в любой подстроке, начинающейся с начала, число скобок ")" не превышает число скобок "(".

Функция возвращает False во всех остальных случаях.

Примеры:

"()" => true

")(()()" => false

"(" => false

"(()) (( () () ) () )" => true

"()())" => false

# Решение задачи 24-3

```
def valid_parentheses(x):  
    while "()" in x:  
        x = x.replace("()", "")  
    return x == ""
```

```
def braces(s):  
    d = {'(':1, ')':-1}  
    r = 0  
    for i in s:  
        r += d[i]  
        if r < 0: return False  
    return r == 0
```

## Задача 24-1

Напишите функцию, которая сортирует числовой список, не используя никаких функций, вроде `sort`, `sorted` и т.д.

# Сортировка "пузырьком"

```
def bubble(lst):  
    for i in range(len(lst)):  
        for j in range(len(lst)-1):  
            if lst[j] > lst[j+1]:  
                lst[j], lst[j+1] = lst[j+1], lst[j]  
  
    return lst  
print(bubble([1,2,-7,6,0,5]))
```



# Сортировка "выбором"

```
def selection(lst):  
    for i in range(len(lst)-1):  
        mini, i_mini = lst[i], i  
        for j in range(i+1, len(lst)):  
            if lst[j] < mini:  
                mini, i_mini = lst[j], j  
        lst[i], lst[i_mini] = lst[i_mini], lst[i]  
  
    return lst  
print(selection([1,2,-7,6,0,5]))
```

# QuickSort

## Быстрая сортировка, сортировка

**Хоара** ([англ. quicksort](#)), часто называемая **qsort** (по имени в [стандартной библиотеке языка Си](#)) — [алгоритм сортировки](#), разработанный английским информатиком [Тони Хоаром](#) во время его работы в [МГУ](#) в [1960 году](#).

[Один из самых быстрых](#) известных универсальных алгоритмов сортировки массивов: в среднем  $O(n \cdot \log n)$  обменов при упорядочении  $n$  элементов

```
import random
def quicksort(nums):
    if len(nums) <= 1:
        return nums
    else:
        q = random.choice(nums)
        l_nums = [n for n in nums if n < q]
        e_nums = [q] * nums.count(q)
        b_nums = [n for n in nums if n > q]
        print(f"{l_nums=}")
        print(f"{e_nums=}")
        print(f"{b_nums=}")
        input()
        return quicksort(l_nums) + e_nums + quicksort(b_nums)
print(quicksort(list(map(int, input().split()))))
#11 1 9 2 8 3 12 7 4 6 5 10
```

# View

```
CREATE VIEW view_name AS
```

```
SELECT select_statement
```

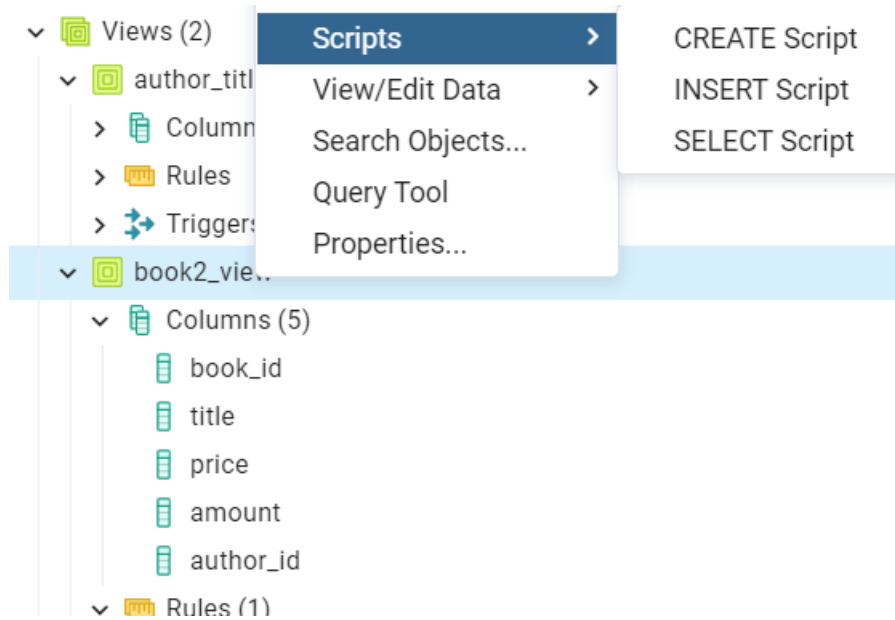
```
SELECT * FROM view_name
```

 – можем использовать в запросах

```
CREATE OR REPLACE VIEW
```

 – есть ограничения

# Как посмотреть View – через Create script



# Запрос в несколько этапов

Вывести авторов, общее количество книг которых на складе максимально.

Шаг 1. Найдем суммарное количество книг на складе по каждому автору. Поскольку фамилии автора в этой таблице нет, то группировку будем осуществлять по `author_id`.

```
SELECT author_id, SUM(amount) AS sum_amount FROM book GROUP BY author_id
```

Давайте перепишем сложный запрос с помощью создания view

## Шаг 2

Шаг 2. В результирующей таблице предыдущего запроса необходимо найти максимальное значение. Для этого запросу, созданному на шаге 1, необходимо присвоить имя (например, query\_in) и использовать его в качестве таблицы-источника после FROM. Затем уже находить максимум по столбцу sum\_amount.

```
SELECT MAX(sum_amount) AS max_sum_amount
FROM
(
  SELECT author_id, SUM(amount) AS sum_amount
  FROM book
  GROUP BY author_id
) query_in
```

# Шаг 3

Шаг 3. Выведем фамилию автора и общее количество книг для него.

```
SELECT name_author, SUM(amount) as Количество  
FROM  
    author INNER JOIN book  
    on author.author_id = book.author_id  
GROUP BY name_author
```

# Шаг 4

Шаг 4. Включим запрос с шага 2 в условие отбора запроса с шага 3. И получим всех авторов, общее количество книг которых максимально.

```
SELECT name_author, SUM(amount) as Количество
FROM author INNER JOIN book on author.author_id = book.author_id GROUP BY
name_author
HAVING SUM(amount) =
(/* вычисляем максимальное из общего количества книг каждого автора */
SELECT MAX(sum_amount) AS max_sum_amount
FROM (/* считаем количество книг каждого автора */
SELECT author_id, SUM(amount) AS sum_amount
FROM book GROUP BY author_id )
query_in );
```



Шаг 4'

```
CREATE VIEW max_sum_amount_1 as
```

```
SELECT MAX(sum_amount) AS max_sum_amount
```

```
FROM (/* считаем количество книг каждого автора */
```

```
SELECT author_id, SUM(amount) AS sum_amount
```

```
FROM book8 GROUP BY author_id )
```

```
query_in
```

```
SELECT author_name, SUM(amount) as Количество
```

```
FROM author9 INNER JOIN book8 on author9.author_id = book8.author_id GROUP  
BY author_name
```

```
HAVING SUM(amount) =
```

```
(select * from max_sum_amount_1)
```

# Типы данных

INTEGER — целое число (smallint, bigint, int2, int4, int8)

NUMERIC — число с плавающей точкой (decimal, real, double precision, float)

VARCHAR() — текстовые данные (char)

TEXT — набор с максимальной длиной 65535

DATE — дата (SELECT current\_time)

TIMESTAMP — дата и время (SELECT current\_timestamp)

BOOLEAN — логический тип

JSON — текстовый json

Массивы:

```
CREATE TABLE arr (id int, x int[]);
```

```
INSERT INTO arr VALUES
```

```
(1, '{1,2,3}'::int[]);
```

```
SELECT * FROM arr;
```

```
SELECT id, x[1] as the_first FROM arr
```

# Объединение, пересечение, «разность»

```
SELECT expression1, expression2,... expression_n  
FROM tables  
[WHERE conditions]  
UNION  
SELECT expression1, expression2,... expression_n  
FROM tables  
[WHERE conditions];
```

UNION — Объединяет в одну таблицу результаты 2-х и более запросов.  
Каждый оператор SELECT в операторе UNION должен иметь одинаковое количество полей в наборах результатов с одинаковыми типами данных.

UNION ALL — Для получения списка со всеми дубликатами.

INTERSECT — Возвращает пересечение результатов нескольких запросов.

EXCEPT — Возвращает исключение результатов второго запроса из первого.

# Index

Индекс – это дополнительная структура данных для ускорения работы запросов.

Результат запроса с индексом и без должны быть одинаковыми.

Индексы важны для поиска, для сортировки, группировки, соединения таблиц.

Индексы не всегда полезны!!!

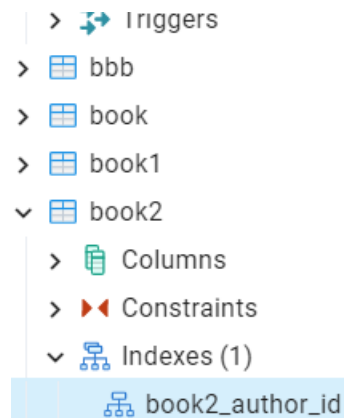
Затраты на содержание, могут замедлять работу при больших объемах при вводе и коррекции данных.

# CREATE INDEX

CREATE INDEX имя-индекса

ON имя-таблицы (имя-столбца, ...)

CREATE INDEX book2\_author\_id  
ON book2 (author\_id)



```
1  -- Index: book2_author_id
2  Loading...
3  -- DROP INDEX IF EXISTS public.book2_author_id;
4
5  CREATE INDEX IF NOT EXISTS book2_author_id
6      ON public.book2 USING btree
7      (author_id ASC NULLS LAST)
8      TABLESPACE pg_default;
9
```

# PyQt6

PyQt — это библиотека Python для создания приложений с графическим интерфейсом с помощью инструментария Qt.

Созданная в Riverbank Computing, PyQt является свободным ПО (по лицензии GPL) и разрабатывается с 1999 года.

Последняя версия PyQt6 — на основе Qt 6 — выпущена в 2021 году, и библиотека продолжает обновляться (Released: Dec 4, 2023)

# Инсталлировать PyQt6

PyCharm:

File \ Settings \ Project \ Python Interpretator \ + \ PyQt6 \ Install  
Package

```
pip install pyqt6
```

# Самое простое приложение

```
from PyQt6.QtWidgets import QApplication, QWidget
```

```
app = QApplication([]) # приложение
```

```
window = QWidget() # класс, от которого унаследованы все виджеты
```

```
window.show() # окно по умолчанию скрыто.
```

```
app.exec() # запускаем цикл событий, которые обрабатывает приложение
```

```
# Приложение не доберётся сюда, пока вы не выйдете и цикл событий не  
остановится.
```



# Самое простое приложение

```
from PyQt6.QtWidgets import QApplication, QWidget
import sys # Только для доступа к аргументам командной строки
# Приложению нужен один (и только один) экземпляр QApplication.
# Передаём sys.argv, чтобы разрешить аргументы командной строки для
приложения.
# Если не будете использовать аргументы командной строки, QApplication([]) тоже
работает
app = QApplication(sys.argv)
# Создаём виджет Qt — окно.
window = QWidget()
window.show() # Важно: окно по умолчанию скрыто.
# Запускаем цикл событий.
app.exec()
# Приложение не доберётся сюда, пока вы не выйдете и цикл событий не
остановится.
```

# Цикл событий

Основной элемент всех приложений в Qt — класс QApplication.

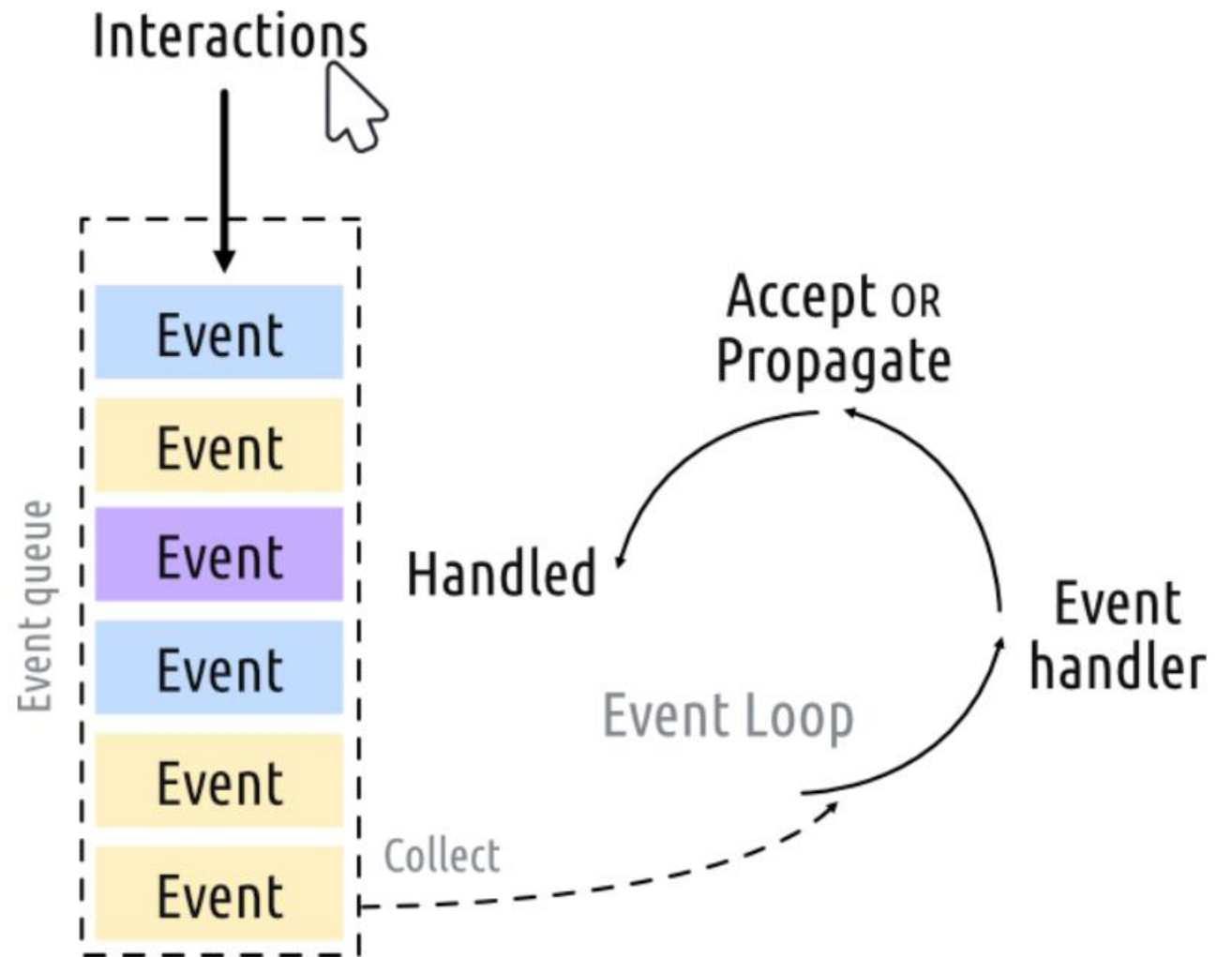
Для работы каждому приложению нужен один — и только один — объект QApplication, который содержит цикл событий приложения.

Это основной цикл, управляющий всем взаимодействием пользователя с графическим интерфейсом.

При каждом взаимодействии с приложением — будь то нажатие клавиши, щелчок или движение мыши — генерируется событие, которое помещается в очередь событий.

В цикле событий очередь проверяется на каждой итерации: если найдено ожидающее событие, оно вместе с управлением передаётся определённому обработчику этого события.

Последний обрабатывает его, затем возвращает управление в цикл событий и ждёт новых событий.



# Кнопка Push Me

```
import sys
from PyQt6.QtWidgets import QApplication, QPushButton
app = QApplication(sys.argv)
window = QPushButton("Push Me")
window.show()
app.exec()
```

# Окно

```
import sys
```

```
from PyQt6.QtWidgets import QApplication, QMainWindow
```

```
app = QApplication(sys.argv)
```

```
window = QMainWindow()
```

```
window.show()
```

```
# Запускаем цикл событий.
```

```
app.exec()
```

# Class QMainWindow

```
import sys
```

```
from PyQt6.QtWidgets import QApplication, QMainWindow, QPushButton
```

```
class MainWindow(QMainWindow):
```

```
    def __init__(self):
```

```
        super(MainWindow, self).__init__()
```

```
        self.setWindowTitle("My App")
```

```
app = QApplication(sys.argv)
```

```
window = MainWindow()
```

```
window.show()
```

```
app.exec()
```

# Окно фиксированного размера с кнопкой

```
from PyQt6.QtCore import QSize, Qt
from PyQt6.QtWidgets import QApplication, QMainWindow, QPushButton

class MainWindow(QMainWindow):      # Подкласс QMainWindow для настройки главного окна
    def __init__(self):
        super().__init__()
        self.setWindowTitle("My App")
        button = QPushButton("Press Me!")
        self.setFixedSize(QSize(400, 300))
        self.setCentralWidget(button)      # Устанавливаем центральный виджет Window.

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

# Нажатие кнопки (сигналы и слоты)

```
from PyQt6.QtWidgets import QApplication, QMainWindow, QPushButton

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("My App")
        self.button = QPushButton("Press Me!")
        self.button.setCheckable(True)
        self.button.clicked.connect(self.the_button_was_clicked)
        self.setCentralWidget(self.button)

    def the_button_was_clicked(self):
        print("Clicked!")          #self.button.setText('Other')

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

```
import sys

from PyQt6.QtWidgets import QApplication, QMainWindow,
QPushButton

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("My App")
        self.count = 0
        button = QPushButton("Press Me!")
        button.clicked.connect(self.the_button_was_clicked)
        self.setCentralWidget(button)

    def the_button_was_clicked(self):
        self.count += 1
        print("Clicked!" + str(self.count))
        self.setWindowTitle(f"Clicked! {self.count}")
```

# Clicked

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



# Одноразовая кнопка

```
class MainWindow(QMainWindow):  
    def __init__(self):  
        super().__init__()   
        self.setWindowTitle("My App")  
        self.button = QPushButton("Press Me!")  
        self.button.clicked.connect(self.the_button_was_clicked)  
        self.setCentralWidget(self.button)  
  
    def the_button_was_clicked(self):  
        self.button.setText("You already clicked me.")  
        self.button.setEnabled(False)  
        # Также меняем заголовок окна.  
        self.setWindowTitle("My Oneshot App")  
        # self.setWindowTitle("A new window title")
```

# Задание

Создайте графическое приложение, в котором реализована кнопка “Hello World”, чтобы нажатие на нее меняло ее название на “Привет Мир” и наоборот.

# Widgets

```
import sys

from PyQt6.QtWidgets import (
    QMainWindow, QApplication,
    QLabel, QCheckBox, QComboBox, QLineEdit,
    QLineEdit, QSpinBox, QDoubleSpinBox, QSlider
)

from PyQt6.QtCore import Qt

class MainWindow(QMainWindow):

    def __init__(self):
        super(MainWindow, self).__init__()

        self.setWindowTitle("My App")

app = QApplication(sys.argv)
w = MainWindow()
w.show()
app.exec()
```

# QLabel

```
from PyQt6.QtCore import QSize, Qt
        ..... QPushButton, QLabel
```

```
class MainWindow(QMainWindow):
```

```
    def __init__(self):
```

```
        super(MainWindow, self).__init__()
```

```
        self.setWindowTitle("My App")
```

```
        label = QLabel("Hello")
```

```
        font = label.font()
```

```
        font.setPointSize(30)
```

```
        label.setFont(font)
```

```
        label.setAlignment(Qt.AlignmentFlag.AlignHCenter | Qt.AlignmentFlag.AlignVCenter)
```

```
        self.setCentralWidget(label)
```

# QLineEdit

```
class MainWindow(QMainWindow):
```

```
    def __init__(self):
```

```
        super(MainWindow, self).__init__()
```

```
        self.setWindowTitle("My App")
```

```
        widget = QLineEdit()
```

```
        widget.setMaxLength(10)
```

```
        widget.setPlaceholderText("Enter your text")
```

```
        #widget.setReadOnly(True) # раскомментируйте, чтобы  
        #сделать доступным только для чтения
```

```
        widget.returnPressed.connect(self.return_pressed)
```

```
        widget.selectionChanged.connect(self.selection_changed)
```

```
        widget.textChanged.connect(self.text_changed)
```

```
        widget.textEdited.connect(self.text_edited)
```

```
        self.setCentralWidget(widget)
```

```
    def return_pressed(self):
```

```
        print("Return pressed!")
```

```
    def selection_changed(self):
```

```
        print("Selection changed")
```

```
    def text_changed(self, s):
```

```
        print("Text changed...")
```

```
        print(s)
```

```
    def text_edited(self, s):
```

```
        print("Text edited...")
```

```
        print(s)
```

# QPushButton

```
class MainWindow(QMainWindow):
    def __init__(self):
        super(MainWindow, self).__init__()

        self.setWindowTitle("My App")
        self.button = QPushButton("Результат!")
        self.button.setCheckable(True)
        self.button.clicked.connect(self.the_button_was_clicked)

        self.setCentralWidget(self.button)
        self.tf = True

    def the_button_was_clicked(self):
        print("Clicked!")
        if self.tf:
            self.setWindowTitle('Result')
            self.tf = False
        else:
            self.tf = True
            self.setWindowTitle('MyApp')
```

# Calculator - 1

```
from PyQt6.QtCore import QSize, Qt
from PyQt6.QtWidgets import QApplication, QMainWindow, QPushButton,
QLabel,
QVBoxLayout, QWidget, QLineEdit
```

```
class MainWindow(QMainWindow):
```

```
    def __init__(self):
```

```
        super().__init__()
```

```
        self.setWindowTitle("Calculator")
```

```
        w, h = 200, 25
```

```
        self.line1 = QLineEdit()
```

```
        self.line1.setFixedSize(QSize(w, h))
```

```
        self.line2 = QLineEdit()
```

```
        self.line2.setFixedSize(QSize(w, h))
```

```
        self.button_add = QPushButton("+")
```

```
        self.button_add.setFixedSize(QSize(w, h))
```

```
        self.button_add.clicked.connect(self.the_button_add)
```

# Calculator - 2

```
self.button_sub = QPushButton("-")  
self.button_sub.setFixedSize(QSize(w, h))  
self.button_sub.clicked.connect(self.the_button_sub)
```

```
self.button_mul = QPushButton("*")  
self.button_mul.setFixedSize(QSize(w, h))  
self.button_mul.clicked.connect(self.the_button_mul)
```

```
self.button_div = QPushButton("/")  
self.button_div.setFixedSize(QSize(w, h))  
self.button_div.clicked.connect(self.the_button_div)
```

```
self.label = QLabel("=")  
self.label.setFixedSize(QSize(w, h))  
font = self.label.font()  
font.setPointSize(10)  
self.label.setFont(font)
```



# Calculator - 3

```
layout = QVBoxLayout()
widgets = [self.line1, self.line2, self.label, self.button_add, self.button_sub,
self.button_mul, self.button_div]
for w in widgets:
    layout.addWidget(w)

widget = QWidget()
widget.setLayout(layout)
self.setCentralWidget(widget)

def the_button_add(self):
    try:
        res = '='+str(eval(self.line1.text() + '+' + self.line2.text()))
    except:
        res = 'Mistake!'
    self.label.setText(res)

def the_button_sub(self):
    try:
        res = '='+str(eval(self.line1.text() + '-' + self.line2.text()))
    except:
        res = 'Mistake!'
    self.label.setText(res)
```

# Calculator - 4

```
def the_button_mul(self):  
    try:  
        res = '='+str(eval(self.line1.text() + '*' + self.line2.text()))  
    except:  
        res = 'Mistake!'  
    self.label.setText(res)  
def the_button_div(self):  
    try:  
        res = '='+str(eval(self.line1.text() + '/' + self.line2.text()))  
    except:  
        res = 'Mistake!'  
    self.label.setText(res)
```

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

# Задание

Доработайте программу калькулятор, чтобы она умела делать целочисленное деление и остаток от деления, т.е. добавьте кнопки `//` и `%`

# Случайный выбор Window Title по нажатию кнопки

```
from PyQt6.QtWidgets import QApplication, QMainWindow, QPushButton
import sys
from random import choice

window_titles = [ 'My App', 'My App',
    'Still My App',    'Still My App',    'What on earth',
    'What on earth',    'This is surprising',
    'This is surprising',    'Something went wrong']

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.n_times_clicked = 0
        self.setWindowTitle("My App")
        self.button = QPushButton("Press Me!")
        self.button.clicked.connect(self.the_button_was_clicked)
        self.windowTitleChanged.connect(self.the_window_title_changed)
        self.setCentralWidget(self.button)
```

```
    def the_button_was_clicked(self):
        print("Clicked.")
        new_window_title = choice(window_titles)
        print("Setting title: %s" % new_window_title)
        self.setWindowTitle(new_window_title)

    def the_window_title_changed(self, window_title):
        print("Window title changed: %s" % window_title)
        if window_title == 'Something went wrong':
            self.button.setDisabled(True)

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

# QMouseEvent

QMouseEvent — одно из основных событий, получаемых виджетами. События QMouseEvent создаются для каждого отдельного нажатия кнопки мыши и её перемещения в виджете. Вот обработчики событий мыши:

Обработчик	Событие
<code>mouseMoveEvent</code>	Мышь переместилась
<code>mousePressEvent</code>	Кнопка мыши нажата
<code>mouseReleaseEvent</code>	Кнопка мыши отпущена
<code>mouseDoubleClickEvent</code>	Обнаружен двойной клик

Например, нажатие на виджет приведёт к отправке QMouseEvent в обработчик событий `.mousePressEvent` в этом виджете.

# Mouse

```
import sys
from PyQt6.QtCore import Qt
from PyQt6.QtWidgets import QApplication, QLabel, QMainWindow, QTextEdit
class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.label = QLabel("Click in this window")
        self.setCentralWidget(self.label)
    def mouseMoveEvent(self, e):
        self.label.setText("mouseMoveEvent")
    def mousePressEvent(self, e):
        self.label.setText("mousePressEvent")
    def mouseReleaseEvent(self, e):
        self.label.setText("mouseReleaseEvent")
    def mouseDoubleClickEvent(self, e):
        self.label.setText("mouseDoubleClickEvent")
app = QApplication(sys.argv)
window = MainWindow()
window.show()
app.exec()
```

# Задача 25-1

Разработайте счетчик нажатий на кнопку, который меняет свою надпись на число нажатий на нее (1, 2, 3 и т.д.)

## Задача 25-2

Реализуйте функцию `is_palindrome()` с использованием **рекурсии**, которая принимает один аргумент `string` — произвольная строка.

Функция должна возвращать значение `True`, если переданная строка является палиндромом, или `False` в противном случае.

Примечание 1. Палиндром — текст, одинаково читающийся в обоих направлениях.

Примечание 2. Пустая строка является палиндромом, как и строка, состоящая из одного символа.



# Задача 25-3

Напишите функцию, которой на вход подается строка, содержащая последовательность слов (которые могут включать буквы верхнего и нижнего регистра). На выходе должна получиться строка в CamelStyle.

Например, "cAmel cAse woRD" => CamelCaseWord