ФЕДЕРАЛЬНОЕ АГЕНТСТВО ВОЗДУШНОГО ТРАНСПОРТА
Троицкий авиационный технический колледж – филиал МГТУ ГА

Отделение «Программирование в компьютерных системах»

Методическое пособие № 3
по подготовке и проведению учебной практики
«Графические интерфейсы Python»

Тема «Библиотека TKinter»

Специальность 09.02.03

г. Троицк, 2017

Содержание

1. Краткое введение в TKinter.

Как справочник по компонентам библиотеки TKinter можно использовать ресурсы:

https://ru.wikibooks.org/wiki/GUI_Help/Tkinter_book

https://ru.wikiversity.org/wiki/Курс_по_библиотеке_Tkinter_языка_Python

В многообразии программ, которые пишут программисты, выделяют приложения с графическим пользовательским интерфейсом (GUI). При создании таких программ становятся важными не только алгоритмы обработки данных, но и разработка для пользователя программы удобного интерфейса, взаимодействуя с которым, он будет определять поведение приложения.

Основная черта любой программы с графическим интерфейсом - интерактивность. Программа не просто что-то считает (в пакетном режиме) от начала своего запуска до конца: ее действия зависят от вмешательства пользователя.

Фактически, графическое приложение выполняет бесконечный цикл обработки событий.

Программа, реализующая графический интерфейс, событийно-ориентированная. Она ждет от интерфейса событий, которые и обрабатывает сообразно своему внутреннему состоянию.

Эти события возникают в элементах графического интерфейса (виджетах) и обрабатываются прикрепленными к этим виджетам обработчиками. Сами виджеты имеют многочисленные свойства (цвет, размер, расположение), выстраиваются в иерархию принадлежности (один виджет может быть хозяином другого), имеют методы для доступа к своему состоянию.

Расположением виджетов (внутри других виджетов) ведают так называемые **менеджеры расположения**. Виджет устанавливается на место по правилам менеджера расположения. Эти правила могут определять не только координаты виджета, но и его размеры. В Tk имеются три типа менеджеров расположения: простой упаковщик (pack), сетка (grid) и произвольное расположение (place).

Некоторые виджеты в программе с GUI должны быть взаимосвязаны определенным образом. Например, полоса прокрутки может быть взаимосвязана с текстовым виджетом: при использовании полоски текст в виджете должен двигаться, и наоборот, при перемещении по тексту полоска должна показывать текущее положение. Для связи между виджетами в Tk используются переменные, через которые виджеты и передают друг другу параметры.

Для языка программирования Python такие виджеты включены в специальную библиотеку — tkinter. Если ее импортировать в программу (скрипт), то можно пользоваться ее компонентами, создавая графический интерфейс.

Последовательность шагов при создании графического приложения имеет свои особенности. Программа должна выполнять свое основное назначение, быть удобной для пользователя, реагировать на его действия.

Основные этапы при программировании программы с GUI:

1. Импорт библиотеки
2. Создание главного окна
3. Создание виджет
4. Установка их свойств
5. Определение событий
6. Определение обработчиков событий
7. Расположение виджет на главном окне
8. Отображение главного окна

1.1. Импорт модуля tkinter

Как и любой модуль, tkinter в Python можно импортировать двумя способами: командами

```
import tkinter
```

или

```
from tkinter import *
```

В дальнейшем мы будем пользоваться только вторым способом, т. к. это позволит не указывать каждый раз имя модуля при обращении к объектам, которые в нем содержатся. Следует обратить внимание, что в версии Python 3 имя модуля пишется со строчной буквы (tkinter), хотя в более ранних версиях использовалась прописная (Tkinter). Итак, первая строка программы должна выглядеть так:

```
from tkinter import *
```

1.2. Создание главного окна

В современных операционных системах любое пользовательское приложение заключено в окно, которое можно назвать главным, т.к. в нем располагаются все остальные виджеты. Объект окна верхнего уровня создается при обращении к классу Tk модуля tkinter.

Переменную связанную с объектом-окном принято называть root

Вторая строчка кода:

```
root = Tk()
```

1.3. Создание виджета

Допустим в окне будет располагаться всего одна кнопка. Кнопка создается при обращении к классу Button модуля tkinter. Объект кнопка связывается с какой-нибудь переменной.

У класса Button (как и всех остальных классов, за исключением Tk) есть обязательный параметр — объект, которому кнопка принадлежит (кнопка не может "быть ничейной"). Пока у нас есть единственное окно (root), оно и будет аргументом, передаваемым в класс при создании объекта-кнопки:

```
but = Button(root)
```

1.4. Установка свойств виджет

У кнопки много свойств: размер, цвет фона и надписи и др. Установим всего одно свойство — текст надписи (text):

```
but["text"] = "Печать"
```

1.5. Определение событий и их обработчиков

Предположим, что задача кнопки вывести какое-нибудь сообщение в поток вывода, используя функцию print. Делать она это будет при нажатии на нее левой кнопкой мыши.

Действия (алгоритм), которые происходят при том или ином событии, могут быть достаточно сложным. Поэтому часто их оформляют в виде функции, а затем вызывают, когда они понадобятся. Пусть у нас печать на экран будет оформлена в виде функции printer:

```
def printer(event):
     print("Нажата кнопка!")
```

Функцию желательно (почти обязательно) размещать в начале кода. Параметр event – это какое-либо событие.

Событие нажатия левой кнопкой мыши выглядит так: <Button-1>.

Требуется связать это событие с обработчиком (функцией printer). Для связи предназначен метод bind. Синтаксис связывания события с обработчиком выглядит так:

```
but.bind("<Button-1>",printer)
```

Обычной ошибкой является попытка при привязке указать в имени функции круглые скобки, например: `but.bind("<Button-1>",printer())`
Этого делать нельзя!

1.6. Размещение виджет

В любом приложении виджеты не разбросаны по окну как попало, а хорошо организованы, интерфейс продуман до мелочей и обычно подчинен определенным стандартам. При необходимости просто как-то отобразить виджет в окне, используют метод pack.

```
but.pack()
```

Если не вставить эту строчку кода, то кнопка в окне так и не появится, хотя она есть в программе.

1.7. Отображение главного окна

Главное окно тоже не появится, пока не будет вызван специальный метод mainloop:

```
root.mainloop()
```

Данная строка кода должна быть всегда в конце скрипта!

В итоге, код программы может выглядеть таким образом:

```
from tkinter import *

def printer(event):
     print("Нажата кнопка!")

root = Tk()
but = Button(root)
but["text"] = "Печать"
but.bind("<Button-1>",printer)

but.pack()
root.mainloop()
```

1.9. Практическая работа.

1. Создайте скрипт из пункта 1.7. Добавьте комментарий к каждой строке кода. Сохраните в отчете по практике.

2. Добавьте в скрипт из п.1.7 вторую кнопку. Измените скрипт так, чтобы он выводил в Python-Shell сообщения о нажатии соответствующей кнопки.

2. Виджеты (графические объекты) и их свойства.

Для построения графического интерфейса в библиотеке Tk отобраны следующие классы виджетов (в алфавитном порядке):

- Button (Кнопка) Простая кнопка для вызова некоторых действий (выполнения определенной команды).
- Canvas (Рисунок) Основа для вывода графических примитивов.
- Checkbutton (Флажок) Кнопка, которая умеет переключаться между двумя состояниями при нажатии на нее.
- Entry (Поле ввода) Горизонтальное поле, в которое можно ввести строку текста.
- Frame (Рамка) Виджет, который содержит в себе другие визуальные компоненты.
- Label (Надпись) Виджет может показывать текст или графическое изображение.
- Listbox (Список) Прямоугольная рамка со списком, из которого пользователь может выделить один или несколько элементов.
- Menu (Меню) Элемент, с помощью которого можно создавать всплывающие (popup) и ниспадающие (pulldown) меню.
- Menubutton (Кнопка-меню) Кнопка с ниспадающим меню.
- Message (Сообщение) Аналогично надписи, но позволяет заворачивать длинные строки и менять размер по требованию менеджера расположения.
- Radiobutton (Селекторная кнопка) Кнопка для представления одного из альтернативных значений. Такие кнопки, как правило, действует в группе. При нажатии на одну из них кнопка группы, выбранная ранее, "отскакивает".
- Scale (Шкала) Служит для задания числового значения путем перемещения движка в определенном диапазоне.
- Scrollbar (Полоса прокрутки) Полоса прокрутки служит для отображения величины прокрутки в других виджетах. Может быть как вертикальной, так и горизонтальной.
- Text (Форматированный текст) Этот прямоугольный виджет позволяет редактировать и форматировать текст с использованием различных стилей, внедрять в текст рисунки и даже окна.
- Toplevel (Окно верхнего уровня) Показывается как отдельное окно и содержит внутри другие виджеты.

Все эти классы не имеют отношений наследования друг с другом - они равноправны. Этот набор достаточен для построения интерфейса в большинстве случаев.

Рассмотрим часть графических объектов (виджет), содержащихся в библиотеке Tkinter:

- кнопка,
- метка,
- поля для ввода,
- радиокнопки (зависимые переключатели),
- флажок (независимый переключатель),
- список,
- рамка,
- шкала,
- полоса прокрутки,
- окно верхнего уровня.

Следует понимать, что графический интерфейс пользователя достаточно стандартен, и поэтому любые подобные библиотеки-модули (в том числе и Tkinter) содержат приблизительно одинаковые виджеты.

Каждый класс виджет имеет определенные свойства, значения которых можно задавать при их создании, а также программировать их изменение при действии пользователя и в результате выполнения программы.

2.1. Кнопка.

Объект-кнопка создается вызовом класса Button модуля tkinter. При этом обязательным аргументом является лишь родительский виджет (например, окно верхнего уровня). Другие свойства могут указываться при создании кнопки или задаваться (изменяться) позже.

Синтаксис:

```
переменная = Button (родительский_виджет, [свойство=значение, … ….])
```

У кнопки много свойств, в примере ниже указаны лишь некоторые из них.

```
from tkinter import *
root = Tk()

but = Button(root,
          text="Это кнопка",      #надпись на кнопке
          width=30, height=5,     #ширина и высота
          bg="white", fg="blue") #цвет фона и надписи

but.pack()
root.mainloop()
```

bg и fg – это сокращения от background (фон) и foreground (передний план). Ширина и высота измеряются в знакоместах (количество символов).

2.2. Метка.

Метки (или надписи) — это виджеты, содержащие строку (или несколько строк) текста и служащие в основном для информирования пользователя.

```
lab = Label(root, text="Это метка! \n
                        Из двух строк.", font="Arial 18")
```

2.3. Однострочное текстовое поле.

Такое поле создается вызовом класса Entry модуля tkinter. В него пользователь может ввести только одну строку текста.

```
ent = Entry(root,width=20,bd=3)
```

bd – это сокращение от borderwidth (ширина границы).

2.4. Многострочное текстовое поле.

Text предназначен для предоставления пользователю возможности ввода не одной строки текста, а существенно больше.

```
tex = Text(root,width=40,
           font="Verdana 12",
           wrap=WORD)
```

Последнее свойство (wrap) в зависимости от своего значения позволяет переносить текст, вводимый пользователем либо по символам, либо по словам, либо вообще не переносить, пока пользователь не нажмет Enter.

2.5. Радиокнопки (зависимые переключатели).

Объект радиокнопка никогда не используется по одному. Их используют группами, при этом в одной группе может быть «включена» лишь одна кнопка.

```
var=IntVar()
var.set(1)
rad0 = Radiobutton(root, text="Первая",
            variable=var, value=0)
rad1 = Radiobutton(root, text="Вторая",
            variable=var, value=1)
rad2 = Radiobutton(root, text="Третья",
            variable=var, value=2)
```

Одна группа определяет значение одной переменной, т. е. если в примере будет выбрана радиокнопка rad2, то значение переменной var будет 2. Изначально также требуется установить значение переменной (выражение var.set(1) задает значение переменной var равное 1).

2.6. Флажок (независимый переключатель).

Объект checkbutton предназначен для выбора не взаимоисключающих пунктов в окне (в группе можно активировать один, два или более флажков или не один). В отличие от радиокнопок, значение каждого флажка привязывается к своей переменной, значение которой определяется опциями onvalue (включено) и offvalue (выключено) в описании флажка.

```
c1 = IntVar()
c2 = IntVar()
che1 = Checkbutton(root,text="Первый флажок",
           variable=c1, onvalue=1, offvalue=0)
che2 = Checkbutton(root,text="Второй флажок",
           variable=c2, onvalue=2, offvalue=0)
```

2.7. Список.

Вызов класса Listbox создает объект, в котором пользователь может выбрать один или несколько пунктов в зависимости от значения опции selectmode. В примере ниже значение SINGLE позволяет выбирать лишь один пункт из списка.

```
sp = ['Linux', 'Python', 'Tk', 'Tkinter']
lis = Listbox(root, selectmode=SINGLE, height=4)
for i in sp:
     lis.insert(END, i)
```

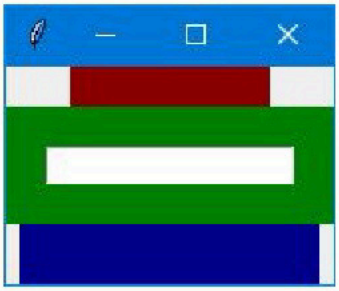
Изначально список (Listbox) пуст. С помощью цикла for в него добавляются пункты из списка (тип данных) sp. Добавление происходит с помощью специального метода класса Listbox — insert. Данный метод принимает два параметра: куда добавить и что добавить.

2.8. Рамка.

Рамки (фреймы) хороший инструмент для организации остальных виджет в группы внутри окна, а также оформления.

```
from tkinter import *
root = Tk()

fra1 = Frame(root, width=500, height=100, bg="darkred")
fra2 = Frame(root, width=300, height=200, bg="green", bd=20)
fra3 = Frame(root, width=500, height=150, bg="darkblue")

fra1.pack()
fra2.pack()
fra3.pack()

root.mainloop()
```

Данный скрипт создает три фрейма разного размера. Свойство bd (сокращение от borderwidth) определяет расстояния от края рамки до заключенных в нее виджетов (если они есть).



На фреймах также можно размещать виджеты как на основном окне (root). Здесь текстовое поле находится на рамке fra2.

```
ent1 = Entry(fra2,width=20)
ent1.pack()
```

2.9. Шкала.

Назначение шкалы — это предоставление пользователю выбора какого-то значения из определенного диапазона. Внешне шкала представляет собой горизонтальную или вертикальную полосу с разметкой, по которой пользователь может передвигать движок, осуществляя тем самым выбор значения.

```
sca1 = Scale(fra3, orient=HORIZONTAL, length=300,
          from_=0, to=100, tickinterval=10, resolution=5)
sca2 = Scale(root, orient=VERTICAL, length=400,
          from_=1, to=2, tickinterval=0.1, resolution=0.1)
```

Свойства:

- orient определяет направление шкалы;
- length – длина шкалы в пикселях;

- from_ и to – с какого значения шкала начинается и каким заканчивается (т.е. диапазон значений);
- tickinterval – интервал, через который отображаются метки для шкалы;
- resolution - минимальная длина отрезка, на которую пользователь может передвинуть движок.

2.10. Полоса прокрутки.

Данный виджет позволяет прокручивать содержимое другого виджета (например, текстового поля или списка). Прокрутка может быть как по горизонтали, так и по вертикали.

```
from tkinter import *
root = Tk()

tx = Text(root,width=40,height=3,font='14')
scr = Scrollbar(root,command=tx.yview)
tx.configure(yscrollcommand=scr.set)

tx.grid(row=0,column=0)
scr.grid(row=0,column=1)
root.mainloop()
```

В примере сначала создается текстовое поле (tx), затем полоса прокрутки (scr), которая привязывается с помощью опции command к полю tx по вертикальной оси (yview). Далее поле tx изменяется (конфигурируется) с помощью метода configure: устанавливается значение опции yscrollcommand.

Здесь используется метод grid, представляющий собой другой способ расположения виджет на окне.

2.11. Окно верхнего уровня.

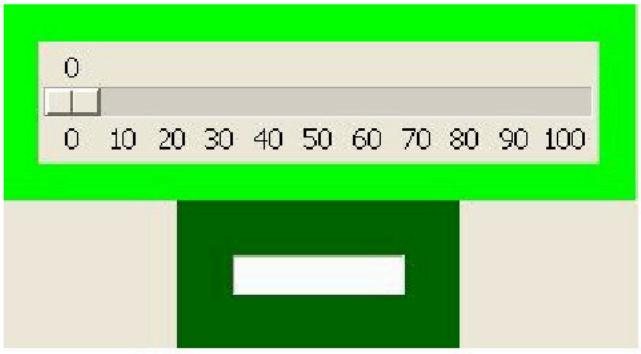
С помощью класса Toplevel создаются дочерние окна, на которых также могут располагаться виджеты. Следует отметить, что при закрытии главного окна (или родительского), окно Toplevel также закрывается. С другой стороны, закрытие дочернего окна не приводит к закрытию главного.

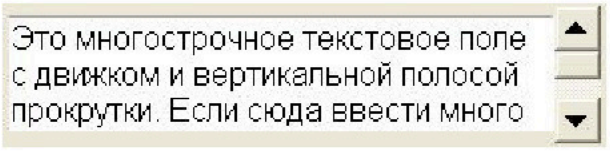
```
win = Toplevel(root,relief=SUNKEN,bd=10,bg="lightblue")
win.title("Дочернее окно")
win.minsize(width=400,height=200)
```

Метод title определяет заголовок окна. Метод minsize конфигурирует минимальный размер окна (есть метод maxsize, определяющий максимальный размер окна). Если значение аргументов minsize будет таким же как у maxsize, то пользователь не сможет менять размеры окна.

2.12. Практическая работа.

1. Создайте четыре скрипта с использованием модуля Tkinter, генерирующие шаблоны представленные ниже.





2. Создайте приложение, состоящее из главного и двух дочерних окон. На каждом из трех окон должны располагаться один или два любых графических объекта.

3. Управление размещением

В Tkinter (библиотеке компонентов графического интерфейса, входящей в Python по "умолчанию") есть три стандартных «менеджера геометрии» (англ. «Geometry Manager») - управляющих размещениями: Grid, Pack и Place.

Занимаются они тем, что располагают на главном окне остальные виджеты, причем каждый из трех делает это по-своему.

Непосредственно внутри одного виджета нельзя использовать более одного менеджера расположения: менеджеры могут наложить противоречащие ограничения на вложенные виджеты и внутренние виджеты просто не смогут быть расположены.

Рассмотрим создание прототипа диалогового окна тремя разными способами.

```
# -*- coding: UTF-8 -*-
from tkinter import *

# Виджеты
root = Tk() # окно
root.title("Вставка строки")

l_in=Label(root,text="Вставить") # метки
l_num=Label(root,text="Кол-во")
l_pos=Label(root,text="Положение")

e_num=Entry(root, width=5, bg="White") # текстовое поле

pos = IntVar() # радиокнопки
pos.set(0)
r_pos1=Radiobutton(root,text="До", variable=pos, value=0)
r_pos2=Radiobutton(root,text="После", variable=pos, value=1)

b_ok=Button(root,text="OK", width=10) # Кнопки
b_canc=Button(root,text="Отмена", width=10)
b_help=Button(root,text="Справка", width=10)

# Размещение виджетов
#...

root.mainloop()
```

3.1. Grid – таблица.

Grid делит пространство родительского виджета на ячейки, количество которых определяется дочерними виджетами. Ячейка идентифицируется номером строки (row) и номером столбца (column). Ячейки можно объединять как по горизонтали (columnspan), так и по вертикали (rowspan).

В соответствие с вышеприведенным примером разработаем схему, которая поможет определить количество необходимых ячеек, их адреса и объединения:

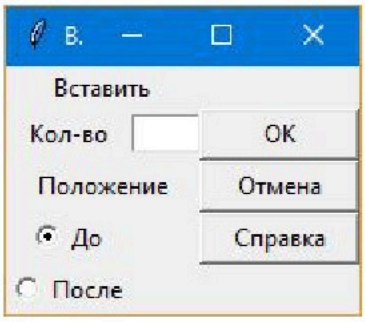
0, 0		
1, 0	1, 1	1, 2
2, 0		2, 2
3, 0		3, 2
4, 0		

Ячейки, в которых будут располагаться виджеты, пронумерованы. Первое число – номер строки, второе – столбца (нумерация начинается с нуля).

Вторая часть кода:

```
...
# Размещение виджетов
l_in.grid(row=0,column=0,columnspan=2)
l_num.grid(row=1,column=0)
e_num.grid(row=1,column=1)
l_pos.grid(row=2,column=0,columnspan=2)
r_pos1.grid(row=3,column=0)
r_pos2.grid(row=4,column=0)
b_ok.grid(row=1,column=2)
b_canc.grid(row=2,column=2)
b_help.grid(row=3,column=2)
...
```

При выполнении получаем следующее:



Положение радио-кнопок относительно друг друга явно не в порядке. Это результат того, что размер ячейки не всегда совпадает с размером виджета; а по-умолчанию положение виджета в ячейке определяется по центру.

Для изменения положения виджета в ячейке используется опция sticky ("липучка"), значение которой может быть любой комбинацией констант S, N, E и W, или NW, NE, SW и SE.

Например, W (запад) означает, что виджет должен быть выровнен по левой границе ячейки. W+E – виджет будет вытянут горизонтально и заполнит целую ячейку по данной оси. W+E+N+S означает, что виджет будет расширен в обоих направлениях.

Слишком близкое расположение кнопок друг к другу исправляется с помощью опций padx и pady, которые позволяют заполнить пустотой место вокруг виджета.

Измененный вариант второй части скрипта:

```
...
# Размещение виджетов
l_in.grid(row=0,column=0,columnspan=2,sticky=W,pady=5)
l_num.grid(row=1,column=0)
e_num.grid(row=1,column=1,sticky=W)
l_pos.grid(row=2,column=0,columnspan=2,sticky=W)
r_pos1.grid(row=3,column=0,sticky=W,padx=5)
r_pos2.grid(row=4,column=0,sticky=W,padx=5)
b_ok.grid(row=1,column=2,padx=10,pady=5)
```

```
b_canc.grid(row=2,column=2,padx=10,pady=5)
b_help.grid(row=3,column=2,padx=10,pady=5)
...
```

3.2. Pack – стопка.

Менеджер Pack располагает виджеты друг за другом по той или иной стороне. Опция side может принимать следующие значения: TOP (значение по-умолчанию), LEFT, BOTTOM и RIGHT.

Для дочерних виджетов, располагающихся на одном родительском окне, можно указывать разные стороны, однако при этом можно и не добиться желаемого результата.

Менеджер Pack просто заполняет внутреннее пространство на основании предпочтения того или иного края, необходимости заполнить все измерение. В некоторых случаях ему приходится менять размеры подчиненных виджетов. Этот менеджер стоит использовать только для достаточно простых схем расположения виджетов.

Разберем более простой пример: расположим четыре рамки сначала горизонтально, вертикально и расположение «2x2».

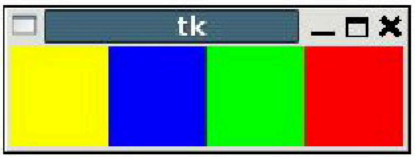
Общая часть кода:

```
from Tkinter import *
root = Tk()

f_1=Frame(root,width=50,height=50,bg="Yellow")
f_2=Frame(root,width=50,height=50,bg="Blue")
f_3=Frame(root,width=50,height=50,bg="Green")
f_4=Frame(root,width=50,height=50,bg="Red")

# Размещение рамок
# ...

root.mainloop()
```

Размещение рамок:

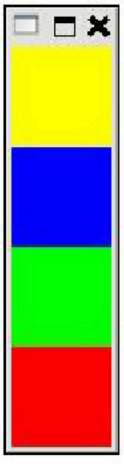
горизонтально

```
f_1.pack(side=LEFT)
f_2.pack(side=LEFT)
f_3.pack(side=LEFT)
f_4.pack(side=LEFT)
```



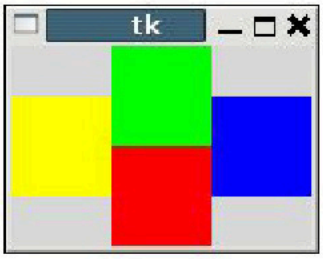
вертикально

```
f_1.pack()
f_2.pack()
f_3.pack()
f_4.pack()
```



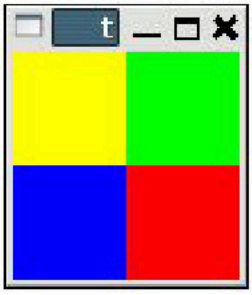
Более сложное расположение

```
f_1.pack(side=LEFT)
f_2.pack(side=RIGHT)
f_3.pack(side=TOP)
f_4.pack(side=BOTTOM)
```



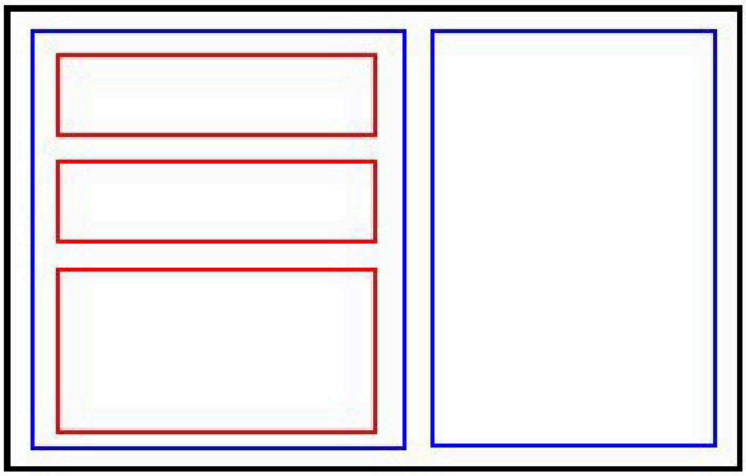
Для размещения фреймов в виде «2x2» можно использовать дополнительные рамки (фреймы):

Размещение «2x2»




```
from Tkinter import *
root = Tk()

f_01=Frame(root)
f_02=Frame(root)
f_1=Frame(f_01,width=50,height=50,bg="Yellow")
f_2=Frame(f_01,width=50,height=50,bg="Blue")
f_3=Frame(f_02,width=50,height=50,bg="Green")
f_4=Frame(f_02,width=50,height=50,bg="Red")

f_01.pack(side=LEFT)
f_02.pack(side=LEFT)
f_1.pack()
f_2.pack()
f_3.pack()
f_4.pack()

root.mainloop()
```

Более сложный пример. Схема размещения фреймов:



Прямоугольники – это рамки; внешние рамки являются родительскими по отношению к внутренним.

Пример скрипта:

```
# -*- coding: UTF-8 -*-
from tkinter import *

# Виджеты
root = Tk() # окно
root.title("Вставка строки")
```

```
f_1=Frame(root) # рамки
f_2=Frame(root)
f_11=Frame(f_1)
f_12=Frame(f_1)
f_13=Frame(f_1)

l_in=Label(f_11,text="Вставить --------------------") # метки
l_num=Label(f_12,text="Кол-во")
l_pos=Label(f_13,text="Положение -----------------")

e_num=Entry(f_12, width=5, bg="White") # текстовое поле

pos = IntVar() # радиокнопки
pos.set(0)
r_pos1=Radiobutton(f_13,text="До ", variable=pos, value=0)
r_pos2=Radiobutton(f_13,text="После ", variable=pos, value=1)

b_ok=Button(f_2,text="OK", width=10) # Кнопки
b_canc=Button(f_2,text="Отмена", width=10)
b_help=Button(f_2,text="Справка", width=10)

# Размещение виджетов
f_1.pack(side=LEFT)
f_2.pack(side=LEFT,fill=Y)
f_11.pack()
f_12.pack()
f_13.pack()

l_in.pack()
l_num.pack(side=LEFT)
e_num.pack(side=LEFT)
l_pos.pack()
r_pos1.pack()
r_pos2.pack()

b_ok.pack()
b_canc.pack()
b_help.pack(side=BOTTOM)

root.mainloop()
```

Обратите внимание на параметр fill в методе pack (), примененном к рамке f_2. Он позволяет заполнять виджету свободное пространство по оси X, Y или обоим направлениям (BOTH). В данном случае, это позволило f_2 стать по высоте равной f_1; иначе ее высота равнялась бы суммарной высоте дочерних виджетов (трех кнопок), что не позволило бы развести их по разным сторонам.

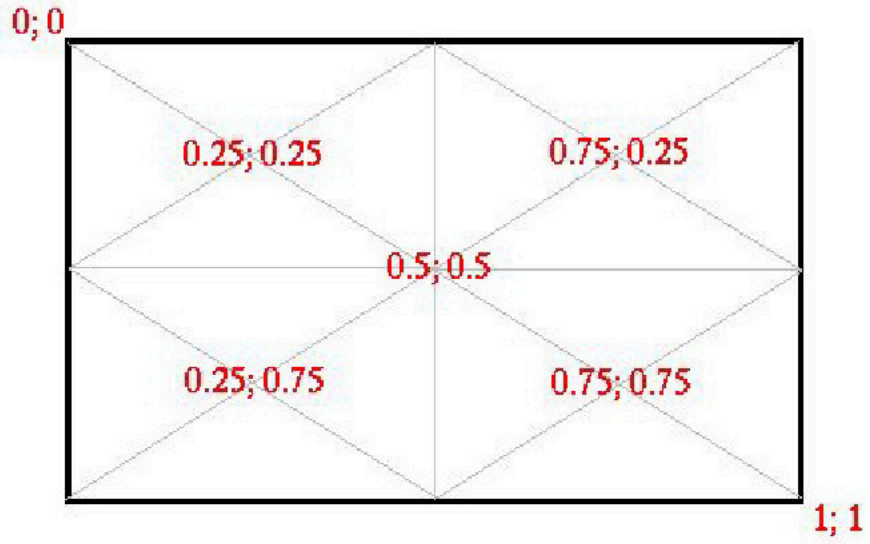
3.3. Place – координаты.

С помощью Place можно явно указывать виджету на его место с помощью координат, как в абсолютных значениях (в пикселях), так и относительно родительского окна (в долях). Также относительно и абсолютно можно задавать размер самого виджета.

Наиболее употребимые параметры данного метода:

- anchor (якорь) – определяет часть виджета, для которой задаются координаты. Принимает значения - N, NE, E, SE, SW, W, NW, или CENTER. По умолчанию NW (верхний левый угол).
- relwidth, relheight (относительные ширина и высота) – определяют размер виджета по отношению к его родителю.
- relx, rely – определяют относительную позицию обычно в родительском виджете. Координата (0;0) – у левого верхнего угла, (1;1) - у правого нижнего.
- width, height – абсолютный размер в пикселах. Значения по умолчанию (когда данные опции опущены) приравниваются к "естественному" размеру виджета.
- x, y - абсолютная позиция в пикселах (по умолчанию равны 0).

Создадим для наглядности общую схему, с указанием ключевых относительных координат...



, а затем напишем код:

```
# -*- coding: UTF-8 -*-
from tkinter import *

# Виджеты
root = Tk() # окно
root.title("Вставка строки")

f_1=Frame(root,width=300,height=150) # рамка

l_in=Label(f_1,text="Вставить --------------------") # метки
l_num=Label(f_1,text="Кол-во")
l_pos=Label(f_1,text="Положение -----------------")

e_num=Entry(f_1, width=5, bg="White") # текстовое поле

pos = IntVar() # радиокнопки
pos.set(0)
r_pos1=Radiobutton(f_1,text="До", variable=pos, value=0)
r_pos2=Radiobutton(f_1,text="После", variable=pos, value=1)

b_ok=Button(f_1,text="OK", width=8) # Кнопки
b_canc=Button(f_1,text="Отмена", width=8)
b_help=Button(f_1,text="Справка", width=8)
```

```
# Расположение виджетов
f_1.pack()
l_in.place(relx=0.03,rely=0.05)
l_num.place(relx=0.07,rely=0.25)
e_num.place(relx=0.3,rely=0.25)
l_pos.place(relx=0.03,rely=0.5)
r_pos1.place(relx=0.07,rely=0.65)
r_pos2.place(relx=0.07,rely=0.8)
b_ok.place(relx=0.7,rely=0.2)
b_canc.place(relx=0.7,rely=0.4)
b_help.place(relx=0.7,rely=0.7)

root.mainloop()
```

Обратите внимание, что виджеты размещаются не на главном окне, а рамке f_1. Рамка введена для того, чтобы диалоговое окно приняло нужную форму. К ней применен метод pack(). Хотя нельзя использовать два менеджера внутри одного родительского окна, к данному случаю это правило не относится, т.к. у рамки родителем является главное окно, а у остальных виджетов – рамка.

Размер виджета не указан, а значит равняется их «естественному» размеру. Указание относительных размеров чревато тем, что при изменении размера родительского окна будет изменяться и сам виджет. Указание же абсолютных координат при изменяемом родительском окне также может приводить к нежелательным результатам.

Чтобы убедиться в «непредсказуемости» менеджера place, достаточно немного подправить вышеприведенный код. Позволим рамке расширяться и занимать все свободное пространство:

```
f_1.pack(fill=BOTH,expand=1)
```

Для кнопок укажем абсолютные координаты и относительные размеры:

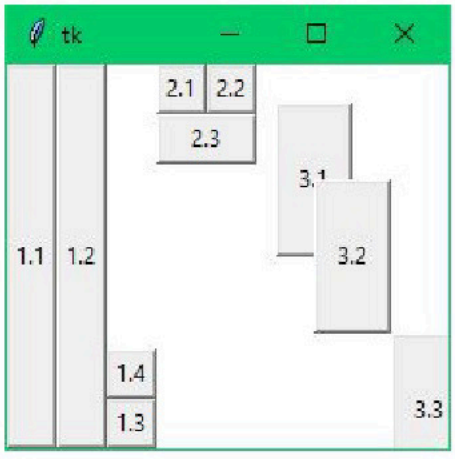
```
b_ok.place(x=200,y=25, relwidth=0.2, relheight=0.2)
b_canc.place(x=200,y=60, relwidth=0.2, relheight=0.2)
b_help.place(x=200,y=95, relwidth=0.2, relheight=0.2)
```

Замените соответствующие строки кода, запустите скрипт, разверните окно на полный экран и убедитесь в изменении размеров кнопок в зависимости от размеров фрейма, в котором они находятся.

3.4. Практическая работа

1. Поместить в отчет скрипты из раздела и соответствующие им скриншоты.

2. Создать скрипт, реализующий интерфейс, представленный на рисунке:



4. Метод bind

4.1. Привязка события.

Приложения с графическим интерфейсом пользователя (GUI) должны не просто красиво отображаться на экране, но и выполнять какие-либо действия, реализуя тем самым потребности пользователя. В отличие от консольных приложений, которые обычно выполняются при минимальных внешних воздействиях, графическое приложение обычно ждет каких-либо внешних воздействий (щелчков кнопкой мыши, нажатий клавиш на клавиатуре, изменения виджетов) и затем выполняет заложенное программистом действие.

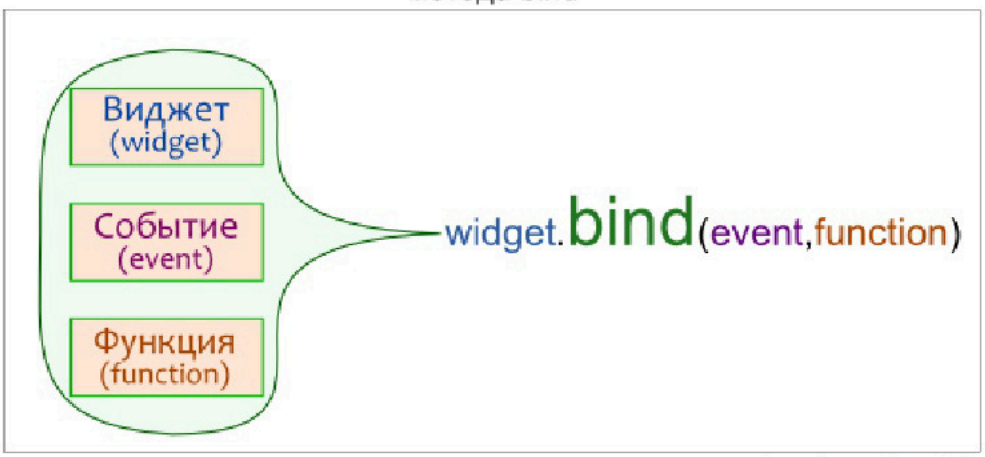
Внешнее воздействие на графический компонент называется событием.

В Tk события описываются в виде текстовой строки - шаблона события, состоящего из трех элементов (модификаторы, тип события и детализация события).

Тип события	Содержание события
`Activate`	Активизация окна
`ButtonPress`	Нажатие кнопки мыши
`ButtonRelease`	Отжатие кнопки мыши
`Deactivate`	Деактивация окна
`Destroy`	Закрытие окна
`Enter`	Вхождение курсора в пределы виджета
`FocusIn`	Получение фокуса окном
`FocusOut`	Потеря фокуса окном
`KeyPress`	Нажатие клавиши на клавиатуре
`KeyRelease`	Отжатие клавиши на клавиатуре
`Leave`	Выход курсора за пределы виджета
`Motion`	Движение мыши в пределах виджета
`MouseWheel`	Прокрутка колесика мыши
`Reparent`	Изменение родителя окна
`Visibility`	Изменение видимости окна

Одним из способов связывания виджета, события и функции (того, что должно происходить после события) является использование метода bind. Синтаксис связывания представлен на рисунке.

Добавление функциональности графическому элементу с помощью метода bind



Рассмотрим два вида событий: щелчок левой кнопкой мыши и нажатие клавиши Enter.

Пример 1.

```
 1 def output(event):
 2     s = ent.get()
 3     if s == "1":
 4         tex.delete(1.0,END)
 5         tex.insert(END,"Выбран вариант 1")
 6     elif s == "2":
 7         tex.delete(1.0,END)
 8         tex.insert(END,"Выбран вариант 2")
 9     else:
10         tex.delete(1.0,END)
11         tex.insert(END,"Введите 1 или 2 в поле слева")
12
13 from tkinter import *
14 root = Tk()
15
16 ent = Entry(root,width=1)
18 but = Button(root,text="Вывести")
19 tex = Text(root,width=20,height=3,font="12",wrap=WORD)
20
21 ent.grid(row=0,column=0,padx=20)
22 but.grid(row=0,column=1)
23 tex.grid(row=0,column=2,padx=20,pady=10)
24
25 but.bind("<Button-1>",output)
26
27 root.mainloop()
```

В строках 16-19 создаются три виджета: однострочное текстовое поле, кнопка и многострочное текстовое поле. В первое поле пользователь должен ввести цифру, затем нажать кнопку и получить ответ во втором поле.

В строках 21-23 используется менеджер grid для размещения виджетов. Свойства padx и pady определяют количество пикселей от виджета до края рамки (или ячейки) по осям x и y соответственно.

В стоке 25 выполняется связывание кнопки с событием нажатия левой кнопки мыши и функцией output. Все эти три компонента (виджет, событие и функция) связываются с помощью метода bind. В данном случае, при нажатии левой кнопкой мыши по кнопке but будет вызвана функция output.

В итоге, когда пользователь кликнет левой кнопкой мыши по кнопке, то выполнится функция output

Данная функция (строки 1-11) выводит информацию во второе текстовое поле. Какую именно информацию, зависит от того, что пользователь ввел в первое текстовое поле. В качестве аргумента функции передается событие (в данном случае).

Внутри веток if-elif-else используются методы delete и insert. Первый из них удаляет символы из текстового поля, второй — вставляет. 1.0 — обозначает первую строку, первый символ (нумерация символов начинается с нуля).

Пример 2.

```
 1 li = ["red","green"]
 2 def color(event):
 3      fra.configure(bg=li[0])
 4      li[0],li[1] = li[1],li[0]
 5
 6 def outgo(event):
 7      root.destroy()
 8
 9 from tkinter import *
10 root = Tk()
11
12 fra = Frame(root,width=100,height=100)
13 but = Button(root,text="Выход")
14
15 fra.pack()
16 but.pack()
17
18 root.bind("<Return>",color)
19 but.bind("<Button-1>",outgo)
20
21 root.mainloop()
```

Здесь создаются два виджета (строки 12, 13): фрейм и кнопка.

Приложение реагирует на два события: нажатие клавиши Enter в пределах главного окна (строка 18) и нажатие левой кнопкой мыши по кнопке but (строка 19). В первом случае вызывается функция color, во втором — outgo.

Функция color изменяет цвет фона (bg) фрейма (fra) с помощью метода configure, который предназначен для изменения значения свойств виджетов в процессе выполнения скрипта. В качестве значения опции bg подставляется первый элемент списка. Затем в списке два элемента меняются местами, чтобы при следующем нажатии Enter цвет фрейма снова изменился.

В функции outgo вызывается метод destroy по отношению к главному окну. Данный метод предназначен для «разрушения» виджета (окно закроется).

4.2. Практическая работа

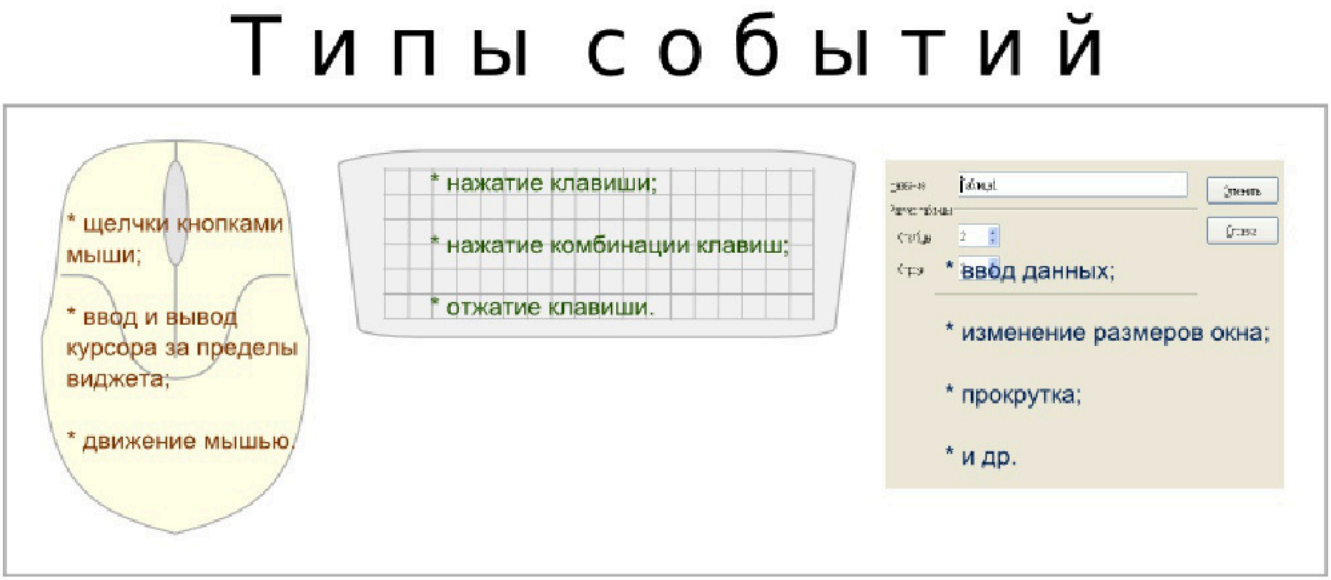
1. Создайте приложение, в котором меняется размер фрейма в зависимости от того, какая из трех объектов-кнопок была нажата.
2. Напишите скрипт, генерирующий окно с меткой и текстовым полем. После ввода пользователем текста в поле и нажатия Enter, введенный текст должен отображаться в метке.

5. Программирование событий

5.1. Типы событий

Можно выделить три основных типа событий:

- производимые мышью,
- нажатиями клавиш на клавиатуре,
- возникающие в результате изменения других графических объектов.



5.2. Способ записи

При вызове метода bind событие передается в качестве первого аргумента.

Название события заключается в кавычки, а также в знаки < и >. Событие описывается с помощью зарезервированных последовательностей ключевых слов.

5.3. События, производимые мышью

- <Button-1> - щелчок левой кнопкой мыши
- <Button-2> - щелчок средней кнопкой мыши
- <Button-3> - щелчок правой кнопкой мыши
- <Double-Button-1> - двойной клик левой кнопкой мыши
- <Motion> - движение мыши
- и т. д.

Пример:

```
from tkinter import *

def b1(event):
     root.title("Левая кнопка мыши")
```

```
def b3(event):
     root.title("Правая кнопка мыши")

def move(event):
     root.title("Движение мышью")

root = Tk()
root.minsize(width = 500, height=400)

root.bind('<Button-1>',b1)
root.bind('<Button-3>',b3)
root.bind('<Motion>',move)

root.mainloop()
```

В примере меняется надпись в заголовке главного окна в зависимости от того двигается мышь, щелкают левой или правой кнопкой мыши.

5.4. События, производимые с помощью клавиатуры

- Буквенные клавиши можно записывать без угловых скобок (например, 'L').
- Для неалфавитных клавиш существуют специальные зарезервированные слова
  - <Return> - нажатие клавиши Enter;
  - <space>- пробел;
  - и т. д.
- Сочетания клавиш пишутся через тире. Например:
  - <Control-Shift> - одновременное нажатие клавиш Ctrl и Shift.

Пример:

```
from tkinter import *

def exit_(event):
     root.destroy()

def caption(event):
     t = ent.get()
     lbl.configure(text = t)

root = Tk()

ent = Entry(root, width = 40)
lbl = Label(root, width = 80)

ent.pack()
lbl.pack()

ent.bind('<Return>',caption)
root.bind('<Control-z>',exit_)

root.mainloop()
```

При нажатии клавиши Enter в пределах текстовой строки (ent) вызывается функция caption, которая помещает символы из текстовой строки (ent) в метку (lbl). Нажатие комбинации клавиш Ctrl + z приводит к закрытию главного окна.

5.5. Практическая работа

1. Напишите программу, реализующую следующий алгоритм:
   - на главном окне находится несколько флажков и текстовое поле.
   - при щелчке левой кнопкой мыши в пределах текстового поля в нем должны отображаться значения включенных флажки (появляться сообщение о том, какие флажки включены),
   - при щелчке правой кнопкой мыши — значения выключенных флажков.

2. Напишите скрипт, генерирующий в окне два текстовых поля и рамку. Размер рамки можно менять с помощью вводимых значений в текстовые поля (определяют длину и ширину) и нажатии клавиши пробел на клавиатуре.

6. Переменные

6.1. Переменные состояния виджетов

Библиотека Tkinter содержит специальные классы, объекты которых выполняют роль переменных для хранения значений о состоянии различных виджет. Изменение значения такой переменной ведет к изменению и свойства виджета, и, наоборот: изменение свойства виджета изменяет значение ассоциированной переменной.

Существует несколько классов Tkinter, предназначенных для обработки данных разных типов.

1. StringVar() - для строк;
2. IntVar() - целых чисел;
3. DoubleVar() - дробных чисел;
4. BooleanVar() - для обработки булевых значений (true и false).

Пример 1.

Ранее мы уже использовали переменную-объект типа IntVar() при создании группы радиокнопок:

```
var=IntVar()
var.set(1)
rad0 = Radiobutton(root,text="Первая",variable=var,value=0)
rad1 = Radiobutton(root,text="Вторая",variable=var,value=1)
rad2 = Radiobutton(root,text="Третья",variable=var,value=2)
```

Здесь создается объект класса IntVar и связывается с переменной var. С помощью метода set устанавливается начальное значение, равное 1. Три радиокнопки относятся к одной группе: об этом свидетельствует одинаковое значение опции (свойства) variable.

Variable предназначена для связывания переменной Tkinter с радиокнопкой. Опция value определяет значение, которое будет передано переменной, если данная кнопка будет в состоянии "включено". Если в процессе выполнения скрипта значение переменной var будет изменено, то это отразится на группе кнопок. Например, это делается во второй строчке кода: включена кнопка rad1.

Если метод set позволяет устанавливать значения переменных, то метод get, наоборот, позволяет получать (узнавать) значения для последующего их использования.

```
def display(event):
     v = var.get()
     if v == 0:
          print ("Включена первая кнопка")
     elif v == 1:
          print ("Включена вторая кнопка")
     elif v == 2:
          print ("Включена третья кнопка")

but = Button(root,text="Получить значение")
but.bind('<Button-1>',display)
```

При вызове функции display в переменную v “записывается“ значение, связанное в текущий момент с переменной var. Чтобы получить значение переменной var, используется метод get (вторая строчка кода).

Пример 2.

Поскольку состояния флажков независимы друг друга, то для каждого должна быть введена собственная ассоциированная переменная-объект.

```
from tkinter import *
root = Tk()

var0=StringVar() # значение каждого флажка ...
var1=StringVar() # ... хранится в собственной переменной
var2=StringVar()

# если флажок установлен, то в ассоциированную переменную ...
# ...(var0,var1 или var2) заносится значение onvalue, ...
# ...если флажок снят, то - offvalue.
ch0 = Checkbutton(root,text="Окружность",variable=var0,
          onvalue="circle",offvalue="-")
ch1 = Checkbutton(root,text="Квадрат",variable=var1,
          onvalue="square",offvalue="-")
ch2 = Checkbutton(root,text="Треугольник",variable=var2,
          onvalue="triangle",offvalue="-")

lis = Listbox(root,height=3)
def result(event):
     v0 = var0.get()
     v1 = var1.get()
     v2 = var2.get()
     l = [v0,v1,v2] # значения переменных заносятся в список
     lis.delete(0,2) # предыдущее содержимое удаляется из Listbox
     for v in l: # содержимое списка l последовательно ...
          lis.insert(END,v) # ...вставляется в Listbox

but = Button(root,text="Получить значения")
but.bind('<Button-1>',result)

ch0.deselect()# "по умолчанию" флажки сняты
ch1.deselect()
ch2.deselect()

ch0.pack()
ch1.pack()
ch2.pack()
but.pack()
lis.pack()

root.mainloop()
```

Пример 3.

Помимо свойства (опции) variable, связывающей виджет с переменной-объектом Tkinter (IntVar, StringVar и др.), у многих виджет существует опция textvariable, которая определяет текст-содержимое или текст-надпись виджета. Несмотря на то, что

«текстовое свойство» может быть установлено для виджета и изменено в процессе выполнения кода без использования ассоциированных переменных, иногда такой способ изменения оказывается более удобным.

```
from tkinter import *
root = Tk()
v = StringVar()
ent1 = Entry (root, textvariable = v,bg="black",fg="white")
ent2 = Entry(root, textvariable = v)
ent1.pack()
ent2.pack()
root.mainloop()
```

Здесь содержимое одного текстового поля немедленно, отображается в другом, т.к. оба поля привязаны к одной и той же переменной v.

6.2. Практическая работа.

1. Напишите скрипт, как в примере с флажками; в отличии от примера значения ассоциированных переменных должны отображаться в метке (Label) через запятую.

2. Напишите программу, в которой пользователь может определить цвет рамки (Frame) с помощью шкалы (Scale).

7. Объект Меню (Menu) в GUI.

Меню — это объект, который присутствует во многих пользовательских приложениях. Находится оно под строкой заголовка и представляет собой выпадающие списки под словами; каждый такой список может содержать другой вложенный в него список. Каждый пункт списка представляет собой команду, запускающую какое-либо действие или открывающую диалоговое окно.

7.1. Создание меню.

```
from tkinter import *
root = Tk()

m = Menu(root) # создается объект Меню на главном окне
root.config(menu=m) # окно конфигурируется с указанием
                    # меню для него
# первый пункт меню
fm = Menu(m)    # создается пункт меню с размещением
                # на основном меню (m)
m.add_cascade(label="File",menu=fm)# пункт располагается
                                  # на основном меню (m)
fm.add_command(label="Open...")    # формируется список команд
                                  # пункта меню
fm.add_command(label="New")
fm.add_command(label="Save...")
fm.add_command(label="Exit")

# второй пункт меню
hm = Menu(m) #второй пункт меню
m.add_cascade(label="Help",menu=hm)
hm.add_command(label="Help")
hm.add_command(label="About")

root.mainloop()
```

Метод add_cascade добавляет новый пункт в меню, который указывается как значение опции menu.

Метод add_command добавляет новую команду в пункт меню. Одна из опций данного метода (в примере выше ее пока нет) — command – связывает данную команду с функцией- обработчиком.

Можно создавать вложенное меню. Для этого создается еще одно меню и с помощью add_cascade привязывается к родительскому пункту.

```
nfm = Menu(fm)
fm.add_cascade(label="Import",menu=nfm)
nfm.add_command(label="Image")
nfm.add_command(label="Text")
```

7.2. Привязка функций к меню.

Каждая команда меню обычно должна быть связана со своей функцией, выполняющей те или иные действия. Связь происходит с помощью опции command метода add_command. Функция обработчик должна быть определена заранее.

Для примера выше далее приводятся исправленные строки добавления команд “About”, “New” и “Exit”, а также функции, вызываемые, когда пользователь щелкает левой кнопкой мыши по соответствующим пунктам подменю.

```
def new_win():
     win = Toplevel(root)

def close_win():
     root.destroy()

def about():
     win = Toplevel(root)
     lab = Label(win,text="Это просто программа-тест \n меню в
Tkinter")
     lab.pack()

….
fm.add_command(label="New",command=new_win)
….
fm.add_command(label="Exit",command=close_win)

….
hm.add_command(label="About",command=about)
```

7.3. Практическая работа.

Напишите приложение с меню, содержащим два пункта: Color и Size. Пункт Color должен содержать три команды (Red, Green и Blue), меняющие цвет рамки на главном окне. Пункт Size должен содержать две команды (500x500 и 700x400), изменяющие размер рамки.

8. Диалоговые окна.

Диалоговые окна, как элементы графического интерфейса, предназначены для вывода сообщений пользователю, получения от него какой-либо информации, а также управления.

8.1. Диалоговые окна открытия и сохранения файла.

Для примера рассмотрим, как запрограммировать с помощью Tkinter вызов диалоговых окон открытия и сохранения файлов и работу с ними. При этом требуется дополнительно импортировать "подмодуль" Tkinter - tkinter.filedialog, в котором описаны классы для окон данного типа.

```
from tkinter import *
from tkinter.filedialog import *

root = Tk()
op = askopenfilename()
sa = asksaveasfilename()

root.mainloop()
```

Здесь создаются два объекта (op и sa): один вызывает диалоговое окно "Открыть", а другой "Сохранить как...".

При выполнении скрипта, они друг за другом выводятся на экран после появления главного окна. Если не создать root, то оно все-равно появится на экране, однако при попытке его закрытия в конце возникнет ошибка.

Разместим многострочное текстовое поле на главном окне и в дальнейшем попробуем туда загружать содержимое текстовых файлов.

```
from tkinter import *
from tkinter.filedialog import *

root = Tk()
txt = Text(root,width=40,height=15,font="12")
txt.pack()

op = askopenfilename()
txt.insert(END,op)

root.mainloop()
```

При запуске скрипта появляется окно с текстовым полем и сразу диалоговое окно "Открыть". При выборе файла в окне появляется его имя, а не содержимое.

Функция askopenfilename() возвращает указатель на файл (полное имя файла), который и передается функции txt.insert() для добавления в текстовое поле.

Содержимое файла можно прочитать методом (функцией) input() модуля fileinput, который может принимать в качестве аргумента адрес файла. Далее с помощью цикла for можно извлекать строки последовательно и помещать их в текстовое поле.

```
.....
import fileinput
.....
for i in fileinput.input(op):
     txt.insert(END,i)
.....
```

Обратите внимание на то, как происходит обращение к функции input модуля fileinput и его импорт. Дело в том, что в Python уже встроена своя функция input (ее назначение абсолютно иное) и во избежание "конфликта" требуется четко указать, какую именно функцию мы имеем ввиду. Поэтому вариант импорта 'from fileinput import input' здесь не подходит.

Окно "Открыть" запускается сразу при выполнении скрипта. На самом деле так не должно быть. Необходимо связать запуск окна с каким-нибудь событием, например, кликом левой кнопкой мыши на пункте меню.

```
from tkinter import *
from tkinter.filedialog import *
import fileinput

def _open():
     op = askopenfilename()
     for l in fileinput.input(op):
          txt.insert(END,l)

root = Tk()

m = Menu(root)
root.config(menu=m)

fm = Menu(m)
m.add_cascade(label="File",menu=fm)
fm.add_command(label="Open...",command=_open)

txt = Text(root,width=40,height=15,font="12")
txt.pack()

root.mainloop()
```

Теперь попробуем сохранять текст, набранный в текстовом поле. Добавим в код пункт меню и следующую функцию:

```
def _save():
     sa = asksaveasfilename()
     letter = txt.get(1.0,END)
     f = open(sa,"w")
     f.write(letter)
     f.close()
```

В переменной sa храниться адрес файла, куда будет производиться запись. В переменной letter – текст, "полученный" из текстового поля. Затем файл открывается для записи, в него записывается содержимое переменной letter, и файл закрывается .

8.2. Информационные диалоговые окна

Еще одна группа диалоговых окон, описанных в модуле tkinter.messagebox, предназначены для вывода сообщений, предупреждений, получения от пользователя ответа "да" или "нет" и т. п.

Дополним нашу программу пунктом Exit в подменю File и пунктом About program в подменю Help.

```
from tkinter.messagebox import *
….
def close_win():
     if askyesno("Exit", "Вы хотите выйти?"):
          root.destroy()

def about():
     showinfo("Editor", "Редактор.\n(тестовый)")
...
fm.add_command(label="Exit",command=close_win)
....
hm = Menu(m)
m.add_cascade(label="Help",menu=hm)
hm.add_command(label="About",command=about)
...
```

В функции about происходит вызов окна showinfo, позволяющее выводить сообщение для пользователя с кнопкой OK. Первый аргумент — это то, что выводится в заголовке окна, а второй — то, что будет содержаться в теле сообщения.

В функции close_win вызывается окно askyesno, которое позволяет получить от пользователя два ответа (true и false). В данном случае при положительном ответе сработает ветка if и главное окно будет закрыто. В случае нажатия пользователем кнопки "No" окно просто закроется (хотя можно было запрограммировать в ветке else какое-либо действие).

8.3. Практическая работа.

1. Напишите программу, описанную в разделе 8.

2. Измените программу: пусть после нажатия пункта Exit пользователю выводилось не окно с вопросом "выйти или нет", а окно с вопросом "сохранить или нет". В случае положительного ответа должна вызываться функция _save и только затем завершаться приложение.

3. Если в текстовом поле что-то содержится, то при открытии файла оно не удаляется, а содержимое файла просто дописывается. Исправьте этот недостаток (перед открытием файла содержимое текстового поля должно удаляться).

9. Геометрические примитивы графического элемента Canvas (холст)

9.1. Canvas (холст)

Canvas (холст) — объект библиотеки tkinter, позволяющий располагать на самом себе другие объекты. Это могут быть как геометрические фигуры, узоры, вставленные изображения, так и другие виджеты (например, метки, кнопки, текстовые поля).

Отображенные на холсте объекты можно изменять и перемещать в процессе выполнения скрипта. Canvas находит широкое применение при создании GUI-приложений с использованием tkinter (создание рисунков, оформление других виджет, реализация функций графических редакторов, программируемая анимация и др.).

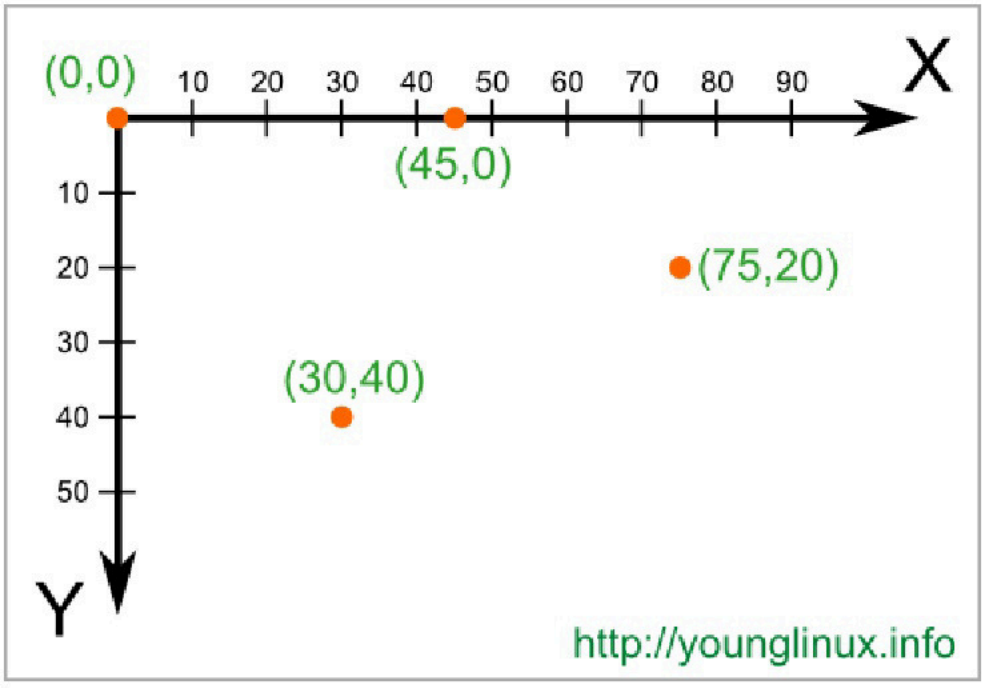
Для того, чтобы создать объект-холст необходимо вызвать соответствующий класс модуля tkinter и установить некоторые значения свойств (опций). Например:

```
canv = Canvas(root, width=500, height=500,
              bg="lightblue", cursor="pencil")
```

Далее с помощью менеджера геометрии разместить на главном окне.

9.2. Система координат

Нулевая точка (0,0) для объекта Canvas располагается в верхнем левом углу. Единицы измерения пиксели (точки экрана). У любой точки первое число — это расстояние от нулевого значения по оси X, второе — по оси Y.



9.3. Графические примитивы Canvas

Активно используемыми графическими примитивами являются:

- линия,
- прямоугольник,
- многоугольник (полигон),
- эллипс,
- дуга (сектор),
- текст.

9.3.1. Линия.

Линия определяется метод create_line.

```
canv.create_line(200,50, 300,50, width=3, fill="blue")
canv.create_line(0,0, 100,100, width=2, arrow=LAST)
```

Четыре числа — это пары координат начала и конца линии, т.е в примере первая линия начинается из точки (200,50), а заканчивается в точке (300,50). Вторая линия начинается в точке (0,0), заканчивается — в (100,100).

Свойство fill позволяет задать цвет линии отличный от черного, а arrow – установить стрелку (в конце, начале или по обоим концам линии).

9.3.2. Прямоугольник.

Прямоугольник определяется методом create_rectangle.

```
x = 75
y = 110
canv.create_rectangle(x,y,
                      x+80,y+50,
                      fill="white",
                      outline="blue")
```

Аналогично линии в скобках первыми аргументами прописываются четыре числа. Первые две координаты обозначают верхний левый угол прямоугольника, вторые — правый нижний. В примере показан вариант описания прямоугольника, в котором начальные координаты объекта могут изменяться, а его размер строго регламентирован.

Опция outline определяет цвет границы прямоугольника.

9.3.3. Многоугольник (полигон).

Чтобы создать произвольный многоугольник, требуется задать пары координат для каждой его точки.

```
canv.create_polygon([250,100],[200,150],[300,150],fill="yellow")
```

Квадратные скобки при задании координат используются для удобочитаемости (их можно не использовать).

Свойство smooth задает сглаживание.

```
canv.create_polygon([250,100],[200,150],[300,150],
                    fill="yellow")
canv.create_polygon([300,80],[400,80],
                    [450,75],[450,200],
                    [300,180],[330,160],
                    outline="white", smooth=1)
```

9.3.4. Эллипс.

При создании эллипса задаются координаты гипотетического прямоугольника, описывающего данный эллипс.

```
canv.create_oval([20,200],[150,300],fill="gray50")
```

9.3.5. Дуга (сектор).

Сектор (как часть окружности) создаются методом create_arc. В зависимости от значения опции style можно получить сектор (по умолчанию), сегмент (CHORD) или дугу (ARC).

Координаты задают прямоугольник, в который вписана окружность, из которой «вырезают» сектор, сегмент или дугу. От опций start и extent зависит угол фигуры.

```
canv.create_arc([160,230],[230,330],start=0,
                    extent=140,fill="lightgreen")
canv.create_arc([250,230],[320,330],start=0,extent=140,
                    style=CHORD,fill="green")
canv.create_arc([340,230],[410,330],start=0,extent=140,
                    style=ARC,outline="darkgreen",width=2)
```

9.3.6. Текст.

Текстовая надпись на объекте canvas создается методом create_text.

```
canv.create_text(20,330,
          text="Опыты с графическими примитивами\nна холсте",
          font="Verdana 12", anchor="w",
          justify=CENTER, fill="red")
```

По умолчанию в заданной координате располагается центр текстовой надписи. Чтобы изменить это и, например, разместить по указанной координате левую границу текста, используется параметр anchor (якорь) со значением w (от англ. west – запад).

Другие значения: n, ne, e, se, s, sw, w, nw.

Если букв, задающих сторону привязки две, то вторая определяет вертикальную привязку (вверх или вниз «уйдет» текст от координаты).

Свойство justify определяет выравнивание текста относительно себя самого.

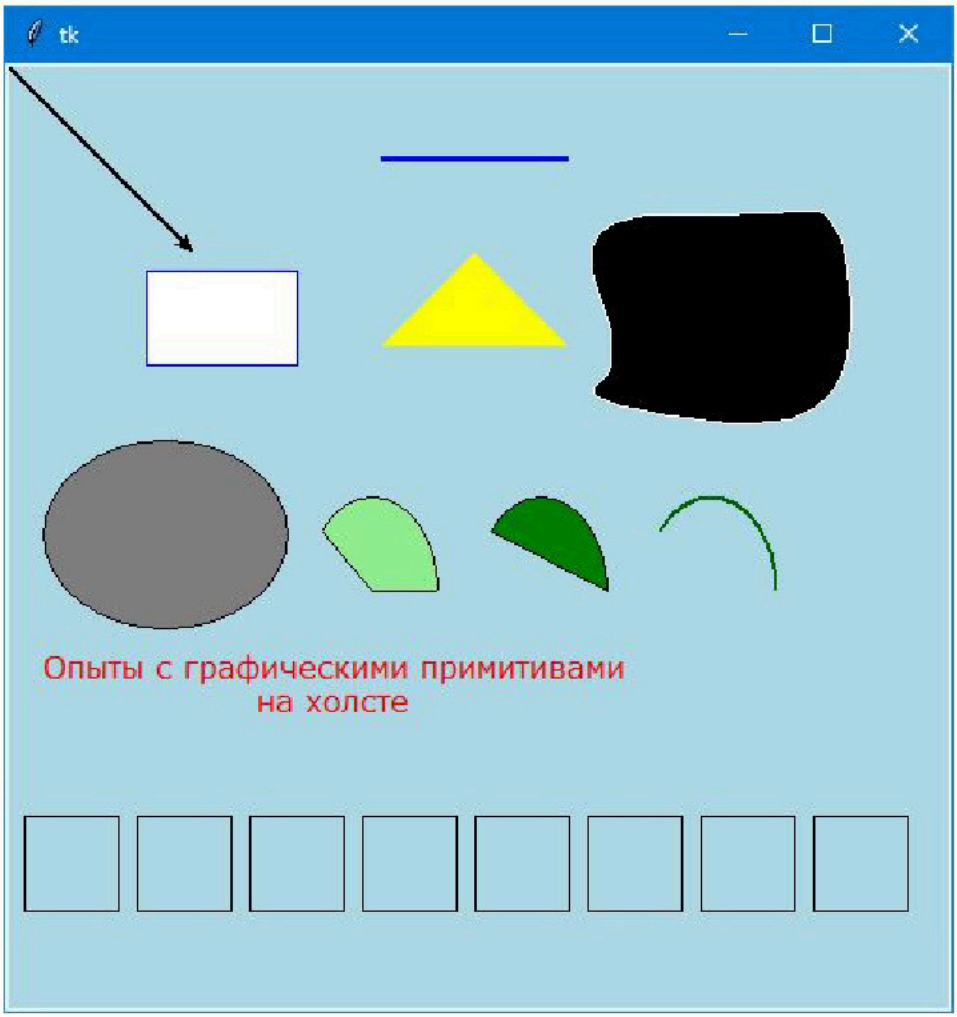
9.3.7. Для создания на холсте повторяющихся элементов, используют циклы.

Например, так:

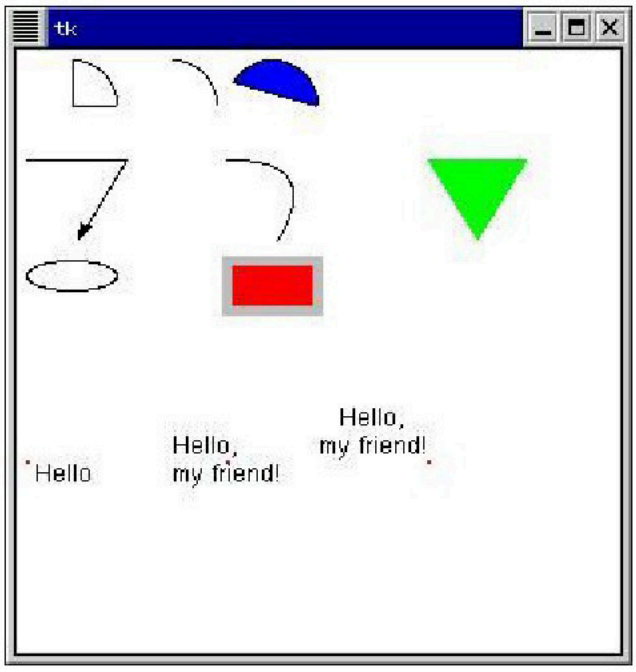
```
x=10
while x < 450:
     canv.create_rectangle(x,400,x+50,450)
     x = x + 60
```

9.4. Практическая работа

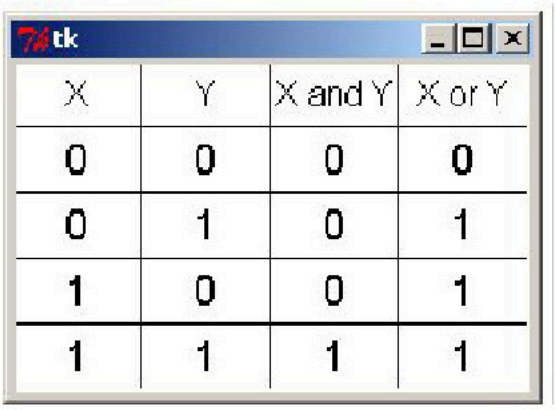
1. Соберите отдельные части кода из раздела 9 в один скрипт и сравните полученный результат с образцом:

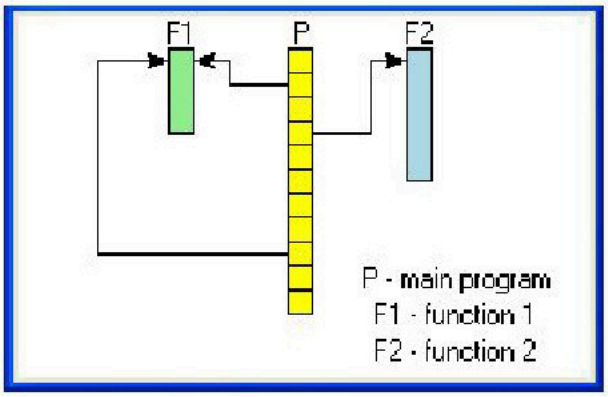


2. Запрограммируйте следующие изображения на виджетах-холстах:



3. Запрограммируйте следующие изображения на виджетах-холстах:





4. Реализовать отображения графика функции.

1. y=math.log(x) - синий, сплошная линия.
2. y=math.sqrt(x) - черный, штриховая линия.
3. y=math.cosh(x) - гиперболический косинус - желтый, пунктирная линия.
4. y=math.sinh(x) - гиперболический синус - зеленый, штрихпунктирная линия.
5. y=(4.*x**3-6.*x**2+1.)*math.sqrt(x+1)/(3.-x) - красный, пунктирная линия.
6. y=2.*math.pow(x,1.5) - полукубическая парабола - пурпурный, сплошная линия.
7. y=2.*x**3.+x*x-5*x+7. - кубическая парабола - голубой, сплошная линия.

10. Canvas (холст): методы, идентификаторы и теги.

Методы объекта canvas, формирующие на нем геометрические примитивы и текст, лишь часть методов холста.

В другую группу выделяют методы, изменяющие свойства уже существующих объектов холста (например, геометрических фигур).

Для обращения к уже созданным графическим элементам в tkinter для объектов холста можно использовать идентификаторы и теги, которые затем передаются другим методам. У любого объекта может быть как идентификатор, так и тег. Использование идентификаторов и тегов различно.

10.1. Идентификаторы.

Рассмотрим методы изменения уже существующих объектов с использованием при этом идентификаторов. Для начала создадим холст и три объекта на нем. При создании объекты "возвращают" свои идентификаторы, которые можно связать с переменными (oval, rect и trial в примере ниже) и потом использовать их для обращения к конкретному объекту.

```
c = Canvas(width=460,height=200,bg='grey80')
c.pack()
oval = c.create_oval(30,10,130,80)
rect = c.create_rectangle(180,10,280,80)
trian = c.create_polygon(330,80,380,10,430,80,
                         fill='grey80', outline="black")
```

Данный скрипт изображает на холсте три фигуры: овал, прямоугольник и треугольник.

Далее можно использовать методы-"модификаторы", указывая в качестве первого аргумента идентификатор объекта:

- метод move перемещает объект по оси X и Y на расстояние указанное в качестве второго и третьего аргументов. Следует понимать, что это не координаты, а смещение, т. е. в примере ниже прямоугольник опустится вниз на 150 пикселей,
- метод itemconfig изменяет указанные свойства объектов,
- метод coords изменяет координаты и размер объекта.

```
c.move(rect, 0,150)
c.itemconfig(trian, outline="red", width=3)
c.coords(oval, 300,200,450,450)
```

Если запустить скрипт, содержащий две приведенные части кода (друг за другом), то мы сразу увидим уже изменившуюся картину на холсте: прямоугольник опустится, треугольник приобретет красный контур, а эллипс сместится и увеличится в размерах.

Обычно в программах изменения должны наступать при каком-нибудь внешнем воздействии. Пусть по щелчку левой кнопкой мыши прямоугольник передвигается на два пикселя вниз (он будет это делать при каждом щелчке мышью):

```
def mooove(event):
     c.move(rect,0,2)
...
c.bind('<Button-1>',mooove)
```

10.2. Теги.

В отличие от идентификаторов, которые являются уникальными для каждого объекта, один и тот же тег может присваиваться разным объектам.

Дальнейшее обращение к такому тегу позволит изменить все объекты, в которых он был указан. В примере ниже эллипс и линия содержат один и тот же тег, а функция color изменяет цвет всех объектов с тегом group1.

Обратите внимание, что в отличие от имени идентификатора (переменная), имя тега заключается в кавычки (строковое значение).

```
oval = c.create_oval(30,10,130,80,tag="group1")
c.create_line(10,100,450,100,tag="group1")
…
def color(event):
     c.itemconfig('group1',fill="red",width=3)
…
c.bind('<Button-3>',color)
```

10.3. Дополнительные методы

Метод delete - удаляет объект по указанному идентификатору или тегу. В tkinter существуют зарезервированные теги: например, ‘all’ обозначает все объекты холста. Так в примере ниже функция clean просто очищает холст.

```
def clean(event):
     c.delete('all')
...
c.bind('<Button-2>',clean)
```

Метод tag_bind - позволяет привязать событие (например, щелчок кнопкой мыши) к определенному объекту. Таким образом, можно реализовать обращение к различным областям холста с помощью одного и того же события. Пример ниже это наглядно иллюстрирует: изменения на холсте зависят от того, где произведен щелчок мышью.

```
from tkinter import *

c = Canvas(width=460,height=100,bg='grey80')
c.pack()

oval = c.create_oval(30,10,130,80,fill="orange")
c.create_rectangle(180,10,280,80,tag="rect",fill="lightgreen")
trian = c.create_polygon(330,80,380,10,430,80,
                         fill='white',outline="black")
def oval_func(event):
     c.delete(oval)
     c.create_text(30,10,text="Здесь был круг",anchor="w")

def rect_func(event):
     c.delete("rect")
     c.create_text(180,10,text="Здесь был\nпрямоугольник",
                   anchor="nw")
def triangle(event):
     c.create_polygon(350,70,380,20,410,70,
```

```
                        fill='yellow',outline="black")

c.tag_bind(oval,'<Button-1>',oval_func)
c.tag_bind("rect",'<Button-1>',rect_func)
c.tag_bind(trian,'<Button-1>',triangle)

mainloop()
```

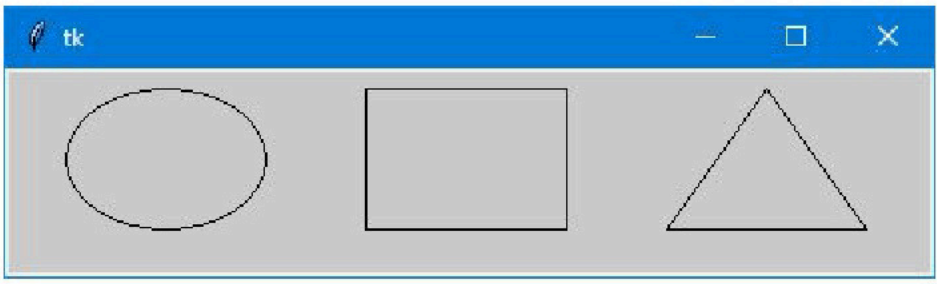
10.4. Практическая работа

1. Прокомментируйте в отчете все блоки скрипта из последнего примера. Добавьте в отчет начальный и конечный скриншот работы скрипта.

.

2. Создайте скрипт, отображающий три фигуры.

.



По нажатию левой кнопки мыши две фигуры из трех должны перемещаться влево, по нажатию правой кнопки эти же две фигуры должны перемещаться вправо. По нажатию средней кнопки должен изменяться цвет всех трех фигур случайным образом.

3. Создайте скрипт, отображающий цветной круг. При щелчке левой кнопкой мыши по кругу, случайным образом должен меняться цвет круга и центр круга должен смещаться в позицию курсора мыши.

11. Особенности работы с виджетом Text

Графический элемент Text, аналогично элементу Canvas (Холст), является корневым графическим виджетом, предоставляя больше возможностей для работы с текстовой информацией.

Помимо разнообразных операций с текстом и его форматированием в экземпляр объекта Text можно вставлять другие стандартные виджеты.

В данном разделе рассматриваются лишь некоторые возможности виджета Text на примере создания окна с текстовым полем, содержащим форматированный текст, кнопку и возможность добавления экземпляров холста.

11.1. Создание текстового поля, определяя некоторые из его свойств:

```
#текстовое поле и его первоначальные настройки
tx = Text(font=('times',12),width=50,height=15,wrap=WORD)
tx.pack(expand=YES,fill=BOTH)
```

11.2. Метод insert

Добавить текст можно методом insert, передавая ему два обязательных аргумента:

- место, куда вставить,
- объект, который следует вставить.

Объектом может быть строка, переменная, ссылающаяся на строку или какой-либо другой объект.

Место вставки указывается с помощью индексов, записываемых в виде 'x.y', где x – это строка, а y – столбец. При этом нумерация строк начинается с единицы, а столбцов с нуля. Например, первый символ в первой строке имеет индекс '1.0', а десятый символ в пятой строке — '5.9'.

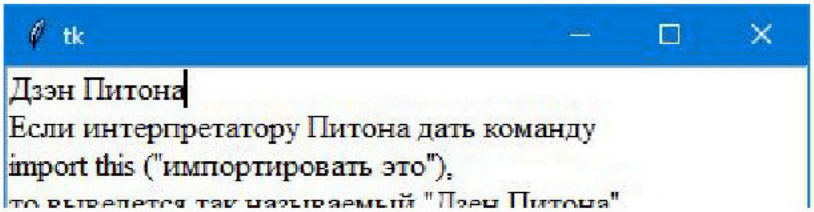
Пример:

```
tx.insert(1.0,'Дзэн Питона\n\
Если интерпретатору Питона дать команду\n\
import this ("импортировать это"),\n\
то выведется так называемый "Дзен Питона".\n
Некоторые выражения:\n\
* Если реализацию сложно объяснить — это плохая идея.\n\
* Ошибки никогда не должны замалчиваться.\n\
* Явное лучше неявного.\n\n')
```

Комбинация символов '\n' создает новую строку (т.е. при интерпретации последующий текст начнется с новой строки). Одиночный символ '\' никак не влияет на отображение текста при выполнении кода, его следует вставлять при переносе текста при написании скрипта.

Когда содержимого текстового поля нет вообще, то единственный доступный индекс — '1.0'. В заполненном текстовом поле вставлять можно в любое место (где есть содержимое).

Если выполнить скрипт, содержащий только данный код (+ импорт модуля Tkinter, + создание главного окна, + mainloop() в конце), то мы увидим текстовое поле с восемью строчками не форматированного текста.



11.3. Форматирование текста

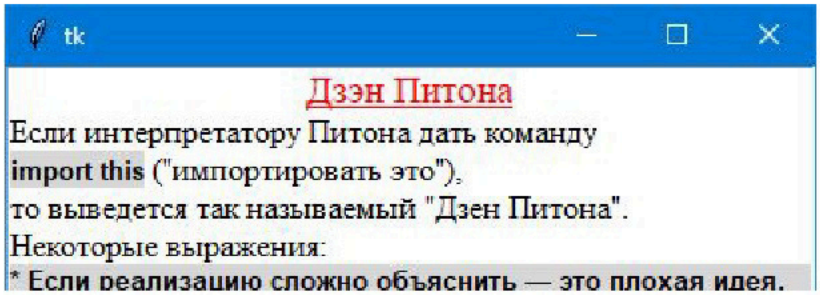
Для применения свойств к тексту сначала необходимо определить теги для нужных областей, а затем для каждого тега установить настройки шрифта и другие свойства текста.

```
#установка тегов для областей текста
tx.tag_add('title','1.0','1.end')
tx.tag_add('special','6.0','8.end')
tx.tag_add('special','3.0','3.11')

#конфигурирование тегов
tx.tag_config('title',foreground='red',
              font=('times',14,'underline'),
              justify=CENTER)
tx.tag_config('special',background='grey85',
              font=('Dejavu',10,'bold'))
```

Добавление тега осуществляется с помощью метода tag_add. Первый атрибут — имя тега, далее с помощью индексов указывается к какой области текстового поля он прикрепляется (начальный символ и конечный). Вариант записи как '1.end' говорит о том, что нужно взять текст до конца указанной строки.

Разные области текста могут быть помечены одинаковым тегом.

Метод tag_config применяет те или иные свойства к тегу, указанному в качестве первого аргумента.



11.4. Добавление объектов

Метод window_create добавляет в многострочное текстовое поле другие виджеты. Например, кнопку с функцией, которую активирует кнопка.

```
def erase():
     tx.delete('1.0',END)
...
#добавление кнопки
bt = Button(tx,text='Стереть',command=erase)
tx.window_create(END,window=bt)
```

В качестве первой опции указывается место добавления, а второй (window) — в качестве значения присваивается переменная, связанная с объектом.

При щелчке ЛКМ (левой кнопкой мыши) по кнопке будет вызываться функция erase, в которой с помощью метода delete удаляется все содержимое поля (от '1.0' до END).

В следующем скрипте при щелчке ЛКМ в любом месте текстового поля будет вызываться функция smiley. В теле данной функции создается объект холста, который в конце с помощью метода window_create добавляется на объект tx. Место вставки указано как CURRENT, т. е. "текущее" - это там, где был произведен щелчок мышью.

```
def smiley(event):
     cv = Canvas(height=30,width=30)
     cv.create_oval(1,1,29,29,fill="yellow")
     cv.create_oval(9,10,12,12)
     cv.create_oval(19,10,22,12)
     cv.create_polygon(9,20,15,24,22,20)
     tx.window_create(CURRENT,window=cv)
…
#ЛКМ -> смайлик
tx.bind('<Button-1>',smiley)
```

11.5. Практическая работа.

1. Соберите приведенный в разделе код в скрипт, исправьте возможные ошибки.. Прокомментируйте в отчете все блоки кода. Добавьте в отчет начальный (без форматирования) и конечный скриншот работы скрипта.

2. Измените функцию erase таким образом, чтобы удалялся не весь текст, а только третья строка.

3. Привяжите оставшиеся области текста к третьему тегу и с помощью метода tag_config измените шрифт, сделав его синим и наклонным.

4. Добавьте в скрипт функцию, которая при нажатии правой кнопки мыши вставляет в текстовое поле “злой” смайлик красного цвета.