Тема 13. Объект меню. Диалоговые окна. Геометрические примитивы

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)
```

Упражнение - пример

Напишем 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.

Примерное решение:

```
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()
```

I. Диалоговые окна в Tkinter

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

Диалоговые окна весьма разнообразны. Здесь 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).

II. Геометрические примитивы графического элемента Canvas (холст)

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.).

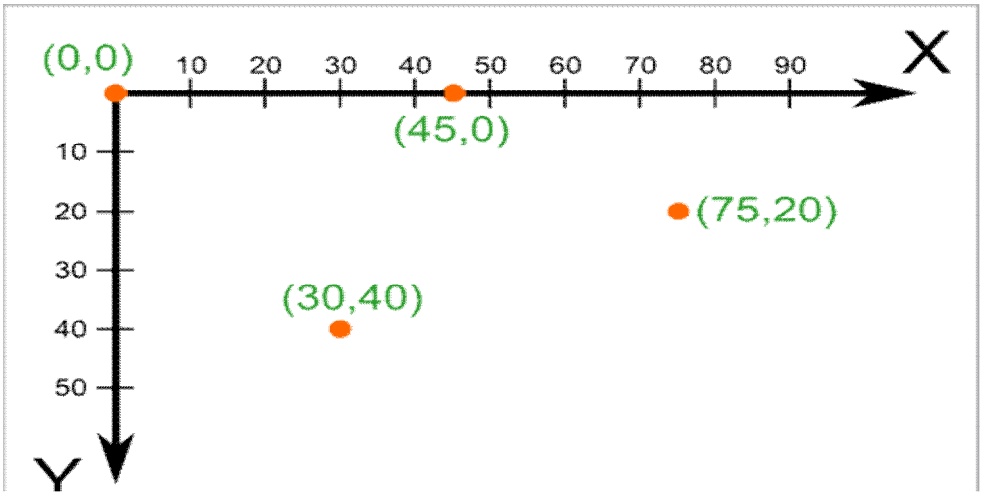
В данном уроке будет рассмотрено создание на холсте графических примитивов (линии, прямоугольника, многоугольника, дуги (сектора), эллипса) и текста.

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

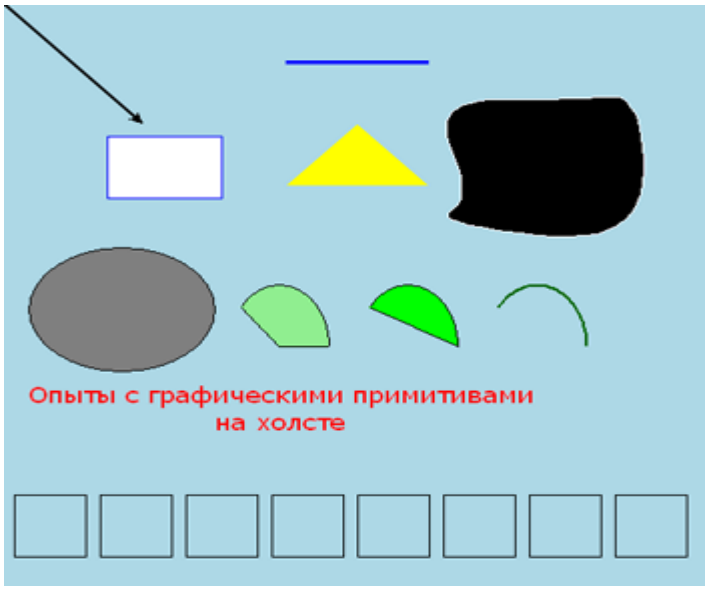
координате левую границу текста, используется якорь со значением w (от англ. `west` – запад). Другие значения: `n, ne, e, se, s, sw, w, nw.`

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:



Canvas (холст) - методы, идентификаторы и теги

Ранне 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)
```

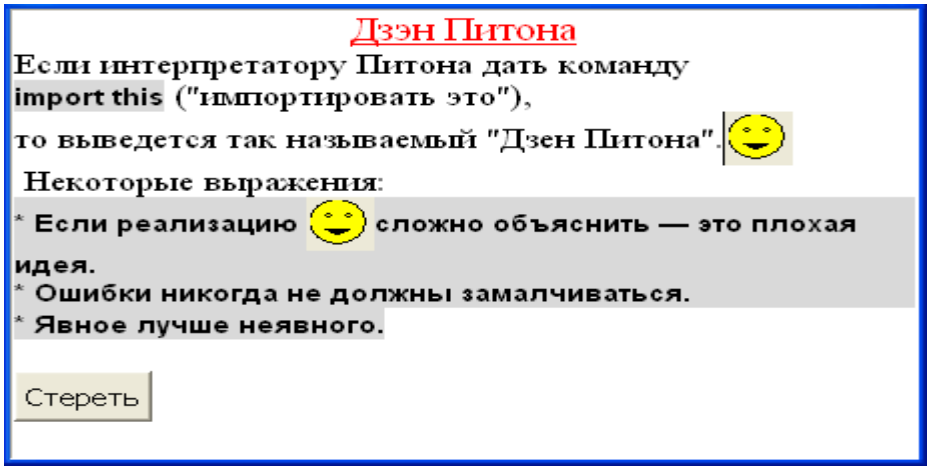
Метод `tag_bind` позволяет привязать событие (например, щелчок кнопкой мыши) к определенному объекту. Таким образом, можно реализовать обращение к различным областям холста с помощью одного и того же события. Пример ниже это наглядно иллюстрирует: изменения на холсте зависят от того, где произведен щелчок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

Особенности работы с виджетом Text модуля Tkinter.

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.