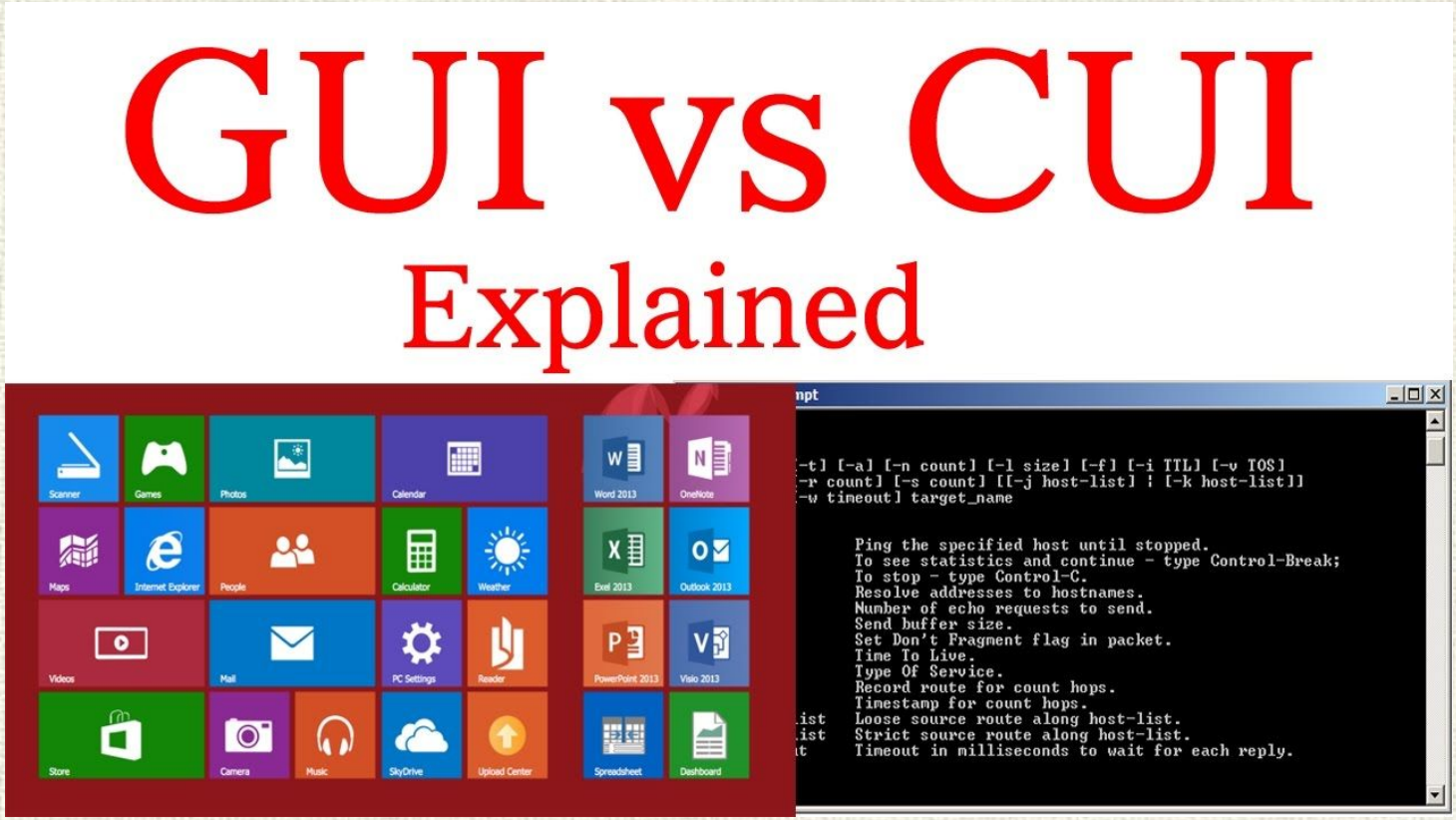
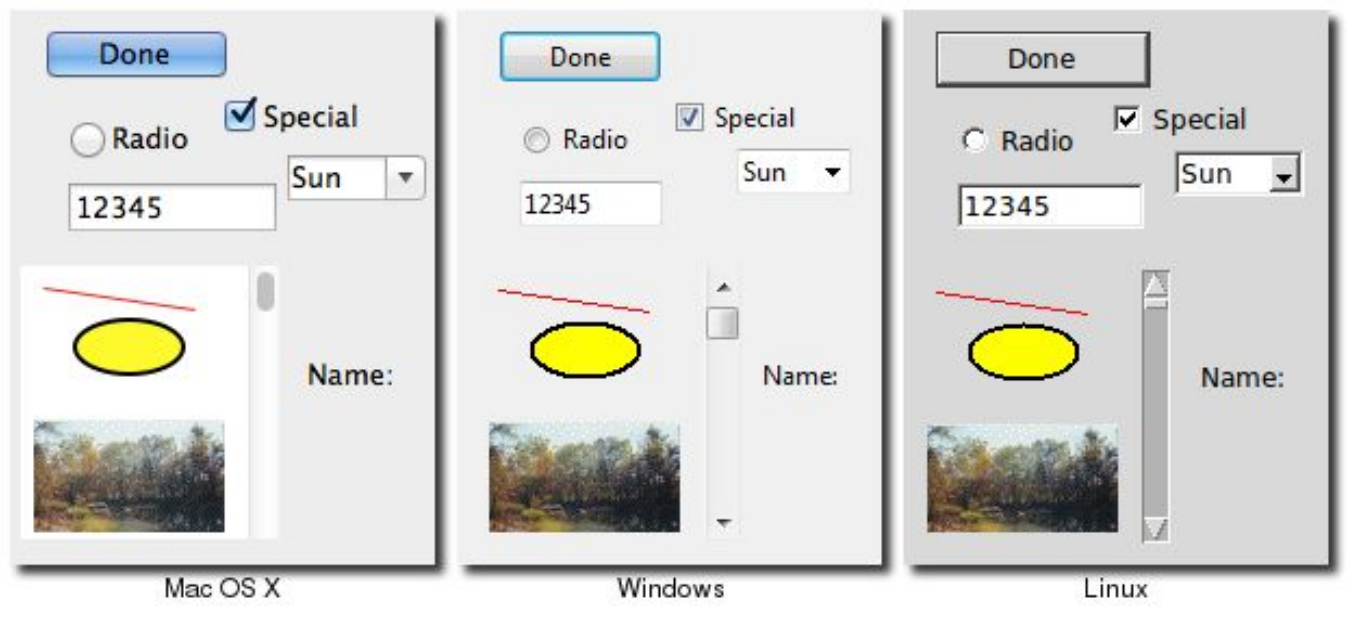
Графический интерфейс Библиотека Tkinter



с использованием иллюстраций и материалов, подготовленных Кондюриной А.А.

Что это?

Графический интерфейс пользователя (англ. graphical user interface, GUI) — разновидность пользовательского интерфейса, в котором его элементы (меню, кнопки, значки, списки и т. п.), выполнены в виде графических изображений.



Интерфейс - совокупность средств, методов и правил взаимодействия (управления, контроля и т.д.) между элементами системы.

Как работает?

Событийно-ориентированное программирование (event-driven programming) — это парадигма программирования, в которой выполнение программы определяется событиями - действиями пользователя (клавиатура, мышь), сообщениями других программ и потоков, событиями операционной системы (например, поступлением сетевого пакета).

Как работает?

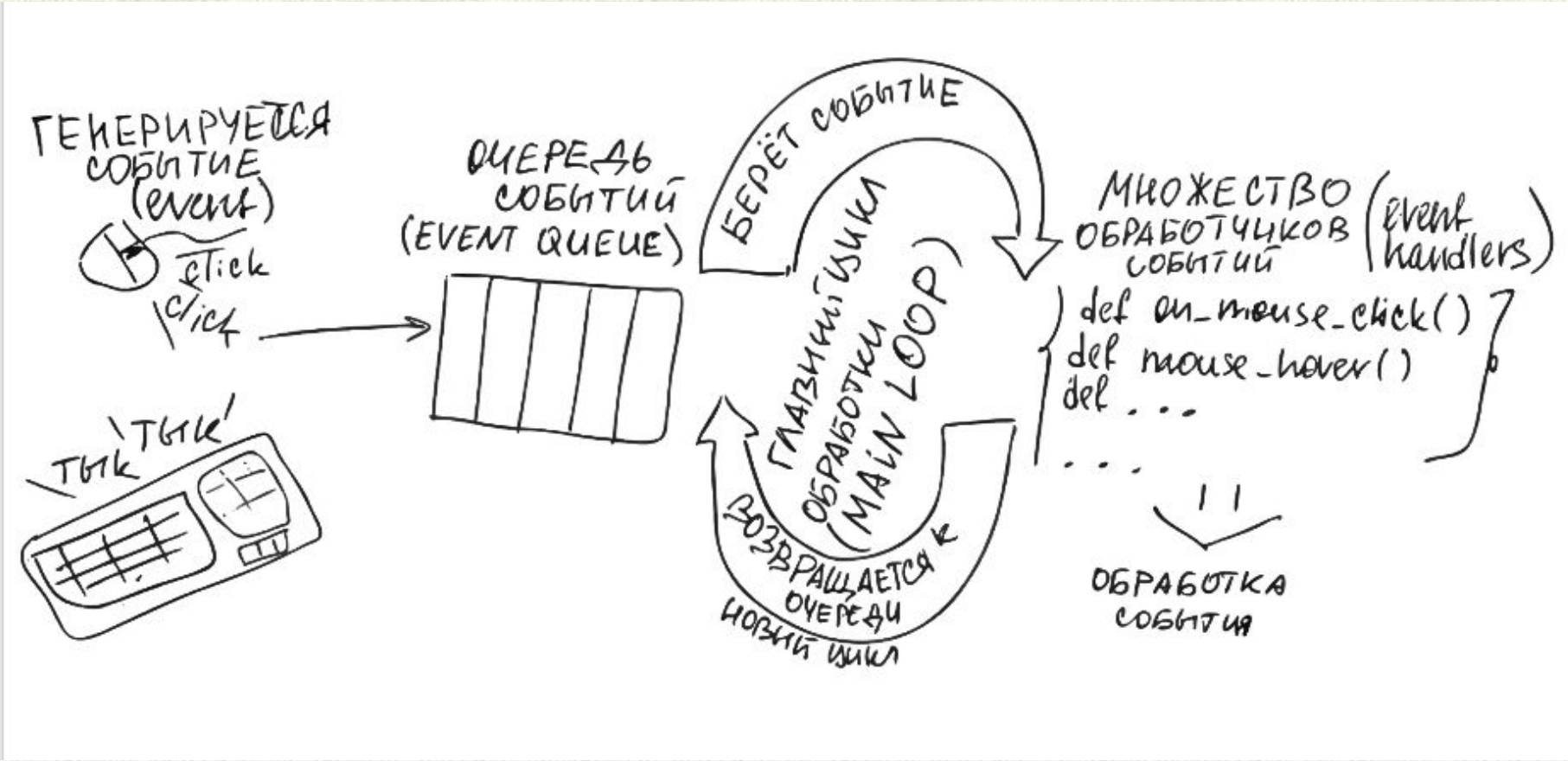
Примеры событий: ввод с клавиатуры, перемещение указателя или нажатие кнопки мыши и т.д.

Примеры элементов графического интерфейса: кнопка, поле ввода, полоса прокрутки, меню, чекбокс, выпадающий список и т.д.

Общая схема включает:

1. Главный цикл приложения (Main loop) - ожидает события (events).
2. Обработчик событий (Event handler) – реагирует на события от главного цикла и определяет, есть ли действие для события.
3. Функция обратного вызова (callback function) - вызывается обработчиком событий, выполняя заданные действия.

Как работает?



Графические библиотеки

Tkinter - стандартная кроссплатформенная библиотека для создания объектно-ориентированного интерфейса на языке Python (является интерфейсом к популярному языку программирования и инструменту создания графических приложений tcl/tk).

PyQT - набор Python библиотек для создания графического интерфейса на базе платформы Qt, считается одним из мощнейших инструментов для GUI.

wxPython - обертка над кроссплатформенной библиотекой виджетов wxWidgets.

PyGTK - набор Python-привязок для библиотеки графического интерфейса GTK+.

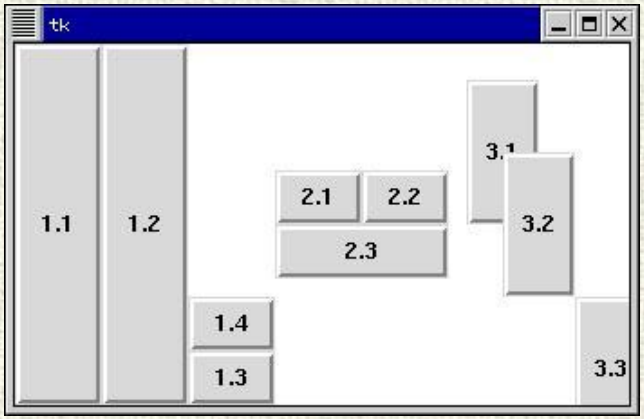
Какие существуют еще?

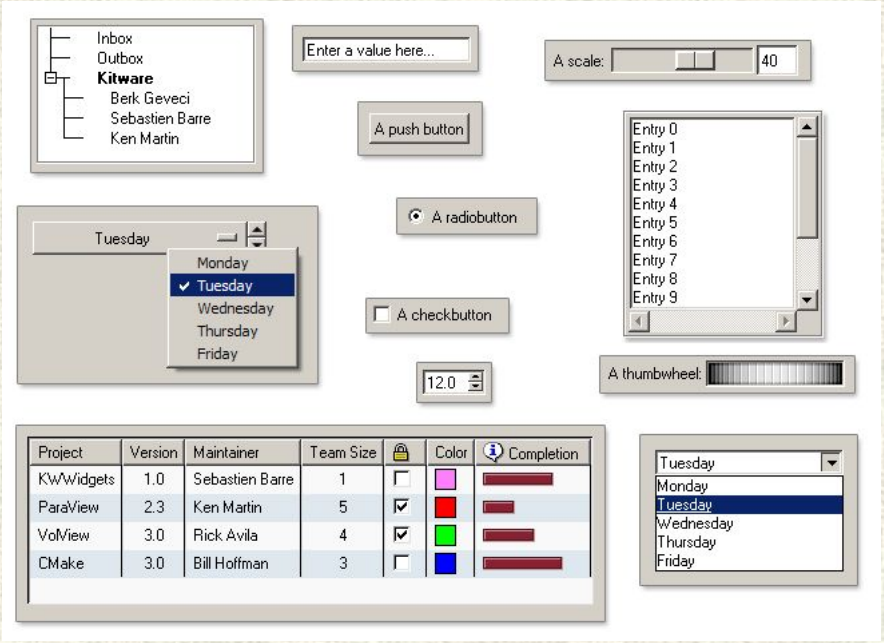
https://docs.python.org/3/faq/gui.html#general-gui-questions

Компоненты библиотек

Состав GUI-инструментов

1) Набор графических компонентов (виджетов – widget)
2) Средства расположения виджетов
3) Для каждого компонента – специфический набор событий





Виджет - примитив графического интерфейса с определенным набором свойств и действий, с которым взаимодействует пользователь (например, кнопки, поля для ввода, метки, флажки, переключатели , списки и т. д.).

Создание приложения

```
# подключение библиотеки
import tkinter
# любое gui-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 Тк модуля tkinter.
# Переменную связанную с объектом-окном принято называть root.
root = tkinter.Тк()
# Окно приложения не появится, пока не запустится главный цикл обработки событий.
root.mainloop()

# или так

from tkinter import *
root = Тк()
root.mainloop()
```

Создание приложения

- Главный цикл запускается методом mainloop().
- Для явного выхода из интерпретатора и завершения цикла обработки событий используется метод quit().
- После вызова метода mainloop дальнейшие команды python исполняться не будут до выхода из цикла обработки событий.

```
root.mainloop()
print('Пока mainloop работает, я ожидаю.')
```

Описание виджетов происходит между созданием родительского окна и запуском главного цикла.

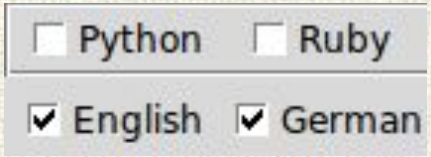
```
Имя_виджета=тип_виджета(родитель, параметры)
Имя_виджета.способ_размещения()
```

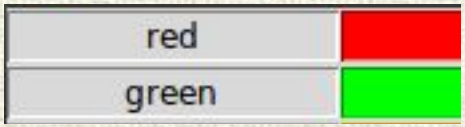
Например:

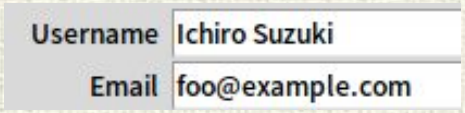
```
button1=Button(root, text="Push me")
button1.pack()
```

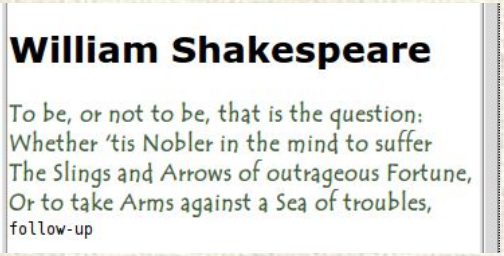
Типы виджетов

- Button (Кнопка). Простая кнопка для вызова некоторых действий (выполнения определенной команды).
- Checkbutton (Флажок). Кнопка, которая умеет переключаться между двумя состояниями при нажатии на нее.
- Label (Надпись). Может показывать текст или графическое изображение.
- Entry (Однострочное текстовое поле ввода). Горизонтальное поле, в которое можно ввести строку текста.
- Text (Многострочное текстовое поле). Позволяет редактировать и форматировать текст с использованием различных стилей, внедрять в текст рисунки и даже окна.



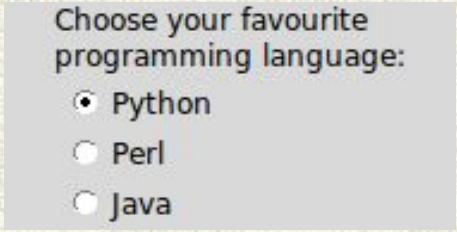


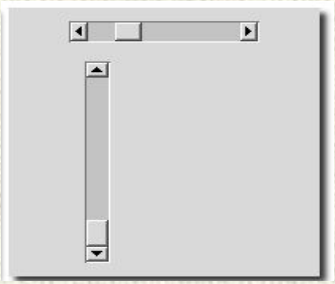


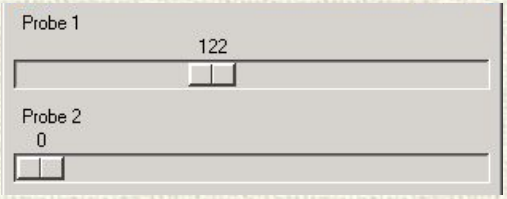


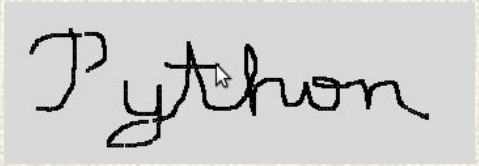
Типы виджетов

- Radiobutton (Селекторная кнопка). Кнопка для представления одного из альтернативных значений.
- Scrollbar (Полоса прокрутки). Служит для отображения величины прокрутки (вертикальной, горизонтальной) в других виджетах.
- Scale (Шкала). Служит для задания числового значения путем перемещения «бегунка» в определенном диапазоне.
- Listbox (Список). Прямоугольная рамка со списком, из которого пользователь может выделить один или несколько элементов.
- Canvas (Рисунок). Основа для вывода графических примитивов.



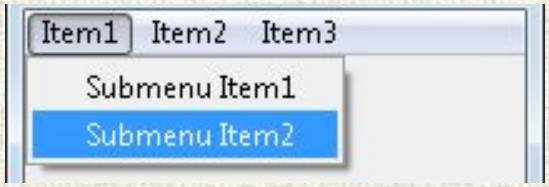


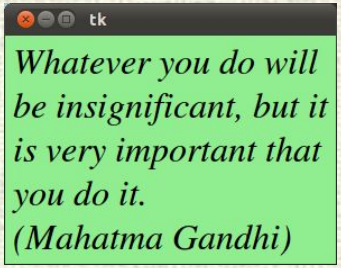


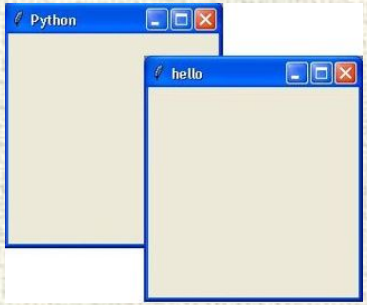


Типы виджетов

- Frame (Рамка). Виджет, который содержит в себе другие визуальные компоненты.
- Menu (Меню). Элемент, с помощью которого можно создавать всплывающие (popup) и ниспадающие (pulldown) меню.
- Menubutton (Кнопка-меню). Кнопка с ниспадающим меню.
- Message (Сообщение). Аналогично надписи, но позволяет переносить длинные строки и менять размер по требованию менеджера расположения.
- Toplevel (Окно верхнего уровня). Показывается как отдельное окно и содержит внутри другие виджеты.

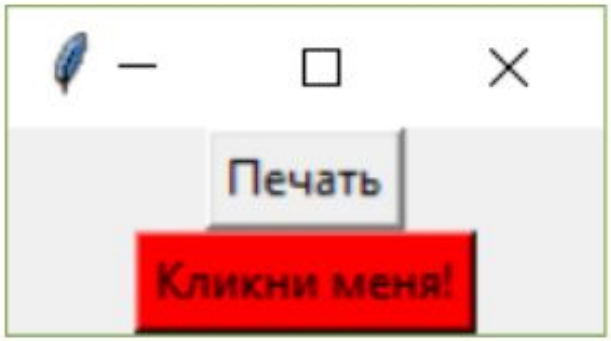






Пример создания виджета

```
# пример использования виджета Кнопка
from tkinter import *

def button_clicked():
    print("Клик!")

root = Tk()
button1 = Button(root)  # кнопка по умолчанию, привязка к окну root
button1["text"] = "Печать"  # определяем свойство text у кнопки
button1.pack()  # упаковываем (добавляем кнопку)
# кнопка с указанием родительского виджета и несколькими аргументами
button2 = Button(root, bg="red",
                 text="Кликни меня!",
                 command=button_clicked)
button2.pack()  # упаковываем (добавляем кнопку)
root.mainloop()
```

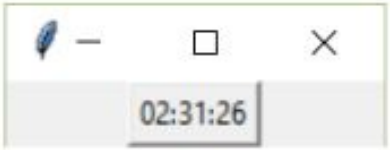


Свойство command вызывает функцию-обработчик виджета.
Метод pack() размещает виджет в окне.

Работа с виджетами

Метод configure() позволяет конфигурировать (изменять настройки) параметры виджетов.
Также можно использовать квадратные скобки (widget['option'] = new_value).

```
#  программа выводит текущее время, после клика по кнопке:
def button_clicked():
    # изменяем текст кнопки
    button['text'] = time.strftime('%H:%M:%S')

root = Tk()
button = Button(root) # создаём виджет
# конфигурируем виджет после создания
button.configure(text=time.strftime('%H:%M:%S'), command=button_clicked)
# также можно использовать квадратные скобки:
# button['text'] = time.strftime('%H:%M:%S')
# button['command'] = button_clicked
button.pack()
root.mainloop()
```



Работа с виджетами

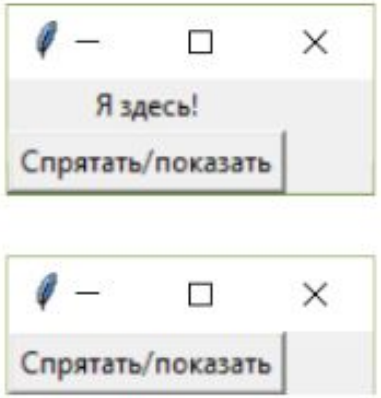
Метод cget() - предназначен для получения информации о конфигурации виджета. Здесь как и в случае с configure можно использовать квадратные скобки (value = widget['option']).

```
#  после клика на кнопку программа показывает цвет кнопки и меняет его
from random import randint
def button_clicked():
    button['text'] = button['bg'] # показываем предыдущий цвет кнопки
    bg = "#%06x" % randint(0, 0xFFFFFF)
    button['bg'] = bg
    button['activebackground'] = bg
root=Tk()
button = Button(root, command=button_clicked)
button.pack()
root.mainloop()
```

Работа с виджетами

Метод destroy() позволяет удалить виджет.
Но, если необходимо только скрыть виджет, то лучше пользоваться упаковщиком grid() и методом grid_remove(). Использование grid_remove() позволяет сохранять взаимное расположение виджетов.

```
def hide_show():
    if label.winfo_viewable():
        label.grid_remove()
    else:
        label.grid()
root=Tk()
label = Label(text=u'Я здесь!')
label.grid()
button = Button(command=hide_show, text=u"Спрятать/показать")
button.grid()
root.mainloop()
```



Работа с виджетами

Менеджер расположения (упаковщик, менеджер геометрии) - это специальный механизм, который размещает (упаковывает) виджеты в окне.

Виды:

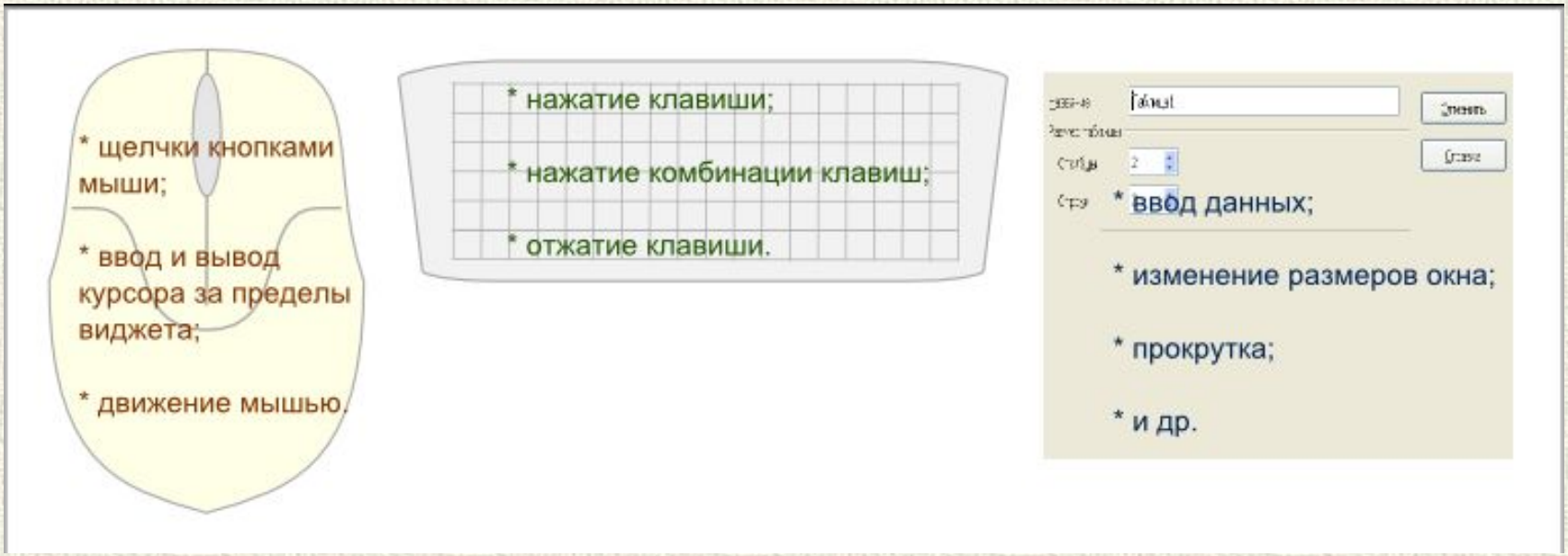
- pack(),
- place(),
- grid().

В одном виджете можно использовать только один тип упаковки, при смешивании разных типов упаковки программа, скорее всего, не будет работать.

События

Событие - внешнее воздействие на графический компонент.

Типы событий



События

Для привязки событий к виджетам используются:

- Свойство command (для Button, Checkbutton, Radiobutton, Spinbox, Scrollbar, Scale):

```
button1 = Button(command=callback)
```

- Метод bind() → widget.bind(modifier, callback):

```
button.bind("<Control-Shift-KeyPress-q>", callback)
#callback - имя функции обработки события (принимает аргумент event)
```

Для описания событий используются модификаторы: Control, Shift, Lock, Button1-Button5 (или B1-B5), Meta, Alt, Double, Triple и др.

Пример

```
from tkinter import *

# если был щелчок ЛКМ -> заголовок такой
def handle_button_1_click(event):
    root.title("Левая кнопка мыши")

# если был щелчок ПКМ -> заголовок такой
def handle_button_3_click(event):
    root.title("Правая кнопка мыши")

# если перемещение курсора -> заголовок такой
def handle_motion(event):
    root.title("Движение мышью")

root = Tk()
root.minsize(width=500, height=400)
# привязка событий
root.bind('<Button-1>', handle_button_1_click)
root.bind('<Button-3>', handle_button_3_click)
root.bind('<Motion>', handle_motion)
root.mainloop()
```

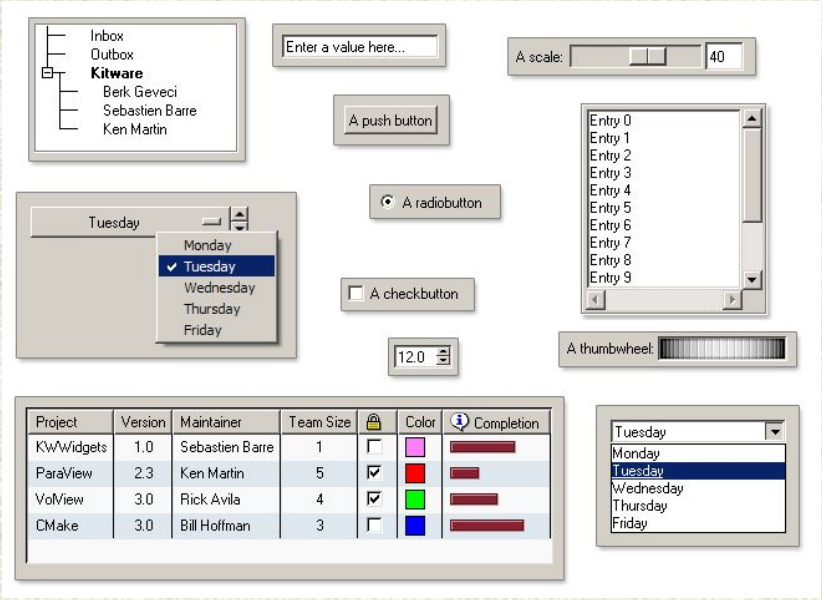
Виджеты

Что это?

Виджет - примитив графического интерфейса с определенным набором свойств и действий, с которым взаимодействует пользователь (например, кнопки, поля для ввода, метки, флажки, переключатели , списки и т. д.).

Группы виджетов:

- ✔выбора (действия): Radiobutton, Menu, Checkbutton, Menubutton, Button.
- ✔отображения текста: Label, Message.
- ✔ввода текста: Entry, Text, Listbox.
- ✔управления: Scale, Scrollbar.
- ✔графики: Canvas.
- ✔контейнеры: Frame.
- ✔И др…



Параметры виджетов

Изменить конфигурацию виджета можно, используя метод configure(). Также можно использовать квадратные скобки (widget['option'] = new_value).

Получить значение атрибута можно методом cget().

Общие параметры:

background - фон

activebackground – фон при наведении курсора

cursor - тип курсора при наведении

image – вставить изображение

font - настроить шрифт текста

takefocus - определяет, получает ли элемент фокус

Подробный список

Button

bg / background - основной цвет кнопки.
fg / foreground - основной цвет текста.
text - текст кнопки.
font - настройка шрифта (подробнее...)
activebackground - цвет кнопки при наведении курсора.
activeforeground - цвет текста при наведении курсора.
image - изображение, которое будет изображено вместо текста.
height - высота кнопки в строках (текст) или пикселях (картинка).
width - ширина в символах (текст) или пикселях (картинка).

Другие свойства

Пример

```
from tkinter import *

root = Tk()

button = Button(root,
                text="Это кнопка",  # надпись на кнопке
                width=30,  # ширина
                height=5,  # высота
                bg="white",  # цвет фона и надписи
                fg="blue",
                font="Verdana 12",  # шрифт текста
                activebackground="black",  # цвет фона и надписи
                activeforeground="red")  # при наведении курсора

button.pack()
root.mainloop()
```

Radiobutton

Представляет одно из альтернативных значений некоторой переменной. Обычно действует в группе.

Когда пользователь выбирает какую-либо опцию, с ранее выбранного в этой же группе элемента выбор снимается.

```
radiobutton_value = IntVar()

radiobutton_0 = Radiobutton(root, text="Первая",
                            variable=radiobutton_value, value=0)
radiobutton_1 = Radiobutton(root, text="Вторая",
                            variable=radiobutton_value, value=1)
radiobutton_2 = Radiobutton(root, text="Третья",
                            variable=radiobutton_value, value=2)

radiobutton_0.pack()
radiobutton_1.pack()
radiobutton_2.pack()
```

Entry

Поле ввода

bg / background - цвет фона (по умолчанию “light gray”).

bd / borderwidth - ширина рамки (по умолчанию 2 пикселя).

cursor - тип курсора при наведении (подробнее...)

justify - выравнивание текста.

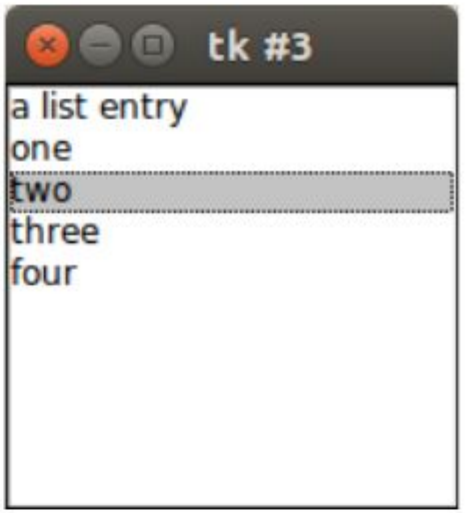
show - символы, которые будут видны вместо вводимого текста (show=”*”), например, если поле ввода типа password.

```
entry = Entry(root,
              width=20,
              bd=3,
              cursor="star",
              justify="right",
              show="$"
              )

entry.pack()
```

Listbox

список, из которого пользователь может выделить один или несколько элементов

```
listbox = Listbox(root)
listbox.pack()

listbox.insert(END, "a list entry")

for item in ["one", "two", "three", "four"]:
    listbox.insert(END, item)
```

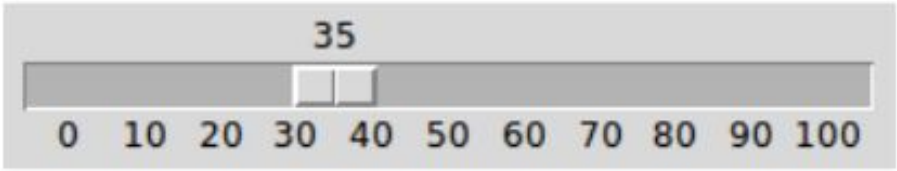


Scale

Шкала. Позволяет задать числовое значение путем перемещения «бегунка».

Представляет собой горизонтальную или вертикальную полосу с разметкой, по которой пользователь может передвигать «бегунок», осуществляя тем самым выбор значения.

```
scale = Scale(root, orient=HORIZONTAL, length=300,
              from_=0, to=100, tickinterval=10, resolution=5)
scale.pack()
```



Scrollbar

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()

text = Text(root, width=40, height=30, font='14')
scroll = Scrollbar(root, command=text.yview)
text.configure(yscrollcommand=scroll.set)

text.grid(row=0, column=0)
scroll.grid(row=0, column=1)
root.mainloop()
```

В примере сначала создается текстовое поле (text), затем полоса прокрутки (scroll), которая привязывается с помощью опции command к полю text по вертикальной оси (yview). Далее поле text изменяется с помощью метода configure: устанавливается значение опции yscrollcommand.

Менеджеры расположения

Менеджеры расположения

Менеджер расположения (упаковщик, менеджер геометрии) - это специальный механизм, который размещает (упаковывает) виджеты в окне.

Виды:

- pack(),
- place(),
- grid().

В одном виджете можно использовать только один тип упаковки, при смешивании разных типов упаковки программа, скорее всего, не будет работать.

Pack

Выполняет простейшее расположение без детальной привязки к месту в окне.

Когда использовать:

- Поместить несколько виджетов в ряд.
- Поместить несколько виджетов друг на друга.
- Поместить виджет внутри фрейма (или любого другого виджета-контейнера) и заполнить им весь фрейм.

Для создания более сложных размещений можно использовать несколько Frame виджетов или перейти к grid() менеджеру.

Pack

Параметры

- side ("left" / "right" / "top" / "bottom") – сторона размещения.
- fill (None / "x" / "y" / "both") – расширение поля виджета.
- expand (True / False) – расширение виджета на всё предоставляемое пространство.
- in_ - явное указание родительского виджета.

Методы

- pack_slaves() - возвращает список всех дочерних виджетов.
- pack_info() - возвращает информацию о конфигурации упаковки.
- pack_propagate() (True/False) - включает/отключает распространении информации о геометрии дочерних виджетов. По умолчанию виджет изменяет свой размер в соответствии с размером потомков. Используется когда необходимо, чтобы виджет имел фиксированный размер и не изменял его по прихоти потомков.
- pack_forget() - удаляет виджет и всю информацию о его расположении из упаковщика. Позднее этот виджет может быть снова размещён.

Pack

```
w = Label(root, text="Red", bg="red", fg="white")
w.pack()
w = Label(root, text="Green", bg="green", fg="black")
w.pack()
w = Label(root, text="Blue", bg="blue", fg="white")
w.pack()
```


```
# ------------------------------------------------------------------------
```

```
w = Label(root, text="Red", bg="red", fg="white")
w.pack(fill=X)
w = Label(root, text="Green", bg="green", fg="black")
w.pack(fill=X)
w = Label(root, text="Blue", bg="blue", fg="white")
w.pack(fill=X)
```


```
# ------------------------------------------------------------------------
```

```
w = Label(root, text="Red", bg="red", fg="white")
w.pack(side=LEFT)
w = Label(root, text="Green", bg="green", fg="black")
w.pack(side=LEFT)
w = Label(root, text="Blue", bg="blue", fg="white")
w.pack(side=LEFT)
```

Pack

```
print(label.pack_info())
```

```
>>> {'ipady': 0, 'padx': 0, 'ipadx': 0, 'fill': 'none',
'in': <tkinter.Tk object .>, 'side': 'left', 'expand': 0,
'pady': 0, 'anchor': 'center'}
```

```
print(root.pack_slaves())
```

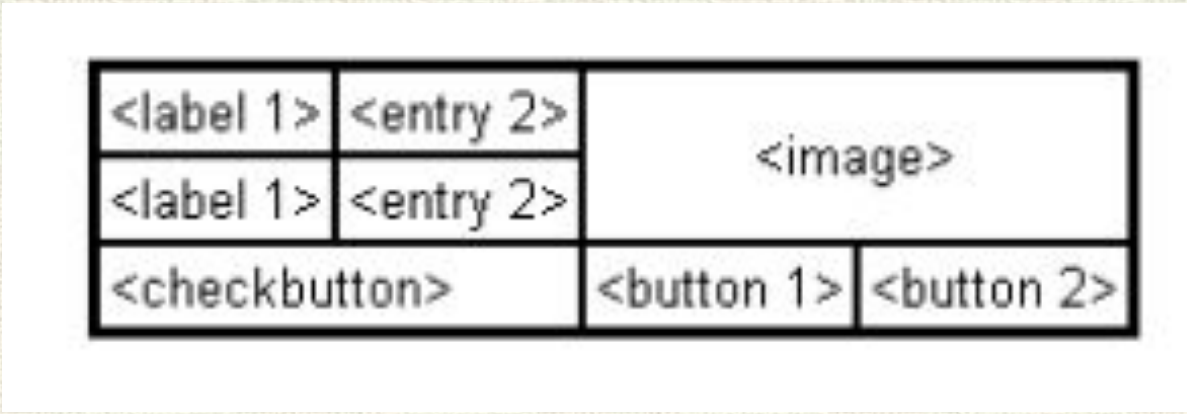
```
>>> [<tkinter.Label object .140511859039592>, <tkinter.Label
object .140511829671048>, <tkinter.Label object
.140511829671496>]
```

Grid

Представляет собой таблицу с ячейками, в которые помещаются виджеты.

Для каждого виджета указывается в какой строке (row) и в каком столбце (column) он находится.

Удобен в случае сложного расположения виджетов.



Grid

Параметры

- row - номер строки.
- rowspan - сколько строк занимает виджет
- column - номер столбца.
- columnspan - сколько столбцов занимает виджет.
- padx / pady - размер внешней границы по горизонтали и вертикали.
- ipadx / ipady - размер внутренней границы по горизонтали и вертикали.

Разница между pad и ipad в том, что при указании pad расширяется свободное пространство, а при ipad расширяется помещаемый виджет.

- sticky ("n", "s", "e", "w" или их комбинация) - указывает к какой границе "приклеивать" виджет. Позволяет расширять виджет в указанном направлении. Границы названы в соответствии со сторонами света. "n" (север) - верхняя граница, "s" (юг) - нижняя, "w" (запад) - левая, "e" (восток) - правая.
- in_ - явное указание родительского виджета.

Grid

```
label_1 = Label(root, text="First")
label_2 = Label(root, text="Second")
entry_1 = Entry(root)
entry_2 = Entry(root)
label_1.grid(row=0)
label_2.grid(row=1)
entry_1.grid(row=0, column=1)
entry_2.grid(row=1, column=1)

# ------------------------------------------------------------------------------

# using stiky

label_1 = Label(root, text="First")
label_2 = Label(root, text="Second")
entry_1 = Entry(root)
entry_2 = Entry(root)
label_1.grid(row=0, sticky=W)
label_2.grid(row=1, sticky=W)
entry_1.grid(row=0, column=1)
entry_2.grid(row=1, column=1)
```

Grid

```
# using rowspan and columnspan

label_1 = Label(root, text="First")
label_2 = Label(root, text="Second")
entry_1 = Entry(root)
entry_2 = Entry(root)
checkbutton = Checkbutton(root, text="Check!")
button_1 = Button(root, text="First")
button_2 = Button(root, text="Second")

label_1.grid(sticky=E)
label_2.grid(sticky=E)
entry_1.grid(row=0, column=1)
entry_2.grid(row=1, column=1)
checkbutton.grid(columnspan=2, sticky=W)
button_1.grid(row=2, column=2)
button_2.grid(row=2, column=3)
```

Grid

Методы

- grid_slaves() - возвращает список всех дочерних виджетов.
- grid_info() - возвращает информацию о конфигурации упаковки.
- grid_propagate() (True/False) - включает/отключает распространении информации о геометрии дочерних виджетов.
- grid_remove() - удаляет виджет из-под управления упаковщиком, но сохраняет информацию об упаковке.
- grid_bbox() - возвращает координаты (в пикселях) указанных столбцов и строк.
- grid_location(x, y) - принимает два аргумента: x и y (в пикселях). Возвращает номер строки и столбца в которые попадают указанные координаты, либо -1 если координаты попали вне виджета.
- grid_size()- возвращает размер таблицы в строках и столбцах.

Grid

grid_columnconfigure() и grid_rowconfigure() - функции конфигурирования параметров сетки. Принимают номер строки/столбца и аргументы конфигурации.

Список возможных аргументов:

- minsize - минимальная ширина/высота строки/столбца.
- weight - "вес" строки/столбца при увеличении размера виджета. 0 означает, что строка/столбец не будет расширяться.
- uniform - объединение строк/столбцов в группы. Строки/столбцы имеющие одинаковый параметр uniform будут расширяться строго в соответствии со своим весом.
- pad - размер рамки. Указывает, сколько пространства будет добавлено к самому большому виджету в строке/столбце.

Place

Используется для сложной геометрии размещения виджетов. Позволяет располагать виджет в фиксированном месте с фиксированным размером. Необходимо указывать координаты каждого виджета.

Возможно указывать координаты размещения в относительных единицах для реализации "резинового макета".

Относительные и абсолютные координаты (а также ширину и высоту) можно комбинировать.

Так например, relx=0.5, x=-2 означает размещение виджета в двух пикселях слева от центра родительского виджета, relheight=1.0, height=-2 - высота виджета на два пикселя меньше высоты родительского виджета.

Place

Параметры

- anchor ("n", "s", "e", "w", "ne", "nw", "se", "sw" или "center") - какой угол или сторона размещаемого виджета будет указана в аргументах x/y/relx/rely. По умолчанию "nw" - левый верхний
- bordermode ("inside", "outside", "ignore") – параметры рамок.
- in_ - явное указание родительского виджета.
- x и y - абсолютные координаты (в пикселях) размещения виджета.
- width и height - абсолютные ширина и высота виджета.
- relx и rely - относительные координаты (от 0.0 до 1.0) размещения виджета.
- relwidth и relheight - относительные ширина и высота виджета.

Place

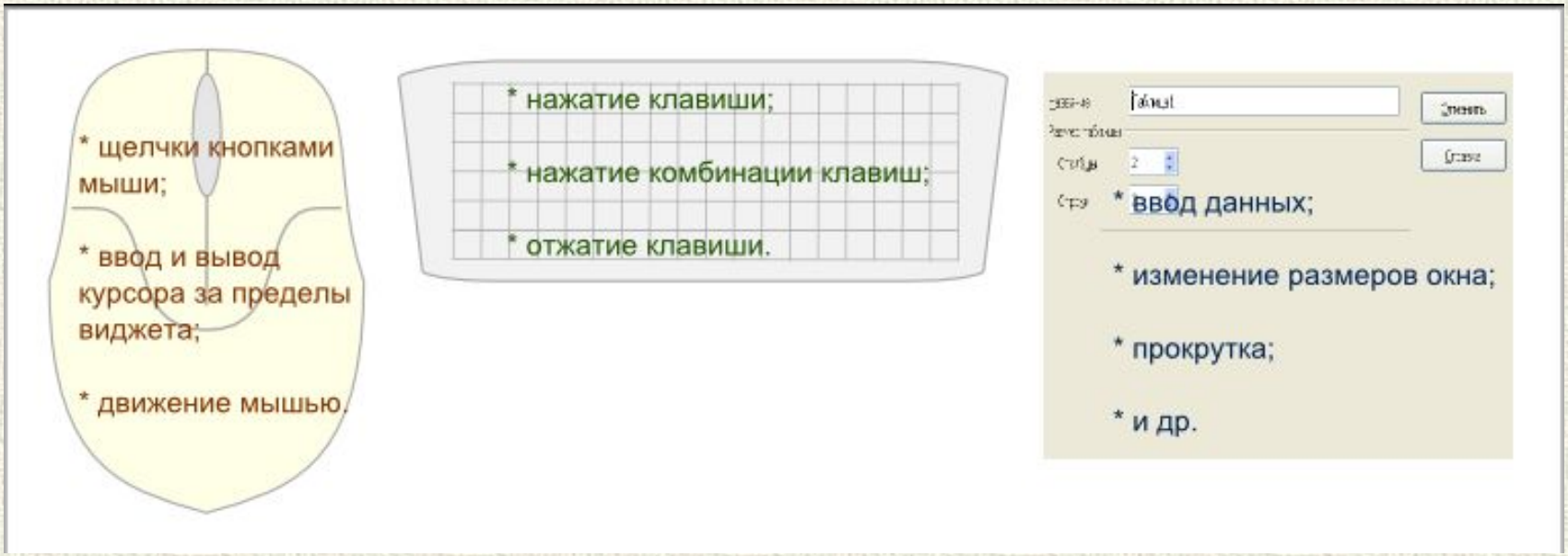
```
root = Tk()
# width x height + x offset + y_offset:
root.geometry("170x200+30+30")

languages = ['Python','Perl','C++','Java','Tcl/Tk']

labels = range(5)

for i in range(5):
  ct = [random.randrange(256) for x in range(3)]
  brightness = int(round(0.299*ct[0] + 0.587*ct[1] + 0.114*ct[2]))
  ct_hex = "%02x%02x%02x" % tuple(ct)
  bg_colour = '#' + "".join(ct_hex)
  l = Label(root,
            text=languages[i],
            fg='White' if brightness < 120 else 'Black',
            bg=bg_colour)
  l.place(x = 20, y = 30 + i*30, width=120, height=25)

root.mainloop()
```

Обработка событий

События

Событие - внешнее воздействие на графический компонент.

Типы событий



События

Для привязки событий к виджетам используются:

- Свойство command (для Button, Checkbutton, Radiobutton, Spinbox, Scrollbar, Scale):

```
button1 = Button(command=callback)
```

- Метод bind() → widget.bind(modifier, callback):

```
button.bind("<Control-Shift-KeyPress-q>", callback)
#callback - имя функции обработки события (принимает аргумент event)
```

При привязке методом bind обязательным для callback функции является аргумент event, содержащий параметры события, активировавшего функцию

Модификаторы

Модификатор – строка-описание события:
Control, Shift, Lock, Button1-Button5 (или B1-B5), Meta, Alt, Double, Triple и др.

Модификатор нажатия клавиши, например, ввод символа "k" - это событие "<KeyPress-k>".

Модификаторы спецклавиш: Cancel, BackSpace, Tab, Return, Shift_L, Control_L, Alt_L, Pause, Caps_Lock, Escape, End, Home, Left, Up, Right, Down, Print, Delete, Insert, F1-F12, Num_Lock, space

События

Тип события	Содержание события
Activate	Активизация окна
ButtonPress	Нажатие кнопки мыши
ButtonRelease	Отжатие кнопки мыши
Deactivate	Деактивация окна
Destroy	Закрытие окна
Enter	Вхождение курсора в пределы виджета
FocusIn	Получение фокуса окном
FocusOut	Потеря фокуса окном
KeyPress	Нажатие клавиши на клавиатуре
KeyRelease	Отжатие клавиши на клавиатуре
Leave	Выход курсора за пределы виджета
Motion	Движение мыши в пределах виджета
MouseWheel	Прокрутка колесика мыши
Reparent	Изменение родителя окна
Visibility	Изменение видимости окна

События

```
from tkinter import *

# если был щелчок ЛКМ -> заголовок такой
def handle_button_1_click(event):
    root.title("Левая кнопка мыши")

# если был щелчок ПКМ -> заголовок такой
def handle_button_3_click(event):
    root.title("Правая кнопка мыши")

# если перемещение курсора -> заголовок такой
def handle_motion(event):
    root.title("Движение мышью")

root = Tk()
root.minsize(width=500, height=400)
# привязка событий
root.bind('<Button-1>', handle_button_1_click)
root.bind('<Button-3>', handle_button_3_click)
root.bind('<Motion>', handle_motion)
root.mainloop()
```

События

```
from tkinter import *

# если нажаты ctrl+z -> выйти
def handle_exit(event):
    root.destroy()

# если нажат Enter -> скопировать введенный текст в лейбл
def handle_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()
# привязка событий
ent.bind('<Return>', handle_caption)
root.bind('<Control-z>', handle_exit)
root.mainloop()
```

Описание события. Event

Аргумент Event предназначен для описания наступившего события.

Атрибуты Event:

- serial - серийный номер события (все события)
- num - номер кнопки мыши (ButtonPress, ButtonRelease)
- focus - имеет ли окно фокус (Enter, Leave)
- height и width - ширина и высота окна (Configure, Expose)
- keycode - код нажатой клавиши (KeyPress, KeyRelease)
- state - состояние события (для ButtonPress, ButtonRelease, Enter, KeyPress, KeyRelease, Leave, Motion - в виде числа; для Visibility - в виде строки)
- time - время наступления события (все события)
- x и y - координаты мыши

Описание события. Event

Атрибуты Event:

- x_root и y_root - координаты мыши на экране (ButtonPress, ButtonRelease, KeyPress, KeyRelease, Motion)
- keysym - набранный на клавиатуре символ (KeyPress, KeyRelease)
- keysym_num - набранный на клавиатуре символ в виде числа (KeyPress, KeyRelease)
- type - тип события в виде числа (все события)
- widget - виджет, который получил событие (все события)
- delta - изменение при вращении колеса мыши (MouseWheel)

Описание события. Event

```
from Tkinter import *
# функция обработки события
def event_info(event):
    txt.delete("1.0", END) # удаляется с начала до конца текста
    for k in dir(event): # цикл по атрибутам события
        if k[0] != "_": # берутся только неслужебные атрибуты
      ev = "%15s: %s\n" % (k, repr(getattr(event, k)))
      txt.insert(END, ev) # добавляется в конец текста

root = Tk()
txt = Text(root) # текстовый виджет
txt.pack()
txt.bind("<KeyPress>", event_info)
root.mainloop()
```

Привязка событий

Дополнительные методы

- bind_all() - создаёт привязку для всех виджетов приложения. Отличие от привязки к окну верхнего уровня заключается в том, что в случае привязки к окну привязываются все виджеты этого окна, а этот метод привязывает все виджеты приложения (у приложения может быть несколько окон).
- bind_class() - создаёт привязку для всех виджетов данного класса. Например, привязка валидации ввода validate() для всех Entry полей.
- unbind() - отвязать виджет от события. В качестве аргумента принимает идентификатор, полученный от метода bind.
- unbind_all() - то же, что и unbind, только для метода bind_all.
- unbind_class() - то же, что и unbind, только для метода bind_class.