Создание GUI на Python с помощью библиотеки Tkinter. Программирование для начинающих

Цикл практических уроков по программированию для начинающих "Создание GUI на Python с помощью библиотеки Tkinter" знакомит с особенностями разработки графического интерфейса пользователя на основе языка программирования Python(создание виджет и обработчиков событий, связывание с событиями и др.).

Курс требует наличия начальных знаний по основам программирования и синтаксису Python (достаточен уровень понимания структурного программирования).

На данный момент курс включает 14 уроков, три из которых (4-й, 5-й и 6-й), знакомящие с менеджерами расположения, можно выделить из статьи "Применение управляющих размещениями Tkinter"

По идеи данный курс должен сформировать у обучающихся базовые понятия построения графического интерфейса пользователя (GUI), а также расширить знания о программировании.

Примечание. В Python ветки 2.x.x при импорте модуля Tkinter, он пишется в большой буквы (иначе возникает ошибка). При импорте tkinter в версиях 3.x.x Питона имя модуля пишется с маленькой буквы.

Введение в tkinter. Урок 1

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.

Современный пользователь в основном взаимодействует с программой с помощью различных кнопок, меню, значков, вводя информацию в специальные поля, выбирая определенные значения в списках и т. д. Эти "изображения" в определенном смысле и формируют GUI, в дальнейшем мы их будем называть виджетами (от англ. widget - "штучка").

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. Мы не будем вдаваться в подробности разработки, а рассмотрим какие этапы приблизительно нужно пройти при программировании, чтобы получить программу 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. 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 строчка программы должна выглядеть так:

```
from tkinter import *
```

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 (хотя понятно, что можно назвать как угодно, но так уж принято). Вторая строчка кода:

```
root = Tk()
```

3. Создание виджет

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)
```

4. Установка свойств виджет

У кнопки много свойств: размер, цвет фона и надписи и др. Мы рассмотрим их на следующем уроке. Пока же установим всего одно свойство — текст надписи (text):

```
but["text"] = "Печать"
```

5-6. Определение событий и их обработчиков

Многообразие событий и способов их обработки будет рассмотрено на следующих уроках. Здесь же просто коснемся данного вопроса в связи с потребностью.

Что же будет делать кнопка и в какой момент она это будет делать? 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 ("Как всегда очередной 'Hello World!'")
```

Не забывайте, что 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)
```

7. Размещение виджет

Если вы заметите, то в любом приложении виджеты не разбросаны по окну как попало, а хорошо организованы, интерфейс продуман до мелочей и обычно подчинен

определенным стандартам. До стандартов нам далеко, нужно просто кнопку как-то отобразить в окне. Самый простой способ — это использование метода pack.

```
but.pack()
```

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

8. Отображение главного окна

Ну и наконец, главное окно тоже не появится, пока не будет вызван специальный метод mainloop:

```
root.mainloop()
```

Данная строчка кода должна быть всегда в конце скрипта!

В итоге, код программы может выглядеть таким образом:

```
from tkinter import *

def printer(event):
     print ("Как всегда очередной 'Hello World!'")

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

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

При программировании графического интерфейса пользователя более эффективным оказывается объектно-ориентированный подход. Поэтому многие «вещи» оформляются в виде классов. В нашем примере также можно использовать класс:

```
from tkinter import *

class But_print:
     def __init__(self):
          self.but = Button(root)
          self.but["text"] = "Печать"
          self.but.bind("<Button-1>",self.printer)
          self.but.pack()
     def printer(self,event):
          print ("Как всегда очередной 'Hello World!'")

root = Tk()
obj = But_print()
root.mainloop()
```

Практическая работа

1. Импортируйте модуль tkinter, создайте объект главного окна, примените к нему метод mainloop. Затем выполните скрипт. Что вы видите?
2. Добавьте кнопку на главное окно с помощью такой команды:

   but = Button(root, text="Печать")

   В данном случае, при создании кнопки, в класс сразу передается и значение

свойства text. Это наиболее часто используемый способ установки свойств (по-сравнению с тем, который приводится в уроке: but["text"] = "Печать").

3. Расположите виджету на главном окне с помощью метода pack. Запустите скрипт. Что вы видите? Нажмите левой кнопкой мыши на кнопку в окне. Что-нибудь происходит?
4. Создайте какую-нибудь функцию и свяжите ее с событием нажатия кнопки.
5. Снова запустите скрипт и нажмите кнопку. По идее, должно что-то произойти.

Виджеты (графические объекты) и их свойства. Часть 1. Урок 2

На этом уроке рассмотрим часть графических объектов (виджет), содержащихся в библиотеке Tkinter: кнопки, поля для ввода, метки, флажки, переключатели и списки. 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.

Кнопки

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 (родит_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).

Метки

Метки (или надписи) — это достаточно простые 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")
```

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).

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.

Радиокнопки (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 будет var будет 2. Изначально также требуется установить значение переменной (выражение var.set(1) задает значение переменной var равное 1).

Флажки

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)
```

Списки

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.

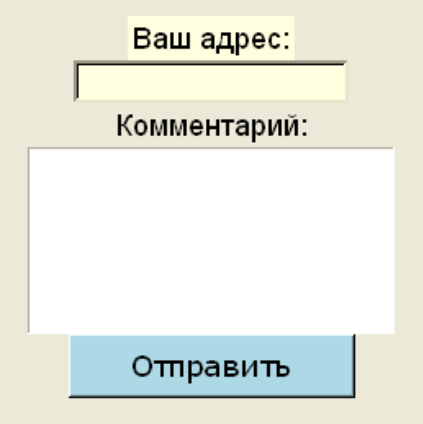
```
r = ['Linux','Python','Tk','Tkinter']
lis = Listbox(root,selectmode=SINGLE,height=4)
for i in r:
     lis.insert(END,i)
```

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

Большинство методов различных виджет мы рассмотрим по ходу изучения данного курса.

Практическая работа

Создайте два скрипта на языке программирования Python и с использованием модуля Tkinter генерирующие шаблоны представленные ниже.



Виджеты (графические объекты) и их свойства. Часть 2. Урок 3

На этом уроке продолжим рассматривать графические объекты (виджеты), содержащихся в библиотеке Tkinter. Это будут рамка (frame), шкала (scale), полоса прокрутки (scrollbar), окно верхнего уровня (toplevel).

Frame (рамка)

Как выяснится позже, 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 boderwidth) 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()
```

Scale (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.

Scrollbar (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 незнакомый нам пока еще метод grid, представляющий собой другой способ расположения виджет на окне.

Toplevel (окно верхнего уровня)

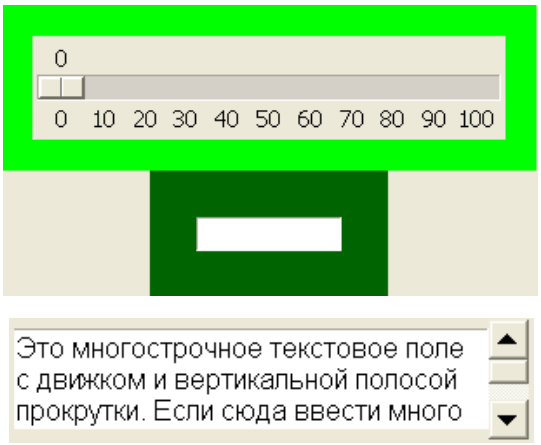
С помощью класс Toplevel создаются дочерние окна, на которых также могут располагаться виджеты. Следует отметить, что при закрытии главного окна (или родительского), окно Toplevel также закрывается. С другой стороны, закрытие дочернего окна не приводит к закрытию главного.

```
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.

практическая работа

1. Создайте два скрипта на языке программирования Python и 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.

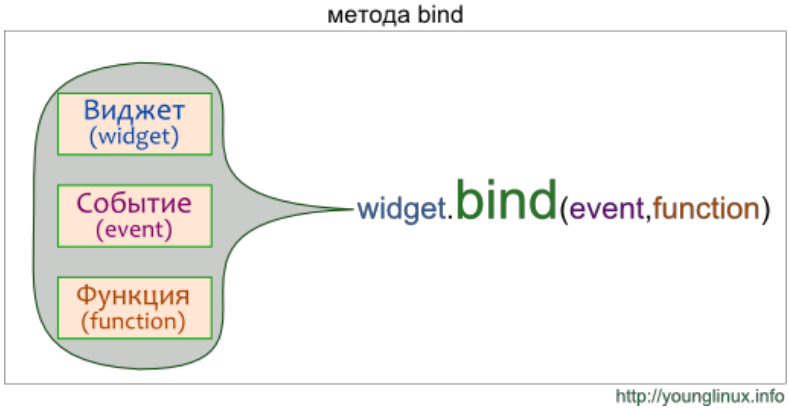
Метод bind модуля Tkinter. Урок 7

Приложения с графическим интерфейсом пользователя (GUI) должны не просто красиво отображаться на экране, но и выполнять какие-либо действия, реализуя тем самым потребности пользователя. На прошлых уроках было рассказано как создать GUI, на этом уроке рассмотрим как добавить ему функциональность, т.е возможность совершать с его помощью те или иные действия.

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. Из такого принципа работы можно вывести следующую схему настройки функциональности GUI: на виджет что-то «влияет» из вне ? выполняется какая-то функция (действие). Внешнее воздействие на графический компонент называется событием. Событий достаточно много (основной их перечень мы рассмотрим на следующем занятии). На этом занятии будем использовать лишь два вида событий: щелчок левой кнопкой мыши () и нажатие клавиши Enter ().

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

Добавление функциональности графическому элементу с помощью метода bind



Рассмотрим различные примеры добавления функциональности GUI.

Пример 1.

```
def output(event):
     s = ent.get()
     if s == "1":
          tex.delete(1.0,END)
          tex.insert(END,"Обслуживание клиентов на втором этаже")
     elif s == "2":
          tex.delete(1.0,END)
          tex.insert(END,"Пластиковые карты выдают в соседнем здании")
     else:
          tex.delete(1.0,END)
          tex.insert(END,"Введите 1 или 2 в поле слева")

from tkinter import *
root = Tk()

ent = Entry(root,width=1)
but = Button(root,text="Вывести")
tex = Text(root,width=20,height=3,font="12",wrap=WORD)

ent.grid(row=0,column=0,padx=20)
but.grid(row=0,column=1)
tex.grid(row=0,column=2,padx=20,pady=10)

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

root.mainloop()
```

Рассмотрим код, начиная с 16-й строки.

В строках 16-18 создаются три виджета: однострочное текстовое поле, кнопка и многострочное текстовое поле. В первое поле пользователь должен что-то ввести, затем нажать кнопку и получить ответ во втором поле.

В строках 20-22 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 24 как раз и происходит 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.

Итак, если вдруг пользователь щелкнет левой кнопкой мыши по кнопке, то выполнится функция output (ни в каком другом случае она выполняться не будет). 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.

```
li = ["red","green"]
def color(event):
     fra.configure(bg=li[0])
     li[0],li[1] = li[1],li[0]

def outgo(event):
     root.destroy()

from tkinter import *
root = Tk()

fra = Frame(root,width=100,height=100)
but = Button(root,text="Выход")

fra.pack()
but.pack()

root.bind("<Return>",color)
but.bind("<Button-1>",outgo)

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).

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.

Программирование событий в Tkinter. Урок 8

Методическая разработка урока
Элективный курс: Модуль tkinter. Создание графического интерфейса пользователя с помощью языка программирования Python
Уровень: Программирование для начинающих

Обычно, чтобы графическое приложение что-то сделало, должно случиться какое-нибудь событие, т. е. воздействие на GUI из вне.

Типы событий

Можно выделить три основных типа событий: производимые мышью, нажатиями клавиш на клавиатуре, а также события, возникающие в результате изменения других графических объектов.

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.

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

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.

```
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.

практическая работа

1. Напишите следующую программу. 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.

Переменные Tkinter. Урок 9

Методическая разработка урока
Элективный курс: Модуль tkinter. Создание графического интерфейса пользователя с помощью языка программирования Python
Уровень: Программирование для начинающих

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 таких 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.

Во втором уроке 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.

Несколько сложнее обстоит дело с флажками. 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.

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).

Объект Меню (Menu) в GUI. Урок 10

Что такое меню

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.

Создание меню в Tkinter

```
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

fm = Menu(m) #создается пункт меню с размещением на основном меню (m)
m.add_cascade(label="File",menu=fm) #пункту 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")

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 создать вложенное меню. Для этого создается еще одно меню и с помощью add_cascade привязать к родительскому пункту.

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

Привязка функций к меню

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

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)
```

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.

Примерный ответ к практической работе

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

def colorR():
     fra.config(bg="Red")
def colorG():
     fra.config(bg="Green")
def colorB():
     fra.config(bg="Blue")

def square():
     fra.config(width=500)
     fra.config(height=500)
def rectangle():
     fra.config(width=700)
     fra.config(height=400)

fra = Frame(root,width=300,height=100,bg="Black")
fra.pack()

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

cm = Menu(m)
m.add_cascade(label="Color",menu=cm)
cm.add_command(label="Red",command=colorR)
cm.add_command(label="Green",command=colorG)
cm.add_command(label="Blue",command=colorB)

sm = Menu(m)
m.add_cascade(label="Size",menu=sm)
sm.add_command(label="500x500",command=square)
sm.add_command(label="700x400",command=rectangle)
```

```
root.mainloop()
```

Диалоговые окна в Tkinter. Урок 11

Методическая разработка урока
Элективный курс: Модуль tkinter. Создание графического интерфейса пользователя с помощью языка программирования Python
Уровень: Программирование для начинающих
*Версии Python: 3.**

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

Диалоговые окна весьма разнообразны. В данном уроке будут рассмотрены лишь несколько.

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.

Давайте теперь разместим многострочное текстовое поле на главном окне и в дальнейшем попробуем туда загружать содержимое небольших текстовых файлов. Поскольку окно сохранения файла нам пока не нужно, то закомментируем эту строчку кода или удалим. В результате должно получиться примерно так:

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

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

op = askopenfilename()

root.mainloop()
```

При запуске скрипта появляется окно с текстовым полем и сразу диалоговое окно "Открыть". Однако, если мы попытаемся открыть какой-нибудь текстовый файл, то в

лучшем случае ничего не произойдет. Как же связать содержимое текстового файла с текстовым полем через диалог "Открыть"?

Что если просто вставить содержимое переменной op в текстовое поле:

```
txt.insert(END,op)
```

После запуска скрипта и попытки открытия файла в текстовом поле оказывается адрес файла. Значит содержимое файла надо прочитать каким-то методом (функцией).

Метод input модуля fileinput может принимать в качестве аргумента адрес файла, читать его содержимое, формируя список строк. Далее с помощью цикла for можно извлекать строки последовательно и помещать их, например, 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. Пусть это будет щелчок 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 (на всякий случай).

Еще одна группа диалоговых окон описана в модуле tkinter.messagebox. Это достаточно простые диалоговые окна 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", "Do you want to quit?"):
        root.destroy()

def about():
    showinfo("Editor", "This is text editor.\n(test version)")
...
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 выведется 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).

Практическая работа

1. Напишите программу, описанную в уроке.
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).

Геометрические примитивы графического элемента Canvas (холст) модуля Tkinter. Урок 12

Методическая разработка урока
Элективный курс: Модуль tkinter. Создание графического интерфейса пользователя с помощью языка программирования Python
Уровень: Программирование для начинающих

Canvas (холст) — это достаточно сложный объект библиотеки tkinter. Он позволяет располагать на самом себе другие объекты. Это могут быть как геометрические фигуры, узоры, вставленные изображения, так и другие виджеты (например, метки, кнопки, текстовые поля). И это еще не все. Отображенные на холсте объекты можно изменять и перемещать (при желании) в процессе выполнения скрипта. Учитывая все это, canvas находит широкое применение при создании GUI-приложений с использованием tkinter (создание рисунков, оформление других виджет, реализация функций графических редакторов, программируемая анимация и др.).

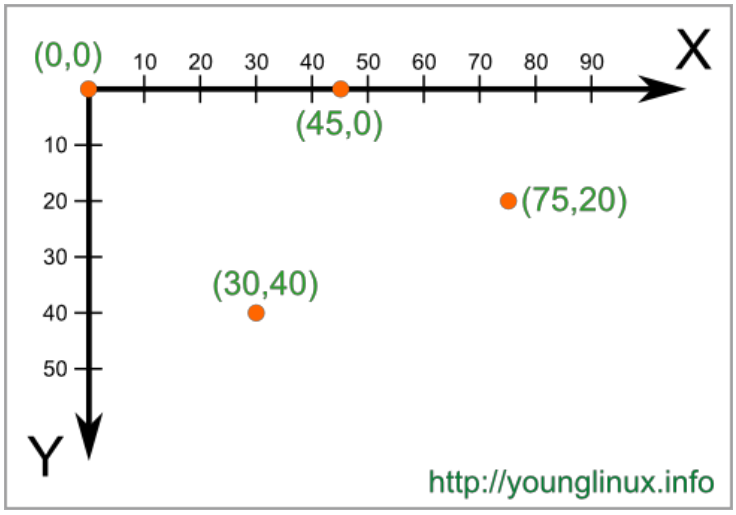
В данном уроке будет рассмотрено создание на холсте графических примитивов (линии, прямоугольника, многоугольника, дуги (сектора), эллипса) и текста.

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 любого менеджера геометрии разместить на главном окне.

Перед тем как создавать геометрические фигуры на холсте следует разобраться с координатами и единицами измерения расстояния. Нулевая точка (0,0) для объекта Canvas располагается в верхнем левом углу. Единицы измерения пиксели (точки экрана). Для «ориентации в пространстве» объекта Canvas рассмотрите рисунок ниже. У любой точки первое число — это расстояние от нулевого значения по оси X, второе — по оси Y.



Чтобы нарисовать линию на холсте следует к объекту (в нашем случае, canv) применить 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).

Метод create_rectangle создает прямоугольник. Аналогично линии в скобках первыми аргументами прописываются четыре числа. Первые две координаты обозначают верхний левый угол прямоугольника, вторые — правый нижний. В примере ниже используется немного иной подход. Он может быть полезен, если начальные координаты объекта могут изменяться, а его размер строго регламентирован.

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

Опция outline определяет цвет границы прямоугольника.

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)
```

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

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

Более сложные для понимания фигуры получаются при использовании метода create_arc. В зависимости от значения опции style можно получить сектор (по умолчанию), сегмент (CHORD) или дугу (ARC). Координаты по-прежнему 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)
```

Последний метод объекта canvas, который будет рассмотрен в этом уроке — это метод создающий текстовую надпись.

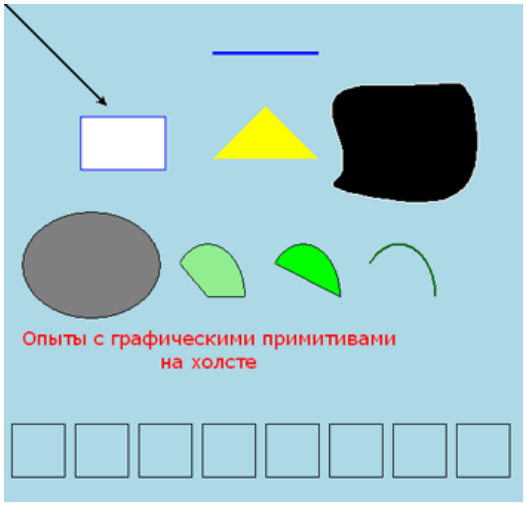
```
canv.create_text(20,330,text="Опыты с графическими примитивами\nна холсте",
          font="Verdana 12",anchor="w",justify=CENTER,fill="red")
```

Трудность здесь может возникнуть с пониманием опции anchor (якорь). По умолчанию в заданной координате располагается центр текстовой надписи. Чтобы изменить это и, например, разместить по указанной координате левую границу текста, используется 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 лишь выравнивание текста относительно себя самого.

В конце следует отметить, что часто требуется «нарисовать» на холсте какие-либо повторяющиеся элементы. Для того, чтобы не загружать код, используют циклы. Например, так:

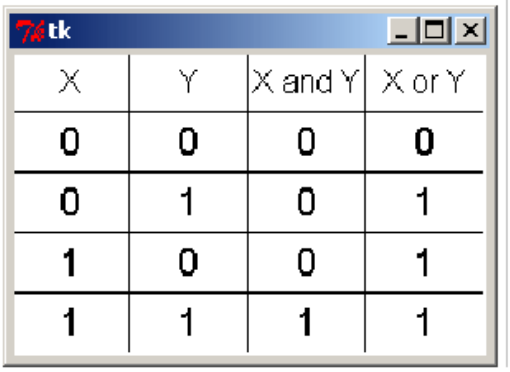
```
x=10
while x < 450:
     canv.create_rectangle(x,400,x+50,450)
     x = x + 60
```

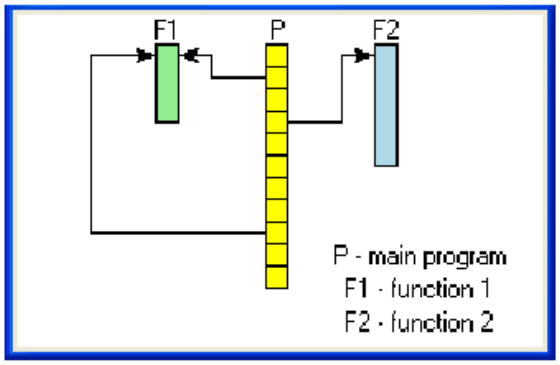
Если вы напишите код приведенный в данном уроке (предварительно совершив импорт модуля Tkinter и создание главного окна, а также не забыв расположить на окне холст, и в конце «сделать» mainloop), то при его выполнении увидите такую картину:



Практическая работа

Запрограммируйте следующие изображения на виджетах-холстах:





Canvas (холст): методы, идентификаторы и теги. Урок 13

Методическая разработка урока
Элективный курс: Модуль tkinter. Создание графического интерфейса пользователя с помощью языка программирования Python
Уровень: Программирование для начинающих

На прошлом уроке были рассмотрены методы объекта canvas, формирующие на нем геометрические примитивы и текст. Однако это лишь часть методов холста. В другую условную группу можно выделить методы, изменяющие свойства уже существующих объектов холста (например, геометрических фигур). И тут возникает вопрос: как обращаться к уже созданным фигурам? Ведь если при создании было прописано что-то вроде canvas.create_oval(30,10,130,80) и таких овалов, квадратов и др. на холсте очень много, то как к ним обращаться?

Для решения этой проблемы 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 немного различается.

Рассмотрим несколько методов 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=460,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")
```

Если вы выполните данный скрипт, то увидите на холсте три фигуры: овал, прямоугольник и треугольник.

Далее можно использовать методы-"модификаторы" указывая в качестве первого аргумента идентификатор объекта. Метод move перемещает объект на 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, coords изменяет координаты (им можно менять 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 сильно 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)
```

Теперь рассмотрим как работают 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)
```

Еще один метод, который стоит рассмотреть, это delete, который 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()
```

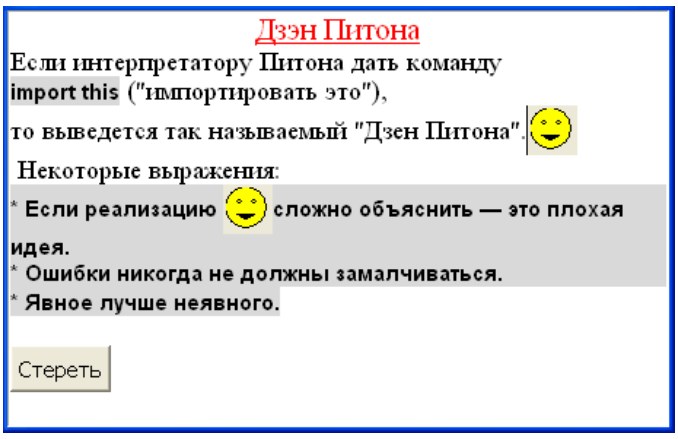
Практическая работа

1. Спишите скрипты, рассмотренные в данном уроке. Выполните их. Объясните увиденное.
2. Подумайте как можно реализовать движение (анимацию) той или иной геометрическое фигуры по холсту. Подсказка: попробуйте использовать цикл while, в теле которого с помощью метода delete удаляется старая фигура, а с помощью move рисуется такая же на новом месте.

Особенности работы с виджетом Text модуля Tkinter. Урок 14

Методическая разработка урока
Элективный курс: Модуль tkinter. Создание графического интерфейса пользователя с помощью языка программирования Python
Уровень: Программирование для начинающих

Графический элемент Text предоставляет большие возможности для работы с текстовой информацией. Помимо разнообразных операций с текстом и его форматированием в экземпляр объекта Text можно вставлять другие виджеты (следует отметить, что такая же возможность существует и для Canvas). В данном уроке рассматриваются лишь некоторые возможности виджета Text на примере создания окна с текстовым полем, содержащим форматированный текст, кнопку и возможность добавления экземпляров холста.



1. Для начала создадим текстовое поле, установив при этом 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)
```

2. Теперь допустим нам нужно добавить какой-нибудь текст. Сделать это можно с помощью метода insert, передав ему два обязательных аргумента: место, куда вставить, и объект, который следует вставить. Объектом может быть строка, переменная, ссылающаяся на строку или какой-либо другой объект. Место вставки может указываться

несколькими способами. Один из них — это индексы. Они записываются 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'.

```
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 текста. Текст не оформлен.

3. Теперь отформатируем разные области текста по-разному. Для этого сначала зададим теги для нужных нам областей, а затем для каждого тега установим настройки шрифта и др.

```
#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 (произвольное), 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.

4. В многострочное текстовое поле можно добавлять не только текст, но и другие объекты. Например, вставим в поле кнопку (ну и функцию заодно).

```
def erase():
     tx.delete('1.0',END)
```

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

Кнопка — это виджет. Виджеты добавляются в текстовое поле с помощью метода window_create, где 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).

5. А вот более интересный пример добавления виджета в поле Text:

```
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)
```

Здесь 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.

Практическая работа:

1. Напишите скрипт, описанный в данном уроке. Выполните его.
2. Измените функцию erase таким образом, чтобы удалялся не весь текст, а только третья строка.
3. Привяжите оставшуюся область текста к третьему тегу и с помощью метода tag_config измените шрифт.
4. Добавьте еще какой-нибудь виджет в текстовое поле.