Лекция 14

Разработка приложений с графическим интерфейсом пользователя

Библиотека Tkinter

Библиотека Tkinter

- Библиотека реализована в модуле с именем
  - tkinter Питоне версии 3
  - Tkinter Питоне версии 2
- Традиционная инструкция импорта:

```
from tkinter import *
```

- Модуль tkinter входит в стандартную поставку Питона и является базовым средством создания приложений с графическим пользовательским интерфейсом (Graphical Users Interface или GUI)
- Как альтернатива GUI существует термин CLI (Command Line Interface)
- CLI-программы имеют начало и конец, GUI-программы исполняют бесконечный цикл

Виджет

- *Виджет* это объект видимый на экране, как правило прямоугольной формы
- Графический пользовательский интерфейс формируется из виджетов, tkinter это *библиотека виджетов*
- Виджет может находиться внутри другого виджета, что формирует отношение родитель - потомок или master - slave
- Главное окно программы является корнем дерева виджетов (root widget)
- Обход дерева виджетов позволяет сформировать изображение на экране
- По ходу работы программы внешний вид виджета может меняться, при этом вызывается функция которая вновь отображает виджет на экране (обновляет виджет)
- При отображении используется двойная буферизация

Программа управляемая событиями

- Программа выполняет бесконечный цикл в котором происходит проверка событий
- С событием связывается код, исполняемый при возникновении события, этот код называют *обработчик события* (event handler или callback)
- Обработчик события оформляется в виде функции или связанного метода
- Источники событий:
  - мышь
  - клавиатура
  - таймер
  - завершение процесса или треда
- Программирование приложения с графическим пользовательским интерфейсом сводится к созданию иерархии виджетов и написанию обработчиков событий

События и виджеты

- На уровне прикладного программирования используется метафора "виджет генерирует событие"
- Связывание события с виджетом или *маршрутизация* события происходит внутри библиотеки.
- Событие мыши связывается с виджетом, если курсор находится над виджетом
- Событие клавиатуры связывается с виджетом, если виджет является активным для клавиатурного ввода или, другими словами, *виджет имеет фокус ввода*
- Часть событий обрабатывается внутри библиотеки. Такое событие может передаваться или не передаваться на уровень прикладного программирования
- Установка обработчика события не обязательна. В отсутствие обработчика событие будет проигнорировано.

Пример приложения

- Простое приложение, процедурный стиль

```
from tkinter import *
root = Tk()
Label(root, text='Simple application').pack()
Button(root, text='Quit', command=root.destroy).pack()
root.mainloop()
```

- Простое приложение, объектно-ориентированный стиль

```
from tkinter import *

class Application():
  def __init__(self):
    self.root = Tk()
    Label(self.root, text='Object-style application').pack()
    Button(self.root, text='Quit', command=root.destroy).pack()
    self.root.mainloop()

if __name__ == '__main__':
  Application()
```

Устаревший стиль оформления приложений

- Вызов Tk() это создание объекта класса Tk. При этом объект сохраняется в глобальной переменной, которая доступна функциям библиотеки и представляет собой корневой виджет или главное окно приложения.
- Большинство функций, получающих родительский виджет в качестве параметра, по умолчанию принимают корневой виджет:

```
from tkinter import *

Tk()
Label(text='Simple application').pack()
Button(text='Quit', command=exit).pack()
mainloop()
```

Такой стиль написания приложений настоятельно не рекомендуется

Классы виджетов

- Canvas : "холст", на нем можно "рисовать"
- Button : кнопка
- Checkbutton : кнопка-флажок
- Radiobutton : кнопка-переключатель
- Entry : однострочный редактор текста
- Text : многострочный редактор текста
- ScrolledText : многострочный редактор текста с возможностью прокрутки
- Frame : рамка, внутри рамки можно размещать другие виджеты
- LabelFrame : рамка с надписью для группировки виджетов
- Label : текстовая строка, надпись
- Message : надпись из нескольких текстовых строк
- Listbox : меню в виде списка с возможностью выбора нескольких элементов
- Spinbox : список выбора с двумя стрелками (колесом) прокрутки
- PanedWindow : окно из нескольких зон с подвижными разделителями
- Scale : шкала с ползунковым регулятором (слайдер)
- Scrollbar : скроллбар (полоса прокрутки)
- Menu : главное меню и подменю в главном меню
- Menubutton : кнопка инициирующая выпадающее меню класса Menu
- OptionMenu : выпадающее меню

Класс Button, аргументы

- text : текст отображаемый внутри кнопки
- image : объект-картинка отображаемая вместо текста
- bitmap : имя одной из стандартных картинок вместо текста
- height : высота кнопки в буквах или пикселях
- width : ширина кнопки в буквах или пикселях
- bd или borderwidth : ширина рамки вокруг кнопки (2 пикселя)
- anchor : позиция текста внутри кнопки в виде сторон света или CENTER
- justify : привязка нескольких строк: LEFT, CENTER, RIGHT
- padx : дополнительное пространство слева и справа
- pady : дополнительное пространство сверху и снизу
- font : название шрифта для отображения текста
- state : состояние кнопки DISABLED, ACTIVE или NORMAL (NORMAL)
- default : исходное состояние кнопки - NORMAL или DISABLED (NORMAL)
- takefocus : 0 или 1 - может получить фокус ввода с клавиатуры
- В случае прохождения курсора над кнопкой можно изменить
  - cursor : курсор
  - overrelief : стиль кнопки (рельеф)
- repeatdelay : задержка автоматического нажатия
- repeatinterval : интервал автоматического нажатия
- command : функция-обработчик сигнала

Button, управление цветом

- Цвета кнопки в нормальном состоянии
  - fg или foreground : цвет текста
  - bg или background : цвет фона
- Цвета кнопки в активном (нажатом) состоянии
  - activeforeground : цвет текста
  - activebackground : цвет фона
- Цвета кнопки в запрещенном состоянии
  - disabledforeground : цвет текста
  - bg или background : цвет фона
- Цвета кнопки имеющей фокус ввода
  - highlightbackground : цвет текста
  - highlightcolor : цвет фона
  - highlightthickness : толщина рамки

Аргумент command

- Аргумент command позволяет связать с виджетом функцию (установить callback), которая будет вызываться при воздействии пользователя на виджет
- Callback-функция может быть установлена для виджетов:
  - Button
  - Checkbutton
  - Radiobutton
  - Scale
  - Scrollbar
  - Spinbox
- При вызове callback-функции ей может быть передан аргумент. Для Scrollbar это направление перемещения, для Scale - текущее положение движка

Callback в виде функции

- Пример установки callback функции, вызываемой при щелчке левой кнопкой мыши на изображении кнопки:

```
def on_run():
  print('Run button clicked')

root = Tk()
Button(root, text='Run', command=on_run).pack()
Button(root, text='Quit', command=root.destroy).pack()
root.mainloop()
```

- Универсальная функция-callback оформляется с возможностью передачи любого количества позиционных и/или именованных аргументов:

```
def on_run(*pargs, **nargs):
  print('Run button clicked')
```

Callback в виде метода класса

- При установке callback в качестве параметра должно быть указано *имя функции*
- Метод класса может быть связан с объектом. Такой *связанный метод* эквивалентен функции и также может быть использован как callback:

```
class Application():
  def __init__(self):
    self.root = Tk()
    Button(self.root, text='Run', command=self.on_run).pack()
    Button(self.root, text='Quit', command=self.root.destroy).pack()
    self.root.mainloop()

  def on_run(self):
    print('Run button clicked')
```

- Включение обработчиков событий в класс, содержащий код создания виджетов, это пример инкапсуляции присущей объектно-ориентированному стилю программирования

Метод config

- Метод *config()* позволяет устанавливать параметры виджета после того как он был создан
- Метод config воспринимает любое количество именованных аргументов; это те же аргументы, которые могут быть переданы виджету при его создании:

```
config(**options)
```

- Таким образом, метод config позволяет менять свойства (опции) виджета по ходу исполнения программы
- При вызове метода config без параметров он возвращает словарь, содержащий текущие свойства виджета

Вызов без параметров полезен при отладке или при необходимости изменить значение опции на противоположное

Объекты-переменные

- С виджетами могут быть связаны специальные *объекты-переменные* хранящие состояние виджета
- Объекты-переменные имеют тип классов-оберток, позволяющих библиотеке tkinter отслеживать изменение связанных величин
- Начальное значение переменной может быть задано параметром value при создании объекта-переменной
- Классы объектов-переменных:
  - *StringVar* - текстовая строка
  - *IntVar* - целое число
  - *DoubleVar* - число с плавающей запятой
  - *BooleanVar* - логическая величина True / False
- Методы
  - get() - получить текущее значение величины
  - set(val) - установить новое значение величины

Установка callback

- Установка функции-обработчика (callback) для объектов-переменных осуществляется вызовом метода trace()

```
trace(mode, callback) - установить функцию callback,
                        которая будет вызвана:

  - при вызове метода get(), если mode='r' или при чтении переменной
  - при вызове метода set(), если mode='w' или при записи переменной
  - при вызове метода get() или set(), если mode='rw'
```

- Функция callback должна быть определена, как функция, воспринимающая любое количество позиционных аргументов:

```
def cb(*args):
  print('Callback called')

vs = StringVar()
vs.trace('w', cb)
vs.set('abc') # => 'Callback called'
```

Использование объектов-переменных

- Информация содержащаяся в виджете может быть помещена в объект-переменную
- Объект-переменная передается виджету при его создании через именованный параметр:
  - *textvariable* для Button, Checkbutton и Radiobutton - надпись на кнопке
  - *textvariable* для Entry, Label и Spinbox - текст виджета
  - *variable* для OptionMenu и Scale - выбранный элемент и положение движка
  - *listvariable* для Listbox - выбранные элементы
- Пример:

```
root = Tk()
vt = StringVar(value='abc')
vi = IntVar(value=50)
Entry(root, textvariable=vt).pack()
Scale(root, variable=vi).pack()
```

Менеджеры компоновки

- Менеджер компоновки определяет правила размещения виджетов внутри объемлющего их виджета (master widget)
- Ссылка на объемлющий виджет передается как первый позиционный параметр функции, создающей виджет.
  Если этот параметр не задан, виджет будет размещен внутри корневого виджета.
- Модуль tkinter содержит:
  - менеджер компоновки pack
  - менеджер компоновки grid
  - менеджер компоновки place
- Компоновка виджета производится вызовом его метода pack(), grid() или place(). Таким образом виджет сообщает своему мастер-виджету где и каким способом он хочет разместиться.

Менеджер компоновки pack

- Менеджер компоновки pack упаковывает подчиненные виджеты внутри объемлющего виджета в порядке вызова их методов
- Аргумент *side* указывает с какой стороны следует использовать свободное пространство для размещения виджета
- Аргумент *fill* указывает на то, что виджету следует попытаться занять все свободное пространство по оси X, Y или в обоих направлениях
- После размещения место отведенное под виджет считается занятым и процедура повторяется для оставшегося свободного пространства
- Объемлющий виджет стремится сохранить минимальный размер, но так, что бы все внутренние виджеты были видны
- Менеджер компоновки pack вызывается как метод объекта-виджета

Аргументы метода pack()

- side : LEFT, TOP, RIGHT, BOTTOM
край свободного пространства возле которого будет размещен виджет
- fill : X, Y, BOTH, NONE виджет должен заполнить все свободное пространство вдоль указанной оси или в двух направлениях
- anchor : NW, N, NE, E, SE, S, SW, W, CENTER виджет не занимающий все свободное пространство "тяготеет" в указанном направлении
- expand : YES, NO - разрешает максимальное расширение виджета
- padx, pady : дополнительное пространство вокруг виджета
- ipadx, ipady : дополнительное пространство внутри виджета

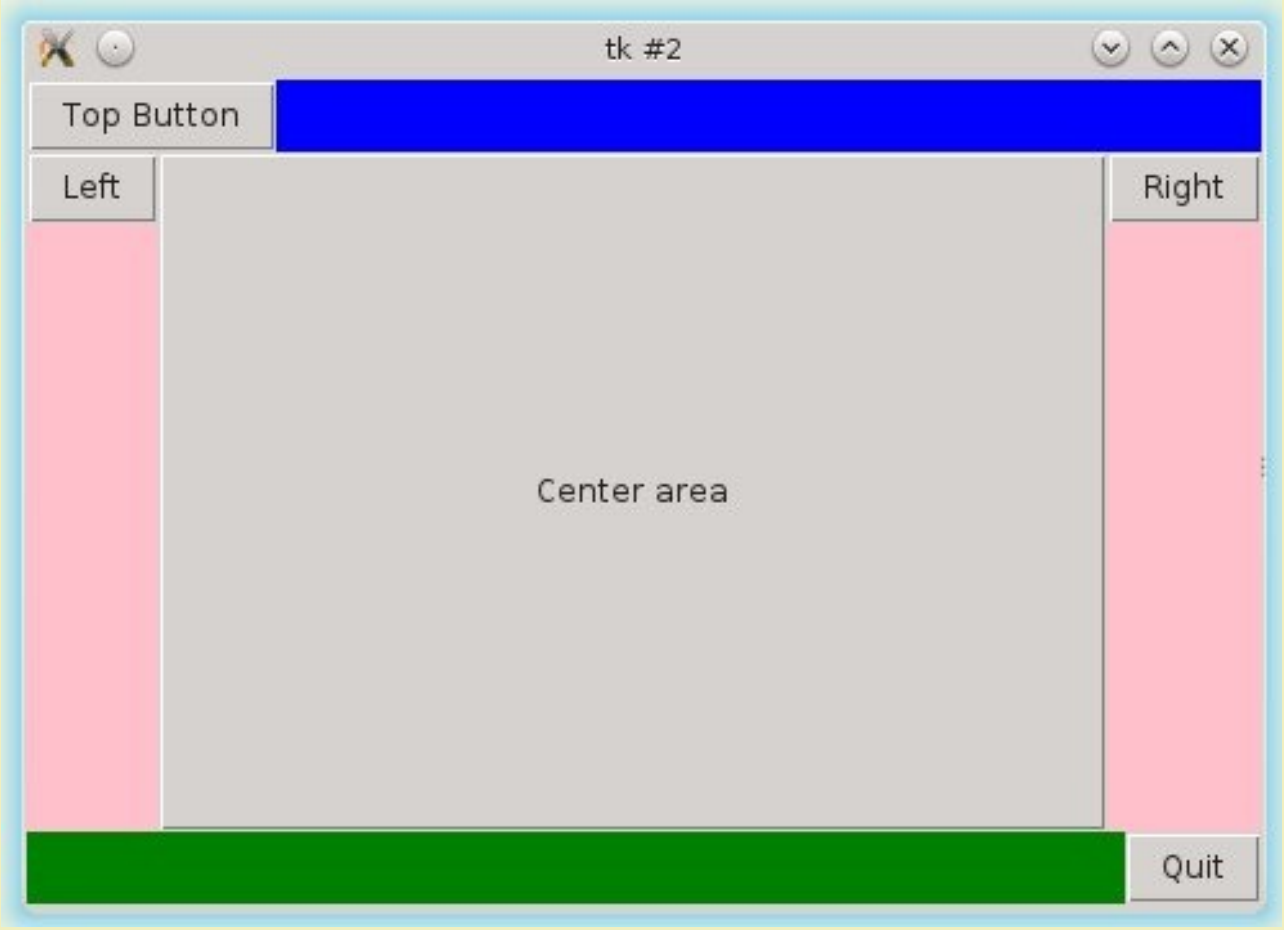
Пример pack компоновки

```
root = Tk()
root.geometry('480x320')

# Три фрейма по вертикали
ftop = Frame(root, bg='blue')
ftop.pack(side=TOP, fill=X)
fcenter = Frame(root, bg='pink')
fcenter.pack(side=TOP, fill=BOTH, expand=YES)
fbottom = Frame(root, bg='green')
fbottom.pack(side=BOTTOM, fill=X)

# Кнопки внутри фреймов
Button(ftop, text='Top Button').pack(side=LEFT)
Button(fcenter, text='Left').pack(side=LEFT, anchor=N)
Button(fcenter, text='Center area').pack(side=LEFT, fill=BOTH, expand=YES)
Button(fcenter, text='Right').pack(side=RIGHT, anchor=N)
Button(fbottom, text='Quit', command=root.destroy).pack(side=RIGHT)

root.mainloop()
```


tk #2
Top Button
Left
Right
Center area
Quit

Менеджер компоновки grid

- Площадь объемлющего виджета (master widget) делится на ячейки имеющие вид таблицы
- Менеджер компоновки grid раскладывает подчиненные виджеты (slave widgets) в ячейки таблицы, идентифицируемые номером столбца и номером строки
- Менеджер компоновки grid вызывается как метод объекта-виджета, аргументы:

```
- row        - номер строки
- column     - номер столбца
- rowspan    - количество строк, которое занимает виджет
- columnspan - количество столбцов, которое занимает виджет
- sticky     - направления, к которым должны стремиться края виджета
```

- Направления задаются символами сторон света и их сочетаниями, например N, E, NS, NSEW.
  По умолчанию виджет стремится к центру.

Расширяемые строки и столбцы

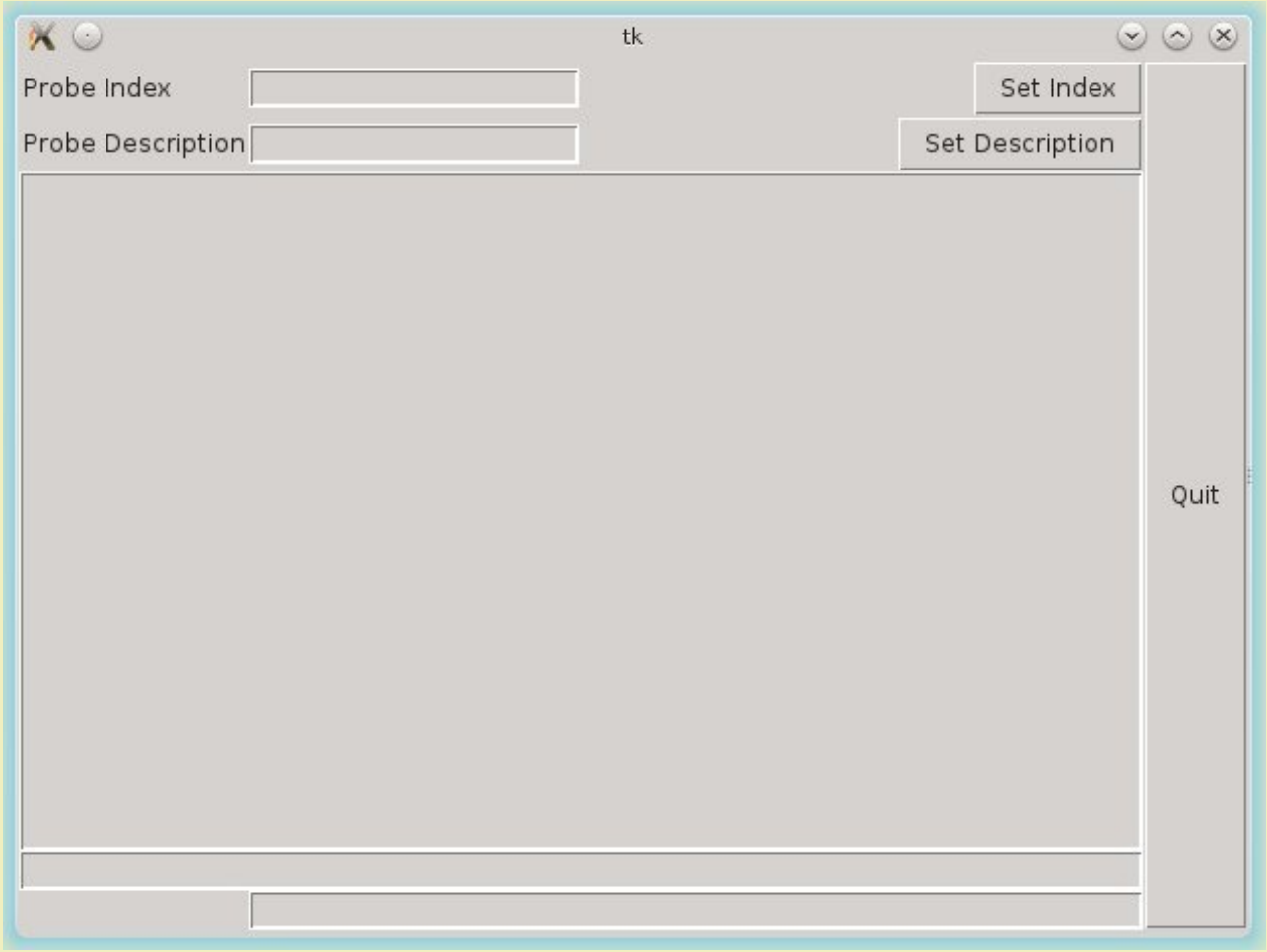
- Строки и столбцы таблицы размещения виджетов могут быть сконфигурированы как расширяемые
- Для конфигурации используются методы *объемлющего виджета*
  - rowconfigure(index, weight)
  - columnconfigure(index, weight)

  которые назначают вес строке или столбцу.

- При изменении размера объемлющего виджета строки и столбцы расширяются или сужаются пропорционально назначенному им весу

Пример grid компоновки

```
root = Tk()
Label(root, text='Probe Index').grid(row=0, column=0, sticky=W)
Label(root, text='Probe Description').grid(row=1, column=0, sticky=W)
Entry(root).grid(row=0, column=1, sticky=W)
Entry(root).grid(row=1, column=1, sticky=W)
Button(root, text='Set Index').grid(row=0, column=2, sticky=E)
Button(root, text='Set Description').grid(row=1, column=2, sticky=E)
Button(root, text='Quit', command=exit).grid(row=0, column=3,
                                             rowspan=5, sticky=NS)
Text(root).grid(row=2, column=0, columnspan=3, sticky=NSEW)
Entry(root).grid(row=3, column=0, columnspan=3, sticky=EW)
Entry(root).grid(row=4, column=1, columnspan=2, sticky=EW)
root.rowconfigure(2, weight=1)
root.columnconfigure(1, weight=1)
root.mainloop()
```


tk
Probe Index
Set Index
Probe Description
Set Description
Quit

Менеджер компоновки place

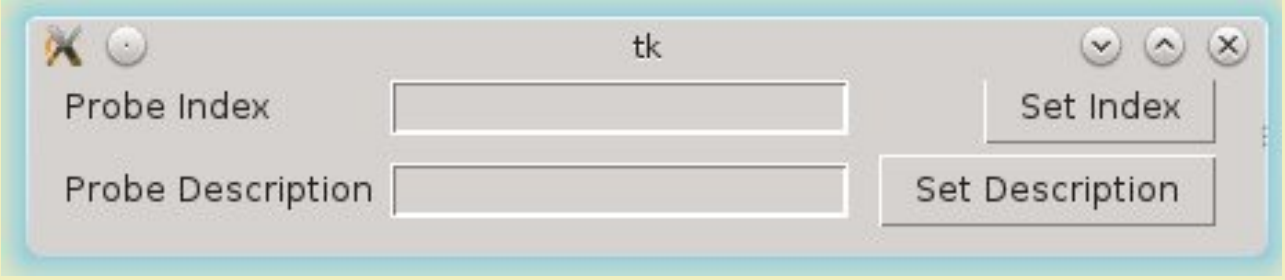
- Менеджер компоновки place() также является методом объекта-виджета
- Метод place() размещает виджет внутри объемлющего виджета в явно указанных координатах
- Именованные аргументы метода place():

```
- x, y           - абсолютные координаты виджета
- relx, rely     - относительные координаты виджета
- width, height  - абсолютные ширина и высота виджета
- relwidth, relheight - относительные ширина и высота виджета
- anchor         - угол или сторона виджета размещенная в координатах,
                   по умолчанию NW (левый-верхний угол)
```

- Относительные координаты и размеры лежат в пределах от 0 до 1.0 и соотносятся с шириной и высотой master-виджета. Метод place() также позволяет указать относительные размеры и координаты соотнесенные с master-виджетом

Пример place компоновки

```
root = Tk()
root.geometry('446x60')
Label(root, text='Probe Index').place(x=10, y=10)
Label(root, text='Probe Description').place(x=10, y=40)
Entry(root).place(x=130, y=10)
Entry(root).place(x=130, y=40)
Button(root, text='Set Index').place(x=430, y=10, anchor=E)
Button(root, text='Set Description').place(x=430, y=40, anchor=E)
root.mainloop()
```



Взаимодействие с библиотекой matplotlib

- График построенный библиотекой matplotlib может быть встроен в canvas-виджет библоитеки tkinter

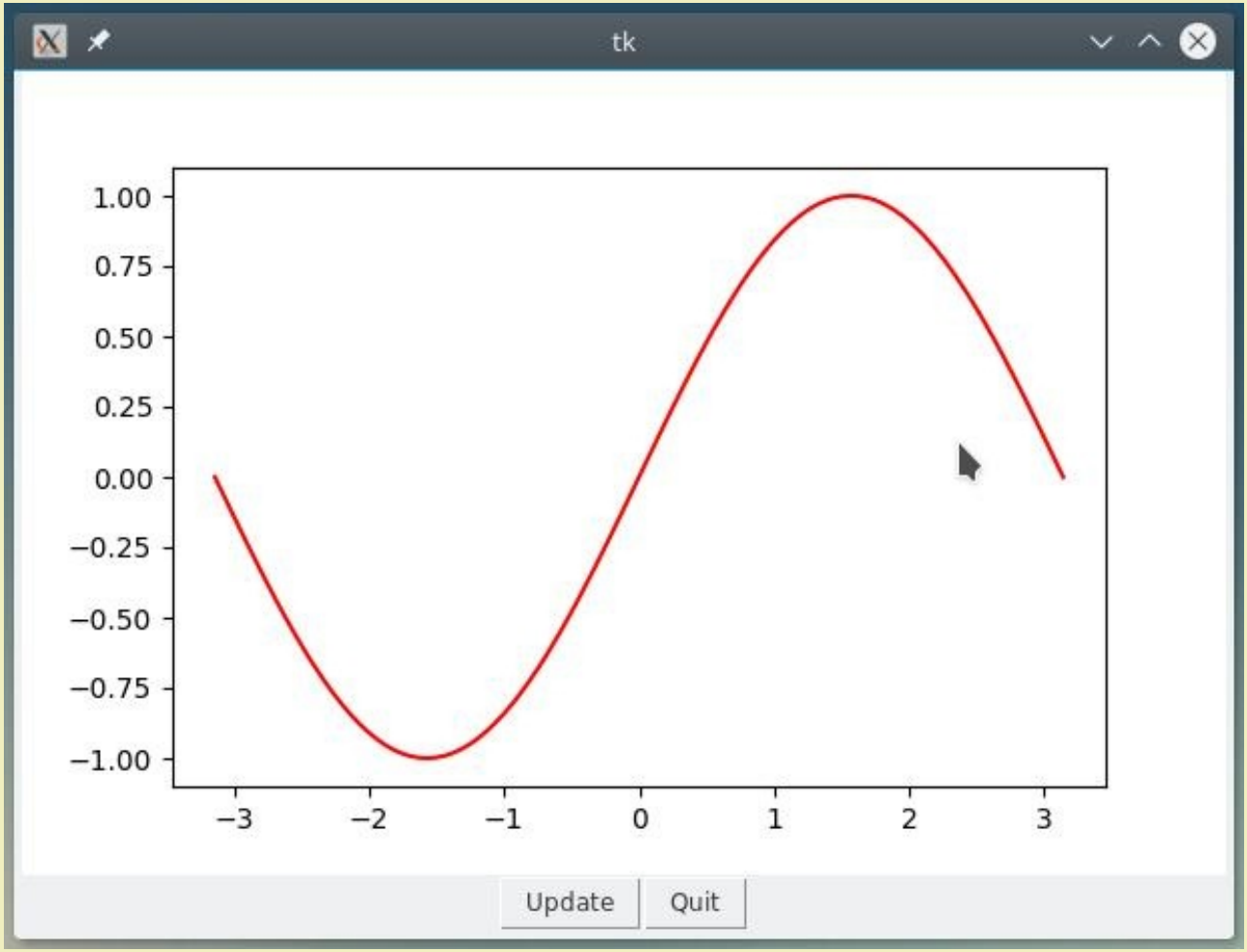
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from tkinter import *

xa = np.linspace(-np.pi, np.pi, 101, endpoint=True)
ya = np.sin(xa)

fig = plt.Figure(figsize=(6, 4))
axes = fig.add_subplot(1, 1, 1)
axes.plot(xa, ya, color='red')

root = Tk()
canvas = FigureCanvasTkAgg(fig, master=root)
canvas._tkcanvas.pack(fill=BOTH, expand=YES)

frame = Frame(root) ; frame.pack()
Button(frame, text='Update', command=update).pack(side=LEFT)
Button(frame, text='Quit', command=root.destroy).pack(side=RIGHT)
root.mainloop()
```


tk
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−3
−2
−1
0
1
2
3
Update
Quit

Окна сообщений

- Окно сообщения это окно содержащее текст и одну или более кнопок, позволяющих окно закрыть
- Окна сообщений реализованы в виде функций субмодуля messagebox
- Каждая функция имеет два аргумента
  - первый аргумент, title - текст для отображения в заголовке окна
  - второй аргумент, message - текст для отображения в окне

```
from tkinter import messagebox
messagebox.showinfo("showinfo", "Info message")
messagebox.showwarning("showwarning", "Warning message")
messagebox.showerror("showerror", "Error message")
result = messagebox.askyesno("ashyesno", "Yes or No")
result = messagebox.askokcancel("ashokcancle", "Is it OK?")
result = messagebox.askretrycancel("askyesnocancel", "Retry/Cancel")
result = messagebox.askyesnocancel("askyesnocancel", "Yes/No/Cancel")
```

Простые диалоги

- Простой диалог это окно с полем ввода и кнопками, позволяющими закрыть окно с подтверждением или отказом от введенной информации
- Простые диалоги реализованы в виде функций субмодуля simpledialog
- Каждая функция имеет два аргумента
  - первый аргумент, title - текст для отображения в заголовке окна
  - второй аргумент, message - текст для отображения в окне
- Вызов простых диалогов

```
from tkinter import simpledialog
result = simpledialog.askstring("askstring", "Enter string")
result = simpledialog.askinteger("askinteger", "Enter integer number")
result = simpledialog.askfloat("askfloat", "Enter float number")
```

Диалоги выбора файла

- Субмодуль filedialog содержит окна диалогов выбора файлов, команда импорта:

```
from tkinter import filedialog
```

- Функции, открывающие диалог выбора файла

```
- askopenfilename()   - выбор файла для открытия
- askopenfilenames()  - выбор нескольких файлов для открытия
- asksaveasfilename() - выбор файла для сохранения
- askdirectory()      - выбор директории
```

- Варианты тех же функций открывающие выбранные файлы:

```
- askopenfile()   - открыть выбранный файл для чтения
- askopenfiles()  - открыть нескольких выбранных файлов
                    для чтения
- asksaveasfile() - открыть выбранный файл для записи
```

Диалог выбора цвета

- Диалог выбора цвета находится в субмодуле colorchooser
- Инструкция импорта:

```
from tkinter import colorchooser
```

- Вызов диалога:

```
color = colorchooser.askcolor()
```

- Функция возвращает кортеж из двух элементов.
  - первый элемент это представление цвета в виде кортежа из трех чисел (R, G, B),
  - второй элемент это представление цвета в виде строки #rrggbb

Виджет редактора текста

- Виджет редактора текста представлен классом ScrolledText и находится в субмодуле scrolledtext. Инструкция импорта:

```
from tkinter import scrolledtext
```

- Виджет ScrolledText это сочетание виджета Frame с находящимися внутри него виджетами Text и Scrollbar. Scrollbar привязан к тексту и осуществляет его прокрутку
- Виджет ScrolledText имеет те же параметры и атрибуты, что и входящий в него виджет Text

Субмодуль ttk (Themed Tk)

- Субмодуль ttk содержит 17 виджетов, частично повторяющих виджеты из базового модуля tkinter.
- Для виджетов модуля ttk может быть задан общий стиль изображения в виде объекта класса ttk.Style

```
from tkinter import ttk

style = ttk.Style()
style.configure('TButton', foreground="red", background="yellow")
style.configure('TLabel', foreground="green")

ttk.Label(root, text='Styled Application').pack(padx=4, pady=8)
ttk.Button(root, text='Quit', command=root.destroy).pack()

root.mainloop()
```

Ttk: новые виджеты

- Новые виджеты в составе ttk:
  - Combobox - выпадающий список выбора
  - Notebook - фрейм с несколькими страницами
  - Progressbar - полоса с заполнением
  - Treeview - виджет для просмотра дерева
  - Separator - разделительная линия
  - Sizegrip - "уголок" для изменения размеров окна

Субмодуль tix (Tk Interface Extension)

- Tix это самостоятельный проект - надстройка над библиотекой Tk. Tix устанавливается как отдельный пакет.
- Виджеты субмодуля tix

```
- Balloon - всплывающее окно с поясняющим текстом
- ButtonBox - группа кнопок
- StdButtonBox - группа кнопок стандартных диалогов
- ComboBox - выпадающий список выбора
- Control - аналог Spinbox
- LabelEntry - сочетание Label и Entry
- LabelFrame - сочетание Label и Frame
- Meter - аналог Progressbar
- OptionMenu - выпадающее меню
- PopupMenu - меню в новом окне
- Select - группирующий виджет для radio/checkbox button
- TList - список выбора, оформленный как таблица
- Виджеты для выбора файлов
- Виджеты для работы с деревьями
- Менеджер компоновки Form
- а также ряд виджетов, повторяющих уже имеющиеся в tkinter и ttk
```

Другие библиотеки виджетов

- PyGTK - интерфейс к библиотеке GTK+
  - GTK+ является основой оконной системы Gnome
- PyQt - интерфейс к библиотеке Qt
  - Qt является основой оконной системы KDE
  - с 2018 года поддержка Питона в библиотеке Qt стала официальной
    - Модуль PySide для версии Qt4
    - Модуль PySide2 для версии Qt5
- wxPython - интерфейс к библиотеке wxWidgets
  - библиотека спроектирована как мультиплатформенная с возможностью разработки приложений визуально неотличимых от оригинальных
- Все три библиотеки работают в Linux, Windows и MacOS/X, они не входят в состав Питона и должны устанавливаться отдельно

Литература к лекции

1. https://docs.python.org/3/library/tk.html

2. John E. Grayson "Python and Tkinter Programming", Manning Publications Co., 2000, ISBN 1-884777-81-3

3. Burkhard A. Meier "Python GUI Programming Cookbook", Second Edition, Packt Publishing, 2017, ISBN 978-1-78712-945-0