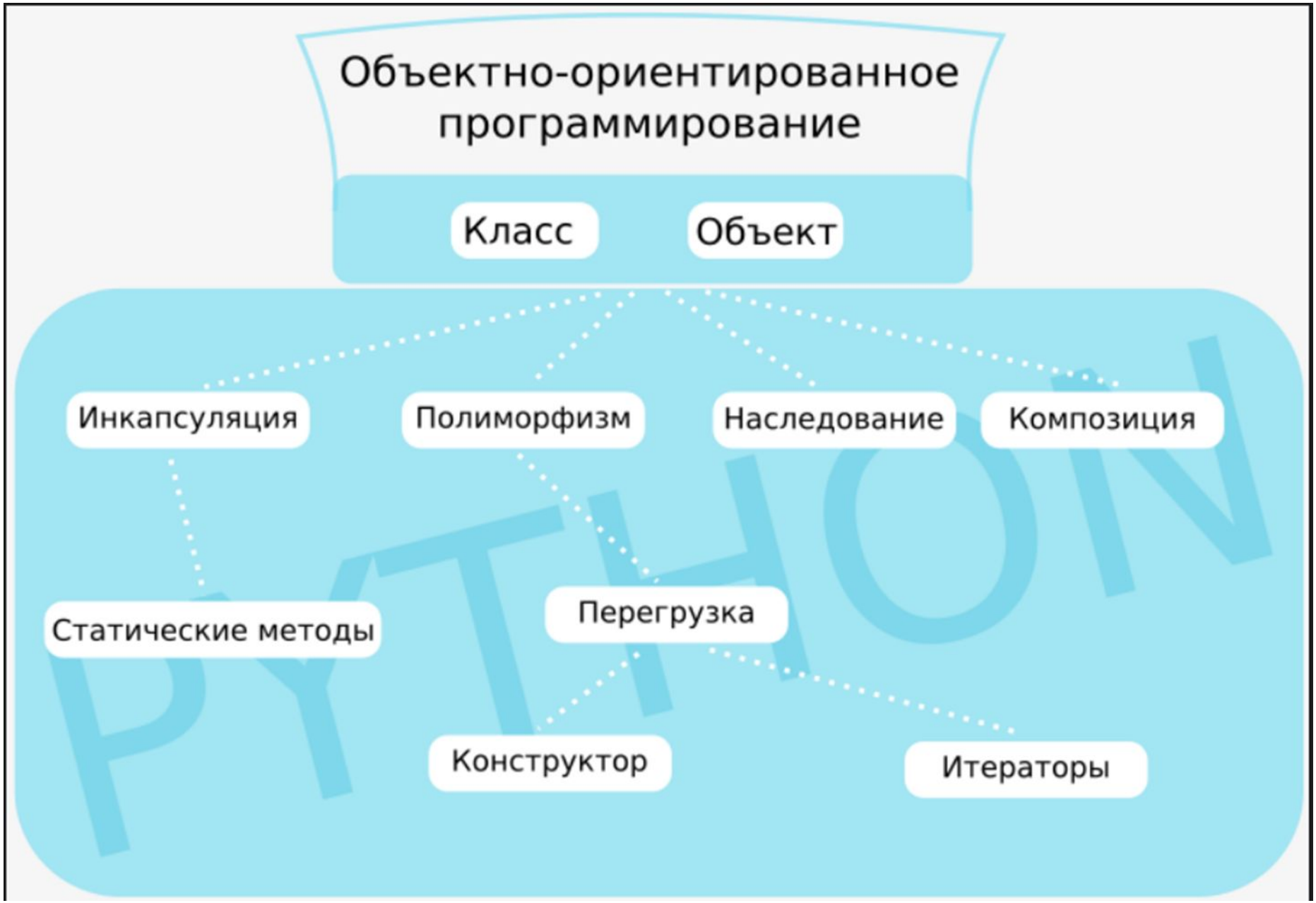
# Объектно-ориентированное программирование. Язык Python

---

3 курс 6 семестр

- 01_07.02.2022_(дистант)_Лекция 1 из 19_конспект
- 02_XX.02.2022_(дистант)_Лекция 1 из 19_Контрольные вопросы
- 03_XX.02.2022_(дистант)_Лекция 1 из 19_Тесты
- 04_XX.02.2022_(дистант)_ПР№ 1 из 15_Задания)
- 05_15.02.2022_ПР№ 2 из 15_Задания)(*.py)
- 06_05.04.2022_(дистант)_Лекция 2 из 19_конспект_Интерпритатор Питона как Калькулятор
- 07_05.04.2022_(дистант)_Лекция 2 из 19_Контрольные вопросы
- 08_05.04.2022_(дистант)_Лекция 2 из 19_тесты
- 09_05.04.2022_(дистант)_ПР№ 2 из 10_Задания)(*.py)
- XX_до 21.04.2022_Проект «Создание собственного программного продукта»


Объектно-ориентированное программирование
Класс
Объект
Инкапсуляция
Полиморфизм
Наследование
Композиция
Статические методы
Перегрузка
Конструктор
Итераторы
PYTHON

# Объектно-ориентированное программирование. Язык Python

# Объектно-ориентированное программирование. Язык Python

## § 46. Что такое ООП?

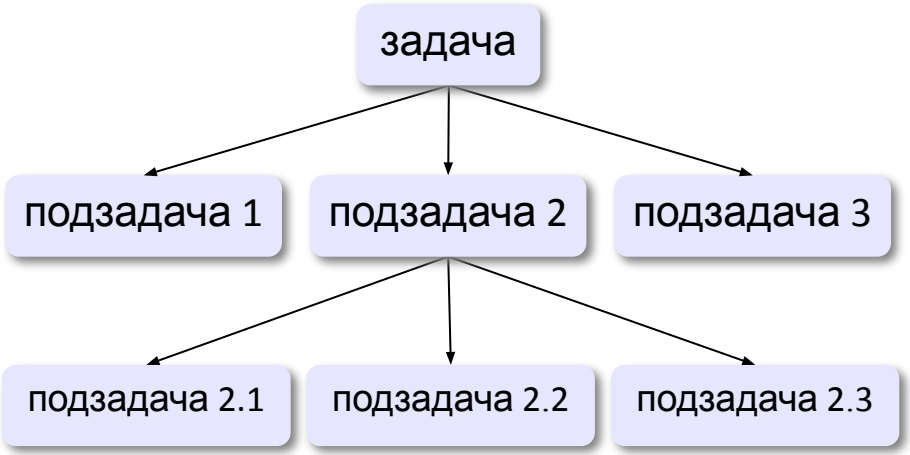
# Зачем нужно что-то новое?

Главная проблема – **сложность**!

- программы из миллионов строк
- тысячи переменных и массивов

*Э. Дейкстра*: «Человечество еще в древности придумало способ управления сложными системами: «**разделяй и властвуй**»».

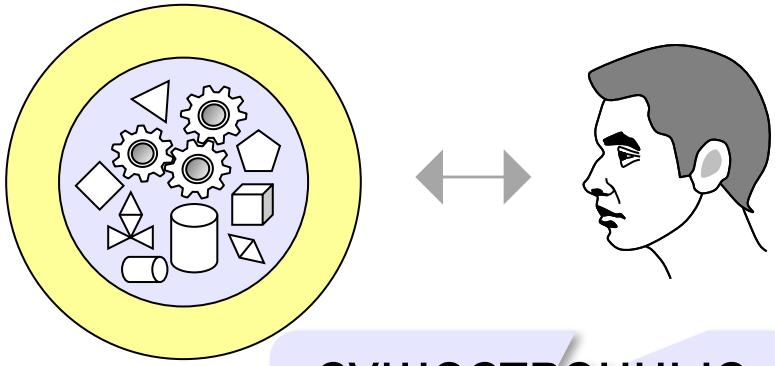
**Структурное программирование**:




**декомпозиция** по задачам

человек мыслит иначе, объектами

# Как мы воспринимаем объекты?



существенные свойства

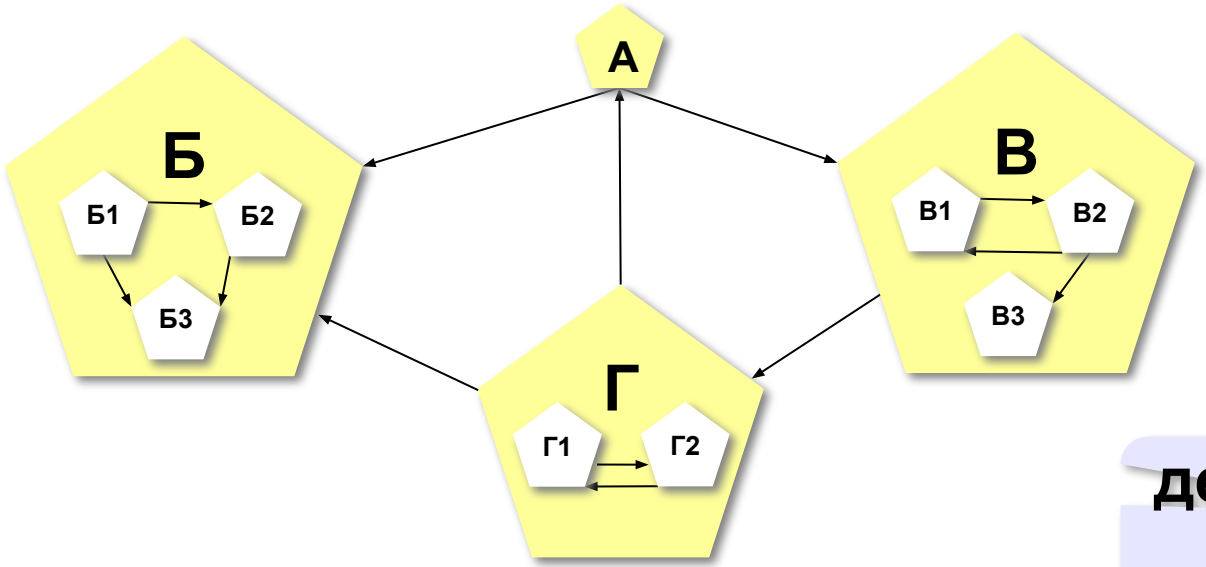
**Абстракция** – это выделение существенных свойств объекта, отличающих его от других объектов.

! Разные цели – **разные модели**!

# Использование объектов

**Программа** – множество объектов (моделей), каждый из которых обладает своими свойствами и поведением, но его внутреннее устройство скрыто от других объектов.

! Нужно «разделить» задачу на объекты!




**декомпозиция** по объектам

# Объектно-ориентированное программирование. Язык Python

## § 47. Объекты и классы

# С чего начать?

**Объектно-ориентированный анализ (ООА):**

- выделить **объекты**
- определить их существенные **свойства**
- описать **поведение** (команды, которые они могут выполнять)

? Что такое объект?

**Объектом** можно назвать то, что имеет чёткие границы и обладает *состоянием* и *поведением*.

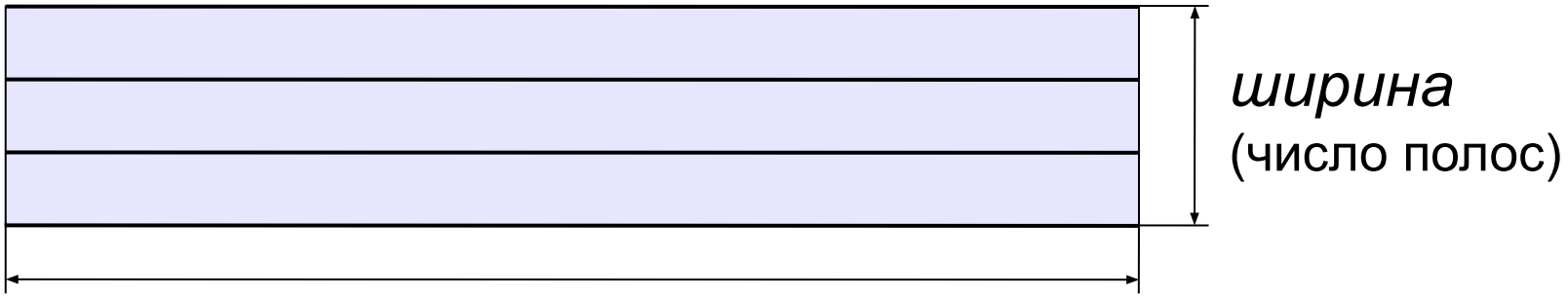
**Состояние определяет поведение:**

- лежачий человек не прыгнет
- незаряженное ружье не выстрелит

**Класс** – это множество объектов, имеющих общую структуру и общее поведение.

# Модель дороги с автомобилями

## Объект «Дорога»:




название класса

| Дорога |
| --- |
| длина<br>ширина |
| |

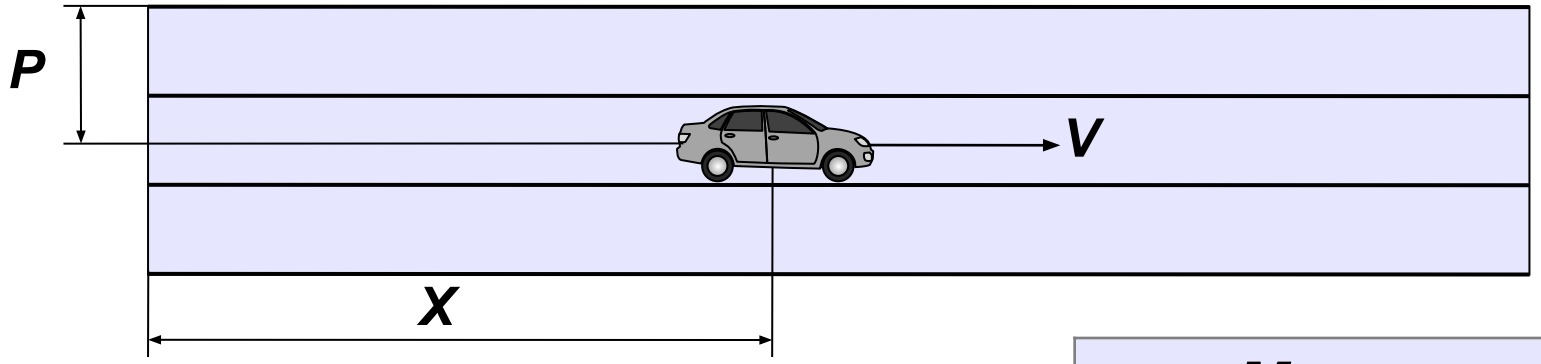
**свойства** (состояние)

**методы** (поведение)

# Модель дороги с автомобилями

## Объект «Машина»:

свойства: координаты и скорость




- все машины одинаковы
- скорость постоянна
- на каждой полосе – одна машина
- если машина выходит за правую границу дороги, вместо нее слева появляется новая машина

| *Машина* |
|---|
| X (координата)<br>P (полоса)<br>V (скорость) |
| двигаться |

**Метод** – это процедура или функция, принадлежащая классу объектов.

# Модель дороги с автомобилями

## Взаимодействие объектов:

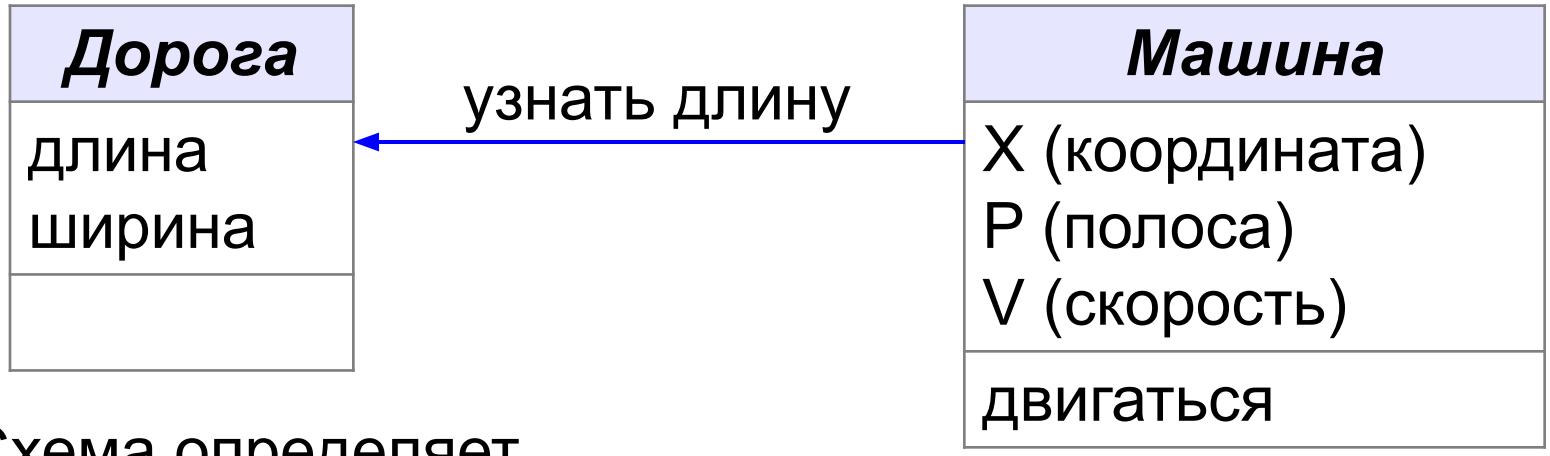



Схема определяет

- **свойства** объектов
- **методы**: операции, которые они могут выполнять
- **связи** (обмен данными) между объектами

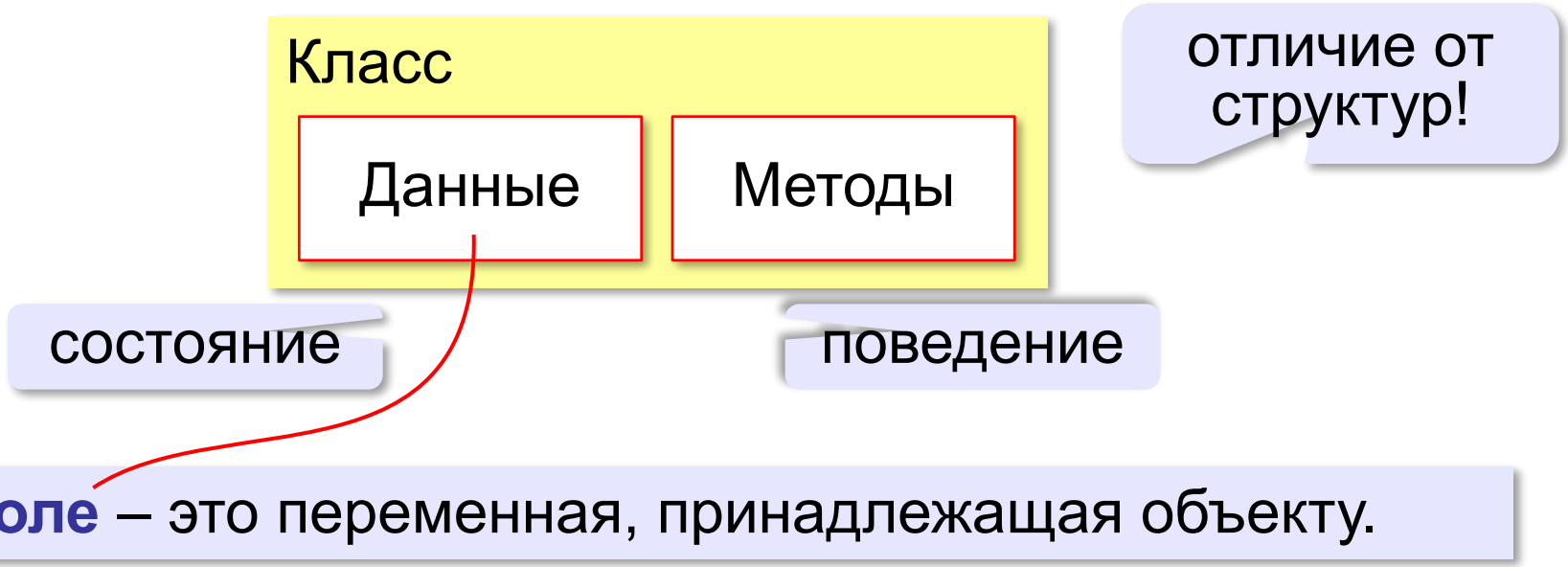
! Ни слова о внутреннем устройстве объектов!

# Объектно-ориентированное программирование. Язык Python

## § 48. Создание объектов в программе

# Классы

- программа – множество взаимодействующих **объектов**
- любой объект – экземпляр какого-то **класса**
- **класс** – описание группы объектов с общей структурой и поведением




**Поле** – это переменная, принадлежащая объекту.

# Класс «Дорога»

**Описание класса**:

```
class TRoad:
  pass
```

Объекты-экземпляры не создаются!

**Создание объекта**:

```
road = TRoad()
```

вызов конструктора

**Конструктор** – это метод класса, который вызывается для создания объекта этого класса.

Конструктор по умолчанию строится автоматически!

# Новый конструктор – добавлений полей

*initialization* – начальные установки

```
class TRoad:
  def __init__( self ):
    self.length = 0
    self.width = 0
```

ссылка для обращения к самому объекту

оба поля обнуляются

точечная запись

! Конструктор задаёт начальные значения полей!

```
road = TRoad()
road.length = 60
road.width = 3
```

изменение значений полей

# Конструктор с параметрами

автоматически

```
class TRoad:
  def __init__( self, length0, width0 ):
    self.length = length0
    self.width = width0
```

**Вызов**:

```
road = TRoad( 60, 3 )
```

Нет защиты от неверных входных данных!

# Защита от неверных данных

```
class TRoad:
  def __init__( self, length0, width0 ):
    if length0 > 0:
      self.length = length0
    else:
      self.length = 0
    if width0 > 0:
      self.width = width0
    else:
      self.width = 0
```

```
self.length = length0 if length0 > 0 else 0
self.width = width0 if width0 > 0 else 0
```

# Класс «Машина»

дорога, по которой едет

полоса

```
class TCar:
  def __init__( self, road0, p0, v0 ):
    self.road=road0
    self.P=p0
    self.V=v0
    self.X=0
```

скорость

координата

# Класс «Машина» – метод move

```
class TCar:
  def __init__( self, road0, p0, v0 ):
    ...
  def move( self ):
    self.X += self.V
    if self.X > self.road.length:
      self.X = 0
```

перемещение за Δt = 1

если за пределами дороги

**Равномерное движение**:

$$X = X_0 + V \cdot \Delta t$$

$\Delta t = 1$ интервал дискретизации

перемещение за одну единицу времени

# Основная программа

```
N = 3
cars = []
for i in range(N):
  cars.append( TCar(road, i+1, 2*(i+1)) )

for k in range(100):  # 100 шагов
  for i in range(N):  # для каждой машины
    cars[i].move()

print( "После 100 шагов:" )
for i in range(N):
  print( cars[i].X )
```

# Что в этом хорошего и плохого?

**ООП** – это метод разработки **больших** программ!

- основная программа – простая и понятная
- классы могут разрабатывать разные программисты независимо друг от друга (+интерфейс!)
- повторное использование классов

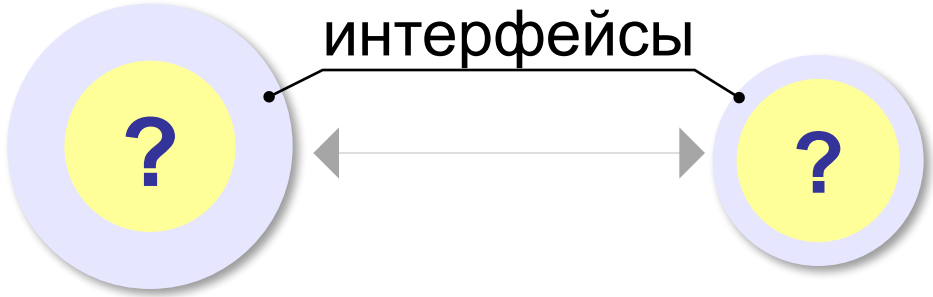
- неэффективно для небольших задач

# Объектно-ориентированное программирование. Язык Python

## § 49. Скрытие внутреннего устройства

# Зачем скрывать внутреннее устройство?

**Объектная модель задачи**:




- защита внутренних данных
- проверка входных данных на корректность
- изменение устройства с сохранением интерфейса

**Инкапсуляция** («помещение в капсулу») – скрытие внутреннего устройства объектов.

Также объединение данных и методов в одном объекте!

# Пример: класс «перо»

```
class TPen:
  def __init__( self ):
    self.color="000000"
```

R G B

По умолчанию все члены класса открытые (в других языках – **public**)!

```
class TPen:
  def __init__( self ):
    self.__color="000000"
```

Как обращаться к полю?

Имена скрытых полей (**private**) начинаются с двух знаков подчёркивания!

# Пример: класс «перо»

```
class TPen:
  def __init__( self ):
    self.__color="000000"
  def getColor( self ):
    return self.__color
  def setColor( self, newColor ):
    if len(newColor) !=6:
      self.__color="000000"
    else:
      self.__color=newColor
```

метод чтения

метод записи

если ошибка, чёрный цвет

Защита от неверных данных!

# Пример: класс «перо»

**Использование**:

```
pen = TPen()
pen.setColor ( "FFFF00" )
print ( "цвет пера:", pen.getColor() )
```

установить цвет

Не очень удобно!

прочитать цвет

```
pen.color = "FFFF00"
print ( "цвет пера:", pen.color )
```

# Свойство color

**Свойство** – это способ доступа к внутреннему состоянию объекта, имитирующий обращение к его внутренней переменной.

```
class TPen:
  def __init__ ( self ):
    ...
  def __getColor ( self ):
    ...
  def __setColor ( self, newColor ):
    ...
  color = property ( __getColor,
                     __setColor )
```

метод чтения

свойство

метод записи

```
pen.color = "FFFF00"
print ( "цвет пера:", pen.color )
```

# Изменение внутреннего устройства

## Удобнее хранить цвет в виде числа:

```
class TPen:
  def __init__( self ):
    self.__color = 0
  def __getColor( self ):
    return "{:06x}".format( self.__color )
  def __setColor( self, newColor ):
    if len(newColor) != 6:
      self.__color = 0
    else:
      self.__color = int( newColor, 16 )
  color = property (__getColor, __setColor)
```

число

число

число

Интерфейс не изменился!

# Преобразование int → hex

## Целое – в шестнадцатеричную запись:

16711935 → "FF00FF"

```
x = 16711935
sHex = "{:x}".format(x)
```

? Что плохо?

в шестнадцатеричной системе

255 → "FF" "0000FF"

правильно так!

```
x = 16711935
sHex = "{:06x}".format(x)
```

дополнить нулями слева

занять 6 позиций

# Преобразование hex → int

"FF00FF" → 16711935

```
sHex = "FF00FF"
x = int ( sHex, 16 )
```

система счисления

# Свойство «только для чтения»

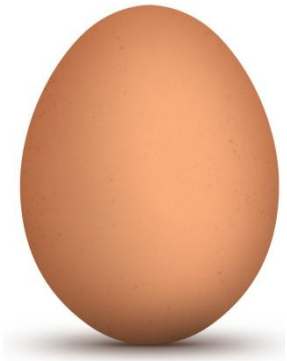
**Скорость машины** можно только читать:

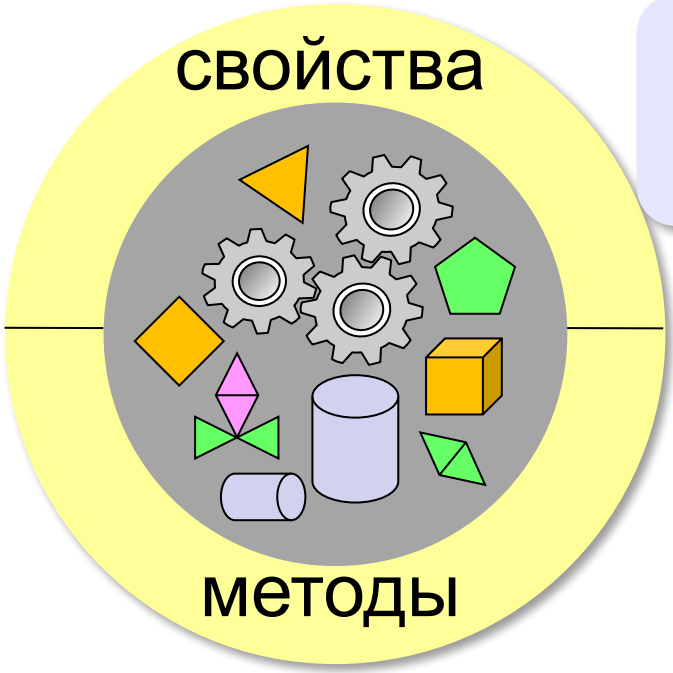
```
class TCar:
  def __init__( self ):
    self.__v = 0
  v = property( lambda x: x.__v )
```

нет метода записи

# Скрытие внутреннего устройства

**Инкапсуляция** («помещение в капсулу»)






внутреннее устройство (`private`)

интерфейс (`public`)

# Объектно-ориентированное программирование. Язык Python

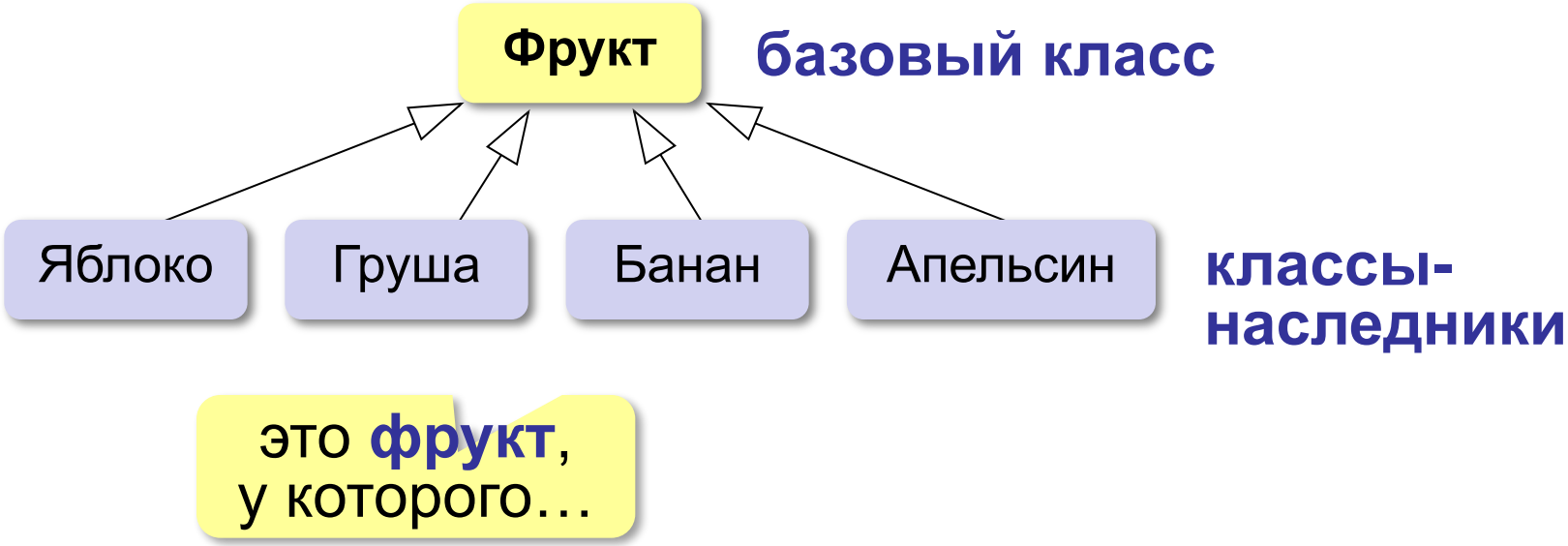
## § 50. Иерархия классов

# Классификации

? Что такое классификация?

**Классификация** – разделение изучаемых объектов на группы (классы), объединенные общими признаками.

? Зачем это нужно?



# Что такое наследование?

класс *Двудольные*
семейство *Бобовые*
род *Клевер*
**горный клевер**

наследует свойства (имеет все свойства)

Класс Б является **наследником** класса А, если можно сказать, что Б – **это разновидность** А.

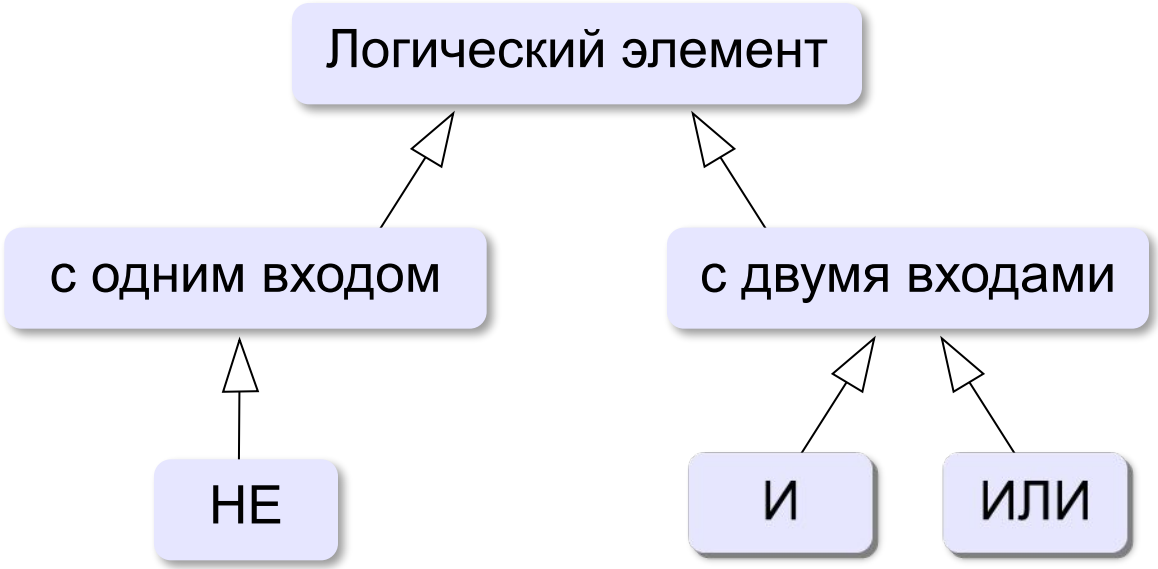
✓ яблоко – фрукт
✓ горный клевер – клевер
✗ машина – двигатель

яблоко – **это** фрукт

горный клевер – **это** растение рода *Клевер*

машина **содержит** двигатель (часть – целое)

# Иерархия логических элементов




**Объектно-ориентированное программирование** – это такой подход к программированию, при котором программа представляет собой множество взаимодействующих **объектов**, каждый из которых является экземпляром определенного **класса**, а классы образуют иерархию **наследования**.

# Базовый класс

| *ЛогЭлемент* |
| --- |
| In1 (вход 1)<br>In2 (вход 2)<br>Res (результат) |
| calc |

```
class TLogElement:
  def __init__ ( self ):
    self.__in1 = False
    self.__in2 = False
    self._res = False
```

? Зачем хранить результат?

поле доступно наследникам!

можно моделировать элементы с памятью (триггеры)

# Базовый класс

```
class TLogElement:
  def __init__( self ):
    self.__in1 = False
    self.__in2 = False
    self._res = False
  def __setIn1 ( self, newIn1 ):
    self.__in1 = newIn1
    self.calc()
  def __setIn2 ( self, newIn2 ):
    self.__in2 = newIn2
    self.calc()
  In1 = property (lambda x: x.__in1, __setIn1)
  In2 = property (lambda x: x.__in2, __setIn2)
  Res = property (lambda x: x._res )
```

пересчёт выхода

только для чтения

# Метод calc

? Как написать метод **calc**?

```
class TLogElement:
    ...
    def calc ( self ):
        pass
```

заглушка

! Нужно запретить создавать объекты **TLogElement**!

# Абстрактный класс

- все логические элементы должны иметь метод `calc`
- метод `calc` невозможно написать, пока неизвестен тип логического элемента

**Абстрактный метод** – это метод класса, который объявляется, но не реализуется в классе.

**Абстрактный класс** – это класс, содержащий хотя бы один абстрактный метод.

нет логического элемента «вообще», как не «фрукта вообще», есть конкретные виды

Нельзя создать объект абстрактного класса!

`TLogElement` – абстрактный класс из-за метода `calc`

# Абстрактный класс

```
class TLogElement:
  def __init__ ( self ):
    self.__in1 = False
    self.__in2 = False
    self._res = False

    if not hasattr ( self, "calc" ):
      raise NotImplementedError(
            "Нельзя создать такой объект!")
```

если у объекта нет атрибута (поля или метода) с именем `calc`…

создать («поднять», «выбросить») исключение

# Что такое полиморфизм?

```
class TLogElement:
  def __init__( self ):
    ...
  def __setIn1 ( self, newIn1 ):
    self.__in1 = newIn1
    self.calc()
```

для каждого наследника вызывается свой метод **calc**

**Полиморфизм** – это возможность классов-наследников по-разному реализовать метод с одним и тем же именем.

греч.: $\pi o \lambda \upsilon$ — много, $\mu o \rho \varphi \eta$ — форма

# Элемент «НЕ»

наследник от **TLogElement**

вызов конструктора базового класса

```
class TNot ( TLogElement ):
  def __init__ ( self ):
    TLogElement.__init__ ( self )

  def calc ( self ):
    self._res = not self.In1
```

? Почему не **__in1**?

! Это уже не абстрактный класс!

# Элемент «НЕ»

**Использование**:

```
n = TNot()
n.In1 = False
print ( n.Res )
```

создание объекта

установка входа

вывод результата

# Элементы с двумя входами

наследник от **TLogElement**

```
class TLog2In ( TLogElement ):
  pass
```

? Можно ли создать объект этого класса?

нельзя, он абстрактный

# Элементы с двумя входами

## Элемент «И»:

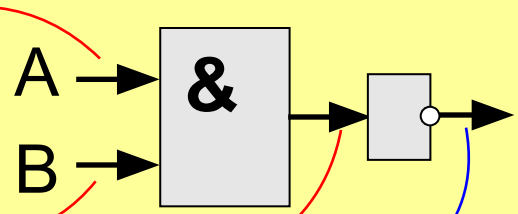
```
class TAnd ( TLog2In ):
  def __init__ ( self ):
    TLog2In.__init__ ( self )
  def calc ( self ):
    self._res = self.In1 and self.In2
```

## Элемент «ИЛИ»:

```
class TOr ( TLog2In ):
  def __init__ ( self ):
    TLog2In.__init__ ( self )
  def calc ( self ):
    self._res = self.In1 or self.In2
```

# Пример: элемент «И-НЕ»

```
elNot = TNot()
elAnd = TAnd()
print ( "  A | B | not(A&B) " );
print ( "--------------------" );
for A in range(2):
  elAnd.In1 = bool(A)
  for B in range(2):
    elAnd.In2 = bool(B)
    elNot.In1 = elAnd.Res
    print ( " ", A, "|", B, "|",
                        int(elNot.Res) )
```



# Модульность

*Идея*: выделить классы в отдельный модуль **logelement.py**.

```
class TLogElement:
   ...
class TNot ( TlogElement ):
   ...
class TLog2In ( TLogElement ):
  pass
class TAnd ( TLog2In ):
   ...
class TOr ( TLog2In ):
   ...
```

# Модульность

## В основную программу:

```
import logelement
elNot=logelement.TNot()
elAnd=logelement.TAnd()
...
```

# Сообщения между объектами

Задача – автоматическая передача сигналов по цепочке!

```
class TLogElement:
  def __init__ ( self ):
    ...
    self.__nextEl = None
    self.__nextIn = 0
  ...
  def link ( self, nextEl, nextIn ):
    self.__nextEl = nextEl
    self.__nextIn = nextIn
```

адрес следующего элемента в цепочке

номер входа следующего элемента

установка связи

# Сообщения между объектами

**После изменения выхода «дергаем» следующий элемент**:

```
class TLogElement:
  ...
  def __setIn1 ( self, newIn1 ):
    self.__in1 = newIn1
    self.calc()
    if self.__nextEl:
      if self.__nextIn == 1:
        self.__nextEl.In1 = self._res
      elif __nextIn == 2:
        __nextEl.In2 = self._res
```

если следующий элемент установлен…

передать результат на нужный вход

# Сообщения между объектами

## Изменения в основной программе:

```
elNot = TNot()
elAnd = TAnd()
elAnd.link ( elNot, 1 )
print ( "  A | B | not(A&B) " );
print ( "--------------------" );
for A in range(2):
  elAnd.In1 = bool(A)
  for B in range(2):
    elAnd.In2 = bool(B)
    elNot.In1 = elAnd.Res
    print ( " ", A, "|", B, "|",
                        int(elNot.Res) )
```

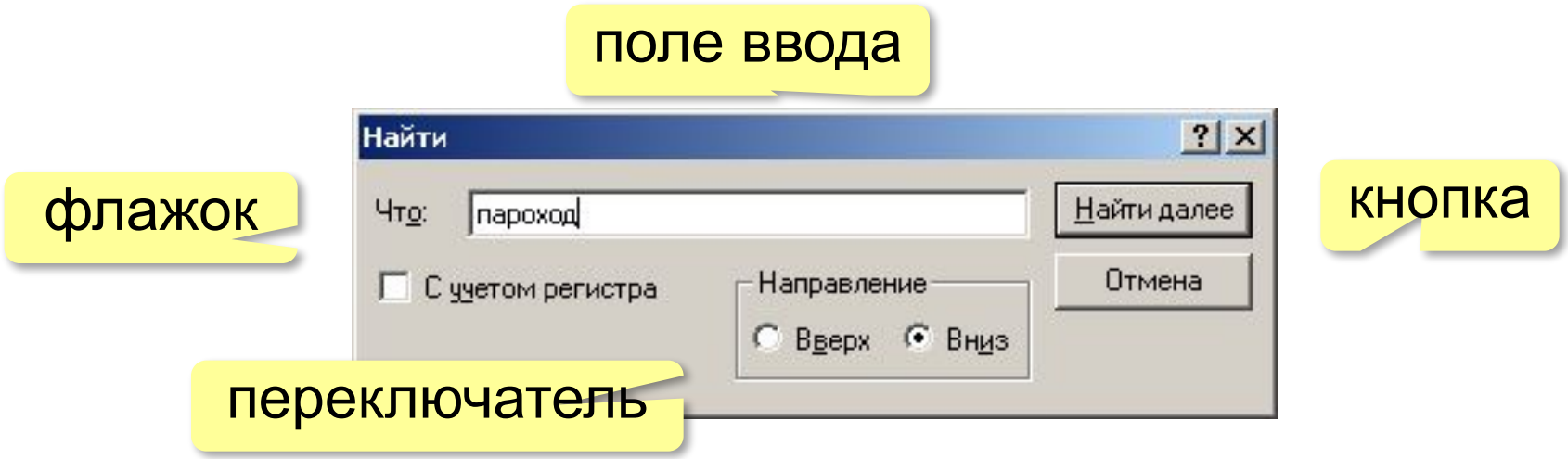
установить связь

это уже не нужно!

# Объектно-ориентированное программирование. Язык Python

## § 51. Программы с графическим интерфейсом

# Интерфейс: объекты и сообщения

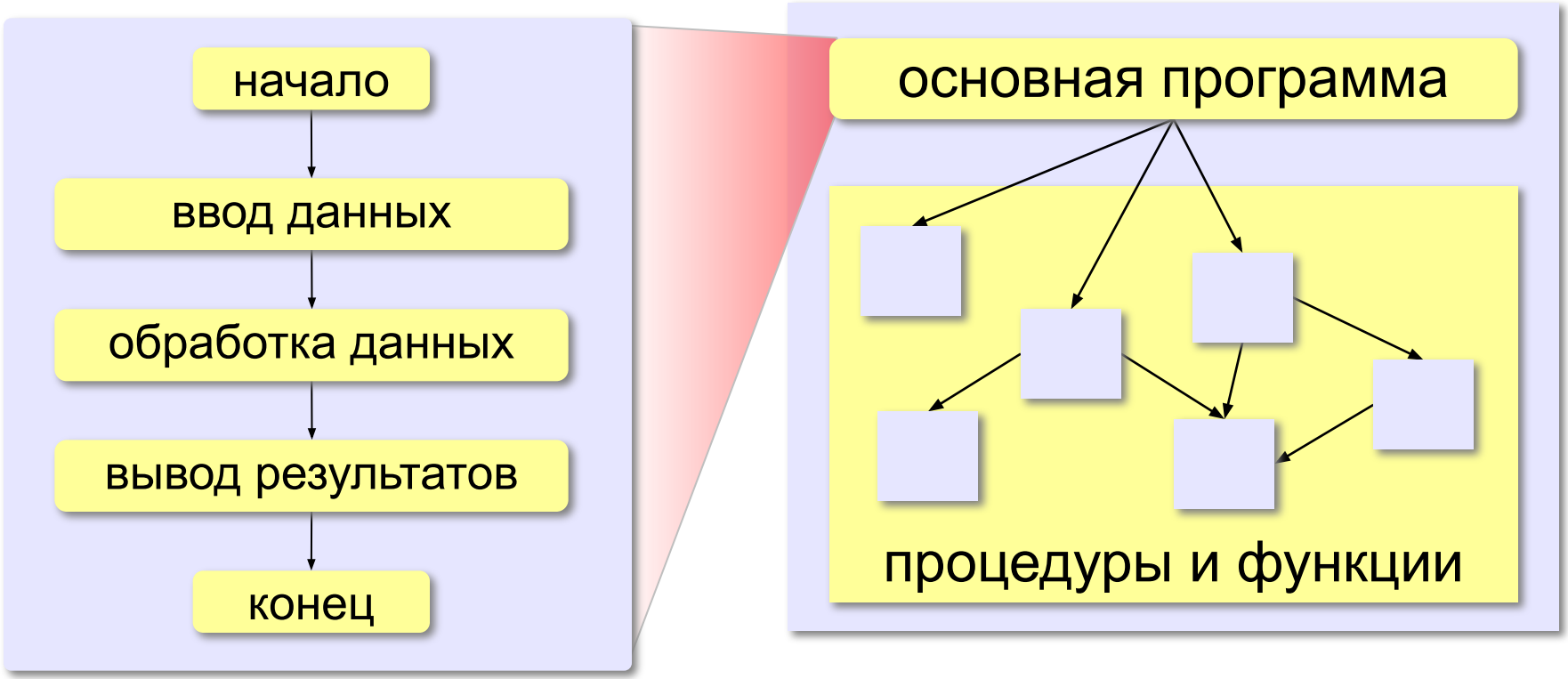



Все элементы окон – объекты, которые обмениваются данными, посылая друг другу сообщения.

**Сообщение** – это блок данных определённой структуры, который используется для обмена информацией между объектами.

- адресат (кому) или *широковещательное*
- числовой код (тип) сообщения
- параметры (дополнительные данные)

# Классические программы




Порядок выполнения команд определяется программистом, пользователь не может вмешаться!

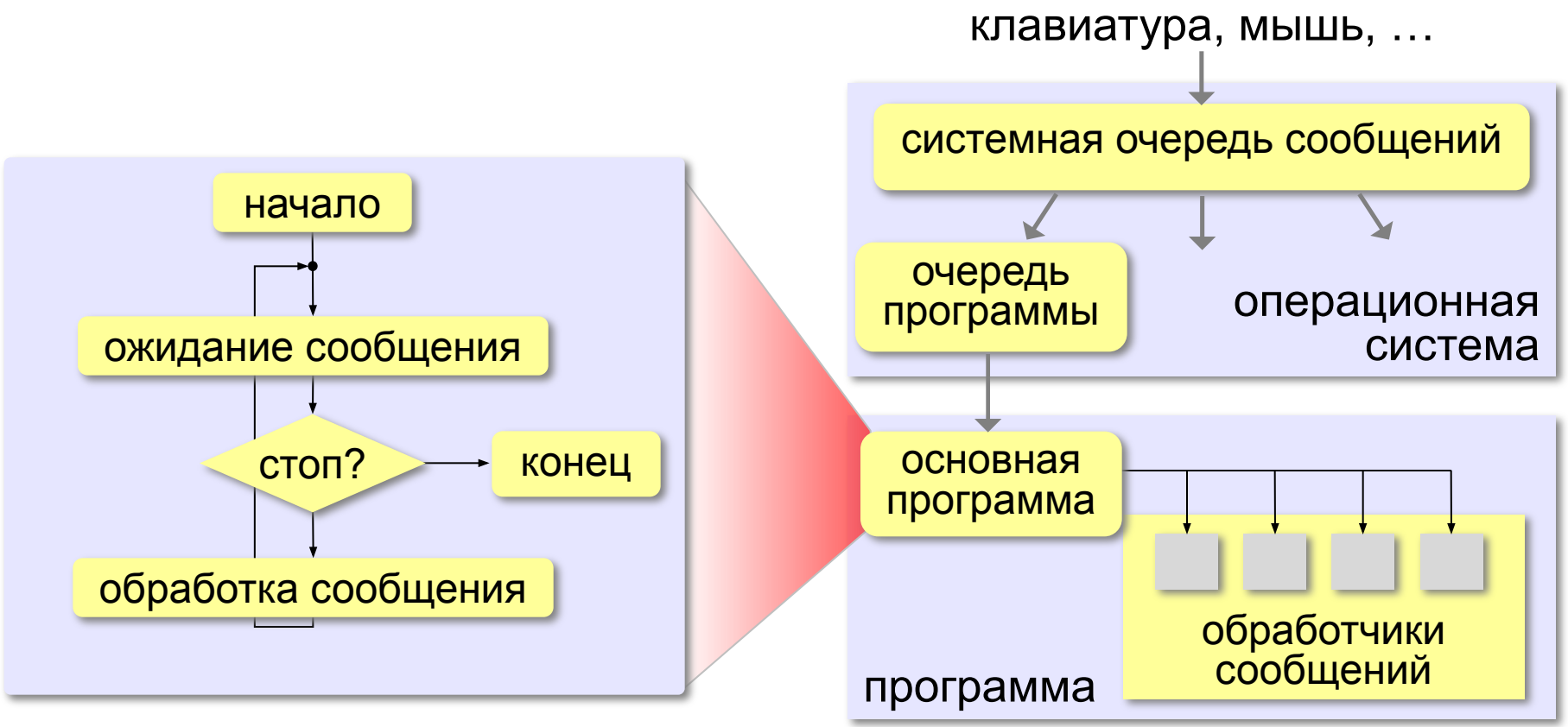
# Программы, управляемые событиями

**Событие** – это переход какого-либо объекта из одного состояния в другое.

- нажатие на клавишу
- щелчок мышью
- перемещение окна
- поступление данных из сети
- запрос к веб-серверу
- завершение вычислений
- …

Программа начинает работать при наступлении событий!

# Программы, управляемые событиями




! Программа управляется событиями!

# Что такое RAD-среда?

**RAD** = *Rapid Application Development* — быстрая разработка приложений

**Форма** – это шаблон, по которому строится окно программы или диалога

**Этапы разработки:**

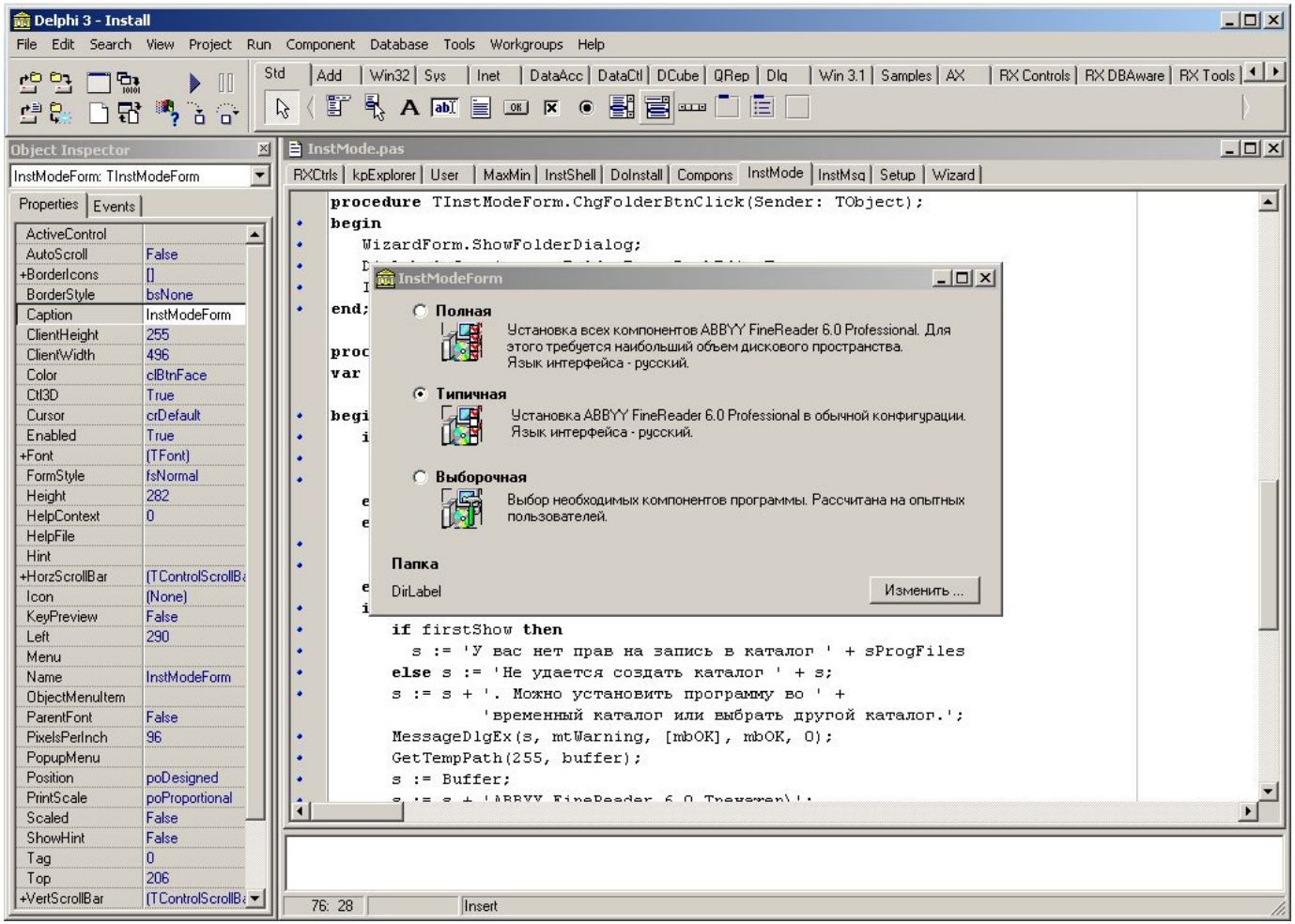
- создание **формы**
- минимальный код добавляется автоматически
- расстановка **элементов интерфейса** с помощью мыши и настройка их свойств
- создание **обработчиков** событий
- написание **алгоритмов** обработки данных

выполняются при возникновении событий

# RAD-среды: Delphi

**Язык:** *Object Pascal*, позднее *Delphi*:

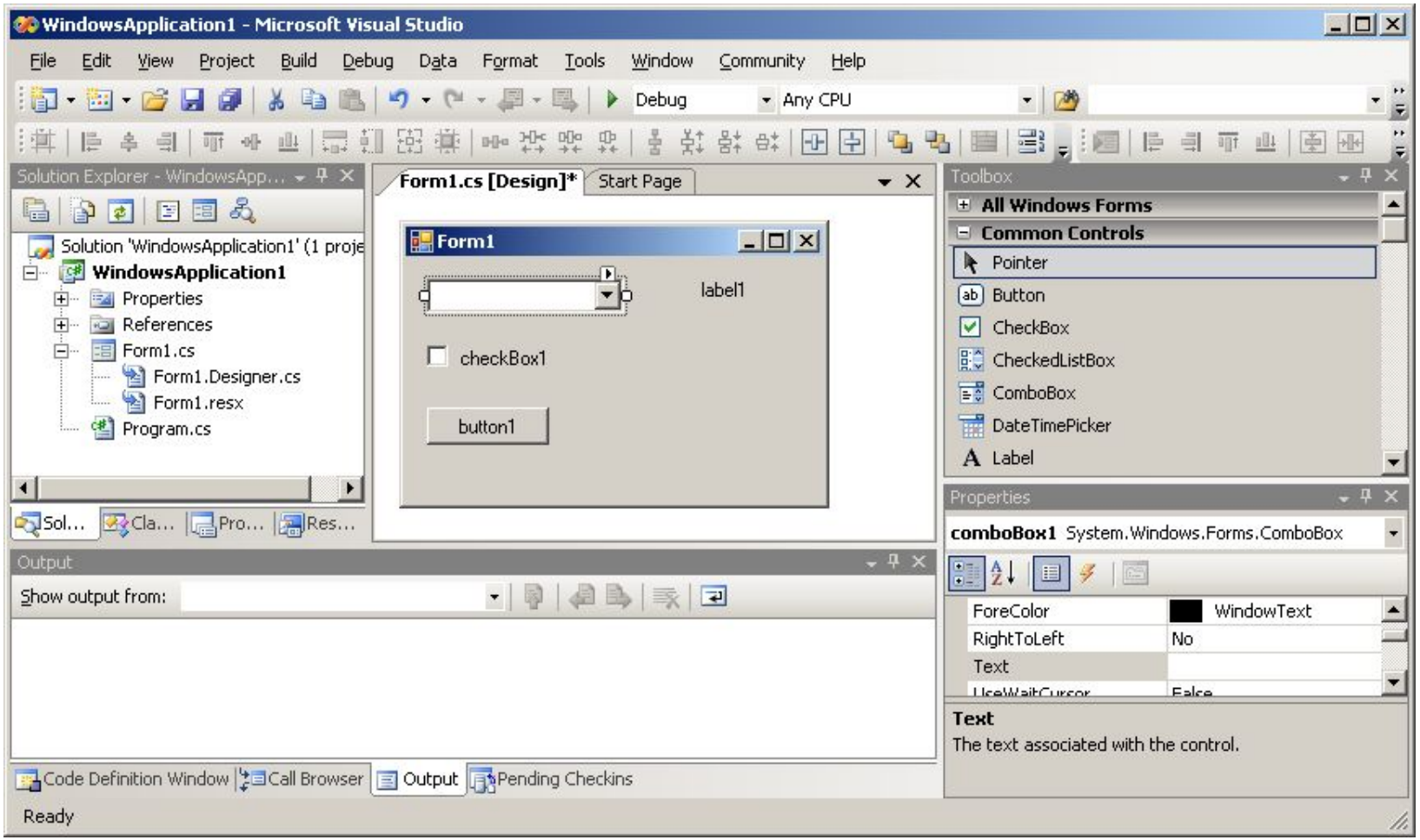
1995: *Borland*, сейчас: *Embarcadero Technologies*



# RAD-среды: MS Visual Studio

**Языки:** *Visual Basic, Visual C++, Visual C#, Visual F#*

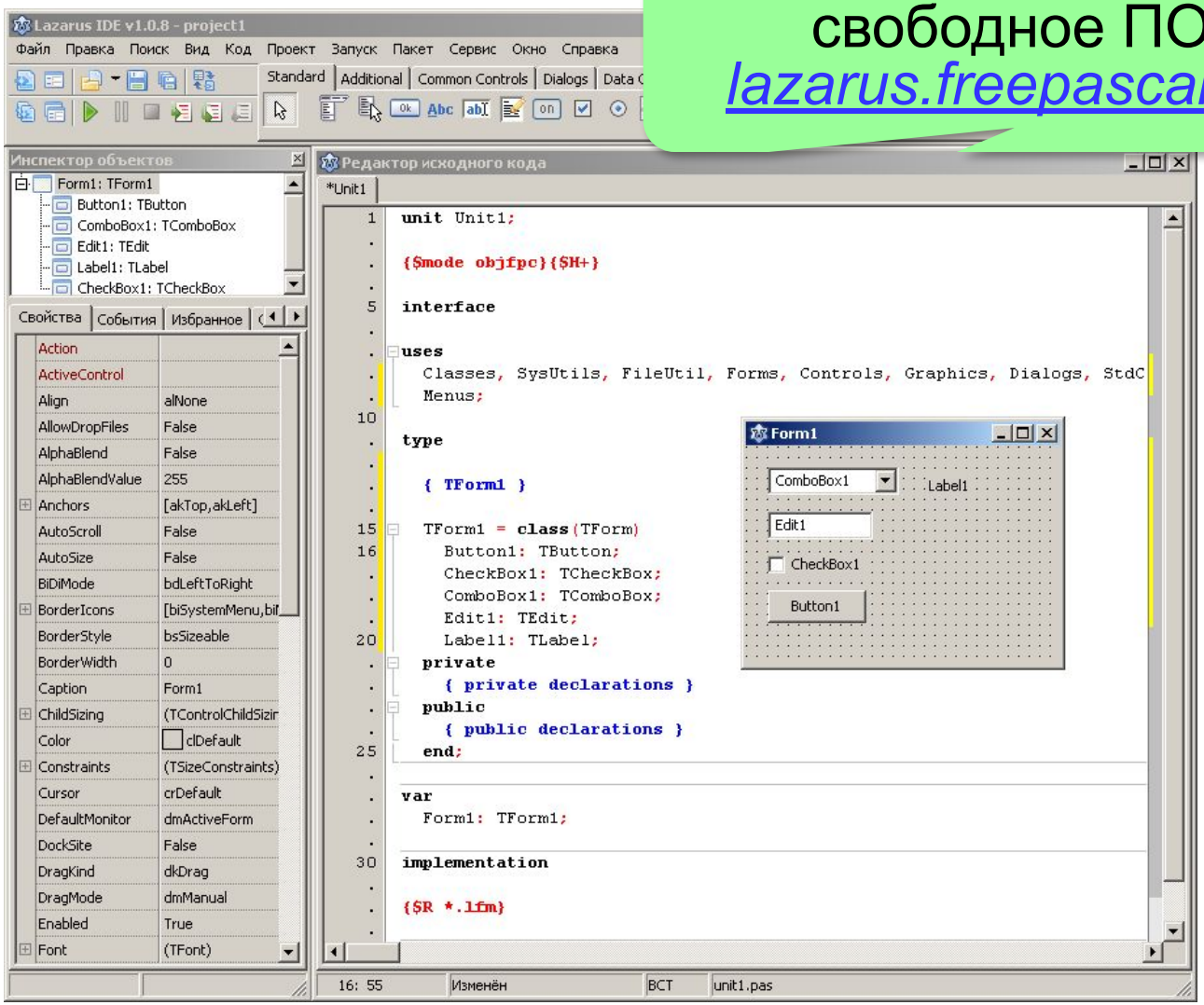
с 1995 по н.в.: *Microsoft*



# RAD-среды: Lazarus

## Языки: *FreePascal, Delphi*

свободное ПО:
lazarus.freepascal.org



# Объектно-ориентированное программирование. Язык Python

## § 52. Графический интерфейс: основы

# Графические библиотеки для Python

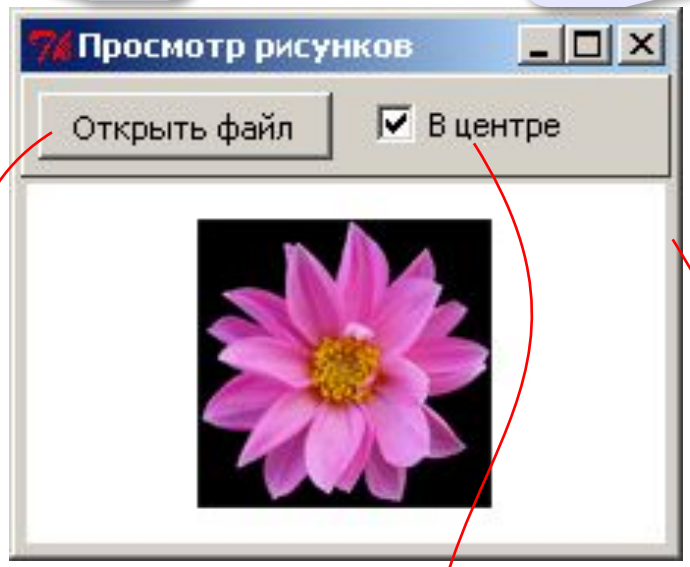
- *tkinter* (стандартная библиотека Python )
- *wxPython* (http://wxpython.org)
- *PyGTK* (http://pygtk.org)
- *PyQt* (http://www.riverbankcomputing.com/software/pyqt/intro)

*simpletk* – «обёртка» над *tkinter* (http://kpolyakov.spb.ru/school/probook/python.htm)

# Общие принципы

**компонент** (*виджет*, элемент)

**форма** (окно верхнего уровня)



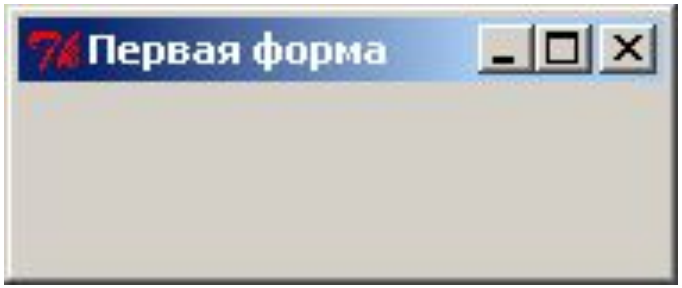

щелчок по выключателю

щелчок по кнопке

изменение размеров

обработчики событий

# Простейшая программа




импорт всех функций из **simpletk**

```
from simpletk import *
app=TApplication("Первая форма")
app.Run()
```

объект-приложение (программа)

заголовок окна

запуск программы

# Свойства формы

```
app = TApplication("Первая форма")
```

x y

```
app.position = (100, 300)
```

начальные координаты

ширина высота

```
app.size = (500, 200)
```

по ширине по высоте

можно ли менять размеры

```
app.resizable = (True, False)
```

по ширине по высоте

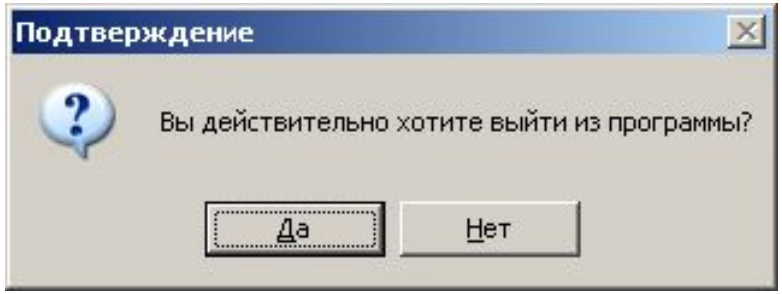
минимальные размеры

```
app.minsize = (100, 200)
app.maxsize = (900, 700)
```

# Обработчик события

событие

*Задача.* Запросить подтверждение при закрытии окна.




Обработчик события – это функция!

```
from tkinter.messagebox import askokcancel
```

```
def AskOnExit():
    if askokcancel ( "Подтверждение",
"Вы действительно хотите выйти из программы?" ):
        app.destroy()
```

удалить из памяти

**Привязка обработчика**:

```
app.onCloseQuery = AskOnExit
```

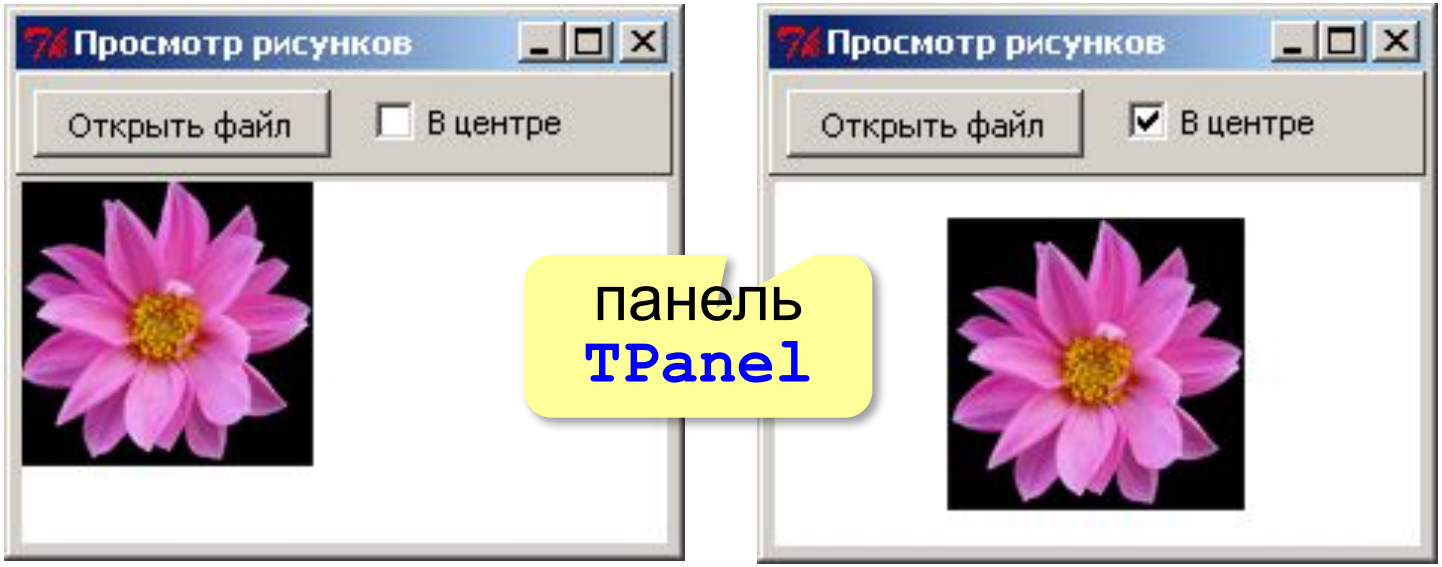
# Объектно-ориентированное программирование. Язык Python

## § 53. Использование компонентов

# Просмотр рисунков

кнопка **TButton**

выключатель **TCheckBox**



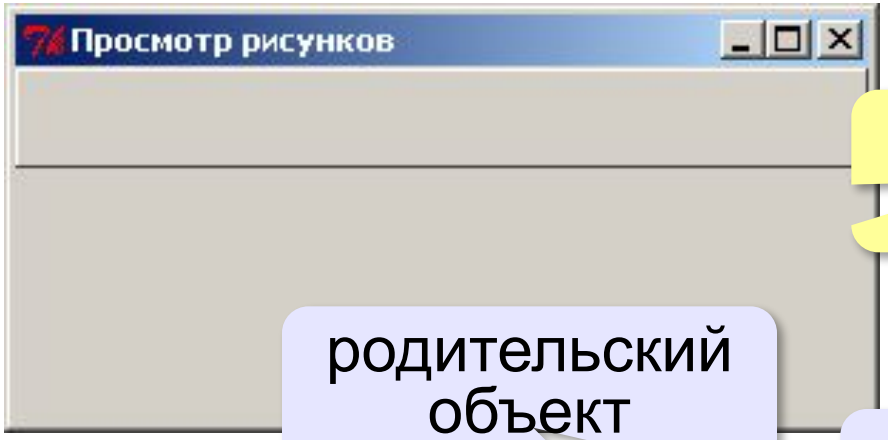

панель **TPanel**

рисунок **TImage**

# Настройка формы

```
from simpletk import *
app = TApplication( "Просмотр рисунков" )
app.position = (200, 200)
app.size = (300, 300)
# сюда будем добавлять компоненты!
app.Run()
```

# Верхняя панель




панель **TPanel**

родительский объект

рельеф - приподнятый

```
panel = TPanel ( app,
                 relief = "raised",
                 height = 35,
                 bd = 1 )
```

ширина рамки

высота

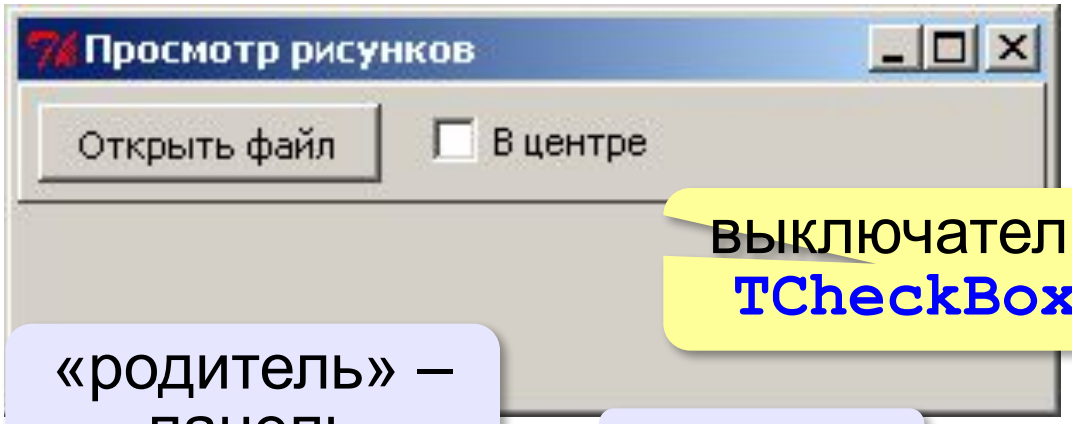
```
panel.align = "top"
```

прижать к верхней границе

выравнивание

# Кнопка и выключатель

кнопка **TButton**




выключатель **TCheckBox**

«родитель» – панель

ширина

```
openBtn = TButton ( panel, width = 15,
                    text = "Открыть файл" )
openBtn.position = (5, 5)
```

координаты

```
centerCb = TCheckBox ( panel,
                       text = "В центре" )
centerCb.position = (115, 5)
```

# Поле для рисунка

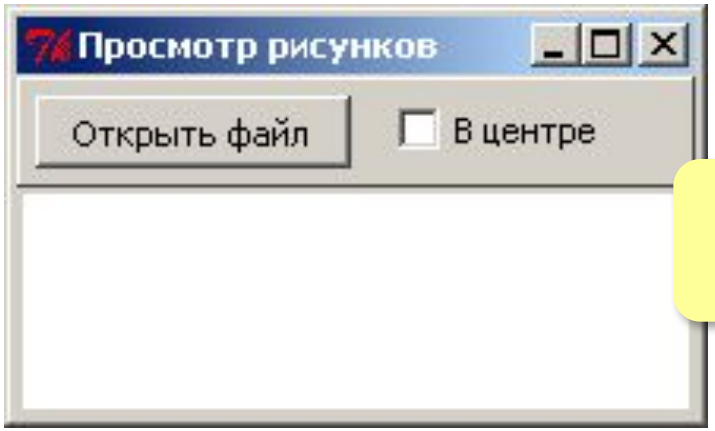



рисунок **TImage**

«родитель» – главное окно

фон – белый

```
image = TImage ( app, bg = "white" )
image.align = "client"
```

заполнить все свободное место

# Выбор файла

## После щелчка по кнопке:

```
выбрать файл с рисунком
if файл выбран:
   загрузить рисунок в компонент image
```

## Выбор файла:

```
from tkinter import filedialog
fname = filedialog.askopenfilename (
     filetypes = [ ("Файлы GIF", "*.gif"),
                   ("Все файлы", "*.*") ] )
```

## Загрузка рисунка:

если имя файла не пустое

```
if fname:
   image.picture = fname
```

# Выбор файла

**Обработчик щелчка по кнопке**:

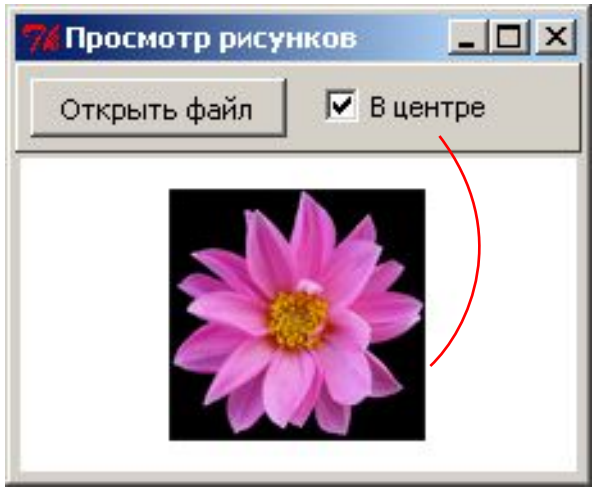
```
from tkinter import filedialog


def selectFile( sender ):
  fname = filedialog.askopenfilename(
     filetypes = [ ("Файлы GIF", "*.gif"),
                   ("Все файлы", "*.*")] )
  if fname:
    image.picture = fname
```

объект-источник события

**Привязка обработчика**:

```
openBtn.onClick = selectFile
```

# Центрирование




объект-источник события

**Обработчик**:

```
def cbChanged( sender ):
    image.center = sender.checked
    image.redrawImage()
```

перерисовать рисунок

включен (True/False)?

**Привязка обработчика**:

```
centerCb.onChange = cbChanged
```

обработчик события «изменение состояния»

- программа на основе ООП
- использование компонентов скрывает сложность

# Новый класс – «всё в одном»

Идея: убрать все действия в новый класс!

**Основная программа**:

```
class TImageViewer ( TApplication ):
    ...

app = TImageViewer()
app.Run()
```

# Класс TImageViewer: конструктор

```
class TImageViewer ( TApplication ):
 def __init__(self):
    TApplication.__init__ ( self, "Просмотр рисунков" )
    self.position = (200, 200)
    self.size = (300, 300)
    self.panel = TPanel(self, relief = "raised",
                                height = 35, bd = 1)
    self.panel.align = "top"
    self.image = TImage ( self, bg = "white" )
    self.image.align = "client"
    self.openBtn = TButton ( self.panel,
                 width = 15, text = "Открыть файл" )
    self.openBtn.position = (5, 5)
    self.openBtn.onClick = self.selectFile
    self.centerCb = TCheckBox ( self.panel,
                                 text = "В центре" )
    self.centerCb.position = (115, 5)
    self.centerCb.onChange = self.cbChanged
```

**self.** сохраняем всё в полях объекта **TImageViewer**

# Класс TImageViewer: обработчики

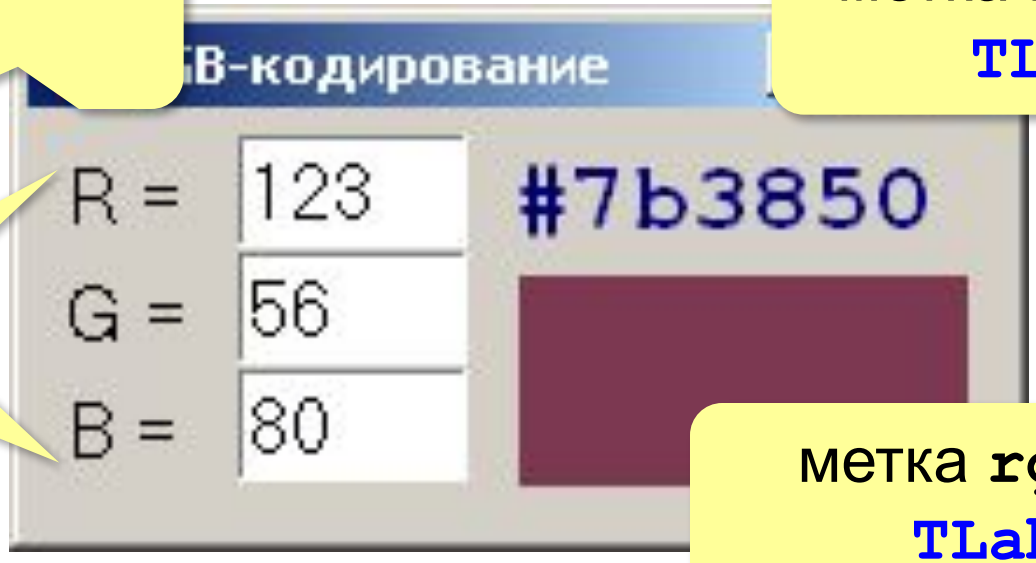
```
class TImageViewer ( TApplication ):
 def __init__(self):
    ...
 def selectFile ( self, sender ):
   fname = filedialog.askopenfilename(
      filetypes = [ ("Файлы GIF", "*.gif"),
                    ("Все файлы", "*.*")] )
   if fname:
     self.image.picture = fname

 def cbChanged ( self, sender ):
   self.image.center = sender.checked
   self.image.redrawImage()
```

# Ввод и вывод данных

для веб-страниц

поле ввода **rEdit**
**TEdit**

метка **rgbLabel**
**TLabel**




метки
**TLabel**

метка **rgbRect**
**TLabel**

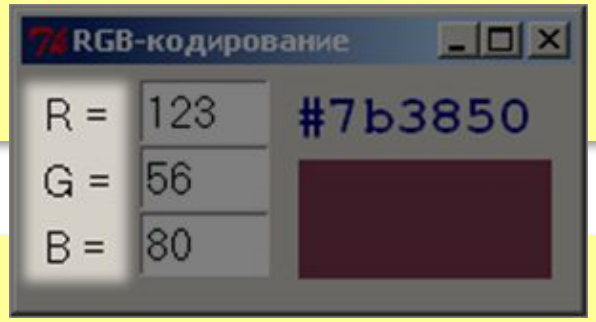
поле ввода **bEdit**
**TEdit**

поле ввода **gEdit**
**TEdit**

# Основная программа

**Объект-приложение**:

```
app = TApplication ( "RGB-кодирование" )
app.size = (210, 90)
app.position = (200, 200)
```

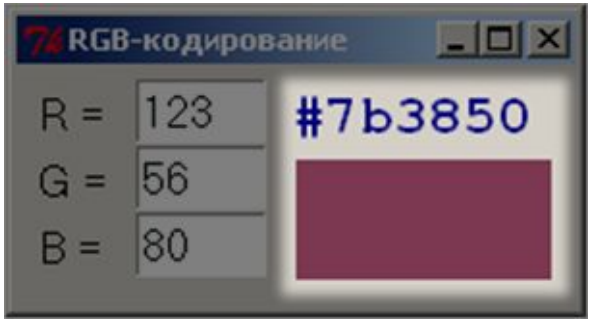



**Метки RGB**:

шрифт

```
f = ( "MS Sans Serif", 12 )

lblR = TLabel ( app, text = "R = ", font = f )
lblR.position = (5, 5)

lblG = TLabel ( app, text = "G = ", font = f )
lblG.position = (5, 30)

lblB = TLabel ( app, text = "B = ", font = f )
lblB.position = (5, 55)
```

# Компоненты




rgbLabel

rgbRect

**Метки для вывода результата**:

шрифт

```
fc = ( "Courier New", 16, "bold" )

rgbLabel = TLabel ( app, text = "#000000",
                    font = fc, fg = "navy" )
rgbLabel.position = (100, 5)

rgbRect = TLabel ( app, text = "",
                   width = 15, height = 3 )
rgbRect.position = (105, 35)
```

цвет текста

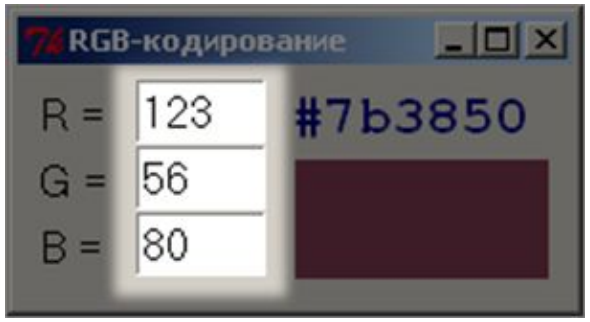
ширина и высота в **символах**!

# Компоненты

rEdit

gEdit

bEdit




**Поля ввода**:

шрифт тот же, что и для меток

```
rEdit = TEdit ( app, font = f, width = 5 )
rEdit.position = (45, 5)
rEdit.text = "123"
```

ширина в **символах**!

остальные – аналогично…

# Обработчик события «изменение поля»

объект-источник события

```
def onChange ( sender ):

   r = int ( rEdit.text )
   g = int ( gEdit.text )
   b = int ( bEdit.text )
   s = "#{:02x}{:02x}{:02x}".format(r, g, b)

   rgbRect.background = s

   rgbLabel.text = s
```

преобразовать строки в числа

шестнадцатеричный код

изменить фон

изменить текст метки

# Запуск программы

**Подключение обработчиков**:

```
rEdit.onChange = onChange
gEdit.onChange = onChange
bEdit.onChange = onChange
```

После того, как все поля будут созданы!

**Запуск программы**:

```
app.Run()
```

# Обработка ошибок

Если вместо числа ввести букву?

```
Exception in Tkinter callback
Traceback (most recent call last):
… line 48, in onChange
ValueError: invalid literal for int() with base
10: '12w'
```

неверные данные для функции `int`

Программа не должна «вылетать»!

# Обработка ошибок

попытаться выполнить

```
try:
   # «опасные» команды
except:
   # обработка ошибки
```

если **исключение** (аварийная ситуация)

? Какие у нас опасные операции?

# Обработка ошибок

```
def onChange ( sender ):
  s = "?"    # текст метки
  bkColor = "SystemButtonFace"
  try:
    # получить новый цвет из полей ввода
  except:
    pass
  rgbLabel.text = s
  rgbRect.background = bkColor
```

цвет прямоугольника

# Обработка ошибок

```
def onChange ( sender ):
 s = "?"
 bkColor = "SystemButtonFace"
 try:
  r = int ( rEdit.text )
  g = int ( gEdit.text )
  b = int ( bEdit.text )
  if r in range(256) and \
     g in range(256) and b in range(256):
   s = "#{:02x}{:02x}{:02x}".format(r, g, b)
   bkColor = s
 except:
   pass
 rgbLabel.text = s
 rgbRect.background = bkColor
```

# Объектно-ориентированное программирование. Язык Python

## § 54. Совершенствование компонентов

# Новый класс для ввода целого числа

*Задача*: построить поле для ввода целых чисел, в котором

- есть защита от ввода неверных символов
- есть методы для чтения/записи целого числа

На основе класса **`TEdit`**!

```
class TIntEdit ( TEdit ):
    ...
```

**Изменения**:

- автоматическая блокировка недопустимых символов (всех, кроме цифр)
- свойство **`value`** – значение (целое число)

# Добавление свойства

```
class TIntEdit ( TEdit ):

  def __init__ ( self, parent, **kw ):
    TEdit.__init__ ( self, parent, **kw )
    self.__value = 0


  def __setValue ( self, value ):
    self.text = str ( value )

  value = property ( lambda x: x.__value,
                     __setValue )
```

объект-«родитель»

остальные параметры (словарь)

поле хранит целое значение

# Проверка символов

**onValidate – обработчик события «проверка данных»**

```
class TIntEdit ( TEdit ):
  def __init__ ( self, parent, **kw ):
    ...
    self.onValidate = self.__validate

  def __validate ( self ):
    try:
      newValue = int ( self.text )
      self.__value = newValue
      return True
    except:
      return False
```

установить обработчик

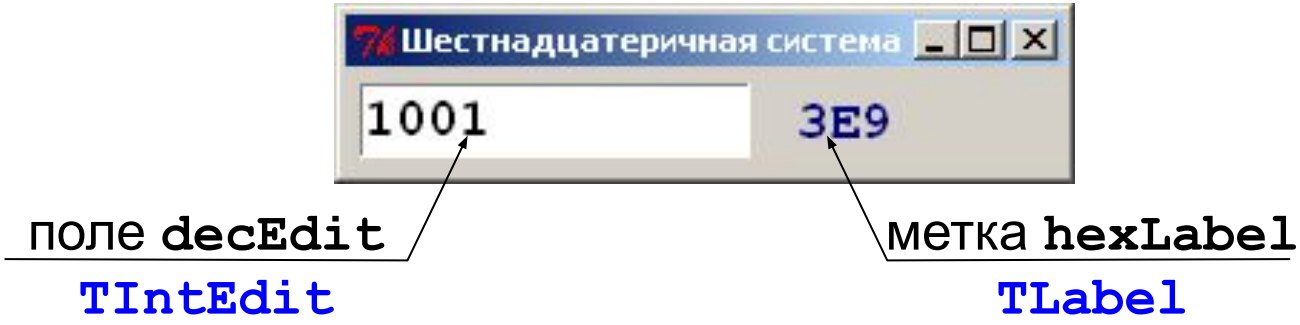
пытаемся получить целое

если удачно, запомнили

неудачно, отказаться от изменений

! В модуль **int_edit.py**!

# Поле для ввода целых чисел




**Объект-приложение**:

```
app = TApplication ( "Шестнадцатеричная система" )
app.size = (250, 36)
app.position = (200, 200)
```

**Метка**:

шрифт

```
f = ( "Courier New", 14, "bold" )
hexLabel = TLabel ( app, text = "?",
                    font = f, fg = "navy" )
hexLabel.position = (155, 5)
```

цвет текста

# Поле для ввода целых чисел

**Поле ввода**:

```
from int_edit import TIntEdit
decEdit = TIntEdit ( app, width = 12, font = f )
decEdit.position = (5, 5)
decEdit.text = "1001"
```

шрифт

ширина в символах

**Обработчик события**:

```
def onNumChange ( sender ):
  hexLabel.text = "{:X}".format (
                                sender.value )
decEdit.onChange = onNumChange
```

в шестнадцатеричную систему

установить обработчик

**Запуск**:

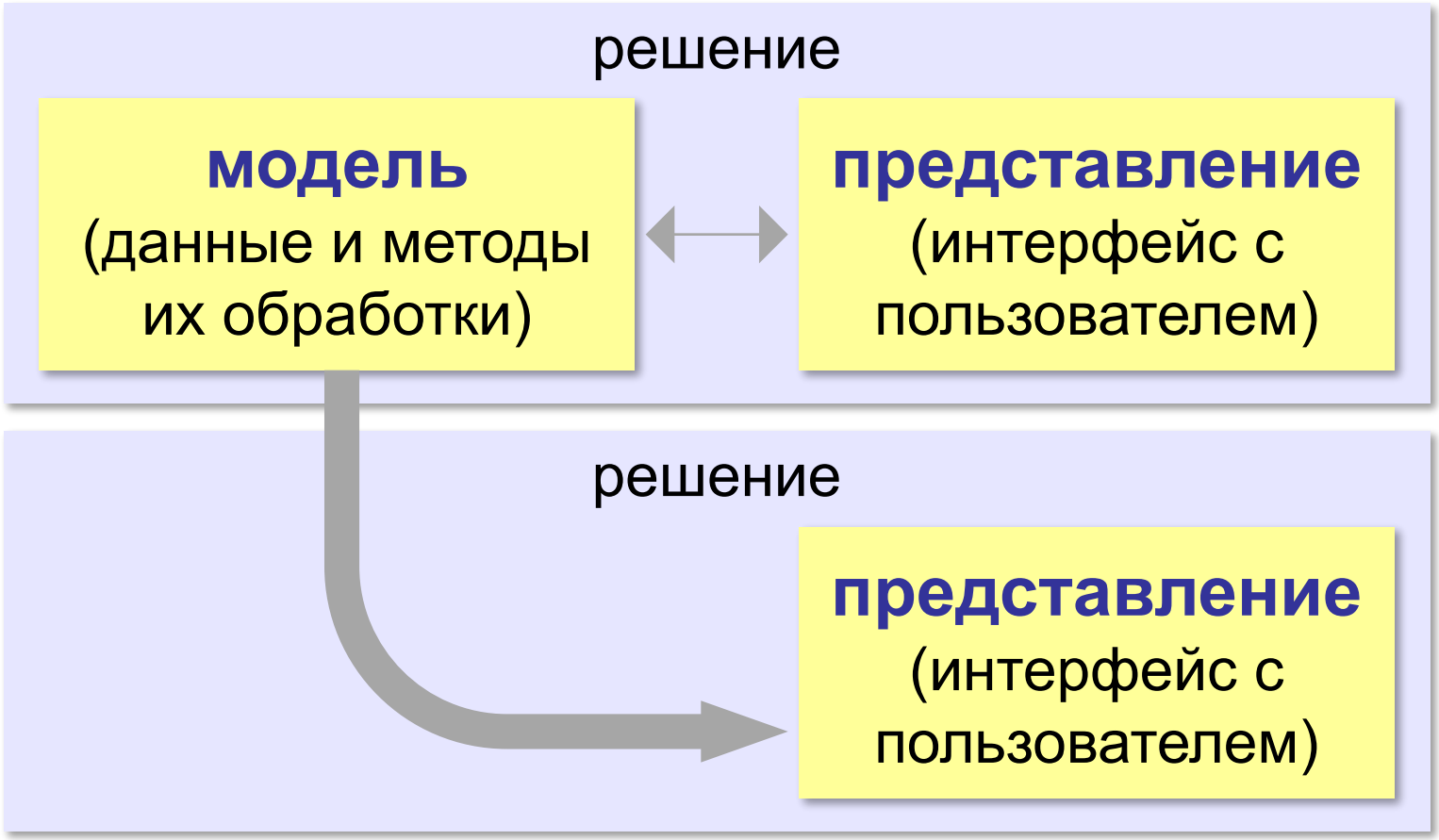
```
app.Run()
```

# Объектно-ориентированное программирование. Язык Python

## § 55. Модель и представление

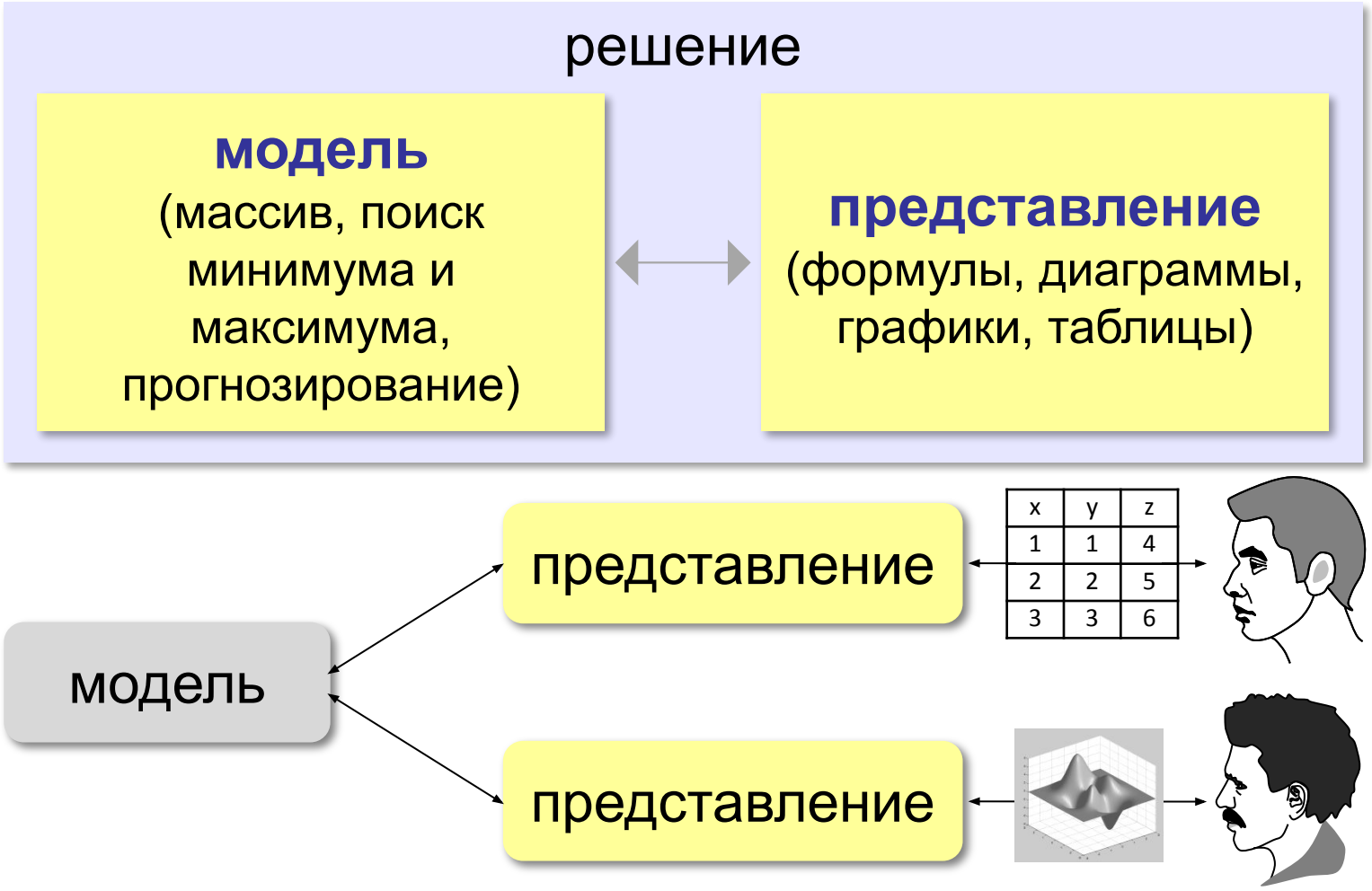
# Еще одна декомпозиция

*Задача*: повторное использование написанного ранее готового кода.



# Модель и представление

*Задача*: хранить и использовать данные об изменении курса доллара.



# Модель и представление

*Задача*: вычисление арифметического выражения:

- целые числа
- знаки арифметических действий `+ - * /`

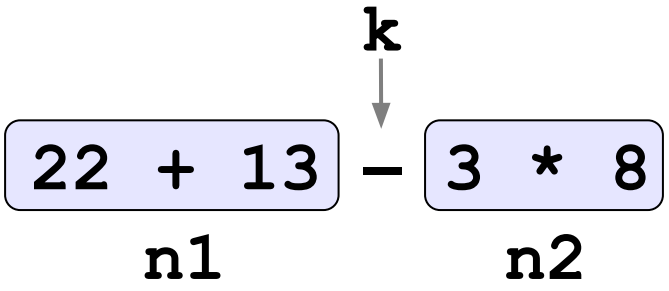
**Модель**:

- символьная строка
- алгоритм вычисления:

функция `lastOp` (глава 6)

```
k = номер последней операции
n1 = значение левой части
n2 = значение правой части
результат = операция(n1, n2)
```

! Рекурсия!




? Чего не хватает?

# Модель

**Псевдокод**:

```
k = номер последней операции
if k < 0:
   результат = строка в число
else:
   n1 = значение левой части
   n2 = значение правой части
   результат = операция(n1, n2)
```

# Модель: вычисления

```
def Calc ( s ):
  k = lastOp ( s )
  if k < 0:              # вся строка - число
    return int(s)
  else:
    n1 = Calc ( s[:k] )    # левая часть
    n2 = Calc ( s[k+1:] ) # правая часть
     # выполнить операцию
    if   s[k] == "+": return n1+n2
    elif s[k] == "-": return n1-n2
    elif s[k] == "*": return n1*n2
    else: return n1 // n2
```

# Вспомогательные функции

## Приоритет операции:

```
def priority ( op ):
  if op in "+-": return 1
  if op in "*/": return 2
  return 100
```

## Модуль:

```
model.py:
  Calc
  priority
  lastOp
```

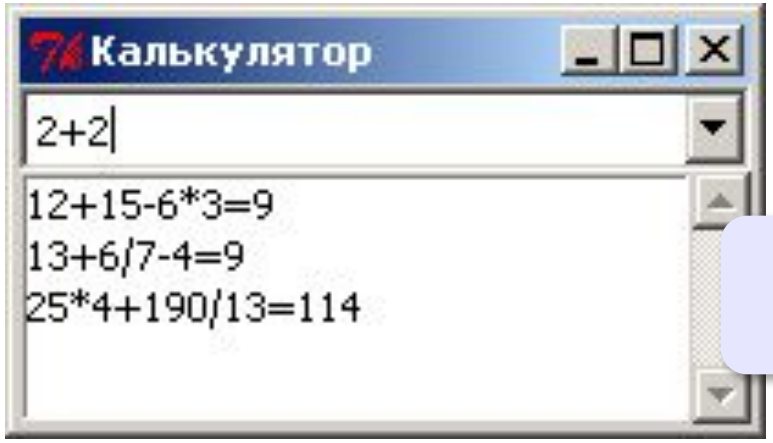
## Номер последней операции:

```
def lastOp ( s ):
  minPrt = 50    # любое между 2 и 100
  k = -1
  for i in range(len(s)):
    if priority(s[i]) <= minPrt:
      minPrt = priority(s[i])
      k = i
  return k
```

? Почему <=?

# Представление

выпадающий список **TComboBox**




список **TListBox**

## Объект-приложение:

```
app = TApplication ( "Калькулятор" )
app.size = (200, 150)
...
app.Run()
```

# Компоненты

## Выпадающий список:

список значений

```
Input = TComboBox ( app, values = [],
                    height = 1 )
Input.align = "top"
Input.text = "2+2"
```

высота

прижать к верху

текст

## Список для запоминания результатов:

```
Answers = TListBox ( app )
Answers.align = "client"
```

заполнить все свободное место

# Логика работы

```
if нажата клавиша Enter:
   вычислить выражение
   добавить результат в начало списка
   if выражения нет в выпадающем списке:
      добавить его в выпадающий список
```

## Обработчик нажатия Enter:

```
def doCalc ( event ):
   ...
```

## Установка обработчика:

```
Input.bind ( "<Key-Return>", doCalc )
```

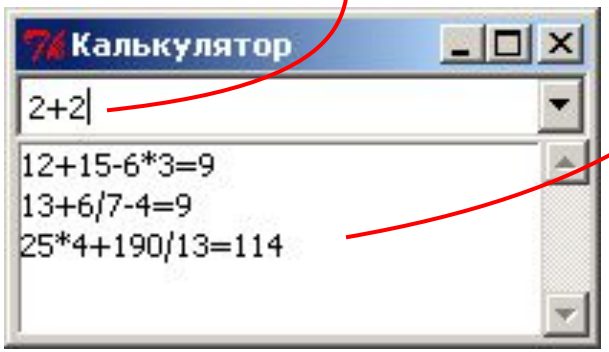
«связать»

клавиша Enter

# Обработчик нажатия на клавишу Enter

```
from model import Calc
def doCalc ( event ):
    expr = Input.text   # прочитать выражение
    x = Calc ( expr )   # вычислить
    Answers.insert ( 0, expr + "=" + str(x) )
    if not Input.findItem ( expr ):
        Input.addItem ( expr )
```

если еще нет в списке



# Калькулятор


! Самостоятельно!

# Конец фильма

**ПОЛЯКОВ Константин Юрьевич**
д.т.н., учитель информатики
ГБОУ СОШ № 163, г. Санкт-Петербург
kpolyakov@mail.ru

**ЕРЕМИН Евгений Александрович**
к.ф.-м.н., доцент кафедры мультимедийной дидактики и ИТО ПГГПУ, г. Пермь
eremin@pspu.ac.ru

# Источники иллюстраций

1. www.picstopin.com
2. maugav.info
3. yoursourceisopen.com
4. ru.wikipedia.org
5. иллюстрации художников издательства «Бином»
6. авторские материалы