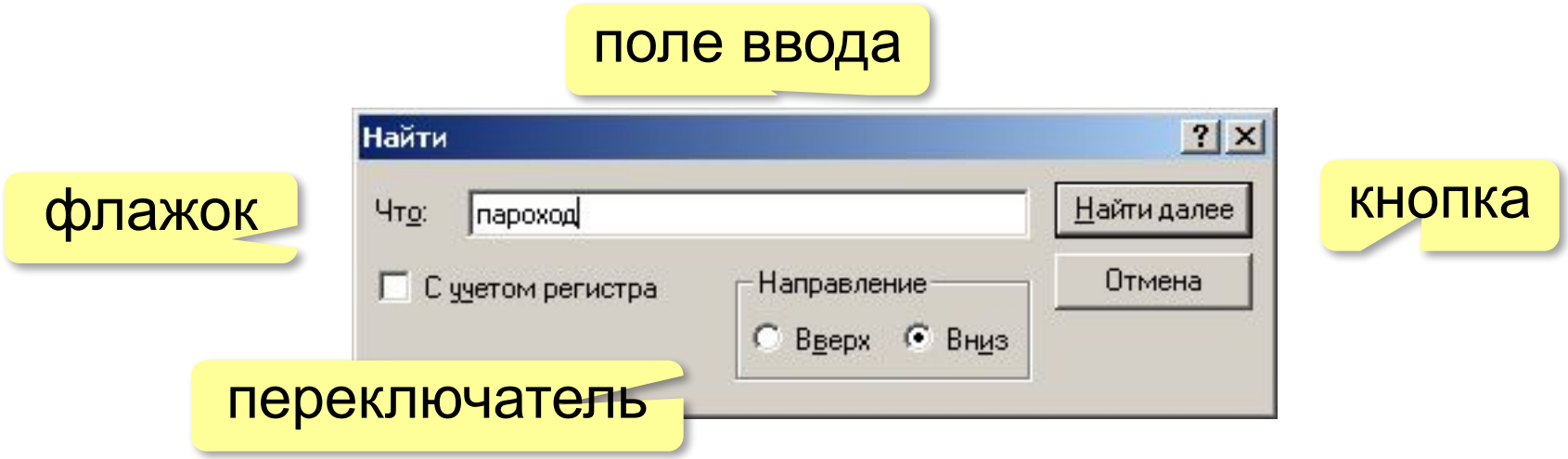
Объектно-ориентированное программирование. Язык Python

§ 46. Программы с графическим интерфейсом

Интерфейс: объекты и сообщения

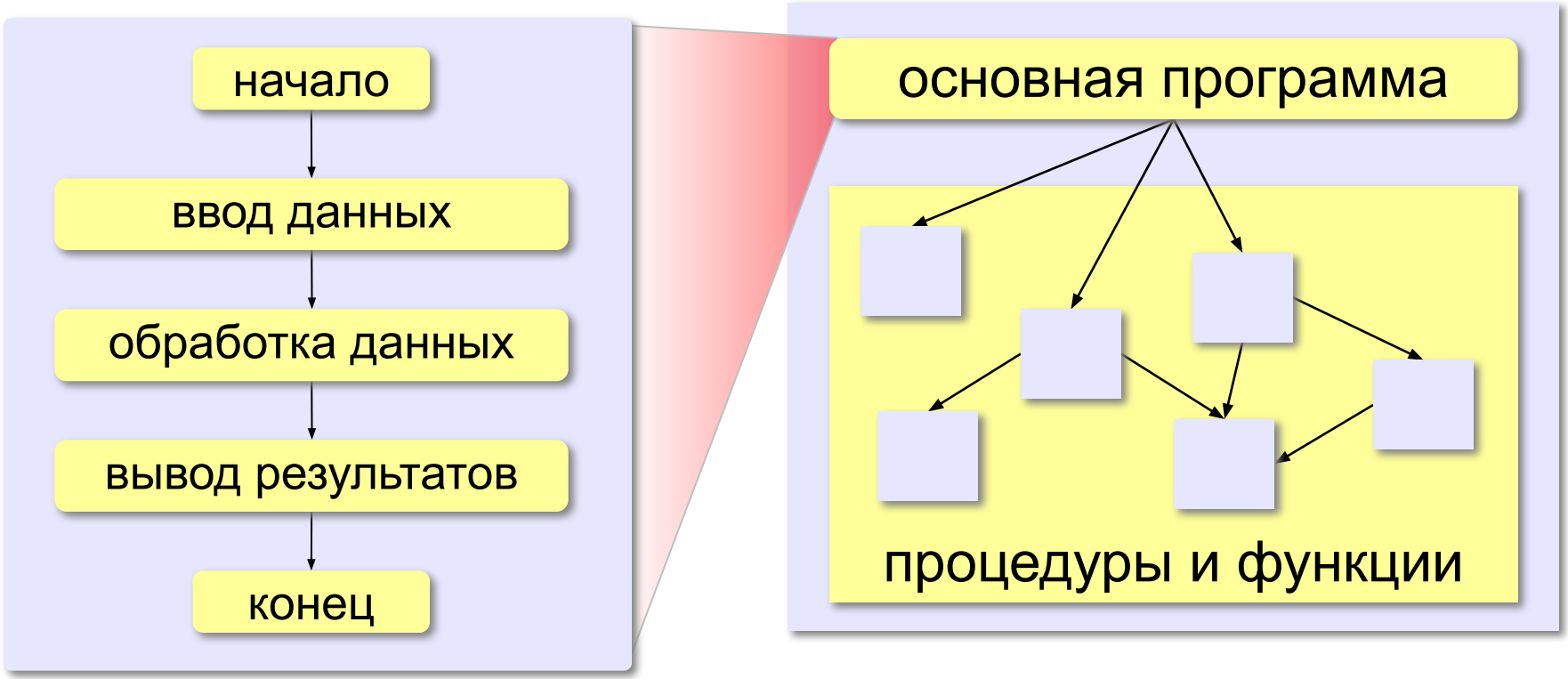


Все элементы окон – объекты, которые обмениваются данными, посылая друг другу сообщения.

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!

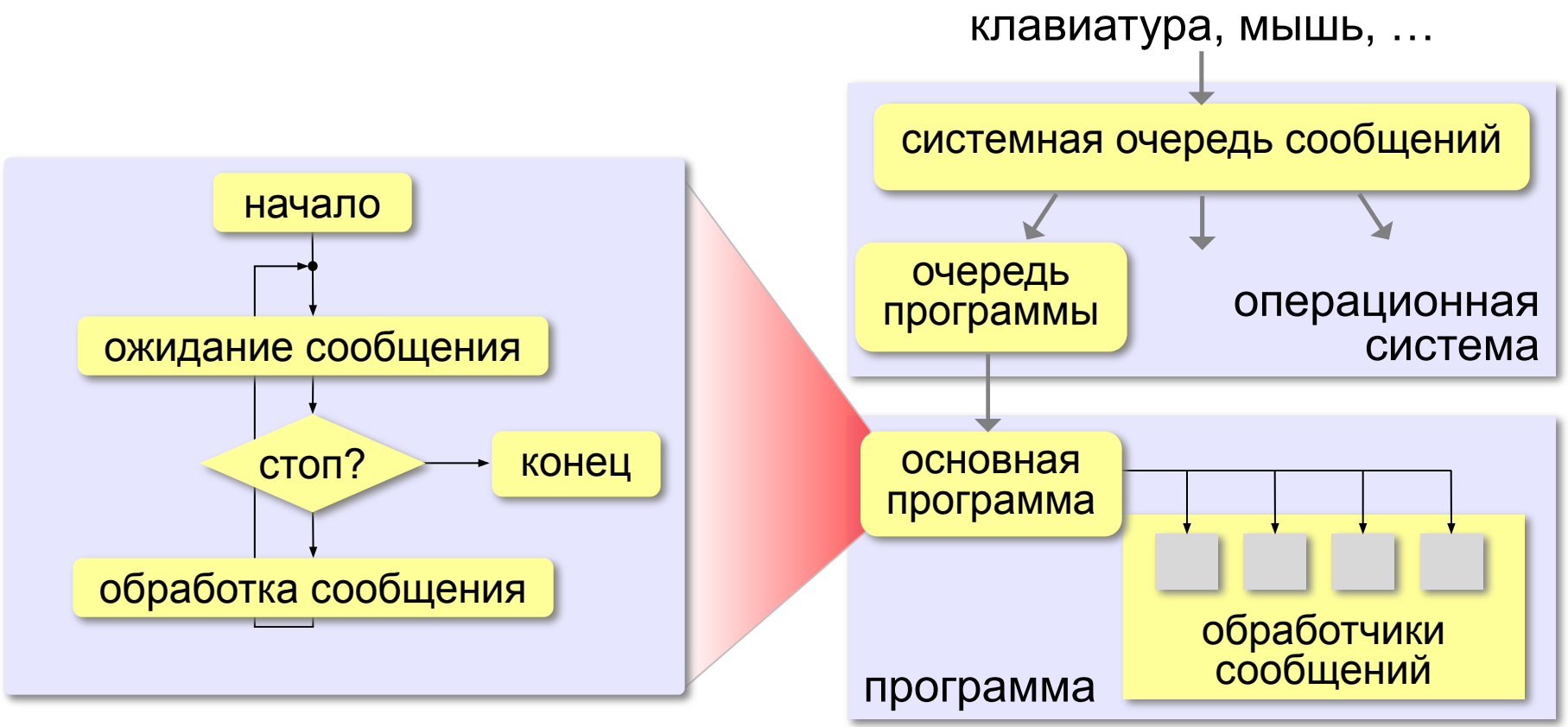
Программы, управляемые событиями

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:

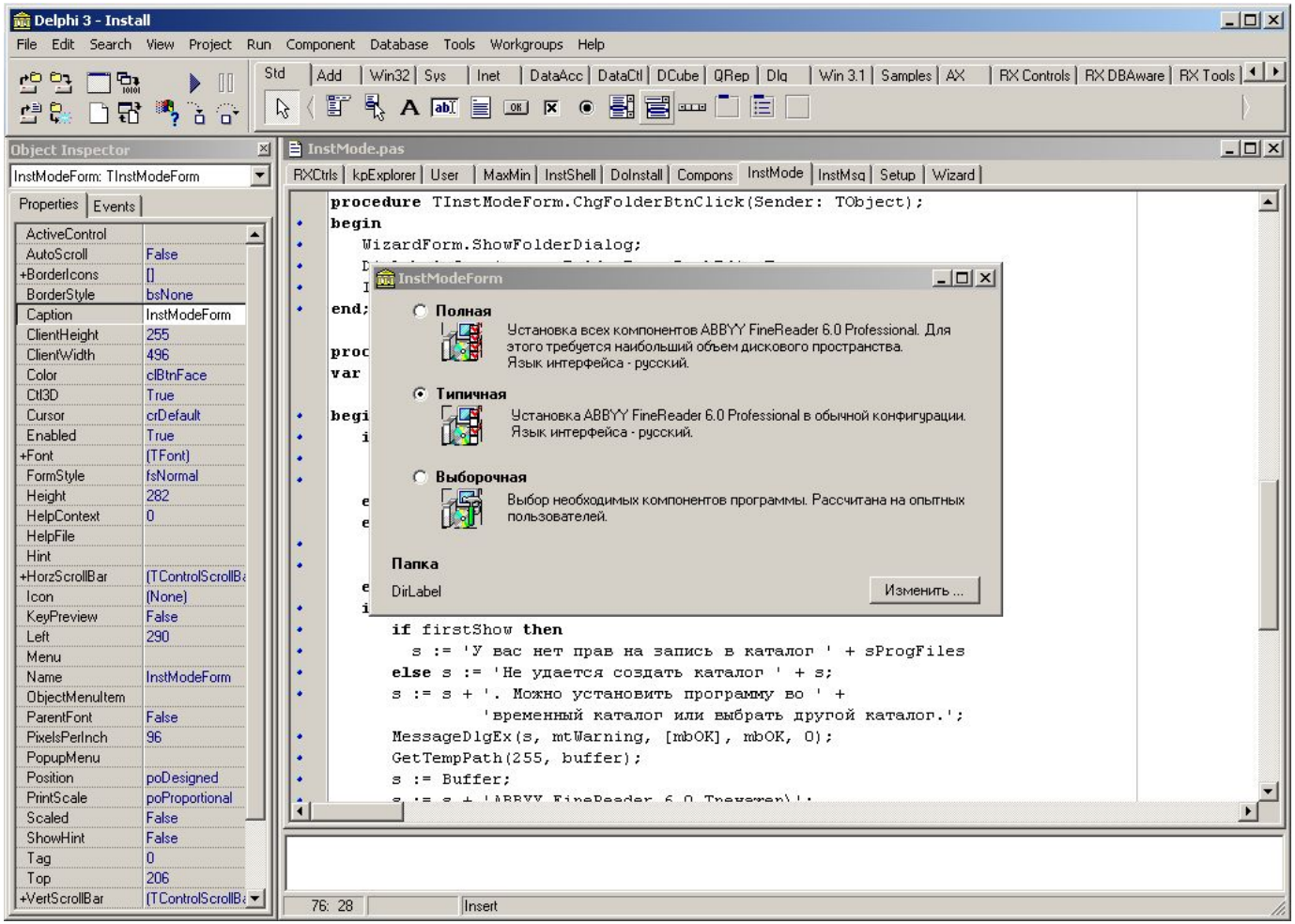
- создание **формы**
- минимальный код добавляется автоматически
- расстановка **элементов интерфейса** с помощью мыши и настройка их свойств
- создание **обработчиков** событий
- написание **алгоритмов** обработки данных

выполняются при возникновении событий

RAD-среды: Delphi

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

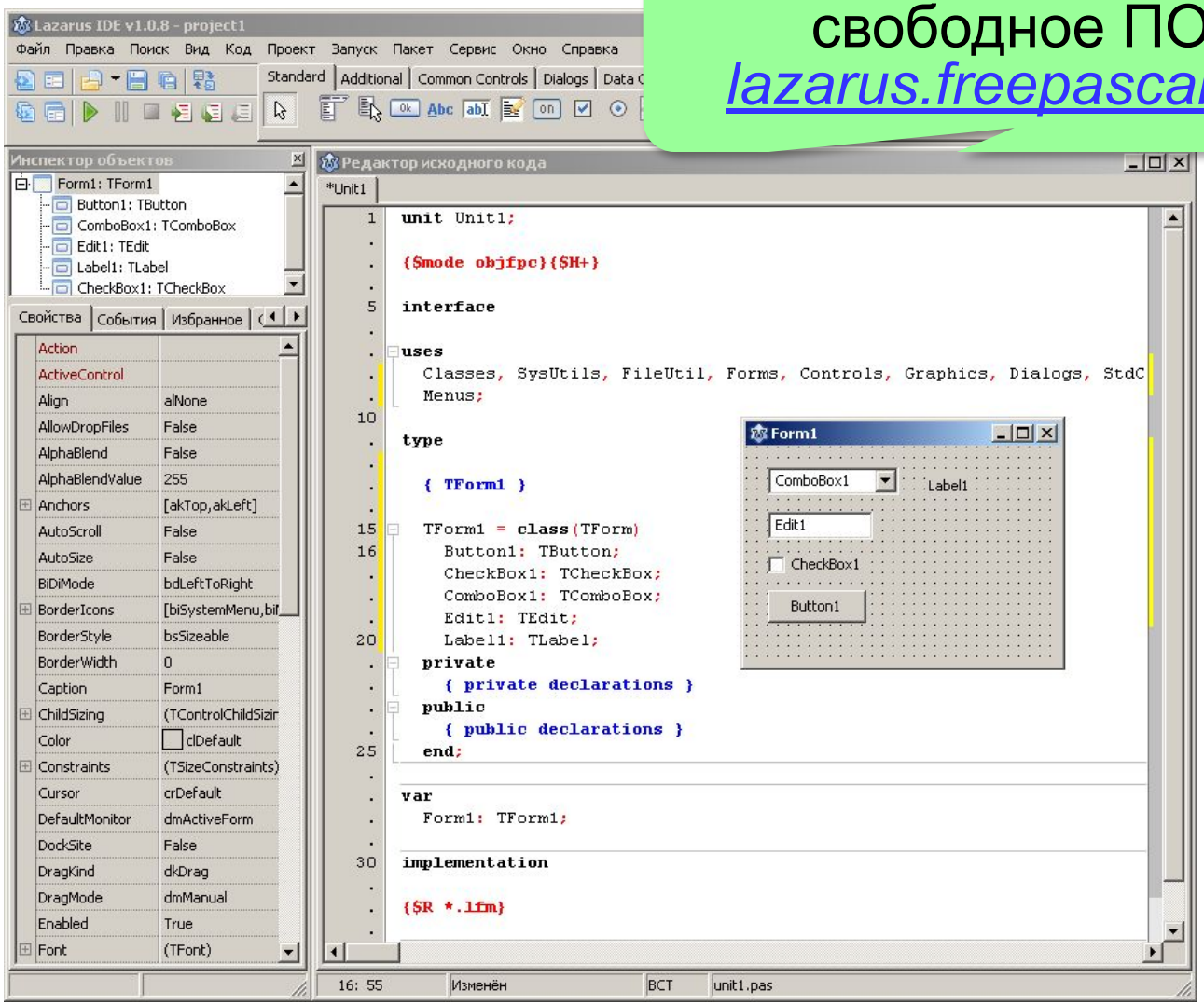
1995: *Borland*, сейчас: *Embarcadero Technologies*



RAD-среды: Lazarus

Языки: *FreePascal, Delphi*

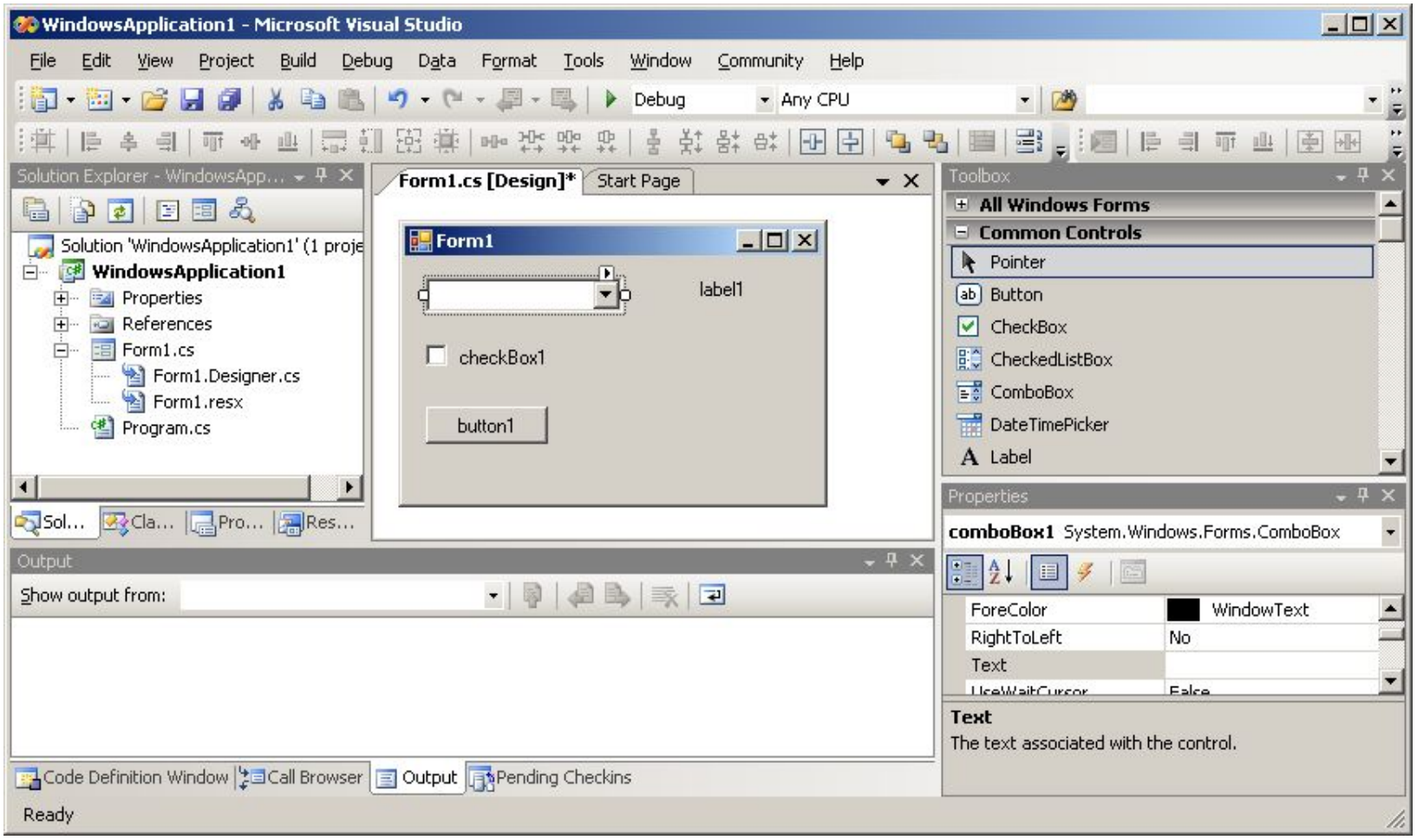
свободное ПО:
lazarus.freepascal.org



RAD-среды: MS Visual Studio

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

с 1995 по н.в.: *Microsoft*



Объектно-ориентированное программирование. Язык Python

§ 47. Графический интерфейс: основы

Графические библиотеки для Python

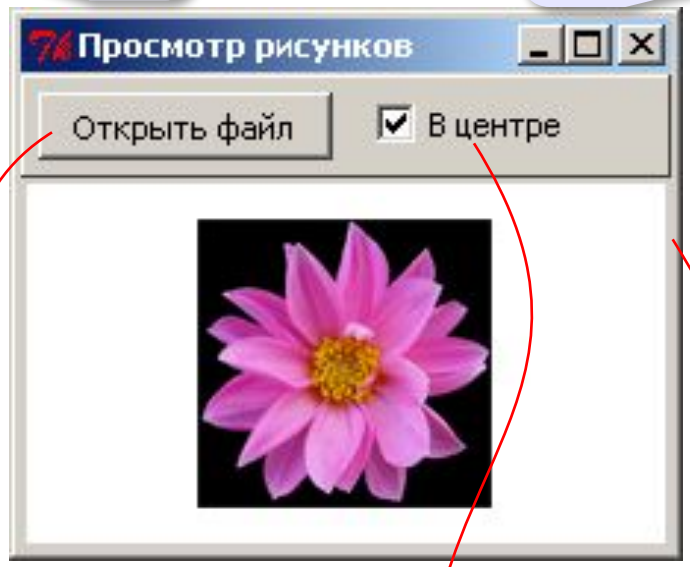
- *tkinter* (стандартная библиотека Python)
- *wxPython* (http://wxpython.org)
- *PyGTK* (http://pygtk.org)
- *PyQt* (http://www.riverbankcomputing.com/software/pyqt/intro)

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

Общие принципы

компонент (*виджет*, элемент)

форма (окно верхнего уровня)



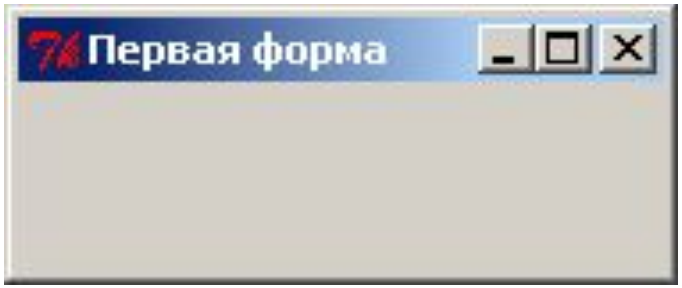
щелчок по выключателю

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

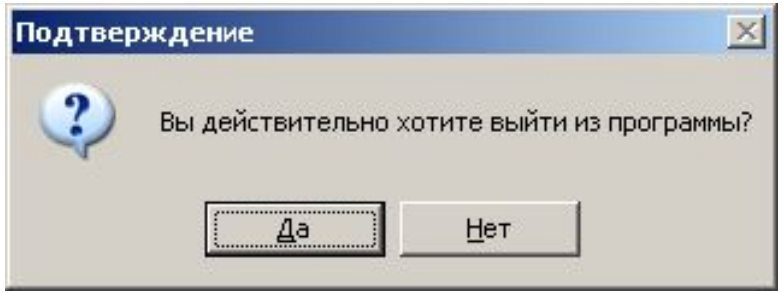
минимальные размеры

```
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
```

информация о событии

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

удалить из памяти

Привязка обработчика:

```
app.onCloseQuery = askOnExit
```

Задание

«A»: Соберите и запустите программу, которая описывается в теоретической части. Сделайте так, чтобы форма открывалась в максимально возможном размере: 500 пикселей в ширину и 300 пикселей в высоту. Нужно сделать так, чтобы её высоту нельзя было сделать менее 200 пикселей, а ширину – менее 400 пикселей.

«B»: Доработайте программу уровня B так, чтобы при щелчке на форме (событие **onClick**) появлялось диалоговое окно с сообщением «*Вы щёлкнули по форме*». Используйте для этого функцию **showinfo** из модуля **tkinter.messagebox**. Она принимает те же аргументы, что и функция **askokcancel**.

Задание

«С»: Доработайте программу уровня В так, что при одиночном щелчке мышью сообщение не появлялось, но цвет формы менялся случайным образом. При двойном щелчке по форме цвет должен становиться серым и должно появляться сообщение «*Вы сделали двойной щелчок*».

(Подсказка: изучите документацию по модулю **simpletk** – свойства и методы главного окна программы, с. 1-2).

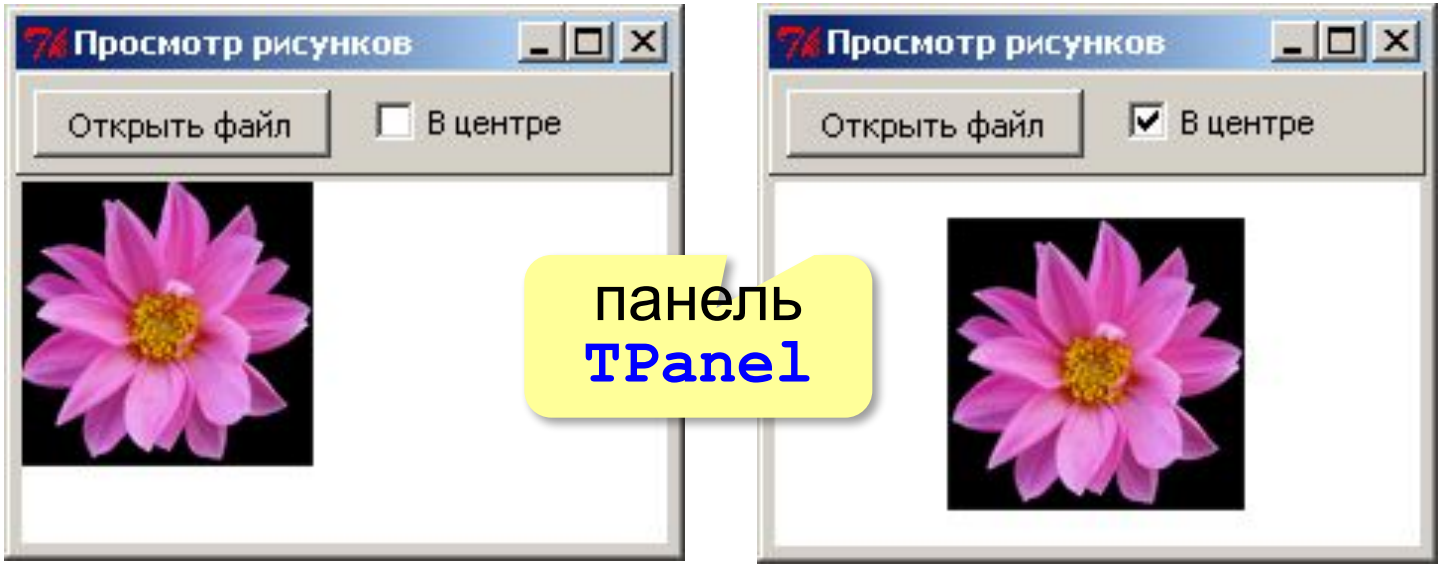
Объектно-ориентированное программирование. Язык Python

§ 48. Использование компонентов

Просмотр рисунков

кнопка **TButton**

выключатель **TCheckBox**



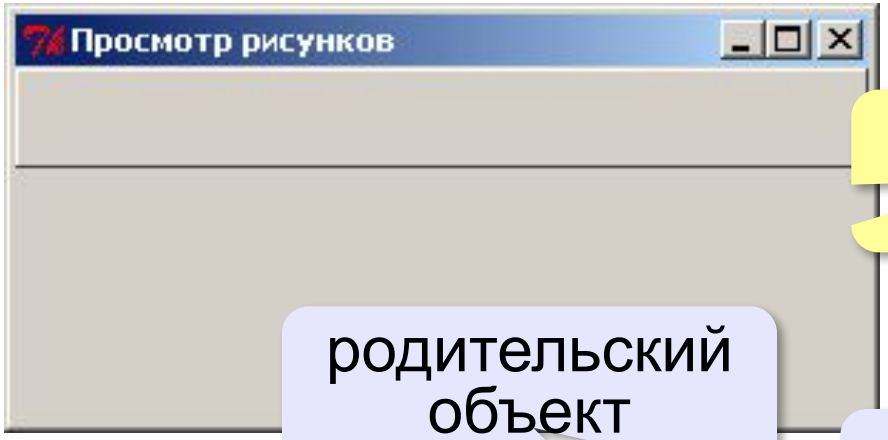
панель **TPanel**

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

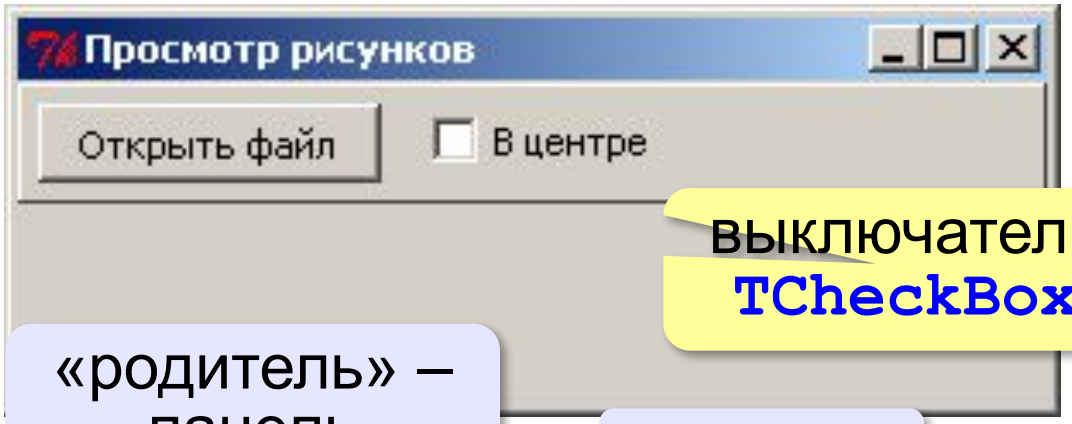
```
panel.align = "top"
```

прижать к верхней границе

выравнивание

Кнопка и выключатель

кнопка **TButton**



выключатель **TCheckBox**

«родитель» – панель

ширина

```
openBtn = TButton ( panel, width = 110,
                    height=30,
                    text = "Отк
openBtn.position = (5, 5)
```

координаты

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

Поле для рисунка

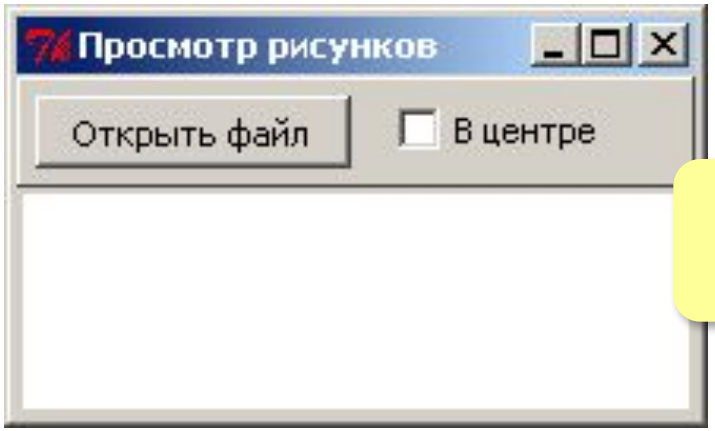


рисунок **TImage**

«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:

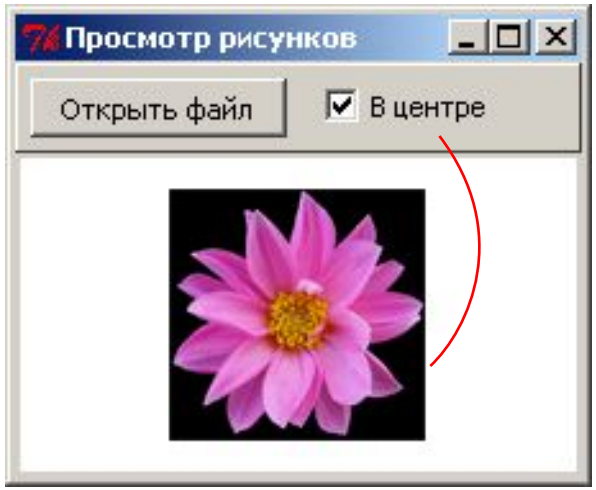
```
from tkinter import filedialog


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

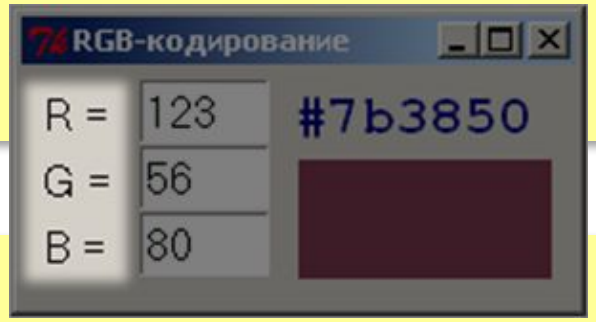
поле ввода **bEdit**
TEdit

поле ввода **gEdit**
TEdit

Основная программа

Объект-приложение:

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

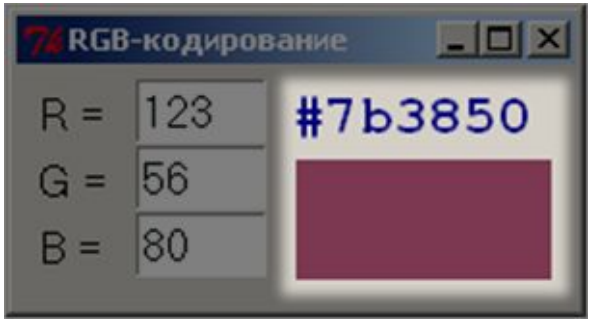


Метки RGB:

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 = 90, height = 44 )
rgbRect.position = (105, 35)
```

цвет текста

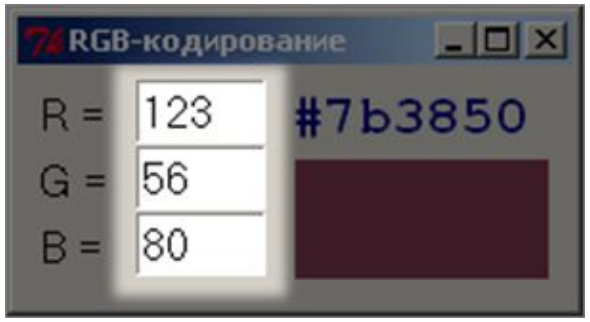
ширина и высота в **пикселях**!

Компоненты

rEdit

gEdit

bEdit



Поля ввода:

шрифт тот же, что и для меток

```
rEdit = TEdit ( app, font = f, width = 50 )
rEdit.position = (45, 5)
rEdit.text = "123"
```

ширина в **пикселях**!

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 = f"#{r:02X}{g:02X}{b:02X}"

    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 = f"#{r:02X}{g:02X}{b:02X}"
    bkColor = s
 except:
   pass
 rgbLabel.text = s
 rgbRect.background = bkColor
```

Задание

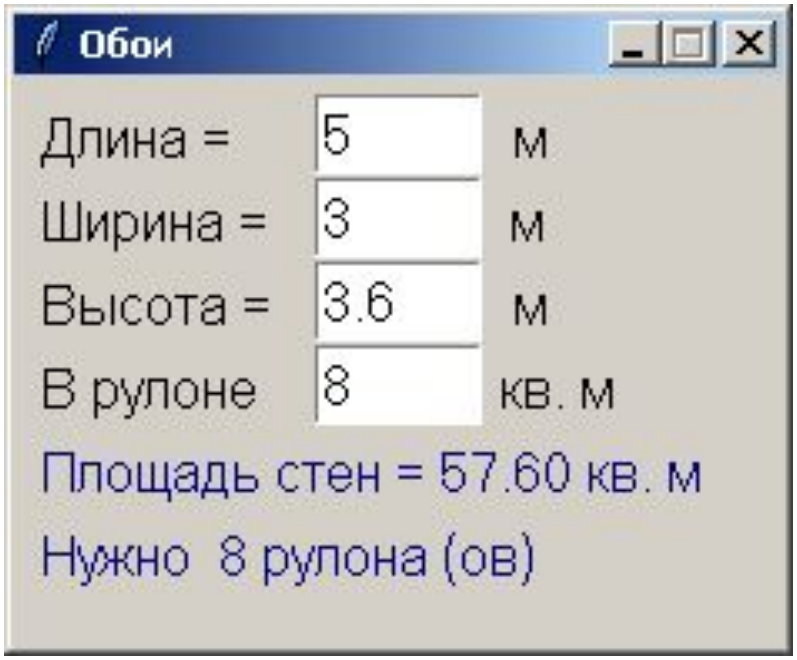
«А»: Постройте программу, которая вычисляет площадь комнаты.

Требования:

1) размер окна нельзя менять
2) при попытке закрыть окно выдаётся запрос на подтверждение
3) площадь пересчитывается сразу же, как только изменяются значения длины или ширины комнаты
4) если длина или ширина отрицательны или не числа, вместо площади выводится знак вопроса

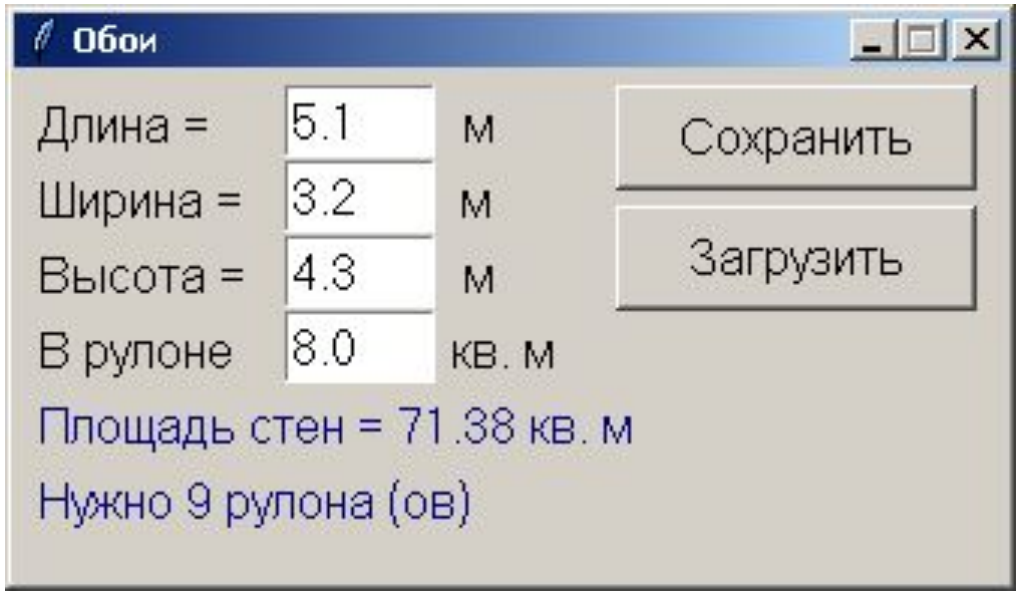
Задание

«B»: Постройте программу, которая вычисляет площадь стен комнаты и определяет, сколько рулонов обоев нужно на оклейку всех стен. Количество рулонов – целое число. Остальные требования такие же, как в варианта «A».



Задание

«С»: Доработайте программу так, чтобы по щелчку по кнопке «Сохранить» все данные сохранялись в файле с расширением **.dat** (имя файла можно выбрать), а по щелчку по кнопке «Загрузить» данные загружались из файла (имя файла также выбирается).



Объектно-ориентированное программирование. Язык Python

§ 49. 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

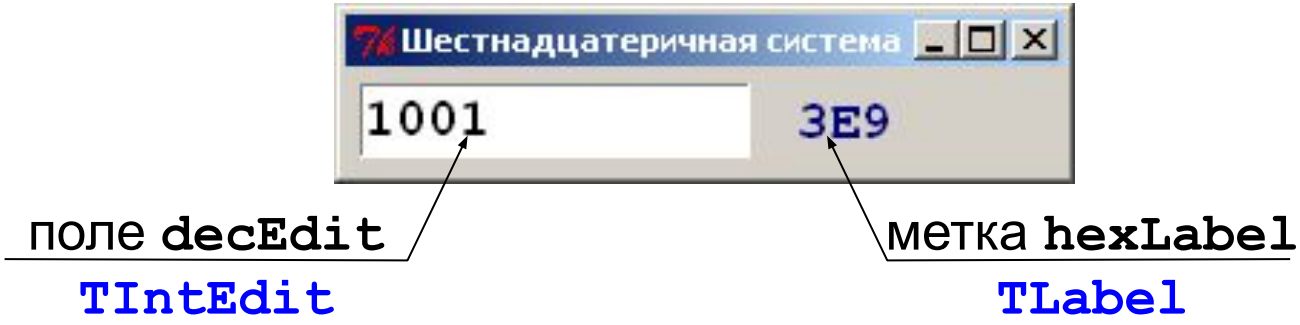
пытаемся получить целое

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 = 140, font = f )
decEdit.position = (5, 5)
decEdit.text = "1001"
```

шрифт

ширина в пикселях

Обработчик события:

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

в шестнадцатеричную систему

установить обработчик

Запуск:

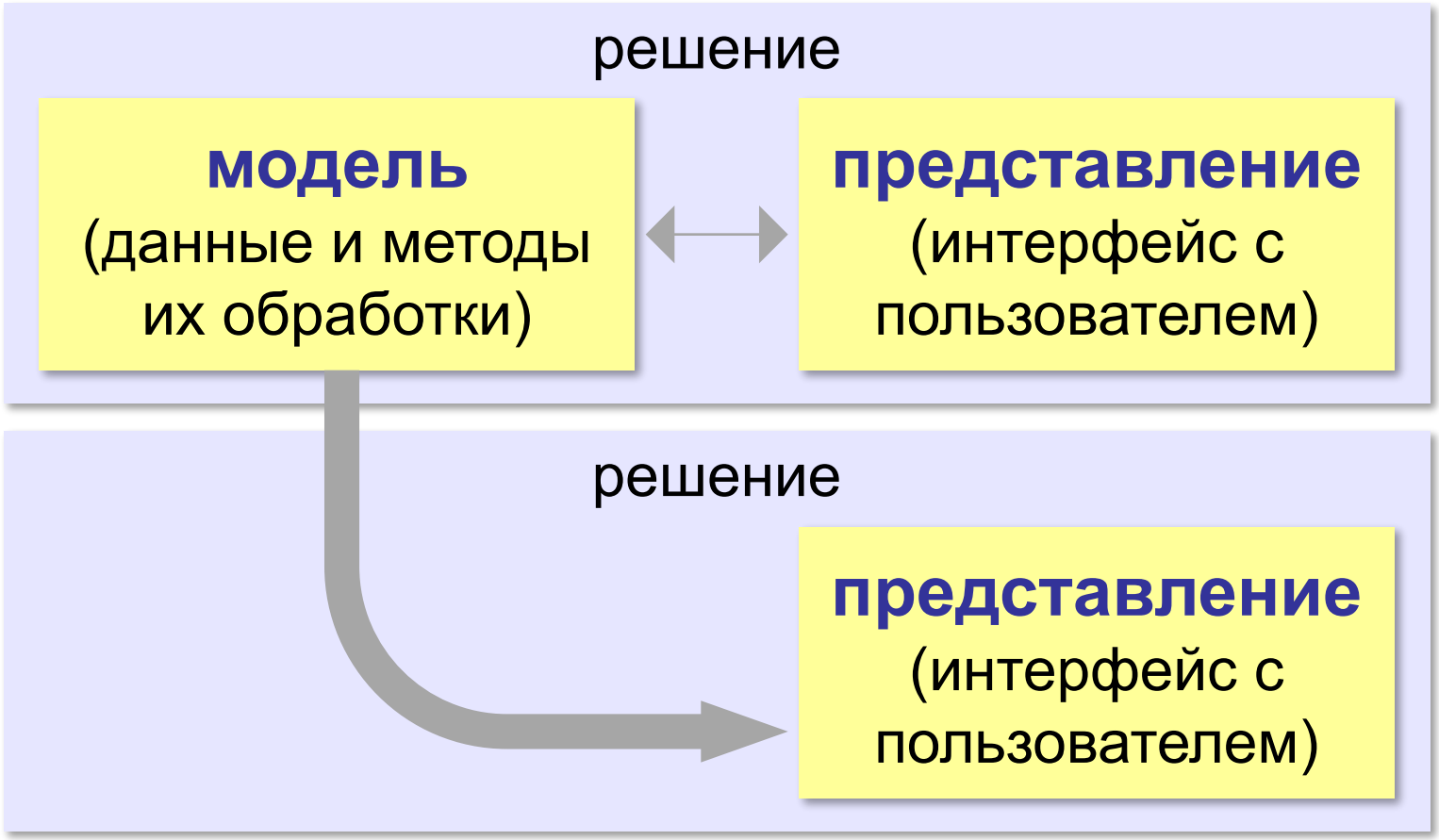
```
app.run()
```

Объектно-ориентированное программирование. Язык Python

§ 50. Модель и представление

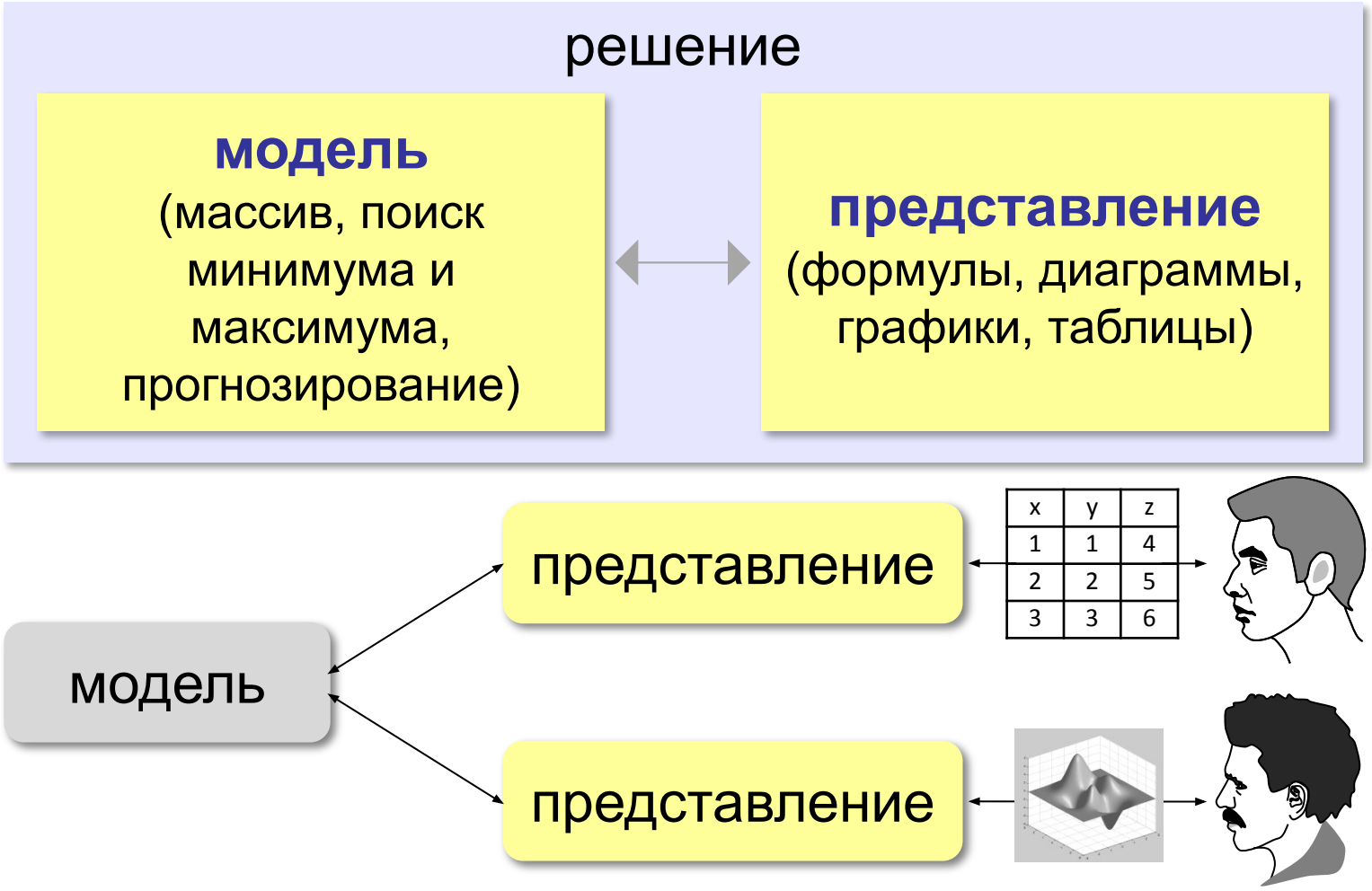
Еще одна декомпозиция

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 `+ - * /`

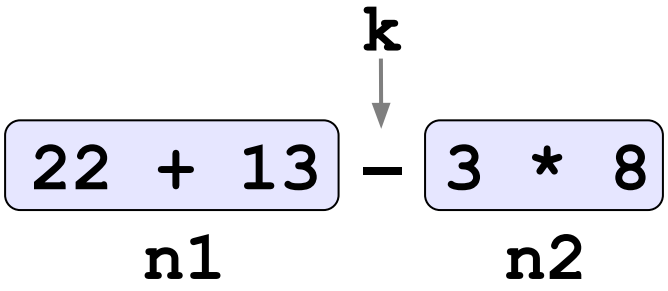
Модель:

- 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
```

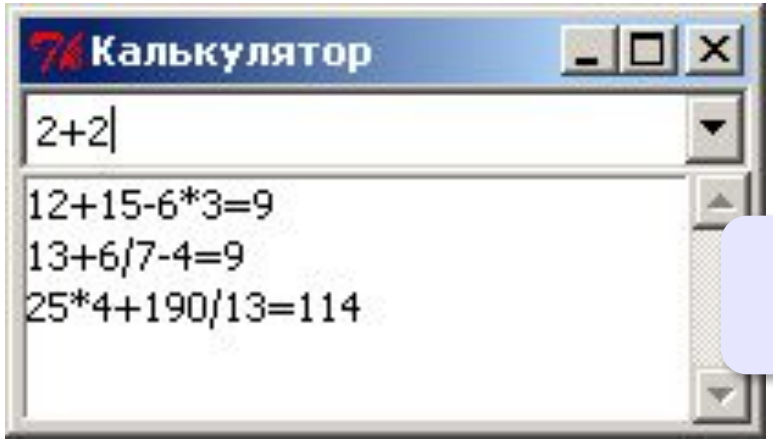
Номер последней операции:

```
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 = [] )
Input.align = "top"
Input.text = "2+2"
```

прижать к верху

текст

Список для запоминания результатов:

```
Answers = TListBox ( app, values = [] )
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 )
```

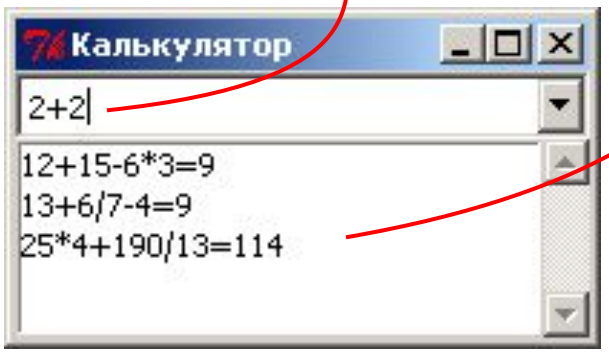
«связать»

клавиша Enter

Обработчик нажатия на клавишу Enter

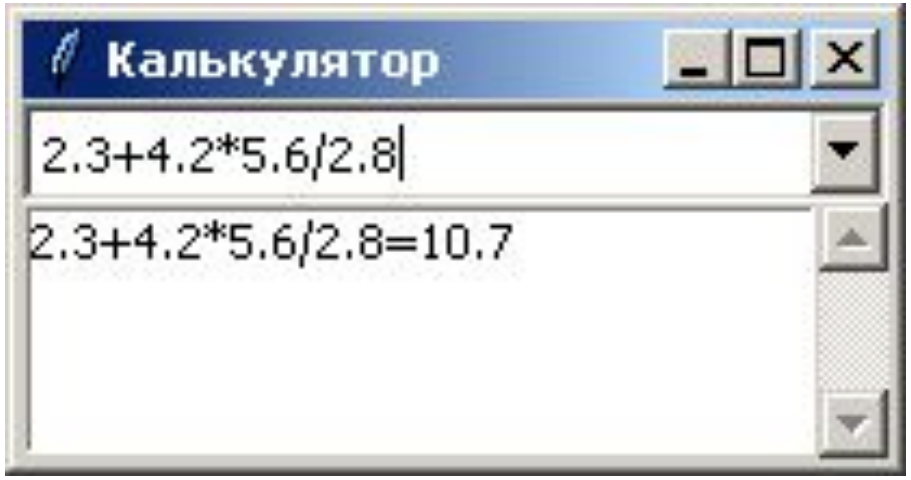
```
from model import Calc
def doCalc ( event ):
    expr = Input.text   # прочитать выражение
    x = Calc ( expr )   # вычислить
    Answers.insert ( 0, expr + "=" + str(x) )
    if not Input.findItem ( expr ):
        Input.addItem ( expr )
```

если еще нет в списке



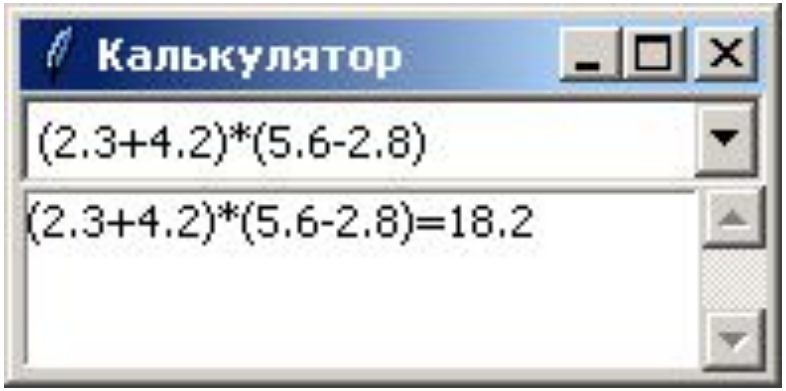
Задание

«А»: Измените программу так, чтобы она могла вычислять значения выражений с вещественными числами.



Задание

«B»: Измените программу так, чтобы она могла вычислять значения выражений со скобками.



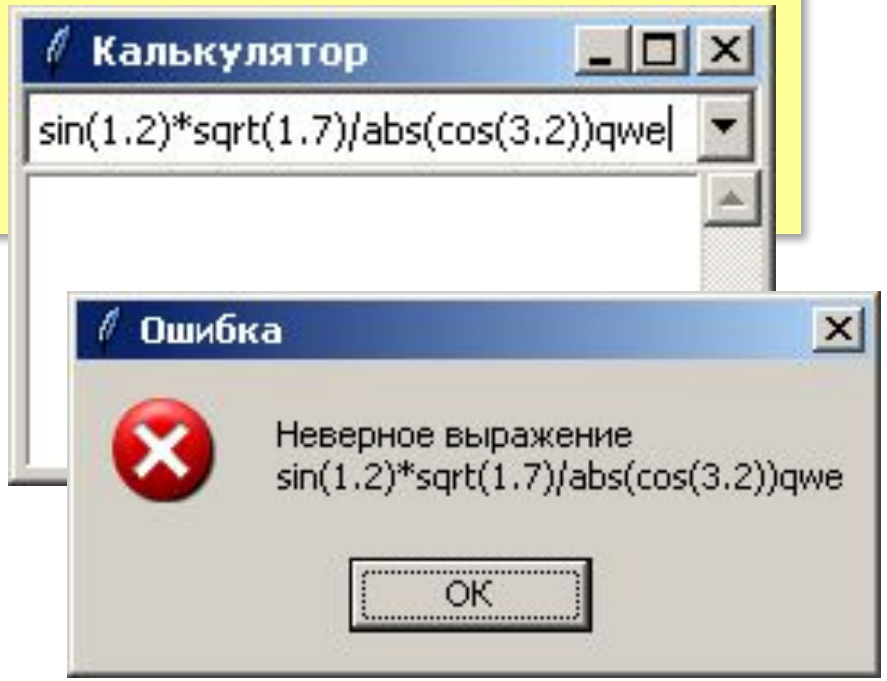
Задание

«С»: Измените программу так, чтобы она могла вычислять значения выражений, содержащих вызовы функций **`abs`**, **`sin`**, **`cos`**, **`sqrt`**.

Задание

«D»: Измените программу так, чтобы вся логика программы содержалась в классе **TCalculator**. Основная программа должны выглядеть так:

```
class TCalculator(TApplication):
    # здесь должно быть описание класса

app = TCalculator()
app.run()
```



При вводе неверного выражения нужно выводить сообщение об ошибке. Используйте функцию **showerror** из модуля **tkinter.messages**.

Задание

«D»: (продолжение) Все результаты вычислений и сообщения об ошибках записываются в файл **results.txt**:

```
...
sin(1.2)*sqrt(1.7)=1.215230290196084
Неверное выражение sin(1.2)*sqrt(1.7)qwe
```

Оформите процедуру записи в файл как метод **log** класса **TCalculator**.

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. medium.freecodecamp.org
6. www.istockphoto.com
7. иллюстрации художников издательства «Бином»
8. авторские материалы