Встановлювати властивості об'єктів не обов'язково при з створенні. Існують ще пара способів, за допомогою яких можна це зробити після.

Зверніть увагу, що першим аргументом вказується "господар, майстер" - місце, де розташовується віджет. В даному випадку його можна було не вказувати. Однак віджети не обов'язково розташовуються на root'е. Вони можуть розміщуватися в інших віджети.

Нехай в програмі текст, введений людиною в поле, при натисканні на кнопку розбивається на список слів, слова сортуються за алфавітом і виводяться в мітці. Код виконує все це треба помістити в функцію:

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

У функцій, які викликаються при настанні події за допомогою методу bind (), повинен бути один параметр. Зазвичай його називають event (подія).

У наведеній функції за допомогою методу get () з поля забирається текст, який представляє собою рядок. Вона перетворюється в список слів за допомогою методу split (). Потім список сортується. В кінці змінюється властивість text мітки. Йому присвоюється рядок, отримана зі списку за допомогою строкового методу join ().

Тепер необхідно пов'язати виклик функції з подією:

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

В даному випадку це робиться за допомогою методу bind (). Йому передається подія і викликається функція. Подія буде передано в функцію і присвоєно параметру event. В даному випадку подією є клацання лівою кнопкою миші, що позначається рядком `'<Button-1>'`.

У будь-якому додатку віджети не розкидані по вікну як попало, а добре організовані, інтерфейс продуманий до дрібниць і зазвичай підпорядкований певним стандартам. Поки розташуємо елементи один за одним за допомогою найбільш простого менеджера геометрії tkinter - методу pack ():

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

Метод mainloop () об'єкта Tk запускає головний цикл обробки подій, що в тому числі призводить до відображення головного вікна з усіма його причандаллям на екрані:

```
root. mainloop ( )
```

Повний код програми:

```
from tkinter import *
```

```
root = Tk ( )

e = Entry ( width = 20 )
b = Button ( text = "Перетворити" )
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 результаті виконання даного скрипта з'являється вікно, в текстове поле якого можна ввести список слів, натиснути кнопку і отримати його відсортоване варіант:

Спробуємо тепер реалізувати в нашій програмі об'єктно-орієнтований підхід. Це необов'язково, але нерідко буває доречним. Нехай комплект з мітки, кнопки і поля являє собою один об'єкт, який породжується від якогось класу, скажімо, Block. Тоді в основний гілці програми буде головне вікно, об'єкт типу Block і запуск вікна. Оскільки блок повинен бути прив'язаний до головного вікна, то непогано б передати в конструктор класу вікно-батько:

```
from tkinter import *

root = Tk ( )

first_block = Block ( root )

root. mainloop ( )
```

Тепер напишемо сам клас Block:

```
class Block:
    def  __init__ ( self , master ) :
        self . e  = Entry ( master , width = 20 )
        self . b  = Button ( master , text = "Перетворити" )
        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 )
```

Тут віджети є значеннями полів об'єкта типу Block, функція-обробник події натискання на кнопку встановлюється не за допомогою методу bind (), а за допомогою властивості кнопки 'command'. В цьому випадку в викликається функції (в даному випадку це метод) не потрібно параметр event. У метод ми передаємо тільки сам об'єкт.

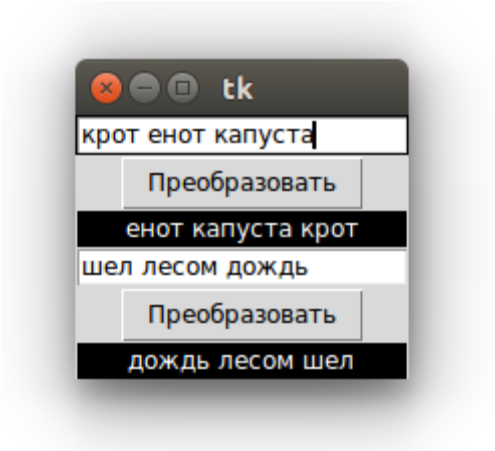
Однак, якщо код буде виглядати так, то необхідності в класі немає. Сенс з'явиться, якщо нам буде потрібно кілька або безліч схожих об'єктів-блоків. Припустимо, нам потрібно кілька блоків, що складаються з мітки, кнопки, поля. Причому у кнопки кожної групи буде своя функція-обробник кліка.

Тоді можна винести установку значення для властивості command в окремий метод, куди передавати прив'язують до кнопки функцію-обробник події. Повний код програми:

```
from tkinter import *

class Block:
     def  __init__ ( self , master ) :
          self . e  = Entry ( master , width = 20 )
          self . b  = Button ( master , text = "Перетворити" )
          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 ( )
```

Функція eval () перетворює рядок у виконуваний код. В результаті виходить `self.b['command'] = self.strToSortlist` або `self.b['command'] = self.strReverse`.

При виконання цього коду в вікні будуть виведені два однотипних блоку, кнопки яких виконують різні дії.



Клас можна зробити більш гнучким, якщо жорстко не ставити властивості віджетів, а передавати значення як аргументи в конструктор, після чого привласнювати їх відповідним опцій при створенні об'єктів.