С. В. БУКУНОВ,

СУНО]

а.

ЛАНЬ

О. В. БУКУНОВА

РАЗРАБОТКА  
ПРИЛОЖЕНИЙ  
С ГРАФИЧЕСКИМ  
ПОЛЬЗОВАТЕЛЬСКИМ  
ИНТЕРФЕЙСОМ  
НА ЯЗЫКЕ PYTHON

Учебное пособие ЛАНЬ

g

■  
•САНКТ-ПЕТЕРБУРГ\*МОСКВА\*КРАСНОДАР\*

•2023\*

УДК 004.42

ББК 32.973.26-018.2я723

Б 90 Букунов С. В. Разработка приложений с графическим пользовательским интерфейсом на языке Python : учеб­ное пособие для СПО / С. В. Букунов, О. В. Букунова. — Санкт-Петербург : Лань, 2023. — 88 с. : ил. — Текст : непосредственный.

ISBN 978-5-507-45192-0

Настоящее пособие содержит основные сведения о создании при­ложений на языке Python с графическим пользовательским интерфей­сом с помощью библиотеки Tk. Рассматриваются основные типы эле­ментов, управляющих работой приложения, и способы их программ­ной реализации. Даются понятия визуального проектирования и событийного программирования. Приводятся многочисленные прак­тические упражнения с примерами программного кода, помогающие понять основные способы практической реализации графических пользовательских интерфейсов. Рассматриваются возможности биб­лиотеки Tk по созданию компьютерной анимации. В пособии содер­жатся краткие теоретические сведения, упражнения и примеры про­грамм с подробными комментариями, а также задания для самостоя­тельной работы. В учебном пособии последовательно вводится понятийный аппарат, формулируются основные объекты, приводят­ся примеры программ на языке Python.

Соответствует современным требованиям Федерального государ­ственного образовательного стандарта среднего профессионального образования и профессиональным квалификационным требованиям.

Предназначено для студентов ссузов обучающихся по направле­нию «Информатика и вычислительная техника», имеющих базовые навыки программирования на языке Python, в том числе с использо­ванием объектно-ориентированного подхода, желающих повысить уровень своих знаний в данной предметной области.

УДК 004.42 ББК 32.973.26-018.2я723

Рецензент

Б. Г. ВАТЕР — доктор физико-математических наук, профессор ка­федры математики Санкт-Петербургского государственного архи­тектурно-строительного университета.

© Издательство «Лань», 2023 © С. В. Букунов, О. В. Букунова, 2023 © Издательство «Лань»,

Обложка

*П. И. ПОЛЯКОВА*

художественное оформление, 2023

ПЕРЕЧЕНЬ ПРОФЕССИОНАЛЬНЫХ УМЕНИИ И ЗНАНИИ, КОТОРЫМИ ДОЛЖЕН ОВЛАДЕТЬ

СТУДЕНТ

В результате освоения учебной дисциплины обучающийся дол­жен

знать: основные этапы разработки программного обеспечения; основные принципы технологии структурного и объектно­ориентированного программирования; способы оптимизации и прие­мы рефакторинга; основные принципы отладки и тестирования про­граммных продуктов;

уметь: осуществлять разработку кода программного модуля на языках низкого и высокого уровней; создавать программу по разра­ботанному алгоритму как отдельный модуль; выполнять отладку и тестирование программы на уровне модуля; осуществлять разработку кода программного модуля на современных языках программирова­ния; уметь выполнять оптимизацию и рефакторинг программного ко­да; оформлять документацию на программные средства;

иметь практический опыт в: разработке кода программного продукта на основе готовой спецификации на уровне модуля; исполь­зовании инструментальных средств на этапе отладки программного продукта; проведении тестирования программного модуля по опреде­ленному сценарию; использовании инструментальных средств на

этапе отладки программного продукта; разработке мобильных при­ложений.





ВВЕДЕНИЕ

Язык программирования Python в настоящее время можно смело отнести к наиболее популярным языкам [1-3]. Причины высокой по­пулярности Python обусловлены, в первую очередь, его универсаль­ностью. Универсальность означает возможность решения с помощью Python широкого круга задач. Для этого разработано большое коли­чество разнообразных Python-библиотек.

В частности, с помощью языка Python можно:

* создавать приложения для обработки больших объемов дан­ных (Big Data) в инженерных и научных расчетах (библиотеки NymPy, SciPy);
* создавать приложения для преобразования и визуализации данных (библиотеки pandas, matplotlib, sympy, plotly);
* создавать приложения для обработки изображений (библиотек scikit-learn, scikit-image);
* разрабатывать различные системные утилиты;
* разрабатывать мощные и надежные веб-приложения (фрейм- ворки Django, Flask, Pyramid, Tornado, TurboGears);
* работать с базами данных;
* разрабатывать приложения с графическим интерфейсом (биб­лиотеки Tk, PyQt, wxPython, turtle);
* разрабатывать компьютерные игры (библиотека Pygame);
* разрабатывать мобильные приложения (библиотека kivy).

Вторая составляющая универсальности языка Python — его

мультипарадигменность. Python поддерживает основные парадигмы программирования: процедурную, объектно-ориентированную и функциональную.

Язык Python относится к семейству интерпретируемых языков. Поэтому программы, написанные на языке Python, не отличаются вы­соким быстродействием, в отличие от программ, написанных с по­мощью компилируемых языков программирования (например, С++).

Язык Python (так же как, например, и Java) относится к семей­ству так называемых кроссплатформенных языков. Это означает, что программа, написанная на языке Python, будет работать в любой опе­рационной системе. Для этого только необходимо, чтобы в этой опе­рационной системе был установлен интерпретатор Python.

Язык Python является преемником многих других языков про­граммирования, в частности языков семейства C и командной обо­лочки Unix. Поэтому он легко интегрируется с программным кодом, написанном на таких языках, как C, C++ и FORTRAN, что является дополнительной причиной большой популярности Python.

Во многих современных вычислительных средах применяется общий набор унаследованных библиотек, написанных на FORTRAN и C, содержащих реализации алгоритмов интегрирования, оптимиза­ции, линейной алгебры, быстрого преобразования Фурье и др. По­этому многочисленные компании и национальные лаборатории ис­пользуют Python в качестве «клея» для объединения написанных за последние тридцать лет программ.

Язык Python довольно часто используется для создания прото­типов. В таких случаях сначала на языке Python разрабатывается пробная версия, а затем концепция переносится на более быстрые компилируемые языки программирования, например на C++.

В настоящее время большинство приложений, разрабатываемых для конечного пользователя, имеют так называемый графический ин­терфейс пользователя (от англ. GUI, Graphical User Interface). Под графическим интерфейсом пользователя подразумеваются различные окна с расположенными на них элементами управления (полями для ввода/вывода, полями со списками, кнопками, флажками, переключа­телями и др.), которые используются для взаимодействия пользовате­ля с программой.

Существует достаточно большое количество различных библио­тек для разработки приложений с графическим интерфейсом. При этом следует различать библиотеки, ориентированные на создание приложений для конкретной операционной системы и кроссплатфор­менные библиотеки, которые предназначены для разработки прило­жений, работающих в различных операционных системах.

Цель настоящего пособия заключается в ознакомлении студен­тов с основными возможностями языка Python по созданию приложе­ний с графическим пользовательским интерфейсом с помощью биб­лиотеки Tk.

Для написания программ использовалась интегрированная среда разработки PyCharm Community Edition компании JetBrains, а также интерактивный интерпретатор IDLE Python 3.7 64-bit.

1. МОДУЛЬ TKINTER. ОСНОВНЫЕ ВИДЖЕТЫ. ОСНОВНЫЕ МЕТОДЫ
   1. КРАТКИЕ СВЕДЕНИЯ О БИБЛИОТЕКЕ TK И МОДУЛЕ TKINTER

tkinter — это модуль Python, который работает с библиотекой Tk. С помощью этой библиотеки написано достаточно большое ко­личество проектов. Основная причина использования этой библиоте­ки при создании приложений с GUI на языке Python заключается в том, что установочный файл Python уже включает модуль tkinter в состав стандартной библиотеки. Поэтому для использования возмож­ностей библиотеки Tk достаточно подключить модуль tkinter с по­мощью инструкции import.

Возможные варианты подключения модуля tkinter:

* import tkinter;
* from tkinter import\*;
* import tkinter as tk.

Можно импортировать из модуля tkinter отдельные классы, но такой способ используется достаточно редко.

Библиотека Tk содержит компоненты графического интерфейса пользователя, написанные на языке Tcl. Не вдаваясь в подробности, модуль tkinter можно охарактеризовать как переводчик с языка Python на язык Tcl, который понимает библиотека Tk. Следует отме­тить, что все элементы управления графического интерфейса, созда­ваемые с использованием модуля tkinter, принято называть видже­тами (от англ. widgets).

* 1. ОСНОВНЫЕ ЭТАПЫ СОЗДАНИЯ ОКОННОГО ИНТЕРФЕЙСА

Процесс создания графических интерфейсов представляет собой типичный пример так называемой быстрой разработки. В основе ее лежит технология визуального проектирования и событийного про­граммирования. Смысл этой технологии заключается в том, что ос­новную часть рутинной работы по созданию приложения берет на се­бя среда разработки. Задача же программиста заключается в создании различных диалоговых окон (визуальное проектирование) и функций по обработке событий (событийное программирование), происходя­щих в этих окнах. В качестве событий окна могут выступать следу­ющие события: срабатывание временного фактора (таймера), нажатие кнопки, ввод текста, выбор пункта меню и др.

Для создания приложения с графическим интерфейсом, как пра­вило, необходимо выполнить следующие этапы:

* создать главное окно приложения;
* создать необходимые виджеты и выполнить конфигурацию их свойств (опций);
* определить события, на которые будет реагировать програм­ма;
* написать обработчики событий, т. е. программные коды, кото­рые будут запускаться при наступлении того или иного события;
* расположить виджеты в главном окне приложения;
* запустить цикл обработки событий.

Последовательность выполнения этапов может отличаться от вышеперечисленной, но первый и последний пункты всегда остаются на своих местах.

Каждая программа по созданию графического интерфейса должна начинаться с вызова конструктора Tk(), создающего объект окна. При помощи метода geometry^ этого объекта можно дополни­тельно задать размеры окна, передав в качестве аргумента строку «ширина х высота». Кроме этого, можно также создать заголовок окна, задав методу title() строковый аргумент с названием заголовка. Если размеры и заголовок не указаны, то будут использоваться зна­чения, заданные по умолчанию.

После создания любого виджета его необходимо разместить в окне. Расположение виджетов в окне приложения может быть произ­вольным, но, как правило, интерфейс приложения должен быть про­думан до мелочей и организован по определенным стандартам.

Для размещения виджетов в окне приложения предназначены специальные методы, которые называются менеджерами размеще­нии

* pack() — располагает виджеты по определенной стороне окна при помощи аргумента side, который может принимать следующие четыре значения: TOP (вверху), BOTTOM (внизу), LEFT (слева) и RIGHT (справа); значение по умолчанию — TOP (т. е. виджеты рас­полагаются вертикально);
* place() — помещает виджет в точку окна с координатами X и Y, переданными ему в качестве аргументов с числовыми значениями в пикселях;

• grid() — позволяет разместить виджет в ячейку таблицы в со­ответствии с заданными числовыми параметрами row и column, определяющими номер строки и номер столбца для этой ячейки.

Завершает каждую программу так называемый цикл обработки событий окна, вызываемый методом mainloop(). Этот метод реагиру­ет на любые действия пользователя в окне во время работы програм­мы (закрытие окна для выхода из программы, изменение размеров окна и др.).

* 1. ТЕКСТОВОЕ ПОЛЕ. ВИДЖЕТ LABEL. МЕТОД PACK()

Самым простым примером виджета может служить виджет Label (метка), который отображает обычный текст или изображение в окне приложения. Для создания метки используется конструктор Label(), которому в качестве аргументов необходимо передать имя окна и текст для самой метки в виде text = „строка”. Дополнительно в качестве аргументов конструктору можно задать вид и размер шрифта, его цвет и насыщенность, цвет фона и др. Шрифт можно за­дать двумя способами: в виде строки или в виде кортежа. Второй ва­риант является более предпочтительным, особенно в случае, когда имя шрифта состоит из двух и более слов.

Пример задачи: вывести текст в окне приложения.

Решение:

from tkinter import \* window = Tk() #Создаем окно

window.title("Вывод текста") #Создаем заголовок окна

window.geometry("500x300")

#Задаем размеры окна

= "Кафедра #Задаем шрифт как

#Создаем первую метку label = Label(window, информационных технологий", font = "Arial

text

\

16")

строку

label.pack(side = TOP)

#Размещаем метку в окне

#Создаем вторую метку label\_1 =Label(window, text = "СПБГАСУ",\

font = ("Arial", 24, "bold")) #Задаем шрифт как кортеж

label\_1.pack(padx = 150, pady = 50)#Размещаем

метку в окне

#Цикл обработки событий окна window.mainloop()

ЛАНЬ®

Результат работы программы представлен на рисунке 1.

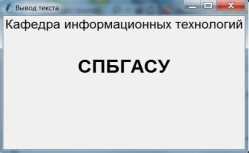


Рис. 1

Если значение аргумента side метода pack() не задано, то по умолчанию будет использоваться значение TOP, т. е. создаваемые виджеты будут располагаться вертикально один под другим. Если side = LEFT, то все создаваемые виджеты будут располагаться друг за другом слева направо и т. д.

Пример задачи: реализовать различные варианты расположе­ния текста в окне приложения.

Решение:

from tkinter import \* window = Tk() #Создаем окно

lab 1 = Label(width=8, height=3, bg='yellow',

text="1")# Создаем четыре метки

lab 2 = Label(width=8, height=3, bg='red',

text="2T')

lab 3 = Label(width=8, height=3,

bg='lightgreen', text="3")

lab 4 = Label(width=8, height=3,

bg='lightblue', text="4")

#Располагаем метки вертикально сверху вниз

lab 1.pack()

lab 2.pack()

lab 3.pack()

lab 4.pack()

#Располагаем метки вертикально снизу вверх

lab 1.pack(side = BOTTOM)

lab 2.pack(side = BOTTOM)

lab 3.pack(side = BOTTOM)

lab 4.pack(side = BOTTOM)

#Располагаем метки вертикально слева направо

lab\_1.pack(side = LEFT) lab\_2.pack(side = LEFT) lab\_3.pack(side = LEFT) lab\_4.pack(side = LEFT)

#Располагаем метки вертикально справа налево

lab\_1.pack(side = RIGHT)

lab\_2.pack(side = RIGHT)

lab\_3.pack(side = RIGHT)

lab\_4.pack(side = RIGHT)

#Комбинированное расположение меток lab\_1.pack(side = BOTTOM) lab\_2.pack(side = TOP) lab\_3.pack(side = LEFT) lab\_4.pack(side = RIGHT)

#Комбинированное расположение меток lab\_l.pack(side = LEFT) lab\_2.pack(side = LEFT) lab\_3.pack(side = BOTTOM)

Результат работы программы представлен на рисунке 2.

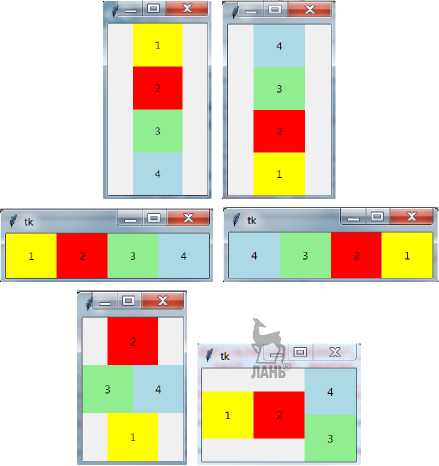


Рис. 2

* 1. ГРУППИРОВКА ВИДЖЕТОВ. ВИДЖЕТ FRAME

Из результатов работы программы видно, что расположить че­тыре метки в виде квадрата (две метки сверху, две метки сверху), ва­рьируя значения аргумента side не получается. Для решения задачи оптимизации размещения виджетов в окне приложения (в частности, для группировки виджетов в определенном порядке) используют вспомогательный виджет — фрейм (рамка), который является объек­том класса Frame. В этом случае фреймы располагают в главном окне, а виджеты — во фреймах.

Объект фрейм создается при помощи конструктора Frame(), ко­торому в качестве аргумента передается имя окна. После этого имя фрейма может быть передано в качестве первого аргумента конструк­тора виджета, чтобы указать, что именно данный фрейм, а не окно, является контейнером для виджета. При добавлении виджета во фрейм с помощью аргумента side можно указывать его привязку к определенной стороне фрейма, используя все те же константы TOP, BOTTOM, LEFT и RIGHT.

Пример задачи: с помощью фрейма реализовать вариант рас­положения четырех виджетов Label в виде квадрата.

Решение:

from tkinter import \* window = Tk() #Создаем окно

frame\_top = Frame(window)#Фрейм для верхнего ряда меток

frame\_bot = Frame(window)#Фрейм для нижнего ряда меток

* Создаем четыре метки с привязкой к конкретному

фрейму, а не окну lab\_l = Label(frame\_top,

width=8,

width=8,

width=8,

width=8,

height=3, height=3, height=3, height=3,

bg='yellow', text="l") Ц, lab\_2 = Label(frame\_top,

bg='red', text="2") lab\_3 = Label(frame\_bot,

bg='lightgreen', text="3") lab\_4 = Label(frame\_bot,

bg='lightblue', text="4")

* Располагаем фреймы вертикально один под другим frame\_top.pack() frame\_bot.pack()

#Располагаем метки во фреймах слева направо

lab\_1.pack'

lab\_2.pack'

side = side = side = side =

LEFT)

LEFT)

LEFT)

LEFT)

lab\_3.pack'

lab\_4.pack'

#Цикл обработки событий окна window.mainloop() u

Результат работы программы представлен на рисунке 3.

Рис. 3

В библиотеке Tk() существует еще один класс LabelFrame. Объектом этого класса является фрейм с подписью, которую можно задать с помощью аргумента text.

Пример задачи: реализовать предыдущую задачу с помощью фрейма с подписью.

Решение: для решения задачи достаточно создать фреймы для меток с помощью конструктора класса LabelFrame.

frame\_top = LabelFrame(window, text = "Верхний ряд")#Фрейм для верхнего ряда меток

frame\_bot = LabelFrame(window, text = "Нижний ряд" )#Фрейм для нижнего ряда меток

Результат работы программы представлен на рисунке 4.

Рис. 4

Метод pack() может получать дополнительные аргументы:

* fill — определяет возможность виджета заполнять свободное пространство вдоль какой-либо из осей, например, вдоль оси x: fill = X; значение по умолчанию — NONE, другие значения — BOTH, X, Y;
* expand — определяет положение виджета при расширении окна во весь экран; значение по умолчанию expand = 0 определяетположение виджета вверху по вертикали и посередине по горизонта­ли; expand = 1 определяет положение виджета посередине по верти­кали и посередине по горизонтали;
* padx, pady — задают внешние отступы виджета от границ ок­на вдоль соответствующих осей на определенное количество пиксе­лей (количество пикселей до края по горизонтали и по вертикали);
* ipadx, ipady — задают внутренние отступы виджета от границ окна вдоль соответствующих осей на определенное количество пик­селей (количество пикселей до края по горизонтали и по вертикали);
* anchor (якорь) — определяет положение виджета в окне или фрейме в терминах сторон света и может принимать следующие зна­чения: N (North — север), S (South — юг), W (West — запад), E (East — восток), а также их комбинации (например, SE (юго-восток, т. е. правый нижний угол)).

Пример задачи: реализовать различные варианты расположе­ния и заполнения метки с текстом с помощью различных значений аргументов метода раск().

Решение:

from tkinter import \* window = Tk() #Создаем окно

window.title("Вывод текста") #Создаем заголовок окна

window.geometry("500x300") #Задаем размеры окна #Создаем метку

label = Label (window, bg = "blue" , fg = "yel­low", \

text = "СПБГАСУ",\

font = "Arial 16") #Задаем шрифт как

строку

#Размещаем метку в окне по горизонтали — посередине label.pack()

вертикали

вверху, по

#Размещаем метку в окне по вертикали — посередине, по горизонтали — посередине label.pack(expand = 1)

#Заполняем всё пространство по горизонтали label.pack(expand = 1, fill = X)

#Заполняем всё пространство по обеим осям label.pack(expand = 1, fill = BOTH)

#Размещаем метку с текстом в левом верхнем углу label.pack(anchor = NW)

#Цикл обработки событий окна window.mainloop()

Результат работы программы представлен на рисунке 5.

|  |  |
| --- | --- |
| 0 Вывод текста ... \ = \Ш | 0 Вывод текста |
|  | fat,  лань® EBama |

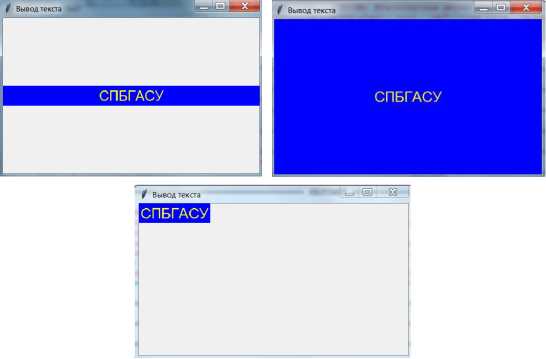


Рис. 5

* 1. УПРАВЛЕНИЕ РАБОТОЙ ПРИЛОЖЕНИЯ С ПОМОЩЬЮ КНОПОК. ВИДЖЕТ BUTTON

Виджет Button — это элемент управления графического интер­фейса Кнопка, который предназначен для совершения определенных действий в окне приложения. Кнопка может содержать текст или изображение, передающее смысловое назначение кнопки.

Объект Button создается с помощью одноименного конструкто­ра Button(). В качестве аргументов конструктору передается имя окна и набор опций. Каждый аргумент определяется как пара «аргумент = значение». В качестве аргументов могут выступать цвет фона и тек­ста, шрифт, ширина рамки и др. Определяющим аргументом является аргумент command. В качестве значения этого аргумента указывает­ся имя функции или метода, которые должны быть вызваны при нажатии пользователя на данную кнопку (обычно в графических биб­лиотеках такие функции называются обработчиками событий).

Размер кнопки по умолчанию определяется шириной и высотой текста. Для изменения этих параметров можно воспользоваться свой­ствами width и height. При этом единица измерения — знакоместо.

Краткое описание наиболее популярных аргументов приведено в таблице 1.

*Таблица 1*

|  |  |
| --- | --- |
| Аргумент | Описание |
| activebackground | Цвет фона активного элемента во время нажатия и установ­ки курсора над кнопкой |
| activeforeground | Цвет текста активного элемента во время нажатия и уста­новки курсора над кнопкой |
| bg | Цвет фона |
| fg | Цвет переднего плана (цвет текста) |
| bd | Ширина рамки в пикселях (по умолчанию bd = 2) |
| font | Шрифт |
| height | Высота кнопки (для текста — в количестве строк, для изоб­ражений — в пикселях) |
| width | Ширина кнопки (для текста — в символах, для изображе­ний — в пикселях) |
| highlightcolor | Цвет рамки при наведении курсора |
| command | Функция, вызываемая при нажатии на кнопку |
| image | Изображение для вывода вместо текста |
| justify | Вид выравнивания (LEFT — по левому краю, CENTER — по центру, RIGHT — по правому краю) |
| padx | Количество пикселей до границы рамки по горизонтали |
| pady | Количество пикселей до границы рамки по вертикали |
| relief | Вид рельефности рамки: SUNKEN — утопленная, RIDGE — выпуклая кайма, RAISEN — выпуклая, GROOVE — канавка |
| state | Состояние: NORMAL — рабочее, DISABLED — отключена |
| underline | Порядковый номер символа в тексте, который надо под­черкнуть (по умолчанию-1) |
| wraplength | Параметр, опредУжющий ширину, в которую вписывается текст |

Следует отметить, что большинство аргументов, представлен­ных в таблице 1, могут использоваться для обоих описанных выше виджетов (Label и Button) (у меток отсутствуют аргументы command и state). Значения, присвоенные различным аргументам, определяют внешний вид виджета. Аргументы можно изменять.

Библиотека Tkinter поддерживает следующие способы конфи­гурирования свойств виджетов:

* при создании объекта с помощью конструктора;
* с помощью методов config() или configure();
* путем обращения к свойству виджета как к элементу словаря. Пример задачи: реализовать различные способы конфигуриро­вания свойств кнопки.

Решение:

from tkinter import \* window = Tk() #Создаем окно window.title("Изменение заголовок окна

свойств") #Создаем

#Создаем кнопку btn = Button(window, bg "yellow", \

-

ЛАНЬ

"blue" , fg =

text = "Запуск программы",\ font = "Arial 16") #Задаем свойства кнопки в конструкторе

#Функция для изменения свойств кнопки

def change(): #Задаем свойства кнопки как элементы

словаря

btn['text'] = "Программа запущена..." btn['bg'] = '#000000' btn['fg'] = '#ffffff'

#Связываем кнопку с функцией-обработчиком событий btn.configure(command = change) #Изменяем свойства кнопки с помощью метода configure()

#Размещаем кнопку в окне btn.pack()

#Цикл обработки событий окна window.mainloop()

Результат работы программы представлен на рисунке 6.

Запуск программы ■ Программа запущена...

Рис. 6

Для получения значения каждого аргумента любого виджета можно воспользоваться методом cget(), передав ему в качестве аргу­мента имя соответствующего аргумента.

Задать цвет текста или фона можно двумя способами:

* в виде текстовой строки, например: “red”, “blue”, “yellow”, “lightgreen” и т. д.;
* в виде строкового представления шестнадцатеричного кода,

например: “#ff0000” — красный, “#ff7d00” — оранжевый,

“#ffff00” — желтый, “#00ff00” — зеленый, “#007dff” — голубой, “#0000ff” — синий, “#7d00ff” — фиолетовый, “#000000” — черный, “#ffffff” — белый, “#555555” — серый.

Пример задачи: написать программу, которая будет создавать две кнопки с разной функциональностью: при нажатии на одну из них происходит изменение цвета окна, а при нажатии на другую — выход из приложения.

Решение:

#Функция, переключающая цвет окна с желтого на зе­леный и обратно def color\_switch():

if window.cget("bg") == "yellow": #Получаем цвет фона с помощью метода cget()

window.configure(bg = "green") # Цвет фона зеленый (используем метод configure()) else:

window['bg'] = "yellow" #Цвет фона желтый (используем "терминологию" словаря)

from tkinter import \* window = Tk() #Создаем окно

window.title("Работа dLjj.4, кнопками") #Создаем

ЛАНЬ

заголовок окна

window.geometry("500x300") #Задаем размеры окна #Создаем кнопку для выхода из программы btn\_exit = Button(window, text = "Выход", bg = "#ff0000", fg = "green",\

width = 12, command = exit) #Создаем кнопку для изменения цвета фона

|  |  |  |
| --- | --- | --- |
| btn switch = | Button(window, | text = "Цвет окна", |
| bg = "blue", | fg = "red",\ |  |
|  | width = | 15, font = ("Arial", |
| 16, "bold"), | \ |  |

command = color\_switch)

btn\_switch.pack(padx = 150, pady = 50) #Размещаем обе кнопки в окне

btn\_exit.pack(padx = 150, pady = 20)

#Цикл обработки событий окна window.mainloop()

Результат работы программы представлен на рисунке 7.

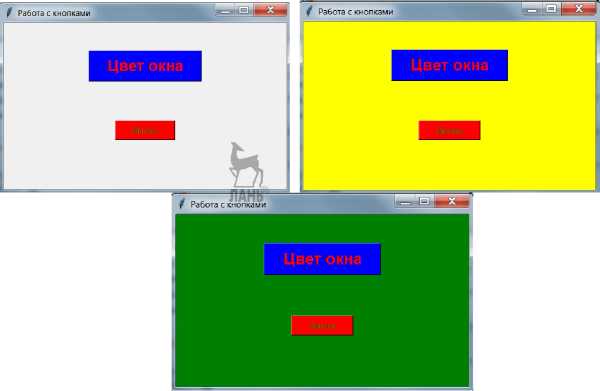


Рис. 7

Пример задачи: реализовать объектно-ориентированную вер­сию предыдущей программы.

Решение: создадим в программе собственный класс

My\_Button, объектами которого будут кнопки, которые могут вы­полнять два различных действия: изменять цвет фона окна и осу­ществлять выход из программы. В классе My\_Button создадим сле­дующие методы:

* конструктор с аргументами;
* метод setFunc() для конфигурирования обработчика собы­тий; для преобразования строки с именем функции в исполняемый код в данном методе применим встроенную функцию eval();
* метод color\_change() по изменению цвета фона окна;
* метод m\_exit(), реализующий выход из приложения с помо­щью встроенной функции exit().

from tkinter import \* class My\_Button:

#Конструктор с аргументами

def init (self, mwindow, mtext = "Цвет

окна", mwidth = 15, mheight = 3,\

mbg = "blue", mfg = "red", mpdx =

150, mpdy = 50):

#Создаем кнопку с параметрами по умолчанию self.btn = Button(mwindow, text = mtext, width = mwidth, height = mheight,\

bg = mbg, fg = mfg)

self.btn.pack(padx = mpdx, pady = mpdy) #Конфигурирование функции обработчика событий def setFunc(self, func):

self.btn['command'] = eval('self.' + func) #Функция, переключающая цвет окна с желтого на зеленый и обратно

def color\_change(self):

if window.cget("bg") = "yellow": #Получаем цвет фона с помощью метода cget()

window.configure(bg = "green") # Цвет фона зеленый (используем метод configure()) else:

window['bg'] = "yellow" #Цвет фона желтый (используем "терминологию" словаря)

#Функция для выхода из программа: def m\_exit(self): exit()

window = Tk() #Создаем окно

window.title("Работа с кнопками") #Создаем заголовок окна

window.geometry("500x300") #3адаем размеры окна #Создаем кнопку для изменения цвета фона btn\_switch = My\_Button(window) btn\_switch.setFunc('color\_change')#Связываем кнопку с обработчиком (функция color\_change) #Создаем кнопку для выхода из программы btn\_exit = My\_Button(window, "Выход", 12, 2,"#ff0000", "green", 150, 20)

btn\_exit.setFunc('m\_exit')#Связываем кнопку с

обработчиком (функция m\_exit)

#Цикл обработки событий окна window.mainloop()

* 1. ВЫВОД СООБЩЕНИЙ С ПОМОЩЬЮ ДИАЛОГОВЫХ ОКОН. МОДУЛЬ MESSAGEBOX

В программе с графическим интерфейсом вывод сообщений пользователю реализуется в виде различных диалоговых окон. Для работы с диалоговыми окнами в пакете tkinter предназначены не­сколько модулей.

Любой модуль пакета необходимо импортировать отдельно, т. е. кроме инструкции from tkinter import\* необходима дополнительная инструкция для подключения конкретного модуля. За вывод конкрет­ного окна отвечает соответствующий метод.

Методы для вывода диалоговых окон с различными сообщения­ми содержатся в модуле messagebox.

Ниже представлены возможные варианты подключения этого модуля к программе и вызова его методов:

* import tkinter.messagebox ^ tkinter.messagebox.askyesno();
* from tkinter.messagebox import\* ^ askyesno();
* from tkinter import messagebox ^ messagebox.askyesno();
* from tkinter import messagebox as mb ^ mb.askyesno() (вме­сто mb может быть любой идентификатор).

При создании диалогового окна можно указать заголовок окна, а также само выводимое сообщение. Оба этих параметра задаются в виде строковых значений для двух аргументов соответствующего ме­тода. п

В таблице 2 перечислены методы для вывода на экран диалого­вых окон с различными сообщениями.

*Таблица 2*

|  |  |
| --- | --- |
| Метод | Кнопки |
| showinfo() | OK |
| showwarning() | OK |
| showerror() | OK |
| askquestion() | Yes (возвращает строку ‘yes’) и No (возвращает стро­ку ‘no’) |
| askokcancel() | OK (возвращает 1) и Cancel (не возвращает ничего) |
| askyesno() | Yes (возвращает 1) и No (не возвращает ничего) |
| askretrycancel() | Retry (возвращает 1) и Cancel (не возвращает ничего) |

Методы, которые выводят диалоговое окно, содержащее един­ственную кнопку ОК, не возвращают никакого значения при нажа­тии на нее пользователем. Если же метод возвращает значение, то это значение можно использовать для дальнейшего условного ветв­ления.

Из данных таблицы 2 видно, что:

* из всех методов только метод askquestion() возвращает два значения;
* кнопка No метода askyesno() не возвращает значения;
* кнопка Cancel не возвращает значения.

Каждый из всех вышеперечисленных методов может иметь тре­тий аргумент. Например, чтобы получить в диалоговом окне три кнопки (Abort, Retry и Ignore) в качестве третьего аргумента можно задать аргумент type = “abortretryignore”.

Пример задачи: реализовать вывод различных диалоговых окон в зависимости от выбора пользователя.

Решение:

from tkinter import \*

import tkinter.messagebox as box

def dialog():

var = box.askyesno("Bbi6op

"Продолжаем ввод?")  
if var == 1:

box.showinfo("Продолжение" , "Продолжаем... else:

действий",

-О ( "П

ь,

^ ^ ® /11тт „ box.showwarning( Прекращение ,

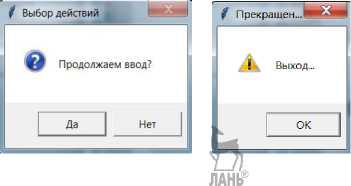
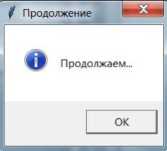
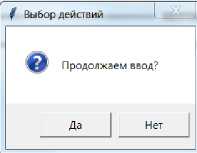
"Выход...")

window = Tk() #Создаем окно

window.title("Вывод сообщений") #Создаем заголовок окна

window.geometry("500x300") #Задаем размеры окна #Создаем кнопку для выхода из программы btn = Button(window, text = "Выбор решения", bg = "red", fg = "#00ff00",\

width = 20, font = ("Arial", 16, "bold"), command = dialog) btn.pack(padx = 100, pady = 100) #Размещаем кнопку в окне

#Цикл обработки событий окна window.mainloop()

Результат работы программы представлен на рисунке 8.

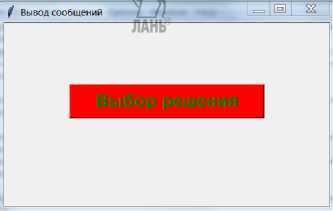


Рис. 8

* 1. ДИАЛОГОВЫЕ ОКНА ДЛЯ РАБОТЫ С ФАЙЛАМИ. МОДУЛЬ FILEDIALOG

Для работы с файлами через диалоговые окна необходимо ис­пользовать специальные методы модуля filedialog. Для получения имени открываемого файла предназначен метод askopenfilename(). Для получения имени сохраняемого файла предназначен метод asksaveasfilename(). Оба метода возвращают имя файла, который необходимо соответственно открыть или сохранить. При этом сами методы не открывают и не сохраняют файлы. Для того чтобы открыть или сохранить файл, необходимо использовать стандартные средства языка Python.

Пример задачи: реализовать возможность открытия и сохране­ния файлов с помощью диалоговых окон.

Решение:

from tkinter import \*

from tkinter import filedialog as fd def insertText():

file\_name = fd.askopenfilename()#Получаем имя файла

а = open(file\_name) #Jnrhsdftv файл для чтения s = f.read() # Считываем информацию из файла text.insert(1.0, s) #Вставляем считанную информацию в текстовое поле

f.close() #Закрываем файл

def extractText():

#Получаем имя файла, в который надо сохранить информацию

file\_name =

fd.asksaveasfilename(filetypes=((”TXT files”, ”\*.txt”),

("HTML files”, ”\*.html;\*.htm”), (”All files”, ”\*.\*”) ))

f = open(file\_name, 'w') #Открываем файл для записи

s = text.get(1.0, END) #Считываем информацию из текстового поля

f.write(s) #Записываем считанную информацию в

файл

f.close() #3акрываем файл

**ЛАНЬ®**

window = Tk() #Создаем окно text = Text(width=50, height=25) text.grid(columnspan=2)

b1 = Button(text="Открыть", command=insertText) b1.grid(row=1,sticky=E)

b2 = Button(text="Сохранить", command=extractText) b2.grid(row=1, column=1, sticky=W)

#Цикл обработки событий окна window.mainloop()

В программе создаются две кнопки и одно многострочное поле. При нажатии кнопки «Открыть» открывается диалоговое окно для от­крытия файла. После выбора файла его содержимое отображается в тек­стовом поле. При нажатии кнопки «Сохранить» открывается диалого­вое окно для сохранения информации в файл. После задания имени фай­ла информация из текстового поля записывается в указанный файл.

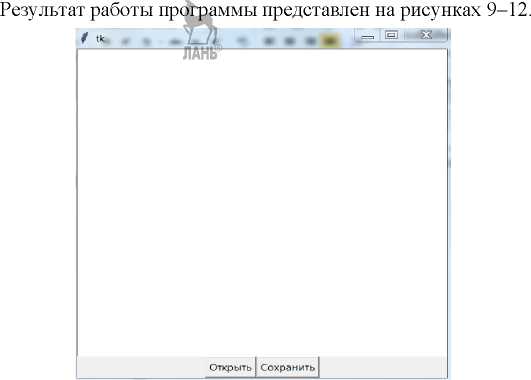


Рис. 9

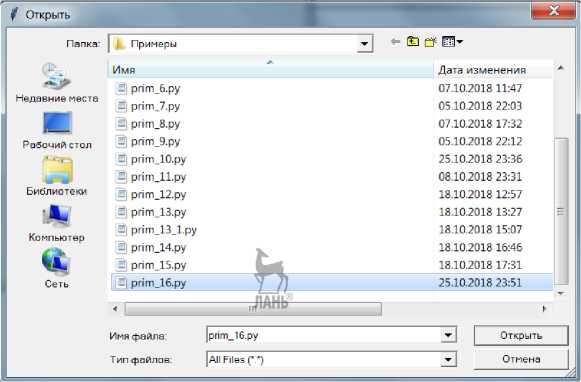




Рис. 11

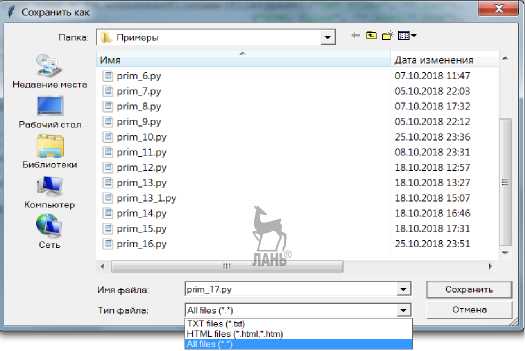


Рис. 12

В качестве аргумента обоих методов можно задать переменную filetype, которая позволяет перечислить типы файлов, которые будут открываться или сохраняться, и их расширения.

Если диалоговые окна будут закрыты без выбора или указания имени файлов будет генерироваться соответствующее исключение.

* 1. ПРИЕМ ДАННЫХ ОТ ПОЛЬЗОВАТЕЛЯ. ВИДЖЕТ ENTRY

Для получения данных, введенных пользователем в приложении с графическим интерфейсом, в модуле tkinter существует виджет Entry, который представляет собой однострочное поле ввода. Объект ввода создается при помощи конструктора Entry() с передачей ему в качестве аргументов имени родительского контейнера (окна или фрейма), а также используемых аргументов, каждая из которых пере­дается в виде пары аргумент = значение.

Как правило, виджет Entry для ввода текста располагают рядом с меткой Label, в которой описывается, что должен вводить пользо­ватель, или рядом с кнопкой, которую пользователь может нажать, чтобы выполнить какие-то действия над введенными данными. По­этому размещение виджетов в одном фрейме является, пожалуй, наиболее оптимальным вариантом размещения.

Свойства виджета Entry во многом схожи с двумя предыдущи­ми виджетами. Наиболее часто используемые опции и их краткое описание представлены в таблице 3.

*Таблица 3*

|  |  |
| --- | --- |
| Аргумент | Краткое описание |
| bd | Ширина рамки в пикселях (по умолчанию bd = 2) |
| bg | Цвет фона |
| fg | Цвет переднего плана (цвет текста) |
| font | Шрифт для текста |
| highlightcolor | Цвет рамки при наведении |
| selectbackground | Цвет фона выделенного текста |
| selectforeground | Цвет переднего плана выделенного текста |
| show | Использование маскирующих символов вместо видимых |
| state | Состояние: NORMAL — рабочее, DISABLED — отключе­но |
| width | Ширина поля ввода в символах |

Пример задачи: реализовать использование виджета Entry для получения данных от пользователя.

Решение:

from tkinter import \*

import tkinter.messagebox as box

window = Tk() #Создаем окно

window.title("Ввод данных") #Создаем заголовок окна

frame = Frame Ы1^ам)#Создаем фрейм, в который бу­дет размещено поле для ввода

entry = Entry(frame) #Создаем поле ввода и привязываем его к фрейму

#Функция для отображения данных, считанных из поля ввода

def dialog():

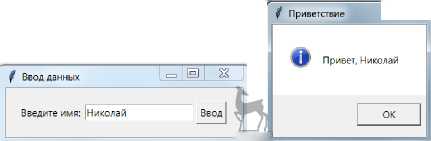
box.showinfo("Приветствие", "Привет, " + en-

try.get())

#Создаем кнопку, которая будет вызывать функцию dialog()

btn = Button(frame, text = "Ввод", command = dia­log)

#Создаем метку с поясняющим текстом lb = Label(frame, text = "Введите имя: ") #Добавляем все виджеты на фрейм lb.pack(side = LEFT) entry.pack(side = LEFT) btn.pack(side = RIGHT, padx = 5) frame.pack(padx = 20, pady = 20)

#Цикл обработки событий окна window.mainloop()

Результат работы программы представлен на рисунке 13.

ЛАНГ

Рис. 13

В отличие от свойств методы виджета Entry отличаются от ме­тодов двух предыдущих виджетов.

Основные методы виджета:

* get() — позволяет считать текст из текстового поля;
* insert() — позволяет вставить текст в текстовое поле; аргументы метода — позиция, в которую надо вставить текст (0 — начало тексто­вого поля, END — конец текстового поля, произвольное число — вставка будет производиться с указанной позиции), и сам текст;

• delete() — позволяет удалить текст из текстового поля; метод получает один или два аргумента: в первом случае удаляется один символ в указанной позиции, во втором случае удаляется срез между двумя указанными позициями, за исключением последней позиции; для очистки поля целиком для первого аргумента необходимо задать 0, а для второго — END.

* 1. РАБОТА С МНОГОСТРОЧНЫМ ТЕКСТОМ.

ВИДЖЕТ TEXT

Для ввода многострочного текста используется виджет Text, ко­торый создается с помощью соответствующего конструктора Text(). Размеры виджета по умолчанию составляют 80x24 знакомест. С по­мощью аргументов width и height эти размеры можно изменять. По аналогии с предыдущими виджетами можно конфигурировать и дру­гие свойства виджета (цвет фона, цвет текста, параметры шрифта и др.). Существует возможность реализации переноса слов на новую строку целиком, а не по буквам. Для этого необходимо задать аргу­менту wrap значение WORD.

Пример задачи: реализовать использование виджета Text для работы с многострочным текстом.

Решение:

from tkinter import \* window = Tk() #Создаем окно

window.title("MHorocTpo4Hbrn текст") #Создаем

заголовок окна

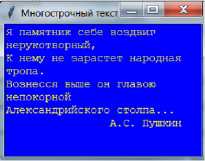
#Создаем многострочное поле ввода

text = Text(width=30, height=10, bg = "blue", fg = "yellow", wrap = WORD) text.pack()

**ЛАНЬ®**

#Цикл обработки событий окна window.mainloop()

Результат работы программы представлен на рисунке 14.



Основные методы виджета Text аналогичны методам виджета Entry: get(), insert() и delete(). Однако существуют и различия. Так, например, при вставке или удалении элемента в однострочном тек­стовом поле достаточно просто указать индекс удаляемого элемента. Для выполнения этих операций в многострочном поле необходимо указывать два индекса: номер строки и номер столбца (номер симво­ла в строке). При этом следует иметь в виду, что:

* строки нумеруются, начиная с единицы, а столбцы — начиная с нуля;
* методы get() и delete() не обязательно получают два аргумен­та, в случае необходимости получения доступа только к одному сим­волу им можно передать только один аргумент с номером позиции.

Пример задачи: написать программу, которая позволяет произ­водить стандартные операции с текстом в многострочном поле, а именно: добавлять в первую строку заданный текст, считывать с начала второй строки семь символов и выводить их в текстовом поле Label, удалять весь текст из многострочного поля, начиная с восьмой позиции во второй строке.

Решение:

from tkinter import \* window = Tk() #Создаем окно

window.title("MHorocTpo4Hbrn текст") #Создаем

заголовок окна

#Функция для вставки текста

def insertText():

s = "Кафедра информационных технологий" text.insert(1.0, s)

#Функция для считывания текста def getText():

s = text.get(2.0, 2.7) label['text'] = s #Функция для удаления текста def deleteText():

text.delete(2.8, END)

#Создаем многострочное текстовое поле text = Text(width=25, height=5) text.pack()

#Создаем фрейм frame = Frame(window) frame.pack()

#Создаем три кнопки

b\_insert = Button(frame, text="Вставить текст",

command=insertText)

b\_insert.pack(side=LEFT)

b\_get = Button(frame, text="C4MTaTb текст", com-

mand=getText)

b\_get.pack(side=LEFT) b\_delete = Button(frame, text="Удалить текст'

command=deleteText)

b\_delete.pack(side=LEFT)

#Создаем метку для вывода текста label = Label() label.pack()

#Цикл обработки событий окна window.mainloop()

ЛАНЬ

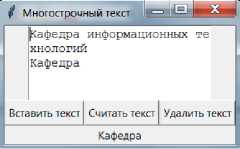
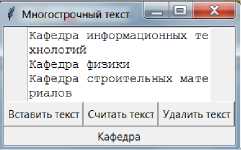
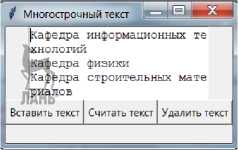
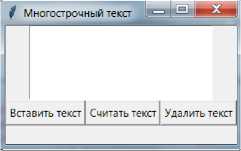
Результат работы программы представлен на рисунке 15.

Рис. 15

К особенности виджета Text можно отнести возможность различ­ного форматирования разных частей текста в многострочном текстовом поле. Для этого можно использовать методы tag\_add() и tag\_config(). Метод tag add() используется для добавления тега. При этом необходи­мо задать имя тега и ту часть текста, для которой он предназначен. Ме­тод tag configp предназначен для задания стиля оформления для тега.

Пример задачи: реализовать форматирование текста в много­строчном поле.

Решение:

from tkinter import \* window = Tk() #Создаем окно

window.title("Форматирование текста") #Создаем заголовок окна

#Создаем многострочное текстовое поле text = Text(width=45, height=10) text.pack()

#Вставляем текст

text.insert(1.0, "Первая строка\пВторая

строка\пТретья строка")

#Форматируем первую строку text.tag\_add('first', 1.0, 'l.end')

text.tag\_config('first', font = ("Verdana", 18, 'bold'), justify=CENTER)

#Форматируем вторую строку text.tag\_add('second', 2.0, '2.end')

text.tag\_config('second', font = ("Arial", 24, 'bold'), justify=RIGHT)

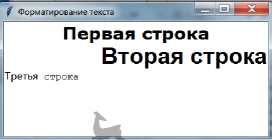
#Форматируем первое слово в третьей строке text.tag\_add('third', 3.0, '3.6')

text.tag\_config('third', font = ("Calibri", 12))

#Цикл обработки событий окна

window.mainloop()

Результат работы программы представлен на рисунке 16.



16

1

* 1. ИСПОЛЬЗОВАНИЕ ПОЛОСЫ ПРОКРУТКИ

ВИДЖЕТ SCROLLBAR

В том случае, если размер отображаемого текста превышает вы­соту виджета, то текст будет автоматически смещаться вниз. Для просмотра всего текста можно прокручивать его с помощью колесика

мышки или клавиш со стрелками на клавиатуре. Однако, как правило, в этих случаях используют полосу прокрутки (скроллер).

Скроллеры представляют собой объекты класса Scrollbar и со­здаются с помощью одноименного конструктора, после чего связы­ваются с нужным виджетом. Следует отметить, что скроллеры можно использовать не только с виджетами Text, но также и с другими ви­джетами.

Конфигурирование виджета Scrollbar подчиняется тем же правилам, что и конфигурирование других виджетов. Для задания варианта расположения виджета предназначен аргумент orient, ко­торый может принимать два значения: HORIZONTAL и VERTICAL. В качестве обработчика события необходимо указать виджет, в котором будет установлена полоса прокрутки и вариант ее расположения (xview — горизонтальная прокрутка, yview — вертикальная прокрутка). В свою очередь, виджет, в котором, устанавливается полоса прокрутки, также необходимо связать с ней с помощью одного из аргументов xscrollcommand или yscrollcommand.

Пример задачи: реализовать многострочное поле с вертикаль­ной полосой прокрутки.

Решение:

from tkinter import \* window = Tk() #Создаем окно

window.title("Полоса прокрутки") #Создаем

заголовок окна

#Создаем многострочное поле ввода

text = Text(width=30, height=5, bg = "blue", fg = "yellow", wrap = WORD) text.pack(side = LEFT)

#Создаем вертикальную полосу прокрутки

scroll = Scrollbar(orient = VERTICAL, command =

text.yview)

scroll.pack(side = RIGHT, fill = Y)

#Конфигурируем поле ввода с полосой прокрутки text.config(yscrollcommand = scroll.set)

#Цикл обработки событий окна window.mainloop()

Результат работы программы представлен на рисунке 17.

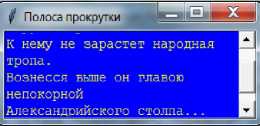


Рис. 17

1. ВЫБОР ИЗ СПИСКА. ВИДЖЕТ LISTBOX

С помощью виджета Listbox можно добавлять в приложение список элементов, предлагаемых пользователю для выбора. Для со­здания списка используется конструктор Listbox(), которому в каче­стве аргументов необходимо передать имя родительского контейнера (например, окна или фрейма) и возможные опции, наиболее популяр­ные из которых представлены в таблице 4.

*Таблица 4*

|  |  |
| --- | --- |
| Опция | Краткое описание |
| bd | Ширина рамки в пикселях (по умолчанию bd =2) |
| \_bg | Цвет фона |
| fg | Цвет переднего плана (цвет текста) |
| font | Шрифт для текста |
| height | Количество строк в списке (по умолчанию height = 10) |
| selectbackground | Цвет фона выделенного текста |
| selectmode | Режим выбора: SINGLE — одиночный (по умолчанию), EXTENDED — множественный, позволяет выбирать лю­бое количество элементов списка, используя клавиши Shift (сплошное выделение) и Ctrl (выборочное выделение), MULTIPLE — множественный, элементы выделяются щелчком левой кнопки мыши. |
| width | Ширина поля ввода списка в символах (по умолчанию width = 20) |
| yscrollcommand | Привязка к полосе прокрутки |

Полоса прокрутки добавляется в виджет Listbox так же, как и в текстовое поле.

В таблице 5 представлены основные методы объектов класса Listbox.

*Таблица 5*

|  |  |
| --- | --- |
| Метод | Краткое описание |
| insert(n, str) | Добавление элемента str в список, где n — порядковый но­мер элемента в списке (0 — вставка в начало списка, END — вставка в конец списка), str — элемент списка в виде строки |

|  |  |
| --- | --- |
| Метод | Краткое описание |
| get(n) | Получение элемента списка, где и — порядковый номер эле­мента в списке. Для получения нескольких элементов списка необходимо указать срез их индексов (например, вызов get(0, END) позволяет получить список всех элементов) |
| curselection() | Возвращает порядковый номер (индекс) выбранного элемен­та списка. Если выделено несколько элементов, то метод возвращает кортеж индексов выбранных элементов |
| delete(n) | Удаляет элемент с порядковым номером n из списка. Для удаления нескольких элементов списка необходимо указать соответствующий срез индексов |

Пример задачи: реализовать возможность выбора языка про­граммирования из поля со списком. При нажатии кнопки на экране должно появляться диалоговое окно с названием выбранного языка. Решение:

from tkinter import \* import tkinter.messagebox as box #Функция по выводу диалогового окна def dialog():

box.showinfo("Выбор языка", "Вы выбрали язык "

+ \

listbox.get(listbox.curselection())) window = Tk() #Создаем окно

window.title("Работа со списком") #Создаем

заголовок окна

#Создаем фрейм для виджетов

frame = Frame(window)

#Создаем виджет Listbox listbox = Listbox(frame)

#Заполняем список

for i in('Java', 'C++', 'Python', 'C#', 'JavaS­cript', 'PHP'):

listbox.insert(END, i) listbox.insert(3,"FORTRAN")

#Создаем кнопку

btn = Button(frame, text = "Выберите язык", bg = "lightgreen", command = dialog)

#Размещаем виджеты btn.pack(side = RIGHT, padx = 15)

listbox.pack(side = LEFT) frame.pack(padx=30, pady=30) #Цикл обработки событий окна window.mainloop()

*h*

Результат работы программы представлен на рисунке 18.

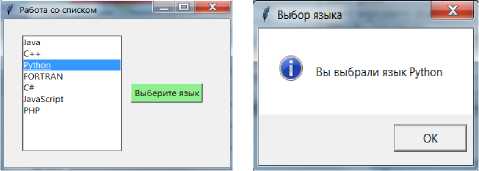


Рис. 18

1. ИСПОЛЬЗОВАНИЕ ПЕРЕКЛЮЧАТЕЛЕЙ. ВИДЖЕТ RADIOBUTTON

Виджет Radiobutton работает по принципу переключателя и ис­пользуется в программах с графическим интерфейсом для реализации выбора «один из многих». Реализовать множественный выбор с по­мощью виджета Radiobutton нельзя.

Все радиокнопки переключателя группируются вместе и свя­зываются между собой через общую переменную. Разным положе­ниям переключателя соответствуют разные значения этой перемен­ной.

Сами переменные создаются с помощью специальных классов- переменных:

* BooleanVar — позволяет принимать своим экземплярам толь­ко значения булева типа (0 или 1);
* IntVar — позволяет принимать своим экземплярам только значения целого типа;
* DoubleVar — позволяет принимать своим экземплярам только значения вещественного типа;
* StringVar — позволяет принимать своим экземплярам только значения строкового типа.

Для создания переменных используются соответствующие кон­структоры. Например, конструктор DoubleVar() инициализирует пу­стую переменную-объект вещественного типа, а конструктор StringVar() создает пустую строку.

Для создания самого объекта положения переключателя исполь­зуется конструктор Radiobutton(). В качестве аргументов этому кон­структору необходимо передать:

* имя родительского контейнера (например, окна или фрей­ма);
* текстовую строку в виде пары text = ’текст’, которая будет являться меткой;
* управляющую переменную-объект в виде пары variable = имя переменной;
* значение для присваивания в виде пары value = значение. Каждый объект положения переключателя содержит метод

select(). С помощью этого метода можно определить положение, в которое будет установлен переключатель при запуске прог­раммы.

Второй способ установки переключателя в определенное поло­жение заключается в использовании метода set().

Строковое значение, которое присваивается переменной в ре­зультате выбора переключателя, может быть получено из перемен­ной-объекта с помощью метода get().

Для снятия положения переключателя используется метод deselect().

Пример задачи: реализовать возможность выбора языка про­граммирования с помощью переключателей. При нажатии кнопки на экране должно появляться диалоговое окно с названием выбранного языка.

Решение:

from tkinter import \* import tkinter.messagebox as box #Функция по выводу диалогового окна def dialog():

box.showinfo("Выбор языка", "Вы выбрали язык " + book.get())

window = Tk() #Создаем окно

window.title("Работа с переключателем") #Создаем

заголовок окна

#Создаем фрейм для виджетов

frame = Frame(window)

#Создаем строковую переменную-объект для хранения результата выбора book = StringVar()

|  |  |  |  |
| --- | --- | --- | --- |
| #Создаем виджеты положения переключателя | | | |
| r\_1 = | Radiobutton(frame, | text = ' | 'Java', variable = |
| book, | value = 'Java') |  |  |
| r\_2 = | Radiobutton(frame, | text = | 'C++', variable = |
| book, | value = 'C++') |  |  |
| r\_3 = | Radiobutton(frame, | text = | 'Python', variable |
| = book, value = 'Python') | |  |  |
| II  1  и | Radiobutton(frame, | text = | 'FORTRAN', varia- |

ble = book, value = 'FORTRAN') r\_5 = Radiobutton(frame, text book, value = 'C#') r 6 = Radiobutton(frame, text

= 'C#'

variable =

= 'JavaScript', va­riable = book, value = 'JavaScript')

r\_7 = Radiobutton(frame, text = 'PHP', variable =

book, value = 'PHP')

#Устанавливаем переключатель при запуске в первое

положение

r\_1.select()

#book.set('Java') Можно и так установить переключатель в первое положение #Создаем кнопку

btn = Button(frame, text = "Выберите язык", bg = "lightgreen", command = dialog)

#Размещаем виджеты

r\_1.pack(padx

r\_2.pack(padx

10)

10)

10)

10)

10)

10)

10)

r\_3.pack(padx r\_4.pack(padx r\_5.pack(padx r\_6.pack(padx r\_7.pack(padx btn.pack()

frame.pack(padx = 120,pady = 30 )

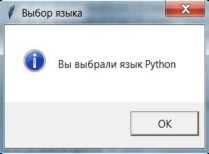
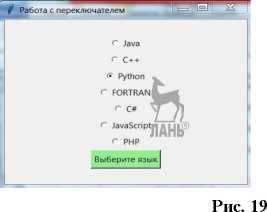
жна

#Цикл обработки событий window.mainloop()

окн

*п*

**ЛАНЬ**

Результат работы программы представлен на рисунке 19.

1. РАБОТА С ФЛАЖКАМИ.

ВИДЖЕТ CHECKBUTTON

Виджет Checkbutton (флажок) используется в программах с графическим интерфейсом для реализации как выбора «один из мно­гих», так и множественного выбора «один из многих».

Как и в случае с переключателями, в графических интерфейсах обычно используется группа таких виджетов. Однако в отличие от переключателей пользователь может выбрать как один, так и не­сколько флажков одновременно.

Как и в случае с переключателями, каждому флажку необходи­мо поставить в соответствие переменную-объект, однако устанавли­вать связи между флажками не нужно. Переменные-объекты исполь­зуются для получения сведений о состоянии флажков. По значению связанной с виджетом переменной можно установить, активирован ли данный флажок, после чего определиться с ходом выполнения программы.

Связанные с флажками переменные могут быть только тех же четырех типов, что и переменные, использующиеся при работе с пе­реключателями (т. е. BooleanVar, IntVar, DoubleVar и StringVar).

Для создания объекта флажка используется конструктор Checkbutton(). В качестве аргументов этому конструктору необхо­димо передать пять аргументов:

* имя родительского контейнера, например окна или фрейма;
* текстовую строку в виде пары text = ’текст’, которая будет являться меткой;
* управляющую переменную-объект в виде пары variable = имя переменной;
* значение для присваивания в виде пары onvalue = значение для случая, когда флажок активирован;

• значение для присваивания в виде пары offvalue = значение для случая, когда флажок не активирован (сброшен).

Программно включать флажки можно (по аналогии с переклю­чателями) с помощью методов select() и set(). Для программного вы­ключения флажка используется метод deselect().

Значение, которое присвоено в результате установки флажка, может быть получено из переменной-объекта с помощью метода getO-

Состояние флажка можно поменять на противоположное, ис­пользуя метод toggle().

Пример задачи: реализовать предыдущую задачу с помощью флажков.

Решение:

from tkinter import \* import tkinter.messagebox as box #Функция по выводу диалогового окна def dialog():

s = "Вы выбрали: "

window = Tk() #Создаем окно

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| if | var 1.get | ) == 1: | s | += "\nJava" |
| if | var 2.get | ) == 1: | s | += "\nC++" |
| if | var 3.get | ) == 1: | s | += "\nPython" |
| if | var 4.get | ) == 1: | s | += "\nFORTRAN" |
| if | var 5.get | ) == 1: | s | += "\nC#" |
| if | var 6.get | ) == 1: | s | += "\nJavaScript |
| if | var 7.get | ) == 1: | ks | += "\nPHP" |
| box.showinfo | | "Выбор | языка ", s) | |

window.title("Работа с флажками") #Создаем

заголовок окна

#Создаем фрейм для виджетов

frame = Frame(window)

#Создаем целочисленные переменные-объекты для

хранения результата выбора

var\_1 = IntVar()

var\_2 = IntVar()

var\_3 = IntVar()

var\_4 = IntVar()

var\_5 = IntVar()

var\_6 = IntVar()

var\_7 = IntVar()

#Создаем флажки

#Размещаем виджеты btn.pack(side = RIGHT) check\_1.pack(side = LEFT) check\_2.pack(side = LEFT) check\_3.pack(side = LEFT) check\_4.pack(side = LEFT) check\_5.pack(side = LEFT) check\_6.pack(side = LEFT) check\_7.pack(side = LEFT) frame.pack(padx = 20,pady = 30 ) #Цикл обработки событий окна window.mainloop()

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| check | 1 = Checkbutton( | frame, | text | = 'Java', | varia- |
| ble = | var 1,\ | onvalue | = 1, | offvalue | = 0) |
| check | 2 = Checkbutton | frame, | text | = 'C++', | varia- |
| ble = | var 2, \ | onvalue | = 1, | offvalue | = 0) |
| check | 3 = Checkbutton( | frame, | text | = 'Python' | , var- |
| iable | = var 3,\ | onvalue | = 1, | offvalue | = 0) |
| check | 4 = Checkbutton(frame, text = 'FORTRAN', | | | | |
| variable = var 4,\ | |  |  |  |  |
|  |  | onvalue | = 1, | offvalue | = 0) |
| check | 5 = Checkbutton | (frame, | text | = 'C#', | varia- |
| ble = | var 5,\ | onvalue | = 1, | offvalue | = 0) |
| check | 6 = Checkbutton | frame, | text | = 'JavaScript', | |
| variable = var 6,\ | |  |  |  |  |
|  |  | onvalue | = 1, | offvalue | = 0) |
| check | 7 = Checkbutton | frame, | text | = 'PHP', | varia- |
| ble = | var 7,\ | onvalue | = 1, | offvalue | = 0) |
| #Создаем кнопку | | w |  |  |  |
| btn = | Button(frame, | text = | "Выбор языка", | | bg = |
| "lightgreen", command | | = dialog) | |  |  |

Результат работы программы представлен на рисунке 20.





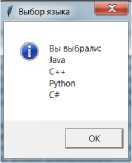


Рис. 20

1. СОЗДАНИЕ МЕНЮ. ВИДЖЕТ MENU

Для управления работой приложения с графическим пользова­тельским интерфейсом, как правило, используется меню. При выборе того или иного пункта меню выполняется предназначенная для него команда. В результате выполнения команд меню, как правило, выпол­няется какое-то действие или открывается какое-то диалоговое окно.

В библиотеке Tk меню представляет собой объект класса Menu. Этот объект должен быть связан с тем виджетом, в котором это меню будет располагаться. Обычно в роли такого виджета выступает глав­ное окно приложения. Для связывания окна и меню аргументу menu конструктора окна необходимо присвоить экземпляр Menu через имя связанной с экземпляром переменной. Для создания меню как объек­та используется конструктор Menu().

Для добавления пунктов меню используется метод add\_command(). В качестве аргумента метода выступает метка с названием пункта меню.

Пример задачи: реализовать простейшее меню из двух пунктов «Файл» и «Справка».

Решение:

from tkinter import \* window = Tk() #Создаем окно

window.title("Pa6oTa с меню") #Создаем заголовок окна

#Создаем меню в главном окне mainmenu = Menu(window)

#Добавляем пункты меню mainmenu.add\_command(label = "Файл") mainmenu.add\_command(label = "Справка")

#Конфигурируем окно с меню window.config(menu = mainmenu)

#Цикл обработки событий окна window.mainloop()

Результат работы программы представлен на рисунке 21.

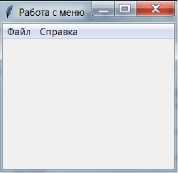


Рис. 21

Пункты меню «Файл» и «Справка» — это команды. Для связи с нужной функцией-обработчиком в эти команды можно добавить ар­гумент command. Однако в большинстве приложений, как правило, команды добавляют не к пунктам основного меню, а к пунктам до­черних меню, которые, в свою очередь, реализуются в виде раскры­вающихся списков при выборе пункта главного меню. В результате выбор пункта главного меню не запускает никакие процедуры обра­ботки, а приводит лишь к раскрытию соответствующего списка с ко­мандами, которые уже могут быть запущены по щелчку мыши.

Для реализации такого подхода создаются новые экземпляры класса Menu, которые связываются с главным меню с помощью ме­тода add\_cascade().

Пример задачи: добавить для каждого пункта предыдущего меню раскрывающиеся списки.

Решение:

from tkinter import \* window = Tk()

window.title("Меню с выпадающим списком")

#Создаем меню в главном окне

mainmenu = Menu(window) window.config(menu=mainmenu)

#Создаем пункты подменю для пункта меню "Файл" filemenu = Menu(mainmenu, tearoff=0) #Создаем еще один объект Menu

filemenu.add\_command(label="Открыть...")

#Добавляем в него пункты меню filemenu.add\_command(label="Новый") filemenu.add\_command(label="Сохранить...") filemenu.add\_command(label="Выход")

#Создаем пункты подменю для пункта меню "Справка" helpmenu = Menu(mainmenu, tearoff=0) #Создаем еще один объект Menu

helpmenu.add\_command(label="Помош;ь")#Добавляем в

него пункты меню

helpmenu.add\_command(label="0 программе") #Связываем два созданных меню с главным меню mainmenu.add\_cascade(1аЬе1="Файл", menu=filemenu) mainmenu.add\_cascade(label="Справка", menu=helpmenu)

window.mainloop()

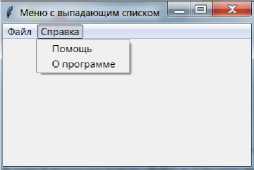
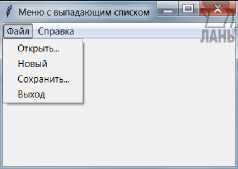
Результат работы программы представлен на рисунке 22.

Рис. 22

Из текста программы видно, что команды добавляются не к ос­новному меню, а к дочерним меню (filemenu и helpmenu). В качестве родительского окна для дочерних меню указывается не главное окно приложения (window), а основное меню (mainmenu).

Для управления возможностью открепления дочернего меню в конструкторе класса Menu можно использовать аргумент tearoff. При значении аргумента tearoff = 0 открепить дочернее меню будет не­возможно. В противном случае с помощью щелчка мыши по специ­альной линии можно было бы реализовать плавающее меню (когда tearoff = 0, эта линия отсутствует в окне приложения).

Используя описанный выше подход можно реализовывать в программах многоуровневые меню.

Пример задачи: реализовать многоуровневое меню для преды­дущей задачи путем добавления в пункт меню «Справка» дополни­тельного уровня.

Решение:

from tkinter import \*

**n**

window = Tk()

window.title("MHoroypoBHeBoe меню")

#Создаем меню в главном окне mainmenu = Menu(window) window.config(menu=mainmenu)

#Создаем пункты подменю для пункта меню "Файл" filemenu = Menu(mainmenu, tearoff=0) #Создаем еще один объект Menu

filemenu.add\_command(label="Открыть...")

#Добавляем в него пункты меню

filemenu.add\_separator()#Добавляем линию-

разделитель

filemenu.add\_command(label="Новый")

filemenu.add\_separator()#Добавляем линию-

разделитель

filemenu.add\_command(1аЬе1="Сохранить.. filemenu.add\_separator()#Добавляем линию-

разделитель

filemenu.add\_command(label="Выход")

#Создаем пункт подменю "Помощь" для пункта меню "Справка"

helpmenu = Menu(mainmenu, tearoff=0) #Создаем еще один объект Menu

#Добавляем еще один уровень меню к пункту подменю "Помощь"

helpmenu1 = Menu(helpmenu, tearoff=0) helpmenu1.add\_command(label="Локальная справка") helpmenu1.add\_separator()#Добавляем линию-

разделитель

helpmenu1.add\_command(label="На сайте")

#Связываем два созданных пункта меню с пунктом подменю "Помощь"

helpmenu.add\_cascade(label="noMom;b",

menu=helpmenu1)

helpmenu.add\_separator()#Добавляем линию-

разделитель

#Создаем пункт подменю "О программе" для пункта меню "Справка"

helpmenu.add\_command(label="0 программе") #Связываем два созданных меню с главным меню mainmenu.add\_cascade(label="Файл", menu=filemenu) mainmenu.add\_cascade(label="Справка", menu=helpmenu)

window.mainloop()

Результат работы программы представлен на рисунке 23.

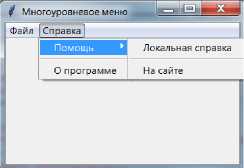


Рис. 23

Замечание. С помощью метода add\_separator() в пункты ме­ню можно добавлять линию-разделитель. Обычно такой прием ис­пользуют в тех случаях, когда необходимо разделить группы ко­манд.



1. ОБРАБОТКА СОБЫТИЙ
   1. СВЯЗЫВАНИЕ ВИДЖЕТОВ С СОБЫТИЯМИ И ДЕЙСТВИЯМИ. МЕТОД BIND()

Схема создания любого приложения с графическим интерфей­сом пользователя заключается в связывании между собой виджета, события и действия.

Например:

* виджет — кнопка, событие — щелчок левой кнопкой мыши по кнопке, действие — запуск вычислительного процесса, открытие диалогового окна, вывод изображения и т. д.;
* виджет — текстовое поле, событие — нажатие клавиши Enter на клавиатуре, действие — получение текста из поля с помощью ме­тода get() для последующей обработки программой.

За выполняемое действие отвечает функция-обработчик (или метод), которая вызывается при наступлении события. При этом один и тот же виджет может быть связан с несколькими событиями.

Для вызова функции-обработчика для виджета можно использо­вать аргумент command, который надо связать с именем этой функ­ции. В качестве события, запускающего в этом случае функцию на выполнение, будет выступать щелчок левой кнопкой мыши.

Однако существует и другой способ связывания между собой виджета, события и действия — это метод bind(). Функции- обработчики, вызываемые методом bind(), должны иметь обязатель­ный аргумент event, через который передается событие. В данном случае имя event — это просто некое соглашение. В принципе, иден­тификатор может иметь любое другое имя. Обязательное условие од­но — он должен быть первым в списке аргументов функции, или сто­ять на втором месте в списке аргументов метода в объектно­ориентированных программах.

Пример задачи: написать объектно-ориентированную про­грамму, в которой одна и та же функция-обработчик используется для разных событий. В результате работы программы должен изме­няться цвет текста на кнопке и цвет фона кнопки как при щелчке по ней мышью, так и при нажатии клавиши Enter.

Решение:

from tkinter import \* window = Tk()

window.title("Обработка событий")

#Класс-кнопка class MyButton:

#Конструктор

def init (self):

self.btn = Button(text='06pa6oTKa',

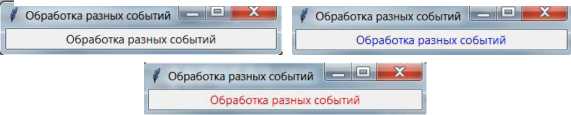
width=20, height=3,font = ("Arial", 14, "bold" )) self.btn.bind('<Button-1>', self.change) self.btn.bind('<Return>', self.change) self.btn.pack()

#Метод-обработчик

def change(self, event):

self.btn['fg'] = "blue" self.btn['bg'] = "yellow"

#Создаем кнопку как объект класса MyButton MyButton()

window.mainloop()

Результат работы программы представлен на рисунке 24.

Рис. 24

Однако такой подход вряд ли можно назвать оптимальным, по­скольку код обоих функций-обработчиков в данном случае фактиче­ски идентичен.

Другим решением этой задачи могло бы быть создание одной функции-обработчика, например, с именем change\_color(), которая в качестве дополнительного аргумента получала бы цвет шрифта. Од­нако при этом возникает проблема передачи этого аргумента в метод bind(), поскольку указание списка аргументов для функции change\_color() в вызове метода bind() означало бы вызов самой этой функции.

def change\_color(event, col): lb['fg'] = col

lb.bind('<Button-1>', change\_color(event, 'blue'))

При этом надо иметь в виду, что все функции в языке Python возвращают значение. Даже в том случае, когда оператор return от­сутствует в функции, функция вернет значение None. Поэтому в дан­ном случае даже при правильно переданных аргументах функция change\_color() метода bind() получит в качестве второго аргумента значение None, а не объект-функцию.

Для решения этой проблемы можно использовать лямбда- функции.

Пример задачи: реализовать предыдущую программу с исполь­зованием лямбда-функций при работе с кнопками.

Решение:

from tkinter import \*

**ЛАНЬ®**

window = Tk()

window.title("06pa6oTKa разных событий")

#Функция обработчик (одна функция для нескольких событий)

def change\_color(col):  
lb['fg'] = col

#Создаем метку

lb = Label(text = "Обработка разных событий") lb.pack()

#Создаем кнопку, окрашивающую текст в синий цвет, и сразу размещаем ее

Button(command = lambda col = 'blue': change\_color(col)).pack()

#Создаем кнопку, окрашивающую текст в красный цвет, и сразу размещаем ее

Button(command = lambda col = 'red': change\_color(col)).pack()

window.mainloop()

* 1. ВИДЫ СОБЫТИЙ

Под событием в приложениях с графическим интерфейсом поль­зователя, как правило, подразумевается воздействие пользователя на элементы интерфейса. Принято выделять три основных типа событий:

• события, связанные с мышью;

* события, связанные с клавиатурой;
* события, связанные с изменением виджетов.

Иногда событие может представлять собой комбинацию указан­ных событий (например, нажатие мыши при нажатой клавише на кла­виатуре).

К наиболее часто используемым событиям, связанным с мышью, относятся следующие события:

* <Button-1> щелчок левой кнопкой мыши;
* <Button-2> щелчок средней кнопкой мыши;
* <Button-3> щелчок правой кнопкой мыши;
* <Double-Button-1> двойной щелчок левой кнопкой мыши;
* <Motion> движение мышью;
* и др.

Пример задачи: написать программу, в которой необходимо реализовать возможность изменения заголовка главного окна в зави­симости от действий пользователя (щелчок левой кнопкой, щелчок правой кнопкой, движение мышью).

Решение:

from tkinter import \*

#Функции-обработчики различных событий def mouse\_left(event):

window.title("Левая кнопка мыши")

def mouse\_right(event):

window.title("Правая кнопка мыши")

def mouse\_move(event): x = event.x y = event.y

ss = "Движение мышью {0}x{1}".format(x, y) window.title(ss)

window = Tk() window.minsize(width

height

300)

window.bind('<Button-1>', mouse\_left) window.bind('<Button-3>', mouse\_right) window.bind('<Motion>', mouse\_move)

window.mainloop()

Событие (event) в tkinter — это объект со своими атрибутами. В предыдущей программе в качестве виджета выступает главное окно приложения, а в качестве события — перемещение мыши. В функции mouse\_move() происходит извлечение значений атрибутов x и у объ­екта event, в которых хранятся координаты местоположения курсора мыши в системе координат главного окна приложения.

При обработке событий, связанных с клавиатурой, следует иметь в виду, что:

* буквенные клавиши можно записывать и без угловых скобок, например ‘f’, ‘k’ и т. д.;
* для небуквенных клавиш существуют специальные зарезерви­рованные слова, например:
* <Return; нажатие клавиши <Enter>;
* <space> пробел;
* сочетания пишутся через тире; в случае использования так называемого модификатора он указывается первым, например:
* <Shift-Up> — одновременное нажатие клавиш <Shift> и стрелки вверх;
* <Control-B1-Motion> движение мышью с нажатой левой

кнопкой и клавишей <Ctrl>.

При работе с фокусом следует иметь в виду, что:

* событие получения фокуса обозначается как <FocusIn>;
* событие снятия фокуса обозначается как <FocusOut>;
* фокус перемещается по виджетам при нажатии клавиш <Tab>, <Ctrl+Tab>, <Shift+Tab>, а также при щелчке по ним мы­шью (за исключением кнопок).

Пример задачи: написать программу, в которой:

* при нажатии клавиши <Enter> текст, введенный пользовате­лем в текстовое поле, копируется в метку;
* при нажатии комбинации клавиш <Ctrl+t> этот текст выделя­ется;

• при нажатии комбинации клавиш <Ctrl+q> происходит выход из приложения.

Решение:

from tkinter import \*

#Функции-обработчики различных событий def close\_win(event): window.destroy()

def text\_to\_Label(event): s = t.get()

lbl.configure(text = s)

def select\_All(event):

window.after(10, select\_all, event.widget)

def select\_all(widget):

widget.selection\_range(0, END)

widget.icursor(END) #Установка курсора в конец

window = Tk()

#Создаем текстовое поле t = Entry(width = 50) t.focus\_set() #Устанавливаем фокус в текстовое поле t.pack()

#Создаем метку

lbl = Label(height = 4, fg = 'orange', bg = 'dark- blue', font = 'Times 16 bold') lbl.pack(fill = X)

t.bind('<Return>', text\_to\_Label) t.bind('<Control-t>', select\_All) window.bind('<Control-q>', close\_win)

window.mainloop()

Результат работы программы представлен на рисунке 25.

**Обработка событий с клавиатуры**

Рис. 25

В данной программе метод after() выполняет функцию, указанную во втором аргументе (функция select\_all()), через проме­жуток времени, указанный в первом аргументе. В третьем аргументе передается значение атрибута widget объекта event. В качестве ви­джета используется текстовое поле t, которое передается в качестве аргумента в функцию select\_all().

1. СОЗДАНИЕ ГРАФИЧЕСКИХ ИЗОБРАЖЕНИЙ И АНИМАЦИИ

Для создания графических изображений в модуле tkinter ис­пользуется класс Canvas. Объектом этого класса является так назы­ваемый холст, на который и наносятся изображения. Для прорисовки различных изображений используются соответствующие методы класса Canvas.

При создании холста необходимо указать его размеры в виде ширины и высоты. Для размещения выводимого объекта в графиче­ском окне необходимо задать координаты этого объекта в системе координат окна. Как и во всех других графических приложениях, в модуле tkinter используется система координат, в которой ось абс­цисс направлена слева направо, а ось ординат — сверху вниз. При этом в качестве начала координат выступает левый верхний угол графического окна.

* 1. СОЗДАНИЕ ГРАФИЧЕСКИХ ПРИМИТИВОВ

На первом этапе создания любого графического изображения необходимо создать объект класса Canvas. Для создания объекта- холста используется конструктор Canvas(). В качестве аргументов конструктору необходимо передать идентификатор родительского окна (как правило, это главное окно приложения) и размеры холста (ширину и высоту). Как и большинство конструкторов, конструктор класса Canvas имеет и другие аргументы, для которых установлены значения по умолчанию (например, цвет фона).

Простейшим графическим примитивом является линия. Для прорисовки линии в классе Canvas предназначен метод create\_line(). Обязательными аргументами этого метода являются координаты начала и конца линии, задаваемые в пикселях. Остальные аргументы не являются обязательными, поскольку содержат значения по умол­чанию. В качестве таких аргументов, в частности, могут выступать:

* fill — определяет цвет линии;
* activefill — определяет цвет линии при наведении на неё кур­сора;
* width — определяет толщину линии;
* arrow — задает наличие стрелки у линии; если стрелка долж­на находиться в начале линии, то arrow = FIRST, если в конце — то arrow = LAST;
* arrowshape — определяет форму стрелки, задаваемую стро­кой из трех чисел;
* dash — задает пунктирную линию в виде кортежа из двух чи­сел: первое число определяет количество заполняемых пикселей, вто­рое число — количество пропускаемых пикселей.

Пример задачи: написать программу, которая будет рисовать в графическом окне линии разного типа.

Решение:

from tkinter import \*

#Создаем окно window = Tk() window.title('Линии')

#Создаем холст

c = Canvas(window, width=200, height = 200,

bg='yellow')

c.pack()

#Создаем сплошную линию

c.create\_line(20, 15, 150, 45, width = 3, fill =

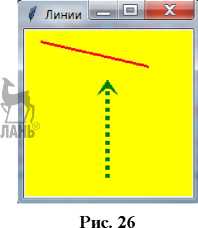
'red')

#Создаем линию со стрелкой

c.create\_line(100, 180, 100, 60, fill='green',

width=5, arrow=LAST, dash=(10,2),arrowshape="10 20 10") window.mainloop()

Результат работы программы представлен на рисунке 26.



Для рисования прямоугольников в модуле tkinter используется метод create\_rectangle(). Обязательными аргументами метода явля­ются координаты левого верхнего и правого нижнего углов. Осталь­ные аргументы не являются обязательными. Кроме необязательныхаргументов, совпадающих с необязательными аргументами метода create\_line(), метод create\_rectangle() содержит необязательный ар­гумент outline, который задает цвет рамки прямоугольника. Необхо­димо отметить, что в случае прямоугольника некоторые аргументы имеют смысловое значение, отличное от смыслового значения этих аргументов в методе create\_line(). В частности, аргумент width опре­деляет толщину линии рамки, аргумент fill — цвет заполнения пря­моугольника (цвет заливки).

Пример задачи: написать программу, которая будет рисовать в графическом окне различные прямоугольники. Для одного из прямоугольников реализовать эффект изменения его рамки из сплошной линии на пунктирную при наведении мыши на этот пря­моугольник.

Решение:

from tkinter import \*

#Создаем окно window = Tk()

window. ^^е('Прямоугольники')

#Создаем холст

c = Canvas(window, width=300, height=200) c.pack()

#Создаем прямоугольники

c.create\_rectangle(20, 20, 280, 40, outline =

'red')

c.create\_rectangle(100, 60, 200, 180,

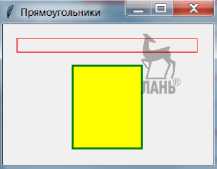
fill='yellow', outline='green',

width=3, activedash=(5, 4))

window.mainloop()

Результат работы программы представлен на рисунке 27.

|  |  |  |
| --- | --- | --- |
| $ Прямоугольники Le | | de 1- |
| 1 1 | | |
|  | | |
|  |  |  |

Рис. 27

Для рисования ломаных линий и многоугольников произволь­ной формы в модуле tkinter используется метод create\_polygon(). В качестве аргументов данного метода необходимо задать координаты всех точек линии или многоугольника.

Пример задачи: написать программу, которая будет рисовать в графическом окне различные многоугольники произвольной формы, например ромб, трапецию, параллелепипед.

Решение:

from tkinter import \*

„„

#Создаем окно window = Tk()

window.title('Многоугольники произвольной формы') #Создаем холст

c = Canvas(window, width=450, height=250) c.pack()

#Создаем фигуры произвольной формы #Рисуем ромб

(180, 10), (100, 90), (260, 90),

c.create\_polygon fill = 'yellow',

#Рисуем трапецию c.create\_polygon((40, 180), (10, 180),

fill='orange' #Рисуем параллелепипед c.create\_polygon((300, 180), (400, 230), fill=

c.create\_polygon((300, 150), (400, 180), fill=

outline = 'red')

190,

110), (160, outline='green'

110)

, width=3)

180), (400,

= 'blue') 150), (430,

= 'blue')

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| c.create polygon((400 | | , 230) | , (400, 180 | ), (430, |
| 150), (430, 200 | ), fill='red' | | , outline = ' | blue') |
| c.create line(( | 300, | 230) , | (330, 200), | fill = |
| 'blue', dash = | (2,2)) |  |  |  |
| c.create line(( | 330, | 150), | (330, 200), | fill = |
| 'blue', dash = | (2,2)) |  |  |  |
| c.create line(( | 330, | 200), | (430, 200), | fill = |
| 'blue', dash = | (2,2)) |  |  |  |

230)

red'

180)

red'

(300,

outline

(330,

outline

window.mainloop()

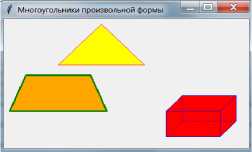


Рис. 28

Для рисования эллипсов и кругов в модуле tkinter используется метод create\_oval(). В качестве аргументов метода необходимо задать координаты гипотетического прямоугольника, в который вписывает­ся эллипс. Круг в данном случае представляет собой частный случай эллипса. Для его создания в качестве описывающего прямоугольника необходимо задать квадрат.

Пример задачи: написать программу, которая будет рисовать в графическом окне круг и эллипс.

Решение:

from tkinter import \*

#Создаем окно window = Tk()

window.title('Эллипс и круг')

#Создаем холст

c = Canvas(window, width=250, height=200) c.pack()

#Рисуем круг

c.create\_oval((50, 10), (150, 110), outline =

'red', width = 3)

#Рисуем эллипс

c.create\_oval(10, 120, 190, 190, fill='grey80',

outline='white')

window.mainloop()

И

Результат работы программы представлен на рисунке 29.

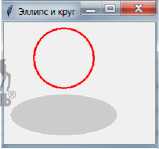


Рис. 29

Для создания более сложных фигур в модуле tkinter использу­ется метод create\_arc(). В зависимости от значения его аргумента style можно построить сектор (строится по умолчанию), сегмент (style = CHORD) или дугу (style = ARC). По аналогии с методом create\_oval() в качестве аргументов метода необходимо задать коор­динаты прямоугольника, в который вписана окружность (или эллипс), из которой «вырезается» сектор, сегмент или дуга. Значение аргумен­та start определяет начальное положение фигуры, значение аргумен­та extent задает угол поворота (можно задавать как положительные значения (отсчет против часовой стрелки), так и отрицательные зна­чения (отсчет по часовой стрелке)).

Пример задачи: написать программу, которая будет рисовать в графическом окне дугу размером 50о из точки 140о, сегмент размером 90о из точки 240о и два сектора размерами 60о из точки 0о и 30о из точки 160о В качестве фона для большей наглядности нарисовать круг серого цвета.

Решение:

from tkinter import \*

#Создаем окно window = Tk()

window.title('CeKTopa, дуги и сегменты')

#Создаем холст

c = Canvas(window, width=350, height=200) c.pack()

#Рисуем круг

c.create\_oval((10, 10), (190, 190), fill = 'light-

grey', outline = 'white')

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| #Рисуем | сектор размером 60 градусов из точки 0 | | | |
| градусов  c.create | arc( | (10, 10), (190, 190), | start = | 0, |
| extent = | О  VD | fill='blue') |  |  |
| #Рисуем | сектор размером 30 градусов | | из точки | 160 |
| градусов  c.create | arc( | (10, 10), (190, 190), | start = | 160, |
| extent = | О  со | fill='red') |  |  |
| #Рисуем | дугу | размером 50 градусов | из точки | 140 |
| градусов  c.create | arc( | " О ‘  (10, 10), (190, 190), | start = | 140, |
| extent = | О  ю  1 | style = ARC, |  |  |

outline='darkgreen', width = 4)

Рис. 30

**г**

#Рисуем сегмент размером градусов

с.create\_arc((10, 10), (190, 190), start =

extent = 90, style = CHORD, fill ='orange') window.mainloop()

градусов из точки

240

240,

Результат работы программы представлен на рисунке 30.



Для размещения текста в графическом окне в модуле tkinter ис­пользуется метод create\_text(). Обязательными аргументами метода являются координаты местоположения выводимого текста и сам текст (аргумент text). В качестве необязательных аргументов метода могут выступать:

* justify — определяет способ выравнивания текста относитель­но самого себя (CENTER — по центру, LEFT — по левой границе, RIGHT — по правой границе); по умолчанию в задаваемой точке графического окна располагается центр текстовой надписи;
* fill — задает цвет текста;
* font — задает тип шрифта (в виде строки);
* anchor (якорь) — используется для привязки текста к кон­кретному месту графического окна. Для значения этого аргумента используются обозначения сторон света. Например, для размеще­ния по указанной координате левой границы текста необходимо за­дать для якоря значение W (от англ. west — запад). Возможные другие значения аргумента anchor: N (север — вверху), E (во­сток — справа), S (юг — внизу), W (запад — слева), NE (северо­восток — вверху справа), SE (юго-восток — внизу справа), SW (юго-запад — внизу слева), NW (северо-восток — вверху слева). Если сторона привязки текста задается двумя буквами, то первая буква определяет вертикальную привязку (вверх или вниз от задан­ной координаты), а вторая — горизонтальную (влево или вправо от заданной координаты).

Пример задачи: написать программу, которая будет выводить на экран круговую диаграмму, отображающую структуру инвестици­онного портфеля, состоящего из двух активов: акций Лукойла (доля в портфеле составляет 20%) и акций Сбербанка (доля в портфеле со­ставляет 80%). Кроме самой диаграммы необходимо вывести пояс­няющий текст.

Решение:

from tkinter import \*

#Создаем окно window = Tk()

window.title('CTpyKTypa инвестиционного портфеля') #Создаем холст

c = Canvas(window, width=500, height=200) c.pack()

#Рисуем сектор размером 72 градуса из точки 0 градусов

c.create\_arc((10, 10), (190, 190), start = 0, ex­

tent = 72, fill='blue')

#Рисуем сектор размером 288 градусов из точки 72 градуса

c.create\_arc((10, 10), (190, 190), start = 72,

extent = 288, fill='red')

#Рисуем два прямоугольника

c.create\_rectangle((250,100),(300,120), fill = 'blue')

c.create\_rectangle((250,150),(300,170), fill = 'red')

#Добавляем текст

c.create\_text((350,50), text = "Круговая

диаграмма", justify = CENTER,

fill = "green", font = "Times 14 bold")

c.create\_text((325,110), text = "Лукойл - 20%",

anchor = W,

fill = "darkgreen", font = "Times 10 bold") c.create\_text ( (325, 160), text = "Сбербанк - 80%", anchor = W,

fill = "darkgreen", font = "Times 10 bold") window.mainloop()



Рис. 31

* 1. ИДЕНТИФИКАЦИЯ ГРАФИЧЕСКИХ ОБЪЕКТОВ. ИДЕНТИФИКАТОРЫ И ТЕГИ

В модуле tkinter существует два способа идентификации графи­ческих объектов, расположенных в графическом окне, — идентифи­каторы и теги. Идентификаторы являются уникальными для каждого объекта, т. е. не могут существовать два разных объекта с одинако­выми идентификаторами. Теги не являются уникальными, т. е. один и тот же тег может присутствовать в атрибутах разных объектов. Таким образом, использование тегов позволяет делать группировки объек­тов по тегам и, как следствие, при необходимости изменять свойства всей группы одновременно. При этом каждый отдельный графиче­ский объект может иметь как идентификатор, так и тег.

В качестве идентификаторов создаваемых графических объектов можно использовать числовые значения, которые возвращают все ме­тоды по созданию объектов. Эти идентификаторы можно присвоить каким-то переменным, которые в дальнейшем можно использовать для работы с этими объектами.

Пример задачи: реализовать возможность перемещения прямо­угольника в пределах графического окна с помощью стрелок на кла­виатуре. Для идентификации соответствующих клавиш клавиатуры использовать их идентификаторы (<Up>, <Down>, <Left>, <Right>). Для реализации перемещения использовать метод move(). Для пере­дачи объекта-прямоугольника в метод move() использовать его иден­тификатор, например, rect.

Решение:

from tkinter import \*

#Создаем окно window = Tk()

window.title('Идентификаторы объектов')

#Создаем холст

can = Canvas(window, width=400, height=200)

can.focus\_set() can.pack()

#Создаем прямоугольник

rect = can.create\_rectangle( (150, 75) , (250, 125), fill = 'blue')

can . bind (' <Up> ' , lambda event ^ can. move (rect, 0, —10)) #Смещение вверх на 10 пикселей can.bind('<Down>', lambda event: can.move(rect, 0, 10))#Смещение вниз на 10 пикселей

can.bind('<Left>', lambda event: can.move(rect, —25, 0))#Смещение влево на 25 пикселей can.bind('<Right>', lambda event: can.move(rect, 25, 0))#Смещение вправо на 25 пикселей window.mainloop()

Свойства созданных объектов можно изменять по ходу работы программы. Для этого в модуле tkinter используется метод itemconfig().

Для изменения местоположения объекта используется метод coord(). Если при вызове этого метода в качестве аргумента задать ему только идентификатор или тег объекта, то метод вернет текущие координаты объекта.

Пример задачи: реализовать возможность изменения цвета и размера прямоугольника при нажатии клавиши <Enter>.

Решение:

from tkinter import \*

#Создаем окно window = Tk()

window.title('Идентификаторы объектов')

#Создаем холст

can = Canvas(window, width=400, height=200) can.focus\_set()

can.pack()

#Создаем прямоугольник

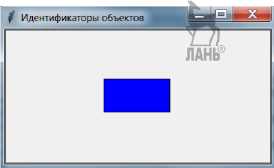
rect = can.create\_rectangle( (150, 75) , (250, 125), fill = 'blue')

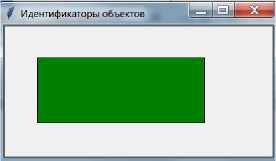
#Функция изменяет цвет и размер прямоугольника def change(event):

can.itemconfig(rect, fill='green') can.coords(rect, 50, 50, 300, 150)

can.bind('<Return>', change) window.mainloop()

Результат работы программы представлен на рисунке 32.



Рис. 32

В отличие от идентификаторов один и тот же тег можно присво­ить различным объектам. Фактически теги позволяют осуществлять группировку объектов в программе, что позволяет, например, одно­временно изменять свойства объектов, принадлежащих к одной груп­пе (имеющих одинаковый тег). Принципиальное отличие тега от идентификатора с точки типов данных заключается в том, что иден­тификатор объекта представляет собой целочисленную переменную, в то время как тег должен представлять собой строку.

Пример задачи: реализовать с помощью тега возможность од­новременного изменения цвета всех объектов при нажатии левой кнопки мыши.

Решение:

from tkinter import \*

#Создаем окно window = Tk()

window.title('Использование тегов')

#Создаем холст

can = Canvas(window, width=400, height=200) can.focus\_set()

can.pack()

#Создаем группу фигур под одним тегом group\_l circ = can.create\_oval((150,75),(250,125), fill = 'yellow', tag = "group\_l")

rect = can.create\_rectangle((50,25),(150,75), fill = 'green',tag = "group\_1")

#Функция изменяет цвет всех фигур из группы с

тегом group\_1

def change\_color(event):

can.itemconfig('group\_1', fill=,red') can.bind(,<Button-1>', change\_color) window.mainloop()

Результат работы программы представлен на рисунке 33.

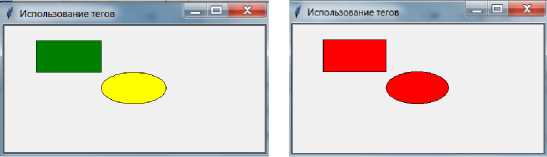


Рис. 33

Удаление объектов из графического окна осуществляется с по­мощью метода deleteO- В качестве аргумента методу необходимо пе­редать идентификатор удаляемого объекта или его тег. Для удаления из графического окна всех созданных объектов в качестве аргумента данному методу необходимо передать константу ALL.

С помощью метода tag bind() можно осуществить привязку определенного события (например, нажатия кнопки мыши) к опреде­ленному графическому объекту в графическом окне. Такой подход позволяет использовать одно и то же событие для обращения к раз­личным частям окна.

Пример задачи: реализовать возможность замены изображения графического объекта на его текстовое название при щелчке левой кнопки мыши по выбранному объекту.

Решение:

from tkinter import \*

#Создаем окно window = Tk()

window.title('Использование идентификаторов') #Создаем холст

can = Canvas(window, width=400, height=200) can.pack()

#Создаем фигуры

circ = can.create\_oval((150,75),(250,125), fill = 'yellow')

rect = can.create\_rectangle((50,25),(150,75), fill = 'green')

trial = can.create\_polygon((350,20), (310,80)

(390,80),fill = "blue", outline = "yellow") #Функции меняют изображение фигур на текст def change\_circ(event): can.delete(circ)

-fe)'

can.create\_text((200,100) , text='Эллипс'

fill = "red")

def change\_rect(event): can.delete(rect) can.create\_text((100,50), text=,Прямоугольник,, fill = "red")

def change\_trial(event): can.delete(trial)

can.create\_text((350,40), text='Треугольник,, fill = "red")

can.tag\_bind(circ, '<Button-1>', change\_circ) can.tag\_bind(rect, '<Button-1>', change\_rect) can.tag\_bind(trial, '<Button-1>', change\_trial)

window.mainloop()

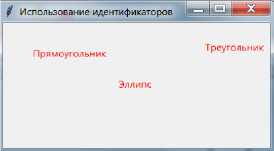
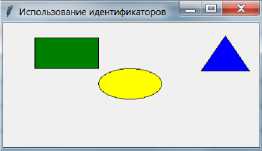
Результат работы программы представлен на рисунке 34.

Рис. 34

1. СОЗДАНИЕ АНИМАЦИИ

С помощью методов класса Canvas в графическом окне можно реализовывать эффект движения изображений объектов.

К основным методам, используемым для создания анимации, относятся методы move() и after(). Первый из них реализует переме­щение объекта в заданную точку графического окна, а второй —

вызов функции, переданной ему в качестве второго аргумента через заданные интервалы времени, задаваемые его первым аргументом.

В качестве вспомогательных функций при создании анимации, как правило, используются рассмотренная выше функция coords(), возвращающая список текущих координат объекта, а также функции winfo\_reqwidth() и winfoheight(), возвращающие ширину и высоту графического окна соответственно.

Пример задачи: реализовать простую анимацию, представля­ющую собой перемещение круга от левой границы графического ок­на до его правой границы. По достижении правой границы объект (круг) должен исчезать из окна.

Решение:

from tkinter import \*

#Создаем окно window = Tk()

window.title('Простая анимация')

#Создаем холст

can = Canvas(window, width=400, height=200) can.pack()

width = can.winfo\_reqwidth()-4 #Вычисляем ширину холста

#Создаем круг

ball = can.create\_oval((0,100),(50,150), fill = 'blue')

print(can.coords(ball)[0],can.coords(ball)[1],

can.coords(ball)[2])

#Функция для реализации движения def motion():

can.move(ball, 5, 0) 1

if can.coords(ball)[2] < width: window.after(10, motion) else:

can.delete(ball)

motion()

window.mainloop()

Комментарии к программе.

Для получения значения ширины окна используется метод winfo\_reqwidth(). От полученного значения отнимаются четыре пик­селя, которые занимают две границы холста (левую и правую).

Метод move() увеличивает координату x объекта ball, передан­ного ему в качестве первого аргумента, на один пиксель.

Метод coords() возвращает список текущих координат объекта ball, переданного ему в качестве аргумента. Третий элемент этого списка содержит значение координаты x объекта ball.

Метод after() в качестве аргументов получает интервал времени (в данном случае это 10 миллисекунд) и функцию, которую нужно вызывать через этот интервал (функция motion()).

1. ДОБАВЛЕНИЕ ИЗОБРАЖЕНИЙ ИЗ ФАЙЛОВ

Модуль tkinter позволяет работать с файлами изображений в форматах GIF и PGM/PPM, которые могут быть выведены на видже­тах Label, Text, Button и Canvas. Для этих целей используется класс Photoimage.

Для создания объекта-изображения, как и в случае со всеми остальными виджетами, используется конструктор Photoimage() это­го класса. В качестве аргумента конструктору необходимо указать имя файла с изображением в виде file = ‘имя файла’.

Для уменьшения размера изображения можно воспользоваться методом subsample(), передав ему в качестве аргументов параметры дискретизации по горизонтали и по вертикали в виде x = значение и у = значение. Например, значения x = 2, y = 2 приведут к отбрасыва­нию каждого второго пикселя — результатом будет уменьшение изображения в два раза по сравнению с оригиналом.

Для увеличения размера изображения в классе Photoimage су­ществует и обратный метод zoom(), увеличивающий размер изобра­жения в соответствии с переданными ему аргументами x и y.

После создания объекта-изображения его можно добавлять на виджеты с помощью передачи аргумента image в соответствующих конструкторах.

У объектов-виджетов Text существует метод image\_create(), с помощью которого изображение встраивается в текстовое поле. Дан­ный метод принимает два аргумента: первый — для определения по­зиции размещения (например, ‘1.0’ указывает первую строку и пер­вый символ), а второй представляет собой ссылку на само изображе­ние в виде аргумента image.

Объекты класса Canvas имеют аналогичный метод create\_image(), тоже принимающий два аргумента, только первый ар­гумент, отвечающий за расположение изображения, представлен в виде пары координат (x, y), которые определяют координаты точки графического окна, куда помещается изображение.

Замечание. Несмотря на то что методы image\_create() класса Text и create\_image() класса Canvas очень похожи, они все-таки от­личаются друг от друга.

Пример задачи: реализовать размещение графического изобра­жения из внешнего файла на различных виджетах.

Решение:

from tkinter import \* window = Tk()

window.title( 'Работа с изображением' )

#Создаем объект-изображение

img = PhotoImage( file = 'python.gif' )

#Уменьшаем полученное изображение в два раза small\_img = PhotoImage.subsample( img , x = 2 ,

У = 2 )

#Создаем метку с изображением

label = Label ( windc^jj^® , image = img , bg =

'yellow' )

#Создаем кнопку с изображением

btn = Button( window , bg = 'red', image =

small\_img )

#Создаем текстовое поле с изображением

txt = Text( window , width = 25 , height = 7, fg =

'darkgreen' )

txt.image\_create( '1.0' , image = small\_img )

txt.insert( '1.1', 'Python is Fun!' )

#Создаем холст с изображением

can = Canvas( window , width = 100 , height = 100 , bg = 'cyan' )

can.create\_image( ( 50 , 50 ), image = small\_img ) can.create\_line( 0 , 0 , 100 , 100, width = 25 ,

fill = 'yellow' )

#Размещаем созданные виджеты в окне приложения

label, pack ( side = TOP ) 1AHI

btn.pack( side = LEFT , padx = 10 )

txt.pack( side = LEFT )

can.pack( side = LEFT, padx = 10 )

window.mainloop()

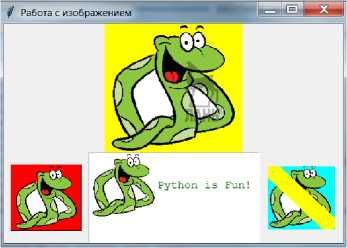


Рис. 35



1. СПЕЦИАЛИЗИРОВАННЫЕ МЕНЕДЖЕРЫ РАЗМЕЩЕНИЙ

Менеджер размещения pack() представляет собой наиболее про­стой, но не самый эффективный метод для конфигурации графиче­ских объектов в графическом окне. Он используется, как правило, для создания относительно простых графических интерфейсов.

Для создания достаточно сложных графических интерфейсов используются другие менеджеры размещений.

* 1. МЕНЕДЖЕР РАЗМЕЩЕНИЙ GRID()

Менеджер размещений grid() (от англ. grid — сетка) позволяет располагать виджеты в ячейках таблицы. Способ размещения видже­тов в виде таблицы представляет собой более гибкий и удобный под­ход при разработке относительно сложных интерфейсов. При этом отпадает необходимость в использовании большого количества фреймов, без которых не обходится разработка интерфейса с помо­щью менеджера размещений pack().

Проектирование графического интерфейса с помощью метода grid() происходит по следующей схеме. Родительский контейнер (графическое окно) условно разделяется на табличные ячейки. До­ступ к каждой ячейки такой таблицы осуществляется по её адресу. Адрес каждой ячейки представляет собой сочетание номера строки и номера столбца. И строки, и столбцы нумеруются начиная с нуля. Ячейки можно объединять как по вертикали, так и по горизонтали. При объединении ячеек в качестве адреса объединенной ячейки вы­ступает адрес первой ячейки.

Для размещения виджета в ячейке используются аргументы row и column. В качестве значений этих аргументов выступают номера строк и столбцов соответственно. Для объединения ячеек по горизон­тали используется аргумент columnspan, которому в качестве значе­ния задается количество объединяемых ячеек. Для объединения ячеек по вертикали используется аргумент rowspan.

Для установки внешних и внутренних отступов (как и при ис­пользовании метода раск()) можно использовать аргументы padx, pady, ipadx и ipady.

Аргумент sticky (от англ. — липкий) позволяет размещать ви­джет в определенном месте ячейки. По аналогии с аргументом anchor аргумент sticky может принимать значения направлений сторон све­та. Например, для случая sticky = SE виджет будет расположен в пра­вом нижнем углу ячейки. Виджеты можно растягивать по всему объ­ему ячейки (sticky = N+S+W+E) или только вдоль одной из её осей (N+S или W+E). Очевидно, что эти эффекты будут заметны только тогда, когда размеры виджета не превышают размеров ячейки.

Существует возможность делать виджеты невидимыми. Для это­го можно использовать методы grid\_remove() и grid\_forget(). Эти методы отличаются по возможности запоминания предыдущего по­ложения виджета: метод grid\_remove() запоминает его, а метод grid\_forget() — нет. Поэтому для отображения виджета в прежней ячейке после использования метода grid\_remove() достаточно вы­звать метод grid() без аргументов. Применение метода grid\_forget() влечет за собой необходимость переконфигурации положения вид­жета.

Скрытие виджетов, как правило, используют тогда, когда появ­ление виджета в одной части интерфейса обусловлено действиями пользователя в другой его части.

Пример задачи: реализовать размещение различных виджетов в окне приложения с помощью менеджера размещений grid().

Решение:

from tkinter import \* window = Tk()

window.title( 'Электрички' )

window.resizable(0,0) #Деактивируем кнопку

изменения размеров окна

#Размещаем в ячейке (0,0) метку с текстом Label(text="Введите номер станции

назначения:").grid(row=0, column=0, sticky=N+W, pady=10, padx=10)

#Размещаем в ячейке (0,1) однострочное текстовое поле

station\_numb = Entry(width = 3)

station\_numb .grid(row=0, column=l,sticky=N+W,

pady=10, padx=10 )

#Размещаем в ячейке (0,2) поле со списком #объединяем ячейки (0,2),( 0,3) и (0,4) station\_list = Listbox(window, width = 50) station\_list.grid(row=0, column=2, columnspan = 5, padx=10)

#Размещаем в ячейке (1,0) метку с текстом

Label(text="Минимальное время в пути до станции №:").grid(row=1, column=0, sticky=W, pady=10, padx=10)

#Размещаем в ячейке (1,1) однострочное текстовое поле

station\_numb = Entry(width = 3)

station\_numb.grid(row=1, column=1, sticky=W,

pady=10, padx=10 )

#Размещаем в ячейке (1,2) метку с текстом Label(text="составляет").grid(row=1, column=2,

sticky=W, pady=10, padx=10)

#Размещаем в ячейке (1,3) однострочное текстовое поле

station\_numb = Entry(width = 5)

station\_numb.grid(row=l, column=3, sticky=W,

pady=10, padx=10 ) o.

#Размещаем в ячейке (1,4) метку с текстом Label(text="минут").grid(row=1, column=4,

sticky=W, pady=10, padx=10)

#Размещаем в ячейке (2,0) кнопку "Подобрать вариант"

btn\_1 = Button(window, text = "Подобрать вариант", bg = "lightgreen")

btn\_1.grid(row=2, column=0, sticky=S, pady=30, padx=10)

#Размещаем в ячейке (2,2) кнопку "Очистить" btn\_2 = Button(window, text = "Очистить", bg =

|  |  |  |
| --- | --- | --- |
| "orange", width = 20)  btn 2.grid(row=2, column=2, | sticky=S, | pady=30, |
| padx=10)  #Размещаем в ячейке (2,4) кнопку "Выход" btn 3 = Button(window, text = "Выход", bg | | = "red", |
| width = 20)  btn 3.grid(row=2 , column=4, | sticky=S, | pady=30, |
| padx=10)  window.mainloop() |  |  |

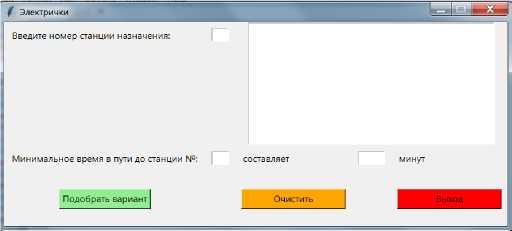


Рис. 36

* 1. МЕНЕДЖЕР РАЗМЕЩЕНИЙ Р1\_АСЕ()

Менеджер размещений р1асе() реализует размещение виджетов по координатам. При использовании этого метода для задания положения виджета используются или абсолютные значения (в пикселях), или от­носительные значения (в долях родительского окна). Для задания раз­мера самого виджета также можно использовать эти два способа.

Основные аргументы метода place():

* anchor — определяет область виджета, для которой задаются координаты; аргумент принимает следующие значения: N, NE, E, SE, SW, W, NW или CENTER; значение по умолчанию — NW (левый верхний угол);
* relwidth, relheight (относительные ширина и высота) — опре­деляют размер виджета в долях родительского виджета;
* relx, rely — определяют относительную позицию виджета по отношению к родительскому виджету ((0;0) — в левом верхнем углу, (1;1) — в правом нижнем углу, (0,5;0,5) — по центру, (0,25;0,25) — по центру левого верхнего квадранта, (0,75;0,25) — по центру право­го верхнего квадранта, (0,25;0,75) — по центру левого нижнего квад­ранта, (0,75;0,75) — по центру правого нижнего квадранта);
* width, height — абсолютные размеры виджета в пикселях; значения по умолчанию устанавливаются равными естественному размеру виджета (устанавливается при его создании и конфигуриро­вании);
* x, y — абсолютные значения координат позиции в пикселях; значения по умолчанию — x = 0, y = 0.

Пример задачи: реализовать варианты абсолютного и относи­тельного позиционирования кнопки с помощью метода place().

Решение:

from tkinter import \* window = Tk()

window.geometry("400x200" ) window.title( 'Метод placet)' )

#Размещаем метку по абсолютным координатам Label(text="Введите номер станции

назначения:").р1асе(х=50, у = 20)

#Размещаем однострочное текстовое поле по

абсолютным координатам station\_numb = Entry(width = 3) station\_numb.place(x = 315, y = 20)

#Размещаем кнопку "Подобрать вариант" по

относительным координатам

btn = Button(window, text = "Подобрать вариант", bg = "lightgreen") btn.place(relx=0.25, rely=0.75)

window.mainloop()

Результат работы программы представлен на рисунках 37 и 38.

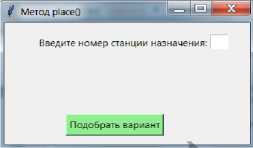
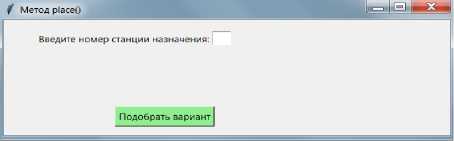


Рис. 37

Различия между двумя способами размещения виджетов прояв­ляются при изменении размеров окна. В этом случае виджеты, позиция которых была установлена с помощью абсолютных координат, не из­меняют своего положения в окне, в отличие от виджетов, позиция ко­торых была установлена с помощью относительных координат.



Этот же эффект проявляется и при задании размеров виджетов. Если размеры виджетов заданы в относительных единицах, то они будут изменяться при изменении размеров родительского виджета. Поэтому в том случае, если виджет не должен изменять своих разме­ров, последние нужно задавать в абсолютных координатах или вооб­ще их не указывать (тогда размеры виджета будут равны естествен­ным размерам).

Комбинируя различные варианты позиционирования и установ­ки размеров виджетов, можно получать интересные эффекты, а также неожиданные спецэффекты. Поэтому при использовании метода place() требуется определенная осторожность и внимательность.

Выбор метода place() для размещения виджетов является обос­нованным в тех случаях, когда разрабатывается достаточно сложный интерфейс, а изменение размеров окна не подразумевается. В таких ситуациях метод place() позволяет выполнить точную настройку и создать наиболее аккуратный интерфейс.

В случае необходимости реализации запрета на изменение раз­меров окна программными средствами для деактивации кнопки изме­нения размеров окна можно использовать метод resizeable() с нуле­выми значениями обоих аргументов.



ПРАКТИЧЕСКИЕ ЗАДАНИЯ К ГЛАВЕ 1

1. Составьте программу, которая будет представлять окно с расположенными в нем семью кнопками разного цвета, соответству­ющими семи цветам радуги. При нажатии на каждую из кнопок на экран должно выводиться название цвета (например, фиолетовый) и его шестнадцатеричный код (#7d00ff). Цвет надписи должен соответ­ствовать цвету выбранной кнопки.

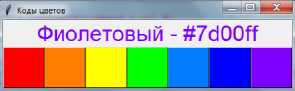
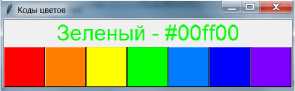
Возможный результат работы программы представлен на рисун­ке 39.

Рис. 39

1. Составьте объектно-ориентированную программу, в которой объектами будут выступать блоки, состоящие из поля для ввода тек­ста (виджет Entry), кнопки (виджет Button) и метки для вывода тек­ста (виджет Label).

Комментарии к программе:

1. создайте в программе класс, например, с именем Group\_Widgets, атрибутами которого будут три разных виджета (Entry, Button и Label);
2. включите в состав методов класса следующие методы:

* конструктор; ЛАНЬ®
* метод, который будет считывать введенные данные из поля данных (виджет Entry), сортировать их по возрастанию и выводить результат в виджете Label;
* метод, который будет считывать введенные данные из поля данных (виджет Entry), сортировать их по убыванию и выводить ре­зультат в виджете Label;
* метод, который будет считывать введенные данные из поля данных (виджет Entry), перемешивать их случайным образом и вы­водить результат в виджете Label;
* метод, который будет связывать кнопку с одним из вышепере­численных методов;

1. в основной программе создайте три объекта класса Group\_Widgets, каждый из которых будет преобразовывать введен­ные пользователем данные одним из трех вышеперечисленных спо­собов (сортировать по возрастанию, сортировать по убыванию, пере­мешивать случайным образом) и протестируйте работу программы.

Возможный результат работы программы представлен на рисун­ке 40.

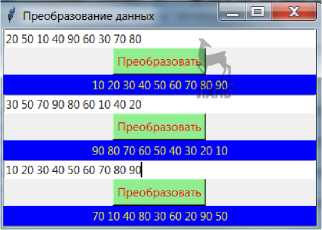


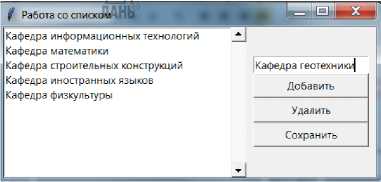
Рис. 40

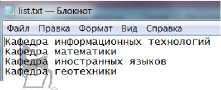
1. Составьте программу с пользовательским графическим ин­терфейсом, которая будет позволять:

* добавлять в список элемент, введенный пользователем в тек­стовое поле;
* удалять из списка выбранные пользователем элементы;
* сохранять выбранные пользователем элементы в файл, при этом каждый сохраняемый элемент должен записываться с новой строки.

Комментарии к программе.

1. создайте в программе один список (Listbox), одно поле ввода (Entry) и три кнопки (Button), например, с именами «Добавить», «Удалить» и «Сохранить»;
2. добавьте в список вертикальную полосу прокрутки (Scrollbar);
3. для каждой из кнопок напишите обработчики событий;
4. при нажатии кнопки «Добавление» введенная пользователем строка должна добавляться в конец списка и исчезать из поля ввода;
5. в функции по удалению выбранных элементов вначале преоб­разуйте кортеж выбранных элементов в список, а затем с помощью метода revers() измените порядок следования элементов в нем на противоположный для того, чтобы удаление элементов списка проис­ходило с конца списка (в противном случае удаление элемента будет приводить к изменению индексов всех следующих за ним элемен­тов — в результате программа будет работать неправильно);
6. в функции для записи выбранных элементов в файл, преобра­зуйте кортеж строк-элементов списка, который вернет метод get(), в одну строку с помощью метода join() через разделитель “\n”.

Возможный результат работы программы представлен на рисун­ке 41.

ЛАНЬ Рис. 41

|  |  |
| --- | --- |
| % Работа со списком i |  |
| Кафедра информационных технологий | I]- |
| Кафедра математики |  |
| IM Ш А1ч 1«Ы!|ШЯ!1! 1!1 !Ы! 1« ■ ДиШгИИУУШЦ | 1 |
| Кафедра иностранных языков | Добавить |
|  |
| Кафедра геотехники | Удалить |
|  | Сохранить |
|  | d |

|  |  |
| --- | --- |
| 4 Рябпт л ГП ГПИГК-ПМ - — Ш § ^ ^ f | |
| Кафедра информационных технологий ж 1 Кафедра математики  ■ Кафедра иностранных языков |  Кафедра геотехники Добавить  Удалить  Сохранить  d |  |

1. Составьте программу, имитирующую интерфейс добавления товаров в корзину.

Комментарии к программе:

1. создайте в программе два списка (один с полосой прокрутки, другой без нее) и три кнопки;
2. в первом списке отобразите список товаров, заданный про­граммно, а второй список должен быть изначально пустым;
3. при нажатии кнопки «Добавить в корзину» товар должен пе­ремещаться из одного списка в другой, при этом каждый новый товар должен добавляться в конец списка;
4. при нажатии на кнопку «Удалить из корзины» товар должен удаляться из второго списка;
5. при нажатии кнопки «Оформить заказ» информация о вы­бранных товарах и их количестве должна записываться в файл.

Возможный результат работы программы представлен на рисун­ке 42.



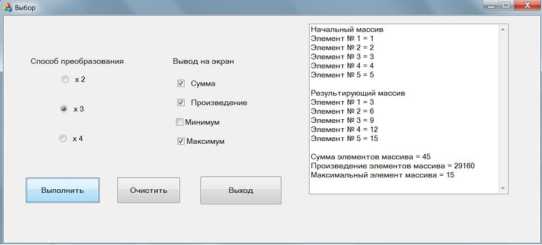
Ваш заказ:

Самоучитель Python - 3 шт. Самоучитель Java - 2 шт. Самоучитель C++ - 1 шт.

Рис. 42

5. Составьте программу, которая будет преобразовывать задан­ный целочисленный массив из пяти элементов разными способами в зависимости от выбранного переключателя, а именно, увеличи­вать каждый элемент исходного массива в 2, 3 или 4 раза. В зависи­мости от выбранного флажка на экран должны выводиться различ­ные итоговые результаты, а именно, сумма, произведение всех эле­ментов нового массива, минимальный и максимальный его элемент.

Возможный результат работы программы представлен на рисун­ке 43.



|  |  |  |  |
| --- | --- | --- | --- |
| f Добавление товаров в корзину • | | 1 |  |
| , Самоучитель Java | ■» | ЛАНь | Самоучитель Java |
|  |  |  | Самоучитель Java |
| Самоучитель Python Самоучитель C# |  | Добавить в корзину | Самоучитель Python Самоучитель Python |
| Самоучитель JavaScript Самоучитель РНР Самоучитель FORTRAN Самоучитель Rubi Самоучитель Go Самоучитель Objective-С |  | Удалить из корзины | Самоучитель Python Самоучитель C++ |
|  | Оформить заказ |  |
| d |  |  |

1. Составьте программу с графическим пользовательским ин­терфейсом для работы с внешними файлами.

Комментарии к программе:

1. в программе создайте две кнопки и одно многотекстовое поле;
2. при нажатии кнопки «Открыть» на экране должно появляться диалоговое окно для открытия файла;
3. после выбора файла его содержимое должно отображаться в текстовом поле;
4. при нажатии кнопки «Сохранить» должно открываться диа­логовое окно для сохранения информации в файл; после задания имени файла информация из текстового поля должна записываться в указанный файл;
5. добавьте в программу код обработки исключений, которые могут генерироваться в случае, если диалоговые окна были закрыты без выбора или указания имени файла; при этом в случае генерации исключения на экран должно выводиться диалоговое окно с соответ­ствующим сообщением о том, что файл не загружен или не сохранен;
6. добавьте в интерфейс кнопку «Очистить», при нажатии на которую будет удаляться вся информация из текстового поля; при этом перед удалением пользователь должен подтвердить свои наме­рения через соответствующее диалоговое окно.
7. Реализуйте еще одну версию предыдущей программы для случая выполнения всей функциональности с помощью меню. Ко­манду очистки текстового поля поместите в контекстное меню.



ПРАКТИЧЕСКИЕ ЗАДАНИЯ К ГЛАВЕ 2

1. Составьте программу, которая будет позволять изменять раз­меры многострочного текстового поля.

Комментарии к программе:

1. размеры многострочного текстового поля (Text) должны определяться значениями, вводимыми пользователем в однострочные текстовые поля (Entry);
2. изменение размера должно происходить при совершении сле­дующих событий:

* нажатии соответствующей кнопки;
* нажатии на клавиатуре клавиши <Enter>;

1. цвет фона экземпляров Entry должен быть светло-серым

(lightgrey), когда поле не в фокусе, и белым (white), когда поле в фо­кусе.

Возможный результат работы программы представлен на рисун­ке 44.



Рис. 44

1. Составьте программу с графическим пользовательским ин­терфейсом, представляющую собой системную утилиту, позволяю­щую проводить манипуляции с файлами.

Комментарии к программе:

1. программа должна предоставлять пользователю следующие возможности:

* выводить на экран список доступных директорий;
* выводить на экран список файлов из выбранной директо­рии;
* давать пользователю производить определенные действия с файлами, например:
* дублировать все файлы в текущей директории;
* дублировать конкретный файл;
* удалять дубликаты файлов из директории;
* удалять пустые директории;
* удалять из конкретной директории файлы определенного типа (с определенным расширением);
* переименовывать какие-либо файлы по выбранному при­знаку;
* переименовывать конкретный файл;
* перемещать файлы из одной директории в другую;

1. набор возможных действий с файлами может быть произ­вольным, но не менее трех;
2. в программе необходимо использовать как можно большее количество функций и переменных из модулей os, sys, shutil, psutil;
3. внешний вид интерфейса — произвольный.

Возможный результат работы программы представлен на рисун­ках 45-47.

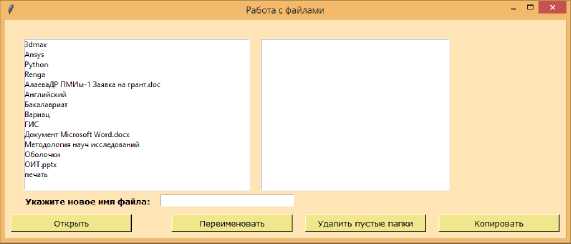
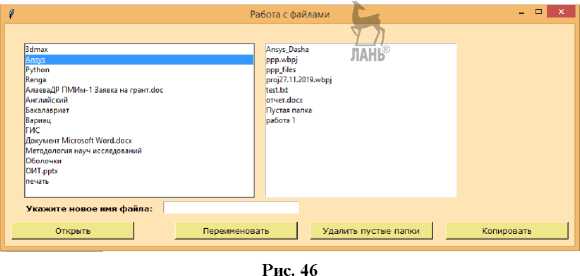


Рис. 45



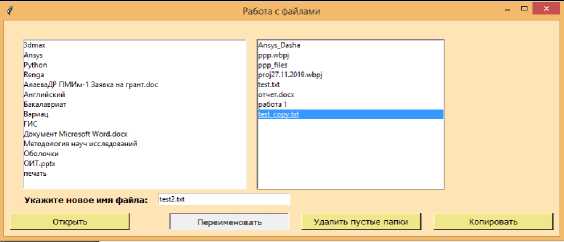


Рис. 47





ПРАКТИЧЕСКИЕ ЗАДАНИЯ К ГЛАВЕ 3

1. Составьте программу с графическим пользовательским ин­терфейсом, которая будет отображать на экране структуру инвести­ционного портфеля клиента в виде круговой диаграммы в соответ­ствии со своим вариантом задания.

Возможный результат работы программы представлен на рисун­ке 48.



Рис. 48

2. Составьте программу с графическим пользовательским ин­терфейсом, которая будет отображать на экране структуру инвести­ционного портфеля клиента в виде столбчатой диаграммы в соответ­ствии со своим вариантом задания.

Возможный результат работы программы представлен на рисун­ке 49.



Рис. 49

3. Составьте программу с графическим пользовательским ин­терфейсом и элементами анимации. Тема произвольная.

Возможный результат работы программы представлен на рисун­ке 50.

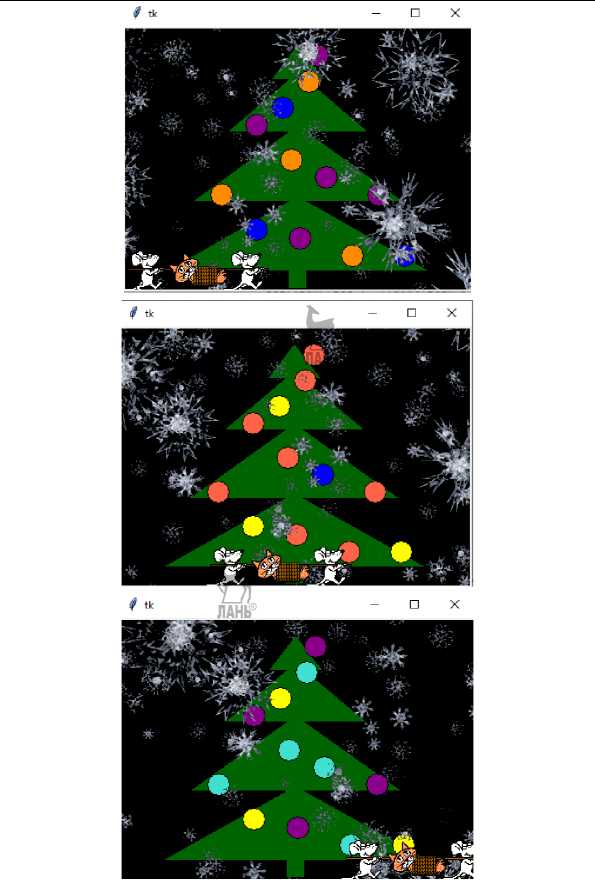


Рис. 50

ПРАКТИЧЕСКИЕ ЗАДАНИЯ К ГЛАВЕ 4

Составьте программу с графическим пользовательским интер­фейсом, которая по заданному расписанию движения электричек вы­числяет минимальное время, за которое пассажир сможет добраться до нужной ему станции.

Вид интерфейса — произвольный.

Менеджер размещения — в соответствии со своим вариантом.

Набор виджетов — в соответствии со своим вариантом задания.

Комментарии к программе.

Программа должна:

* выводить на экран полный список станций на маршруте;
* выводить на экран расписание электричек;
* запрашивать у пассажира номер нужной ему станции;
* выводить на экран список электропоездов, с помощью кото­рых пассажир сможет добраться до выбранной им станции и время в пути;
* выводить на экран минимальное время, за которое пассажир сможет добраться до нужной ему станции;
* обрабатывать ошибки ввода данных пользователем.

Возможный результат работы программы представлен на рисун­ке 51.



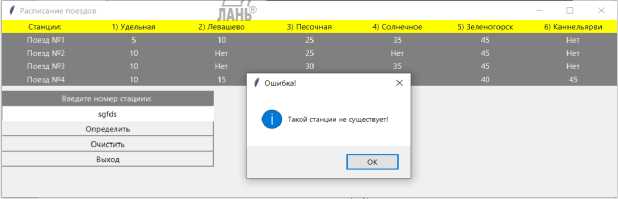


Рис. 51

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

1. Седжвик, Р. Программирование на языке Python : учебный курс / Р. Седжвик, К. Уэйн, Р. Дондеро. — СПб. : Альфа-книга, 2017. — 736 с.
2. Букунов, С. В. Основы программирования на языке Python: учебное пособие / С. В. Букунов, О. В. Букунова. — СПб. : СПбГАСУ, 2019. — 254 с.
3. Букунов, С. В. Объектно-ориентированное программирование на языке Python : учеб. пособие / С. В. Букунов, О. В. Букунова. — СПб. : СПбГАСУ, 2020. — 119 с.





ОГЛАВЛЕНИЕ

[Введение 3](#bookmark3)

1. [Модуль tkinter. Основные виджеты. Основные методы 5](#bookmark4)
   1. [Краткие сведения о библиотеке Tk и модуле tkinter 5](#bookmark5)
   2. [Основные этапы создания оконного интерфейса 5](#bookmark6)
   3. [Текстовое поле. Виджет Label. Метод pack() 7](#bookmark7)
   4. [Группировка виджетов. Виджет Frame 10](#bookmark10)
   5. [Управление работой приложения с помощью кнопок. Виджет Button 13](#bookmark12)
   6. [Вывод сообщений с помощью диалоговых окон. Модуль messagebox 19](#bookmark14)
   7. [Диалоговые окна для работы с файлами. Модуль filedialog 21](#bookmark16)
   8. [Прием данных от пользователя. Виджет Entry 25](#bookmark18)
   9. [Работа с многострочным текстом. Виджет Text 27](#bookmark19)
   10. Использование полосы прокрутки. Виджет Scrollbar 30
   11. [Выбор из списка. Виджет, Listbox 32](#bookmark26)
   12. [Использование переключателей. Виджет Radiobutton 34](#bookmark28)
   13. Работа с флажками. Виджех/Checkbutton 37
   14. [Создание меню. Виджет Menu 40](#bookmark32)
2. [Обработка событий 45](#bookmark34)
   1. [Связывание виджетов с событиями и действиями. Метод bind() 45](#bookmark35)
   2. [Виды событий 47](#bookmark38)
3. [Создание графических изображений и анимации 51](#bookmark39)
   1. [Создание графических примитивов 51](#bookmark40)
   2. [Идентификация графических объектов. Идентификаторы и теги 59](#bookmark45)
   3. [Создание анимации 63](#bookmark47)
   4. [Добавление изображений из файлов 65](#bookmark50)
4. [Специализированные менеджеры размещений 68](#bookmark51)
   1. [Менеджер размещений grid() 68](#bookmark52)
   2. [Менеджер размещений р1асе() 71](#bookmark54)

[Практические задания к главе 1 74](#bookmark56)

[Практические задания к главе 2 ДАНЬ 79](#bookmark58)

[Практические задания к главе 3 82](#bookmark60)

[Практические задания к главе 4 84](#bookmark62)

[Рекомендуемая литература 85](#bookmark63)

Сергей Витальевич БУКУНОВ,

Ольга Викторовна БУКУНОВА

РАЗРАБОТКА ПРИЛОЖЕНИЙ С ГРАФИЧЕСКИМ  
ПОЛЬЗОВАТЕЛЬСКИМ ИНТЕРФЕЙСОМ НА ЯЗЫКЕ  
PYTHON

*Учебное пособие*

Зав. редакцией

литературы по информационным технологиям и системам связи  
О. Е. Гайнутдинова  
Ответственный редактор Е. О. Сапарова  
Подготовка макета Е. С. Илларионова  
Корректор Т. А. Быченкова  
Выпускающий В. А. Иутин

ЛР № 065466 от 21.10.97  
Гигиенический сертификат 78.01.10.953.П.1028  
от 14.04.2016 г., выдан ЦГСЭН в СПб  
Издательство «ЛАНЬ»  
[lan@lanbook.ru](mailto:lan@lanbook.ru); [www.lanbook.com](http://www.lanbook.com)  
196105, Санкт-Петербург, пр. Юрия Гагарина, д. 1, лит. А  
Тел./факс: (812) 336-25-09, 412-92-72  
Бесплатный звонок по России: 8-800-700-40-71



Подписано в печать 12.10.22.

Бумага офсетная. Гарнитура Школьная. Формат 84x108 V32.  
Печать офсетная/цифровая. Усл. п. л. 4,62. Тираж 30 экз.

Заказ № 1418-22.

Отпечатано в полном соответствии  
с качеством предоставленного оригинал-макета  
в АО «Т8 Издательские Технологии».  
109316, г. Москва, Волгоградский пр., д. 42, к. 5.