Урок 1. Что такое Tkinter

Tkinter – это пакет для Python, предназначенный для работы с библиотекой Tk. Библиотека Tk содержит компоненты графического интерфейса пользователя (graphical user interface – GUI), написанные на языке программирования Tcl.

Под графическим интерфейсом пользователя (GUI) подразумеваются все те окна, кнопки, текстовые поля для ввода, скроллеры, списки, радиокнопки, флажки и др., которые вы видите на экране, открывая то или иное приложение. Через них вы взаимодействуете с программой и управляете ею. Все эти элементы интерфейса вместе будем называть виджетами (widgets).

В настоящее время почти все приложения, которые создаются для конечного пользователя, имеют GUI. Редкие программы, подразумевающие взаимодействие с человеком, остаются консольными. В предыдущих двух курсах мы писали только консольные программы.

Существует множество библиотек GUI. Tk далеко не самая популярная, хотя с ее использованием написано не мало проектов. Однако по ряду причин она была выбрана для Python по-умолчанию. Установочный файл Питона обычно уже включает пакет tkinter в составе стандартной библиотеки наряду с другими модулями.

Не вдаваясь в подробности, Tkinter можно охарактеризовать как переводчик с языка Python на язык Tcl. Вы пишете программу на Python, а код модуля tkinter у вас за спиной переводит ваши инструкции на язык Tcl, который понимает библиотека Tk.

Приложения с графическим интерфейсом пользователя событийно-ориентированные. Вы уже должны иметь представление о структурном и желательно объектно-ориентированном программировании. Событийно-ориентированное ориентировано на события. То есть та или иная часть программного кода начинает выполняться лишь тогда, когда случается то или иное событие.

Событийно-ориентированное программирование базируется на объектно-ориентированном и структурном. Даже если мы не создаем собственных классов и объектов, то все-равно ими пользуемся. Все виджеты – объекты, порожденные встроенными классами.

События бывают разными. Сработал временной фактор, кто-то кликнул мышкой или нажал Enter, начал вводить текст, переключил радиокнопки, прокрутил страницу вниз и т. д. Когда случается что-то подобное, то, если был создан соответствующий обработчик, происходит срабатывание определенной части программы, что приводит к какому-либо результату.

Tkinter импортируется стандартно для модуля Python любым из способов: `import tkinter`, `from tkinter import *`, `import tkinter as tk`. Можно импортировать отдельные классы, что делается редко. В данном курсе будет в основном использоваться `from tkinter import *`.

Далее, чтобы написать GUI-программу, надо выполнить приблизительно следующее:

1. Создать главное окно.
2. Создать виджеты и выполнить конфигурацию их свойств (опций).

3. Определить события, то есть то, на что будет реагировать программа.
4. Определить обработчики событий, то есть то, как будет реагировать программа.
5. Расположить виджеты в главном окне.
6. Запустить цикл обработки событий.

Последовательность не обязательно такая, но первый и последний пункты всегда остаются на своих местах. Посмотрим все это в действии.

В современных операционных системах любое пользовательское приложение заключено в окно, которое можно назвать главным, так как в нем располагаются все остальные виджеты. Объект окна верхнего уровня создается от класса Tk модуля tkinter. Переменную, связываемую с объектом, часто называют root (корень):

```
root = Tk()
```

Пусть в окне приложения располагаются текстовое поле (entry), метка (label) и кнопка (button). Данные объекты создаются от соответствующих классов модуля tkinter. Мы сразу сконфигурируем некоторые их свойства с помощью передачи аргументов конструкторам этих классов:

```
e = Entry(root, width=20)
b = Button(root, text="Преобразовать")
l = Label(root, bg='black', fg='white', width=20)
```

Устанавливать свойства объектов не обязательно при из создании. Существуют еще пара способов, с помощью которых можно это сделать после.

Обратите внимание, что первым аргументом указывается "хозяин, мастер" – место, где располагается виджет. В данном случае его можно было не указывать. Однако виджеты не обязательно располагаются на root'e. Они могут размещаться в других виджетах.

Пусть в программе текст, введенный человеком в поле, при нажатии на кнопку разбивается на список слов, слова сортируются по алфавиту и выводятся в метке. Код выполняющий все это надо поместить в функцию:

```
def strToSortlist(event):
      s = e.get()
      s = s.split()
      s.sort()
      l['text'] = ' '.join(s)
```

У функций, которые вызываются при наступлении события с помощью метода bind(), должен быть один параметр. Обычно его называют event (событие).

В приведенной функции с помощью метода get() из поля забирается текст, представляющий собой строку. Она преобразуется в список слов с помощью метода split(). Потом список сортируется. В конце изменяется свойство text метки. Ему присваивается строка, полученная из списка с помощью строкового метода join().

Теперь необходимо связать вызов функции с событием:

```
b.bind('<Button-1>', strToSortlist)
```

В данном случае это делается с помощью метода bind(). Ему передается событие и вызываемая функция. Событие будет передано в функцию и присвоено параметру event. В данном случае событием является щелчок левой кнопкой мыши, что обозначается строкой `'<Button-1>'`.

В любом приложении виджеты не разбросаны по окну как попало, а хорошо организованы, интерфейс продуман до мелочей и обычно подчинен определенным стандартам. Пока расположим элементы друг за другом с помощью наиболее простого менеджера геометрии tkinter – метода pack():

```
e.pack()
b.pack()
l.pack()
```

Метод mainloop() объекта Tk запускает главный цикл обработки событий, что в том числе приводит к отображению главного окна со всеми его причиндалами на экране:

```
root.mainloop()
```

Полный код программы:

```
from tkinter import *

root = Tk()

e = Entry(width=20)
b = Button(text="Преобразовать")
l = Label(bg='black', fg='white', width=20)

def strToSortlist(event):
      s = e.get()
      s = s.split()
      s.sort()
      l['text'] = ' '.join(s)

b.bind('<Button-1>', strToSortlist)

e.pack()
b.pack()
l.pack()
root.mainloop()
```

В результате выполнения данного скрипта появляется окно, в текстовое поле которого можно ввести список слов, нажать кнопку и получить его отсортированный вариант:

Попробуем теперь реализовать в нашей программе объектно-ориентированный подход. Это необязательно, но нередко бывает уместным. Пусть комплект из метки, кнопки и поля представляет собой один объект, порождаемый от некого класса, скажем, Block. Тогда в основной ветке программы будет главное окно, объект типа Block и запуск окна. Поскольку

блок должен быть привязан к главному окну, то неплохо бы передать в конструктор класса окно-родитель:

```
from tkinter import *

root = Tk()

first_block = Block(root)

root.mainloop()
```

Теперь напишем сам класс Block:

```
class Block:
      def __init__(self, master):
            self.e = Entry(master, width=20)
            self.b = Button(master, text="Преобразовать")
            self.l = Label(master, bg='black', fg='white', width=20)
            self.b['command'] = self.strToSortlist
            self.e.pack()
            self.b.pack()
            self.l.pack()
      def strToSortlist(self):
            s = self.e.get()
            s = s.split()
            s.sort()
            self.l['text'] = ' '.join(s)
```

Здесь виджеты являются значениями полей объекта типа Block, функция-обработчик события нажатия на кнопку устанавливается не с помощью метода bind(), а с помощью свойства кнопки 'command'. В этом случае в вызываемой функции (в данном случае это метод) не требуется параметр event. В метод мы передаем только сам объект.

Однако, если код будет выглядеть так, то необходимости в классе нет. Смысл появится, если нам потребуется несколько или множество похожих объектов-блоков. Допустим, нам нужно несколько блоков, состоящих из метки, кнопки, поля. Причем у кнопки каждой группы будет своя функция-обработчик клика.

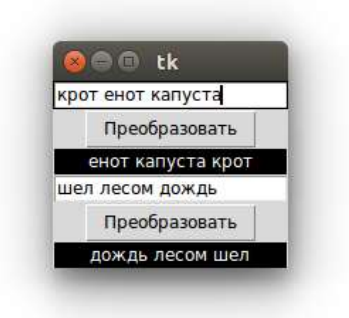
Тогда можно вынести установку значения для свойства command в отдельный метод, куда передавать привязываемую к кнопке функцию-обработчик события. Полный код программы:

```
from tkinter import *

class Block:
      def __init__(self, master):
            self.e = Entry(master, width=20)
            self.b = Button(master, text="Преобразовать")
            self.l = Label(master, bg='black', fg='white', width=20)
            self.e.pack()
            self.b.pack()
            self.l.pack()
      def setFunc(self, func):
            self.b['command'] = eval('self.' + func)
      def strToSortlist(self):
            s = self.e.get()
            s = s.split()
            s.sort()
            self.l['text'] = ' '.join(s)
```

```
    def strReverse(self):
        s = self.e.get()
        s = s.split()
        s.reverse()
        self.l['text'] = ' '.join(s)

root = Tk()

first_block = Block(root)
first_block.setFunc('strToSortlist')

second_block = Block(root)
second_block.setFunc('strReverse')

root.mainloop()
```

Функция eval() преобразует строку в исполняемый код. В результате получается `self.b['command'] = self.strToSortlist` или `self.b['command'] = self.strReverse`.

При выполнения этого кода в окне будут выведены два однотипных блока, кнопки которых выполняют разные действия.



Класс можно сделать более гибким, если жестко не задавать свойства виджетов, а передавать значения как аргументы в конструктор, после чего присваивать их соответствующим опциям при создании объектов.

Практическая работа

Напишите простейший калькулятор, состоящий из двух текстовых полей, куда пользователь вводит числа, и четырех кнопок "+", "-", "*", "/". Результат вычисления должен отображаться в метке. Если арифметическое действие выполнить невозможно (например, если были введены буквы, а не числа), то в метке должно появляться слово "ошибка".