МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

УТВЕРЖДАЮ
Зав.кафедрой,
к. ф.-м. н., доцент
_____ А. С. Иванов

ОТЧЕТ О ПРАКТИКЕ

студента 3 курса 351 группы факультета КНиИТ
Голикова Артема Олеговича

вид практики: учебная
кафедра: математической кибернетики и компьютерных наук
курс: 3
семестр: 5
продолжительность: 2 нед., с 16.12.2019 г. по 29.12.2019 г.

Руководитель практики от университета,

к. ф.-м. н., доцент _____ А. С. Иванова

Руководитель практики от организации (учреждения, предприятия),

к. ф.-м. н., доцент _____ А. С. Иванова

Тема практики:«Графика в языке Python»

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Python является широко используемым, высокоуровневым языком программирования, который был назван в честь знаменитого британского комедийного телешоу «Летающий цирк Монти Пайтона». Язык Python простой по своей структуре, и в то же время невероятно гибкий и мощный. Учитывая, что код Python легко читаемый и без излишней строгости в синтаксисе, многие считают, что он является лучшим вводным языком программирования.

Python – это интерпретируемый, интерактивный, объектно–ориентированный язык программирования. Он включает в себя модули, исключения, динамическую типизацию, высокоуровневые динамические типы данных и классы. Python сочетает в себе отличную производительность с понятным синтаксисом. В нем реализованы интерфейсы ко многим системным вызовам и библиотекам, а также различным оконным системам и он расширяем с помощью C и C++. Python используется как язык расширения для приложений, которым нужен программный интерфейс. И наконец, Python — это кроссплатформенный язык: он работает на многих версиях Unix, на Mac и на компьютерах под управлением MS-DOS, Windows. [1]

Главной целью практической работы является знакомство с графикой в языке Python на примере библиотеки Tkinter. Будут рассмотрены базовые понятия, элементы и конструкции графического интерфейса. Закрепить имеющиеся знания и приобрести новые. Цель ознакомительной практики – приобретение навыков методической работы.

Поставлены следующие задачи:

— ознакомится со средствами языка Python и библиотеки Tkinter;
— рассмотреть возможности Tkinter на примерах;
— разработать задачи и тест по заданной теме.

1 Теоретический материал

1.1 Графический интерфейс пользователя. Модуль Tkinter

Под графическим интерфейсом пользователя (GUI) подразумеваются все те окна, кнопки, текстовые поля для ввода, скроллеры, списки, радиокнопки, флажки и др., которые вы видите на экране, открывая то или иное приложение. Через них вы взаимодействуете с программой и управляете ею.

Приложения с графическим интерфейсом пользователя событийно-ориентированные. То есть та или иная часть программного кода начинает выполняться лишь тогда, когда случается то или иное событие.

Событийно-ориентированное программирование базируется на объектно-ориентированном и структурном. Даже если мы не создаем собственных классов и объектов, то все-равно ими пользуемся. Все виджеты – объекты, порожденные встроенными классами. [2]

События бывают разными. Сработал временной фактор, кто-то кликнул мышкой или нажал Enter, начал вводить текст, переключил радиокнопки, прокрутил страницу вниз и т. д. Когда случается что-то подобное, то, если был создан соответствующий обработчик, происходит срабатывание определенной части программы, что приводит к какому-либо результату.

Tkinter – это стандартная библиотека GUI для Python. Python в сочетании с Tkinter обеспечивает быстрый и простой способ создания приложений с графическим интерфейсом. Tkinter предоставляет мощный объектно-ориентированный интерфейс для инструментария Tk GUI.

Первым делом необходимо импортировать модуль Tkinter:

```
from tkinter import *
```

Главным элементом GUI Tkinter является окно. Окнами называют контейнеры, в которых находятся все GUI элементы.

Таким образом минимальное приложение на Tkinter будет таким:

```
from tkinter import *
window = Tk()
window.mainloop()
```

`window.mainloop()` указывает Python, что нужно запустить цикл событий Tkinter. Данный метод требуется для событий вроде нажатий на клавиши

или кнопки, он также блокирует запуск любого кода, что следует после, пока окно, на котором оно было вызвано, не будет закрыто.

Рисунок 1 – Пустое окно приложения

1.2 Методы окон

— `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 рамки).

— `iconofy / deiconofy` – свернуть / развернуть окно.

— `withdraw` – "спрятать"(сделать невидимым) окно. Для того чтобы снова показать его, надо использовать метод deiconify.

— `minsize / 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).

— `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 "ширина x высота + x + y"(пример: geometry("600x400+40+80") – поместить окно в точку

с координатами 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_DELETE_WINDOW` (удаление окна).

— `raise / 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.

```
from tkinter import *
def window_deleted():
    print ('Окно закрыто')
    window.quit() # явное указание на выход из программы
window = Tk()        # инициализация окна
window.title('Пример приложения') #заголовок окна
window.geometry('640x480') # ширина=640, высота=480
window.protocol('WM_DELETE_WINDOW', window_deleted) # обработчик
   закрытия окна
window.resizable(True, False) # размер окна может быть изменён
   только по горизонтали
window.mainloop()
```

1.3 Основные виджеты

GUI элементы, к числу которых относятся текстовые боксы, ярлыки и кнопки, называются виджетами и помещаются внутри окон. Все виджеты в Tkinter обладают некоторыми общими свойствами. Они создаются вызовом конструктора соответствующего класса. Первый аргумент (как правило

неименованный, но можно использовать имя master) это родительский виджет, в который будет упакован (помещён) наш виджет. Родительский виджет можно не указывать, в таком случае будет использовано главное окно приложения. Далее следуют именованные аргументы, конфигурирующие виджет. Это может быть используемый шрифт (font=...), цвет виджета (bg=...), команда, выполняющаяся при активации виджета (command=...) и т.д.
Button – самая обыкновенная кнопка

```
from tkinter import *
window = Tk()
window.geometry('400x200')
button1 = Button(window, text = 'Click!', width = 25, height = 10,
   bg = 'aquamarine', fg = 'black', font = 'arial 14')
button1.pack()
window.mainloop()
```

Создаётся кнопка, при этом мы указываем её свойства (начинать нужно с указания окна, в примере – window). Здесь перечислены некоторые из них:

- `text` – текст отображаемый на кнопке (в примере – Click!);
- `width, height` – соответственно, ширина и длина кнопки;
- `bg` – цвет кнопки (в примере цвет – аквамарин);
- `fg` – цвет текста на кнопке (в примере цвет – черный);
- `font` – шрифт и его размер (в примере – arial, размер - 14).

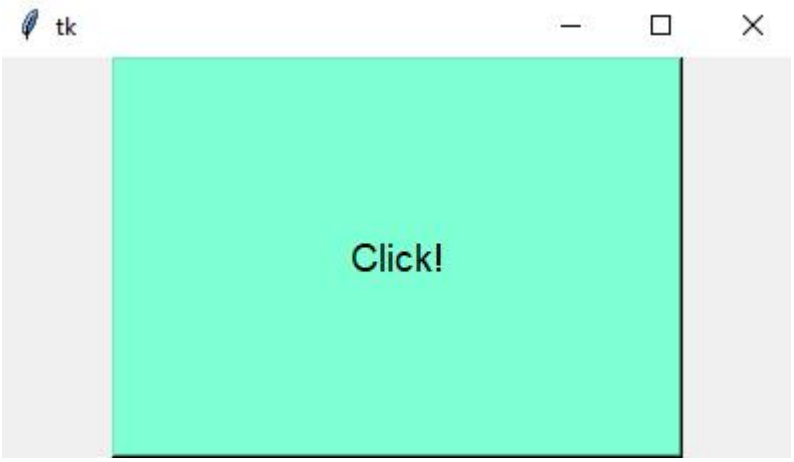

Рисунок 2 – Пример кнопки

Label – это виджет, предназначенный для отображения какой-либо над-

писи без возможности редактирования пользователем. Имеет те же свойства, что и перечисленные свойства кнопки. *Entry* – это виджет, позволяющий пользователю ввести одну строку текста.

```
from tkinter import *
window = Tk()
window.geometry('400x200')
entry1 = Entry(window, bd = 5, width = 15, show = '#')
entry1.pack()
window.mainloop()
```

— `bd` – ширина бордюра элемента.

— `width` – задает длину элемента в знакоместах.

— `show` – отображает заданный символ.

Рисунок 3 – Пример виджета Entry

Text – это виджет, который позволяет пользователю ввести любое количество текста. Имеет дополнительное свойство wrap, отвечающее за перенос (чтобы, например, переносить по словам, нужно использовать значение word).

```
from tkinter import *
window = Tk()
window.geometry('400x200')
text1 = Text(window, height = 7, width = 20, font = 'Arial 14',
   wrap = WORD)
text1.insert('1.0', '1st\n2nd\n3rd\n4th')
text1.delete('2.0','3.0')
```

```
text1.get('1.0', END)
text1.pack()
window.mainloop()
```

Методы insert, delete и get добавляют, удаляют или извлекают текст. Аргументы начала и конца места вставки/удаления в виде 'x.y', где x – это строка, а y – столбец.

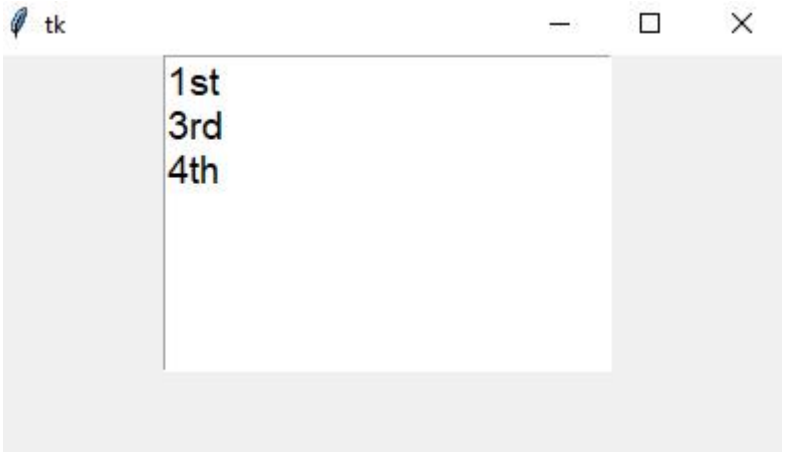

Рисунок 4 – Пример виджета Text

Listbox – это виджет, который представляет собой список, из элементов которого пользователь может выбирать один или несколько пунктов. Имеет дополнительное свойство selectmode, которое, при значении SINGLE, позволяет пользователю выбрать только один элемент списка, а при значении EXTENDED – любое количество.

```
from tkinter import *
window = Tk()
window.geometry('400x200')
listbox1 = Listbox(window, height = 5, width = 15, selectmode =
   EXTENDED)
listbox2 = Listbox(window, height = 5, width = 15, selectmode =
   SINGLE)
list1 = ["Саратов","СГУ","КНиИТ","ПИ"]
list2 = ["151","251","351","451"]
for i in list1:
    listbox1.insert(END,i)
for i in list2:
```

```
    listbox2.insert(END,i)
listbox1.pack()
listbox2.pack()
window.mainloop()
```

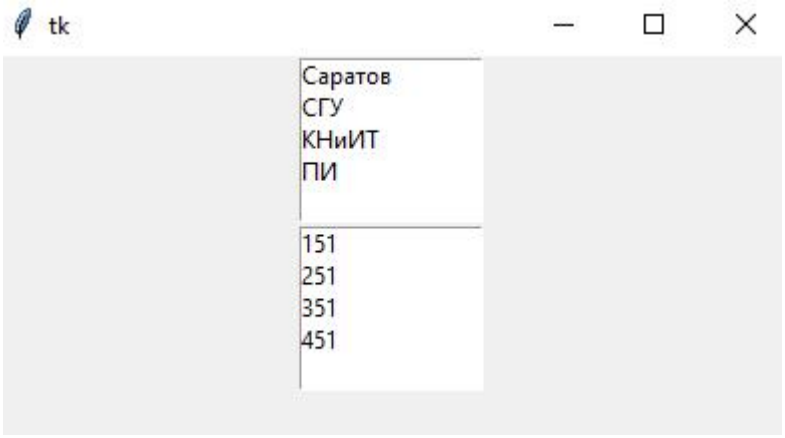

Рисунок 5 – Пример виджета ListBox

Frame (рамка) – предназначен для организации виджетов внутри окна.

```
from tkinter import *
window = Tk()
window.geometry('400x200')
frame1 = Frame(window, bg = 'green', bd = 3)
frame2 = Frame(window, bg = 'red', bd = 3)
button1 = Button(frame1, text = 'Ответ верный')
button2 = Button(frame2, text = 'Ответ неверный')
frame1.pack()
frame2.pack()
button1.pack()
button2.pack()
window.mainloop()
```

Рисунок 6 – Пример виджета Frame

Checkbutton – это виджет, который имеет два состояния: «включен» и «выключен». Состояние «включен» визуально символизируется соответствующей отметкой.

```
from tkinter import *
window = Tk()
window.geometry('400x200')
check1 = Checkbutton(window, text = '1 пункт', variable = 1,
   onvalue = 1, offvalue = 0)
check2 = Checkbutton(window, text = '2 пункт', variable = 0)
check1.pack()
check2.pack()
window.mainloop()
```

`variable` – свойство, отвечающее за прикрепление к виджету переменной. onvalue, offvalue – свойства, которые присваивают прикреплённой к виджету переменной значение, которое зависит от состояния (onvalue – при выбранном пункте, offvalue – при невыбранном пункте).

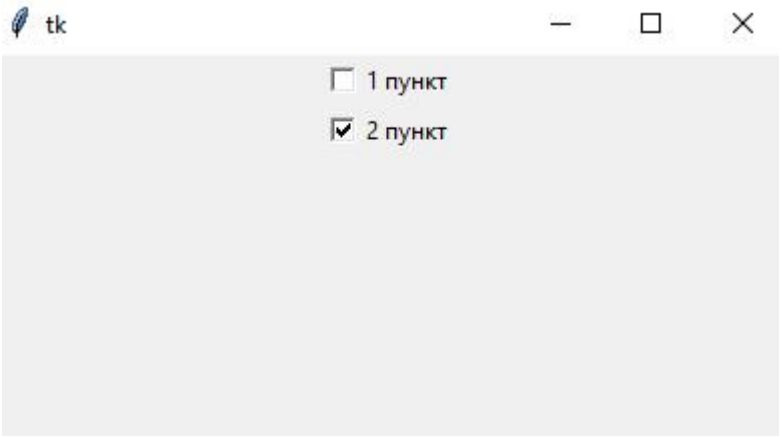

Рисунок 7 – Пример виджета CheckButton

Radiobutton – виджет, который выполняет функцию, схожую с функцией Checkbutton. Разница в том, что в в Radiobutton пользователь может выбрать лишь один из пунктов.

```
from tkinter import *
window = Tk()
window.geometry('400x200')
choice = IntVar()
rbutton1 = Radiobutton(window, text='1 пункт', variable = choice,
   value = 1)
rbutton2 = Radiobutton(window, text='2 пункт', variable = choice,
   value = 2)
rbutton3 = Radiobutton(window, text='3 пункт', variable = choice,
   value = 3)
rbutton1.pack()
rbutton2.pack()
rbutton3.pack()
window.mainloop()
```

Объявлена переменная choice типа int. В зависимости от того, какой пункт выбран, она меняет свое значение. Если присвоить этой переменной какое-либо значение, поменяется и выбранный виджет.

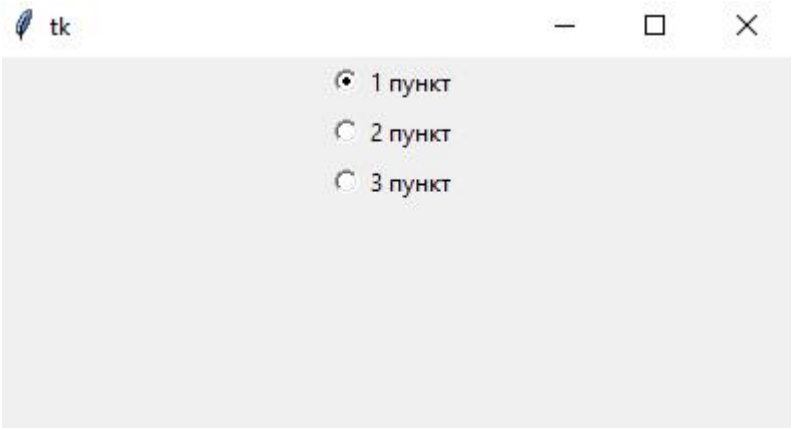

Рисунок 8 – Пример виджета RadioButton

Scale (шкала) – это виджет, позволяющий выбрать какое-либо значение из заданного диапазона.

```
from tkinter import *
window = Tk()
```

```
window.geometry('400x200')
def click(window):
    a = scale1.get()
    print ("Оценка: ", a)
scale1 = Scale(window, orient = HORIZONTAL, length = 300, from_ =
   0, to = 10, tickinterval = 1, resolution = 5)
button1 = Button(window, text = "Оценить")
scale1.pack()
button1.pack()
button1.bind("<Button-1>", click)
window.mainloop()
```

Программа выводит в консоль выбранное значение шкалы.

- `orient` – как расположена шкала на окне. Возможные значения: HORIZONTAL, VERTICAL (горизонтально, вертикально).
- `length` – длина шкалы.
- `from_` – с какого значения начинается шкала.
- `to` – каким значением заканчивается шкала.
- `tickinterval` – интервал, через который отображаются метки шкалы.
- `resolution` – шаг передвижения (минимальная длина, на которую можно передвинуть движок).

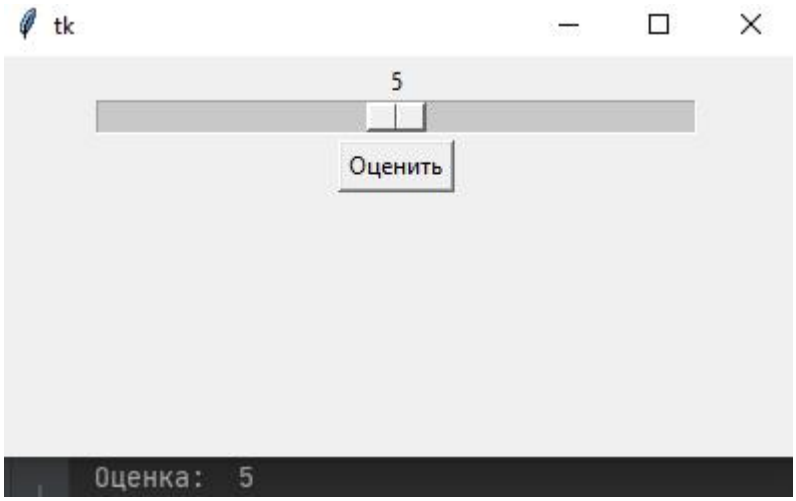

Рисунок 9 – Пример виджета Scale

Scrollbar – виджет даёт возможность пользователю "прокрутить"другой виджет (например текстовое поле) и часто бывает полезен. Использование этого виджета достаточно нетривиально. Необходимо сделать две привязки:

command полосы прокрутки привязываем к методу xview/yview виджета, а xscrollcommand/yscrollcommand виджета привязываем к методу set полосы прокрутки.

```
from tkinter import *
window = Tk()
window.geometry('400x200')
text1 = Text(window)
scrollbar1 = Scrollbar(window, orient = "vertical", command =
   text1.yview)
text1.configure(yscrollcommand = scrollbar1.set)
scrollbar1.pack(side = "right", fill = "y")
text1.pack(side = "left", fill = "both", expand=True)
window.mainloop()
```

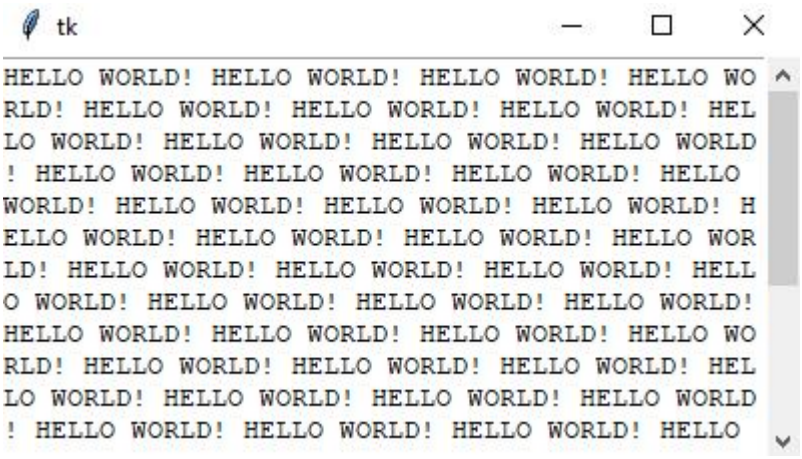

Рисунок 10 – Пример виджета ScrollBar

1.4 Упаковщики

Упаковщик (менеджер геометрии, менеджер расположения) это специальный механизм, который размещает (упаковывает) виджеты на окне. В Tkinter есть три упаковщика: pack, place, grid. Стоит обратить внимание на то, что в одном виджете можно использовать только один тип упаковки, при смешивании разных типов упаковки программа, скорее всего, не будет работать.

pack() – является самым интеллектуальным (и самым непредсказуемым). При использовании этого упаковщика с помощью свойства side нужно указать к какой стороне родительского виджета он должен примыкать. Как пра-

вило этот упаковщик используют для размещения виджетов друг за другом (слева направо или сверху вниз).

```
from tkinter import *
window = Tk()
window.geometry('400x200')
button1 = Button(text = "Кнопка 1")
button2 = Button(text = "Кнопка 2")
button3 = Button(text = "Кнопка 3")
button4 = Button(text = "Кнопка 4")
button5 = Button(text = "Кнопка 5")
button1.pack(side = 'left')
button2.pack(side = 'top')
button3.pack(side = 'left')
button4.pack(side = 'bottom')
button5.pack(side = 'right')
window.mainloop()
```

Рисунок 11 – Пример использования pack()

Для создания сложной структуры с использованием этого упаковщика обычно используют Frame, вложенные друг в друга.

При применении этого упаковщика можно указать следующие аргументы:

— `side` (lef/right/top/bottom) – к какой стороне должен примыкать размещаемый виджет;

— `fill` (None/x/y/both) – необходимо ли расширять пространство предоставляемое виджету;

— `expand` (True/False) – необходимо ли расширять сам виджет, чтобы он занял всё предоставляемое ему пространство;

— `in_` – явное указание в какой родительский виджет должен быть помещён; Кроме основной функции у виджетов есть дополнительные методы для работы с упаковщиками;

— `pack_slaves` – возвращает список всех дочерних упакованных 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 своих потомков;

— `pack_forget` – удаляет виджет и всю информацию о его расположении из упаковщика. Позднее этот виджет может быть снова размещён. [3]

grid() – упаковщик представляет собой таблицу с ячейками, в которые помещаются виджеты.

```
from tkinter import *
window = Tk()
text1 = Text(wrap = NONE)
vert_scrollbar = Scrollbar(orient = 'vert', command = text1.yview)
text1['yscrollcommand'] = vert_scrollbar.set
hor_scrollbar = Scrollbar(orient = 'hor', command = text1.xview)
text1['xscrollcommand'] = hor_scrollbar.set
# размещаем виджеты
text1.grid(row = 0, column = 0, sticky = 'nsew')
vert_scrollbar.grid(row = 0, column = 1, sticky = 'ns')
hor_scrollbar.grid(row = 1, column=0, sticky = 'ew')
# конфигурируем упаковщик, чтобы текстовый виджет расширялся
window.rowconfigure(0, weight = 1)
window.columnconfigure(0, weight = 1)
window.mainloop()
```

При применении этого упаковщика можно указать следующие аргументы:

— `row` – номер строки, в который помещаем виджет;

— `rowspan` – сколько строк занимает виджет;

— `column` – номер столбца, в который помещаем виджет;
— `columnspan` – сколько столбцов занимает виджет;
— `padx / pady` – размер внешней границы (бордюра) по горизонтали и вертикали;
— `ipadx / ipady` – размер внутренней границы (бордюра) 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 в какой родительский виджет должен быть помещён;

Для каждого виджета указываем, в какой он находится строке, и в каком столбце. Если нужно, указываем, сколько ячеек он занимает (если, например, нам нужно разместить три виджета под одним, необходимо "растянуть"верхний на три ячейки).

Дополнительные функции

— `grid_slaves` – возвращает список всех дочерних упакованных виджетов.
— `grid_info` – возвращает информацию о конфигурации упаковки.
— `grid_propagate` – включает/отключает распространении информации о геометрии дочерних виджетов. По умолчанию виджет изменяет свой размер в соответствии с размером своих потомков.
— `grid_forget` – удаляет виджет и всю информацию о его расположении из упаковщика. Позднее этот виджет может быть снова размещён.
— `grid_remove` – удаляет виджет из-под управления упаковщиком, но сохраняет информацию об упаковке. Этот метод удобно использовать для временного удаления виджета.
— `grid_bbox` – возвращает координаты (в пикселях) указанных столбцов и строк.
— `grid_location` – принимает два аргумента: x и y (в пикселях). Возвра-

щает 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_columnconfigure и grid_rowconfigure` – важные функции для конфигурирования упаковщика. Методы 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. Строка/столбец с "весом"равным 2 будет расширяться вдвое быстрее, чем с весом 1.

  – `uniform` – объединение строк/столбцов в группы. Строки/столбцы имеющие одинаковый параметр uniform будут расширяться строго в соответствии со своим весом.

  – `pad` – размер бордюра. Указывает, сколько пространства будет добавлено к самому большому виджету в строке/столбце. [4]

Рисунок 12 – Пример использования grid()

place() – представляет собой простой упаковщик, позволяющий размещать виджет в фиксированном месте с фиксированным размером. Также он позволяет указывать координаты размещения в относительных единицах для

реализации "резинового"размещения. При использовании этого упаковщика, нам необходимо указывать координаты каждого виджета.

Основными параметрами являются:

— `anchor` (якорь) – определяет часть виджета, для которой задаются координаты. Принимает значения N, NE, E, SE, SW, W, NW или CENTER. По умолчанию NW (верхний левый угол);

— `bordermode` ("inside "outside "ignore") – определяет в какой степени будут учитываться границы при размещении виджета;

— `in_` – явное указание в какой родительский виджет должен быть помещён;

— `relwidth, relheight` (относительные ширина и высота) – определяют размер виджета в долях его родителя;

— `relx, rely` – определяют относительную позицию в родительском виджете. Координата (0; 0) – у левого верхнего угла, (1; 1) – у правого нижнего;

— `width, height` – абсолютный размер виджета в пикселях. Значения по умолчанию (когда данные опции опущены) приравниваются к естественному размеру виджета, то есть к тому, который определяется при его создании и конфигурировании;

— `x, y` – абсолютная позиция в пикселях. Значения по умолчанию приравниваются к нулю.

```
from tkinter import *
window = Tk()
button1 = Button(text = 'Абсолютное', bg = 'red').place(x = 75, y
   = 20)
button2 = Button(text = 'Относительное', bg = 'green').place(relx
   = 0.3, rely = 0.5)
window.mainloop()
```

На примере видна разница между относительным и абсолютным позиционированием при растягивании окна.

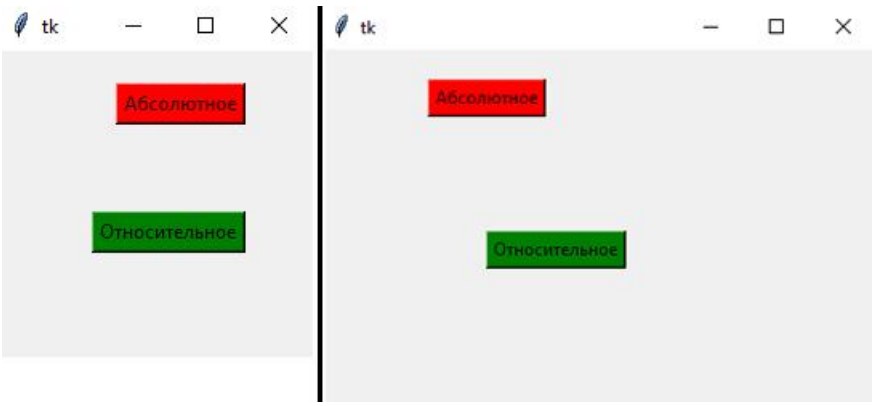

Рисунок 13 – Пример использования place()

1.5 Привязка событий

command

Для большинства виджетов, реагирующих на действие пользователя, активацию виджета (например нажатие кнопки) можно привязать с использованием опции command. К таким виджетам относятся: Button, Checkbutton, Radiobutton, Spinbox, Scrollbar, Scale.

```
button = Button(command=callback)
```

bind Метод привязывает событие к какому-либо действию (нажатие кнопки мыши, нажатие клавиши на клавиатуре и т.д.). bind принимает три аргумента:

— название события;

— функцию, которая будет вызвана при наступлении события;

— третий аргумент (необязательный) – строка "+ означает, что эта привязка добавляется к уже существующим. Если третий аргумент опущен или равен пустой строке - привязка замещает все другие привязки данного события к виджету.

Метод bind возвращает идентификатор привязки, который может быть использован в функции unbind. Стоит обратить внимание на то, что если bind привязан к окну верхнего уровня, то Tkinter будет обрабатывать события всех виджетов этого окна. Функция, которая вызывается при наступлении события, должна принимать один аргумент. Это объект класса Event, в котором описано наступившее событие. Объект класса Event имеет следующие атрибуты (в скобках указаны события, для которых этот атрибут установлен):

— `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;
— `x_root и y_root` – координаты мыши на экране (ButtonPress, ButtonRelease, KeyPress, KeyRelease, Motion);
— `char` – набранный на клавиатуре символ (KeyPress, KeyRelease);
— `keysym` – символ на клавиатуре(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).

Эта функция может возвращать строки "continue"и "break". Если функция возвращает "continue"то Tkinter продолжит обработку других привязок этого события, если "break"– обработка этого события прекращается. Если функция ничего не возвращает (если возвращает None), то обработка событий продолжается (т.е. это эквивалентно возвращению "continue"). Пример, показывающий разницу между char, keycode, keysym, keysymnum:

```
from tkinter import *
def show_key(event):
    text = ', '.join(map(str,[event.keysym, event.char,
       event.keysym_num, event.keycode]))
    label.config(text = text)
window = Tk()
label = Label(window, text = "Нажмите любую клавишу")
label.pack()
window.bind("<KeyPress>", show_key)
window.mainloop()
```

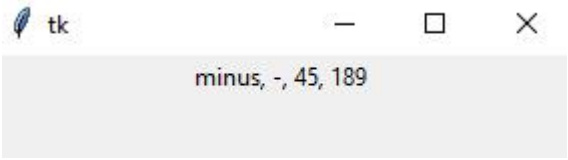

Рисунок 14 – Результат при нажатии клавиши минуса

Есть три формы названия событий. Самый простой случай это символ ASCII. Так описываются события нажатия клавиш на клавиатуре:

```
widget.bind("z", callback)
```

callback вызывается каждый раз, когда будет нажата клавиша "z". Второй способ длиннее, но позволяет описать больше событий. Он имеет следующий синтаксис:

```
<modifier-modifier-type-detail>
```

Название события заключено в угловые скобки. Внутри имеются ноль или более модификаторов, тип события и дополнительная информация (номер нажатой клавиши мыши или символ клавиатуры). Поля разделяются дефисом или пробелом. Пример (привязываем одновременное нажатие Ctrl+Shift+q):

```
widget.bind("<Control-Shift-q>", callback)
```

Третий способ позволяет привязывать виртуальные события – события, которые генерируются самим приложением. Такие события можно создавать самим, а потом привязывать их. Имена таких событий помещаются в двойные угловые скобки: «Paste». Есть некоторое количество уже определённых виртуальных событий.

Список модификаторов:

- *Return* – Enter
- *Escape* – Esc
- *Control* – Ctrl
- *Alt*
- *Shift*
- *Lock*
- *Extended*
- *Prior* – PgUp

— $Next$ – PgDown
— $Button1, B1$ – нажата первая (левая) кнопка мыши
— $Button2, B2$ – вторая (средняя) кнопка мыши
— $Button3, B3$ – третья (правая)
— $Button4, B4$ – четвёртая
— $Button5, B5$ – пятая
— $Mod1, M1, Command$
— $Mod2, M2, Option$
— $Mod3, M3$
— $Mod4, M4$
— $Mod5, M5$
— $Meta, M$
— $Double$ – двойной щелчок мыши (например, <Double-Button-1>)
— $Triple$ – тройной
— $Quadruple$ – четверной

Типы событий:

— $Activate, Deactivate$ – активизация/деактивация окна
— $MouseWheel$ – прокрутка колесом мыши
— $KeyPress, KeyRelease$ – нажатие и отпускание клавиши на клавиатуре
— $ButtonPress, ButtonRelease, Motion$ – нажатие, отпускание клавиши мыши, движение мышью
— $Configure$ – изменение положения или размера окна
— $Map, Unmap$ – показывание или сокрытие окна (например, в случае сворачивания/разворачивания окна пользователем)
— $Visibility$ – изменение видимости окна
— $Expose$ – событие генерируется, когда необходимо всё окно или его часть перерисовать
— $Destroy$ – закрытие окна
— $FocusIn, FocusOut$ – получение или потеря фокуса
— $Enter, Leave$ – Enter генерируется когда курсор мыши "входит"в окно, Leave – когда "уходит"из окна
— $Property$ – свойство X, принадлежащее этому окну, изменяется или удаляется

```
from tkinter import *
```

```
window = Tk()
window.geometry('400x200')
def leftclick(event):
    print ('Вы нажали левую кнопку мыши')
def rightclick(event):
    print ('Вы нажали правую кнопку мыши')
def keypress(event):
    print ('Вы нажали ', event.keysym)
button1=Button(window, text = 'Нажми')
button1.pack()
button1.bind('<Button-1>', leftclick)
button1.bind('<Button-3>', rightclick)
window.bind('<KeyPress>', keypress)
window.mainloop()
```

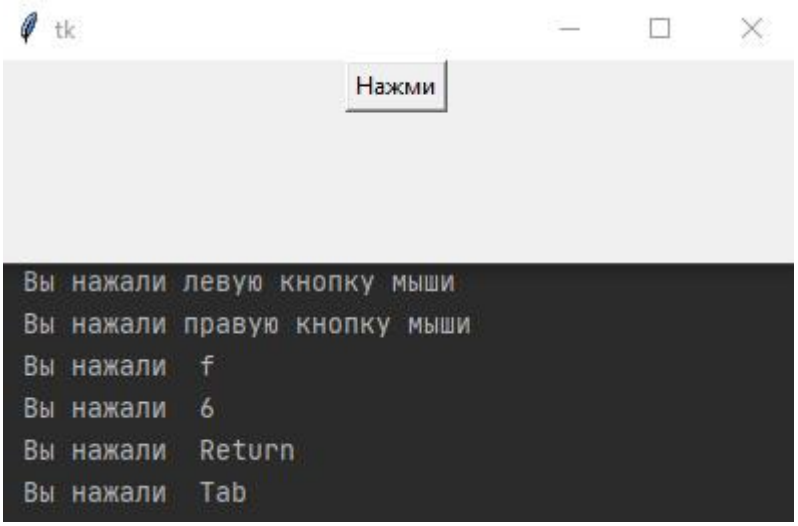

Рисунок 15 – Пример использования событий

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.

— `bindtags` – позволяет изменить порядок обработки привязок. По умол-

чанию порядок следующий: виджет, класс, окно, all.

— `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`.

1.6 Изображения

Для работы с изображениями в Tkinter имеется два класса BitmapImage и PhotoImage. BitmapImage представляет собой простое двухцветное изображение, PhotoImage – полноцветное изображение.

BitmapImage Конструктор класса принимает следующие аргументы:

— `background` и foreground – цвета фона и переднего плана для изображения. Поскольку изображение двухцветное, то эти параметры определяют соответственно чёрный и белый цвет.

— `file и maskfile` – пути к файлу с изображением и к маске (изображению, указывающему какие пиксели будут прозрачными).

— `data и maskdata` – вместо пути к файлу можно указать уже загруженные в память данные изображения. Данная возможность удобна для встраивания изображения в программу. [5]

```
from tkinter import *
BITMAP = """
#define im_width 32
#define im_height 32
static char im_bits[] = {
0xaf,0x6d,0xeb,0xd6,0x55,0xdb,0xb6,0x2f,
0xaf,0xaa,0x6a,0x6d,0x55,0x7b,0xd7,0x1b,
0xad,0xd6,0xb5,0xae,0xad,0x55,0x6f,0x05,
0xad,0xba,0xab,0xd6,0xaa,0xd5,0x5f,0x93,
0xad,0x76,0x7d,0x67,0x5a,0xd5,0xd7,0xa3,
0xad,0xbd,0xfe,0xea,0x5a,0xab,0x69,0xb3,
0xad,0x55,0xde,0xd8,0x2e,0x2b,0xb5,0x6a,
0x69,0x4b,0x3f,0xb4,0x9e,0x92,0xb5,0xed,
0xd5,0xca,0x9c,0xb4,0x5a,0xa1,0x2a,0x6d,
0xad,0x6c,0x5f,0xda,0x2c,0x91,0xbb,0xf6,
0xad,0xaa,0x96,0xaa,0x5a,0xca,0x9d,0xfe,
```

```
0x2c,0xa5,0x2a,0xd3,0x9a,0x8a,0x4f,0xfd,
0x2c,0x25,0x4a,0x6b,0x4d,0x45,0x9f,0xba,
0x1a,0xaa,0x7a,0xb5,0xaa,0x44,0x6b,0x5b,
0x1a,0x55,0xfd,0x5e,0x4e,0xa2,0x6b,0x59,
0x9a,0xa4,0xde,0x4a,0x4a,0xd2,0xf5,0xaa
};
"""
window = Tk()
window.geometry('400x200')
image = BitmapImage(data = BITMAP, background = 'orange',
   foreground = 'green')
button = Button(window, image = image)
button.pack()
window.mainloop()
```

Рисунок 16 – Пример BitmapImage

PhotoImage PhotoImage позволяет использовать полноцветное изображение. Кроме того у этого класса есть несколько методов для работы с изображениями. Аргументы конструктора:

- `file` – путь к файлу с изображением;
- `data` – вместо пути к файлу можно указать уже загруженные в память данные изображения. Изображения в формате GIF могут быть закодированы с использованием base64. Данная возможность удобна для встраивания изображения в программу;
- `format` – явное указание формата изображения;
- `width, height` – ширина и высота изображения;
- `gamma` – коррекция гаммы;
- `palette` – палитра изображения.

```
from tkinter import *
window = Tk()
```

```
window.geometry('400x200')
image = PhotoImage(file = 'sgu.gif')
button = Button(window, image = image)
button.pack()
window.mainloop()
```

Рисунок 17 – Пример PhotoImage

Canvas Рисование в Tkinter реализовано при помощи виджета Canvas. Это функционал высокого уровня, который позволяет создавать графику в Tkinter. Рисование можно использовать для создания графиков статистики, самодельных пользовательских виджетов. [6]

```
from tkinter import *
window = Tk()
canvas = Canvas(window, width = 400, height = 400, bg = "orange")
# Рисуем зеленую диагональ.
canvas.create_line(0, 0, 400, 400, width = 5, fill = "green")
# Рисуем голубой прямоугольник
canvas.create_rectangle(50, 50, 150, 150, fill = "aquamarine",
   outline = "green")
# Рисуем красный овал
canvas.create_oval([200 , 100], [300 ,200], fill = "red")
# Рисуем белый треугольник
canvas.create_polygon([150, 300], [50, 300] ,[100, 200] ,fill =
   "white", outline = "green")
# Пишем текст
canvas.create_text(300 ,20 ,text = "Hello World!", font = "Arial
```

```
   20", justify = CENTER, fill = "red")
canvas.pack()
window.mainloop()
```

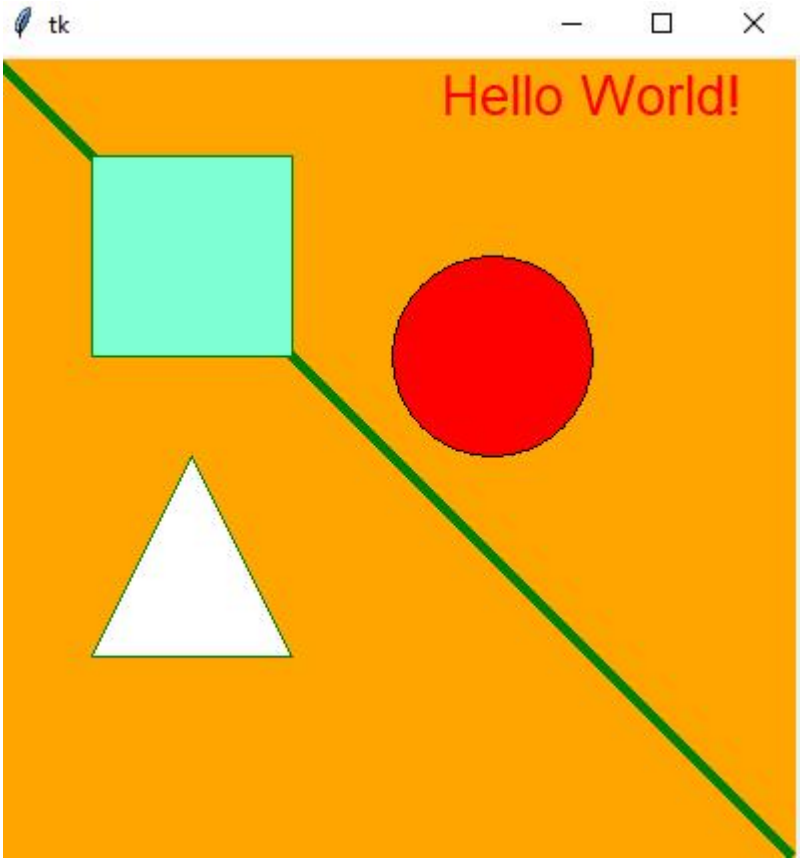

Рисунок 18 – Пример Canvas

1.7 Преимущества и недостатки Tkinter

В Python есть довольно много GUI фреймворков, однако только Tkinter встроен в стандартную библиотеку языка. У Tkinter есть несколько преимуществ. Он кроссплатформенный, поэтому один и тот же код можно использовать на Windows, macOS и Linux. Визуальные элементы отображаются через собственные элементы текущей операционной системы, поэтому приложения, созданные с помощью Tkinter, выглядят так, как будто они принадлежат той платформе, на которой они работают. В плане использования, Tkinter является относительно легким по сравнению с другими библиотеками. Tkinter имеет и свои слабости. Во-первых, графические интерфейсы имеют устаревший внешний вид. Во-вторых, набор виджетов ограничен и не отвечает современным стандартам. Есть альтернативы, которые развиваются сильнее,

например PyQt. [7] В-третьих, так как большинство вызовов Tkinter формируется как команда Tcl и интерпретируется Tcl, возможны потери в скорости работы. [8]

2 Практические задания

1. Создайте окно размером (200 x 200) с заголовком «Tkinter» и кнопкой размером (10 x 2), при нажатии на которую появляется Label с текстом «Hello World», а кнопка исчезает. Кнопка и Label находятся в середине окна.
2. Создайте окно, в котором содержится 2 рамки (frame), в каждой из которых находится по 3 RadioButton. Первый блок – выбор цвета фона, второй – цвет текста. Так же присутствует кнопка, при нажатии на которую рамки и кнопка пропадает, но появляется Label «Hello World» с выбранными свойствами в RadioButton.
3. Создайте интерфейс, с помощью которого можно получить случайное целое число из интервала [a, b].
4. Создайте окно, в котором будет отображено текущее время. Черный фон, зеленые цифры, жирный шрифт 20-го размера.
5. Создайте интерфейс, с помощью которого можно переводить километры в мили и обратно. (1 км = 0,62 мили, 1 миля = 1,61 км)
6. Создайте интерфейс, с помощью которого пользователь сможет осуществить вывод и добавление данных для СУБД.
7. С помощью Canvas нарисуйте разноцветный узор из геометрических фигур.

3 Проверочный тест

1. Выберите неверное утверждение:

   а) Tkinter – стандартная библиотека GUI для Python
   б) Tkinter поддерживает кроссплатформенность
   в) Интерфейсы на Tkinter выглядят современно
   г) Главным элементом GUI Tkinter является окно

2. В чем разница между виджетами RadioButton и CheckButton?
3. Что из перечисленного не является виджетом?

   а) Label
   б) ListBox
   в) Frame
   г) Pack

4. Для чего нужны упаковщики? Перечислите их. В чем разница между ними?
5. При использовании упаковщика place указаны атрибуты: anchor = CENTER, relx = 0.5, rely = 0.5. Как будет расположен виджет?
6. На клавиатуре нажали клавишу "/". Что покажет метод event.keysym?

   а) /
   б) 47
   в) 220
   г) slash

7. Пользователь хочет добавить изображение с помощью PhotoImage. Какой из форматов изображения ему подойдет?

   а) jpg/jpeg
   б) gif
   в) png
   г) tif

Ответы

1. в
2. Главная разница между данными виджетами заключается в том, что пользователь может выбрать только один пункт из элементов RadioButton, а CheckButton позволяет выбрать несколько.
3. г
4. Упаковщик это специальный механизм, который размещает виджеты на окне. `pack()`, `grid()`, `place()`. Менеджер `pack` просто заполняет внутреннее пространство на основании предпочтения того или иного края, необходимости заполнить все измерение. `grid()` представляет собой таблицу с ячейками, в которые помещаются виджеты. `place()` позволяет размещать виджет в фиксированном месте с фиксированным размером. Также он позволяет указывать координаты размещения в относительных единицах.
5. Центр виджета окажется в центре окна.
6. г
7. б

ЗАКЛЮЧЕНИЕ

При выполнении данной работы были описаны элементы графических интерфейсов, в частности элементы модуля Tkinter, такие как: окна, виджеты, события, их методы и атрибуты. На многочисленных примерах были рассмотрены возможности функционала Tkinter. Разработаны задания и тест для освоения полученных знаний, а так же для проверки и закрепления уже имеющихся навыков работы с Python. По итогом практики так же были приобретены навыки методической работы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 *Слаткин, В.* Секреты Python. 59 рекомендаций по написанию эффективного кода / В. Слаткин. — Москва: Вильямс, 2016. — 272 с.

2 Основы Tk [Электронный ресурс]. — URL: https://intuit.ru/studies/courses/49/49/lecture/27080?page=2/ (Дата обращения 22.09.2020). Загл. с экр. Яз. рус.

3 TkDocs Documentation [Электронный ресурс]. — URL: https://tkdocs.com/ (Дата обращения 25.09.2020). Загл. с экр. Яз. англ.

4 Tcl8.5.19/Tk8.5.19 Documentation [Электронный ресурс]. — URL: http://www.tcl.tk/man/tcl8.5/contents.htm (Дата обращения 23.09.2020). Загл. с экр. Яз. англ.

5 *Shipman, J. W.* Tkinter 8.5 reference: aGUI for Python / J. W. Shipman. — New Mexico: New Mexico Tech Cemputer Center, 2013. — 168 pp.

6 Создание графического интерфейса [Электронный ресурс]. — URL: https://metanit.com/python/tutorial/9.1.php (Дата обращения 20.09.2020). Загл. с экр. Яз. рус.

7 *Прохоренок, Н.* Python 3 и PyQt. Разработка приложений / Н. Прохоренок. — Петербург: БХВ-Петербург, 2012. — 704 с.

8 *Moore, A. D.* Python GUI Programming with Tkinter / A. D. Moore. — Birmingham: Packt Publishing, 2018. — 452 pp.

ПРИЛОЖЕНИЕ А
Примеры задач

Задача №1

```
from tkinter import *
window = Tk()
window.geometry('200x200')  # Создаем окно размером 200x200
window.title('Tkinter')  # Задаем название заголовка

def show_label(event):
    label = Label(window, text='Hello World') # Инициализация Label
    label.place(anchor = CENTER, relx = 0.5, rely = 0.5) #
       Размещение Label
    button.place_forget() # Убираем кнопку из окна

button = Button(window, text = 'Hi!', width = '10', height = '2')
   # Инициализация кнопки
button.bind('<Button-1>', show_label)   # Событие, при нажатии
   л.кн. мыши вызывает функцую
button.place(anchor = CENTER, relx = 0.5, rely = 0.5) # Размещение
   кнопки
window.mainloop()
```

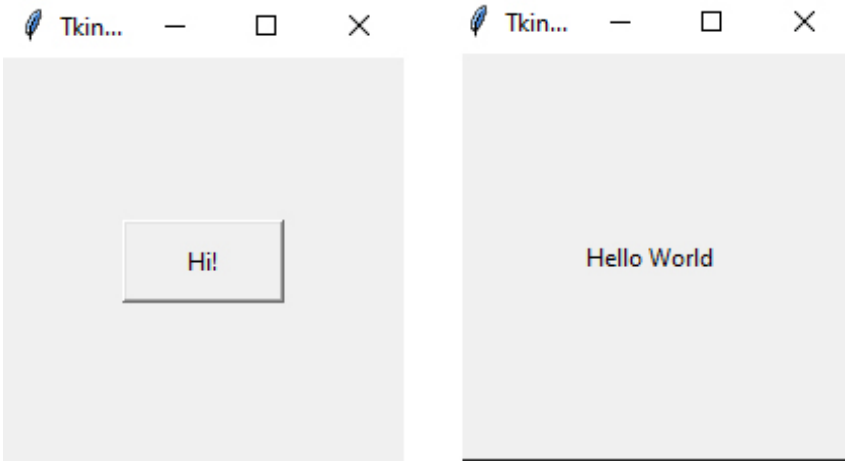

Рисунок 19 – Задание №1

Задача №2

```
from tkinter import *
```

```
window = Tk()
window.geometry('300x300')
window.title('Tkinter')

def show_label(event):
    # Инициализация Label с выбранными параметрами
    label = Label(window, text='Hello World', bg = bgcolor.get(),
       fg = textcolor.get())
    label.place(anchor = CENTER, relx = 0.5, rely = 0.5)
    # Скрытие ненужных элементов
    button.place_forget()
    frame1.pack_forget()
    frame2.pack_forget()

button = Button(window, text = 'Hi!', width = '10', height = '2')

# Инициализация фреймов
frame1 = Frame(window)
frame2 = Frame(window)
label_colorbg = Label(frame1, text = 'Цвет фона:')
label_colortext = Label(frame2, text ='Цвет текста:')
# Переменные, которые хранят выбранные RadioButon
bgcolor = StringVar()
textcolor = StringVar()
 # Задаем варианты по умолчанию
bgcolor.set('black')
textcolor.set('white')

# Инициализация RadioButton
radiobutton1 = Radiobutton(frame1, text = 'Желтый', variable =
   bgcolor, value = 'yellow')
radiobutton2 = Radiobutton(frame1, text = 'Зеленый', variable =
   bgcolor, value = 'green')
```

```
radiobutton3 = Radiobutton(frame1, text = 'Черный', variable =
   bgcolor, value = 'black')
radiobutton4 = Radiobutton(frame2, text = 'Красный', variable =
   textcolor, value = 'red')
radiobutton5 = Radiobutton(frame2, text = 'Голубой', variable =
   textcolor, value = 'aquamarine')
radiobutton6 = Radiobutton(frame2, text = 'Белый', variable =
   textcolor, value = 'white')
button.bind('<Button-1>', show_label)

# Размещение виджетов
frame1.pack()
label_colorbg.pack()
radiobutton1.pack()
radiobutton2.pack()
radiobutton3.pack()
frame2.pack()
label_colortext.pack()
radiobutton4.pack()
radiobutton5.pack()
radiobutton6.pack()
button.place(anchor = S, relx = 0.5, rely = 1)
window.mainloop()
```

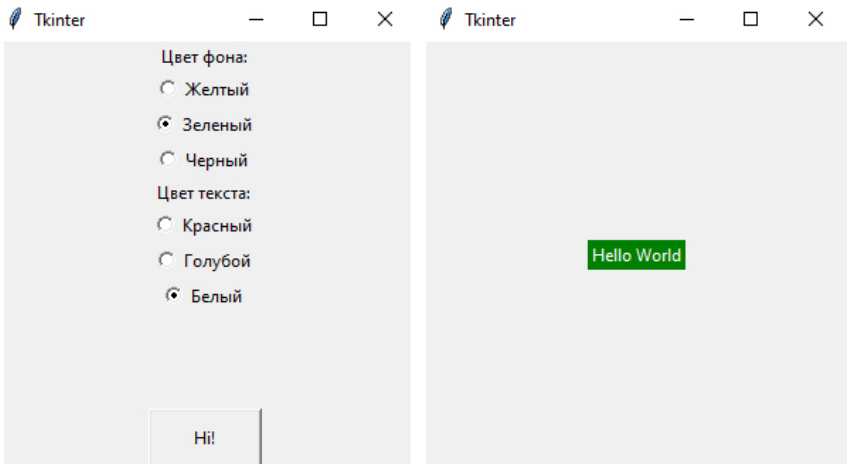

Рисунок 20 – Задание №2

Задача №3

```
from tkinter import *
import random

# функция выбирает случайное число из диапозона
def randomize():
    a = int(a_entry.get())
    b = int(b_entry.get())
    result_label["text"] = str(random.randint(a, b))

window = Tk()
window.title("Рандомайзер")
a_label = Label(window, text = "a=")
b_label = Label(window, text = "b=")
a_entry = Entry(window, width = 5)
b_entry = Entry(window, width = 5)

rand_button = Button(text="Генерировать", command=randomize)
result_label = Label(window)
a_label.grid(row = 0, column = 0, padx = 10)
a_entry.grid(row = 0, column = 1, padx = 1)
b_label.grid(row = 0, column = 3, padx = 1)
b_entry.grid(row = 0, column = 4, padx = 10)
rand_button.grid(row = 2, column = 2, sticky = "nsew")
result_label.grid(row = 1, column = 2)

window.mainloop()
```

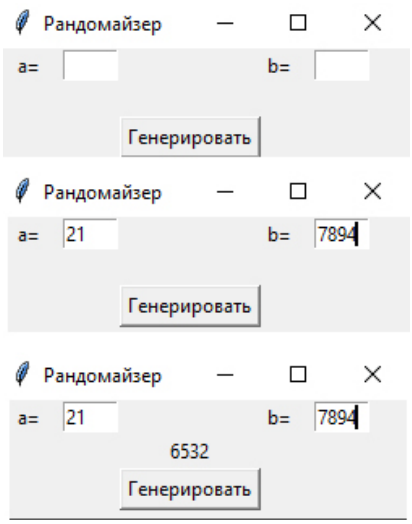

Рисунок 21 – Задание №3

Задача №4

```
from tkinter import *
# Импортируем библиотеку для работы со временем
import time
window = Tk()
window.geometry('250x100')
window.title('Время')
# Создаем Label в указанном формате
clock = Label(window, font = ('times', 20, 'bold'), bg= 'black',
   fg = 'green')
clock.pack(fill = BOTH, expand = 1)

def tick():
# Забираем текущее время с компьютера
    time_now = time.strftime('%H:%M:%S')
# При изменении времени будем менять Label
    clock.config(text = time_now)
# Каждые 200 мс вызываем функцию — обновление времени
    clock.after(200, tick)

tick()
window.mainloop()
```

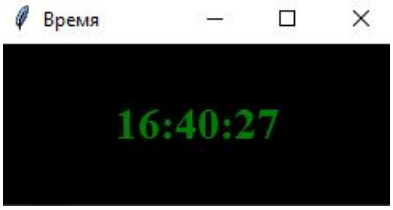

Рисунок 22 – Задание №4

Задача №5

```
from tkinter import *

# функция перевода км в мили
def km_to_miles():
    km = float(km_entry.get())
    miles = km * 0.62
    result_miles["text"] = str(miles) + " мили"

# Функция перевода миль в км
def miles_to_km():
    miles = float(miles_entry.get())
    km = miles * 1.61
    result_km["text"] = str(km) + " км"

# Создание окна
window = Tk()
window.title("Конвертер")
window.resizable(width=False, height=False)

# Инициализация фреймов и их элементов
frame1 = Frame(window)
km_entry = Entry(frame1, width = 10)
miles_label = Label(frame1, text = "км")
frame2 = Frame(window)
miles_entry = Entry(frame2, width = 10)
km_label = Label(frame2, text = "мили")
```

```
# Разметка виджетов
km_entry.grid(row = 0, column = 0, sticky = "e")
miles_label.grid(row = 0, column = 1, sticky = "w")
miles_entry.grid(row = 1, column = 0, sticky = "e")
km_label.grid(row = 1, column = 1, sticky = "w")

convert_button1 = Button(window, text = "\N{RIGHTWARDS BLACK
   ARROW}", command = km_to_miles)
convert_button2 = Button(window, text = "\N{RIGHTWARDS BLACK
   ARROW}", command = miles_to_km)
result_miles = Label(window, text = "мили")
result_km = Label(window, text = "км")

# Разметка виджетов
frame1.grid(row = 0, column = 0, padx = 10)
convert_button1.grid(row = 0, column = 1, pady = 10)
result_miles.grid(row = 0, column = 2, padx = 10)
frame2.grid(row = 1, column = 0, padx = 10)
convert_button2.grid(row = 1, column = 1, pady = 10)
result_km.grid(row = 1, column = 2, padx = 10)

window.mainloop()
```

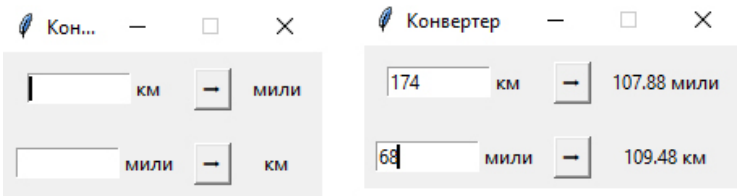

Рисунок 23 – Задание №5

Задача №6

```
import pyodbc
from tkinter import *

# Создаем подключение к БД
cnxn = pyodbc.connect("Driver={ODBC Driver 17 for SQL Server};"
```

```
                        "Server=DESKTOP-RDIDH6R;"
                        "Database=Discount;"
                        "Trusted_Connection=yes;")
cursor = cnxn.cursor()

# Функция, которая выводит в отдельном окне содержимое таблицы БД –
# информацию о клиентах
def ViewClients():
    # Создаем новое окно
    window_clients = Tk()
    window_clients.title("Клиенты")
    window_clients.geometry("600x400")
    # Создаем полосу прокрутки
    scrollbar = Scrollbar(window_clients)
    scrollbar.pack(side = RIGHT, fill = Y)
    # Отправляем запрос в БД на получение данных
    cursor.execute("SELECT * FROM Clients")
    result = cursor.fetchall()
    clients_list = []
    # Создаем ListBox, в который заносим данные из БД
    clients = Listbox(window_clients, yscrollcommand =
       scrollbar.set)
    for row in result:
        temp = []
        for item in row:
            temp.append(str(item))
        clients_list.append(temp)
    for row in clients_list:
        clients.insert('end',row)
    clients.pack(fill = BOTH, expand = 1)

# Функция добавляет в БД новые данные, которые пользователь ввел
# в поля Entry
def AddClient():
```

```
    cursor.execute('''
                           INSERT INTO Clients (fname, mname,
                              lname, birthday, phone)
                           VALUES
                           (?, ?, ?, ?, ?)
                           ''', (ent_first_name.get(),
                              ent_middle_name.get(),
                                 ent_last_name.get(),
                                    ent_birthday.get(),
                                 ent_phone.get()))
    cnxn.commit()

# Функция очищает поля для ввода
def ClearEntry():
    ent_first_name.delete(0, END)
    ent_middle_name.delete(0, END)
    ent_last_name.delete(0, END)
    ent_birthday.delete(0, END)
    ent_phone.delete(0, END)

#Создание и разметка виджетов основного окна
window = Tk()
window.title("Добавить клиента")

frm_form = Frame(relief = SUNKEN, borderwidth = 3)
frm_form.pack()

lbl_first_name = Label(master = frm_form, text = "Имя:")
ent_first_name = Entry(master = frm_form, width = 50)
lbl_first_name.grid(row = 0, column = 0, sticky = "e")
ent_first_name.grid(row = 0, column = 1)

lbl_middle_name = Label(master = frm_form, text = "Отчество:")
ent_middle_name = Entry(master = frm_form, width = 50)
```

```
lbl_middle_name.grid(row = 1, column = 0, sticky = "e")
ent_middle_name.grid(row = 1, column = 1)

lbl_last_name = Label(master = frm_form, text = "Фамилия:")
ent_last_name = Entry(master = frm_form, width = 50)
lbl_last_name.grid(row = 2, column = 0, sticky = "e")
ent_last_name.grid(row = 2, column = 1)

lbl_birthday = Label(master = frm_form, text = "Дата рождения:")
ent_birthday = Entry(master = frm_form, width = 50)
lbl_birthday.grid(row = 3, column = 0, sticky = "e")
ent_birthday.grid(row = 3, column = 1)

lbl_phone = Label(master = frm_form, text = "Телефон:")
ent_phone = Entry(master = frm_form, width = 50)
lbl_phone.grid(row = 4, column = 0, sticky = E)
ent_phone.grid(row = 4, column = 1)

frm_buttons = Frame()
frm_buttons.pack(fill = X, ipadx = 5, ipady = 5)
btn_clients = Button(master = frm_buttons, text = "Клиенты",
   command = ViewClients)
btn_clients.pack(side = RIGHT, padx = 10, ipadx = 10)
btn_submit = Button(master = frm_buttons, text = "Отправить",
   command = AddClient)
btn_submit.pack(side = RIGHT, padx = 10, ipadx = 10)
btn_clear = Button(master = frm_buttons, text = "Очистить",
   command = ClearEntry)
btn_clear.pack(side = RIGHT, ipadx = 10)

window.mainloop()
```

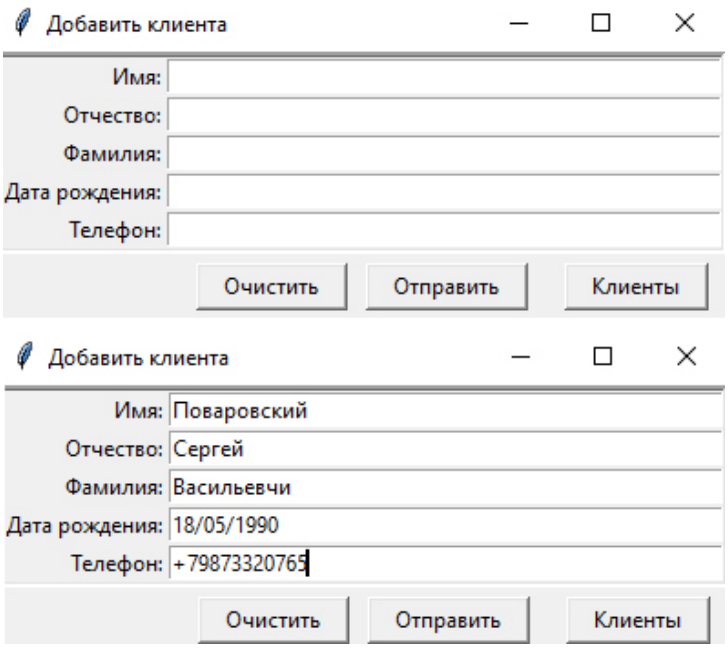

Рисунок 24 – Задание №6 - Элементы интерфейса

Результаты Сообщения

	id	fname	mname	lname	birthday	phone
1	1	Николай	Иванович	Петров	16/05/1989	+79875431982
2	2	Дмитрий	Сергеерич	Федорчук	28/10/1994	+79035711090
3	3	Светлана	Игоревна	Носова	22/01/1978	+79617651234
4	4	Алена	Викторовна	Ростова	14/09/1997	+79078768776
5	5	Антон	Витальевич	Рогачев	20/12/1991	+79548761209
6	6	Дмитрий	Сергеевич	Калашников	04/04/1984	+79378734987
7	7	Ульяна	Константинова	Мельникова	10/05/1988	+79178761256
8	8	Simple	NULL	John	15/11/1987	+79271507652

Рисунок 25 – Содержимое БД

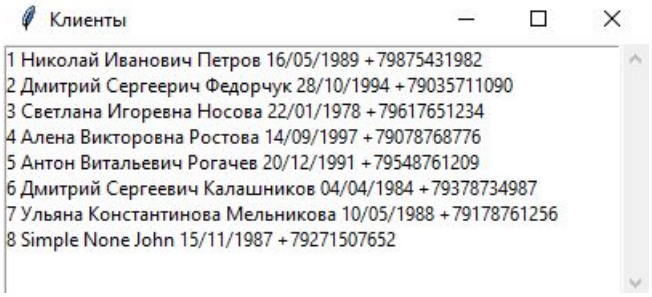

Рисунок 26 – Данные из БД в интерфейсе

Результаты | Сообщения

	id	fname	mname	lname	birthday	phone
1	1	Николай	Иванович	Петров	16/05/1989	+79875431982
2	2	Дмитрий	Сергеерич	Федорчук	28/10/1994	+79035711090
3	3	Светлана	Игоревна	Носова	22/01/1978	+79617651234
4	4	Алена	Викторовна	Ростова	14/09/1997	+79078768776
5	5	Антон	Витальевич	Рогачев	20/12/1991	+79548761209
6	6	Дмитрий	Сергеевич	Калашников	04/04/1984	+79378734987
7	7	Ульяна	Константинова	Мельникова	10/05/1988	+79178761256
8	8	Simple	NULL	John	15/11/1987	+79271507652
9	13	Поваровский	Сергей	Васильевчи	18/05/1990	+79873320765

Рисунок 27 – В таблицу добавлена строка из интерфейса

Задача №7

```
from random import *
from tkinter import *

# Создаем окно и виджет Canvas
size = 400
window = Tk()
window.title('Узор')
canvas = Canvas(window, width = size, height = size)
canvas.pack()
count = 0

# Цикл заполняет окно случайными кругами
while count < 750:
    # Выбираем цвет из списка случайном образом
    colors = choice(['aqua', 'blue', 'fuchsia', 'green', 'maroon',
       'orange',
                'pink', 'purple', 'red', 'yellow', 'violet',
                   'indigo', 'chartreuse', 'lime', 'aquamarine'])
    # Задаем позицию и размер фигуры случайными значениями
    x = randint(0, size)
    y = randint(0, size)
    d = randint(0, size/5)
    # Создаем круг по заданным параметрам
    canvas.create_oval(x - d/2, y - d/2, x + d/2, y + d/2, fill =
       colors)
    # Обновляем окно
```

```
    window.update()
    count += 1
window.mainloop()
```

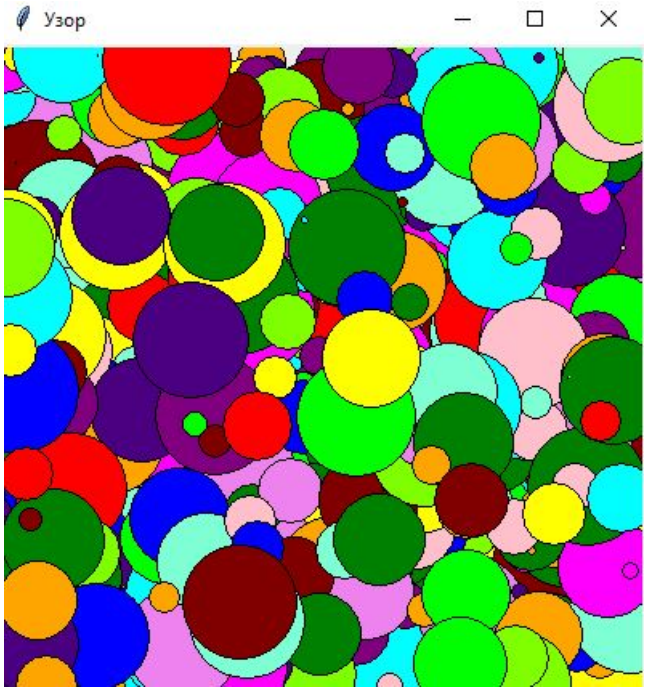

Рисунок 28 – Задание №7. Узор.