

Занятие 26

PyQt6

type()

Что напечатает?

```
a = [1,2]
```

```
a += '34'
```

```
print(a)
```

```
a = [1,2]
```

```
a = a + '34'
```

```
print(a)
```

Задача 25-1

Простой вариант:

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

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 (последний релиз 30.10.2023)

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

Окно

```
import sys
```

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

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

```
window = QMainWindow()
```

```
window.show()
```

```
# Запускаем цикл событий.
```

```
app.exec()
```


Цикл событий

Основной элемент всех приложений в Qt — класс QApplication.

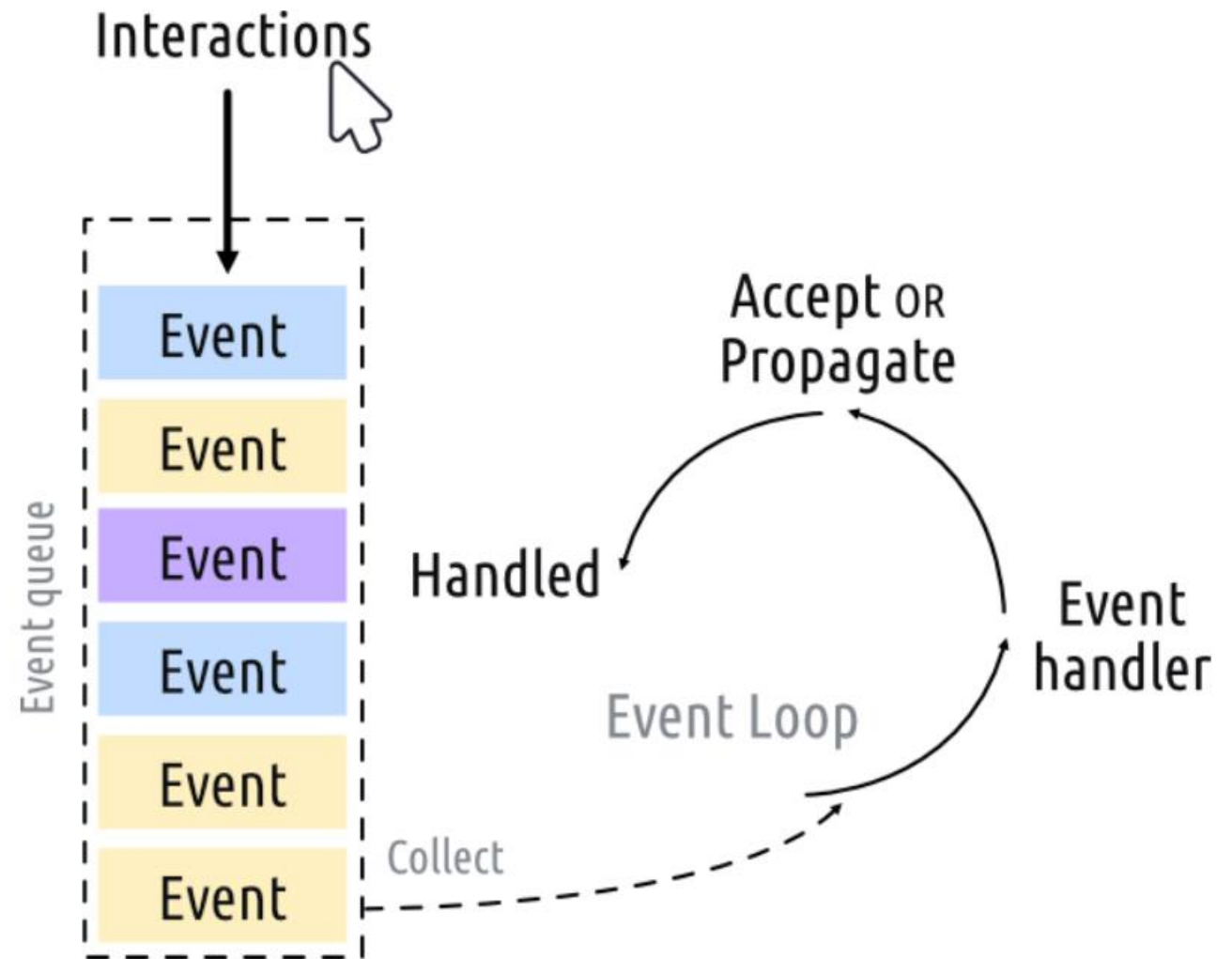
Для работы каждому приложению нужен один — и только один — объект QApplication, который содержит цикл событий приложения.

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.



app.py

```
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

```
import sys

from PyQt6.QtCore import QSize, Qt

from PyQt6.QtWidgets import QApplication, QMainWindow, QPushButton

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

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

Нажатие кнопки

```
import sys
from PyQt6.QtWidgets import QApplication, QMainWindow, QPushButton
class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("My App")
        button = QPushButton("Press Me!")
        button.setCheckable(True)
        button.clicked.connect(self.the_button_was_clicked)
        # Устанавливаем центральный виджет Window.
        self.setCentralWidget(button)
    def the_button_was_clicked(self):
        print("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 заголовки окна.  
        self.setWindowTitle("My Oneshot App")  
        # self.setWindowTitle("A new window title")
```

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()
```

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

```
widget = QLabel("Hello")
```

```
widget = QLabel("1") # Создана метка с текстом 1.
```

```
widget.setText("2") # Создана метка с текстом 2.
```

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

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

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

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

```
        widget = QLabel("Hello")
```

```
        font = widget.font()
```

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

```
        widget.setFont(font)
```

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

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


QPushButton

```
button = QPushButton("Результат!")
button.setCheckable(True)
button.clicked.connect(self.the_button_was_clicked)

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

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)
```

Виджеты

Класс виджета	Описание виджета
QCheckbox	Чекбокс
QComboBox	Окно выпадающего списка
QDateEdit	Для редактирования даты и времени
QDateTimeEdit	Для редактирования даты и времени
QDial	Поворотный циферблат
QDoubleSpinBox	Спиннер для чисел с плавающей точкой
QFontComboBox	Список шрифтов
QLCDNumber	Довольно неприятный дисплей LCD
QLabel	Просто метка, не интерактивная
QLineEdit	Поле ввода со строкой
QProgressBar	Индикатор выполнения
QPushButton	Кнопка
QRadioButton	Переключаемый набор, в котором активен только один элемент
QSlider	Слайдер
QSpinBox	Спиннер для целых чисел
QTimeEdit	Поле редактирования времени

QGridLayout

```
import sys

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

from PyQt6.QtCore import Qt

class MainWindow(QMainWindow):
    def __init__(self):
        super(MainWindow, self).__init__()
        self.setWindowTitle("My App")

        layout = QGridLayout()
        button = QPushButton('Button')
        le = QLineEdit('LineEdit')
        label = QLabel('Label')
        combo = QComboBox()
```

```
        combo.addItem(['First', 'Second', 'Third'])
        layout.addWidget(button, 0, 0)
        layout.addWidget(le, 0, 1)
        layout.addWidget(label, 1, 0)
        layout.addWidget(combo, 1, 1)
```

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

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

Меню, тулбар

```
import sys
```

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

```
from PyQt6.QtGui import QIcon, QAction
```

```
class Example(QMainWindow):
```

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

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

```
        self.initUI()
```

```
    def initUI(self):
```

```
        textEdit = QTextEdit()
```

```
        self.setCentralWidget(textEdit)
```

```
        exitAction = QAction(QIcon('t1.png'), 'Exit',  
self)
```

```
        exitAction.setShortcut('Ctrl+Q')
```

```
        exitAction.setStatusTip('Exit application')
```

```
        exitAction.triggered.connect(self.close)
```

```
        self.statusBar()
```

```
        menubar = self.menuBar()
```

```
        fileMenu = menubar.addMenu('&File')
```

```
        fileMenu.addAction(exitAction)
```

```
        toolbar = self.addToolBar('Exit')
```

```
        toolbar.addAction(exitAction)
```

```
        self.setGeometry(300, 300, 350, 250)
```

```
        self.setWindowTitle('Main window')
```

```
        self.show()
```

```
if __name__ == '__main__':
```

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

```
    ex = Example()
```

```
    sys.exit(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()
```

MDI – Multiple Document Interface

```
from PyQt6.QtWidgets import QApplication, QMainWindow, QMdiArea, QMdiSubWindow, QTextEdit
from PyQt6.QtGui import QAction
import sys
```

```
class MDIWindow(QMainWindow):
    count = 0
```

```
    def __init__(self):
        super().__init__()

        self.mdi = QMdiArea()
        self.setCentralWidget(self.mdi)
        bar = self.menuBar()
        file = bar.addMenu("File")
        file.addAction("New")
        file.addAction("Cascade")
        file.addAction("Tiled")
        file.triggered[QAction].connect(self.WindowTrig)
        self.setWindowTitle("MDI Application")
```

```
    def WindowTrig(self, p):
        if p.text() == "New":
            MDIWindow.count = MDIWindow.count + 1
            sub = QMdiSubWindow()
            sub.setWidget(QTextEdit())
            sub.setWindowTitle("Sub" + str(MDIWindow.count))
            self.mdi.addSubWindow(sub)
            sub.show()
        if p.text() == "Cascade":
            self.mdi.cascadeSubWindows()
        if p.text() == "Tiled":
            self.mdi.tileSubWindows()
```

```
app = QApplication(sys.argv)
mdiwindow = MDIWindow()
mdiwindow.show()
app.exec()
```


ООП – работа с атрибутами

Создатели Python предложили следующий ряд соглашений:

- если имя атрибута начинается с одного нижнего подчеркивания (`_name`), то он считается **защищенным (protected)**
- если имя атрибута начинается с двух нижних подчеркиваний (`__name`), то он считается **приватным (private)**

В Python все атрибуты являются **публичными (public)** по умолчанию.

Любой атрибут объекта может быть доступен за пределами класса, независимо от того, является он публичным, защищенным или приватным.

```
class AnyClass:
    def __init__(self):
        self.a = 'Public'
        self._b = 'Protected'
        self.__c = 'Private'

inst = AnyClass()
print(inst.a)
print(inst._b)
print(inst.__c)
```

Private атрибуты

```
class Anyclass:  
    def __init__(self, x, y):  
        self.x = x  
        self.__y = y  
any = Anyclass(123, 456)  
print(any.x)  
print(any.__y)  
print(any._Anyclass__y)
```

Что будет?

Два нижних подчеркивания (**dooble underscore**) - дандер

Если поставить перед именем атрибута два подчеркивания, к нему нельзя будет обратиться напрямую.

```
class SomeClass():  
    def __init__(self):  
        self.__param = 42 # приватный атрибут
```

```
obj = SomeClass()
```

```
obj.__param # AttributeError: 'SomeClass' object has no attribute '__param'
```

```
obj._SomeClass__param # - обходной способ.
```

```
obj.__dict__
```

Методы по работе с атрибутами

```
class AnyClass:
    def __init__(self):
        self.a = 'Public'
        self._b = 'Protected'
        self.__c = 'Private'
        print(self.__dict__)
    def get_c(self): # set delete
        return self.__c

inst = AnyClass()
print(inst.a)
print(inst._b)
print(inst._AnyClass__c)
print(inst.get_c())
```

Доступ к __атрибутам

```
class Person:
```

```
    def __init__(self, name):
```

```
        self.__name = name # устанавливаем имя
```

```
        self.__age = 1 # устанавливаем возраст
```

```
    def set_age(self, age):
```

```
        if 1 < age < 110:
```

```
            self.__age = age
```

```
        else:
```

```
            print("Недопустимый возраст")
```

```
    def get_age(self):
```

```
        return self.__age
```

```
    def get_name(self):
```

```
        return self.__name
```

```
    def display_info(self):
```

```
        print(f"Имя: {self.__name}\tВозраст: {self.__age}")
```

```
tom = Person("Tom")
```

```
tom.display_info() # Имя: Tom  Возраст: 1
```

```
tom.set_age(25)
```

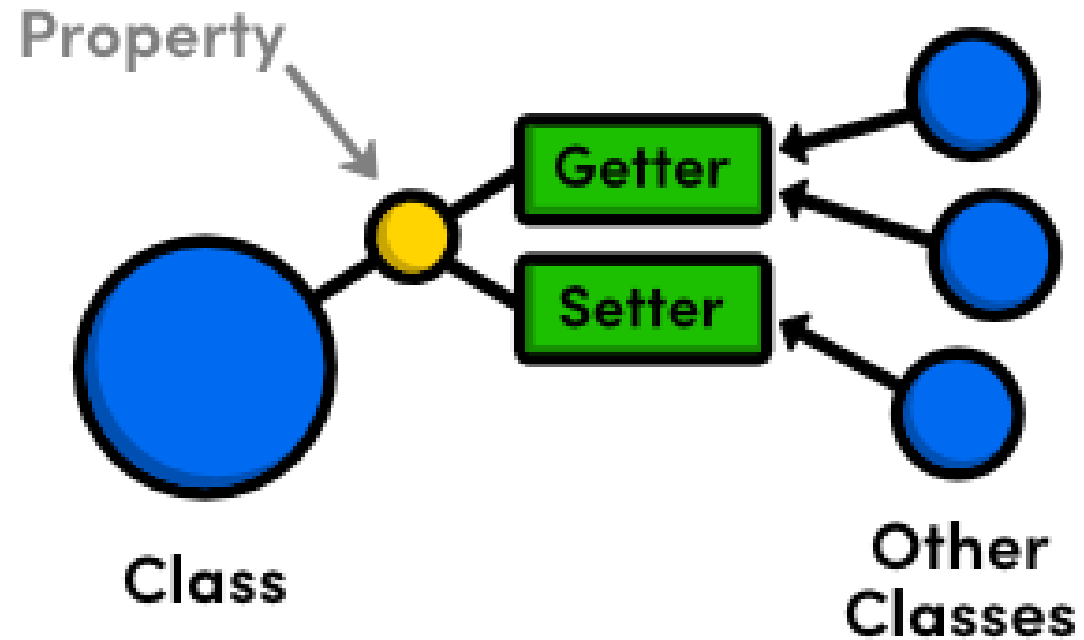
```
tom.display_info() # Имя: Tom  Возраст: 25
```

```
#tom.__name
```

```
#tom._Person__name
```

Выполните эту программу

Методы доступа к свойствам



Декоратор @property

```
class Cat:
```

```
    def __init__(self, name):
```

```
        self.__name = name                # ИМЯ КОШКИ
```

```
    @property
```

```
    def name(self):                        # геттер свойства name
```

```
        return self.__name
```

```
    @name.setter
```

```
    def name(self, name1):                # сеттер свойства name
```

```
        if isinstance(name1, str) and name1.isalpha():
```

```
            self.__name = name1
```

```
        else:
```

```
            raise ValueError('Некорректное имя')
```

```
    @name.deleter
```

```
    def name(self):                        # делитер свойства name
```

```
        del self.__name
```

```
c = Cat('abc')
```

```
print(c.name)
```

```
c.name = 'def'
```

```
print(c.name)
```

```
del c.name
```

```
print(c.name)
```

@property декоратор может быть использован для определения методов в классе , которые действуют как атрибуты.

В этих методах можно реализовать разнообразные полезные функции.

Задание

Доопределите проверку ввода свойства `__name`, чтобы первая буква была большая, а остальные маленькие, например, Tom, Kitty, Васька

Если имя не подходит, то программа должна написать «Формат имени должен быть “Имя”».

Методы getattr, setattr, delattr, hasattr

```
#hasattr(object, name)
```

```
class A:
```

```
    x = 123
```

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

```
        self.x = 456
```

```
a = A()
```

```
y = 'x'
```

```
print(hasattr(a, 'x'))
```

```
print(hasattr(A, 'x'))
```

```
print(hasattr(a, y))
```

```
print(hasattr(A, y))
```

getattr(object, name, default)

```
class MyObj:
```

```
    name = 'Petr Ivanov'
```

```
    phone = '+79211234567'
```

```
    country = 'RU'
```

```
x = getattr(MyObj, 'phone')
```

```
print(x)
```

```
s = 'country'
```

```
y = getattr(MyObj, s)
```

```
print(y)
```

```
# Удалим атрибут phone из объекта MyObj:
```

```
delattr(MyObj, 'phone')
```

```
# Пробуем получить атрибут phone из объекта MyObj:
```

```
x = getattr(MyObj, 'phone', '+71234567890')
```

```
print(x)
```

```
setattr(object, name, value)
```

```
class MyObj:
```

```
    name = 'Ivan Sidorov'
```

```
    phone = '+7987987654321'
```

```
setattr(MyObj, 'phone', '+71112223344')
```

```
setattr(MyObj, 'country', 'RU')
```

```
# Получим атрибуты из объекта MyObj:
```

```
human = MyObj()
```

```
x = getattr(human, 'phone', '+7000000000')
```

```
y = human.country
```

```
print(x, y)
```

Встроенные методы

Можно перегрузить встроенные методы, если нам нужно как-то преобразовать значения, или устроить проверку вводимых атрибутов, или какие-то побочные эффекты.

- `__getattr__`
- `__setattr__`
- `__delattr__`

Встроенные методы: `__getattr__`

автоматически вызывается при получении несуществующего свойства класса

```
class SomeClass():
```

```
    attr1 = 42
```

```
    def __getattr__(self, attr):
```

```
        return attr.upper()
```

```
obj = SomeClass()
```

```
obj.attr1 # 42
```

```
obj.attr2 # ATTR2
```

```
# Выполните этот код
```

__getattrute__

Перехватывает все обращения к атрибутам (в том числе и к существующим атрибутам):

```
class SomeClass():  
    attr1 = 42  
    def __getattrute__(self, attr):  
        return attr.upper()  
obj = SomeClass()  
obj.attr1 # ATTR1  
obj.attr2 # ATTR2  
# Выполните этот код
```

Встроенные методы: `__setattr__`

`#автоматически вызывается при изменении свойства класса;`

```
class SomeClass():  
    age = 42  
    def __setattr__(self, name, value):  
        print(name, value)  
        self.__dict__[name] = value  
        #self.age = value # рекурсия!!!
```

```
obj = SomeClass()
```

```
obj.age # 42
```

```
obj.age = 100 # Вызовет метод __setattr__
```

```
# Выполните этот код
```

`__delattr__` удаление атрибута

```
class Car:
    def __init__(self):
        self.speed = 100
    def __delattr__(self, attr):
        self.speed = 42

# Создаем объект
porsche = Car()
print(porsche.speed)
# 100
delattr(porsche, 'speed') ← Удаление атрибута у объекта
print(porsche.speed) → 42
```


Метаклассы

Мы можем добавлять методы и атрибуты из программы

Мы можем создавать экземпляры классов из программы

Как создавать классы из программы?

Для этого используется функция:

`type(Имя класса, Родительские классы, Словарь атрибутов и методов)`

type() - Метаклассы

```
class Foo(object):  
    bar = True  
Foo = type('Foo', (), {'bar': True})    # То же самое!  
#-----  
class FooChild(Foo):  
    pass  
FooChild = type('Foochild', (Foo,), {})    # То же самое  
#-----  
def echo_bar(self):  
    print(self.bar)  
FooChild = type('FooChild', (Foo,), {'echo_bar': echo_bar})    # Можно определить метод  
  
my_foo = FooChild()  
my_foo.echo_bar()
```

Задание

Создайте класс с помощью функции `type()`

с атрибутом `age = 50`

и методом `show`, который печатает этот возраст.

Задача 26-1

Напишите функцию, которая сравнивает две строки и выдает True, если между ними есть не более, чем 1 разница в буквах, и False во всех остальных случаях. Если две строки равны, то True.

Например:

'abc' и 'abc' – True, 'abc' и 'abcd' – True,

'bc' и 'abc' – True, 'ahc' и 'abc' – True

'abc' и 'acb' – False, 'abc' и 'a' – False, "" и ' ' - False

Задача 26-2

Создайте класс Pet, используя функцию type и с методом dis, на основе функции dis, определенной заранее, и печатающему все атрибуты класса Pet (например, name, age).

Функцию dis для метода Pet.dis определите заранее.

Подсказка:

```
def dis(self): # метод определяется заранее
    # в цикле печатать атрибуты и их значения из словаря self.__dict__
Pet = type('Pet', (), {'dis': dis}) # Определяется класс
my_pet = Pet()
my_pet.name = 'Tom'
```

Задача 26-3

Создайте метод `Person`, определите в нем атрибут `self._age`

Используйте декоратор `@property` для определения методов `getter`, `setter`, `deleter`.

В методе `setter` определите проверку, что возраст не может быть меньше 1 и больше 100, при попытке установить этот возраст программа должна печатать «Недопустимый возраст».

Задача 26-4 (необязательное)

Создайте графическое приложение с пунктами меню: Описание, Инструкция, Помощь.