Tkinter

It’s over

Виджет TopLevel

- TopLevel – это виджет верхнего уровня. Данный виджет обычно используется для создания многооконных программ. А также для диалоговых окон.

Toplevel

Toplevel[5] - окно верхнего уровня. Обычно используется для создания многооконных программ, а также для диалоговых окон.

Методы виджета

- **title** - заголовок окна
- **overrideredirect** - указание оконному менеджеру игнорировать это окно. Аргументом является True или False. В случае, если аргумент не указан - получаем текущее значение. Если аргумент равен True, то такое окно будет показано оконным менеджером без обрамления (без заголовка и бордюра). Может быть использовано, например, для создания splashscreen при старте программы.
- **iconify / deiconify** - свернуть / развернуть окно
- **withdraw** - "спрятать" (сделать невидимым) окно. Для того, чтобы снова показать его, надо использовать метод *deiconify*.
- **minsize** и **maxsize** - минимальный / максимальный размер окна. Методы принимают два аргумента - ширина и высота окна. Если аргументы не указаны - возвращают текущее значение.
- **state** - получить текущее значение состояния окна. Может возвращать следующие значения: normal (нормальное состояние), icon (показано в виде иконки), iconic (свёрнуто), withdrawn (не показано), zoomed (развёрнуто на полный экран, только для Windows и Mac OS X)
- **resizable** - может ли пользователь изменять размер окна. Принимает два аргумента - возможность изменения размера по горизонтали и по вертикали. Без аргументов возвращает текущее значение.
- **geometry** - устанавливает геометрию окна в формате ширинахвысота+x+y (пример: geometry("600x400+40+80") - поместить окно в точку с координатам 40,80 и установить размер в 600x400). Размер или координаты могут быть опущены (geometry("600x400") - только изменить размер, geometry("+40+80") - только переместить окно).
- **transient** - сделать окно зависимым от другого окна, указанного в аргументе. Будет сворачиваться вместе с указанным окном. Без аргументов возвращает текущее значение.
- **protocol** - получает два аргумента: название события и функцию, которая будет вызываться при наступлении указанного события. События могут называться WM_TAKE_FOCUS (получение фокуса), WM_SAVE_YOURSELF (необходимо сохраниться, в настоящий момент является устаревшим), WM_DELETE_WINDOW (удаление окна).
- **tkraise** (синоним lift) и **lower** - поднимает (размещает поверх всех других окон) или опускает окно. Методы могут принимать один необязательный аргумент: над/под каким окном разместить текущее.
- **grab_set** - устанавливает фокус на окно, даже при наличии открытых других окон
- **grab_release** - снимает монопольное владение фокусом ввода с окна

- Создадим небольшую демонстрацию возможностей виджета:
- Создадим кнопку на окне root:

```
31
32      Button(root, text="Open", command=open_win, padx=40, pady=5).place(relx=0.5, rely=0.5, anchor=CENTER)
33
```

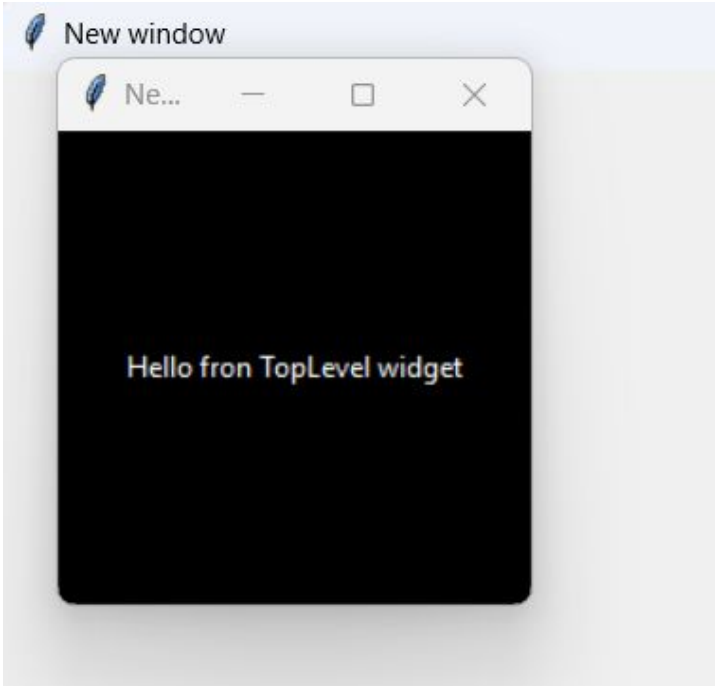
- Для того, чтобы создать новое окно, мы используем виджет «TopLevel»:

```
def open_win():
    win = Toplevel()
```

- Для него доступны всё те же свойства, что и для корневого окна: мы можем указать и разрешение окна, и background и т.д.

```
def open_win():
    win = Toplevel()
    win.geometry("200x200")
    Label(win, text="Hello fron TopLevel widget", background="#000", foreground="#fff").pack(expand=1, fill=BOTH)
```

- Результат работы кода:



- Итог: у нас появилось наше дополнительное окно. При этом у нас есть доступ и к родительскому окну, что во многих приложениях не допускается (когда у нас активно дочернее окно – окно верхнего уровня, то родительское банально просто будет не доступно).
- Как это можно отключить? За это отвечает метод grab_set – он устанавливает фокус на окно, даже при наличии других открытых окон.

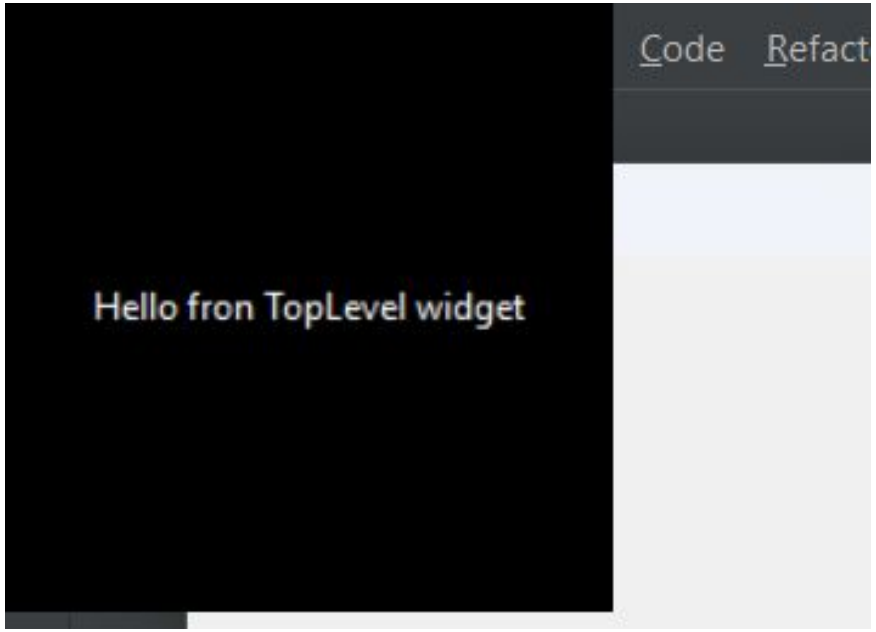
```
def open_win():
    win = Toplevel()
    win.geometry("200x200")
    win.grab_set()
    Label(win, text="Hello fron TopLevel widget", background="#000", foreground="#fff").pack(expand=1, fill=BOTH)
```

- Теперь закрыть программу из корневого родительского окна не получится, пока мы не закроем наше дочернее.
- На этом дочернем окне вы можете размещать всё, что угодно и можно было размещать на родительском окне: формы, текстовые поля, кнопки и т.д. Данное окно ничем не отличается от родительского, оно просто вызывается из родительского.

- Еще один метод, который можно назвать полезным – это метод overrideredirect.
- Overrideredirect – указание одному оконному менеджеру игнорировать это окно. Аргументов является True или False. В случае, если аргумент не указан – получаем текущее значение. Если аргумент равен True, то такое окно будет показано оконным менеджером без обрамления (без заголовка и бордюра). Может быть использовано, к примеру, для создания splashscreen’а при старте программы.
- Используется следующим образом:

```
def open_win():
    win = Toplevel()
    win.geometry("200x200")
    win.overrideredirect(True)
    win.grab_set()
    Label(win, text="Hello fron TopLevel widget", background="#000", foreground="#fff").pack(expand=1, fill=BOTH)
```

- Но с получением фокуса на это окно – мы его никак не закроем:



- Поэтому данный метод лучше применять без grab_set метода (чтобы он банально не получил на этот самый фокус на окно), либо если с этим методом мы можем удалять заставку через некоторое время.
- В этом нам поможет метод, с которым мы уже работали, под названием after:

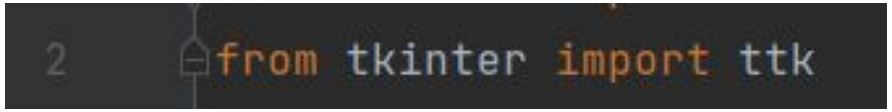
```
23    def open_win():
24        win = Toplevel()
25        win.geometry("200x200")
26        win.overrideredirect(True)
27        win.grab_set()
28        Label(win, text="Hello fron TopLevel widget", background="#000", foreground="#fff").pack(expand=1, fill=BOTH)
29        win.after(3000, lambda: win.destroy())
```

Библиотека TTK

- Ну и последние темы tkinter’а будут посвящены оформлению и стилизации виджетов с помощью библиотеки ttk.
- Ttk (themed tk) – это расширение tcl/tk с новым набором виджетов. В ttk используется новый движок для создания виджетов. Этот движок обладает поддержкой тем и стилей оформления. Благодаря этому виджеты ttk выглядят более естественно в различных операционных системах.

- Очевидно, все мы знаем и помним, что когда мы использовали виджеты «из коробки», без всяких прикрас, то их внешний вид как бы мягко говоря был (и есть) «не очень».
- В модуле ttk есть даже несколько тем на выбор.
- В частности, ttk имеет четыре встроенных темы: default, classic, alt, clam. Кроме того, дополнительно по Windows есть темы winnative, xpnative и vista, а под Mac OS X – aqua.
- Кроме того, есть еще и пакеты с темами, которые можно скачать с репозитория PyPi, которым мы уже пользовались.

- В ttk включены следующие виджеты, которые можно использовать вместо соответствующих виджетов tk: Button, Checkbutton, Entry, Frame, Label, LabelFrame, Menubutton, PanedWindow, Radiobutton, Scale и Scrollbar.
- Кроме того имеется несколько новых виджетов: Combobox, Notebook, Progressbar, Separator, Sizegrip и Treeview.
- Чтобы использовать ttk, достаточно его отдельно подключить.



- И попробуем явно переопределить тему. Но для начала создадим какой-нибудь элемент на окне root:

```
14    btn_click = Button(text="ttk theme")
15    btn_click.pack()
16    ttk.Button(text="ttk theme 2").pack()
```

- Результат:

- Реализация изменения ширины кнопки при использовании ttk следующая:

```
16    ttk.Button(text="ttk theme 2", width=21, ).pack()
17
```

- А с высотой подобное не прокатит. Атрибута height для подобного типа нет в модуле. Вот тут то как раз мы и можем добавлять собственные стили оформления.
- Но для начала поработаем с стилями оформления в целом.

- У нас есть 4 предустановленных темы + 4 под винду и одна под мак.
- В этом нам поможет класс Style.
- Style – это класс для работы со стилями и темами. Именно этот класс надо использовать для конфигурирования внешнего вида виджетов.
- Создадим его:

```
9   s = ttk.Style()
10  print(s.theme_names())
```

- Мы его создаём, а заодно и можем посмотреть все возможные темы, которые он предлагает.

- Собственно, темы:

```
main ×
C:\Users\79137\PycharmProjects\pythonProject\venv\Scripts\python.exe C:/Users/79137/PycharmProjects/pythonProject/main.py
('winnative', 'clam', 'alt', 'default', 'classic', 'vista', 'xpnative')
```

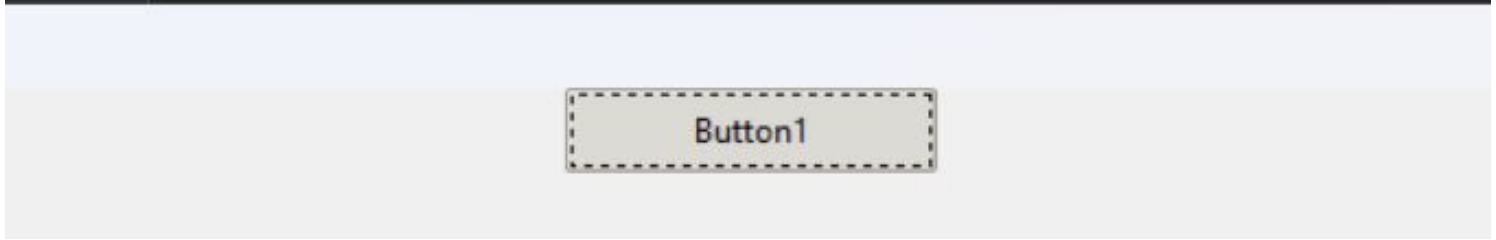
- Для того, чтобы посмотреть, какая тема используется по умолчанию, можно воспользоваться следующим методом:

```
s = ttk.Style()
print(s.theme_names(), s.theme_use())
```

- По умолчанию обычно используется «Vista».

- Для того, чтобы переопределить тему, мы можем воспользоваться тем же самым методом s.theme_use() и указать, какую тему мы хотим использовать.

```
8
9   s = ttk.Style()
10  print(s.theme_names())
11  s.theme_use('clam')
12
13  ttk.Button(root, text="Button1", width=20).pack()
14
```



- Можете самостоятельно пожамкать различные темы и остановиться на той, какая понравится больше всего.
- Соответственно, глобально для всех виджетов будет выбрана именно та тема, которую вы запишите в методе.

- Что если нам нужно точечно подредактировать стили для какого-либо виджета?
- Для манипулирования стилями, применяется уже метода конфига.
- Выглядит он следующим образом: «style.configure()».
- Он отвечает за конфигурирование внешнего вида виджетов. В качестве аргументов принимает название стиля виджета (например, «TButton») и список опций конфигурирования.
- Для того, чтобы использовать данный метод от объекта s, объекта класса style, мы должны использовать следующий формат:

```
s.configure(".", foreground="red")
```

- Формат следующий: первым параметром идёт название стиля, а следующим параметром идут опции.
- Что касается названия стиля: во-первых, в качестве названия стиля можно использовать точку (которая на скриншоте). Точка – это означает «все». Т.е. все виджеты, все элементы – для всех элементов назначается данный стиль.
- Стиль будет применён глобально, ко всем компонентам, но если компонент поддерживает подобный вариант стиля.
- Например в листинге выше я указал цвет шрифта «foreground = “red”».
- Кстати в tkinter’е обычном мы могли использовать сокращение fg, но здесь надо писать полностью. И если мы напишем так, то для всех элементов ttk будет использован красный цвет шрифта.

- Не факт, конечно, что вам потребуется применять подобное ко всем элементам в приложении, но кто знает.
- Другая возможность – это точечно определять стили для каких-нибудь элементов и вот здесь вариант названия опции уже отличается: во-первых, используется большая буква «T», а дальше название виджета. Например, если нам нужна кнопка, то это будет «TButton». Если entry, то «TEntry» и т.д.

```
s.configure("TButton", foreground="red")
```

- Таким образом, для всех кнопок у нас будет красный цвет шрифта. Но только для кнопок.

- В качестве примера записи:

```
s = ttk.Style()
print(s.theme_names())
s.theme_use('clam')

s.configure("TButton", foreground="red")

ttk.Button(root, text="Button1", width=20).pack()
ttk.Button(root, text="Button2", width=20).pack()
ttk.Entry(root, width=20).pack()
```

- Если же мы хотим еще более точечно применить стили не только лишь ко всем виджетам типа Button, а к какой-либо конкретной кнопке, то можно сделать следующее:

- Ну и последнее, на что обратим внимание на этой паре – это виджет, который относится исключительно к библиотеке ttk.
- Виджет Combobox.
- Создадим его:

```
select = ttk.Combobox(root, values=["Январь", "Февраль", "Март", "Апрель", "Май", "Июнь"])
```

- Первым параметром мы размещаем его на окне root, а вторым устанавливаем значения его списка.
- Двух этих параметров достаточно.

```
select = ttk.Combobox(root, values=["Январь", "Февраль", "Март", "Апрель", "Май", "Июнь"])
select.place(relx=0.5, rely=0.5, anchor=CENTER)
```

- Как можно заметить – изначальное значение у нас пустое – как поставить начальное значение?
- Это делается с помощью метода «current», применяем его и указываем индекс того элемента, который хотим поставить изначальным значением:

```
select = ttk.Combobox(root, values=["Январь", "Февраль", "Март", "Апрель", "Май", "Июнь"])
select.place(relx=0.5, rely=0.5, anchor=CENTER)
select.current(2)
```

- Ну и как отследить выбор какого-либо другого элемента списка – для этого мы навешиваем на наш комбо бокс метод.
- Но навешиваем его через bind:

```
select = ttk.Combobox(root, values=["Январь", "Февраль", "Март", "Апрель", "Май", "Июнь"])
select.place(relx=0.5, rely=0.5, anchor=CENTER)
select.current(2)
select.bind("<<ComboboxSelected>>", choose)
```

- И по данному событию мы будем выполнять какую-нибудь функцию «choose».

- Для того, чтобы получить выбранный элемент есть два элемента: это элемент current, который возвращает индекс выбранного элемента и метод get(), который возвращает значение индекса.
- Ну и не забываем, что при использовании метода bind, нам необходимо по умолчанию указать параметр события event в функции – обязательное условие при использовании bind.

```
def choose(event):
    print(select.current(), select.get())
```

- Запускаем и тыкаем :)