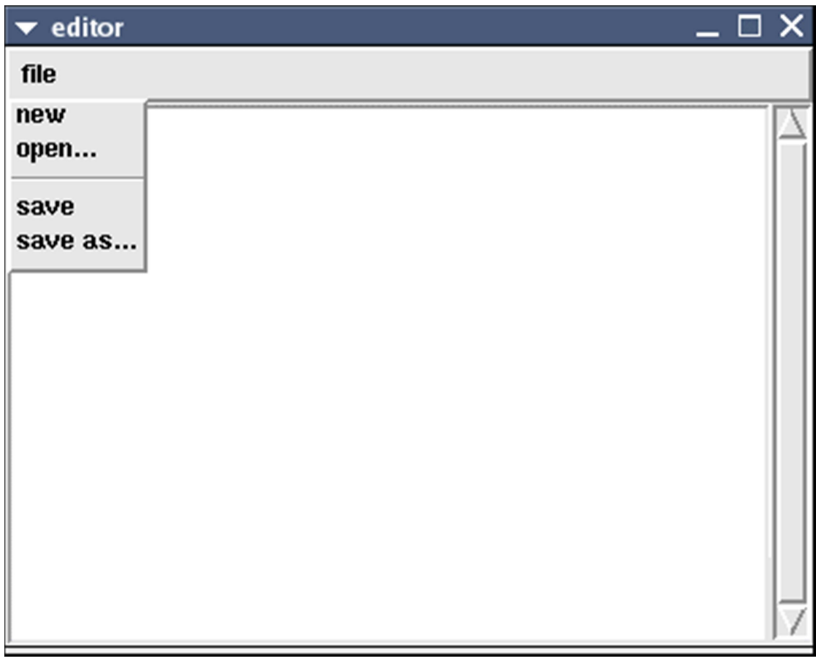
Занятие 31. Основы TKinter. Проектирование окна управления файлами

ПЛАН ЗАНЯТИЯ

1. Главный текстовый элемент управления редактора

1.1 Вид окна



Замечание: Код для этой версии главного окна редактора довольно длинный. Возможно на следующих страницах вам потребуется к нему обращаться. Здесь он не приводится, но его можно вызвать в выпадающем окне, если щелкнуть мышкой по текстовой иконке, расположенной выше [ниже]. И далее в тексте с помощью этой текстовой иконки также можно будет вызывать код, обсуждаемый в том или ином разделе. [В русском переводе оформление документа изменено. Текст программы вывести в выпадающее окно нельзя. Пожалуйста, для просмотра открывайте нужный код в любом текстовом редакторе. Прим. пер.]

1.2 Проектирование окна

Запустите editor_01.py.

Все модули спроектированы так, что их можно запускать как отдельные программы. В конце мы соберем их всех вместе. Фрагмент кода, связанный с текущим обсуждением, будет приводиться прямо здесь.

Главному окну нашего текстового редактора нужны панель меню с двумя заголовками главного меню [`file` (файл) и `edit` (правка)], элемент управления text и линейка прокрутки. Использование метода `pack` для создания интерфейса Tkinter подразумевает, что элементы управления будут размещаться в определенной последовательности. Посмотрите анимацию, потом взгляните на разъяснение и код:

Последовательность проктирования приложения

1. Инициализируем главное окно [`Tk()`].

```
def __init__(self, master):
  self.master = master
  self.master.title('editor')
  self.master.iconname('editor')
  self.master.geometry('600x400+100+100')
```

2. Добавляем фрейм для панели меню, который является дочерним по отношению к главному окну [`Frame( )`].

```
    self.myBar = Frame(self.master, relief = RAISED, bd=2)
```

3. Добавляем в меню пункт *file* [`Menubutton()`], дочерний по отношению к фрейму панели меню, и упаковываем [pack] его к левой стороне [LEFT].

```
    self.fileMenu()
    self.editMenu()
```

4. Теперь упаковываем (pack) заполненный фрейм панели меню к верхней [TOP] стороне окна. Пусть фрейм заполнит все доступное пространство по оси X. Сделаем фрейм расширяемым при изменении размеров окна.

```
    self.myBar = Frame(self.master, relief = RAISED, bd=2)
```

5. Сейчас добавим дочернюю к главному окну линейку прокрутки [`Scrollbar()`]. Упакуем ее [pack] к правой [RIGHT] стороне. Пусть она заполнит все доступное пространство по оси X. И сделаем ее расширяемой при изменении размеров окна.

```
    self.myScroll = Scrollbar(self.master)
    self.myScroll.pack(side=RIGHT, fill=Y)
```

6. Наконец, добавляем элемент управления Text как дочерний к главному окну [`Text()`]. Упаковываем его к левой [LEFT] стороне. Пусть он заполнит свободное место вдоль обоих [BOTH] осей. И сделаем его расширяемым при изменении размеров окна.

```
    self.myText = Text(self.master,
                       background = 'white',
                       height = 30,
                       width = 90,
                       yscrollcommand=(self.myScroll, 'set'))
    self.myText.pack(side=LEFT, fill=BOTH, expand=YES)
```

Сначала это может показаться трудным для понимания, но, если читать вдумчиво, смысл уловить можно. Со временем это будет получаться автоматически. "Ну-ка, посмотрим, что у нас идет вначале?" Вот полный рабочий код. [Просто представьте, что после `fileMenu()` идет `editMenu()`. Я опустил этот кусок кода для ясности. Довольно скоро мы с ним встретимся.]:

класс main

```
# класс родительского окна
class main:
  def __init__(self, master):
    self.master = master
    self.master.title('editor')
    self.master.iconname('editor')
    self.master.geometry('600x400+100+100')
    self.myBar = Frame(self.master, relief = RAISED, bd=2)
    self.fileMenu()
    self.myBar.pack(side = TOP, expand = YES, fill = X)
    self.myScroll = Scrollbar(self.master)
    self.myScroll.pack(side=RIGHT, fill=Y)
    self.myText = Text(self.master,
                       background = 'white',
                       height = 30,
                       width = 90,
                       yscrollcommand=(self.myScroll, 'set'))
    self.myText.pack(side=LEFT, fill=BOTH, expand=YES)
    self.myScroll.configure(command = self.myText.yview)
    self.master.protocol('WM_DELETE_WINDOW', self.exitMethod)
    self.master.mainloop()
```

Строки `fileMenu()` создают меню. Это метод будет описан на следующей странице. После этого мы исследуем взаимосвязь между линейкой прокрутки и элементом управления Text, а также `exitMethod`.

2. Главное меню редактора

На предыдущей странице командой `fileMenu()` в панели главного меню был создан пункт *file*. Он вызывает один из методов `класса main`, чтобы вставить кнопку меню. Так сделано главным образом для ясности [чтобы не было "лишнего" кода].

Чтобы создать главное меню, в Tkinter используются три элемента управления: фрейм `Frame()` как контейнер для кнопок главного меню, кнопка меню `Menubutton()` для каждого пункта главного меню и меню `Menu()`, отображающее команды подменю для каждого пункта главного меню. В коде, приведенном ниже:

- Добавляем элемент `Menubutton()` как дочерний по отношению к фрейму меню [`myBar`].
- Добавляем меню `Menu()`, дочернее по отношению к кнопке меню Menubutton [`mbutton`].
- Используем методы `add_command()` и `add_separator()` класса `Menu()`, чтобы добавить в меню новые пункты.
- Завершаем построение меню командой `mButton.configure(menu = menu)`.

После того как мы заполнили меню [`Menu()`] командами, его следует связать с `Menubutton()`. Метод `configure()` - это способ, каким можно добавлять параметры потом. Параметр `menu = menu` нельзя вставлять до тех пор, пока это меню не будет объявлено и в него не будут введены команды.

метод fileMenu

```
# добавление меню file в панель меню
  def fileMenu(self):
    mButton = Menubutton(self.myBar, text = 'file  ', underline = 0)
    mButton.pack(side = LEFT)
    menu = Menu(mButton, tearoff = 0)
    menu.add_command(label = 'new', command = self.getMessage)
    menu.add_command(label = 'open...', command = self.getMessage)
    menu.add_separator({})
    menu.add_command(label = 'save', command = self.getMessage)
    menu.add_command(label = 'save as...', command = self.getMessage)
    mButton.configure(menu = menu)
    return mButton
```

Все вызовы команд `command` в этих примерах указывают на одну и ту же штуку [`self.getMessage`] - обобщенное или родовое [generic] окно сообщений. Позднее мы вернемся назад и добавим настоящие команды, сделав это приложение функциональным.

3. Элемент управления Text, линейка прокрутки Scrollbar и управление закрытием окна

Вообще говоря, элемент управления Text и линейка прокрутки Scrollbar - отдельные сущности. Но во время работы программы они действуют в общей связки, обмениваясь друг с другом информацией. Вот этот код:

класс main

```
# класс родительского окна
class main:
  def __init__(self, master):
    self.master = master
    self.master.title('editor')
    self.master.iconname('editor')
    self.master.geometry('600x400+100+100')
    self.myBar = Frame(self.master, relief = RAISED, bd=2)
    self.fileMenu()
    self.myBar.pack(side = TOP, expand = YES, fill = X)
    self.myScroll = Scrollbar(self.master)
    self.myScroll.pack(side=RIGHT, fill=Y)
    self.myText = Text(self.master,
                       background = 'white',
                       height = 30,
                       width = 90,
                       yscrollcommand=(self.myScroll, 'set'))
    self.myText.pack(side=LEFT, fill=BOTH, expand=YES)
    self.myScroll.configure(command = self.myText.yview)
    self.master.protocol('WM_DELETE_WINDOW', self.exitMethod)
    self.master.mainloop()
```

```
yscrollcommand=(self.myScroll, 'set')
```

Эта строчка говорит, что элемент управления Text управляется линейкой прокрутки Scrollbar.

```
self.myScroll.configure(command = self.myText.yview)
```

Эта строка говорит, что линейка прокрутки Scrollbar должна отслеживать текст в текстовом элементе управления. Метод configure() дает возможность

вносить изменения в установки свойств. Таким образом self.myScroll = Scrollbar(self.master) фактически становится self.myScroll = Scrollbar(self.master, command = self.myText.yview) [но эта команда не может быть определена, пока не будет объявлен элемент управления Text()].

Строка self.master.protocol('WM_DELETE_WINDOW', self.exitMethod) передает программе управление над закрытием окна. Команда WM_DELETE_WINDOW это часть протокола окна, вызываемого при нажатии кнопки закрытия. Эта строчка перехватывает WM_DELETE_WINDOW и перенаправляет его методу self.exitMethod. Это дает нам возможность спросить у пользователя, действительно ли он хочет выйти, вызвав соответстующий диалог.

метод exitMethod

```
# выход из редактора
  def exitMethod(self):
    self.dialog = yesno(self.master)
    self.myMssg = 'Do you want to exit?'
    self.returnValue = self.dialog.go(message = self.myMssg)
    if self.returnValue:
      self.master.destroy()
```

Итак, у нас есть главное окно текстового редактора с несколькими работающими элементами управлениями. Пришло время заняться диалоговыми окнами меню file.