

Python. Лекция 12.

Создание приложений с GUI.

Обзор графических библиотек

Строить **графический интерфейс пользователя** (GUI, **G**raphical **U**ser **I**nterface) для программ на языке Python можно при помощи соответствующих **библиотек компонентов графического интерфейса** или, используя кальку с английского, **библиотек виджетов**.

Следующий список далеко не полон, но отражает многообразие существующих решений:

1. `Tkinter` Многоплатформенный пакет имеет хорошее управление расположением компонентов. Интерфейс выглядит одинаково на различных платформах (Unix, Windows, Macintosh). Входит в стандартную поставку Python. В качестве документации можно использовать руководство "An Introduction to Tkinter" ("Введение в Tkinter"), написанное Фредриком Лундом: <http://www.pythonware.com/library/tkinter/introduction/>
2. `wxPython` Построен на многоплатформенной библиотеке `wxWidgets` (раньше называлась `wxWindows`). Выглядит родным для всех платформ, активно совершенствуется, осуществлена поддержка GL. Имеется для всех основных платформ. Возможно, займет место Tkinter в будущих версиях Python. Сайт: <http://www.wxpython.org/>
3. `PyGTK` Набор визуальных компонентов для GTK+ и Gnome. Только для платформы GTK.
4. `PyQT/PyKDE` Хорошие пакеты для тех, кто использует Qt (под UNIX или Windows) или KDE.
5. `Pythonwin` Построен вокруг MFC, поставляется вместе с оболочкой в пакете `win32all`; только для Windows.
6. `pyFLTK` Аналог Xforms, поддержка OpenGL. Имеется для платформ Windows и Unix. Сайт: <http://pyfltk.sourceforge.net/>
7. `AWT`, `JFC`, `Swing` Поставляется вместе с Jython, а для Jython доступны средства, которые использует Java. Поддерживает платформу Java.
8. `anygui` Независимый от нижележащей платформы пакет для построения графического интерфейса для программ на Python. Сайт: <http://anygui.sourceforge.net/>
9. `PythonCard` Построитель графического интерфейса, сходный по идеологии с HyperCard/MetaCard. Разработан на базе `wxPython`. Сайт: <http://pythoncard.sourceforge.net/>

Список актуальных ссылок на различные графические библиотеки, доступные из Python, можно найти по следующему адресу: http://phaseit.net/claird/comp.lang.python/python_GUI.html

Библиотеки могут быть многоуровневыми. Например, PythonCard использует wxPython, который, скажем, на платформе Linux базируется на многоплатформенной GUI-библиотеке wxWindows, которая, в свою очередь, базируется на GTK+ или на Motif, а те - тоже используют для вывода X Window. Кстати, для Motif в Python имеются свои привязки.

В лекции будет рассматриваться пакет Tkinter, который по сути является оберткой для Tcl/Tk - известного графического пакета для сценарного языка Tcl. На примере этого пакета легко изучить основные принципы построения графического интерфейса пользователя.

О графическом интерфейсе

Почти все современные графические интерфейсы общего назначения строятся по модели WIMP - Window, Icon, Menu, Pointer (окно, иконка, меню, указатель). Внутри окон рисуются **элементы графического интерфейса**, которые для краткости будут называться **виджетами** (widget - штука). Меню могут располагаться в различных частях окна, но их поведение достаточно однотипно: они служат для выбора действия из набора предопределенных действий. Пользователь графического интерфейса "объясняет" компьютерной программе требуемые действия с помощью указателя. Обычно указателем служит курсор мыши или джойстика, однако есть и другие "указательные" устройства. С помощью иконок графический интерфейс приобретает независимость от языка и в некоторых случаях позволяет быстрее ориентироваться в интерфейсе.

Основной задачей графического интерфейса является упрощение коммуникации между пользователем и компьютером. Об этом следует постоянно помнить при проектировании интерфейса. Применение имеющихся в наличии у программиста (или дизайнера) средств при создании графического интерфейса нужно свести до минимума, выбирая наиболее удобные пользователю виджеты в каждом конкретном случае. Кроме того, полезно следовать принципу наименьшего удивления: из формы интерфейса должно быть понятно его поведение. Плохо продуманный интерфейс портит ощущения пользователя от программы, даже если за фасадом интерфейса скрывается эффективный алгоритм. Интерфейс должен быть удобен для типичных действий пользователя. Для многих приложений такие действия выделены в отдельные серии экранов, называемые "мастерами" (wizards). Однако если приложение - скорее конструктор, из которого пользователь может строить нужные ему решения, типичным действием является именно построение решения. Определить типичные действия не всегда легко, поэтому компромиссом может быть гибридный вариант, в котором есть "мастера" и хорошие возможности для собственных построений. Тем не менее, графический интерфейс не является самым эффективным интерфейсом во всех случаях. Для многих предметных областей решение проще выразить с помощью деклараций на некотором формальном языке или алгоритма на сценарном языке.

Основы Tk

Основная черта любой программы с графическим интерфейсом - **интерактивность**. Программа не просто что-то считает (в пакетном режиме) от начала своего запуска до конца: ее действия зависят от вмешательства пользователя. Фактически, графическое приложение выполняет бесконечный цикл обработки событий. Программа, реализующая графический интерфейс, **событийно-ориентирована**. Она ждет от интерфейса событий, которые и обрабатывает сообразно своему внутреннему состоянию.

Эти события возникают в элементах графического интерфейса (виджетах) и обрабатываются прикрепленными к этим виджетам обработчиками. Сами виджеты имеют многочисленные свойства (цвет, размер, расположение), выстраиваются в иерархию принадлежности (один виджет может быть хозяином другого), имеют методы для доступа к своему состоянию.

Расположением виджетов (внутри других виджетов) ведают так называемые **менеджеры расположения**. Виджет устанавливается на место по правилам менеджера расположения. Эти правила могут определять не только координаты

виджета, но и его размеры. В Tk имеются три типа менеджеров расположения: простой упаковщик (pack), сетка (grid) и произвольное расположение (place).

Но этого для работы графической программы недостаточно. Дело в том, что некоторые виджеты в графической программе должны быть взаимосвязаны определенным образом. Например, полоска прокрутки может быть взаимосвязана с текстовым виджетом: при использовании полоски текст в виджете должен двигаться, и наоборот, при перемещении по тексту полоска должна показывать текущее положение. Для связи между виджетами в Tk используются переменные, через которые виджеты и передают друг другу параметры.

Классы виджетов

Для построения графического интерфейса в библиотеке Tk отобраны следующие классы виджетов (в алфавитном порядке):

1. `Button` (Кнопка) Простая кнопка для вызова некоторых действий (выполнения определенной команды).
2. `Canvas` (Рисунок) Основа для вывода графических примитивов.
3. `Checkbutton` (Флажок) Кнопка, которая умеет переключаться между двумя состояниями при нажатии на нее.
4. `Entry` (Поле ввода) Горизонтальное поле, в которое можно ввести строку текста.
5. `Frame` (Рамка) Виджет, который содержит в себе другие визуальные компоненты.
6. `Label` (Надпись) Виджет может показывать текст или графическое изображение.
7. `Listbox` (Список) Прямоугольная рамка со списком, из которого пользователь может выделить один или несколько элементов.
8. `Menu` (Меню) Элемент, с помощью которого можно создавать всплывающие (popup) и ниспадающие (pulldown) меню.
9. `Menubutton` (Кнопка-меню) Кнопка с ниспадающим меню.
10. `Message` (Сообщение) Аналогично надписи, но позволяет заворачивать длинные строки и менять размер по требованию менеджера расположения.
11. `Radiobutton` (Селекторная кнопка) Кнопка для представления одного из альтернативных значений. Такие кнопки, как правило, действует в группе. При нажатии на одну из них кнопка группы, выбранная ранее, "отскакивает".
12. `Scale` (Шкала) Служит для задания числового значения путем перемещения движка в определенном диапазоне.
13. `Scrollbar` (Полоса прокрутки) Полоса прокрутки служит для отображения величины прокрутки в других виджетах. Может быть как вертикальной, так и горизонтальной.
14. `Text` (Форматированный текст) Этот прямоугольный виджет позволяет редактировать и форматировать текст с использованием различных стилей, внедрять в текст рисунки и даже окна.
15. `Toplevel` (Окно верхнего уровня) Показывается как отдельное окно и содержит внутри другие виджеты.

Все эти классы не имеют отношений наследования друг с другом - они равноправны. Этот набор достаточен для построения интерфейса в большинстве случаев.

События

В системе современного графического интерфейса имеется возможность отслеживать различные события, связанные с клавиатурой и мышью, и происходящие на "территории" того или иного виджета. В Tk события описываются в виде текстовой строки - шаблона события, состоящего из трех элементов (модификаторы, тип события и детализация 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 |

Примеры описаний событий строками и некоторые названия клавиш приведены ниже:

"<ButtonPress-3>" или просто "<3>" - щелчок правой кнопки мыши (то есть, третьей, если считать на трехкнопочной мыши слева-направо). "<Shift-Double-Button-1>" - двойной щелчок мышью (левой кнопкой) с нажатой кнопкой Shift. В качестве модификаторов могут быть использованы следующие (список неполный):

Control, Shift, Lock,
Button1-Button5 или B1-B5,
Meta, Alt, Double, Triple.

Просто символ обозначает событие - нажатие клавиши. Например, "k" - тоже, что "<KeyPress-k>". Для неалфавитно-цифровых клавиш есть специальные названия:

Cancel, BackSpace, Tab, Return, Shift_L, Control_L, Alt_L,
Pause, Caps_Lock, Escape, Prior, Next, End, Home, Left,
Up, Right, Down, Print, Insert, Delete, F1, F2, F3, F4, F5, F6, F7,
F8, F9, F10, F11, F12, Num_Lock, Scroll_Lock, space, less

Здесь <space> обозначает пробел, а <less> - знак меньше. <Left>, <Right>, <Up>, <Down> - стрелки. <Prior>, <Next> - это PageUp и PageDown. Остальные клавиши более или менее соответствуют надписям на стандартной клавиатуре.

Примечание:

Следует заметить, что `Shift_L`, в отличие от `Shift`, нельзя использовать как модификатор.

В конкретной среде комбинации, означающие что-то особенное в системе, могут не дойти до графического приложения. Например, известный всем `Ctrl-Alt-Del`.

Следующая программа позволяет печатать направляемые виджету события, в частности - `keysym`, а также анализировать, как различные клавиши можно представить в шаблоне события:

```
from Tkinter import *
    tk = Tk()          # основное окно приложения
    txt = Text(tk)    # текстовый виджет, принадлежащий окну tk
    txt.pack()        # располагается менеджером pack

    # 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
                # готовится описание атрибута события
                ev = "%15s: %s\n" % (k, repr(getattr(event, k)))
                txt.insert(END, ev) # добавляется в конец текста

    # привязывается виджету txt функция event_info для обработки событий,
    # соответствующих шаблону <KeyPress>
    txt.bind("<KeyPress>", event_info)
    tk.mainloop()      # главный цикл обработки событий
```

При нажатии клавиши `Esc` в окне можно увидеть примерно следующее:

```
char: '\x1b'
    delta: 9
    height: 0
    keycode: 9
    keysym: 'Escape'
    keysym_num: 65307
    num: 9
    send_event: False
    serial: 159
    state: 0
    time: -1072960858
    type: '2'
    widget: <Tkinter.Text instance at 0x401e268c>
    width: 0
    x: 83
    x_root: 448
    y: 44
    y_root: 306
```

Следует объяснить некоторые из этих атрибутов:

- `char` Нажатый символ (для некоторых событий - ??)
- `height`, `width` Высота и ширина.
- `focus` Был ли в момент события фокус у окна?
- `keycode` Код символа (скан-код клавиатуры).
- `keysym` Символическое имя клавиши.
- `serial` Серийный номер события. Увеличивается по мере возникновения событий.
- `time` Время возникновения события. Все время увеличивается.
- `widget` Виджет, в котором возникло событие.
- `x`, `y` Координаты указателя в виджете во время события.

- `x_root, y_root` Координаты указателя на экране во время события.

В принципе, совсем необязательно, чтобы события обрабатывал тот же виджет, который их первично принял. Например, можно перенаправить все события внутри подчиненных виджетов на данный виджет с помощью метода `grab_set()` (`grab_release()` освобождает виджет от этой обязанности). В Tk существуют и другие возможности управления событиями, которые можно изучить по документации.

Создание и конфигурирование виджета

Создание виджета происходит вызовом конструктора соответствующего класса. Вызов конструктора имеет следующий синтаксис:

```
Widget([master[, option=value, ...]])
```

Здесь `Widget` - класс виджета, `master` - виджет-хозяин, `option` и `value` - конфигурационная опция и ее значение (таких пар может быть несколько).

Каждый виджет имеет свойства, которые можно устанавливать (конфигурировать) с помощью методов `config()` (или `configure()`) и читать с помощью методов, подобных методам работы со словарями. Ниже приведен возможный синтаксис для работы со свойствами:

```
widget.config(option=value, ...)
    widget["option"] = value
    value = widget["option"]
    widget.keys()
```

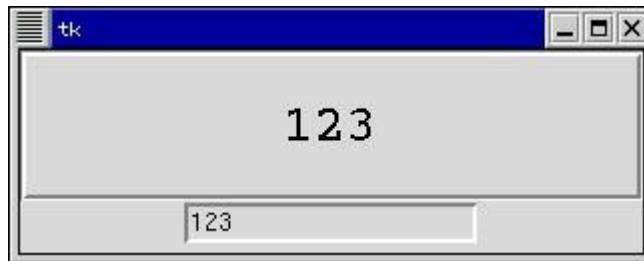
В случае, когда имя свойства совпадает с ключевым словом языка Python, принято использовать после имени одиночное подчеркивание. Так, свойство `class` нужно задавать как `class_`, а `to` как `to_`.

Изменять конфигурацию виджета можно в любой момент. Это изменение прорисовуется на экране по возвращении в цикл обработки событий или при явном вызове `update_idletasks()`.

Следующий пример показывает окно с двумя виджетами внутри - полем ввода и надписью. С помощью переменной надпись напрямую связана с полем ввода. Этот пример нарочно использует очень много свойств, чтобы продемонстрировать возможности по конфигурированию:

```
from Tkinter import *
    tk = Tk()
    tv = StringVar()
    Label(tk,
        textvariable=tv,
        relief="groove",
        borderwidth=3,
        font=("Courier", 20, "bold"),
        justify=LEFT,
        width=50,
        padx=10,
        pady=20,
        takefocus=False,
    ).pack()
    Entry(tk,
        textvariable=tv,
        takefocus=True,
    ).pack()
    tv.set("123")
    tk.mainloop()
```

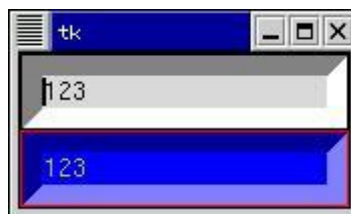
В результате на экране можно увидеть:



Виджеты конфигурируются прямо при создании. Более того, виджеты не связываются с именами, их только располагают внутри виджета-окна. В данном примере использованы свойства `textvariable` (текстовая переменная), `relief` (рельеф), `borderwidth` (ширина границы), `justify` (выравнивание), `width` (ширина, в знаках), `padx` и `pady` (прослойка в пикселях между содержимым и границами виджета), `takefocus` (возможность принять фокус при нажатии клавиши Tab), `font` (шрифт, один из способов его задания). Эти свойства достаточно типичны для многих виджетов, хотя иногда единицы измерения могут отличаться, например, для виджета `Canvas` ширина задается в пикселях, а не в знаках.

В следующем примере демонстрируются возможности по назначению цветов фону, переднему плану (тексту), выделению виджета (подсветка границы) в активном состоянии и при отсутствии фокуса:

```
from Tkinter import *
tk = Tk()
tv = StringVar()
Entry(tk,
      textvariable=tv,
      takefocus=True,
      borderwidth=10,
      ).pack()
mycolor1 = "#%02X%02X%02X" % (200, 200, 20)
Entry(tk,
      textvariable=tv,
      takefocus=True,
      borderwidth=10,
      foreground=mycolor1,           # fg, текст виджета
      background="#0000FF",         # bg, фон виджета
      highlightcolor='green',       # подсветка при фокусе
      highlightbackground='red',    # подсветка без фокуса
      ).pack()
tv.set("123")
tk.mainloop()
```



При желании можно задать стилевые опции для всех виджетов сразу: с помощью метода `tk_setPalette()`. Помимо использованных выше свойств в этом методе можно использовать `selectForeground` и `selectBackground` (передний план и фон выделения), `selectColor` (цвет в выбранном состоянии, например, у `Checkbox`), `insertBackground` (цвет точки вставки) и некоторые другие.

Примечание:

Получить значение из поля ввода можно и при помощи метода `get()`. Например, если назвать объект класса `Entry` именем `e`, получить значение можно так: `e.get()`. Правда, этот метод не обладает той же гибкостью, что метод `get()` экземпляров класса для форматированного текста `Text`: можно взять только все значение целиком.

Виджет форматированного текста

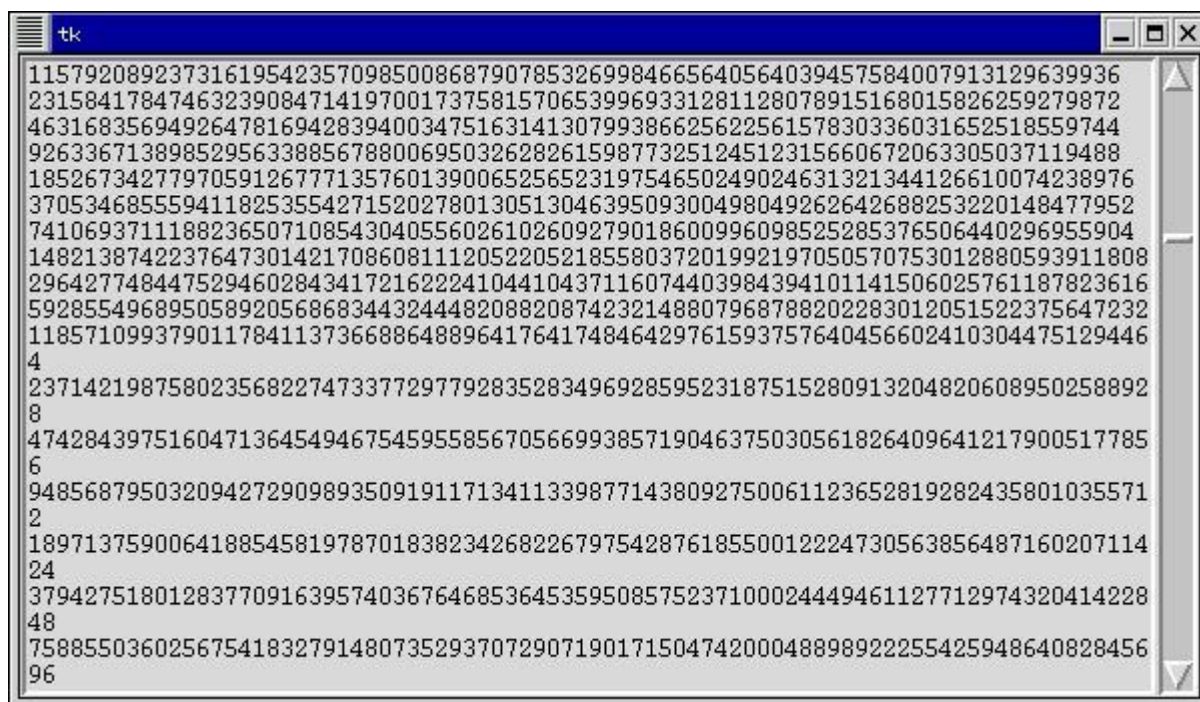
Для того чтобы показать работу с нетривиальным виджетом, можно взять виджет `ScrolledText` из одноименного модуля Python. Этот виджет аналогичен рамке с форматированным текстом и вертикальной полосой прокрутки:

```
from Tkinter import *
    from ScrolledText import ScrolledText

    tk = Tk()                                # окно верхнего уровня
    txt = ScrolledText(tk)                   # виджет текста с прокруткой
    txt.pack()                               # виджет размещается

    for x in range(1, 1024):                # виджет наполняется текстовым содержимым
        txt.insert(END, str(2L**x)+"\n")

    tk.mainloop()
```



Теперь следует рассмотреть методы и свойства виджета с форматированным текстом более подробно.

Для навигации в тексте в Tk предусмотрены специальные индексы. Индексы вроде `1.0` и `END` уже встречались - это начало текста (первая строка, нулевой символ) и его конец. (В Tk строки нумеруются с единицы, а символы строки - с нуля). Более полный список индексов:

1. `L.C` Здесь `L` - номер строки, а `C` - номер символа в строке.
2. `INSERT` Точка вставки.
3. `CURRENT` Символ, ближайший к курсору мыши.
4. `END` Позиция сразу за последним символом в тексте

5. `M.first`, `M.last` Индексы начала и конца помеченного тегом `M` участка текста.
6. `SEL_FIRST`, `SEL_LAST` Индексы начала и конца выделенного текста.
7. `M` Пользователь может определять свои именованные позиции в тексте (аналогично `END`, `INSERT` или `CURRENT`). При редактировании текста маркеры будут сдвигаться с заданными для них правилами.
8. `@x,y` Символ текста, ближайший к точке с координатами `x`, `y`.

Следующий пример показывает, как снабдить форматированный текст гипертекстовыми возможностями:

```
from Tkinter import *
import urllib
tk = Tk()
txt = Text(tk, width=64) # поле с текстом
txt.grid(row=0, column=0, rowspan=2)
addr=Text(tk, background="White", width=64, height=1) # поле адреса
addr.grid(row=0, column=1)
page=Text(tk, background="White", width=64) # поле с html-кодом
page.grid(row=1, column=1)

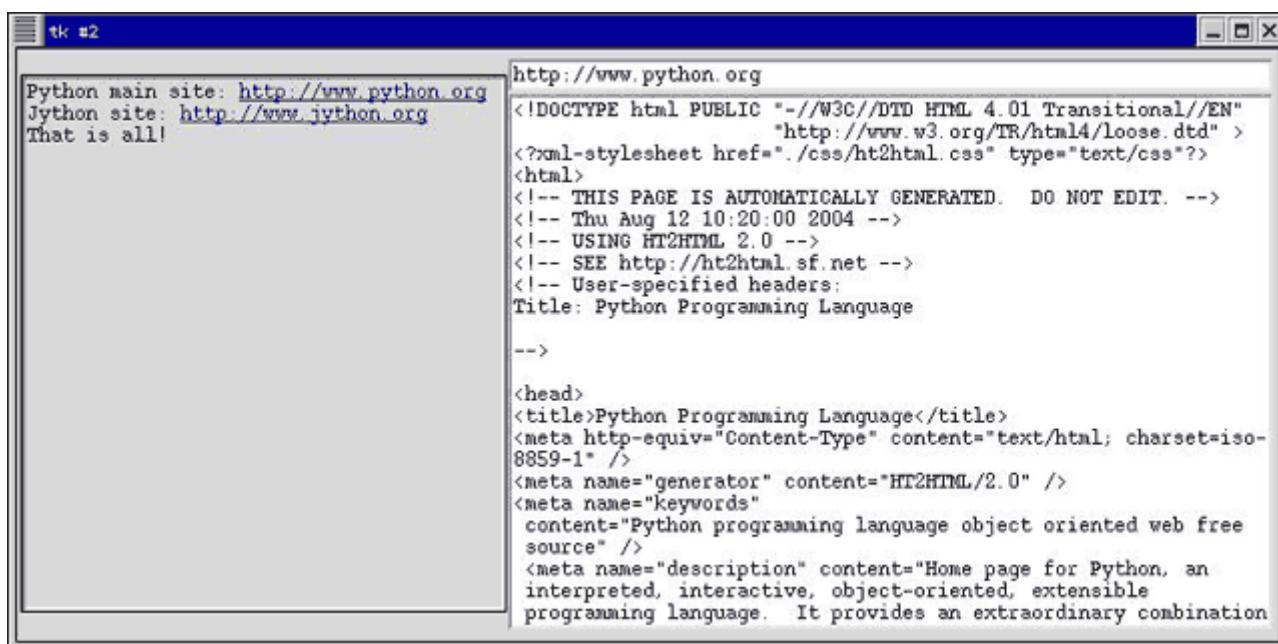
def fetch_url(event):
    click_point = "@%s,%s" % (event.x, event.y)
    trs = txt.tag_ranges("href") # список областей текста, отмеченных как href
    url = ""
    # определяется, на какой участок пришелся щелчок мыши, и берется
    # соответствующий ему URL
    for i in range(0, len(trs), 2):
        if txt.compare(trs[i], "<=", click_point) and \
            txt.compare(click_point, "<=", trs[i+1]):
            url = txt.get(trs[i], trs[i+1])
    html_doc = urllib.urlopen(url).read()
    addr.delete("1.0", END)
    addr.insert("1.0", url) # URL помещается в поле адреса
    page.delete("1.0", END)
    page.insert("1.0", html_doc) # показывается HTML-документ

textfrags = ["Python main site: ", "http://www.python.org",
             "\nJython site: ", "http://www.jython.org",
             "\nThat is all!"]
for frag in textfrags:
    if frag.startswith("http:"):
        txt.insert(END, frag, "href") # URL помещается в текст с меткой href
    else:
        txt.insert(END, frag) # фрагмент помещается в текст

# ссылки отмечаются подчеркиванием и синим цветом
txt.tag_config("href", foreground="Blue", underline=1)
# при щелчке мыши на тексте, отмеченном как "href",
# следует вызывать fetch_url()
txt.tag_bind("href", "<1>", fetch_url)

tk.mainloop() # запускается цикл событий
```

В результате (после нажатия на гиперссылку) можно увидеть примерно следующее:



```
tk #2
Python main site: http://www.python.org
Jython site: http://www.jython.org
That is all!

http://www.python.org
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd" >
<?xml-stylesheet href="/css/ht2html.css" type="text/css"?>
<html>
<!-- THIS PAGE IS AUTOMATICALLY GENERATED. DO NOT EDIT. -->
<!-- Thu Aug 12 10:20:00 2004 -->
<!-- USING HT2HTML 2.0 -->
<!-- SEE http://ht2html.sf.net -->
<!-- User-specified headers:
Title: Python Programming Language
-->

<head>
<title>Python Programming Language</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-
8859-1" />
<meta name="generator" content="HT2HTML/2.0" />
<meta name="keywords"
content="Python programming language object oriented web free
source" />
<meta name="description" content="Home page for Python, an
interpreted, interactive, object-oriented, extensible
programming language. It provides an extraordinary combination
```

Для придания некоторым участкам текста особых свойств необходимо их отметить тегом. В данном случае URL отмечается тегом `href`. Позднее с помощью метода `tag_config()` задаются свойства отображения текста, отмеченного таким тегом. Методом `tag_bind()` привязывается некоторое событие (щелчок мыши) с вызовом заданной функции (`fetch_url()`).

В самой функции `fetch_url()` нужно в начале определить, на какой именно участок текста пришелся щелчок мыши. Для этого с помощью метода `tag_ranges()` получаются все интервалы, которые отмечены как `href`. Для определения конкретного URL проводятся сравнения (методом `compare()`) точки щелчка мышью с каждым из интервалов. Так находится интервал, на который попал щелчок, и с помощью метода `get()` получается текстовое значение найденного интервала. Найдя URL, его в поле записываются адреса, и получается HTML-код, соответствующий URL.

Этот пример показывает основные принципы работы с форматированным текстом. Примененными методами арсенал виджета не исчерпывается. О других методах и свойствах можно узнать из документации.

Менеджеры расположения

Следующий пример достаточно нагляден, чтобы понять принципы работы менеджеров расположения, имеющихся в Tk. В трех рамках можно применить различные менеджеры: pack, grid и place:

```
from Tkinter import *
tk = Tk()

# Создаем три рамки
frames = {}
b = {}
for fn in 1, 2, 3:
    f = Frame(tk, width=100, height=200, bg="White")
    f.pack(side=LEFT, fill=BOTH)
    frames[fn] = f
    for bn in 1, 2, 3, 4: # Создаются кнопки для каждой из рамок
        b[fn, bn] = Button(frames[fn], text="%s.%s" % (fn, bn))

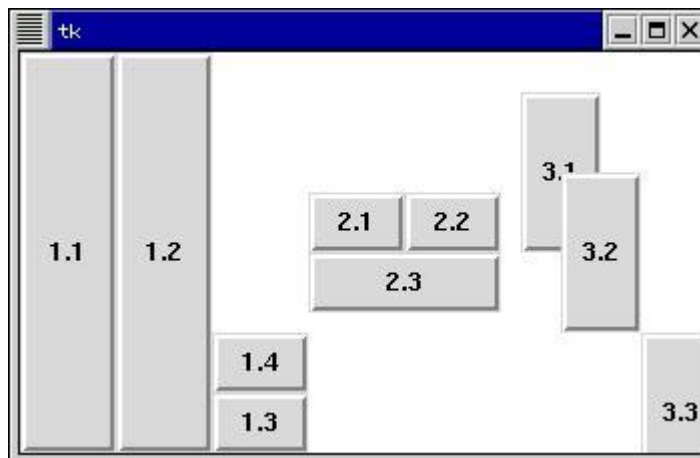
# Первая рамка:
# Сначала две кнопки прикрепляются к левому краю
b[1, 1].pack(side=LEFT, fill=BOTH, expand=1)
b[1, 2].pack(side=LEFT, fill=BOTH, expand=1)
# Еще две - к нижнему
b[1, 3].pack(side=BOTTOM, fill=Y)
b[1, 4].pack(side=BOTTOM, fill=BOTH)

# Вторая рамка:
# Две кнопки сверху
b[2, 1].grid(row=0, column=0, sticky=NW+SE)
b[2, 2].grid(row=0, column=1, sticky=NW+SE)
# и одна на две колонки в низу
b[2, 3].grid(row=1, column=0, columnspan=2, sticky=NW+SE)

# Третья рамка:
# Кнопки высотой и шириной в 40% рамки, якорь в левом верхнем углу.
# Координаты якоря 1/10 от ширины и высоты рамки
b[3, 1].place(relx=0.1, rely=0.1, relwidth=0.4, relheight=0.4, anchor=NW)
# Кнопка строго по центру. Якорь в центре кнопки
b[3, 2].place(relx=0.5, rely=0.5, relwidth=0.4, relheight=0.4, anchor=CENTER)
# Якорь по центру кнопки. Координаты якоря 9/10 от ширины и высоты рамки
b[3, 3].place(relx=0.9, rely=0.9, relwidth=0.4, relheight=0.4, anchor=CENTER)

tk.mainloop()
```

Результат следующий:



Менеджер pack просто заполняет внутреннее пространство на основании предпочтения того или иного края, необходимости заполнить все измерение. В некоторых случаях ему приходится менять размеры подчиненных виджетов. Этот менеджер стоит использовать только для достаточно простых схем расположения виджетов.

Менеджер `grid` помещает виджеты в клетки сетки (это очень похоже на способ верстки таблиц в HTML). Каждому располагаемому виджету даются координаты в одной из ячеек сетки (`row` - строка, `column` - столбец), а также, если нужно, столько последующих ячеек (в строках ниже или в столбцах правее) сколько он может занять (свойства `rowspan` или `columnspan`). Это самый гибкий из всех менеджеров.

Менеджер `place` позволяет располагать виджеты по произвольным координатам и с произвольными размерами подчиненных виджетов. Размеры и координаты могут быть заданы в долях от размера виджета-хозяина.

Непосредственно внутри одного виджета нельзя использовать более одного менеджера расположения: менеджеры могут наложить противоречащие ограничения на вложенные виджеты и внутренние виджеты просто не смогут быть расположены.

Изображения в Tkinter

Средствами Tkinter можно выводить не только текст, примитивные формы (с помощью виджета `Canvas`), но и растровые изображения. Следующий пример демонстрирует вывод иконки с растровым изображением (для этого примера нужно предварительно установить пакет Python Imaging Library, PIL):

```
import Tkinter, Image, ImageTk

FILENAME = "lena.jpg" # файл с графическим изображением

tk = Tkinter.Tk()
c = Tkinter.Canvas(tk, width=128, height=128)
src_img = Image.open(FILENAME)

img = ImageTk.PhotoImage(src_img)
c.create_image(0, 0, image=img, anchor="nw")
c.pack()
Tkinter.Label(tk, text=FILENAME).pack()

tk.mainloop()
```

В результате получается:



Здесь использован виджет-рисунок (`Canvas`). С помощью функций из пакетов `Image` и `ImageTk` из PIL получается объект-изображение, подходящее для включения в рисунок Tkinter. Свойство `anchor` задает угол, который привязывается к координатам (0, 0) в рисунке. В данном примере это северо-западный угол (NW - North-West). Другие возможности: `n` (север), `w` (запад), `s` (юг), `e` (восток), `ne`, `sw`, `se` и `c` (центр).

В следующем примере показаны графические примитивы, которые можно использовать на рисунке (приведенные комментарии объясняют свойства графических объектов внутри виджета-рисунка):

```

from Tkinter import *

tk = Tk()
# Рисунок 300x300 пикселей, фон - белый
c = Canvas(tk, width=300, height=300, bg="white")

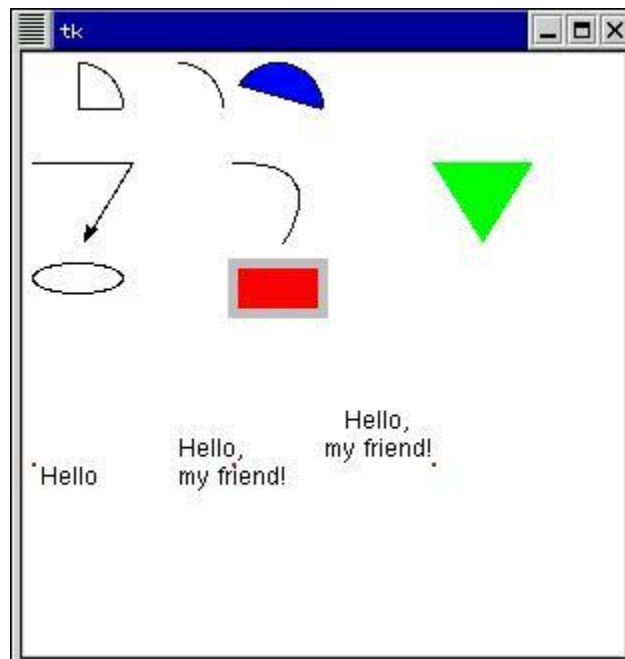
c.create_arc((5, 5, 50, 50), style=PIESLICE) # Сектор ("кусочек пирога")
c.create_arc((55, 5, 100, 50), style=ARC) # Дуга
c.create_arc((105, 5, 150, 50), style=CHORD, # Сегмент
             start=0, extent=150, fill="blue") # от 0 до 150 градусов
# Ломаная со стрелкой на конце
c.create_line([(5, 55), (55, 55), (30, 95)], arrow=LAST)
# Кривая (сглаженная ломаная)
c.create_line([(105, 55), (155, 55), (130, 95)], smooth=1)
# Многоугольник зеленого цвета
c.create_polygon([(205, 55), (255, 55), (230, 95)], fill="green")
# Овал
c.create_oval((5, 105, 50, 120), )
# Прямоугольник красного цвета с большой серой границей
c.create_rectangle((105, 105, 150, 130), fill="red",
                  outline="grey", width="5")

# Текст
c.create_text((5, 205), text=" Hello", anchor="nw")
# Эта точка визуально обозначает угол привязки
c.create_oval((5, 205, 6, 206), outline="red")
# Текст с заданным выравниванием
c.create_text((105, 205), text="Hello,\nmy friend!",
             justify=LEFT, anchor="c")
c.create_oval((105, 205, 106, 206), outline="red")
# Еще один вариант
c.create_text((205, 205), text="Hello,\nmy friend!",
             justify=CENTER, anchor="se")
c.create_oval((205, 205, 206, 206), outline="red")

c.pack()
tk.mainloop()

```

В результате работы этой программы на экране появится окно:



Следует заметить, что методы `create_*` создают объекты, свойства которых можно менять в дальнейшем: переместить в другое место, перекрасить, удалить, изменить порядок и т.д. В следующем примере можно нарисовать кружок, меняющий цвет по щелчку мыши:

```
from Tkinter import *
    from random import choice

    colors = "Red Orange Yellow Green LightBlue Blue Violet".split()
    R = 10

    tk = Tk()
    c = Canvas(tk, bg="White", width="4i", height=300, relief=SUNKEN)
    c.pack(expand=1, fill=BOTH)

    def change_ball(event):
        c.coords(CURRENT, (event.x-R, event.y-R, event.x+R, event.y+R))
        c.itemconfigure(CURRENT, fill=choice(colors))

    oval = c.create_oval((100-R, 100-R, 100+R, 100+R), fill="Black")
    c.tag_bind(oval, "<1>", change_ball)
    tk.mainloop()
```

Здесь нарисован кружок радиуса R , с ним связана функция `change_ball()` по нажатию кнопки мыши. В указанной функции заданы новые координаты кружка (его центр расположен в месте щелчка мыши) и затем изменен цвет случайным образом методом `itemconfigure()`. Тег `CURRENT` в Tkinter использован для указания объекта, который принял событие.

Графическое приложение на Tkinter

Теперь следует рассмотреть небольшое приложение, написанное с использованием Tkinter. В этом приложении будет загружен файл с графическим изображением. Приложение будет иметь простейшее меню File с пунктами Open и Exit, а также виджет Canvas, на котором и будут демонстрироваться изображения (опять потребуется пакет PIL):

```
from Tkinter import *
import Image, ImageTk, tkFileDialog
global img, imgobj

def show():
    global img, imgobj
    # Запрос на имя файла
    filename = tkFileDialog.askopenfilename()
    if filename != (): # Если имя файла было задано пользователем
        # рисуется изображение из файла
        src_img = Image.open(filename)
        img = ImageTk.PhotoImage(src_img)
        # конфигурируется изображение на рисунке
        c.itemconfigure(imgobj, image=img, anchor="nw")

tk = Tk()
main_menu = Menu(tk) # формируется меню
tk.config(menu=main_menu) # меню добавляется к окну
file_menu = Menu(main_menu) # создается подменю
main_menu.add_cascade(label="File", menu=file_menu)
# Заполняется меню File
file_menu.add_command(label="Open", command=show)
file_menu.add_separator() # черта для отделения пунктов меню
file_menu.add_command(label="Exit", command=tk.destroy)

c = Canvas(tk, width=300, height=300, bg="white")
# готовим объект-изображение на рисунке
imgobj = c.create_image(0, 0)
c.pack()

tk.mainloop()
```

Приложение (с загруженной картинкой) будет выглядеть так:



Стоит отметить, что здесь пришлось применить две глобальные переменные. Это не очень хорошо. Существует другой подход, когда приложение создается на основе окна верхнего уровня. Таким образом, само приложение становится особым виджетом. Переделанная программа представлена ниже:

```
from Tkinter import *
    import Image, ImageTk, tkFileDialog

class App(Tk):
    def __init__(self):
        Tk.__init__(self)
        main_menu = Menu(self)
        self.config(menu=main_menu)
        file_menu = Menu(main_menu)
        main_menu.add_cascade(label="File", menu=file_menu)
        file_menu.add_command(label="Open", command=self.show_img)
        file_menu.add_separator()
        file_menu.add_command(label="Exit", command=self.destroy)

        self.c = Canvas(self, width=300, height=300, bg="white")
        self.imgobj = self.c.create_image(0, 0)
        self.c.pack()

    def show_img(self):
        filename = tkFileDialog.askopenfilename()
        if filename != ():
            src_img = Image.open(filename)
            self.img = ImageTk.PhotoImage(src_img)
            self.c.itemconfigure(self.imgobj, image=self.img, anchor="nw")

app = App()
app.mainloop()
```

В объекте заключена информация, которая до этого была глобальной со всеми следующими из этого ограничениями. Можно пойти дальше и выделить в отдельный метод настройку меню (если приложение будет динамически изменять меню, объекты-меню тоже могут быть сохранены в приложении).

Примечание:

На некоторых системах новые версии Python плохо работают с национальными кодировками, в частности, с кодировками для кириллицы. Это связано с переходом на Unicode Tcl/Tk. Проблем можно избежать, если использовать кодировку UTF-8 в строках, которые должны выводиться в виджетах.

Заключение

В этой лекции было дано представление о (невизуальном) программировании графического интерфейса для Python на примере пакета Tkinter. Программа с графическим интерфейсом - событийно-управляемая программа, проводящая время в цикле обработки событий. События могут быть вызваны функционированием графического интерфейса или другими причинами (например, по таймеру). Обычно события возникают в виджетах и некоторые из них должны обрабатываться приложением. В Tkinter событие представлено отдельным объектом, из атрибутов которого можно установить, каково было положение указателя (курсора мыши), в каком виджете произошло событие и т.п.

Здесь были рассмотрены классы элементов интерфейса (виджеты), их свойства и методы. Виджеты имеют большое количество свойств и методов. Некоторые свойства и методы достаточно универсальны (их имеют все или почти все виджеты), другие же специфичны для конкретного класса виджетов. Графический пакет Python Imaging Library (PIL) предоставляет класс объекта для расположения в виджете-рисунке растрового графического изображения.

Виджеты располагаются внутри другого виджета (например, рамки) в соответствии с набором правил. Этот набор правил реализуют менеджеры расположения, которых в Tkinter три: pack, grid и place.

Приложение с графическим интерфейсом можно построить на базе окна верхнего уровня, простым наследованием. Этот подход позволяет инкапсулировать информацию, которую в противном случае пришлось бы делать глобальной.

Нужно отметить, что для построения интерфейса можно использовать не только чистый Tkinter. Например, в Python доступны модули `ScrolledText` и `Tix`, пополняющие набор виджетов. Кроме того, можно найти пакеты для специальных виджетов (например, для отображения дерева).

Построение графического интерфейса невизуальными способами - не такая сложная задача, если использовать Tkinter. Этот пакет входит в стандартную поставку Python и потому может использоваться почти везде, где установлен Python.